Université de NANCY I
de Mathématiques

Sc
N

Reçu Ex. 29-11-75

Sc. N. 75/65

DESCRIPTION FORMELLE, REPRESENTATION, INTERROGATION DES INFORMATIONS COMPLEXES

SYSTEME PIVOINES

UNIVERSITÉ NANCY BIBLIOTHÈQUE

Thèse soutenue le 2 Décembre 1975

par

marion créhange

pour obtenir le grade de Docteur d'Etat ès Sciences Mathématiques

devant la commission d'examen :

75
65

D 095 091237 2

GR
BEL
RNIAME Examinateurs
IR
Mme C. ROLLAND

Université de NANCY I

U.E.R. de Mathématiques

DESCRIPTION FORMELLE, REPRESENTATION, INTERROGATION DES INFORMATIONS COMPLEXES

SYSTEME PIVOINES

Thèse soutenue le 2 Décembre 1975

par

marion créhange

pour obtenir le grade de Docteur d'Etat ès Sciences Mathématiques

devant la commission d'examen :

M. J. LEGRAS	Président
M. F. BODART	
M. C. DELOBEL	
M. J. C. DERNIAME	Examinateurs
M. C. PAIR	
Mme C. POLLAND	

於巴里 梨白

R. Harada

Merci !

Merci d'abord à mon mari et à mes enfants qui par leur soutien moral, leur aide et leur compréhension quasi permanente m'ont permis d'avoir l'esprit libre.

Merci à mon "patron" Claude Pair qui, si gentiment mais aussi très fermement, m'a stimulée et m'a guidée, avec tant d'attention et de compétence. Je lui suis particulièrement reconnaissante d'avoir si bien su, à plusieurs reprises, comprendre mes intentions et les replacer dans un contexte plus général que celui où je les formulais.

Merci à Monsieur J. Legras qui m'a orientée et fait débuter dans cette Informatique alors naissante, et me fait l'honneur de présider le jury.

Merci à Monsieur F. Bodart et à Monsieur C. Delobel qui ont accepté de venir de loin pour m'éclairer de leur jugement et de leurs conseils.

Merci aux autres membres du jury, mes amis Colette Rolland et Jean-Claude Derniame avec qui j'espère pouvoir collaborer efficacement sur des problèmes de fond.

Merci aux étudiants de troisième cycle qui ont travaillé avec moi sur les problèmes d'où est issue cette thèse ou sur l'implémentation de certains points de celle-ci.

Merci à Monsieur J.C. Gardin qui m'a, avec tant de gentillesse et de fine intelligence, initiée à Marseille, il y a bien longtemps déjà, aux problèmes et à l'état d'esprit des systèmes de données.

Merci à Laurence Gallier et à Mireille Franc qui, toujours souriantes, m'ont fort bien tapé cette longue thèse, et à Monsieur F. Deberdt qui en a effectué le tirage.

Et merci au C.N.R.S. qui, en m'accueillant en détachement, m'a donné la liberté de temps et d'esprit nécessaire à ce travail.

DESCRIPTION FORMELLE , REPRESENTATION , INTERROGATION DES INFORMATIONS COMPLEXES

Système PIVOINES

I - INTRODUCTION 1

I.1 - Sous l'angle des idées : motivations 1

I.1.1 - Permettre une économie d'expression et même de pensée 1

I.1.2 - Rendre plus "scientifique" l'étude des systèmes de données 3

I.1.3 - Faciliter les dialogues entre demandeurs et auteurs 4

I.1.4 - Améliorer la compatibilité entre systèmes 4

I.1.5 - Aider à réaliser l'aspect Base de données dans les systèmes de prise en compte de gros problèmes informatiques 5

I.2 - Sous l'angle des faits : genèse et plan de notre travail 6

II - INFORMATIONS,DONNEES ET STRUCTURES DANS LA PERSPECTIVE "ACCES" 11

II.1 - Aperçu intuitif des notions d'information et d'accès 11

II.2 - Quelques exemples d'information . Notion de repère 15

II.3 - Définition d'une information (au sens informatique) 24

II.3.1 - Information 24

II.3.2 - Graphe généralisé 27

II.4 - Algèbre sur les accès 29

II.4.1 - Qu'est-ce qu'un accès ? 29

II.4.2 - Calculs sur les relations d'accès 29

II.4.2.1 - La composition 29

II.4.2.2 - Juxtaposition de q relations de même arité 31

II.4.2.3 - Réciproque 33

II.4.2.4 - Itération 34

II.4.2.5 - Négation 35

II.5 - Structure de donnée 36

II.5.1 - Présentation et première définition 36

II.5.2 - Aperçu de la théorie logique des structures 44

II.5.2.1 - Structure d'information. Nouveau point de vue 44

II.5.2.2 - Donnée et information, dans cette optique 47

II.5.2.3 - Coup d'oeil sur PIVOINES 48

III - LES SYSTEMES DE DONNEES . QUELQUES SYSTEMES REALISES PAR NOTRE EQUIPE OU D'AUTRES 51

III.1 - Définition des "systèmes de données" 51

III.2 - Problèmes traités par notre équipe 56

III.2.1 - Actes juridiques du Moyen-Age 56

III.2.2 - Dossiers médicaux 64

III.3 - Quelques systèmes de données ou modèles "classiques", vus sous l'angle de la structure des données 70

III.3.1 - Systèmes à structure de données fixée 70

III.3.2 - Systèmes à structure de données non fixée 72

III.3.3 - Diverses formalisations et études générales des sytèmes de données 73

IV - SCHEMA GENERAL DU DEROULEMENT DE L'INTERROGATION D'UNE INFORMATION . STRATEGIE DE RECHERCHE 81

Ce chapitre est une "vue d'avion" de notre travail et son développement fait l'objet de toutes la thèse.

IV.1 - Comment exprime-t-on une demande d'interrogation ? 81

IV.1.1 - Décomposition d'une demande en deux parties 81

IV.1.2 - Indépendance du langage d'accès vis-à-vis de la structure 84

IV.1.3 - Nature descriptive du langage 85

IV.1.4 - Pochoirs et inconnues 87

IV.1.5 - Partage des responsabilités de choix d'une stratégie de recherche : filtres et motifs 90

IV.2 - Traduction de la demande en une succession de demandes élémentaires, avec choix d'une stratégie 93

IV.2.1 - Pourquoi parle-t-on de stratégie ? 93

IV.2.2 - Différentes étapes de la traduction d'un motif en programme indéterministe. Où intervient la représentation ? 97

IV.2.2.1 - Cas schématique 97

IV.2.2.2 - Cas général : la représentation "déforme" la structure. Structure PHYLOG 99

IV.2.3 - Principe du choix d'une stratégie 101

IV.3 - Déroulement du programme objet indéterministe. Mode global ou mode sériel ? 103

IV.4 - Récapitulation : schéma du déroulement de l'interrogation 106

V - LE LANGAGE PIVOINES : SYNTAXE ET SEMANTIQUE 108

V.1 - Présentation du langage des motifs 109

V.1.0 - Généralités 109

V.1.1 - Matériaux de construction des motifs 111

V.1.2 - Présentation intuitive, par des exemples, de l'écriture des motifs 113

V.1.3 - Trois remarques sur la construction des motifs 116

V.2 - Grammaire G des motifs 119

V.3 - Sémantique du langage des motifs 122

V.3.1 - Formalisation de la notion de pochoir 122

V.3.2 - Présentation de la signification d'un motif et du mode de définition de celle-ci 124

V.3.3 - Quelques définitions relatives aux significations 126

V.3.4 - Règles de définition de la signification S 128

V.3.5 - Algorithme de "décryptage" et propriétés de G 136

V.3.6 - Exemple 141

V.3.7 - Occurrences d'un motif M dans l'information I . Rôle des inconnues 147

V.3.7.1 - Définition des occurrences de M dans I 148

V.3.7.2 - Cas de US , unité de sélection 149

V.3.7.3 - Résultat d'un filtre F réduit à un motif M 151

V.4 - Passage d'un ensemble d'arcs et points aux motifs dont il est la signification 153

V.4.1 - Terminologie 153

V.4.2 - Le motif trivial M_o et les grammaires G', G'' , G''' 155

V.4.3 - Transformation T sur les G - ramifications 157

V.4.3.1 - Présentation 157

V.4.3.2 - Les quatre familles de transformations élémentaires 158

V.4.3.3 - Association des transformations aux règles 168

V.4.3.4 - Exemple 170

V.4.4 - La transformation T transforme un G - motif M en un G' - motif et conserve sa signification 174

V.4.4.1 - Principe de la démonstration 175

V.4.4.2 - Démonstration du lemme pour les transformations T 4 176

V.4.4.3 - Démonstration du lemme pour les transformations T 1 et T' 1 177

V.4.4.4 - Démonstration du lemme pour les transformations T 3 179

V.4.4.5 - Démonstration du lemme pour les transformations T 2 suivies de T 1' 182

V.4.4.6 - Conclusion 186

V.4.5 - Transformation d'un G" - motif M" en un G"' - motif M_o 187

V.4.6 - Conclusions 194

V.5 - Les filtres 197

V.5.1 - Syntaxe des filtres 197

V.5.1.1 - Grammaire 197

V.5.1.2 - Contexte (et contexte total) à gauche d'un sous-filtre 198

V.5.1.3 - Inconnues, inconnues propres 201

V.5.2 - Sémantique des filtres . Définition des résultats d'un filtre F 202

V.5.3 - Premier coup d'oeil sur l'évaluation d'un filtre 207

V.5.4 - Remarque : comparaison de PIVOINES et du calcul du premier ordre . Extensions envisagées 210

VI - REPRESENTATION D'UNE INFORMATION . STRUCTURE PHYLOG 215

VI.1 - Indépendance 215

VI.2 - Représentation d'une structure de données dans une autre 217

VI.3 - Représentation en mémoire par l'intermédiaire de la structure PHYLOG 220

VI.3.1 - Structure intermédiaire et procédures 220

VI.3.2 - Correspondance entre PHYLOG et la représentation en mémoire 221

VI.4 - Problèmes d'acquisition 224

VII - LA TRADUCTION D'UN FILTRE EN PROGRAMME MAQUETTE . STRATEGIE 227

VII.1 - Forme du programme maquette 228

VII.1.1 - Niveau des motifs 228

VII.1.2 - Niveau du filtre 229

VII.1.3 - Exemple 232

VII.2 - Organisation de la traduction 235

VII.3 - Partie purement syntaxique 236

VII.3.1 - Traitement au niveau du filtre 236

VII.3.2 - Les familles d'arborescences GD et DG 241

VII.4 - " Groupement, traduction, choix " 247

VII.4.1 - Organigramme 248

VII.4.2 - Comment est exprimée la correspondance entre la structure logique et PHYLOG ? 249

VII.4.3 - Cheminement dans les familles GD et DG . Groupements candidats 254

VII.4.4 - Choix d'un candidat 258

VII.4.5 - " Existe-t-il encore des candidats ? " 263

VII.4.6 - Retour en arrière . Planches de salut 264

VII.4.7 - Mise à jour des tables 266

VII.5 - Qualité du programme obtenu 268

VIII - EXECUTION DU PROGRAMME MAQUETTE 271

VIII.1 - Le programme objet exécutable 272

VIII.1.1 - Généralités 272

VIII.1.1.1 - Source et résultat d'un motif 273

VIII.1.1.2 - Les inconnues locales 273

VIII.1.1.3 - Constitution du programme objet exécutable 274

VIII.1.1.4 - Vue simpliste de la gestion de l'espace des résultats 277

VIII.1.1.5 - Les résultats booléens 279

VIII.1.2 - Gestion des valeurs des inconnues 280

VIII.1.2.1 - Principe 280

VIII.1.2.2 - Inconnues liées 281

VIII.1.2.3 - Base de nos choix concernant la gestion des valeurs 283

VIII.1.2.4 - Résultats final et intermédiaire d'un motif 284

VIII.1.2.5 - Mémorisation des ensembles U_i^j de k-uples de valeurs 285

VIII.1.2.6 - Synthèse et conséquences des paragraphes précédents 288

VIII.1.2.7 - Utilisation du tableau W par le programme exécutable 290

VIII.1.2.8 - Exemple 293

VIII.1.3 - Traitement d'un appel de module de recherche R (a_1 , a_2 , ... , a_u ; b) 298

VIII.1.3.1 - Organigramme dans le cas où R n'est affecté par aucun opérateur 298

VIII.1.3.2 - Précisions sur les modules 304

VIII.1.3.3 - Modifications à apporter aux modules pour pouvoir prendre en compte le rôle des opérateurs 317

VIII.1.4 - Traitement d'une réunion 323

VIII.1.5 - Environnement de ces traitements 328

VIII.1.5.1 - Début et fin de programme 328

VIII.1.5.2 - Fin de motif 329

VIII.1.5.3 - Ramasse-miettes 330

VIII.1.6 - Redondance des jeux d'arguments 333

VIII.2 - Le programme TRADID 335

VIII.2.1 - Les tableaux et compteurs entretenus 335

VIII.2.1.1 - A l'echelle du filtre 335

VIII.2.1.2 - A l'echelle de l'appel de module de recherche 337

VIII.2.2 - Les modules de TRADID 338

VIII.2.2.1 - Traitement de E_k, $B(E_k)$, EMPILER 338

VIII.2.2.2 - Traitement de (j) 338

VIII.2.2.3 - Traitement d'un appel de module de recherche $R(a_1,\ldots,a_p,\ldots,a_u;b)$ 339

VIII.2.2.4 - Traitement d'une étiquette F_i 345

VIII.2.2.5 - Traitement d'une instruction REU(j) 346

VIII.2.2.6 - Traitement de DECHU(j) 347

IX - APPLICATIONS, DEVELOPPEMENTS, CONCLUSION 351

I. INTRODUCTION

oOOo

I.1. SOUS L'ANGLE DES IDEES : MOTIVATION.

I.1.1. PERMETTRE UNE ECONOMIE D'EXPRESSION ET MEME DE PENSEE.

"N'écrire que ce qui est pleinement significatif" : tel est le rêve de nombreux auteurs de programmes. En effet, contrairement à la communication humaine qui heureusement laisse une place au superflu, vecteur de chaleur et de poésie, la communication entre l'homme et l'ordinateur est en général d'autant plus satisfaisante qu'elle est plus dépouillée.

Ecrire un programme, c'est décrire une action à exercer sur une information d'un "genre" donné (le mot genre qui sera remplacé plus loin par le mot plus précis de "structure" peut être compris comme une généralisation du "type" utilisé dans les langages de programmation). Dans certaines applications, les algorithmes sont complexes et originaux : originaux en ce sens qu'ils ne peuvent pas être assimilés à des algorithmes d'utilisation fréquente pouvant être trouvés ou ajoutés dans l'arsenal des outils de base. Ces algorithmes doivent alors être décrits par le programmeur. Le genre de l'information manipulée n'est en général pas globalement communiqué à l'ordinateur, d'autant moins que le langage employé est moins évolué ; l'algorithme est pensé en fonction de lui : ainsi, c'est par l'action que l'on exerce sur l'information, que l'on décrit implicitement son genre. Dans d'autres problèmes, au contraire, où l'action à exercer n'est pas originale, il semble intuitivement que ce soit le genre des informations à traiter qui soit intéressant et significatif. Il

serait dommage, dans ce cas et lorsque ce n'est pas indispensable, d'avoir à traduire ces problèmes par de longs programmes et surtout d'avoir à mettre en oeuvre pour chacun d'eux le processus intellectuel consistant à adapter au genre des informations la méthode à appliquer.

Les problèmes de banques de données et de documentation automatique appartiennent à cette seconde catégorie ; ils ont tous le même type d'énoncé et font appel aux mêmes méthodes de résolution. Donnons-leur la dénomination commune et un peu plus large de : problèmes de construction de "systèmes de données", que nous définirons au chapitre III. Ce qui les fait différer les uns des autres, et souvent de façon profonde, c'est le genre des informations - des données - qu'ils traitent.

Un des buts de notre étude est que l'auteur d'un système de données, aussi complexe soit-il, n'ait à fournir à l'ordinateur que des renseignements sur le genre et la représentation des informations que ce système doit traiter et que, pour chaque mise en oeuvre du système, l'utilisateur n'ait à transmettre que l'énoncé de cette mise en oeuvre (par exemple une question documentaire). C'est alors un programme général, valable pour tous les systèmes, qui, étant donné ces renseignements et éventuellement quelques indications traduisant des contraintes particulières (occupation de mémoire, ...), fabrique le programme d'exploitation. Une conséquence extrêmement importante est que des changements même profonds dans la structure des données ou dans leur représentation (et ceci indépendamment de la structure) pourront être apportés sans donner lieu à des catastrophes au niveau de la programmation.

Remarquons dès maintenant que les programmes que nous générons sont, dans la mesure du possible, de "bons" programmes, ayant des performances correctes (c'est le programme générateur qui choisit une bonne stratégie).

I.1.2. RENDRE PLUS "SCIENTIFIQUE" L'ETUDE DES SYSTEMES DE DONNEES.

Si le souci de concision et d'économie de pensée est à l'origine de notre étude, il n'est, de loin, pas son seul moteur. Un premier but, théorique et pratique tout à la fois, est de faire d'un système de données un objet d'étude scientifique, que l'on peut décrire, modifier, façonner, critiquer, comparer à d'autres, ..., en ce qui concerne les possibilités offertes pour la description des documents - c'est le point le plus important - , leur interrogation et leurs modifications. Un autre objectif est de faire également, de la stratégie de recherche, un objet manipulable. Un troisième but enfin est de permettre d'étudier plus systématiquement le problème de la représentation interne des informations et d'établir des règles de choix d'une bonne représentation ; ce choix serait accompli en fonction de la nature des informations, mais aussi de l'usage qui doit en être fait - en vue par exemple de favoriser les accès les plus fréquents ou de rendre faciles certaines transformations - et de facteurs étrangers aux objets à représenter (tailles maximales et optimales, fait de privilégier soit le temps soit la place en mémoire, ...).

I.1.3. FACILITER LES DIALOGUES ENTRE DEMANDEURS ET AUTEURS.

Tous les informaticiens ayant eu à étudier l'opportunité puis éventuellement la mise en oeuvre de la création d'un système de données s'accordent à trouver extrêmement difficile le dialogue avec les demandeurs. Cette difficulté peut sans doute être attribuée à deux facteurs : le fait que de tels problèmes soient foncièrement très délicats à définir complètement mais aussi le manque de langage précis à employer pour poser ces problèmes, proposer des solutions, les critiquer et les améliorer. Le fait de rendre plus scientifique l'étude des systèmes doit rendre plus efficace ce dialogue entre client et pourvoyeur en leur donnant un langage commun précis ; en particulier, l'indépendance entre structure des données et représentation doit permettre de centrer ce dialogue sur l'essentiel, la structure. En outre, le fait de pouvoir facilement modifier la structure des données et leur représentation doit permettre aux partenaires de bâtir ensemble et progressivement le système le plus satisfaisant possible.

I.1.4. AMELIORER LA COMPATIBILITE ENTRE SYSTEMES .

Nous pensons que l'outil que nous mettons au point devrait aussi rendre moins aigu le problème de la compatibilité entre différents systèmes :

- en permettant, étant donné une certaine information enregistrée, de la considérer sous plusieurs éclairages différents et de l'interroger sous chacun de ces éclairages ;

- en permettant au contraire d'interroger ensemble plusieurs informations de genres différents. Deux corpus documentaires (par exemple) conçus indépendamment, et dont les genres diffèrent, peuvent être dans certains cas considérés comme deux parties d'une information unique dont le genre est si l'on peut dire la "réunion" des deux genres. Les demandes d'interrogation de ces deux informations peuvent être alors exprimées et traitées en une seule fois, le traitement de certaines parties de ces demandes étant automatiquement pris en compte différemment pour les deux informations constituantes.

I.1.5. AIDER A REALISER L'ASPECT BASE DE DONNEES DANS LES SYSTEMES DE PRISE EN COMPTE DE GROS PROBLEMES INFORMATIQUES.

Nous pensons que notre point de vue est suffisamment souple et général pour être à la source du traîtement de l'aspect "base de données" dans un système de prise en compte de gros problèmes informatiques ; un tel système pourrait un jour prendre corps à partir des réalisations de Civa [20] et Remora [48, 49] .

I.2. SOUS L'ANGLE DES FAITS : GENESE ET PLAN DE NOTRE TRAVAIL.

Ce problème est né d'un faisceau d'expériences. Ayant réalisé avec notre équipe un certain nombre de systèmes de données que nous relatons au chapitre III, nous avons éprouvé un sentiment d'insatisfaction "heuristique" : nous avions l'impression de ne pas faire avancer la science documentaire et de ne pas pouvoir bénéficier de ses acquisitions, celles-ci n'étant pas assez normalisées. Il manquait une base de réflexion et ajouter un spécimen à la faune déjà florissante des systèmes de données, même un système général, ne nous semblait pas résoudre le problème.

Nous avons donc essayé d'introduire une normalisation, non pas au niveau des systèmes eux-mêmes, l'expérience nous ayant amenée à penser que c'est infaisable sans restreindre trop la généralité ou créer des systèmes monstrueux, mais essentiellement au niveau de leur description. Nous avons ainsi donné un cadre, qui n'est d'ailleurs pas original, à la spécification des structures d'information aussi complexes soient-elles (chapitre II) ; au chapitre III, nous avons défini les "systèmes de données", en introduisant les notions de structure logique, structure de représentation, structure d'acquisition ; ensuite nous avons précisé les notions d'accès et de stratégie de recherche et créé un langage général d'accès, PIVOINES (chapitre V), qui devrait non seulement être un langage pivot permettant de traduire toutes les demandes d'accès de langages d'interrogation existants, mais aussi être commode à utiliser directement ; nous avons ensuite étudié les problèmes de représentation d'information (chapitre VI) puis (chapitre VII) donné une façon de traduire une demande exprimée dans le langage PIVOINES en un programme d'accès, en cherchant à optimiser celui-ci au point de vue de la stratégie de recherche : nous avons ainsi abordé

le problème de la production d'un programme efficace à partir d'une description (statique) de l'objet de ce programme ; le chapitre VIII traite de l'exécution du programme objet de cette phase ; le chapitre IX traite des applications et développements de notre travail.

Le chapitre IV est une présentation rapide et panoramique des réalisations et des notions développées dans les chapitres suivants.

Nous voudrions ici situer notre travail par rapport aux systèmes de données et aux tentatives de formalisation qui existent déjà. Mais pour alléger la rédaction il nous paraît préférable de ne décrire ceux-ci, ou quelques uns d'entre eux, qu'après avoir introduit un vocabulaire précis, ce qui est l'objet du chapitre II. Donc ce n'est qu'au chapitre III que nous présenterons brièvement, en plus des systèmes réalisés par notre équipe, certaines thèses ou réalisations connues.

Dès maintenant, cependant, dégageons des différences essentielles entre notre ouvrage et la plupart des systèmes existants:

- notre réalisation n'est pas un "système de données" mais plutôt un méta-système. En effet, nous verrons que chaque système (au sens habituel du terme) est conçu pour une certaine structure d'informations et une certaine représentation de celles-ci. Or, dans notre travail, structure et représentation sont des données que va prendre en compte le méta-système. Remarquons cependant que d'autres réalisations comme Socrate [bibl. 1]ont été effectuées avec une telle optique à des degrés divers.

- les structures d'information que nous pouvons prendre en compte sont arbitrairement complexes et sortent bien sûr des limites des structures arborescentes traitées par la plupart des systèmes existants : elles peuvent comporter

des relations non fonctionnelles, des relations n-aires, ...

- le fait que la représentation puisse "déformer" la structure (§IV.2.2.2 et VI.3.1) permet en particulier de modifier la représentation, dans sa structure même, sans changer la structure logique des documents, ni l'expression des questions, ni le programme qui les traduit.

- la stratégie appliquée par les programmes objet, souvent difficile à choisir en raison de la complexité des structures, est en grande partie déterminée par le programme traducteur, en fonction de critères que l'on peut choisir.

- comme nous l'avons dit dans le paragraphe I.1.1, les demandes d'interrogation de l'information sont exprimées de façon non "programmatoire" (non impérative) mais de façon descriptive, ce qui évoque d'ailleurs l'initiative laissée à l'ordinateur dans le choix d'une stratégie d'exécution.

II. INFORMATIONS, DONNEES ET STRUCTURES DANS LA PERSPECTIVE "ACCES"

oOOo

II.1. APERCU INTUITIF DES NOTIONS D'INFORMATION ET D'ACCES.

Nous ne nous hasarderons pas à donner une définition générale de l'information ni surtout à aborder ce sujet d'un point de vue philosophique : c'est dans son acception informatique que nous prendrons cette notion. Lorsqu'on utilise en mathématique un mot emprunté au langage courant, ce mot prend alors un sens beaucoup plus précis, plus restreint, plus orienté vers le maniement. C'est le sort que subit le mot "information" : au sens usuel (et dans une certaine mesure au sens donné dans la "théorie de l'information"), une information est un message, revêtant une forme qui peut être extrêmement variée, qui a pour raison d'être de réduire l'incertitude régnant dans un système ; par contre, une information au sens où l'entend l'Informatique est un être qui, essentiellement, doit subir un traitement et qui, dans sa nature même bien avant que ce soit dans sa forme, est conçue en vue de ce traitement. Illustrons cette différence par l'exemple suivant d'une information au sens usuel qui, selon le genre de traitement que l'on désire lui appliquer, donnera naissance à des informations - au sens informatique - extrêmement différentes.

Dans toute la suite de ce texte, ce que nous appellerons simplement information (ou donnée : cf. § II.3 et II.5. 2.2) sera pris au sens informatique du terme, sauf dans les cas où la confusion sera impossible ou sans importance.

Exemple II.1 : Une information (au sens usuel) et diverses informations auxquelles elle peut donner naissance.

Pour évoquer l'usage le plus courant du mot information, prenons l'exemple d'un article de journal. On peut songer à de nombreux traitements possibles de ce message. En voici quelques uns :

1°) Usage documentaire :

L'article peut venir grossir un corpus documentaire que l'on désirera interroger en posant des "questions documentaires". Celles-ci consistent à chercher ou bien la présence dans l'information interrogée de certaines informations élémentaires (descripteurs ou mots-clés, noms propres, valeurs numériques) et éventuellement de certaines relations entre elles, ou bien l'ensemble des valeurs qui, dans l'information interrogée, sont liées d'une façon donnée à des informations élémentaires données. Ce sera par exemple : "chercher les documents dans lesquels il est question d'urbanisation, et de sport ou d'art" ou bien "chercher les documents dans lesquels il est question de l'influence du paysage sur le travail" (les notions à chercher sont paysage et travail et ces notions doivent être liées par la relation orientée : "influence de -sur-") ou bien "chercher, dans certains documents, quels sont les divers éléments qui ont une influence sur le travail".

Selon la richesse de l'exploitation documentaire que l'on désire pratiquer, on choisira de considérer l'article envisagé comme constitué simplement d'une indication destinée à l'identification et d'un ensemble de données élémentaires (la plupart du temps des descripteurs) ou d'adjoindre à cette

forme minimale des relations plus ou moins variées liant les données élémentaires entre elles. Autrement dit, l'information (au sens usuel) qu'était l'article de journal va donner naissance, pour un usage documentaire donné, à une information (au sens informatique) qui sera constituée de données élémentaires éventuellement liées par des relations.

Remarquons que ce passage qui implique un appauvrissement du message initial est l'un des problèmes essentiels de la documentation automatique et des banques de données : jusqu'à quel point appauvrir et orienter le message donné pour en faire une information bien adaptée aux traitements que l'on veut exercer sur elle, traitements qu'il est souvent difficile de cerner dès l'abord ? Il est hélas utopique de vouloir tirer d'une information (au sens usuel) une information (informatique) à usage universel, même si l'on se limite au domaine documentaire.

Dans les problèmes que nous venons d'évoquer, c'est le corpus que nous considérons comme l'information à traiter, chaque document en étant une information composante. Chercher la réponse à une question documentaire se ramènera à un problème d'accès : accès direct à certaines informations élémentaires dans l'information, et accès, par l'intermédiaire des relations, à d'autres informations élémentaires.

2°) Etude de la succession des lettres dans le texte :

La signification du texte n'a plus d'importance ici et la seule chose qui importe est de savoir reconnaître l'occurrence d'une lettre donnée et déterminer par quelle(s) lettre(s) elle est suivie. L'information va donc être sim-

plement un ensemble de caractères (lettres, blancs, signes de ponctuation, etc...) liés par une relation de succession. C'est ce que l'on peut appeler une "liste" de caractères et nous préciserons cette notion plus loin (Exemples II.2.1 et II.5.1.1). Les problèmes qui se posent sont, encore ici, des problèmes d'accès.

3°) Essai de traduction automatique après analyse syntaxique :

Le texte est considéré ici comme une succession de phrases ; chaque phrase est considérée comme constituée d'un groupe sujet, d'un groupe verbal, etc... . On a affaire (1) à une décomposition arborescente du texte et l'information à traiter pour la traduction est cette arborescence. Cette information, là encore, est un ensemble de données élémentaires (catégories syntaxiques et mots du vocabulaire employé) liées par des relations (hiérarchiques). Les problèmes seront encore pour l'essentiel des problèmes d'accès, ces accès étant suivis de traductions et d'opérations de reconstitution d'une nouvelle information, résultat de la traduction.

Cet exemple a attiré notre attention sur la nature des informations et sur l'importance des problèmes d'accès. On pourrait en donner de nombreuses autres illustrations allant de la gestion des bibliothèques de programmes aux problèmes de protection et de secret des informations. Mais si les accès sont essentiels, particulièrement dans les systèmes de données, ils ne sont pas seuls : les problèmes de modification sont également d'une grande importance [bibl. 39 et 41]. Nous nous sommes attachée aux premiers, les seconds, tout en restant en filigrane dans cette étude, feront l'objet d'un travail ultérieur.

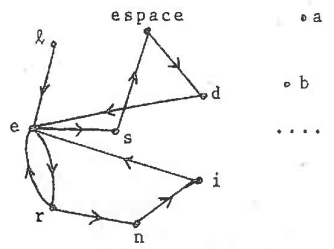
(1) Dans les cas simples, que seuls nous considérons ici.

II.2. QUELQUES EXEMPLES D'INFORMATION. NOTION DE REPERE.

Exemple II.2.1 :

Reprenons le point 2° de l'exemple II.1, dans lequel l'information tirée de l'article de journal est une "liste" de lettres. Précisons cet exemple en supposant que l'article commence comme ceci : "les dernières statistiques..".

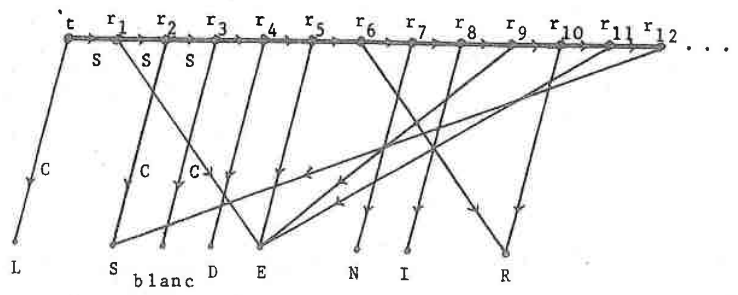
Pour rendre compte fidèlement de cette suite de lettres, il n'est pas suffisant d'établir une relation de succession dans l'ensemble des lettres de l'alphabet (y compris les caractères spéciaux). Si le texte se limitait aux deux premiers mots, cette relation serait représentée par le graphe :



Ce graphe ne permet pas de reconstituer le texte car la relation de succession n'est pas une application : si l'on sait que ℓ est la première lettre, on sait que la deuxième est e mais on ne peut savoir si la troisième est r ou s.

La donnée du graphe n'est donc pas équivalente à celle de la suite de lettres. Il est nécessaire de faire intervenir la notion d'occurrence ou d'emplacement ou rang, c'est-à-dire de repère : on dira que le premier emplacement contient ℓ, le suivant e, le suivant s, le suivant "espace", etc... . On fait ainsi intervenir deux relations : la fonction[1] S, de succession, de l'ensemble des emplacements (des repères) dans lui-même et l'application C, de contenu, de l'ensemble des emplacements dans celui des lettres.

Remarquons que les repères n'ont pour seule raison d'être que de servir de supports aux relations. L'information peut être représentée par le schéma suivant :



Remarque :

La notion de repère est naturelle en informatique. Si l'on veut représenter cette liste en mémoire, une solution simple est que chaque repère soit représenté par un mot de

(1) Dans toute cette thèse, le mot fonction de D dans A signifie : application d'une partie (éventuellement pleine) de l'ensemble de départ D dans l'ensemble d'arrivée A.

mémoire et que ce mot contienne la lettre qui est son image par C. La fonction S sera définie dans un ensemble de mots de mémoire et pourra recevoir diverses matérialisations : contiguïté, chaînage, etc... . La fonction C peut d'ailleurs revêtir d'autres formes que le simple contenu (contenu d'une partie du mot, contenu d'un mot dont l'adresse est contenue dans le mot considéré, etc...) [bibl. 39].

La consultation d'une liste s'opère toujours selon le même processus : elle commence toujours par l'accès à l'élément t appelé tête de la liste ; à partir de cette tête, la fonction C permet ensuite d'accéder au premier élément, puis, à partir de t encore, S donne accès au deuxième repère à partir duquel C donne accès au deuxième élément. Et ainsi de suite. Les fonctions C et S sont appelées tout naturellement relations (ici fonctions) d'accès élémentaires.

Nous parlerons au paragraphe II.4 de la façon de combiner les diverses relations d'accès élémentaires ; il est dès maintenant évident qu'on s'intéressera à leur composition.

Exemple II.2.2 :

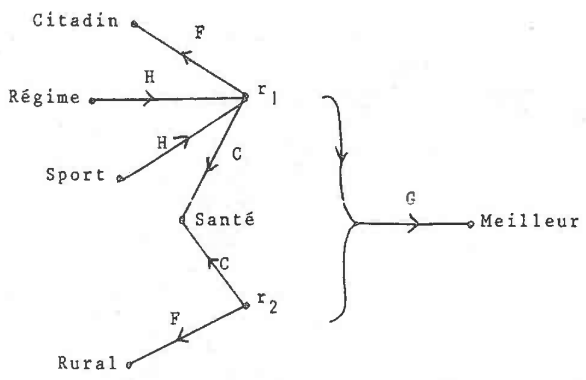
Reprenons le point 1° de l'exemple II.1 et supposons que l'article, pour figurer dans le corpus, avant d'être analysé ait été résumé ainsi (il est bien évident que nous schématisons) : "la santé des citadins est influencée par leur régime et le sport qu'ils pratiquent ; elle est plutôt meilleure que celle des habitants de zones rurales". On suppose également que le résultat de l'analyse soit le suivant, étant donné qu'on dispose des relations : H = "- agit sur -", F = "- appartient à -", G = "la comparaison entre - et - donne -" :

descripteurs : santé
citadins
régime
sport
meilleur
rural

relations : santé F citadin
régime H santé
sport H santé
santé F rural

(santé du citadin, santé du rural)
G meilleur.

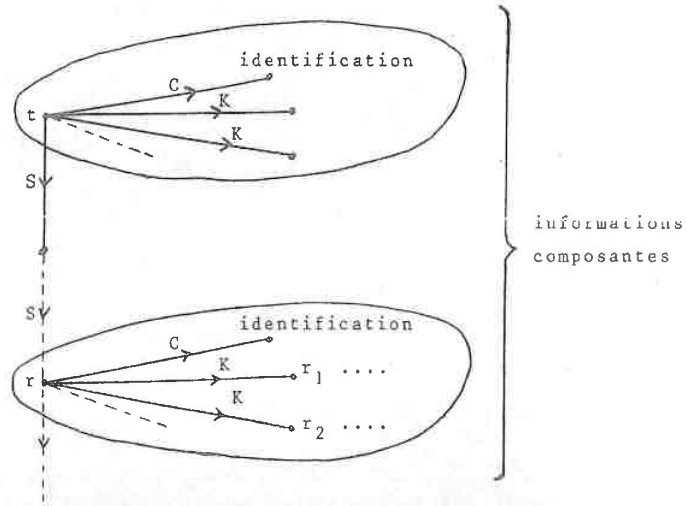
L'information qui traduit cet article peut, dans ces hypothèses, être schématisée ainsi, les repères étant nécessaires pour les mêmes raisons que dans l'exemple II.2.1 (nous adaptons la signification de F, G et H à la présence de ces repères) :



Pour communiquer cette information, il faut préciser :

- l'ensemble V des mots pouvant être utilisés
- l'ensemble R des repères : appelons E la réunion de V et R
- les relations (relations d'accès élémentaires)
  - . H, F, C définies de E vers E
  - . G définie de E^2 vers E (la flèche sur l'accolade traduit l'ordre des arguments de G).

Si l'article fait partie d'un corpus, il doit posséder un renseignement d'identification. Celui-ci sera attaché à un repère r qui devra donner accès à l'ensemble de l'article (en étant par exemple lié à r_1 et r_2, par une relation que nous appelons K). D'autre part, c'est en fait le corpus dans son intégralité qui est considéré comme l'information, chaque document en étant une "information composante". Sauf indication contraire, cette information est considérée comme une "liste de documents" : cette liste a une tête que nous appelons par exemple t et des repères qui se déduisent chacun du précédent par une relation de succession S. Le repère r est un de ceux-ci. Voici quel peut alors être le schéma général du corpus :



Nous reparlerons de ce cas au paragraphe V.3,7.2 et parlerons d'un outil particulièrement adapté à l'interrogation d'une telle information : US (pour "unité de sélection").

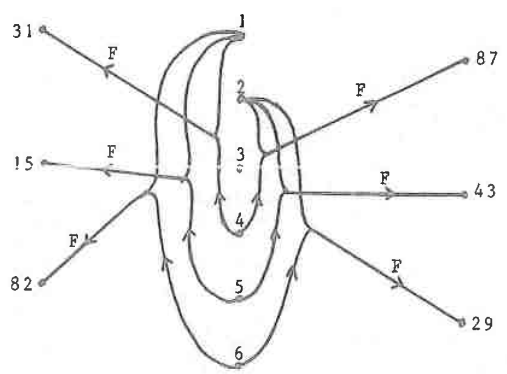
Exemple II.2.3 :

Un tableau de valeurs entières, à n dimensions, comme on en rencontre dans les langages de programmation, est aussi une information. On peut spécifier une telle information par un certain nombre de valeurs des indices et la relation qui, à partir de chaque n-uple de valeurs des indices, donne accès à la valeur de l'élément de tableau correspondant.

Par exemple, le tableau à 2 dimensions suivant :

i_1 \ i_2	1	2
4	31	87
5	15	43
6	82	29

peut être considéré comme l'information schématisée ainsi :



Appelons E l'ensemble de tous les indices et de toutes les valeurs entières (l'ensemble des indices et celui des valeurs entières n'étant pas nécessairement disjoints).

F est une relation (ici fonction) de E^2 vers E qui, à chaque couple pertinent de valeurs d'indices, associe l'élément de tableau correspondant. La flèche placée sur l'accolade montre l'orientation du couple d'éléments de E.

Remarque :

Selon l'usage que l'on désire faire de ce tableau - simple accès à une valeur, connaissant les indices, ou parcours du tableau, par exemple pour la recherche d'une valeur donnée - on ajoutera (dans ce dernier cas) ou pas à la relation F une relation de succession entre les valeurs des indices et même éventuellement des fonctions de calcul : par exemple l'addition, considérée comme une application de E^2 dans E.

On peut très bien, pour certains de leurs usages les plus simples, assimiler à cet exemple tous les cas où l'information est un dessin, une carte (géographique, géologique, ...), une maquette, un film ("tableau" à 3 dimensions, la 3ème étant le temps), un tracé d'encéphalogramme, etc... que l'on a "digitalisés".

Tous les traitements que l'on peut désirer sur ce genre d'information commencent par un accès à certaines de ses parties et cet accès est composé d'accès élémentaires des p-uples d'indices vers les valeurs.

Exemple II.2.4 :

L'état d'une mémoire centrale d'ordinateur ayant une unité de calcul peut être considéré comme une information décrite de la façon suivante [bibl. 39, § 1.4].

La mémoire est constituée de mots (physiques) ; appelons M leur ensemble. Selon l'ordinateur ou l'application envisagée, ou même l'instruction considérée, chaque mot est vu comme une suite d'octets ou de bits, ou ... Mais admettons ici que chaque mot de M est constitué de ℓ bits. Appelons B l'ensemble $\{0,1\}$. Chaque mot a un contenu qui est un élément de B^{ℓ}. Chaque mot a également une adresse qui en général, comme nous le supposerons, peut être contenue dans un mot ou une partie de mot. L'ensemble A des adresses est une partie de B^{ℓ}.

On peut accéder directement non pas aux mots de mémoire mais, puisqu'il s'agit de mémoires adressables, aux adresses. C'est à partir de celles-ci, puisque les mémoires usuelles ne sont pas des mémoires associatives, que se font tous les accès, par l'intermédiaire des relations suivantes :

- 1 - une application bijective (si la mémoire est à adressage fixe) de A dans M qui à chaque adresse associe le mot ayant cette adresse : on l'appelle la fonction d'adressage.

- 2 - une application de M dans B^{ℓ} qui à chaque mot (assimilable à un repère dans les exemples précédents) associe son contenu : on l'appelle la fonction contenu.

- 3 - des opérations de calcul dans B^{ℓ} (et en particulier une fonction de succession dans l'ensemble des adresses).

D'autres exemples sont développés dans le chapitre III décrivant des réalisations que nous avons effectuées. On pourrait aussi emprunter de nombreux exemples à la gestion des entreprises (stocks, chaînes de fabrication, etc...).

II.3. DEFINITION D'UNE INFORMATION (AU SENS INFORMATIQUE).

Nous allons formaliser la notion d'information puis celle de structure d'information (au paragraphe II.5). Cette formalisation a pour origine celle donnée dans [bibl. 39 et 41]. Il est possible d'aller beaucoup plus loin dans cette mathématisation, comme l'ont fait par exemple J.P. FINANCE et J.L. REMY dans [bibl.23] ; cela sort du cadre de notre étude.

II.3.1. INFORMATION.

Les exemples que nous avons donnés nous amènent naturellement à poser la définition suivante d'une information. Cette définition démarque celles données dans [39, 41]. Mais, dans [41], une distinction importante est faite entre "donnée" et "information" : la donnée est la réalisation d'un être abstrait appelé information. Nous n'avons pas besoin ici de cette notion d'information et dans notre texte emploierons indifféremment "information" et "donnée" pour désigner une donnée ; nous donnerons un peu plus de précision sur cette confusion au paragraphe II.5.2.2.

Définition :

Une information $I = (R, V, L)$ est la donnée de :

- R : ensemble fini de repères (voir ci-dessous) ;
- V : ensemble des informations élémentaires pouvant figurer dans l'information, c'est-à-dire matériaux avec lesquels est bâtie l'information ; dans les exemples que nous

avons donnés, ces informations élémentaires sont des lettres, des descripteurs, des indices ou des entiers. V est appelé le vocabulaire de l'information. Pourquoi disons-nous "pouvant figurer" ? Parce qu'il est souvent commode d'inclure dans V plus d'éléments qu'il n'en figure effectivement dans l'information ; ainsi lorsqu'une information comporte des données numériques - disons des entiers -, on pourra sans inconvénient considérer que V inclut l'ensemble de tous les entiers ; de même, si une information contient des descripteurs empruntés à un certain lexique, on pourra considérer ce lexique tout entier comme faisant partie de V. La plupart du temps, on sera amené à décomposer V en ensembles disjoints. Cette notion n'apparaîtra presque pas dans cette thèse mais sera traitée d'une part au cours des applications ultérieures, d'autre part dans le développement du langage de requête (cf. IV.1.1).
Nous poserons $E = V \cup R$.

- $L = \bigcup_{p=2}^{\infty} L_p$, chaque L_p étant un ensemble éventuellement vide de relations, dites "relations d'accès élémentaires" définies de E^{p-1} vers E. Pour toutes les valeurs de p, L_p est fini. De plus, le nombre des valeurs de p pour lesquelles L_p n'est pas vide est fini ; ainsi il existe pour chaque information une valeur P de p telle que :

$$p > P \Longrightarrow L_p = \emptyset .$$

Les relations éléments de L_p seront dites avoir p comme arité. Certaines peuvent être des fonctions de E^{p-1} dans E : nous entendons par fonction de E^{p-1} dans E une application d'une partie de E^{p-1} dans E. Mais les relations peuvent aussi ne pas être fonctionnelles.

Exemple :

Dans l'exemple II.2.3, l'information (tableau d'entiers) est la donnée du triplet (R, V, L) avec :

$R = \emptyset$

$V = \{\text{indices}\} \cup \mathbb{N}$ $\mathbb{N}$ désigne l'ensemble des entiers naturels. Cet ensemble et l'ensemble des indices sont ou non disjoints, au choix de la personne qui définit l'information.

$L = L_3 \quad (P = 3) \quad (L_2 = \emptyset) \quad L_3 = \{F\}$

Précisions sur les repères :

Les repères ont comme rôle de servir d'origine ou de but aux relations d'accès. Pour pouvoir en parler, nous leur avons donné des noms (le plus souvent r_i). Mais ces noms sont en fait inaccessibles, sauf en ce qui concerne certains repères privilégiés comme la tête d'une liste. Les autres repères jouent en quelque sorte le rôle de "variables muettes" dans les cheminements complexes.

Si leur nom est arbitraire, l'existence des repères est cependant essentielle, nous l'avons montré dans l'exemple II.2.1. En confirmation, il n'est que de songer à une information représentée dans une mémoire d'ordinateur : nous avons dit plus haut que les repères étaient en général matérialisés par les mots de mémoire et les informations par les contenus de ceux-ci ; les mémoires n'étant pas en général à accès associatif, ce sont les mots que le programme appelle et dont les inter-relations permettent les cheminements.

Extension de L :

Les éléments du vocabulaire et les repères privilégiés peuvent être considérés comme des relations d'accès définies de E^0 vers E, des "accès directs". Ainsi, pour des facilités de rédaction, ajouterons-nous aux L_p $(p \geq 2)$ l'ensemble L_1 des accès directs. Tout ce que nous dirons au sujet des L_p sera vrai pour L_1 sauf que les éléments de L_1 ne seront pas considérés comme des relations atomiques (cf. § V.1.1).

Nous devons donc corriger l'expression de L et écrire : $L = \bigcup_{p=1}^{\infty} L_p$ avec $L_1 = V \cup \{\text{repères privilégiés}\}$.

II.3.2. GRAPHE GENERALISE.

Il nous a semblé commode, et naturel, de figurer une information (R, V, L) par un graphe $(E = R \cup V, L)$. Mais pour cela il est nécessaire de généraliser un peu la notion de graphe, dans deux voies différentes.

a) Le graphe considéré sera en fait la superposition de plusieurs graphes, un par relation représentée. Chaque arc du graphe sera valué par le nom de la relation dont il fait partie. Entre deux sommets du graphe, pourront bien sûr exister plusieurs arcs, valués différemment (il s'agit d'un multigraphe).

b) La notion de graphe convient parfaitement pour les relations de L_2, mais pas pour celles de L_p $(p > 2)$. Les exemples II.2.2 et II.2.3 peuvent faire deviner la solution adoptée : nous considérerons des "arcs généralisés" (que nous appellerons tout simplement arcs) qui, pour une

relation de L_{p+1}, joindront chacun un p-uple d'éléments de E à un élément de E. Sur les dessins, les p éléments origines seront si possible placés dans l'ordre de leurs rangs dans l'ensemble d'arguments et reliés par une accolade orientée du premier vers le dernier.

Les schémas donnés dans les exemples II.2.2 et II.2.3 représentent de tels graphes (généralisés).

II.4. ALGEBRE SUR LES ACCES.

II.4.1. QU'EST-CE QU'UN ACCES ?

Accéder à une partie de l'information (E, L), c'est, partant d'éléments de V ou de L_j, chercher d'autres éléments par le jeu des relations d'accès. C'est ce "jeu" que nous allons décrire ici.

Nous appellerons (comme [39]) "accès" une partie de l'ensemble E^n. Nous allons étudier comment construire des accès en combinant des relations d'accès élémentaires (éléments de L).

Notation : une relation définie de E^p vers E^n sera notée sous la forme du triplet (p, n, S) où S est la partie de $E^p \times E^n$ constituant la relation.

Un accès sera identifié à une relation (0, n, S).

II.4.2. CALCULS SUR LES RELATIONS D'ACCES.

II.4.2.1. La composition.

La principale opération permettant de bâtir les accès est la composition des relations. Puisque dans le langage PIVOINES, langage d'accès que nous décrirons plus loin (chapitre V), il nous a semblé plus commode d'écrire

la composition de fonctions ou de relations par simple concaténation de gauche à droite (F' o F sera écrit FF'), c'est la convention que nous fixerons dès maintenant. On peut donc écrire :

si $F = (p, n, S)$ et $F' = (n, n', S')$ avec

$S = \{(x,y) \mid x \in E^p, y \in E^n\}$

$S' = \{(y, z) \mid y \in E^n, z \in E^{n'}\}$

alors

$FF' = (p, n', S'')$ avec

$S'' = \{(x, z) \mid x \in E^p \text{ et } z \in E^{n'} \text{ et } \exists\, y \in E^n \ [(x, y) \in S \text{ et } (y, z) \in S']\}$

Nous ne reviendrons plus sur le détail des ensembles S, qui est inutilement compliqué.

Nous pouvons donner maintenant un exemple d'accès : un accès dans l'information donnée en exemple II.2.1 (paragraphe II.2) peut être :

t S S C qui donne comme seule valeur la lettre s.

Cette expression signifie C (S (S (t))). Il est normal de représenter S(t) par tS (ou F(x) par xF) puisque ceci peut être considéré comme la composition de la relation (0, 1, {t}), notée t, et de la relation S = (1, 1, S).

La composition de deux relations S et S' n'est possible que si l'ensemble d'arrivée de S est l'ensemble de départ de S'. Si par exemple S' est une relation d'accès élémentaire appartenant à L_3, c'est-à-dire définie de E^2

vers E, S doit avoir comme ensemble d'arrivée E^2. Or toutes les relations d'accès élémentaires ont E comme ensemble d'arrivée. Il faut donc créer une opération qui permette de bâtir, à l'aide de relations de E^p vers E, des relations de E^p vers E^n. C'est ce que nous appelons la juxtaposition.

II.4.2.2. Juxtaposition de q relations de même arité.

Juxtaposition dans E^+.

Nous poserons d'abord : $E^+ = \bigcup_{k=1}^{\infty} E^k$.

Nous allons définir une opération binaire dans E^+, la juxtaposition. A un élément a de E^p et un élément b de E^q elle associe un élément c de E^{p+q} formé de la concaténation des constituants de a et de ceux de b.

Exemple : si $a = (a_1, a_2)$ et $b = (b_1, b_2, b_3)$ alors

$c = (a_1, a_2, b_1, b_2, b_3)$.

Cette opération est associative et nous parlerons donc de la juxtaposition de n éléments de E^+.

Juxtaposition de q relations de même arité :

Si $F_1, \ldots, F_i, \ldots, F_q$ sont des relations telles que, pour chaque i, $F_i = (p, n_i, S_i)$, alors on notera $\langle F_1, \ldots, F_i, \ldots, F_q \rangle$ et on appellera juxtaposition de ces relations la relation $F = (p, \sum_{i=1}^{q} n_i, S)$ dont les images sont la

juxtaposition (dans E^+) des images par F_1, ..., F_i, ..., F_q.

Exemple 1 :

Supposons que E soit un ensemble de nombres. Si F_1 et F_2 sont respectivement les applications "max" et "min" de E^3 dans E qui à chaque triplet d'éléments de E associent le plus grand et le plus petit d'entre eux, alors $<F_1, F_2>$ est l'application de E^3 dans E^2 qui à chaque triplet fait correspondre le couple formé du plus grand et du plus petit de ses éléments. Par exemple :

(9, 3, 4) a pour image par <max, min> : (9, 3).

Exemple 2 :

Un accès dans l'information de l'exemple II.2.1. peut être formé à l'aide de la juxtaposition ; par exemple : la relation t S S < C , S S C > donnera le couple de valeurs : (s, d).

Remarque :

Soit un p-uple de E^p. Si F_1 en donne m_1 images, ..., F_i : m_i images, ... et F_q : m_q images, alors $<F_1, ..., F_i, ..., F_q>$ donne $m = \prod_{i=i}^{q} m_i$ images du p-uple.

En particulier, si au moins un m_i est nul, il n'y a aucune image.

Cas particulier :

La juxtaposition peut très bien s'appliquer à des fonctions à 0 variable. La juxtaposition de n fonctions à 0 variable dans E est un n-uple de valeurs de E.

Exemple 3 :

L'élément de E^3 cité dans l'exemple 1 peut être noté <9, 3, 4>.

On peut écrire en utilisant la composition :

<9, 3, 4> min = 3 et aussi :
<9, 3, 4> <max, min> = <9, 3>

Exemple 4 :

On peut obtenir un accès par la juxtaposition de plusieurs autres : par exemple, un accès dans l'information citée en exemple II.2.3 (paragraphe II.2) peut être :

<4, 1> F qui donne 31 mais aussi :
<<4, 1> F, <6, 2> F> qui donne le couple <31, 29>.

II.4.2.3. Réciproque.

Une relation R définie de E^p vers E^n est un ensemble de couples formés d'un élément de E^p (p-uple d'éléments de E) et d'un élément de E^n. La réciproque de cette relation est la relation (notée R^{-1}) définie de E^n vers E^p dont les couples sont les (y, x) tels que xRy. Le fait de transformer une relation R en sa réciproque s'appelle l'inversion de R.

Pour les relations définies de E vers E, ou même de E vers E^n, la notion de réciproque est très simple, puisqu'on reste dans le domaine fini.
Seule la réalisation peut être délicate, mais tel n'est pas notre propos dans ce chapitre. Nous adopterons cette opération, en limitant cependant son domaine d'application aux relations d'accès élémentaires : en effet, on pourra toujours se ramener à ce cas. De plus, nous n'utiliserons pas l'inversion lorsqu'il s'agira de relation dont l'ensemble de départ est E^p avec $p \neq 1$. En effet :

- pour $p = 0$: la signification de la réciproque est affaire de convention. On pourrait dire que si a est une fonction à 0 variable dans E^p, a^{-1} est une fonction définie de E^p vers {"vrai", "faux"} qui, à un p-uple, associe "vrai" si ce p-uple est un élément de a et "faux" sinon. Mais nous n'utiliserons pas cela.

- pour $p > 1$: il est probable que la réalisation effective de la réciproque sera beaucoup trop difficile à mettre en oeuvre. Il serait possible de limiter la difficulté - et la portée - en acceptant, pour une relation de E^p vers E^n, de l'inverser à p-1 variables fixées, c'est-à-dire en la considérant en fait comme une relation de E vers E^n. Nous ne retenons pas cette possibilité.

II.4.2.4. Itération.

Il est possible de composer une relation avec elle-même, pourvu que ses ensembles de départ et d'arrivée soient les mêmes. Nous noterons ces compositions par des puissances :

S^2 signifie SS
S^3 signifie SSS
etc ...

Mais il arrive que l'on doive chercher (prenons l'exemple d'une relation de succession) un successeur quelconque d'un élément donné. On posera la définition suivante :

$$a\ S^*\ b \Longleftrightarrow \exists\, k \geq 0 \quad (a\ S^k\ b).$$

Cette opération sur S s'appellera l'itération et S^* sera : "S itéré". Nous limiterons l'emploi de cette opération au cas où S est défini de E vers E.

II.4.2.5. Négation.

Nous poserons : $a \neg R\ b \Longleftrightarrow \neg(a\ R\ b)$ pour tout $a \in E^n$ et $b \in E$.

II.5. STRUCTURE DE DONNEE.

II.5.1. PRESENTATION ET PREMIERE DEFINITION.

De même qu'en Mathématiques on cherche à regrouper les êtres ayant des propriétés communes et en particulier pouvant, dans certaines circonstances, faire l'objet des mêmes raisonnements ou manipulations, de même il est naturel, dans l'étude des systèmes de données, de chercher à assembler les informations qui sont passibles des mêmes traitements.

Par analogie avec les structures algébriques, nous appellerons structure de données l'ensemble des caractéristiques qui font que, pour certains objectifs, les données les possédant se manipulent de la même façon.

L'objectif que nous fixons ici est l'accès ; d'autres (notamment les modifications) pourront s'y ajouter. Dans ce cas, aux critères d'analogie d'accès qui caractériseront chaque structure, devront être ajoutés d'autres critères (analogie de modification) amenant éventuellement à décomposer les structures qui étaient uniquement basées sur l'accès. Nous n'en parlerons pas ici.

Nous avons défini au paragraphe II.4 des opérations permettant de générer des accès à partir des relations d'accès

élémentaires ; les définitions que nous avons données sont générales, indépendantes des propriétés des informations sur lesquelles elles s'appliquent.

Ce ne sont donc pas ces procédés de cnnstruction qui différencieront les sortes d'informations mais les matériaux employés : les relations d'accès élémentaires. La donnée d'une structure d'information consistera, non pas bien sûr en la donnée complète des relations d'accès élémentaires, ce qui déterminerait l'information elle-même, mais en la donnée du nombre de relations d'accès élémentaires de chaque arité et éventuellement aussi de propriétés de ces relations limitant la richesse de leurs combinaisons et reflétant leur "mécanisme".

Nous appellerons "structure libre" une structure dont la donnée ne comporte pas de telles propriétés ; une structure libre est donc seulement la donnée, pour chaque $p \geq 1$, du nombre de relations d'arité p de toutes les informations la possédant. Plus précisément, on dira qu'une structure libre est la donnée d'un ensemble $\Lambda = \bigcup_{p=1}^{\infty} \Lambda_p$ de symboles "relationnels"(1) (qui seront les noms des relations de L) ; tous les Λ_p sont finis, sauf éventuellement Λ_1, et le nombre des valeurs de p pour lesquelles Λ_p n'est pas vide est fini. Le nombre de symboles contenus dans l'ensemble Λ_p sera le nombre de relations d'arité p de l'information.

(1) On peut toujours se ramener au cas où la spécification de la structure comprend celle de V, en élargissant suffisamment cet ensemble.

En résumé, spécifier une structure d'informations consiste à préciser une structure libre et aussi, le plus souvent, des contraintes sur les relations de la structure libre ou leurs combinaisons.

Précisons que la spécification d'une structure pourra aussi contenir des propriétés de décomposition de l'ensemble des repères (exemple § IV.2.1) ; ces propriétés permettront d'introduire des notions du genre des entités de Socrate ou des types ; elles auront donc une grande importance pratique que nous préciserons au cours d'applications ultérieures. Enfin, signalons que c'est au niveau de la spécification de la structure que peut intervenir la décomposition d'une information en informations composantes ou unités de sélection (cf. § V.3.7.2).

Remarque sur le caractère relatif des structures :

Nous avons lié la définition de la notion de structure aux circonstances dans lesquelles les informations sont utilisées : deux informations peuvent ou non avoir la même structure selon l'utilisation qu'on désire en faire.

Remarque sur l'acception du mot "structure" :

Une structure est, nous l'avons dit, un ensemble de propriétés possédées en commun par une classe d'informations. Par abus de langage, nous confondrons classe et propriétés et considérerons aussi une structure comme une classe d'informations. A ce titre, nous parlerons des informations (sous-entendu "faisant partie") de la structure.

Donnons quelques exemples de structures et mettons-y en évidence d'une part la structure libre et d'autre part les contraintes.

Exemple II.5.1.1 : la structure de liste.

La structure de liste est d'une grande importance car on la retrouve très couramment, soit seule, soit faisant partie de structures plus compliquées. L'information donnée dans l'exemple II.2.1 en est une illustration.

Cette structure - au point de vue de l'accès - est caractérisée par le fait qu'on ne peut accéder aux différentes valeurs que les unes à la suite des autres, à partir de la tête. On peut dire qu'on appelle "liste sur un vocabulaire V" une suite d'éléments de V accessibles chacun à partir du précédent, le premier élément de la suite étant appelé tête de la liste.

La structure libre de liste sur V est simplement caractérisée par l'existence :

- d'une tête que l'on désignera par exemple par t ;
- du vocabulaire V ($\wedge_1 = V \cup \{t\}$);
- de deux relations définies de E vers E. Disons que $\wedge_2 = \{S,C\}$ et $\wedge_p = \emptyset$ pour tout $p > 2$.

Mais ces renseignements sont trop vagues et généraux pour bien caractériser la structure. Il faut dire en outre que:

- l'une des deux relations définies de E vers E, soit S, a les propriétés suivantes :
  - . S est une fonction et est définie de R, ensemble des repères, dans lui-même.
  - . S^* est un ordre total d'élément minimal t.
- l'autre relation, C, est une application de R dans V.

Chaque information de la structure de liste s'appelle une liste. Au point de vue des accès (et d'ailleurs des modifications), toutes les listes se traitent de la même façon, seules les réalisations de S et C pouvant différer.

Exemple II.5.1.2 :

L'information "corpus" de l'exemple II.2.2 a pour structure libre :

$|L_2| = 5$ $\Lambda_2 = \{H,F,C,S,K\}$

$|L_3| = 1$ $\Lambda_3 = \{G\}$

$|L_i| = 0$ pour tout $i>3$ $\Lambda_i = \emptyset$ pour tout $i>3$

Λ_1 contient t et les éléments du vocabulaire V.

Les propriétés des relations sont par exemple :

- C et F sont des applications de l'ensemble des repères dans celui des valeurs élémentaires (vocabulaire) ;
- H est une relation de V vers R ;

- S est une fonction définie dans une partie de l'ensemble des repères et son graphe est constitué d'un seul chemin K, qui a pour origine t et ne passe pas deux fois par le même sommet ;
- K est une relation dans R, dont l'ensemble de départ est l'ensemble des sommets du chemin K ;
- le corpus est un ensemble d' "unités de sélection" (cf. § V.3.7.2) et les images de t par S^* en sont les représentants.

Exemple II.5.1.3 : les "catégories formelles" de SYNTOL [26].

SYNTOL est une famille de systèmes documentaires créée par une équipe dirigée par J.C. GARDIN à la Section d'Automatique Documentaire du CNRS à Marseille, dans les années 1961-62. Chaque système est caractérisé par une structure d'information.

Dans certains de ces systèmes, les termes du vocabulaire sont répartis en quatre ensembles disjoints ("catégories formelles") qui reflètent leur nature sémantique :

- . actions (E_1)
- . entités (E_2)
- . états (E_3)
- . prédicats (E_4).

L'emploi et les combinaisons des relations sont limités par des règles dont voici deux exemples :

- la relation binaire P, dite relation prédicative, n'est en fait définie que de $E_1 \cup E_2 \cup E_3$ vers E_4 ;

- il existe une règle, dite règle du développement, qui elle aussi sert à affiner la structure considérée car elle introduit un lien entre trois relations Q, Q', Q" de cette structure : $QQ' \subset Q''$.

Exemple II.5.1.4 :

L'information "état de la mémoire" de l'exemple II.2.4 peut être considérée comme appartenant à une certaine structure de mémoire. Sa structure libre contient les symboles des relations d'adressage, de contenu et de calcul dans B^{ℓ}. Ces relations de plus jouissent de certaines propriétés (que nous ne détaillerons pas).

Nous verrons d'autres exemples au chapitre III, en particulier dans le cas des dossiers médicaux (dans ceux-ci, entre autres, il existe une égalité entre deux relations composées : $D_4\ C = C\ P^{-1}$). Mais en conclusion, disons qu'alors que la donnée d'une structure consiste en la donnée des noms des éléments de L et de propriétés de ces relations, c'est seulement la donnée de chaque information de la structure qui définit complètement une sémantique de L.

REMARQUE : COMMENT UTILISE-T-ON EN PRATIQUE LES SPECIFICATIONS D'UNE STRUCTURE ?

Nous n'envisageons ici que le problème de l'interrogation. Signalons cependant que les spécifications de la structure ont aussi une grande importance dans les problèmes de modification d'informations, en particulier de constitution d'une information, par enregistrement (voir chapitre VI).

- Les noms des relations (éléments de Λ) sont employés pour exprimer les questions.

- Les propriétés sont ou seront employées :

. dans l'expression des questions car elles doivent être respectées ; ceci pourra faire l'objet de vérifications (contraintes d'intégrité).

. pour évaluer "à la main" le déterminisme des relations (si par exemple une relation est une application injective, elle est déterministe) (cf exemple du § IV-2-1).

. pour limiter le champ des recherches (en connaissant par exemple avec précision l'ensemble de départ d'une relation devant faire suite à une relation cherchée).

. pour remplacer certaines recherches par d'autres. Par exemple, dans le cas des relations Q, Q', Q" de SYNTOL (ci-dessus), il peut être intéressant de considérer Q" comme la réunion d'une relation $\hat{Q}$ et de QQ', ce qui permettra de ne représenter en mémoire que $\hat{Q}$ au lieu de Q", et de chercher Q" sous la forme : $\hat{Q}$ ou QQ'(dans le cas où le facteur encombrement est le plus déterminant).

. dans la réalisation de certaines procédures de recherche : ces procédures (cf. § IV.2.2.1) sont évidemment liées à la représentation mais reflètent également des propriétés des relations, ce qu'on pourrait appeler leur "mécanisme" qui est en partie donné avec la structure. C'est au cours d'applications ultérieures que nous pensons approfondir l'étude de ce point.

. dans la détermination d'une représentation adéquate.

. etc...

II.5.2. APERCU DE LA THEORIE LOGIQUE DES STRUCTURES.

Pour développer une théorie des structures et pouvoir faire des démonstrations, établir des classifications, etc..., il a fallu élaborer une définition plus rigoureuse des structures d'informations en termes de systèmes formels [bibl. 23 et 41]. Nous allons donner très brièvement une idée de cette optique ; une différence avec [23 et 41] est que nos relations ne sont pas nécessairement des fonctions.

Il faut signaler que - à la suite des travaux de D. Scott - J.W.de Bakker et W.P. de Roever [bibl. 47] ont développé un système formel de type relationnel que P. Lescanne (à Nancy) essaye d'appliquer plus particulièrement aux structures d'information.

II.5.2.1. Structure d'information. Nouveau point de vue.

Nous pouvons considérer une structure d'information S comme la donnée :

. d'un ensemble $\wedge$ de symboles relationnels ayant chacun une certaine "arité".

. des propriétés des relations qui, dans les données, seront des réalisations de ces symboles relationnels ; on peut considérer l'énoncé de ces propriétés comme des théorèmes.

La notion de système formel [cf. par exemple 32] permet de placer une telle définition dans un cadre plus général. La structure d'information S peut être considérée comme le système formel constitué par le quadruplet (Alpha, $\mathcal{F}$, X, R) dont les constituants sont cités puis commentés ci-dessous :

- Alpha = $\wedge \cup \{ <, >, ,, ^{-1}, *, (,), \underline{\vee}, \underline{\wedge}, \neg, \supset \}$
- $\mathcal{F}$ = ensemble des <u>formules sur $\wedge$</u>
- X = ensemble des <u>axiomes</u> de la structure
- R = ensemble des règles de déduction.

- <u>Formules sur $\wedge$</u> : avec les éléments de Alpha, dont seuls sont spécifiques de la structure les éléments de $\wedge$, on peut a priori bâtir des énoncés, des "formules sur $\wedge$", par le jeu des opérations que nous avons introduites sur les relations ainsi que des connecteurs logiques. On pourrait définir de façon rigoureuse l'ensemble des formules sur $\wedge$ mais nous nous contenterons de donner des exemples.

<u>Exemples de formules</u> :

si $a, b, c \in \wedge_1$

$F, G \in \wedge_2$

$H \in \wedge_3$

alors on peut écrire les formules :

$a\ F\ b$ (ce qui peut se lire : "a et b sont liés par F").

$b\ C\ a$

$a\ F\ G\ a$

$<a\ F\ b\ ,\ c>\ H\ d$

$<a\ F\ b\ ,\ c>\ H\ G\ d$

$a\ G^{-1}\ F^{*}\ b$

On peut construire de nouvelles formules en liant deux énoncés quelconques par $\underline{\wedge}$ (et) ou $\underline{\vee}$ (ou) ou en faisant précéder un énoncé de $\neg$ (non) :

$$a\ F\ b \underline{\wedge} b\ G\ a$$
$$a\ F\ G\ a \underline{\vee} \neg a\ F\ b.$$

On peut construire de nouvelles formules en liant deux énoncés par le symbole d'implication $\supset$:

$$a\ F\ b \supset a\ G^{-1}\ F^{*}\ b$$
$$a\ F\ b \underline{\wedge} b\ G\ a \supset \neg a\ F\ G\ a.$$

- Axiomes de S :

Les axiomes s'expriment par des formules sur $\wedge$.
L'ensemble des axiomes comprend :

. des axiomes généraux qui traduisent les propriétés des symboles $\underline{\wedge}$, $\underline{\vee}$, $\neg$, $\supset$ mais aussi celles des opérateurs que nous autorisons sur les relations (composition, juxtaposition, inversion, itération).

. des axiomes spécifiques de la structure, qui traduisent les propriétés de ses symboles relationnels. Un axiome de la structure de liste pourra par exemple s'écrire : $\neg t\ S^{*}\ t$.

Remarque : les propriétés des opérateurs et symboles relationnels s'expriment en fait le plus souvent par des axiomes dans lesquels on a introduit des variables libres ou par des "schémas d'axiomes" qui permettent d'écrire en une seule fois tout un ensemble (en général infini) d'axiomes.

- Règles de déduction ou d' "inférence" :

Les règles de déduction permettent de déduire des propriétés (exprimées par des formules) à partir de théorèmes : axiomes ou autres propriétés déjà déduites. La plus couramment employée est "modus ponens" : si α et $\alpha \supset \beta$ sont des théorèmes alors β est un théorème.

L'ensemble des théorèmes de la structure S

. comprend les axiomes généraux

. comprend les axiomes spécifiques

. est stable par les règles de déduction.

Ces théorèmes expriment les propriétés des relations.

Remarquons que les seuls constituants du système formel qui soient spécifiques de S sont $\wedge$ et les axiomes spécifiques.

II.5.2.2. Donnée et information, dans cette optique.

Une donnée de la structure S est une réalisation de S. Que signifie ici le terme "réalisation" ?

Donnons-nous un ensemble E et, pour chaque symbole de $\wedge$, une relation dans E de même arité. Toute formule de la structure prend alors la valeur Vrai ou Faux. Nous aurons défini une réalisation de la structure S si et seulement si tous les axiomes de S prennent la valeur Vrai.

Remarque :

Ce que les auteurs appellent généralement une information de S est un ensemble de théorèmes (exprimés par des formules du système formel)qui inclut l'ensemble des

axiomes de S et est stable par les règles d'inférence. Ces auteurs s'intéressent surtout à l'ensemble de toutes les informations de la structure considérée ; ceci n'a pas d'intérêt dans les problèmes qui nous occupent et qui, eux, concernent des données. Nous ne parlerons donc pas des informations en général. A n'importe quelle donnée de S, on peut cependant associer l'information constituée des formules à qui cette donnée donne la valeur Vrai. Mais dans la suite nous ne ferons pas de différence entre une donnée et l'information associée, de sorte que nous emploierons indifféremment les mots "donnée" et "information".

II.5.2.3. Coup d'oeil sur PIVOINES.

Nous verrons, après avoir présenté le langage d'interrogation PIVOINES de façon rigoureuse (chapitre V), qu'une question exprimée en PIVOINES peut être interprétée de manière très simple dans ce nouveau cadre : c'est la spécification d'un ensemble de théorèmes dont on cherche l'intersection avec la donnée interrogée.

III - LES SYSTEMES DE DONNEES . QUELQUES SYSTEMES REALISES PAR NOTRE EQUIPE OU D'AUTRES

oOOo

III-1- DEFINITION DES "SYSTEMES DE DONNEES"

Nous définirons un "système de données" comme un ensemble de moyens pour :

acquérir
interroger
modifier

des données.

Ces termes ont été rencontrés aux chapitres précédents. Revenons-y cependant.

- Les données : cette notion a été définie au chapitre II. Nous avons également défini ce qu'est une structure de données et avons remarqué le caractère relatif de cette notion. Lorsqu'on parle de structure de données, on parle en fait de la structure logique : elle comporte les relations d'accès telles qu'on les conçoit, telles qu'on les formule, en particulier dans les demandes d'accès ; il est fait à ce niveau abstraction de la façon dont elles sont effectivement réalisées ou matérialisées. C'est cet aspect, par contre, qui est pris en considération lorsqu'on s'intéresse à la représentation (interne) des données et à leur forme d'acquisition (forme externe).

La notion de "structure logique" doit être rapprochée des notions de "schéma" de Codasyl [10, 11, 12] et de "conceptual schema" du rapport Ansi-Sparc [3] . Mais pour nous la notion de sous-schéma ou de "user schema" n'est pas différenciée de la première : le sous-schéma et le "user schema" ne sont que des structures logiques dérivées de la structure logique initiale et correspondant à la même structure de représentation.

La plupart des systèmes de données sont spécialement conçus pour des données d'une certaine structure (logique). Ainsi, les relations d'accès font partie intégrante de la spécification de ces systèmes. Qui plus est, ces systèmes sont conçus pour une certaine représentation interne des données : telle relation fonctionnelle se représente par des pointeurs, telle autre par un tableau, etc... Cela n'a d'ailleurs pas une importance capitale pour l'utilisateur qui, le plus souvent, peut ignorer cette représentation. Par contre, ce qui concerne plus l'utilisateur est que généralement le système est conçu pour une certaine forme d'acquisition des données.

- Acquérir : l'acquisition d'une donnée est le fait de la faire passer :

- d'un support externe (externe au système plutôt qu'à la machine, d'ailleurs, car ce support peut être une mémoire, en général secondaire) où elle a une certaine forme (réalisation d'une certaine structure),

- sur un support interne (au système et à la machine) où elle se conforme à une certaine représentation (réalisation d'une certaine structure).

Lorsque forme externe et représentation sont fixées, un programme écrit une fois pour toutes prend en charge l'acquisition.

Dans notre réalisatin, dans laquelle la partie "acquisition" n'a pas été complètement traitée pour l'instant, un "méta-programme" d'acquisition, ayant comme données les descriptions des formes externe et interne, construira le programme d'acquisition (cf § VI-4) ; en fait, ce ne sont pas les descriptions intrinsèques de ces formes qui seront données mais les correspondances entre chacune d'elles et la structure logique.

- Interroger : interroger une donnée, c'est chercher à en extraire des renseignements. Ou bien ces renseignements sont le but en soi de la recherche, ou bien ils sont des données pour des traitements ultérieurs. Mais cette différence n'a guère d'importance en ce qui nous concerne car notre conception est modulaire. Nous nous intéressons au module "interrogation" limité à la production des renseignements auxquels il demande accès, étant bien entendu que ces renseignements auront la propriété d'être communicables à d'autres modules (qui ne nous concernent pas). Nous verrons cependant que le langage de "cadrage" (cf § IV-1-1) permettra d'exprimer quelques traitements sur les renseignements trouvés.

Exprimer une interrogation, une "question", c'est formuler les caractéristiques auxquelles doivent répondre les renseignements à extraire. Lorsqu'on consulte un dictionnaire, on peut chercher l'existence ou la définition du mot qui s'écrit ART, ou de celui qui suit ARSOUILLE, ou du 278ème mot du dictionnaire, ou l'ensemble des mots ayant trois lettres, ou ceux se terminant par T,etc...Nous avons déjà dit (§ II-1)que ce problème était un problème d'accès et que ces accès étaient accomplis en utilisant les relations d'accès élémentaires de la structure de données, composées par les opérations définies sur ces relations. L'exemple du dictionnaire montre que la donnée "dictionnaire" a une structure très riche comprenant entre autres relations :

- l'ordre alphabétique sur les lettres mais aussi sur les mots (on cherche ART dans une table triée) ;

- une relation de succession des lettres et une des mots,qui donnent au dictionnaire une structure de listes de lettres ;

- une relation qui à chaque mot associe son nombre de lettres ;

- etc...

La richesse d'expression des questions dépend :

- de la richesse de la structure de données : si une structure ne comporte qu'une relation de succession entre des mots-clés, on ne pourra pas poser de question sur d'autres relations entre ces mots-clés ;

- de l'outillage offert pour exprimer ces questions et y répondre.

En fait, le problème se pose dans ce sens lorsque l'on utilise un système d'information tout fait (IMS, MAGIS, etc...) mais se pose en sens inverse lorsqu'on veut créer un système : on doit d'abord se forger une idée sur l'éventail de questions que l'on voudrait pouvoir poser puis on en déduit d'une part la structure logique de données et d'autre part l'outillage destiné à l'interrogation.

- Modifier : "modifier une donnée" est une notion facile à comprendre. Mais les problèmes qui se posent sont essentiellement différents selon qu'il s'agit d'effectuer une modification au sein d'une structure fixée ou de modifier la structure elle-même.

Comme la plupart des systèmes de données sont conçus pour une structure, il va de soi qu'ils n'admettent pas de changement de structure (sauf dans les cas rares où seule la sémantique des relations change). Dans un système tel que le nôtre, cette opération est envisageable : nous en parlons brièvement au paragraphe VI-4.

En ce qui concerne les modifications sans changement de structure, de nombreux systèmes en autorisent un nombre restreint : adjonction ou suppression d'un élément, remplacement d'un élément par un autre,... En fait, comme nous l'avons dit au paragraphe II-5-1, la définition d'une

structure doit contenir, en plus des noms de relations et de leurs propriétés, un inventaire des modifications possibles sur les données de la structure. D'ailleurs, la représentation que l'on adopte pour une structure dépend beaucoup des possibilités de modification que l'on désire accorder aux données.

Dans le système PIVOINES, nous n'avons pas encore traité le problème des modifications qui d'ailleurs est très lié à l'accès. Mais nous pensons que notre langage d'interrogation sera un bon point de départ pour l'expression de modifications, et que notre façon d'appréhender les problèmes actuels se prête bien à une extension future.

Remarque 1: qu'est-ce qu'un système documentaire ?

Il est difficile d'établir une comparaison et une différenciation nettes entre système de données et système documentaire. Cependant, on pourrait dire qu'un système documentaire comporte d'une part un système de données et d'autre part éventuellement un "écran" qui permet à l'utilisateur d'ignorer le détail des constituants -en particulier structuraux- du système de données.

Pour reprendre l'exemple du dictionnaire, un système documentaire donnerait des moyens d'interroger le dictionnaire à quelqu'un qui ignorerait quelles sont les relations d'accès mises en jeu pour parvenir à une réponse.

Remarque 2 : système de données et SGBD.

Le terme "système de données" recouvre aussi bien les systèmes très liés à une application, comme ceux que nous avons traités (§ III-2), que les systèmes ayant un certain caractère de flexibilité et d'universalité. Ce sont ces derniers qui méritent le nom de SGBD (systèmes de gestion de bases de données).

III-2- PROBLEMES TRAITES PAR NOTRE EQUIPE.

Nous allons présenter ici deux des systèmes de données élaborés par notre équipe, en passant sous silence des points éventuellement importants pour l'application mais qui nous paraissent peu significatifs pour une généralisation, et au contraire en détaillant d'autres points a priori mineurs pour l'application. Nous essayons aussi d'analyser certaines difficultés rencontrées.

En plus de ces deux systèmes, notre équipe a également réalisé un système d'exploitation de fiches de relevés géologiques [74 et 77] ; ce travail a été exécuté à la demande du CRPG (Centre de Recherche Pétrographique et Géochimique de Nancy, dépendant du CNRS). Le système est en exploitation. De plus, notre équipe a réalisé un système de traitement de documents relatifs à des expériences d'assistance cardio-vasculaire à la demande du laboratoire de Chirurgie Expérimentale du CHU de Nancy ; ce système est prêt à être exploité. Ces deux réalisations ont donné lieu à des thèses de troisième cycle [74 et 75] préparées sous notre responsabilité.

III-2-1- ACTES JURIDIQUES DU MOYEN-AGE [70]

III-2-1-1- Situation du problème.

Il s'agit d'un problème documentaire. Dans le cadre du CRAL (Centre de Recherche et d'Applications Linguistiques) de Nancy, nous avons, en association avec Madame L. FOSSIER (1), décidé d'élaborer un système d'exploitation d'un très gros corpus d'actes juridiques du Moyen-Age -de l'ordre de 150 000 actes- dépouillés par l'IRHT (Institut de Recherche et d'Histoire des Textes). Ces actes notariés sont extrêmement

(1) Attachée à la Section Diplomatique de l'IRHT.

riches en enseignements sur l'époque à laquelle ils ont été émis : non seulement ils renseignent, sur la justice de l'époque mais surtout ils sont une source inépuisable de détails sur les modes de vie, la société, les personnages et même la géographie de la France au Moyen-Age. Nous avons voulu en faire une exploitation "riche" permettant de ne pas se borner à l'essentiel juridique mais de tirer profit de tout le contenu des actes. Il s'ensuit d'une part, à cause de l'étendue du champ, l'obligation de créer un vocabulaire très large dont l'organisation pose de sérieuses difficultés sémantiques (dans lesquelles nous intervenons peu) et d'autre part celle de mettre en place un système de relations entre les termes employés pour préciser leur rôle les uns vis-à-vis des autres.

Notre système fonctionne maintenant complètement et est dans une phase d'"exploitation expérimentale" destinée à rester les différentes hypothèses de travail choisies. Si cette réalisation a duré fort longtemps, c'est en grande partie par manque ou changement trop rapide de "bras" ; mais la phase de spécification du système elle-même qui a fait pourtant l'objet d'une excellente et étroite collaboration entre Madame Fossier et nous-même, a été longue et difficile. Un point intéressant à noter est que, comme dans plusieurs autres de nos réalisations, l'état finalement adopté pour le système est bien plus simple, plus dépouillé, que ne le furent certains états intermédiaires. Bien que nous n'ayons jamais cru en aucune manière approcher la perfection, il nous vient à l'esprit la phrase de Saint-Exupéry dans Terre des Hommes : "Il semble que la perfection soit atteinte non quand il n'y a plus rien à ajouter, mais quand il n'y a plus rien à retrancher".

Ceci prouve en tout cas qu'il y a eu, dans la spécification du système, une phase d'analyse des différentes particularités du problème puis une phase de synthèse visant à minimiser le nombre de dispositifs souhaitables. Nous espérons que notre actuel essai de systématisation tendra à supprimer ces phases analytique et synthétique pour les remplacer par une simple description objective et progressive des particularités du problème.

III-2-1-2- Caractéristiques des documents et des questions.

Exemple très élémentaire de question

MOTS: abbaye, champarts, vin, légumes, verser, particulier,...

NOMS: Saint-Denis, (Demange, Pierre)

RELATIONS: NI(1) abbaye Saint-Denis, NI(1) particulier Demange, SV(2) particulier verser, VA(3) verser abbaye, VØ(4) verser champarts, EP(5) champarts vin, EP(5) champarts légumes,...

Caractéristiques des documents.

Les documents contiennent des données élémentaires (ici abbaye, champarts,...,particulier,...,Saint-Denis,...,Demange,...) mais ces données seules ne suffisent pas, étant donné la richesse d'exploitation que l'on désire (on pourrait imaginer une exploitation beaucoup plus pauvre et moins onéreuse où les notions elles-mêmes et la cooccurrence de certaines notions dans un même document suffiraient).

D'une part on a besoin d'exprimer des relations qui apparaissent dans le document entre des données élémentaires et d'autre part on désire affecter à certaines données élémentaires des noms propres. Par ailleurs, indépendamment des

(1) Relation qui lie un terme à un nom qui le nomme ;
(2) Sujet → Action ; (3) Action → Attribution ;
(4) Action → Objet ; (5) Terme → Prédicat .

documents, les différentes notions ont des rapports sémantiques entre elles et il est utile d'exprimer ceux-ci afin qu'une certaine adaptation ait lieu automatiquement (encore faut-il avoir créé les outils nécessaires) entre des notions employées dans les documents et les questions et ayant une généralité ou une extension différentes (par exemple, si l'on désire être renseigné sur les villas et si un document contient des renseignements relatifs à des immeubles, il ne faut pas rejeter ce document a priori.

Caractéristiques des questions.

Le but d'une question est de sélectionner des documents et, éventuellement, de faire extraire certains renseignements de ceux-ci.

Une question doit pouvoir porter sur les notions (représentées par des "descripteurs" ou "mots clés") mais aussi sur les noms propres et sur les relations.

La réponse aux demandes de descripteurs doit tenir compte des liaisons sémantiques.

L'interrogation des noms propres doit permettre une certaine souplesse ; supposons qu'un document comporte les noms Claude Dupont, Jean Durand. On doit accepter le document (vis-à-vis de cette demande élémentaire) en réponse à la question Dupont ou à la question Claude Dupont mais pas à la question Jean Dupont.

L'interrogation des relations pose beaucoup plus de problèmes, que nous n'exposerons pas ici : il y a un compromis à établir entre une grande finesse d'exploitation, qui d'une part coûte extrêmement cher et d'autre part risque de provoquer certains "silences" (fait qu'un document n'est

pas jugé pertinent alors qu'il le devrait), et une exploitation plus simple donc plus légère et plus sûre -en ce qui concerne les silences- mais risquant de laisser subsister des "bruits" (fait qu'un document est jugé pertinent à tort).

Nous désirions pouvoir poser des questions qui soient constituées de critères successifs (liés logiquement par et) ou alternatifs (liés logiquement par ou). Enfin nous désirions pouvoir poser des questions en cascade, questions de plus en plus fines, et même avoir des"arborescences de questions", plusieurs questions étant des affinements de la même question.

La solution retenue pour les documents.

Cette solution est très fortement inspirée par Syntol [26] .

Un vocabulaire de mots-clés a été créé; il a une structure arborescente. Ce caractère arborescent permet une grande commodité et sobriété de traitement mais est une contrainte très lourde si l'on veut un vocabulaire riche. Nous avons remédié à cette contrainte en décomposant finement les concepts et en donnant un moyen très simple de les associer au niveau des documents.

Les noms propres ont été organisés en deux niveaux : noms simples (Dupont, ou Claude, dans l'exemple donné plus haut) et noms composés formés de noms simples.

Les relations ont été divisées en types à qui l'on a attribué des rôles quasi-grammaticaux :

- la relation de type sujet-verbe
- la relation de type verbe-complément circonstanciel
- ...
- la relation liant un mot-clé à un nom propre.

On énonce pour chaque type les couples de mots liés. Des problèmes se sont posés au niveau de la précision à adopter dans la désignation de chacun de ces mots : par son numéro d'ordre dans la liste des mots du document (solution la plus fine car il est possible qu'un même mot figure plusieurs fois dans la liste), par le mot lui-même par un mot plus global dans la hiérarchie qu'est le vocabulaire,... Pour réaliser le compromis dont il a déjà été question, c'est cette solution qui a été adoptée, l'idée de base de cette décision étant que dans ce problème l'essentiel de la sélection se fait au niveau des mots et que la sélection au niveau des relations est un simple complément.

A l'usage, il s'est avéré que les utilisateurs s'intéressaient plus que prévu aux personnages accompagnés de leur description la plus complète possible. On a donc décidé d'introduire dans le système la notion de "vecteur-personnage" qui revient à superposer au système des relations un autre système qui exprime les relations internes aux vecteurs. Les couples des anciennes relations peuvent être composés non seulement de mots-clés mais aussi de vecteurs (ou même d'ensembles de vecteurs). Cette modification a été lourde à accomplir bien qu'elle ait été facilitée par le caractère très modulaire de la première réalisation.

Solution retenue pour les questions :

Les questions sont exploitées par trains de questions. Chaque question est composée au maximum de quatre parties, chaque partie étant une expression booléenne (avec des restrictions) de critères, plus une partie terminale.

- Dans la première partie, les critères sont des indicateurs. Pour pouvoir combiner entre eux les résultats de plusieurs questions et donner ainsi une grande souplesse au système, nous avons mis en place des "indicateurs" booléens :

ceux-ci sont garnis, à la demande des différentes questions, pour chaque document selon le résultat de la recherche. Cette première partie teste ces indicateurs (qui ont comme portée un train).

- Dans la deuxième partie, les critères sont des mots-clés ou des couples de mots-clés.

- Dans la troisième partie, les critères sont des noms propres.

- Dans la quatrième, les critères sont des relations entre mots de la deuxième partie. Un subterfuge a été mis au point pour exprimer la composition des relations.

- La partie terminale comporte d'une part éventuellement des ordres de chargement des indicateurs et d'autre part des indications sur les renseignements à extraire des documents sélectionnés.

Exemple de question : on cherche la nature des redevances prélevées par les abbayes, et les noms de ces abbayes. Cette demande peut s'exprimer ainsi (la virgule signifie et) :

MOTS : redevance, abbaye, verser
RELATIONS : VA verser abbaye, VØ verser redevance,
EP redevance X, NI abbaye Y
IMPRIMER X, Y

III-2-1-3- La réalisation du système.

La réalisation a été initialement confiée à une promotion d'élèves d'IUT qui malheureusement ont eu trop peu de temps pour mener à bien le travail. Mais ceci nous a poussés à décomposer le problème en sept modules qui ont été traités séparément.

Les différents modules étaient :

- enregistrement des documents.

- traduction d'un train de questions en un "tableau des recherches à effectuer" qui est en fait une liste d'appels de procédure. On peut le comparer au programme objet indéterministe qui, dans notre système général, résulte de l'analyse des questions (cf § IV-2-2-2 et VII).

- prise en compte du tableau des recherches à effectuer, déroulement du fichier de documents.

- quatre modules groupant la réalisation des diverses procédures de recherche et de sortie des résultats.

III-2-1-4- Conclusion.

Ce système est assez général mais comporte cependant certaines particularités qui limitent son utilisation dans d'autres domaines que celui pour lequel il a été écrit. Nous avions pensé en cours de réalisation que, lorsqu'il aurait fait ses preuves, nous essayerions de le généraliser et de le paramétrer suffisamment pour étendre son champ d'application, suffisamment peu cependant pour ne pas avoir à se poser de nouveaux problèmes de conception. Nous espérons que notre système actuel rendra désormais inutile ce genre de reflexion.

Nous avons commencé à étudier la réalisation qui aurait pu être faite du système ancien à l'aide de notre système général et avons pu voir qu'elle aurait été grandement allégée De plus, le fait que les différents réalisateurs qui se sont succédés à temps très partiel après les étudiants de l'IUT aient perdu chaque fois un temps énorme à se mettre au courant des réalisations précédentes aurait été fortement atténué... et il y aurait eu beaucoup moins de réalisateurs successifs.

Mais si, dans la réalisation, notre système actuel permet certainement un très grand allègement, cela doit être encore plus net en ce qui concerne la conception et l'évolution.

III-2-2 DOSSIERS MEDICAUX [71, 72, 73]

Sur la demande du service de diabétologie du CHU de Nancy, dirigé par Monsieur le Professeur G. DEBRY et de Monsieur le Professeur J. MARTIN, notre équipe a élaboré un système de traitement documentaire de dossiers médicaux. Ce travail a donné lieu sous notre responsabilité à deux thèses de troisième cycle [72 et 73] .

Ce problème, par son déroulement, a été une bonne illustration de l'utilité d'un méta-système tel que celui que nous proposons. En effet, si la structure logique des documents a été fixée très rapidement, du moins dans ses grandes lignes, il n'en a pas été du tout de même du langage des questions. Les médecins et les informaticiens ont très longuement et très souvent discuté de l'univers des questions et nous n'avons jamais pu nous mettre d'accord sur une description de celui-ci : ses limites fuyaient à notre approche ; et pourtant nos discussions portaient, ce qui est indispensable, sur des propositions précises à chaque étape. En désespoir de cause, pour ne pas engager de gros travaux sur des bases qui auraient risqué d'être remises en question rapidement, nous avons décidé d'un commun accord de procéder en deux étapes :

- dans la première étape, nous offririons aux médecins un outil proche de la programmation : ils auraient à écrire des programmes sous la forme d'appels de procédures de recherche ; pour cela, nous confectionnerions un ensemble, commode et facile à enrichir, de telles procédures. Cette étape permettrait aux médecins d'une part de pouvoir obtenir rapidement une première exploitation et d'autre part de se forger une idée plus précise de l'univers des questions.

- dans la deuxième étape, un langage d'expression des questions serait fixé et les gros programmes correspondants réalisés.

Ces deux étapes ont été respectées. La première a donné lieu à la thèse de 3ème cycle de J.M. MARTIN [72], la deuxième à celle de P. GERMAIN [73].

Il ne nous semble pas utile de donner ici une description du système car elle existe par ailleurs. Mais nous voudrions en tirer un élément qui a été le point de départ de nos réflexions sur la représentation d'une structure et la structure PHYLOG (cf § IV-2-2 et chapitre VI).

Structure logique et représentation :

Les médecins souhaitaient que les dossiers médicaux aient une forme arborescente car ils ont l'habitude de cette structure pour les dossiers, structure qui est d'ailleurs bien adaptée à la façon dont ces dossiers sont constitués puis évoluent.

Ainsi, à peu de chose près, un dossier est une arborescence ayant un noeud de niveau 1 (c'est-à-dire à un arc de la racine) par rencontre du malade avec le service hospitalier. De chacun de ces noeuds, part un arc par acte médical accompli au cours de cette rencontre. De chacun des noeuds de niveau 2 ainsi atteints part un arc par caractéristique importante (en particulier localisation) concernant cet acte. Puis de chacun des noeuds de troisième niveau partent des arcs vers des noeuds portant les détails concernant ce noeud. Tous les noeuds de l' arborescence portent un descripteur éventuellement qualifié par des "adjectifs" (date,...). Un exemple simple de dossier arborescent est donné au présent paragraphe dans la partie "essentiel de la structure logique".

Si la façon dont se constitue un dossier est fort bien prise en considération dans cette structure, il n'en est pas de même de la façon d'interroger le corpus. L'expérience nous a montré que la plupart du temps ce sont les accès à des noeuds de niveau 3 qui permettent de se situer le plus vite au coeur des parties d'information auxquelles on désire avoir accès. Nous avons donc décidé de favoriser l'accès aux noeuds de troisième niveau, les autres accès s'effectuant à partir de ces noeuds.

Pour les médecins qui, eux,utilisent la structure arborescente, tout se passe comme si elle seule existait. Par contre, la représentation choisie est telle que les accès au troisième niveau seront extrêmement rapides et que les autres accès se feront à partir d'eux. Cela revient à avoir d'une part une structure logique (structure de réflexion et que l'on utilise pour poser les questions) et d'autre part une structure de représentation (qui sera traitée par les programmes d'exploitation obtenus de façon automatique à partir des questions). Nous dirons que la représentation "déforme"la structure. Donnons en quelques mots la description de ces deux structures et la correspondance entre elles.

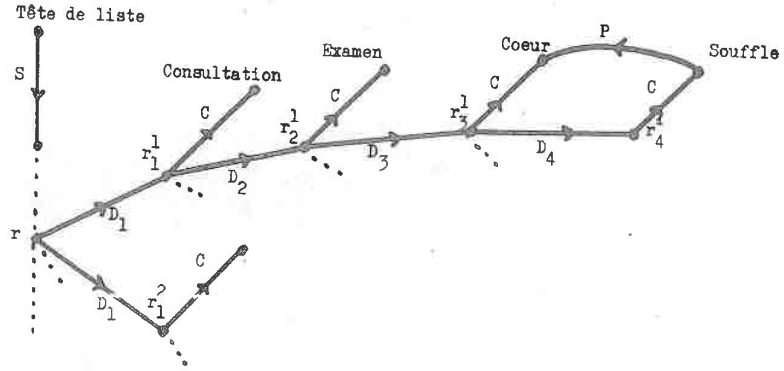
Essentiel de la structure logique :

Le corpus est une liste de documents, chaque document étant considéré comme une"unité de sélection" (cf § II-2 et V-3-7-2) et représenté dans la liste par un repère qui est son "représentant". Ce repère est la racine de l'arborescence dossier. Nous avons appelé D_1, D_2, D_3, D_4 les relations liant respectivement la racine au premier niveau, celui-ci au deuxième, etc... Les noeuds de l'arborescence sont des repères qui sont liés chacun d'une part par une relation C au descripteur étiquetant le noeud et d'autre

part éventuellement, par une autre relation, aux adjectifs qualifiant ce descripteur.

Enfin, une liaison sémantique existe entre les descripteurs de troisième et quatrième niveau : chaque descripteur susceptible de figurer au niveau 4 ne peut y figurer que comme descendant d'un certain descripteur de niveau 3. Nous appelons P l'application ainsi définie de l'ensemble des descripteurs susceptibles d'être utilisés au niveau 4 vers l'ensemble des descripteurs susceptibles de figurer au niveau 3.

Exemple :



La représentation :

La liste de documents est bien représentée par une liste en mémoire : on sait passer d'un document au suivant par une relation S'. Mais à partir d'un représentant de document c'est aux repères de niveau 3 que l'on a accès, à partir de la racine, par une relation A' : en effet, pour chaque document, existe un tableau des noeuds de niveau 3. Chaque élément de ce tableau contient un pointeur vers le noeud de niveau 2 qui est son "père" (appelons B'_1 l'application ainsi représentée), un pointeur vers le noeud de niveau 1 qui est son "grand père" (application B'_2) et -disons-le par simplification- des pointeurs vers les noeuds de niveau 4 qui sont ses"fils" (relation D'). P est représenté par une relation P', C par C'.

Ainsi on peut dire que D_1 est représenté par $A'B'_2$, D_2 par $B'^{-1}_2 B'_1$, D_3 par B'^{-1}_1, D_4 par D'. On peut dire aussi que D_1D_2 est représenté par $A'B'_1$, $D_1D_2D_3$ par A'.

Parmi les détails que nous pourrions fournir sur cette représentation, donnons-en un qui nous semble intéressant : la représentation est telle que la traduction de C^{-1} pour un descripteur de niveau 4 donnerait lieu à des recherches extrêmement lourdes si elle était traitée par inversion de C ; on donnera donc, dans la description de la représentation, la représentation propre de C^{-1} pour le niveau 4, de la façon suivante :

- si C^{-1} est suivi de D_4^{-1} :

  $C^{-1}D_4^{-1}$ sera traduit par la relation C'_5 qui est définie par :

  $C'_5(z) = u$ tel que $zP'C'^{-1}_3 u$ <u>et</u> $u\,D'C'_4 z$

  (si l'on appelle C'_3 et C'_4 les représentations de C pour les niveaux 3 et 4).

- si C^{-1} n'est pas suivi de D_4^{-1}:

  C^{-1} sera traduit par la relation C'_6 qui est définie par :

  $C'_6(z) = x$ tel que $zP'C'^{-1}_3 D'x$ <u>et</u> $xC'_4 z$.

Nous verrons au paragraphe VII-4-2 comment de telles représentations conditionnelles sont exprimées et prises en compte.

III-3- QUELQUES SYSTEMES DE DONNEES OU MODELES "CLASSIQUES", VUS SOUS L'ANGLE DE LA STRUCTURE DES DONNEES.

III-3-1- SYSTEMES A STRUCTURE DE DONNEES FIXEE.

La plupart des systèmes, avons-nous dit en III-1, sont construits pour une structure de données bien définie . C'est le cas de nos systèmes décrits en III-2. Cette structure, selon les systèmes, est plus ou moins riche d'une part, plus ou moins générale d'autre part. Citons quelques systèmes à structure fixée ; cette liste n'est bien sûr ni limitative ni très significative.

a) Les systèmes très simples destinés en général à la bibliographie, en particulier les systèmes les plus élémentaires de la famille SYNTOL [26] , considèrent les documents comme des ensembles de descripteurs. L'information traitée, le corpus, a par exemple une structure de liste d'ensembles (ou de liste de listes) de descripteurs ; ces descripteurs font le plus souvent partie d'organisations sémantiques. Cette structure est simple et générale.

b) Dans d'autres systèmes, comme le système MISTRAL [65] de la compagnie CII, chaque document est décomposé en champs, chaque champ étant constitué comme un document décrit en a). Ces systèmes, eux aussi, sont simples et généraux.

c) La plupart des systèmes des constructeurs d'ordinateurs (INFOL de Control Data [60] , TDMS et IMS d'IBM [61, 62], ...) ont des langages d'interrogation et de mise à jour assez évolués et sont très étudiés au point de vue programmation et gestion de fichiers. Mais la structure des informations qu'ils traitent est presque toujours arborescente (simple et générale, là encore).

d) MIISFIIT [7] travaille sur des données ayant une structure fixée mais assez riche qui est la superposition de plusieurs structures arborescentes. Ce système possède des dispositifs intéressants pour l'acquisition, l'interrogation et la mise à jour des données.

e) Les systèmes les plus élevés dans la gamme SYNTOL [26] traitent des documents qui sont constitués de plusieurs champs, et en particulier d'un champ de relations exprimant des liaisons (dont on précise la nature) entre les descripteurs . Cette structure complexe est très générale et peut répondre à presque tous les besoins de description de documents (s'ils ne comportent pas de fonctions à plusieurs variables). Cependant, le fait de ne pas dissocier la notion de structure logique de celle de représentation rend le système trop figé et lourd, par obligation d'expliciter toutes les liaisons.

III-3-2- SYSTEMES A STRUCTURE DE DONNEES NON FIXEE.

Certains systèmes n'imposent pas de structure mais seulement un modèle de description de celle-ci (voir paragraphe suivant) ; ceci rend leur usage bien plus général.

f) LEAP [22] : dans ce système, les documents sont formés d'"associations", c'est-à-dire d'affirmations construites sur le modèle suivant : une image du mot M, par la relation R, est M' ; une association est la donnée du triplet (R, M, M'). Ce procédé donne une très grande souplesse au système car il permet d'introduire à volonté toutes les relations (binaires) voulues. De plus, l'algèbre des relations est assez developpée.

g) MAGIS [63] : ce système créé par Philips pour la série P/1000 est conçu pour une structure de base arborescente mais superpose à celle-ci des relations supplémentaires représentées par des "chaînages logiques". Ce qui est appelé ici le DDL (Data Description Language) est principalement un outil de descriptions d'organisation physique.

h) SOCRATE [1] : nous parlerons au paragraphe III-3-3 du modèle de structure de données de SOCRATE. Des traits marquants de ce système sont qu'il permet d'exprimer et de respecter des propriétés des relations (cardinaux minimal et maximal des ensembles d'images des valeurs par les relations d'accès, symétrie de celles-ci,...) et, qu'en plus des requêtes complexes qu'il permet d'exprimer, il est particulièrement bien armé pour les modifications.

i) D'autres systèmes plus évolués encore peuvent être qualifiés de "système inférentiels", comme SYNTEX [38] , PLANNER [29] en ce qui concerne sa phase de "pattern matching", les travaux de A. Waksman [55] ,... . Ils utilisent des procédés voisins de la déduction automatique et font le lien entre les SGBD et les tentatives de démonstration automatique.

III-3-3 DIVERSES FORMALISATIONS ET ETUDES GENERALES DES SYSTEMES DE DONNEES.

Divers groupes ou personnalités ont effectué des études générales sur les systèmes de données et en particulier leur décomposition en niveaux les plus indépendants possible (conception, représentation,...) et les caractéristiques qu'ils doivent posséder à chaque niveau (cf [2] , [3] , [8 à 19] , [21] , plus anciennement [26] , [28] , [29], [35] , [43] , [52] , [53] , [55] .

Dans le cadre de notre travail, nous ne nous intéressons pour l'instant qu'à la phase se situant en aval de la spécification d'une structure logique : nous négligeons volontairement la phase intéressante qui amène à cette spécification à partir du "réel perçu". La charnière entre ces deux phases est constituée par ce qui est l'essentiel dans un système de données : le modèle choisi pour décrire la structure logique. Nous allons citer les plus importants de ces modèles en les comparant, ce qui a d'ailleurs déjà été fait, en particulier par C. DELOBEL [19]. Disons tout de suite que ces modèles ont tous la même puissance d'expression.

Les principaux modèles que l'on rencontre, en dehors des modèles hiérarchiques, sont de deux types : les modèles en réseau et les modèles relationnels. D'autres sont basés sur la théorie des ensembles ([9]), d'autres sont des variations sur ces thèmes ([5 bis] ,...). Essayons de décrire très schématiquement les deux premiers modèles et de nous situer par rapport à eux.

De nombreux substantifs (objets, entités, données, records,...) ont été employés pour désigner dans les descriptions de ces différents modèles les unités d'information considérées, et ces substantifs ont souvent un sens différent d'un emploi à l'autre ; nous les éviterons ici pour ne pas risquer d'ambiguïté.

Les modèles en réseau [19, 1, 10, 11, 12, 63].

Dans les modèles en réseau —et en particulier SOCRATE—, mais pas seulement dans ceux-ci, on considère deux niveaux de décomposition de l'information et par conséquent deux niveaux d'unités d'information et deux niveaux de relations. Ainsi, on considère ce qu'on pourrait appeler -par analogie avec les variables et procédures en programmation- des concepts globaux (au niveau de décomposition le moins fin ; par exemple : le concept de personne, de commande,...) et des concepts locaux à chaque concept global (âge, numéro de commande,...). Chaque occurrence g d'un concept global est composée (d'une certaine façon, la même pour toutes les occurrences du même concept global) d'une arborescence d'occurrences de concepts locaux ; si g disparaît, les occurrences correspondantes des concepts locaux disparaissent. Les relations au niveau supérieur sont des relations entre concepts globaux ; les relations au niveau inférieur sont des relations de décomposition arborescente de chaque concept global.

Les modèles en réseau diffèrent les uns des autres par la nature des relations au niveau supérieur (relations les plus générales, ou uniquement relations fonctionnelles,..., relations nommées ou toutes confondues,...). Dans le système SOCRATE [1, 19], celles-ci sont exprimées par des références entre concepts globaux ; elles sont nommées (noms d'anneaux). Les relations que l'on peut exprimer ainsi sont uniquement des relations fonctionnelles et leurs inverses. Cependant toute autre relation peut être exprimée grâce à un subterfuge : l'emploi de concepts globaux auxiliaires qui en réalité sont l'abstraction de n-uples. Le modèle du Data Base Task Group de Codasyl [11, 12] est très voisin de celui de SOCRATE.

Les deux principales critiques que l'on peut formuler au sujet des modèles en réseau ne sont en fait pas complètement liées à ces modèles : la première porte sur le double niveau

de décomposition, la seconde sur les limitations apportées à la nature des relations au niveau supérieur, c'est-à-dire à la restriction des modèles en réseau existants par rapport à un modèle général. Nous ne détaillerons pas cette seconde critique. Quant au double niveau de décomposition, s'il reflète bien le "réel perçu" de prime abord, il se justifie moins dans la structure logique élaborée. On pourrait très bien rendre ces modèles plus homogènes :

- en considérant tous les concepts sur le même plan et en exprimant par des contraintes d'intégrité (c'est-à-dire, dans notre modèle, par des propriétés des relations) et au niveau des modifications, que les unes sont locales à d'autres,

- en considérant également toutes les relations sur le même plan.

L'homogénéité du modèle, en plus de la simplification intrinsèque qu'elle apporterait, permettrait d'éviter de se poser des questions "artificielles" et souvent gênantes en pratique, comme celle de savoir si une relation d'inclusion doit être exprimée au niveau inférieur (le concept origine sera alors un concept local) ou au niveau supérieur (le concept origine sera alors un concept global).

Si les deux niveaux ont été introduits, c'est primo parce qu'ils sont naturels au niveau de la perception de la réalité, secundo parce que les systèmes conçus avec ces modèles ne l'ont pas été avec un souci de grande indépendance entre structure logique et structure physique (interne). Ainsi, choisir de placer une relation à un niveau ou à l'autre permet de choisir la façon dont elle doit être représentée.

Une troisième critique est quelquefois adressée aux systèmes en réseau : celle de reposer sur des relations d'accès et non sur des relations au sens des modèles décrits ci-dessous. Nous pensons que seul est regrettable le fait que ces relations d'accès soient trop liées aux accès physiques.

Les modèles relationnels [19, 13 à 16, 43, 44, 53].

Les modèles relationnels, dont celui de Codd [13 à 16], contrairement aux modèles précédents qui sont hétérogènes et trop orientés vers la description d'accès physiques, sont homogènes et ne sont basés ni sur les accès physiques ni même sur les accès. On y définit des concepts et des relations, celles-ci étant des parties de produits cartésiens de concepts (concepts appelés "domaines" des relations). La notion d'accès intervient cependant lorsque l'on introduit des index.

Notre modèle :

Notre modèle peut être considéré comme intermédiaire entre les modèles précédemment décrits. Il est cependant plus proche du second.

Comme le modèle en réseau (dont il peut très bien rendre compte), il est basé sur l'existence de relations d'accès, orientées ; mais il s'agit là d'accès logiques ne préjugeant absolument pas des accès physiques (cf chapitre VI). Ainsi, il ne nous semble pas que puisse s'appliquer à notre modèle la remarque de C. Delobel [19, p. 108 n°6] exprimant que "le modèle relationnel permet une meilleure "data independence" que les autres et, ne prenant en compte aucun élément sur les chemins d'accès aux données, donne à l'utilisateur plus de liberté pour

introduire de nouvelles relations entre les informations". Notre modèle laisse dans ce domaine tout autant de liberté que celui de Codd.

Comme le modèle relation de Codd, notre modèle place toutes les relations sur le même plan. Les relations sont peu différentes de celles du modèle relationnel ; simplement elles privilégient certains domaines par rapport à d'autres. Cette différence avec le modèle relationnel, qui est très peu conséquente au niveau de la description des données, l'est plus en ce qui concerne le langage d'interrogation. Nous n'utilisons pas, comme le fait E.F. Codd, d'opérations sur les ensembles ; nous pensons, plus que celui-ci, à l'aspect "cheminement". Mais, répétons-le, il s'agit de "cheminements logiques".

Ainsi notre modèle nous semble aussi général que celui de Codd. Nous espérons -et avons l'intention de vérifier- qu'il constitue, comme le dit E.F. Codd [13], "une base saine pour traiter la dérivabilité, la redondance et la consistance des relations". Nous pensons d'ailleurs bientôt soumettre notre modèle au test que constituent les critiques et signalements de danger émis par E.F. Codd dans [13]. Nous ne nous sommes pas intéressée pour l'instant aux problèmes liés à la normalisation car ils concernent principalement l'amont de la conception d'une structure logique. D'ailleurs, le fait que E.F. Codd normalise ces relations rend celles-ci plus proches des nôtres car plus propices aux accès (logiques).

IV. SCHEMA GENERAL DU DEROULEMENT DE L'INTERROGATION D'UNE INFORMATION - STRATEGIE DE RECHERCHE.

oOOo

Ce chapitre est une "vue d'avion" de notre travail et son développement fait l'objet de toute la suite de cette thèse. Les différentes étapes du chapitre sont résumées dans la figure donnée au paragraphe IV - 4.

IV-1- COMMENT EXPRIME-T-ON UNE DEMANDE D'INTERROGATION ?

IV-1-1- DECOMPOSITION D'UNE DEMANDE EN DEUX PARTIES.

Les différents exemples que nous avons donnés et en particulier ceux du chapitre III font ressortir nettement la présence dans une demande d'interrogation de deux spécifications complémentaires :

- l'une est fortement liée à l'information ; elle y définit des accès ; nous appellerons cette partie de la question la partie "accès" ;
- l'autre sert à contrôler (au sens de "control" en Anglais) ces accès, ainsi qu'à exercer des actions sur leurs résultats ; nous appellerons cette partie la partie "cadrage".

Exemple :

Considérons l'interrogation suivante : "dans telle information I (constituée par une liste de dossiers médicaux), sélectionner les numéros des documents contenant tels et/ou tels renseignements, les compter et calculer la moyenne de

l'âge des patients au moment de la constitution de leur dossier ; puis, parmi les documents ainsi sélectionnés, déterminer ceux pour lesquels le malade est une femme ayant eu des enfants et (pour l'ensemble de ces dossiers) dresser une table de contingence du poids de naissance des enfants (par intervalles de 50 g) en fonction de leur rang dans la famille".

Nous avons souligné les éléments de la question qui appartiennent à la partie "accès". Ils font intervenir uniquement des renseignements que l'on espère trouver dans l'information I.

Tout ce qui n'est pas souligné est du domaine du "cadrage". Il s'agit de :

- "dans telle information I" : on conviendra de considérer comme faisant partie du "cadrage" la spécification de l'information à considérer ; cela fait partie du "contrôle" des accès.
- "les compter" : action sur le résultat d'un accès.
- "calculer la moyenne" : action sur le résultat d'un accès
- "parmi les documents ainsi sélectionnés" : il s'agit encore de "contrôler" un accès.
- "dresser une table de contingence de... (par intervalles de 50 g) en fonction de..." : action sur des résultats d'accès et contrôle de ces accès.

Nous ne nous intéressons pas pour l'instant aux parties "cadrage" des questions. Il est difficile de dresser une liste exhaustive de leurs fonctions et même quelquefois de faire une nette séparation entre l'aspect "contrôle d'accès" et l'aspect "traitement des résultats d'accès". Parmi les fonctions de contrôle, on peut cependant citer : la gestion d'un train de questions, indépendantes ou non, c'est-à-dire

l'articulation des diverses questions qu'il comporte, le chargement d'indicateurs (cf actes du Moyen-Age dans le chapitre III), la gestion des fichiers où se trouve l'information, mais aussi la déclaration de types ou de conditions (valeurs limites, ...) pour les inconnues. Parmi les fonctions de traitement des résultats des accès, citons les calculs ou traitements classiques que l'on peut effectuer sur des informations numériques ou alphanumériques, les éditions, les tris, la fabrication de sous-fichiers. Il est probable que, pour représenter ces fonctions de la partie "cadrage" il suffira de mettre en place quelques outils simples que compléteront des parties classiques de langages de programmation (déclarations de type et de procédures, fonctions de calcul).

La présente thèse est consacrée à l'expression et au traitement des accès dans l'information. Nous avons créé un langage unique, PIVOINES, permettant d'exprimer pratiquement toute demande d'accès dans une information quelconque. Sa description, ébauchée dans les paragraphes suivants, fait l'objet du chapitre V.

Nous avons bien sûr comme projet de dessiner et bâtir de nouveaux systèmes de données à l'aide des outils que nous avons forgés. Dans ce cadre, le langage PIVOINES sera utilisable directement de façon très commode, les parties "cadrage" des questions étant dans un premier temps traitées de façon "ad hoc". Mais nous voulons également prendre en charge des systèmes existant déjà. Leur langage d'interrogation sera alors traduit (en ce qui concerne l'aspect "accès") dans le langage PIVOINES qui jouera alors le rôle de langage pivot ; d'où son nom : langage PIVOt pour l'INterrogation d'Ensembles Structurés. En fait, même lorsque nous créerons de nouveaux systèmes, le langage d'interrogation pourra très bien être différent de PIVOINES qui alors jouera encore le rôle de langage pivot.

IV-1-2- INDEPENDANCE DU LANGAGE D'ACCES VIS-A-VIS DE LA STRUCTURE.

Pour exprimer une certaine demande d'accès dans une information I dont on connaît la structure, le vocabulaire et la représentation,il est bien sûr possible d'écrire un programme en toutes lettres. Pour toute autre demande, toute autre structure, ou même toute autre représentation, il faut écrire un nouveau programme.

Une solution plus puissante et en général plus économique est de créer, pour une structure donnée, une représentation donnée et une certaine famille de demandes d'accès, un langage permettant d'exprimer ces demandes puis un compilateur destiné à générer des programmes d'accès. C'est ce qui est réalisé dans tous les systèmes de données existants et en particulier ceux cités au chapitre III. Ces systèmes ont chacun nécessité de très gros efforts de programmation et, auparavant, de spécification de tous leurs paramètres : en effet, si l'on est amené à modifier l'un de ces paramètres -un détail de la structure, de la représentation,...-, cela risque d'entraîner de très importantes modifications des programmes.

Notre réalisation est, comme nous l'avons déjà dit, en fait un méta-système permettant de définir et de réaliser très simplement pour chaque besoin un système particulier. Il devrait donner la possibilité de ne pas prendre dès l'abord de décisions définitives concernant les structures et représentations pour un problème donné. Voici pourquoi : le langage d'accès PIVOINES est en fait une charpente qu'il faut garnir avec les éléments de la structure de données concernée. Nous donnons en effet au Chapitre V une grammaire de ce langage

et celle-ci est incomplète : ses terminaux, pour la plupart, ne sont pas précisés, ils seront donnés par la structure. Mais ce qui est essentiel, c'est que le compilateur, c'est-à-dire le programme qui prend en compte les demandes d'accès exprimées en PIVOINES pour générer les programmes d'exploitation, est écrit une fois pour toutes, indépendamment de la structure qui en est une donnée.

IV-1-3- NATURE DESCRIPTIVE DU LANGAGE.

Comme nous l'avons énoncé dès le premier paragraphe de cette thèse, nous avons souhaité élaborer un langage descriptif et non actif. C'est ce qui a été réalisé : on ne décrit dans une demande d'accès que le fragment d'information dont on veut chercher des occurrences dans l'information interrogée. Il s'agit de ce qu'on appelle en Anglais "pattern matching" et cela généralise ce que l'on trouve dans Snobol [27 et 31], Planner [29] , etc... Ceci suppose que le programme de traduction décidera lui-même de la façon dont ces occurrences seront recherchées (nous parlerons de ceci en détail au paragraphe IV-2 et surtout au Chapitre VII). Cependant, comme nous le verrons au paragraphe IV-1-5, nous avons ressenti le besoin de laisser à l'utilisateur un certain pouvoir sur le déroulement de la recherche dans l'information.

Mais qu'est-ce qu'une demande d'accès ?

A première vue, on peut distinguer deux sortes de demandes d'accès dans les interrogations d'informations :

A - celles qui consistent à chercher la présence pure et simple d'un fragment d'information donné Q dans l'information interrogée I ;

B - celles qui consistent à chercher, pour les occurrences dans I d'un certain fragment d'information, les éléments de I qui y sont liés d'une façon donnée.

Exemple IV-1-3-1 :

Supposons que l'information I interrogée ait une structure de liste sur un ensemble d'entiers.

Demande illustrant A : y-a-t-il dans I deux éléments consécutifs de contenus respectifs 7 et 4 ?

Demande illustrant B : chercher dans I s'il existe des couples d'éléments consécutifs de contenus respectifs 7 et 4 ; si oui, pour chacun de ces couples chercher le contenu de l'élément suivant.

Dans le modèle A, la réponse à la demande est "oui" ou "non". Dans B, la réponse est d'une part "oui" ou "non" et d'autre part, pour une réponse affirmative, l'ensemble des éléments qui conviennent. On peut très bien fondre ces deux modèles en un seul : le modèle B élargi au cas où l'on ne recherche aucun élément de I lié au fragment.

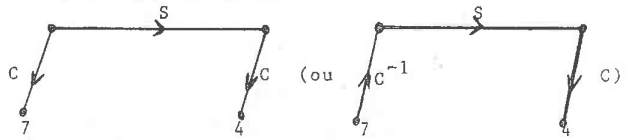
Nous verrons au Chapitre V comment rattacher cette notion de demande d'accès à celle d'accès définie au paragraphe II-4-1 et à celle de théorème (§ II-5-2-1).

IV-1-4 POCHOIRS ET INCONNUES.

On donne une bonne idée d'une demande d'accès en considérant le fragment Q que l'on recherche comme un pochoir, ou un masque, que l'on applique sur l'information : les parties pleines du pochoir correspondent à des morceaux d'information dont la présence dans l'information interrogée I sera imposée, et les parties creuses à des valeurs qui transparaîtront à chaque application du pochoir, c'est-à-dire à chaque occurrence du fragment Q ; plus précisément, ces "fenêtres" sont définies par leurs liens avec les parties imposées et, pour chaque occurrence, laisseront apparaître les valeurs passibles de ces liens dans l'information I (cf par exemple [29 p. 24] et [27, 31, "pattern" de Snobol]). Chaque "fenêtre" sera appelée une inconnue. Donnons aux inconnues dès maintenant les noms qui leur seront donnés dans notre langage : x_i (i = 1,2,3,...).

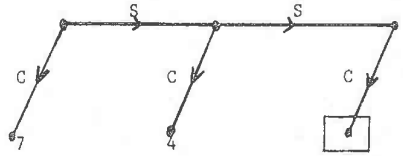
Exemple IV-1-4-1 :

- la demande (pour A) de l'exemple IV-1-3-1 peut être figurée par le pochoir :



Ce pochoir ne comporte qu'une partie pleine. La réponse à cette question sera : "oui" ou "non".

- la demande (pour B) du même exemple peut être figurée par le pochoir :



Le carré met en évidence la partie creuse du pochoir, c'est-à-dire l'inconnue (appelons-la x_1). La réponse à cette demande sera constituée de deux parties :

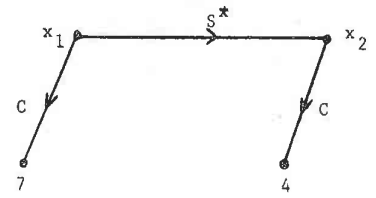
. oui ou non selon que l'on trouve au moins une occurrence ou non ;

. si oui, l'ensemble des valeurs transparaissant dans la partie creuse, c'est-à-dire des "valeurs de l'inconnue x_1". Par exemple, si l'information I est la liste constituée d'éléments ayant pour contenus successifs 1, 7, 6, 5, 7, 4, 2, 11, 4, 8, 7, 4, 7, la réponse à la demande sera :

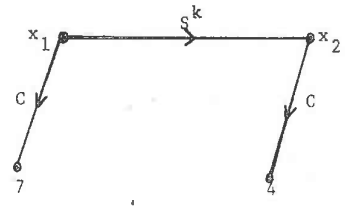
<u>oui</u> et $\{x_1\} = \{2, 7\}$.

Exemple IV-1-4-2 :

L'information I, ici encore, est une liste. On cherche les couples de repères ayant pour contenus respectifs 7 et 4, le repère de contenu 4 étant un successeur quelconque de celui de contenu 7. Le pochoir peut être figuré par :



La partie booléenne de la réponse sera oui s'il existe au moins un couple de repères (x_1, x_2) et un entier positif k tels que le fragment suivant ait une occurrence dans I :



Pour l'information donnée à la fin de l'exemple précédent, dans laquelle nous noterons les repères successifs 1, 2, 3, ..., 13, la réponse est : oui, et $\{(x_1, x_2)\} = \{(2,6),(2,9),(2,12),(5,6),(5,9),(5,12),(11,12)\}$.

Nous donnerons d'autres exemples au paragraphe V-1-2.

Remarque :

Chaque "fenêtre" ne laisse transparaître qu'un sommet du graphe information. Cette restriction simplifie grandement les traitements et n'offre pas de gros inconvénient :

1) si l'on désire que ce soit un arc qui soit "en creux", on peut exprimer cela par une disjonction des différents arcs possibles (en provoquant l'affectation à une certaine inconnue d'une valeur indicative) ;

2) si l'on désire que le fragment "en creux" soit plus important, tout en ayant une composition donnée (par exemple deux sommets liés par un arc), on le décomposera en chacun de ses éléments.

IV-1-5- PARTAGE DES RESPONSABILITES DE CHOIX D'UNE STRATEGIE DE RECHERCHE : FILTRES ET MOTIFS.

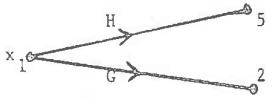
Nous traiterons au paragraphe IV-2, puis de façon détaillée au chapitre VII, de la décomposition d'une demande d'accès en demandes élémentaires et de l'ordre dans lequel seront traitées ces demandes élémentaires. Cette décomposition et cet ordre constituent la stratégie de recherche.

Au paragraphe IV-1-3, nous avons dit que c'est pour une large part le programme traducteur qui a la responsabilité du choix de cette stratégie. Dans le langage PIVOINES, les pochoirs sont représentés par ce que nous appelons des motifs. Et c'est effectivement le programme traducteur qui a la responsabilité totale de la stratégie qui sera appliquée à la recherche relative à chaque motif, à un détail près dont nous ne parlerons pas ici, le choix des "extrémités" (cf §IV-2-3).

Par contre, nous avons choisi de laisser à l'utilisateur une part de responsabilité dans le choix de la stratégie et, pour réaliser cela, d'organiser chaque demande d'accès en deux niveaux partiellement redondants : une demande sera exprimée par une expression booléenne -au sens d'Algol 60- de motifs; c'est cela que nous appellerons un filtre. Avant d'expliquer en quoi les deux niveaux d'organisation -motif et filtre- sont partiellement redondants, expliquons comment cette double organisation permet un partage de responsabilité. Chaque demande d'accès exprimée par un filtre donnera naissance à un programme de recherche. Or ce programme sera principalement constitué des programmes de recherche correspondant à chaque motif du filtre, ces programmes n'étant pas entremêlés (ni à l'écriture ni à l'exécution). La stratégie de chacun de ces programmes aura été déterminée par le programme traducteur mais l'ordre de ces programmes -à l'écriture comme à l'exécution- sera celui dicté par l'ordre des différents motifs dans le filtre. Or lorsque l'utilisateur écrit le filtre, il choisit l'ordre des motifs ; c'est ainsi qu'il influera sur la stratégie de recherche.

Mais revenons à la redondance : nous voulons dire par là que les moyens d'expression offerts par les motifs et les filtres ne sont pas disjoints. Donnons un exemple :

Soit une structure dans laquelle, en particulier, existent deux fonctions, H et G, non partout définies, de l'ensemble des repères dans un ensemble d'entiers. On désire chercher les repères dont l'image par H est 5 et dont l'image par G est 2. Cette demande très simple peut être exprimée par le filtre F formé du motif unique M qui peut être figuré par le pochoir :



Mais on peut aussi représenter la demande par le filtre F' = M_1 <u>et</u> M_2, les deux motifs pouvant être figurés respectivement par les pochoirs :

Dans les deux cas, la réponse finale sera la même, bien sûr. Mais, alors que dans le premier cas c'est le programme traducteur qui choisira le meilleur ordre entre les deux recherches élémentaires, dans le deuxième cas cet ordre est imposé par l'utilisateur : on recherchera d'abord les valeurs de x_1 satisfaisant au premier motif puis, pour chacune d'elles, on cherchera si elle satisfait au deuxième. Remarquons que, comme nous le dirons plus systématiquement en particulier au paragraphe V-5-3, x_1, qui est une inconnue pour le premier motif écrit, sera en fait considéré comme une donnée pour le deuxième.

D'une façon générale, plus la demande sera décomposée en motifs différents, plus grande sera la responsabilité de l'utilisateur sur le choix de la stratégie de recherche. Si celui-ci veut se décharger au maximum de ce choix, il exprimera sa demande sous forme d'un filtre comportant les motifs les plus gros et les moins nombreux possible.

IV-2- TRADUCTION DE LA DEMANDE EN UNE SUCCESSION DE DEMANDES ELEMENTAIRES, AVEC CHOIX D'UNE STRATEGIE.

IV-2-1- POURQUOI PARLE-T-ON DE STRATEGIE ?

Nous allons donner un exemple montrant qu'à une demande (descriptive) donnée peuvent être associés plusieurs programmes de recherche de qualités inégales en ce qui concerne les performances.

Considérons par exemple la structure S définie par :

- un certain ensemble R de repères ;
- un certain ensemble V d'informations élémentaires (vocabulaire) ;
- un certain ensemble L de relations :

$L = \bigcup_{p=1}^{\infty} L_p$ avec L_1 non précisé

$L_2 = \{A, B, C\}$

$L_i = \emptyset$ pour tout $i > 2$

- les propriétés suivantes :

R = R1 U R2 U R3 les ensembles Ri sont disjoints

E = E1 U E2 U E3 les ensembles Ei sont disjoints

A est une application de R1 dans R2

B est une application injective de R2 dans R3

C = C1 U C2 U C3 avec C1 application de R1 dans E1

C2 - - R2 - E2

C3 - - R3 - E3

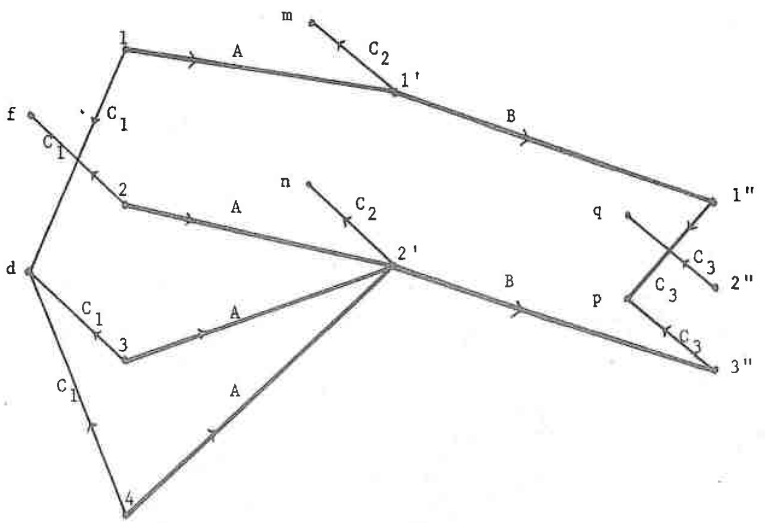
C sera appelé : l'application "contenu".

On suppose de plus que les deux hypothèses suivantes sont vérifiées :

(hypothèse h) : R1 et R3 ont de nombreux éléments, contrairement à R2 ;

(hypothèse h') : les contenus des éléments de R1 sont peu variés, contrairement à ceux de R3.

Exemple d'information de la structure S :



Représentation :

Chaque information de la structure S est (par exemple) représentée sous la forme de trois tableaux T1, T2, T3 : chaque élément de Ri est un indice du tableau Ti. L'élément d'indice j du tableau T1 (resp. T2) est constitué d'une double valeur : le contenu C(j) et l'image de j par l'application A (resp. B), c'est-à-dire un indice dans le tableau T2 (resp.T3). L'élément d'indice k du tableau T3 est égal à C(k). Ainsi, les trois tableaux, pour l'exemple d'information ci-dessus, sont les suivants :

Tableau T1

j	C(j)	A(j)
1	d	1
2	f	2
3	d	2
4	d	2

Tableau T2

j	C(j)	B(j)
1	m	1
2	n	3

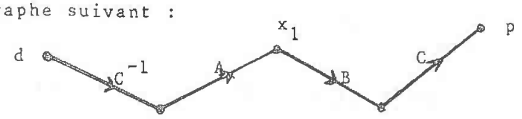
Tableau T3

k	C(k)
1	p
2	q
3	p

Remarquons que cette représentation ne "déforme" pas la sructure S : elle respecte parfaitement les relations de L. En effet, il y a une correspondance biunivoque entre les relations de L et leur représentation.

Demande d'accès :

Faisons par exemple la demande d'accès représentée par le graphe suivant :



Programme résultant :

Si l'on veut chercher "à la main" un programme correspondant à cette demande, de nombreuses solutions sont possibles ; pour choisir entre elles, on peut par exemple raisonner de la façon suivante (en supposant que l'on puisse rechercher indifféremment les relations ou leurs réciproques ; cf § VII-4-3):

- nous pouvons partir de d ou de p (extrêmités) et exécuter la recherche de $C^{-1}(d)$ ou de $C^{-1}(p)$. Nous avons un argument de choix entre les deux : il y aura probablement moins de solutions à $C^{-1}(p)$ qu'à $C^{-1}(d)$, nous dit l'hypothèse (h'). Nous dirons que $C^{-1}(p)$ est plus déterministe que $C^{-1}(d)$. C'est lui que nous choisirons.

- nous avons le choix, pour continuer, entre $C^{-1}(d)$ et $B^{-1}(C^{-1}(p))$. Là, il n'y a pas à hésiter : la deuxième recherche est parfaitement déterministe car B est une injection. Nous choisirons donc $B^{-1}(C^{-1}(p))$ qui donnera à l'exécution un ensemble X'1 de valeurs candidates à être affectées à X1 (nous appelons X1 la variable, dans un sens large, destinée à recevoir les valeurs de l'inconnue x_1).

- nous avons maintenant le choix entre $C^{-1}(d)$ et $A^{-1}(B^{-1}(C^{-1}(p)))$. Tous les deux sont peu déterministes et d'autres considérations doivent entrer en jeu dans le choix. Elles sont évoquées plus loin (§VII-4-4) : par exemple, le fait que l'argument de C^{-1} ait une valeur unique contrairement à celui de A^{-1}. Utilisant ce critère, nous choisirons $C^{-1}(d)$. Ensuite, ce sera naturellement $A(C^{-1}(d))$, qui est déterministe. Nous obtiendrons à l'exécution un ensemble X"1 de valeurs candidates à être affectées à X1. L'ensemble des valeurs qui devront être alors affectées à X1 sera l'intersection des ensembles X'1 et X"1.

Cet exemple donne une idée des espoirs que l'on peut nourrir en cherchant une bonne stratégie et du genre de notion pouvant intervenir dans le choix de celle-ci. Nous allons maintenant présenter brièvement les principales étapes de la génération d'un programme objet pratiquant une "bonne" stratégie. Cette notion de qualité sera précisée par la suite : elle peut faire référence à divers critères : temps, encombrement,...

IV-2-2- DIFFERENTES ETAPES DE LA TRADUCTION D'UN MOTIF EN PROGRAMME INDETERMINISTE. OU INTERVIENT LA REPRESENTATION ?

La responsabilité "de l'ordinateur" n'intervenant qu'au niveau du motif dans la recherche d'une bonne stratégie, c'est à ce niveau que nous nous placerons ici. Nous parlerons d'ailleurs pour l'instant plutôt de pochoir que de motif, n'ayant pas encore défini rigoureusement ce dernier, et assimilerons le pochoir avec le graphe (généralisé) qui en est la représentation imagée.

IV-2-2-1- Cas schématique :

A première vue, on pourrait envisager que la traduction d'un pochoir en un programme de recherche puisse se décomposer en trois étapes, comme suit :

a) décomposition du pochoir en arcs, chaque arc, sommets inclus, devant faire l'objet d'une "recherche élémentaire". Cette recherche sera bien sûr d'une nature un peu différente selon que les sommets seront imposés ou non (par exemple, pour l'arc $7 \xrightarrow{C^{-1}}$ du pochoir $7 \xrightarrow{C^{-1}} \xrightarrow{S} \xrightarrow{C} 4$, le sommet 7 est imposé, par l'autre ; lorsque cet arc a été trouvé et si l'on recherche ensuite

l'arc $\xrightarrow{S}$, le sommet de gauche de ce dernier est imposé).

Chaque recherche élémentaire sera effectuée par un module de programme dont le principal constituant est une prodédure de recherche liée à la relation valuant l'arc considéré. Ces procédures sont l'expression, dans la représentation choisie, de ce que nous avons appelé le "mécanisme" des relations (fin du § II-5-1-). Elles sont écrites par la personne créant le système de données, c'est-à-dire spécifiant la structure et la représentation (cf chapitre VI) : ce sera là pratiquement son seul travail de programmation. Pour une structure de liste par exemple, il faut décrire deux telles procédures : l'une pour S, l'autre pour C.

Mais revenons à la phase a) : son résultat est un ensemble d'arcs à rechercher. Elle est indépendante de la représentation.

b) Choix d'un ordre sur l'ensemble de ces arcs. Le résultat de cette étape est une suite d'arcs à chercher ou plus exactement une suite d'appels de modules tels que ceux décrits à propos de la phase a). Dans cette seconde étape, la représentation peut intervenir dans la détermination de l'ordre. En effet, à égalité de déterminisme, nous verrons qu'on peut faire entrer en ligne de compte un facteur "coût" de la recherche : entre deux recherches élémentaires également sélectives, on préfèrera effectuer d'abord celle dont l'exécution sera la moins coûteuse, en temps par exemple ; or la représentation influe sur ce critère.

c) Constitution d'un programme formé de cette suite d'appels et des modules. Si chaque arc cherché avait toujours une occurrence et une seule, le programme ainsi obtenu aurait un sens et serait exécutable, à condition que la gestion des renseignements à conserver soit bien faite (ne soulevons pas ce problème pour l'instant). Mais il n'en est rien et l'on doit s'intéresser au fait qu'un arc peut avoir de nombreuses occurrences (comment les traiter, alors : toutes ensembles, une par une, ... ?), ou aucune (comment agir dans ce cas ?). On considérera donc le programme constitué de la suite d'appels et des modules comme un <u>programme indéterministe</u> [bibl 40 et 24] en ce sens qu'à certaines de ses étapes il y a le <u>choix</u> entre plusieurs possibilités (ici entre plusieurs occurrences de l'arc cherché) et que ce programme ne porte pas en lui l'aptitude de choisir entre elles. Le problème de l'exécution de ce programme indéterministe sera abordé au paragraphe IV-3 puis étudié au Chapitre VIII.

IV-2-2-2- Cas général - La structure PHYLOG :

Malheureusement cette séparation en trois phrases est peu conforme à la réalité. Il arrive en effet assez souvent (voir § VI-3) que la représentation "déforme" la structure : elle est alors la représentation exacte d'une structure logique [bibl 39] différente de la structure logique donnée. Nous avons pris l'habitude d'appeler cette structure la structure PHYLOG de la structure considérée (c'est-à-dire la structure LOGique de la représentation, autrement dit de la structure PHYsique). Le fait d'accepter cette déformation, outre qu'il est utile en pratique (voir au § III-2-2 l'exemple des dossiers médicaux), donne une grande souplesse à notre système et par exemple permettra, pour une information enregistrée donnée, de la considérer comme ayant tour à tour des structures logiques différentes.

Cette déformation se traduit par le fait suivant ; il n'y a pas correspondance biunivoque (pour une représentation donnée) entre les relations de L et les procédures de recherche ; chacune de ces procédures correspond soit à une relation de L soit à une relation dérivée des relations de L par composition et/ou inversion (nous appellerons une telle relation faisant l'objet d'une procédure un <u>groupement traduisible</u>). Ainsi, dans ce cas, le <u>découpage même du pochoir est lié à la représentation</u>. De plus, on ne peut pas séparer les phases de décomposition et de recherche d'un ordre car, comme on le verra au §VII-4-3, on peut avoir à choisir un ordre entre deux (ou plusieurs) groupements traduisibles qui ont une zone de recouvrement et le fait d'en choisir un comme premier à exécuter amène à réduire l'autre (en lui ôtant au moins la partie commune).

On est donc obligé de fusionner les deux phases a et b en une phase unique que nous appelons : groupement - traduction - choix. Cette phase fait appel à la spécification de la représentation mais nous avons pensé que la spécification complète était surabondante dans cette phase : seule la spécification du passage de la structure logique à la structure PHYLOG est utile. La représentation proprement dite n'intervient que dans la phase c sous la forme des procédures de recherche.

IV-2-3- PRINCIPE DU CHOIX D'UNE STRATEGIE .

Pour exposer ce principe, nous nous placerons dans le cas schématique du paragraphe IV-2-2-1 , où la phase de choix d'un ordre parmi les recherches élémentaires est nettement séparée des autres. Le principe reste le même lorsqu'on passe au cas général.

Dans l'exemple de la recherche du motif $7 \xrightarrow{C^{-1}} \bullet \xrightarrow{S} \bullet \xrightarrow{C} 4$, trois recherches élémentaires seront à effectuer. Si l'on veut déterminer dans quel ordre elles devront être pratiquées, il est naturel d'éliminer tout de suite l'arc $\bullet \xrightarrow{S} \bullet$ comme arc à rechercher en premier. En effet, cette recherche serait beaucoup trop vague, trop "indéterministe" ou même peut-être impossible. Par contre, chacune des deux autres recherches peut être pratiquée la première : elles ont chacune un "point de départ" possible. Nous appellerons <u>extrémités</u> les sommets du pochoir pouvant servir de point de départ à une recherche élémentaire. Nous approfondirons cette notion plus loin (§ VIII-3-2) et en particulier parlerons du cas des relations n - aires ($n \geq 3$). Disons cependant dès maintenant que, pour un motif donné, certaines extrémités sont impératives mais qu'on pourra permettre à l'utilisateur d'en désigner d'autres, ce qui lui donnera un certain pouvoir sur l'exécution de la recherche, même au niveau du motif (cela avait été annoncé au paragraphe IV-1-5).

Un motif étant donné et ses extrémités connues, comment choisit-on une stratégie ? On va procéder par étapes ; à chaque étape, un certain nombre d'arcs (groupements traduisibles, dans le cas général) sont <u>candidats</u> à être traités.

Ce que nous entendons par "traiter" est simple, rappelons que nous nous plaçons ici dans une phase de compilation : traiter un arc est donc donner naissance à l'instruction d'appel du module qui effectuera sa recherche. A la première étape, pouvons-nous dire en première approximation, les arcs candidats sont ceux qui passent par des extrémités du motif.

Parmi les candidats, un arc est choisi pour être traité. Le choix se fait selon des critères que nous citerons au paragraphe VII-4-4 mais dont nous avons déjà donné une idée : déterminisme, nombre de valeurs des arguments, coût de la recherche, ... Nous avons l'intention d'étudier expérimentalement des heuristiques de choix, ce qui sera facilité par la modularité de notre réalisation.

Le choix d'un arc étant effectué, cet arc est remplacé dans l'ensemble des candidats par ses successeurs dans le motif (cette notion sera précisée plus loin mais on peut s'en faire dès maintenant une idée intuitive). Puis on recommence l'étape de choix. Et ainsi de suite.

Pour faciliter la prise en compte des motifs, nous avons trouvé commode de créer à partir de chaque motif deux familles d'arborescences (arborescences "gauche-droite" et arborescences "doite-gauche") qui rendent facile la désignation des successeurs de chaque arc (cf § VII -3-2) ; cette commodité est particulièrement nette dans le cas où la représentation déforme la structure. De plus, ce procédé permet de séparer les actions liées à la syntaxe de PIVOINES de celles liées au choix de la stratégie.

Deux remarques au sujet du processus de choix d'une stratégie :

D'abord, ce processus peut être bien plus compliqué que ne le laissent entendre les lignes précédentes : lorsque la représentation déforme la structure, les candidats ne sont

plus nécessairement des arcs mais peuvent être des groupements traduisibles ; or des choix successifs de groupements traduisibles peuvent dans certains cas aboutir à des impasses, ce qui oblige à revenir sur des choix déjà effectués. Nous en parlerons au paragraphe VII-4.

La seconde remarque est une question. Après une telle succession de choix ponctuels, c'est-à-dire indépendants les uns des autres, sans prise en considération d'un point de vue global, obtient-on en fin de compte un "bon" programme ? Cette question est difficile. Nous ne pourrons jamais être certains d'obtenir le meilleur programme possible. Nous ferons cependant une tentative de réponse partielle en définissant un "bon" programme et, plus tard, en menant une étude expérimentale de comparaison des différents programmes obtenus avec des heuristiques diverses : d'une part comparaison de ces programmes entre eux, d'autre part comparaison des meilleurs avec ce qu'un bon programmeur aurait obtenu à la main.

IV-3- DEROULEMENT DU PROGRAMME OBJET INDETERMINISTE. MODE GLOBAL OU MODE SERIEL ?

Le programme objet de la traduction d'un motif est, nous l'avons dit au paragraphe IV-2-2-1- (c), un programme indéterministe ; il est formé d'une suite d'appels de modules de recherche dont la signification peut être schématisée par "choix" d'une occurrence de l'arc...".

Pour rendre exécutable un tel programme indéterministe, il faut opter pour un "mode d'exécution", puis ou bien placer l'exécution des ordres du programme indéterministe sous le contrôle d'une sorte de moniteur écrit pour le mode choisi, ou bien traduire le programme indéterministe en un programme déterministe (pour le mode choisi). C'est cette dernière solution que nous avons retenue ; le programme traducteur s'appelle "TRADID" (§ VIII-2-).

Les deux modes d'exécution entre lesquels on peut choisir sont les suivants :

- le mode global qui consiste à ne pas effectuer véritablement de choix et à conserver toutes les occurrences qui conviennent ;

- le mode que nous appellerons sériel qui consiste à effectuer un choix à chaque instruction "choix d'une occurrence" et à revenir sur cette instruction pour trouver successivement les autres occurrences, soit à la suite d'un échec (l'occurrence choisie a mené à une impasse), soit à la suite d'un succès (on a trouvé une occurrence de tout le motif et on en cherche une autre).

Pour la plupart des problèmes indéterministes, on peut dire que le mode global nécessite un programme dont la logique est très simple mais la gestion des résultats fort lourde, alors que le mode sériel se réalise par un programme dans lequel il faut prévoir les retours en arrière mais dans lequel la gestion et l'encombrement des résultats intermédiaires sont légers. Dans le cas qui nous occupe, le mode sériel devient extrêmement compliqué. D'abord, le fait que les recherches d'arcs contigus ne se fassent pas nécessairement de manière consécutive rend les retours en arrière très

délicats. Ensuite, l'efficacité du programme risque d'être déplorable pour la raison suivante, dont nous appuyons l'énoncé sur l'exemple du paragraphe IV-2-1 : le premier passage par la recherche de $B^{-1}(C^{-1}(p))$ va donner une valeur v candidate à être affectée à X1 ; mais pour savoir si cette valeur v convient, il faut attendre la recherche de $A(C^{-1}(d))$, recherche qui devra probablement être effectuée plusieurs fois avant de trouver une valeur égale à v ou de prouver qu'il n'y en a aucune ; or les résultats de ces recherches seront perdus (ou alors on revient à un mode au moins partiellement global) et, pour chaque nouvelle solution de $B^{-1}(C^{-1}(p))$, il faudra réitérer les recherches de $A(C^{-1}(d))$.

Pour ces raisons, dont le première est plus particulièrement liée à notre problème, nous avons donné la préférence au mode global. Le programme indéterministe sera donc exécuté dans ce mode et, au moment de la traduction du motif, il n'y aura pas à mettre en place de points de repères (étiquettes) pour des retours en arrière.

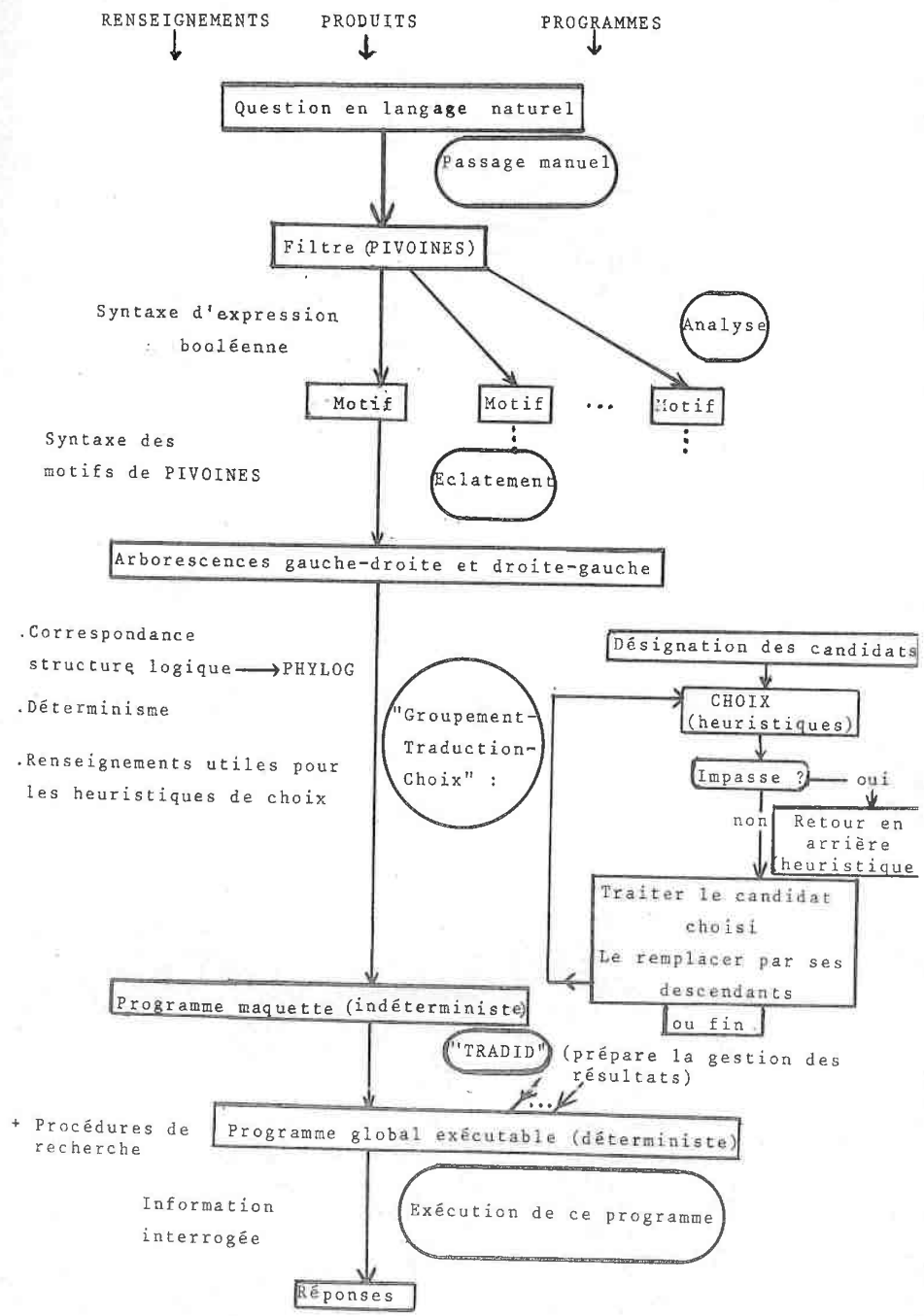
En ce qui concerne le traitement des filtres, dans l'optique du mode global, il y a peu à dire : il se déduit tout naturellement de la nature des filtres ("expressions booléennes de motifs") et du fait qu'ils sont pris en compte dans l'ordre de leur écriture (cf § VII-1-2).

En choisissant de répondre à l'indéterminisme par le mode global, nous esquivons le problème que poserait une véritable résolution de l'indéterminisme : quelle solution choisir à chaque pas ? Pour ne pas donner de faux espoirs au lecteur, nous n'appellerons donc plus le programme indéterministe par ce nom mais l'appellerons "programme maquette"; il est en effet une sorte de maquette de programme, spécifiant ses différentes étapes - sa stratégie -, mais non encore garnie des détails d'obtention et de gestion des résultats.

IV-4- RECAPITULATION : SCHEMA DU DEROULEMENT DE L'INTERROGATION.

Voici la légende du schéma ci-dessous qui représente les différentes étapes de la traduction d'une question, exprimée par un filtre en PIVOINES, en un programme exécutable :

- de haut en bas, nous suivons l'ordre chronologique ;
- le schéma est découpé latéralement en trois zones (liées les unes aux autres) :
  - . au centre sont inscrits les "produits" successifs obtenus,
  - . à droite les programmes mis en oeuvre,
  - . à gauche les renseignements utiles à ces programmes.



V. LE LANGAGE PIVOINES : SYNTAXE ET SEMANTIQUE.

oOOo

Dans ce chapitre, nous allons d'abord définir de façon précise les motifs, au point de vue syntaxique (V.2) puis sémantique (V.3 et 4), montrant leur équivalence avec les pochoirs. Puis nous présenterons les filtres (V.5). Nous concluerons en comparant PIVOINES avec quelques autres langages d'interrogation.

oOOo

V.1. PRESENTATION DU LANGAGE DES MOTIFS.

V.1.0. GENERALITES.

Les motifs expriment des pochoirs, c'est-à-dire des parties d'information dont on désire trouver des occurrences dans l'information interrogée I. Ces parties consistent en des informations élémentaires et des inconnues (cf. § IV.1.4), liées soit par des relations d'accès élémentaires soit par des combinaisons de celles-ci obtenues par l'application d'opérations définies au paragraphe II.4.2.

Nous avons choisi d'exprimer chaque pochoir de la façon la plus naturelle, la plus "photographique", la plus simple, possible. Par exemple, les pochoirs

sont exprimés respectivement par les motifs : a R x_1 et a R x_1 S x_2. D'autres exemples seront donnés au paragraphe V.1.2. Une particularité du langage des motifs est l'emploi d'inconnues et le fait qu'on exprime des arcs, ce qui n'est pas toujours le cas (cf. remarque 1 page suivante).

Un motif M est en fait la description d'un accès, (cf. § II.4.1), c'est-à-dire la spécification d'une partie de E^n (ou d'une relation de E^o vers E^n), pour un certain n.

Chaque occurrence, dans l'information I interrogée, de la partie de E^n spécifiée par ce motif sera "matérialisée" par un n-uple de valeurs pour les n inconnues de M (cf. § V.3.7.3).

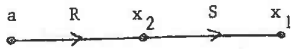
Remarque 1 :

Nous avions primitivement essayé d'exprimer les demandes d'accès (relations de E^o vers E^n) sous forme de "schémas relationnels" généralisant les schémas fonctionnels. Par exemple, les pochoirs ci-dessus étaient exprimés (en écrivant les schémas de gauche à droite) par : a R et a R <I,S> , où <I,S> est la juxtaposition de la fonction identité et de la relation S. Mais cette écriture purement relationnelle devient très rapidement lourde et est même quelquefois impossible, lorsque l'on ne peut pas linéariser le pochoir sans répéter certains sommets.

Remarque 2 :

En plus du souci de simplicité qui nous a guidée vers une solution où toute demande d'accès est presque une photographie du fragment d'information à chercher, un désir de stabilité a également joué dans le choix de l'écriture des motifs dans PIVOINES : deux demandes d'accès relatives à deux fragments peu différents sont peu différentes ; en particulier, les demandes relatives à deux fragments ne différant que par leurs inconnues (par exemple

et



)

diffèrent très peu, ce qui ne serait pas le cas pour une écriture purement relationnelle.

Examinons maintenant de façon précise quels sont les matériaux de construction des motifs.

V.1.1. MATERIAUX DE CONSTRUCTION DES MOTIFS.

Ce sont :

a) les symboles relationnels de la structure considérée (ensemble $\wedge$) ; nous les confondrons, dans la rédaction, avec les relations elles-mêmes (ensemble L) car cela n'apporte aucune ambiguïté ;
les éléments directement accessibles de E ; mais cet ensemble est inclus dans le précédent, c'est $\wedge_1$ (cf. § II.3.1 et II.5.1) ;

b) les symboles x_o, x_1, ... , x_i, ... qui figurent les inconnues ; désignons par X leur ensemble ;

c) le symbole US qui signifie "unité de sélection" et que nous préciserons plus loin (§ V.3.7.2) ; il s'agit d'un "gadget", utile mais non essentiel ;

d) les signes opératoires $\neg$, $^{-1}$, * (la composition est représentée par une simple concaténation), les crochets (< et >) et la virgule.

Ensemble VAL : nous appelons VAL la réunion :

$$VAL = \wedge_1 \cup X \cup \{US\}.$$

Ensemble RELA des relations atomiques :

Nous avons limité l'usage des opérations de réciproque, itération, négation au cas des relations "atomiques" (et même, parmi elles, aux relations binaires pour les deux premières opérations).

Nous appelons relation atomique :

- soit une relation élémentaire (élément de L) d'arité ≥ 2 ;
- soit la réciproque d'une relation de L_2 (R^{-1}) ;
- soit l'itération d'une relation de L_2 ou de sa réciproque (on parlera de $(R^{-1})^*$ mais pas de $(R^*)^{-1}$) ;
- soit la négation d'une relation d'un des types précédents.

L'ensemble de toutes les relations atomiques sera appelé RELA. La forme des éléments de RELA étant définie ci-dessus, nous n'y reviendrons plus et considérerons ces éléments comme des terminaux de la grammaire G des motifs.

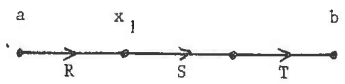
En résumé de ce paragraphe, les matériaux de construction des motifs, c'est-à-dire les terminaux de la grammaire G qui les définit sont :

- les éléments de VAL
- les éléments de RELA
- les crochets et la virgule.

V.1.2. PRESENTATION INTUITIVE, PAR DES EXEMPLES, DE L'ECRITURE DES MOTIFS.

Un motif est une écriture (linéaire) de ce que nous avons appelé un pochoir. Dans les chapitres V.3 et V.4 nous parlerons avec précision de la sémantique des motifs et de l'équivalence des motifs et des pochoirs. Mais pour l'instant nous désirons simplement faire comprendre intuitivement ce qu'est un motif.

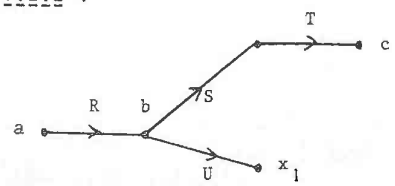
Exemple V.1.2.1 :



Le motif peut s'écrire :

a R x_1 S T b.

Exemple V.1.2.2 :

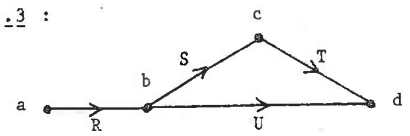


Le motif peut s'écrire : a R b < S T c , U x_1>.

Nous avons défini au paragraphe II.4.2.2 la juxtaposition des relations : < S T , U > en est une. Dans le langage des motifs, nous avons étendu cette notion au cas de morceaux de motifs (ici des "Fins" de motifs, c'est-à-dire

des morceaux commençant par un élément de RELA et finissant par un élément de VAL) ; la notation a également été adaptée à ce cas.

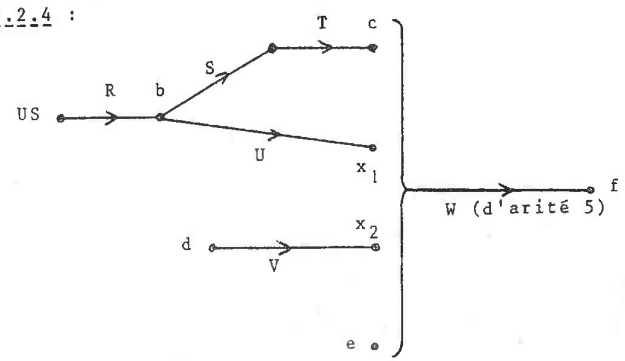
Exemple V.1.2.3 :



Le motif peut s'écrire : a R b < S c T , U > d.

Même remarque que ci-dessus mais < S c T , U > est une juxtaposition de "centres" de motifs.

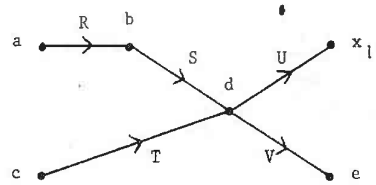
Exemple V.1.2.4 :



Le motif peut s'écrire :

< US R b < S T c , U x_1 > , d V x_2 , e > W f.

Exemple V.1.2.5 :



Le motif peut s'écrire :

< a R b S , c T > d < U x_1 , V e > .

Le morceau < a R b S , c T > est une juxtaposition de "débuts" de motif.

Exemple V.1.2.6 . Remarque sur l'utilisation de x_o :

Soit le même "pochoir" que pour l'exemple précédent mais sans valeur imposée ni inconnue au carrefour d. La syntaxe des motifs obligera à nommer cependant ce point pour exprimer la convergence des arcs étiquetés S et T. C'est à cet usage que servira x_o (cf. § V.3.7.3). En quelque sorte, x_o jouera le rôle d'une inconnue ordinaire mais sa valeur n'aura pas besoin d'être retenue dans le résultat de la recherche. En fait, on ne commettrait aucune erreur, en remplaçant x_o par n'importe quelle inconnue non utilisée ailleurs dans le filtre.

Le motif pourra alors s'écrire :

< a R b S , c T > x_o < U x_1 , V e > .

Exemple V.1.2.7 :

Les motifs correspondant aux "pochoirs" de l'exemple IV.1.4.1 peuvent s'écrire respectivement :

$x_o < C\ 7\ ,\ S\ C\ 4 >$

$x_o < C\ 7\ ,\ S\ C\ 4\ ,\ S\ S\ C\ x_1 >$ ou

$x_o < C\ 7\ ,\ S\ x_2 < C\ 4\ ,\ S\ C\ x_1 > >$.

Mais on peut modifier légèrement les pochoirs en inversant le sens de leur arc le plus "à gauche" et les motifs pourront alors s'écrire respectivement :

$7\ C^{-1}\ S\ C\ 4$

$7\ C^{-1}\ S\ x_o < C\ 4\ ,\ S\ C\ x_1 >$.

V.1.3. TROIS REMARQUES SUR LA CONSTRUCTION DES MOTIFS.

- Format :

Ce que nous allons dire ici n'a aucun but de formalisation, mais seulement un but d'aide à la conceptualisation des motifs.

Nous appelons format d'une partie de motif le couple formé de la nature (VAL ou RELA) de son premier terminal (différent d'un caractère spécial), et de la nature (VAL ou RELA) de son dernier terminal (également différent d'un caractère spécial).

Tous les composants d'un même crochet doivent avoir le même format qui devient le format du crochet. Un motif a comme format VAL VAL.

Dans l'exemple V.1.2.2, le format du crochet est RELA VAL. Nous verrons qu'il s'agit d'une sous-phrase dont la catégorie grammaticale (pour notre grammaire G introduite au paragraphe suivant) est Fin. Dans l'exemple V.1.2.3, le format du crochet est RELA RELA. Sa catégorie grammaticale est Centre. Dans l'exemple V.1.2.4, le format du crochet intérieur est RELA VAL (catégorie : Fin), celui du crochet extérieur est VAL VAL (catégorie : Motif ou Motint). Dans l'exemple V.1.2.5, le format du premier crochet est VAL RELA (catégorie grammaticale : Début) et celui du deuxième est RELA VAL (Fin).

- Relations d'arité supérieure à 2 :

Dans un motif, un nom de relation R d'arité p+1 supérieure à 2 est nécessairement précédé d'un crochet (de format VAL VAL ou RELA VAL) tel que les terminaisons de ses constituants forment le p-uple des arguments de R. Ainsi, dans l'exemple V.1.2.4, ce p-uple pour W est : (c, x_1, x_2, e).

Cette obligation de concordance entre l'arité d'une relation -arité donnée avec la structure- et le nombre de ses arguments -dans les motifs- constitue une "restriction sémantique" à la syntaxe.

- Non unicité des motifs pour un pochoir donné :

Nous pouvons affirmer dès maintenant que l'écriture d'un motif pour un pochoir donné n'est pas unique ; nous préciserons cela au paragraphe V.4. Le choix entre plusieurs possibilités ne revêt pas une très grande importance car le programme d'exploitation obtenu sera approximativement le même dans tous les cas. Ce sont donc des considérations de commodité d'écriture et de rapidité de "compilation" qui seront les principaux critères de choix. La première de ces considérations sera particulièrement sensible dans le cas d'une utilisation de PIVOINES sur console dans un régime conversationnel.

V.2. GRAMMAIRE G DES MOTIFS.

L'ensemble des motifs possibles pour l'interrogation d'une information de structure donnée S est un langage algébrique (ou "context-free") dont les règles de la grammaire G sont fixées mais dont l'ensemble T des terminaux dépend de la structure S. Cet ensemble T, comme nous l'avons vu au paragraphe V.1.1, est : $T = VAL \cup RELA \cup \{ < ; > ; , \}$.

Notation :

Nous employons pour présenter G la notation de Backus en l'allégeant :

- nous notons par : la relation de production ;
- nous n'entourons pas les symboles terminaux ; au contraire, les crochets figurant dans G sont des terminaux ;
- nous numérotons les règles pour pouvoir y faire référence et faisons figurer après chacune d'elles son numéro entre des barres ;
- enfin, pour diminuer le nombre de règles et simplifier l'écriture, nous utilisons la notation suivante : a étant un non-terminal, $a^{\times}$ désigne toute suite finie (de longueur ≥ 1) de a séparés par des virgules. Par exemple, si a est "fin", $fin^{\times}$ désignera : fin, fin, ... , fin.

120

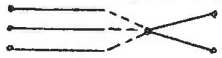
<u>Grammaire G</u> : (avec illustration pour aider à la conceptualisation) :

Motif : Motint[1]/0/

Motint : Motif GG[2]/1/Motifcro/4/Motifcro Fin/2/

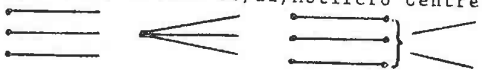
• ou

Débutcro Motif CG/3/



Motifcro : <Motint$^{\times}$>/5/

Début : Débutcro/6/Début CG/22/Motifcro Centre/7/



Débutcro Début CG/8/

---

(1) L'analyse d'un motif s'effectuant au sein d'un filtre, on saura qu'un motif est terminé lorsqu'on lira un symbole n'appartenant pas à T.

(2) Signification mnémonique des suffixes :

CG = "convergent à gauche" ; cro = "constitué d'un crochet"
elem = "élémentaire" ; compo = "composé"
int = "interne"

Signification des préfixes :

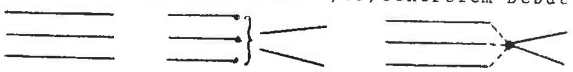
Moti : "de format VAL VAL" ; Début : "de format VAL RELA"
Fin : "de format RELA VAL" ; Centre : "de format RELA RELA".

Débutcro : <Début$^{\times}$>/9/

Fin : Fincro/10/Fincro Fin/11/Centrelem Motif CG/12/

Fincro : <Fin$^{\times}$>/13/

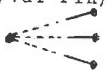
Centre : Centrelem/14/Fincro Centre/15/Centrelem Début CG/16/



Centrelem : Relacompo/17/<Centre$^{\times}$>/18/

Relacompo : Rela/23/Relacompo Rela/24/

Motif CG : Val/19/Val Fin/20/



Début CG : Val Centre/21/

Val : Eléments de Val

Rela : Eléments de RELA

Nous étudierons des propriétés de cette grammaire au paragraphe V.3.5.

V.3. SEMANTIQUE DU LANGAGE DES MOTIFS.

V.3.1. FORMALISATION DE LA NOTION DE POCHOIR.

1°) Arc :

Nous définissons un arc comme un triplet (o, R, b) dans lequel :

- R s'appelle le "nom de relation". C'est un nom de relation atomique (élément de RELA) d'arité donnée p+1 (cf. § V.1.1).

- b s'appelle le "but" de l'arc. C'est soit un élément de VAL (cf. § V.1.1), soit une "inconnue locale" : lorsque, dans une partie d'information à chercher, des informations élémentaires sont liées par une relation non atomique formée par la composition de deux relations, le point intermédiaire "non nommé" est appelé une inconnue locale et affecté d'un "nom d'inconnue locale", inaccessible à l'utilisateur (cf. tableau des règles de définition de la signification, règle n° 23, § V.3.4).

- o s'appelle l' "origine" de l'arc. C'est un p-uple (R étant d'arité p+1) d'éléments tels que b. Nous appelons "point origine" chaque élément de l'origine (si p = 1, origine et point origine sont synonymes).

Buts et points origines sont appelés "sommets" de l'arc. Dans la suite, nous noterons la plupart du temps

les noms de relation par des lettres majuscules, les sommets par des lettres minuscules, et les p-uples seront placés entre crochets < >.

2°) Pochoir :

Un pochoir est défini comme un ensemble P d'arcs et de points isolés (éléments de $\wedge_1$, c'est-à-dire éléments directement accessibles de E).

On appelle "points d'entrée" de P les points isolés et les points origines qui ne sont but d'aucun arc de P ; il se peut que P n'ait aucun point d'entrée. On définit de façon analogue les "points de sortie" de P.

La définition de la sémantique du langage des motifs va consister à associer à chaque motif une "signification" sous la forme d'un pochoir. Au paragraphe V.5, on étudiera à l'inverse le passage d'un pochoir aux motifs qui l'expriment.

V.3.2. PRESENTATION DE LA SIGNIFICATION D'UN MOTIF ET DU MODE DE DEFINITION DE CELLE-CI.

Nous venons de dire que la signification d'un motif M est un pochoir P. Examinons quelques exemples :

Motif	Signification
aRb<Sc,Td>	{(a,R,b),(b,S,c),(b,T,d)}
<a,bRc>Sd	{(<a,c>,S,d),(b,R,c)}
<a<R,SbT>c,e>	{(a,R,c),(a,S,b),(b,T,c),e} e, contrairement à c, est un point isolé.

Le langage des motifs est défini par la grammaire G que nous avons donnée au paragraphe V.2 et c'est à cette grammaire que nous allons rattacher la définition de la signification d'un motif, c'est-à-dire de la sémantique du langage des motifs. Soit une partie de motif appartenant à une catégorie grammaticale A de G. Nous lui associons, par récurrence, une signification (par abus de langage, nous appellerons cette signification : "signification du non-terminal A").Pour chaque règle A : A_1 A_2 ... A_n de G, nous allons donner une règle de définition ou de construction de la signification de A en fonction de celles de A_1, A_2, ..., A_n.

Cette définition, non seulement permet de bien comprendre la sémantique du langage, mais encore donnera de façon immédiate un algorithme de "décryptage" des motifs, générant leur signification. En effet, la signification du motif peut être construite automatiquement à partir du résultat de son analyse syntaxique et même au fur et à mesure de cette analyse. Il serait tout à fait possible de réaliser cet algorithme grâce au système FACE (Fabrication Automatique de Compilateur Efficace) [4 et 37]. Il suffirait de fournir à celui-ci d'une part la grammaire G et d'autre part, pour chaque règle de G, la règle de construction de la signification, sous la forme d'une procédure. Remarquons que ces procédures devraient assurer la gestion d'un espace de résultats intermédiaires de longueur variable (cf. § V.3.5).

Remarque :

On peut dire que nous utilisons ici la notion d'attributs synthétisés [bibl 34] qui consiste à associer à chaque symbole d'une grammaire un ou des attributs (par exemple valeur, longueur, ...), et à associer à chaque règle de la grammaire une ou des formules permettant de calculer les attributs du non-terminal qui en est la partie gauche en fonction des attributs des symboles figurant dans sa partie droite (ainsi les attributs de chaque non-terminal ne dépendent que des descendants du non-terminal et non de ses ancêtres comme dans le cas des "attributs hérités" ; ils résultent d'une "synthèse").

V.3.3. QUELQUES DEFINITIONS RELATIVES AUX SIGNIFICATIONS.

Nous posons la définition suivante :

la signification d'une partie de motif appartenant à la catégorie syntaxique A, que par abus de langage nous appelons signification de A, est un triplet :

S(A) = (d , E , f) dans lequel :

- E est un ensemble d'arcs dont certains peuvent être incomplets, c'est-à-dire avoir des caractéristiques non encore déterminées, notées alors - ;
  exemple : (- , R , b) ;

- d et f ont pour objet d'être des interfaces entre la signification de A et la signification d'autres non-terminaux figurant dans la partie droite d'une même règle de G, pour définir la signification du non-terminal qui est la partie gauche de cette règle. Chacun d'eux peut être inexistant : on dira alors qu'il a la valeur ω.

  S'il existe :
  - d est un but d'arc
  - f est une origine d'arc.

Dans la suite du chapitre, nous ferons appel aux applications d, E, f, dE, Ef, qui à chaque non-terminal feront correspondre, dans un contexte donné, respectivement le 1er, le 2ème, le 3ème, les 1er et 2ème, les 2ème et 3ème termes de sa signification.

Notation de S(A) = (d(A) , E(A) , f(A))

S(A) sera noté : d(A) s'il existe, puis la suite (éventuellement vide) des triplets constituant E(A), puis f(A) s'il existe. Tous les constituants seront séparés par des virgules.

Exemples :

1) $d(A_1) = b$; $E(A_1) = \{(b,R,c),(c,S,d)\}$; $f(A_1) = \langle c,d\rangle$
   $S(A_1)$ sera noté : b,(b,R,c),(c,S,d),<c,d>

2) $d(A_2) = \omega$; $E(A_2) = \{(-,R,c),(c,S,d)\}$; $f(A_2) = d$
   $S(A_2)$ sera noté : (-,R,c),(c,S,d),d .

Cette notation permettra d'écrire de façon simple les règles de définition de la sémantique, en utilisant la simple concaténation (exemple : E(Motifcro),$S_{f(Motifcro)}$(Fin)) et les applications suivantes :

- $\boxed{S_o(A)}$: pour une origine (simple ou multiple) o donnée, $S_o(A)$ désigne la signification de A dans laquelle les origines non encore déterminées sont remplacées par o.

- $\boxed{S^b(A)}$: pour un but b donné, $S^b(A)$ désigne la signification de A dans laquelle les buts non encore déterminés sont remplacés par b. On utilisera aussi $E^b(A)$ avec une signification analogue.

V.3.4. REGLES DE DEFINITION DE LA SIGNIFICATION S.

Règles de la grammaire		Définition (ou construction) de la signification
N°	Texte	
1	Motint : MotifCG	S(Motint) = Ef(MotifCG) cf. remarque après ce tableau
2	Motint : Motifcro Fin	S(Motint) = E(Motifcro), $S_{f(Motifcro)}$(Fin) Exemple : $\underbrace{<aRb,c>}_{Motifcro}$ $\underbrace{Sd}_{Fin}$ S(Motifcro) = (a,R,b),<b,c> S(Fin) = (-,S,d),d S(Motint) = (a,R,b),(<b,c>,S,d),d
3	Motint : Débutcro MotifCG	S(Motint) = $S^{d(MotifCG)}$(Débutcro), Ef(MotifCG) Exemple : $\underbrace{<aR,cS>}_{Débutcro}$ $\underbrace{dTe}_{MotifCG}$ S(Débutcro) = (a,R,-),(c,S,-) S(MotifCG) = d,(d,T,e),e S(Motint) = (a,R,d),(c,S,d), (d,T,e),e

4	Motint : Motifcro	S(Motint) = S(Motifcro)
5(1)	Motifcro : <Motint$^{\times}$> c'est-à-dire : <Motint1,Motint2,...>	S(Motifcro) = E(Motint1),E(Motint2), ...,<f(Motint1), f(Motint2),...> Remarque : Si ces f(Motint) comportent plusieurs valeurs, on enlève les crochets qui les entourent. Exemple : $<\underbrace{a}_{Motint1}, \underbrace{c\ R\ d}_{Motint2}, \underbrace{e < S\ f\ ,\ T\ g >}_{Motint3}>$ S(Motint1) = a S(Motint2) = (c,R,d),d S(Motint3) = (e,S,f),(e,T,g),<f,g> S(Motifcro)= (c,R,d),(e,S,f), (e,T,g),<a,d,f,g>
6	Début : Débutcro	S(Début) = S(Débutcro)

(1) Les règles n° 5, 9, 13, 18 devraient en réalité être décomposées chacune en trois règles dont une récursive.
Exemple pour la règle n° 5 :
Motifcro : Début-de-Motifcro >
et
Début-de-Motifcro : < Motint / Début-de-Motifcro,Motint.

Pour simplifier l'exposé, nous gardons la forme condensée.

7	<u>Début</u> : Motifcro Centre	S(<u>Début</u>) = E(Motifcro), $S_{f(Motifcro)}$(Centre) <u>Exemple</u> : $\underbrace{<aRb,c>}_{Motifcro}$ $\underbrace{<T,S>}_{Centre}$ S(Motifcro) = (a,R,b),<b,c> S(Centre) = (-,T,-),(-,S,-) S(<u>Début</u>) = (a,R,b),(<b,c>,T,-), (<b,c>,S,-)
8	<u>Début</u> : Débutcro DébutCG	S(<u>Début</u>) = $S^{d(DébutCG)}$(Débutcro), E(DébutCG) <u>Exemple</u> : $\underbrace{<aR,cS>}_{Débutcro}$ $\underbrace{d<T,U>}_{DébutCG}$ S(Débutcro) = (a,R,-),(c,S,-) S(DébutCG) = d,(d,T,-),(d,U,-) S(<u>Début</u>) = (a,R,d),(c,S,d),(d,T,-), (d,U,-)
22	<u>Début</u> : DébutCG	S(<u>Début</u>) = E(DébutCG)
9[1]	<u>Débutcro</u> : <Début^{x}> c'est-à-dire : <Début1,Début2,...>	S(<u>Débutcro</u>) = S(Début1),S(Début2), ...

(1) Voir renvoi (1) page précédente

10	<u>Fin</u> : Fincro	S(<u>Fin</u>) = S(Fincro)
11	<u>Fin</u> : Fincro Fin	S(<u>Fin</u>) = E(Fincro), $S_{f(Fincro)}$(Fin) <u>Exemple</u> : $\underbrace{<UeVf,Wg>}_{Fincro}$ $\underbrace{Sd}_{Fin}$ S(Fincro) = (-,U,e),(e,V,f), (-,W,g),<f,g> S(Fin) = (-,S,d),d S(<u>Fin</u>) = (-,U,e),(e,V,f),(-,W,g), (<f,g>,S,d),d
12	<u>Fin</u> : Centrelem MotifCG	S(<u>Fin</u>) = $S^{d(MotifCG)}$(Centrelem), Ef(MotifCG) <u>Exemple</u> : $\underbrace{<RaS,T>}_{Centrelem}$ $\underbrace{dUe}_{MotifCG}$ S(Centrelem) = (-,R,a),(a,S,-), (-,T,-) S(MotifCG) = d,(d,U,e),e S(<u>Fin</u>) = (-,R,a),(a,S,d),(-,T,d), (d,U,e),e

13(1)	Fincro : <Fin$^{\times}$> c'est-à-dire : <Fin1,Fin2,...>	S(Fincro) = E(Fin1), E(Fin2), ...,<f(Fin1),f(Fin2), ...> Exemple : $\underbrace{\langle UeVf}_{Fin1}, \underbrace{Wg\rangle}_{Fin2}$ S(Fin1) = (-,U,e),(e,V,f),f S(Fin2) = (-,W,g),g S(Fincro)=(-,U,e),(e,V,f),(-,W,g), <f,g>
14	Centre : Centrelem	S(Centre) = S(Centrelem)
15	Centre : Fincro Centre	S(Centre) = E(Fincro), $S_{f(Fincro)}$(Centre) Exemple : $\underbrace{\langle UeVf,Wg\rangle}_{Fincro}\ \underbrace{\langle R,S\rangle}_{Centre}$ S(Fincro) = (-,U,e),(e,V,f), (-,W,g),<f,g> S(Centre) = (-,R,-),(-,S,-) S(Centre) = (-,U,e),(e,V,f),(-,W,g), (<f,g>,R,-),(<f,g>,S,-)

(1) Voir renvoi (1) , trois pages plus haut

16	Centre : Centrelem DébutCG	S(Centre) = $S^{d(DébutCG)}$(Centrelem), E(DébutCG) (On pourrait écrire Ef(DébutCG) mais f(DébutCG) = ω). Exemple : $\underbrace{\langle RaS,V\rangle}_{Centrelem}\ \underbrace{d\langle T,U\rangle}_{DébutCG}$ S(Centrelem) = (-,R,a),(a,S,-), (-,V,-) S(DébutCG) = d,(d,T,-),(d,U,-) S(Centre) = (-,R,a),(a,S,d), (-,V,d),(d,T,-), (d,U,-)
17	Centrelem : Relacompo	S(Centrelem) = S(Relacompo)
18(1)	Centrelem : <Centre$^{\times}$> c'est-à-dire : <Centre1,Centre2,...>	S(Centrelem) = S(Centre1), S(Centre2),...
19	MotifCG : Val	S(MotifCG) = S(Val),S(Val) = Val,Val

(1) Voir le renvoi (1) , quatre pages plus haut

20	<u>MotifCG</u> : Val Fin	$S(\underline{MotifCG}) = S(Val), S_{d(Val)}(Fin)$ $= Val, S_{Val}(Fin)$ <u>Exemple</u> : $\underbrace{d}_{Val}\ \underbrace{Ue}_{Fin}$ $S(Fin) = (-,U,e),e$ $S(\underline{MotifCG}) = d,(d,U,e),e$
21	<u>DébutCG</u> : Val Centre	$S(\underline{DébutCG}) = S(Val), S_{d(Val)}(Centre)$ $= Val, S_{Val}(Centre)$ <u>Exemple</u> : $\underbrace{d}_{Val}\ \underbrace{<T,U>}_{Centre}$ $S(Centre) = (-,T,-),(-,U,-)$ $S(DébutCG) = d,(d,T,-),(d,U,-)$
23	<u>Relacompo</u> : Rela	$S(\underline{Relacompo}) = S(Rela) = (-,Rela,-)$

24	<u>Relacompo</u> : Relacompo Rela c'est-à-dire : Relacompo! Rela	On crée une "<u>inconnue locale</u>" y_j différente de toutes celles déjà créées ; elle désignera le but du dernier arc correspondant à Relacompo! et l'origine de celui correspondant à Rela. $S(\underline{Relacompo})$ $= S^{y_j}(Relacompo!), S_{y_j}(Rela)$ $= S^{y_j}(Relacompo!), (y_j, Rela, -)$ <u>Exemple</u> : $\underbrace{R\ \ U}_{Relacompo!}\ \ \underbrace{V}_{Rela}$ $S(Relacompo!) = (-,R,y_k),(y_k,U,-)$ $S(Rela) = (-,V,-)$ $S(\underline{Relacompo}) = (-,R,y_k),(y_k,U,y_j),$ $(y_j,V,-)$
	<u>Val</u> <u>Rela</u>	$S(\underline{Val}) = Val$ $S(\underline{Rela}) = (-,Rela,-)$
0	<u>Motif</u> : Motint	$S(\underline{Motif}) = E(Motint)f'(Motint)$ avec $f'(Motint) = f(Motint) - \mathcal{S}$ où $\mathcal{S}$ est l'ensemble des sommets d'arcs de E(Motint). f'(Motint) est donc la liste des points isolés de Motint. Cf. le 3ème exemple du début du paragraphe V.3.2.

Remarque : nous pouvons remarquer que :

- les E(Moti...) ne comportent jamais d'arcs incomplets ;
  les f(Moti...) sont ≠ ω sauf éventuellement f(Motif) ;
  les d(Moti...) sont = ω sauf d(MotifCG).

- les E(Début...) comportent toujours des arcs dont le but n'est pas déterminé ;
  les f(Debut..)sont = ω ;
  les d(Debut..)sont = ω sauf d(DébutCG).

- les E(Fin...) comportent toujours des arcs dont l'origine n'est pas déterminée ;
  les f(Fin...) sont ≠ ω ;
  les d(Fin...) sont = ω.

- les E(Centre...) comportent toujours des arcs dont l'origine n'est pas déterminée et des arcs dont le but n'est pas déterminé ;
  les f(Centre...) et d(Centre...) sont = ω.

V.3.5. ALGORITHME DE "DECRYPTAGE" ET PROPRIETES DE G.

Comme nous l'avons dit au paragraphe V.3.2, on peut constituer un algorithme de décryptage de motifs, utilisant un analyseur syntaxique. Si cet analyseur est supposé sortir la représentation postfixée de la ramification associée à la phrase analysée, il suffit [bibl. 40 chapitre VII] de remplacer chaque sortie par l'exécution de la règle correspondante de construction de la signification. Cette règle utilise des significations

précédemment construites et qui sont stockées sur une pile de résultats intermédiaires. Le seul problème de programmation que pose la réalisation de cet algorithme est d'ailleurs la forme de cette pile de résultats. On peut, au fur et à mesure de l'élaboration de la signification, éliminer de la pile et stocker par ailleurs les arcs complets trouvés car tout arc figurant dans la signification d'un sous-filtre figure dans celle du filtre. Mais cela n'empêche qu'un résultat partiel peut avoir une longueur arbitraire. La solution sera soit de prévoir une pile dont les éléments seraient de longueur variable, soit de mettre, dans la pile, des pointeurs vers les significations partielles. C'est d'ailleurs ce que l'on ferait si l'on utilisait FACE (cf. § V.3.2), les procédures fournies devant gérer les résultats intermédiaires.

En ce qui concerne l'analyseur, nous avons écarté l'idée d'un analyseur descendant car la grammaire n'est LL(k) [33 et 50] pour aucun k. Prenons le cas d'un motif commençant par <. Ce motif sera un Motint qui sera soit de la forme Motifcro[Fin] (1), soit de la forme Débutcro MotifCG. Or entre Motifcro et Débutcro, le choix peut nécessiter la connaissance d'une partie arbitrairement longue de phrase. Exemple :

<aRbSc.....U,.....> sera un Débutcro

alors que

<aRbSc.....Ud,.....> sera un Motifcro.

L'analyseur sera donc un analyseur ascendant. Voyons si la grammaire G s'y prête bien. Pour cela, parmi plusieurs méthodes voisines possibles ([bibl. 40 § 8.3.4.4 et 8.3.6]; [bibl.24] ; [bibl.46] ; [bibl.56]) utilisons celle décrite dans [56] et tout d'abord dressons le tableau des initiales strictes ($\mathcal{L}$, c'est-à-dire Leftmost symbols) et des finales

(1) [Fin] signifie : "Fin ou Rien du tout".

strictes ($\mathcal{R}$, c'est-à-dire Rightmost symbols) pour chaque symbole non terminal de G.

Dans tout ce paragraphe et, par la suite, lorsque cela sera commode, nous traiterons les symboles Val et Rela eux-mêmes comme des terminaux, ce qui évitera de détailler les éléments des ensembles Val et Rela.

U	$\mathcal{L}$(U)	$\mathcal{R}$(U)
Motif	Motint - Motifcro - Débutcro - MotifCG - Val - <	
Motint	Motifcro - Débutcro - MotifCG - Val - <	Motifcro - Fin - Fincro - MotifCG - Val - >
Motifcro	<	>
Début	Motifcro - Débutcro - DébutCG - Val - <	Débutcro - Centre - Centrelem - Relacompo - Début CG - Rela - >
Débutcro	<	>
Fin	Fincro - Centrelem - Relacompo - Rela - <	Fin - Fincro - MotifCG - Val - >
Fincro	<	>

Centre	Fincro - Centrelem - Relacompo - Rela - <	Centre - Centrelem - Relacompo - Début CG - Rela - >
Centrelem	Relacompo - Rela - <	Relacompo - Rela - >
Relacompo	Relacompo - Rela	Rela
MotifCG	Val	Fin - Fincro - MotifCG - Val - >
DébutCG	Val	Centre - Centrelem - Relacompo - DébutCG - Rela -

Ce tableau nous sert à construire le tableau des trois relations de précédence :

$\doteq$ lie deux symboles adjacents dans une chaîne directement réductible ;

$\lessdot$ lie un symbole qui précède immédiatement une chaîne directement réductible et une initiale stricte de cette chaîne ;

$\gtrdot$ lie une finale stricte d'une chaîne directement réductible et un symbole suivant immédiatement cette chaîne.

	Motif	Motint	Motifcro	Début	Débutcro	Fin	Fincro	Centre	Centrelem	Relacompo	MotifCG	DébutCG	Val	Rela	<	>	,
Motif																	
Motint																≐	≐
Motifcro						≐	<·	≐	<·	<·				<·	<·	·>	·>
Début																≐	≐
Débutcro											≐	≐	<·			·>	·>
Fin																≐ ·>	≐ ·>
Fincro						≐	<·	≐	<·	<·				<·	<·	·>	·>
Centre																≐ ·>	≐ ·>
Centrelem											≐	≐	<·			·>	·>
Relacompo											·>	·>	·>	≐		·>	·>
MotifCG																·>	·>
DébutCG																·>	·>
Val						≐	<·	≐	<·	<·				<·	<·	·>	·>
Rela											·>	·>	·>	·>		·>	·>
<		≐	<·	≐	<·	≐	<·	≐	<·	<·	<·	<·	<·	<·	<·		
>						·>	·>	·>	·>	·>	·>	·>	·>	·>	·>	·>	·>
,		≐	<·	≐	<·	≐	<·	≐	<·	<·	<·	<·	<·	<·	<·		

Nous voyons qu'il s'en faut d'un très petit détail que la grammaire G soit de simple précédence, c'est-à-dire que l'on puisse construire pour elle un analyseur ascendant tel que celui décrit dans [bibl. 56], déterministe avec la lecture d'un caractère à l'avance : en effet, pour presque tous les couples de symboles, il existe au plus une relation de précédence entre ses membres. Cette propriété est en défaut pour les seuls couples formés de Fin ou Centre et de > ou , ; ceci est dû aux règles de la forme b : <a^{x}>. Chacune de ces règles a été considérée comme une infinité de règles respectivement de la forme b : <a> , b : <a,a> , b : <a,a,a> , ... Pour savoir ce qui doit être fait lors de la comparaison entre un Fin ou Centre situé sur la pile et un crochet > ou une virgule lu à l'avance, il faut connaître l'élément situé sous le sommet de la pile. Nous pouvons donc qualifier la grammaire G de grammaire de précédence d'ordre (2,1) [bibl.56].

En réalité, pour les règles de la forme b : <a^{x}> , ce que nous ferions pour élaborer un algorithme de "décryptage", ce serait de remplacer cette forme multiple par l'ensemble des trois règles :

b : début-de-b >

début-de-b : < a / début-de-b , a .

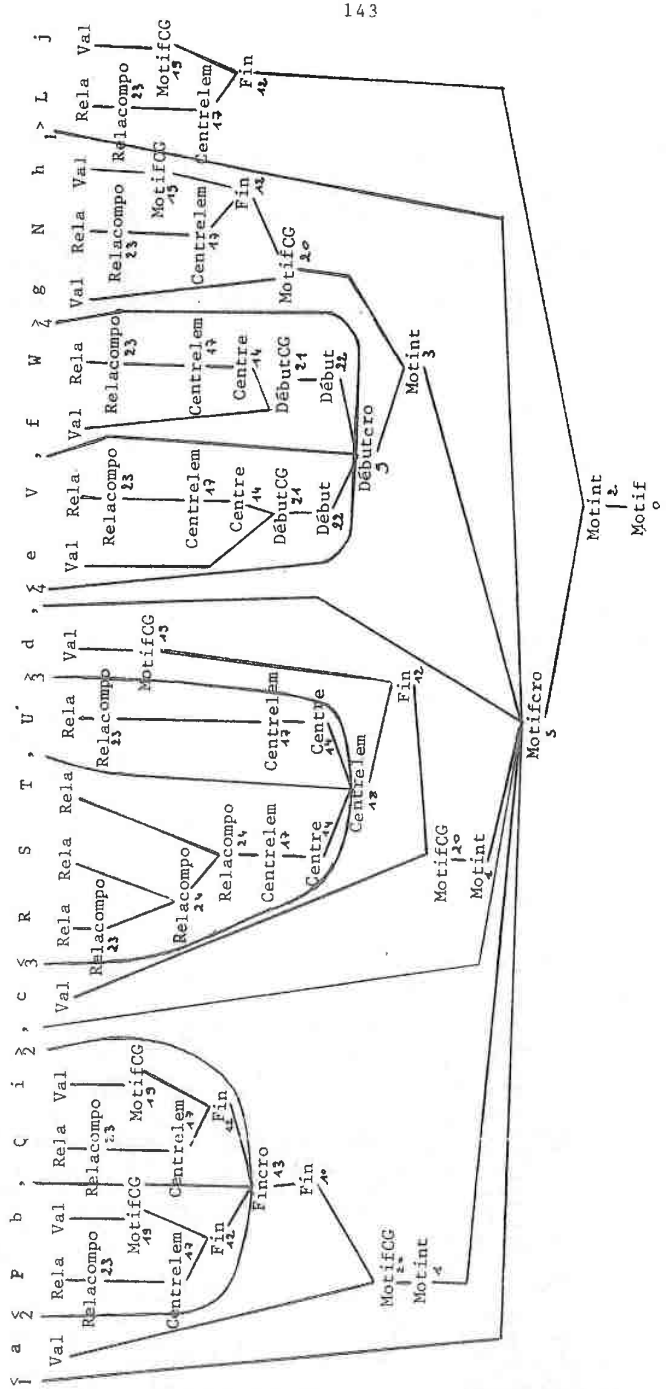
La grammaire G serait alors de simple précédence.

V.3.6. EXEMPLE.

Présentons ici un exemple de motif, les différentes étapes de son "décryptage" et la ramification résultat de son analyse. Dans le "film" du "décryptage", une ligne correspond soit à une entrée sur la pile, soit à une réduction. A

gauche, nous inscrivons dans le cas d'une entrée le caractère dont on entre la signification sur la pile, et dans le cas d'une réduction le numéro de la règle appliquée. A droite, nous représentons l'état de la pile. Ses différents éléments sont séparés par des ; . Les traits horizontaux signifient : "comme ci-dessus". Les arcs (complets) figurant dans la signification d'un non-terminal se retrouvent dans celle de tous ses ascendants donc dans le motif. Nous décidons de les sortir au fur et à mesure de leur formation pour alléger l'écriture... et la pile. Nous encadrons les arcs à sortir.

Nous traitons l'exemple comme si les règles b : <a^{x}> étaient chacune une infinité de règles b : <a> ou b : <a,a> ou ... .



Les numéros sous les non-terminaux sont les numéros de règles.

```
<1   <
a    < ; a
<2   ——— ; <
P    ———— ; (-,P,-)
23   ————————
17   ————————
b    ———————— ; b
19   < ; a ; < ; (-,P,-) ; b,b
12   < ; a ; < ; (-,P,b),b
,    ———————— ; ,
Q
.
.
.    .
12   < ; a ; < ; (-,P,b),b ;,,(-,Q,i),i
>2   ———————————————— ; >
13   < ; a ; (-,P,b),(-,Q,i),<b,i>
10   ————————————
20 { < ; a, [(a,P,b),(a,Q,i)] ,<b,i>
   { < ; a,<b,i>
1    < ; <b,i>
,    ———— ; ,
c    ———— ; c
<3   ———— ; <
R    ———— ; (-,R,-)
23   ————————
S    ———————— ; (-,S,-)
24   < ; <b,i> ; , ; c ; < ; (-,R,y1),(y1,S,-)
```



```
T    ———————————— ;(-,T,-)
24 { < ; <b,i> ; , ; c ; < ; (-,R,y1), [(y1,S,y2)] ,(y2,T,-)
   { < ; <b,i> ; , ; c ; < ; (-,R,y1),(y2,T,-)
17   ————————
14   ————————
,    ———————— ; ,
U    ———————— ; (-,U,-)
23   ————————————
17   ————————————
14   ————————————
>3   ———————————— ; >
18   < ; <b,i> ; , ; c ; (-,R,y1),(y2,T,-),(-,U,-)
d    ———————— ; d
19   ———————— ; d,d
12 { < ; <b,i> ; , ; c ; (-,R,y1), [(y2,T,d)] ,(-,U,d),d
   { < ; <b,i> ; , ; c ; (-,R,y1),(-,U,d),d
20 { < ; <b,i> ; , ; c , [(c,R,y1),(c,U,d)] ,d
   { < ; <b,i> ; , ; c,d
1    < ; <b,i> ; , ; d
,    ———— ; ,
<4   ———— ; <
e    ———— ; e
V    ———— ; (-,V,-)
23   ————————
17   ————————
14   ————————
21   < ; <b,i> ; , ; d ; , ; < ; e,(e,V,-)
```

```
22   < ; <b,i> ; , ; d ; , ; < ; (e,V,-)
 ,   ———————————————————— ; ,
 f   ———————————————————— ; f
 W   ———————————————————— ; (-,W,-)
23   ————————————————————————————————
17   ————————————————————————————————
14   ————————————————————————————————
21   < ; <b,i> ; , ; d ; , ; < ; (e,V,-) ; , ; f,(f,W,-)
22   < ; <b,i> ; , ; d ; , ; < ; (e,V,-) ; , ; (f,W,-)
 ≥   ———————————————————————————— ; >
 4
 9   < ; <b,i> ; , ; d ; , ; (e,V,-),(f,W,-)
 g   ———————————————————— ; g
 N   ———————————————————— ; (-,N,-)
23   ————————————————————————————————
17   ————————————————————————————————
 h   ———————————————————————————— ; h
19   ———————————————————————————— ; h,h
12   < ; <b,i> ; , ; d ; , ; (e,V,-),(f,W,-) ; g ; (-,N,h),h
20  { < ; <b,i> ; , ; d ; , ; (e,V,-),(f,W,-) ; g, [(g,N,h)] ,h
    { < ; <b,i> ; , ; d ; , ; (e,V,-),(f,W,-) ; g,h
 3  { < ; <b,i> ; , ; d ; , ; [(e,V,g),(f,W,g)] ,h
    { < ; <b,i> ; , ; d ; , ; h
 ≥   ———————————————— ; >
 1
 5   <b,i,d,h>
 L   ———————— ; (-,L,-)
23   ————————————————
17   ————————————————
```



```
 j   ———————————————— ; j
19   ———————————————— ; j,j
12   <b,i,d,h> ; (-,L,j),j
 2  { [(<b,i,d,h>,L,j)] ,j
    { j
 0   on supprime j car j est sommet d'un arc.
```

$$C(Motif) = E(Motif)$$
$$= \{(a,P,b),(a,Q,i),(y_1,S,y_2),(y_2,T,d),(c,R,y_1),$$
$$(c,U,d),(g,N,h),(e,V,g),(f,W,g),(<b,i,d,h>,L,j)\}$$

$$f(Motif) = \omega.$$

V.3.7. OCCURRENCES D'UN MOTIF M DANS L'INFORMATION I. ROLE DES INCONNUES.

Dans ce paragraphe, nous sous-entendrons souvent la référence à l'information interrogée I. De plus, a et b désigneront des informations élémentaires et ne seront ni des inconnues ni US.

V.3.7.1. Définition des occurrences de M dans I.

Effectuer dans l'information I la recherche spécifiée par le motif M, c'est chercher si M "a des occurrences" dans I. Définissons cette notion.

- Occurrences d'un arc (a, R, b) :

L'arc (a, R, b) a une occurrence dans I si et seulement si a et b sont des informations élémentaires qui figurent dans I, et y sont liées par la relation R.

- Occurrences d'un point isolé a :

Le point a a une occurrence si et seulement si a est une information élémentaire qui figure dans I.

- Occurrences d'un motif M sans inconnues :

Le motif M a une occurrence dans I si et seulement si chacun des éléments -arc ou point isolé- de sa signification a une occurrence.

- Occurrences d'un motif M quelconque :

Une occurrence d'un motif M ayant n inconnues x_1 x_2, ... , x_n est un n-uple (u_1, u_2, ... , u_n) d'informations élémentaires tel qu'en substituant dans M u_1 à x_1, u_2 à x_2, ... , u_n à x_n on obtienne un motif qui a une occurrence dans I.

Le motif M a une occurrence dans I si et seulement si il existe au moins une occurrence de M dans I.

En ce qui concerne x_o, on lira la remarque du paragraphe V.3.7.3.

V.3.7.2. Cas de US, Unité de Sélection.

De nombreuses informations, et en particulier les corpus documentaires, sont constituées par un ensemble (en général organisé, par exemple en liste) d'informations composantes ayant toutes la même structure ; ce cas est illustré par la fin de l'exemple II.2.2. Nous avons voulu donner le moyen de chercher des occurrences d'un motif à l'intérieur de chacune des informations composantes. C'est ainsi que nous avons introduit la notion d'unité de sélection ; ceci doit être considéré comme un outil commode mais pas du tout comme une composante essentielle de notre système.

Définition :

Lorsqu'une information I peut être décomposée en informations ayant toutes la même structure, on appelle "unité de sélection" chacune des informations composantes, à condition que la propriété suivante soit vérifiée : chaque information composante C doit posséder un représentant, c'est-à-dire un repère à partir duquel on puisse atteindre tout élément de C et qui puisse lui-même être atteint depuis l'extérieur de C.

Remarquons que la décomposition éventuelle d'une information I en unités de sélection fait partie de la spécification de sa structure. Remarquons de plus que l'on peut assurer artificiellement la propriété d'existence des représentants en créant des relations à cet effet.

Lorsqu'un motif (ou un filtre, cf. § V.5.2.3) contient le symbole US, il est conçu comme s'il ne s'adressait qu'à une seule unité de sélection, mais il s'adresse tour à tour à toutes celles-ci. Le symbole US désigne le représentant de chaque unité. Il joue le rôle d'une inconnue particulière qui prend ses valeurs dans l'ensemble des représentants.

V.3.7.3. Résultat d'un filtre F réduit à un motif M.

Par définition, le résultat de F est constitué par :

- un ensemble : l'ensemble des occurrences de M dans I (n-uples de valeurs des inconnues) (cf. § V.3.7.1).
- un booléen qui prend la valeur vrai si et seulement si cet ensemble n'est pas vide.

La recherche du résultat de F (ou M) s'appelle l'évaluation de F (ou M).

Remarque : x_o et les inconnues locales :

Parmi les inconnues figurant dans la signification d'un motif m, distinguons trois groupes :

- les inconnues x_1, x_2, ... que nous appellerons les "inconnues globales" ;
- x_o ;
- les inconnues locales.

Les inconnues locales ne figurent pas explicitement dans m et par conséquent ne peuvent être nommées en dehors de m. Elles n'ont que m comme "portée" (au sens des portées des variables dans les langages de programmation).

L'inconnue x_o figure dans le motif mais uniquement pour des raisons syntaxiques (cf. exemple V.1.2.6). On pose que sa portée est réduite au motif m. Ainsi, lorsque x_o sera employé dans divers motifs du même filtre, ces différentes utilisations seront indépendantes.

Au cours de l'évaluation d'un motif, il est évident que l'on a besoin de conserver les valeurs des inconnues locales et de x_o. Mais seules les valeurs des inconnues x_1, x_2, ... seront conservées dans le résultat final du motif.

Ainsi nous pouvons préciser la définition du résultat d'un filtre F réduit à un motif M ; il comporte :

- l'ensemble des p-uples de valeurs de ses inconnues globales ; nous l'appelons RESU(M) ou RESU(F).

- le booléen précisé ci-dessus, qui sera noté BOOL(M) ou BOOL(F).

Exemple : $M = a\ R\ S\ x_3 < T\ x_1\ ,U\ x_2 >$

L'ensemble fourni par l'évaluation de M est un ensemble de triplets de valeurs de x_1, x_2, x_3. Si l'on remplace x_3 par x_o, RESU(M) sera un ensemble de couples de valeurs de x_1 et x_2.

V.4. PASSAGE D'UN ENSEMBLE D'ARCS ET POINTS AUX MOTIFS DONT IL EST LA SIGNIFICATION.

Ce paragraphe n'est pas d'un intérêt pratique primordial et peut être sauté sans lacune importante pour la suite.

Etant donné un ensemble P d'arcs et points, il est immédiat de trouver un motif M_o dont P est la signification (cf. § V.4.2).Nous montrerons qu'un motif quelconque M ayant P pour signification peut être transformé en M_o par un jeu de transformations respectant la signification (cf. § V.4.3 à V.4.5). Inversement, à partir d'un motif trivial M_o, on sait générer, en effectuant les transformations réciproques, tous les motifs ayant même signification. Nous pourrons donc conclure à la fin du paragraphe V.4 que, étant donné un ensemble P d'arcs et points, on peut trouver tous les motifs dont il est la signification.

V.4.1. TERMINOLOGIE.

1) Etant donné une grammaire K, nous appelons K-ramification toute ramification engendrée par K au sens large [bibl 40 § 6.3]. Nous appelons K-motif toute phrase du langage engendré par K. Toute sous-phrase de ce langage est le mot des feuilles d'une K-ramification.

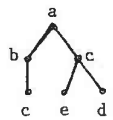
2) Rappel de notations relatives aux ramifications sur un ensemble W [bibl 39,40].

- On note + la "concaténation" de deux ramifications, qui juxtapose deux ramifications à m et n racines en une ramification à m+n racines.

- On note × l' "enracinement" : si a est un élément de W et r une ramification sur W, a×r est la ramification sur W, de racine a, dont les descendants immédiats de a sont les racines de r et dont r est la plus grande sous-ramification propre.

Exemple :

r est

alors a×r est 

- On note respectivement $\varphi(r)$ et $\rho(r)$ le mot des feuilles et le mot des racines de la ramification r.

- Soit une ramification à 1 racine. Nous appelons niveau 1 l'ensemble des noeuds descendants immédiats de cette racine, niveau 2 l'ensemble de leurs descendants immédiats, etc...

3) Règle principale d'une K-ramification.

Nous appelons règle principale de la K-ramification $r = a\times(s_1 + \ldots + s_i + \ldots)$, dans laquelle les ramifications s_i ont une seule racine, la règle de la grammaire K qui s'écrit:

$$a : \rho(s_1) \ldots \rho(s_i) \ldots$$

Dans l'exemple ci-dessus, la règle principale de la deuxième ramification est a : b c.

V.4.2. LE MOTIF TRIVIAL M_o ET LES GRAMMAIRES G', G", G"' .

Soit un ensemble P d'arcs et de points. Nous pouvons écrire le motif M_o qui est la concaténation (mise entre crochets si P comporte plus d'un élément) des arcs et points, séparés par des virgules ; chaque arc est écrit sous la forme de la concaténation de son origine, du nom de relation et du but. Ce motif, que nous appelons motif trivial est un G-motif et a P pour signification (démonstration immédiate).

Exemple : P = {(a,R,b),(<a,c>,S,d),e}
M_o = <aRb,<a,c>Sd,e>

Existence et non-unicité de M_o :

Quel que soit P, il est évident que M_o existe. Il est aussi évident que M_o n'est pas unique dès que P comporte plusieurs éléments : les divers M_o diffèrent par l'ordre de leurs constituants.

Grammaire G"' :

M_o est un G-motif mais peut en fait être considéré comme phrase d'un langage engendré par une grammaire beaucoup plus simple que G, que nous appelons G"' et décrivons ci-dessous.

Motif : Elem/1/<Elem$^{\times}$>/2

Elem : Val/3/Arc/4

Arc : Val Rela Val/5/<Val$^{\times}$> Rela Val/6

On peut vérifier facilement que le langage engendré par G''' est inclus dans celui engendré par G.

Pour démontrer que tout motif peut être transformé en une phrase de G''', nous aurons besoin d'une autre grammaire G' et de sa grammaire réduite G''. Nous les présentons ci-dessous.

Grammaire G' : Axiome : Motif.

Motif : Motint/50
Motint : Val/51/Val FinCG/52/Motifcro/53/Motifcro FinCG/54
Motifcro : <Motint$^{\times}$>/55
FinCG : Relacompo Val/56/Relacompo Val FinCG/57
Relacompo : Rela/59/Relacompo Rela/60

MotifCG : Val/70/Val Fin/71
Centre : Centrecro/72/Centresimple/73/
Fincro Centresimple/74
Début : Débutcro/75/Motifcro Centresimple/85/
Val Centresimple/86
DébutCG : Val Centre/76
Centresimple : Relacompo/77/Relacompo Val Centresimple/78
Débutcro : <Début$^{\times}$>/79
Centrecro : <Centre$^{\times}$>/80
Fin : Fincro/81/Fincro FinCG/82/FinCG/83
Fincro : <Fin$^{\times}$>/84

La grammaire G'' est la réduction de la grammaire G', c'est-à-dire la grammaire G' dans laquelle on a supprimé les règles de production "inutiles" [bibl. par exemple 46, § 2]. Elle comporte les règles de G' de numéro inférieur ou égal à 60.

Remarque :

Il existe un homomorphisme de grammaires, conservant les symboles terminaux, entre G et G'' et entre G et G''' .

Les différentes parties d'un G''-Motif sont beaucoup moins imbriquées qu'elles ne peuvent l'être dans un G-motif. C'est pour cela que les G''-motifs nous serviront d'intermédiaires dans la transformation de G-motifs en G'''-motifs.

V.4.3. TRANSFORMATION T SUR LES G-RAMIFICATIONS.

V.4.3.1. Présentation.

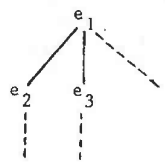
Soit un motif quelconque M. Il est le mot des feuilles d'une G-ramification de racine "Motif". Nous allons chercher à transformer cette G-ramification en une G'-ramification de façon à ce que la signification qui en est le mot des feuilles reste invariante.

Pour définir cette transformation T, nous allons l'associer à la grammaire G : à chaque règle de G, nous ferons correspondre une transformation élémentaire. La transformation de la G-ramification se fera de façon ascendante : par exemple, on considérera tous les noeuds de la ramification dans l'ordre postfixé ; pour chaque noeud N, on effectuera sur la sous-ramification S de racine N la transformation associée à sa règle principale (cf. § V.4.1.3). Ainsi, lorsque la transformation relative à N sera entreprise, les sous-ramifications incluses auront déjà été transformées. Ce processus est le même que celui expliqué pour l'algorithme de décryptage.

Nous allons définir ces transformations élémentaires puis, au paragraphe V.4.4, montrer qu'elles jouent bien leur rôle qui est de transformer toute G-ramification en G'-ramification et de conserver la signification de la partie du motif qui en est le mot des feuilles.

V.4.3.2. Les quatre familles de transformations élémentaires.

a) Pour présenter chacune des transformations élémentaires, nous donnerons les formes des ramifications concernées, d'abord avant transformation puis après celle-ci.

Les transformations élémentaires conservent toutes l'étiquette de la racine de la ramification. Comme nous l'avons dit au paragraphe V.4.3.1, la transformation élémentaire s'appliquant à la ramification S = 

est celle qui est associée à la règle e_1 : e_2 e_3 ... . Lorsqu'elle intervient, les sous-ramifications propres ont déjà été transformées. Elle transforme S en une autre G-ramification S'.

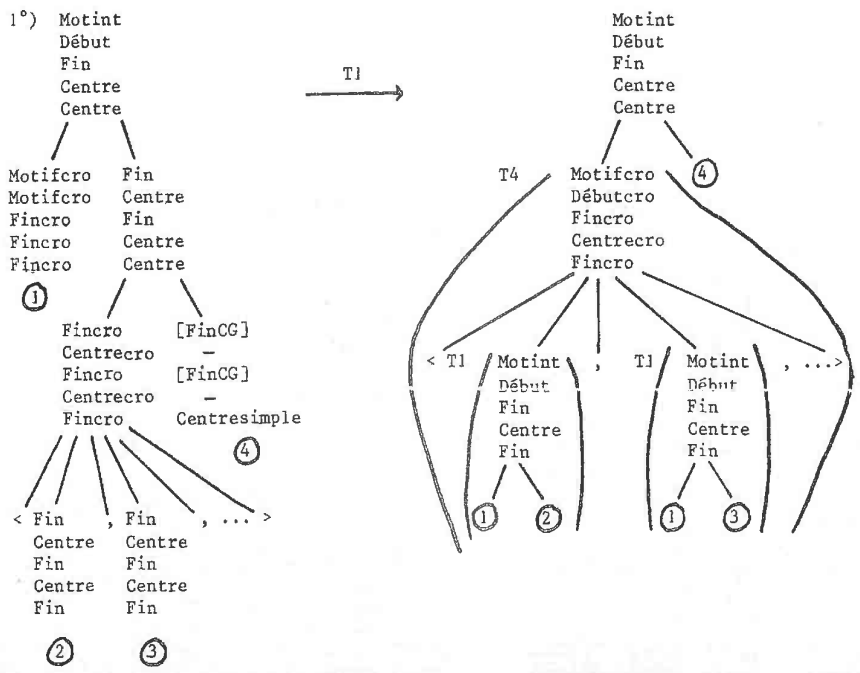
Mais en fait nous voulons démontrer que les transformations changent les G-ramifications en G'-ramifications. Pour cette démonstration (§ V.4.4), nous procédons par récurrence. L'hypothèse de récurrence stipule, lorsqu'on s'intéresse à la transformation élémentaire touchant la ramification S, que les ramifications ayant moins de points que S, et en particulier les sous-ramifications propres de S, se transforment par T en G'-ramifications. Nous ne définirons les transformations élémentaires que sous cette hypothèse. Ainsi, nous ne définirons les transformations élémentaires que sur des ramifications dont les sous-ramifications propres sont des G'-ramifications.

b) Les transformations élémentaires peuvent être regroupées en quatre familles, T1, T2, T3, T4. Les transformations d'une même famile Tk font passer d'une même ramification r à une même ramification r', aux étiquettes près.

Pour abréger la définition des transformations, nous démultiplierons les étiquettes dans les schémas qui décrivent les familles de transformations : si une famille Tk groupe 4 transformations élémentaires, chaque noeud sera étiqueté par quatre termes du vocabulaire ; la première transformation correspondra à toutes les premières étiquettes, ..., la quatrième à toutes les dernières étiquettes. Cependant si l'étiquette d'un noeud est la même pour toutes les transformations, on ne l'écrira qu'une fois.

Lorsqu'une sous-ramification n'a pas besoin d'être détaillée, nous la repérons simplement par un numéro, comme le fait P. Marchand dans [36]. Lorsqu'un non-terminal peut exister ou non, nous le mettons entre crochets carrés.

c) Les transformations T1 : distribution d'une origine.



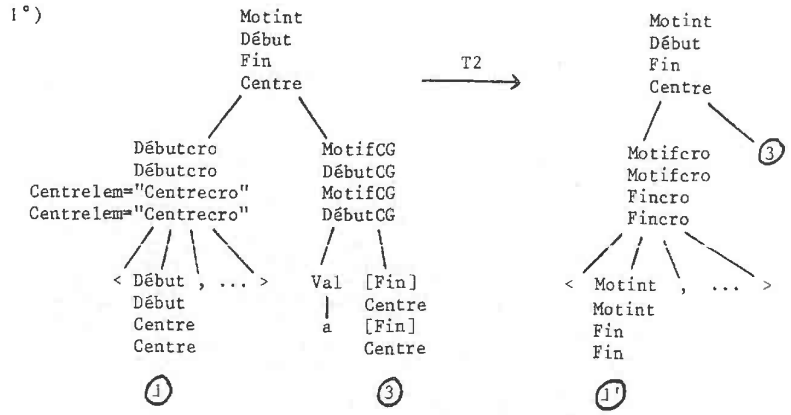
Cette définition de T1 est récursive. Cette récursivité s'ajoute à celle qui est attachée à la transformation ascendante de la ramification correspondant au motif. Remarquons enfin que T1 utilise la transformation T4.

2°) Cas "dégénéré".

Si le noeud droit de niveau 1 est de la forme FinCG (pour Fin) ou Centresimple (pour Centre), alors T1 est l'identité, sauf que l'intermédiaire Fin ou Centre est supprimé. Le mot des feuilles est inchangé.

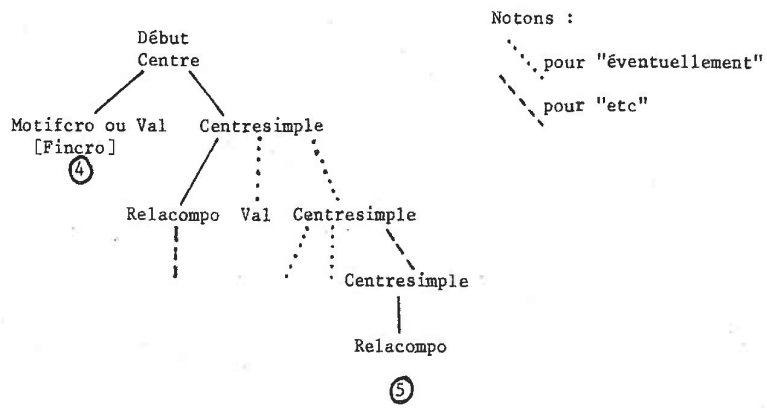
d) Les transformations T2 : distribution d'une extrémité.

Une transformation T2 sera toujours suivie d'une transformation T1. Centrelem peut être de la forme appelée Centrecro dans G' ou de la forme appelée Relacompo. Nous distinguerons ces deux cas.

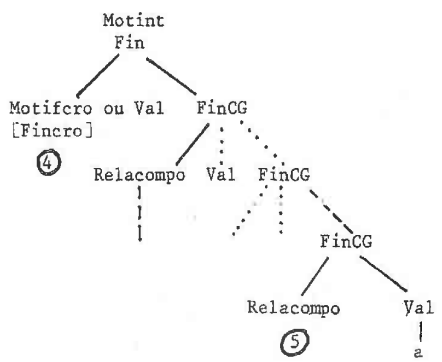


Passage de 1 à 1' :

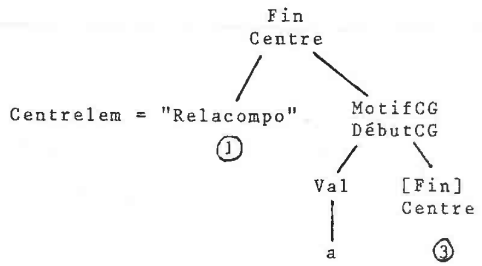
D'après le fait que les sous-ramifications ont déjà été transformées, et d'après la conclusion de V.4.3.2.e), 1 est de la forme (donnée par G') :



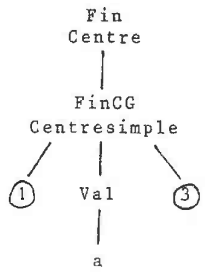
1' est alors de la forme :



2°) Cas "dégénéré".

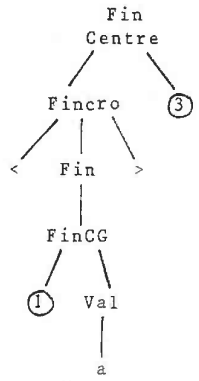


Si 3 est absent ou de la forme FinCG (pour Fin) ou Centresimple (pour Centre), la transformation T2 donne :



Le mot des feuilles est inchangé.

Sinon, la transformation T2 donne :



Dans ce cas, T2, qui est un peu artificielle a priori, a pour but de permettre d'appliquer ensuite la transformation T1.

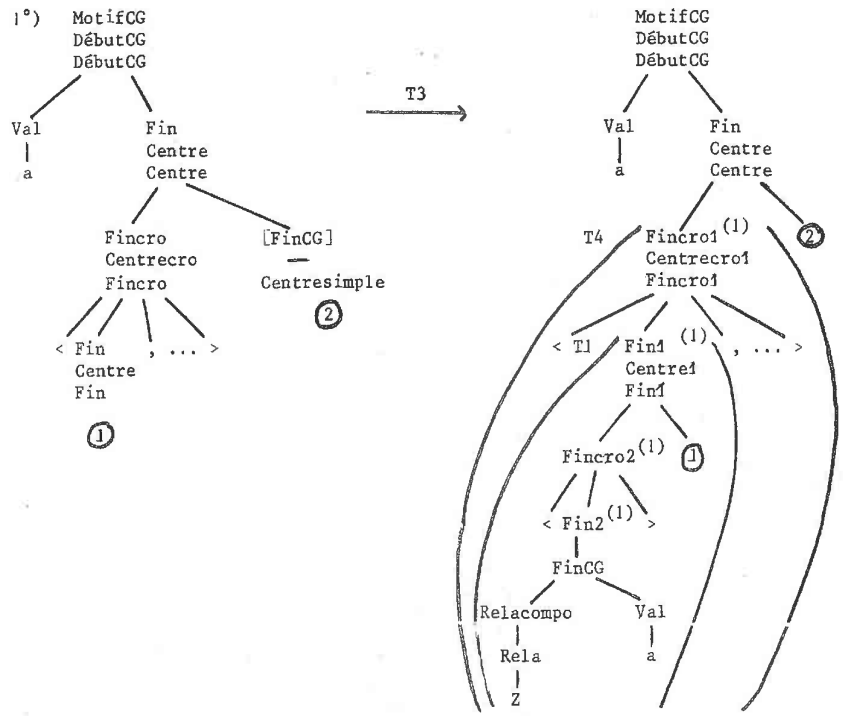
e) Les transformations T3 : transformation des MotifCG et DébutCG.

Si, dans un MotifCG ou DébutCG de la forme Val Fincro... ou Val Centrecro..., on distribue la valeur (Val) comme origine dans le Fincro ou Centrecro, on transforme le MotifCG ou DébutCG

en un Motif ou Début qui n'est plus "convergent à gauche". Pour remédier à cela, on va introduire une relation atomique particulière, Z (supposée différente des relations de la structure). Cette relation Z a les propriétés suivantes :

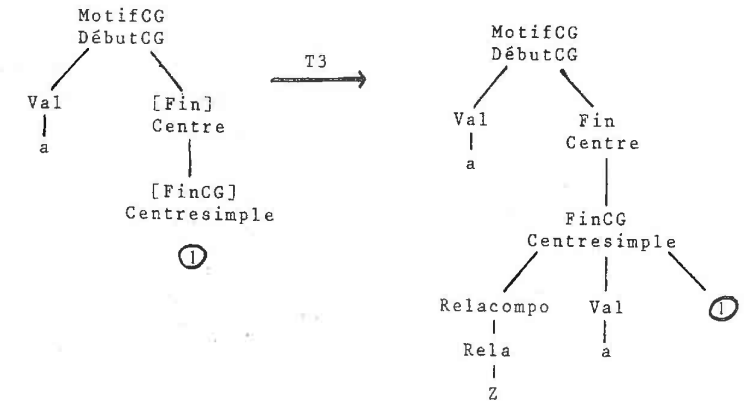
- elle n'apporte aucune signification, c'est-à-dire que les arcs aZb sont "inconsistants", sont à supprimer purement et simplement.

- elle admet un nombre quelconque d'arguments (c'est-à-dire : il existe une relation Z par nombre d'arguments).



(1) Ces numéros sont seulement là pour faciliter la rédaction du paragraphe V.4.4.4.

2°) Cas "dégénéré".



Remarque :

Grâce à cette préparation des MotifCG et DébutCG par la transformation T3, la transformation T2 n'entraîne jamais le changement du nombre d'arguments d'une relation.

Exemple : <<aF,bG>c,d>He

Si le MotifCG c n'était pas modifié par T3, la transformation T2 transformerait <aF,bG>c en <aFc,bGc> et H aurait trois arguments : c, c, d.

Au contraire, T3 modifie c en cZc. T2 transforme alors <aF,bG>cZc en <aFc,bGc>Zc, si bien que c a été distribué dans le crochet mais n'a pas été dédoublé comme argument de H.

Remarque importante : cas des Motint : MotifCG
et des Début : DébutCG.

La transformation T3 diffère de T1 par l'introduction de la relation Z. Or cette introduction, destinée à conserver le caractère de convergence à gauche, est inutile lorsque le MotifCG (ou le DébutCG) constitue en fait un Motint (ou Début). Il y a dans ce cas deux solutions dont nous choisirons la seconde :

- ou faire quand même la transformation T3 qui alourdit l'expression mais n'a pas d'autre inconvénient ;

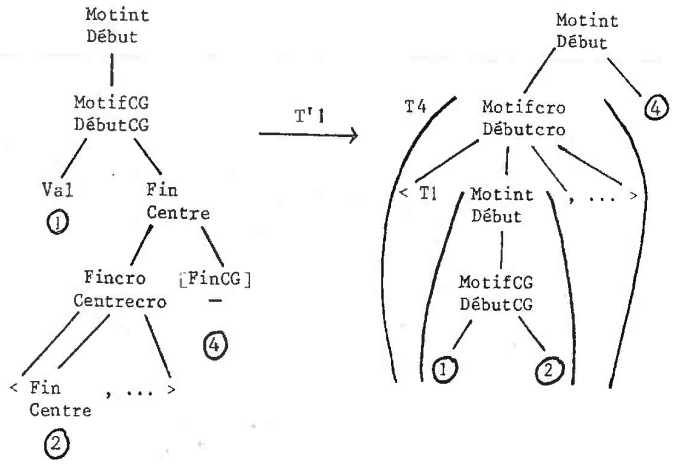
- ou ne pas la faire dans ce cas et faire la transformation T1, mais au niveau de Motint (ou Début), en supprimant l'intermédiaire MotifCG (ou DébutCG). C'est ce que nous ferons. Nous devons donc :

1°) préciser que nous n'effectuons T3 que lorsque MotifCG (ou DébutCG) est précédé[1] de Débutcro ou Centrelem ; remarquons également que nous pouvons aussi éviter (et éviterons) de faire T3 dégénéré lorsque MotifCG (ou DébutCG) est précédé[1] de Relacompo.

2°) ajouter aux variantes de la transformation T1 les variantes suivantes (nous appelons alors la transformation : T'1).

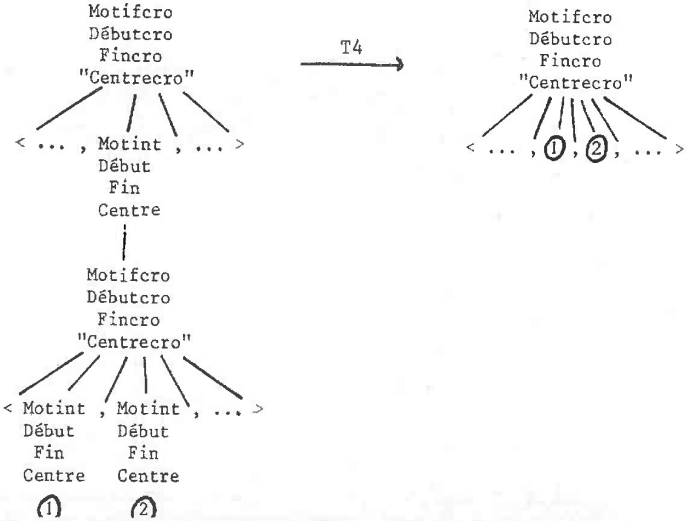
(1) Plus exactement : "est le 2ème terme de la partie droite d'une règle dont le 1er terme est : ".

- Dans le cas "non dégénéré" (cf. V.4.3.2.c) :



- Dans le cas dégénéré, on supprime simplement les intermédiaires MotifCG ou DébutCG et Fin ou Centre. Le mot des feuilles est inchangé.

e) Les transformations T4 : suppression des crochets inutiles.



Le résultat de T4 est que les membres des crochets ne sont pas eux-mêmes des crochets purs (un membre d'un Fincro n'est pas un Fincro seul).

V.4.3.3. Association des transformations aux règles.

Donnons pour chaque règle la transformation que nous lui associons. En plus des transformations T1 à T4, nous considérons quelques transformations purement grammaticales qui transforment légèrement la ramification considérée sans modifier la sous-phrase qui en est le mot des feuilles ; ces transformations sont dues au fait que dans G' on distingue deux sortes de Centrelem, les Centrecro et les Centresimple, alors qu'on ne fait pas cette distinction dans G.

Règle	Transformation
0	Rien
1	T'1(Motint)
2	T1(Motint)
3	T2(Motint) puis T1(Motint)
4	Rien
5	T4(Motifcro)
6	Rien
7	T1(Début)
8	T2(Début) puis T1(Début)

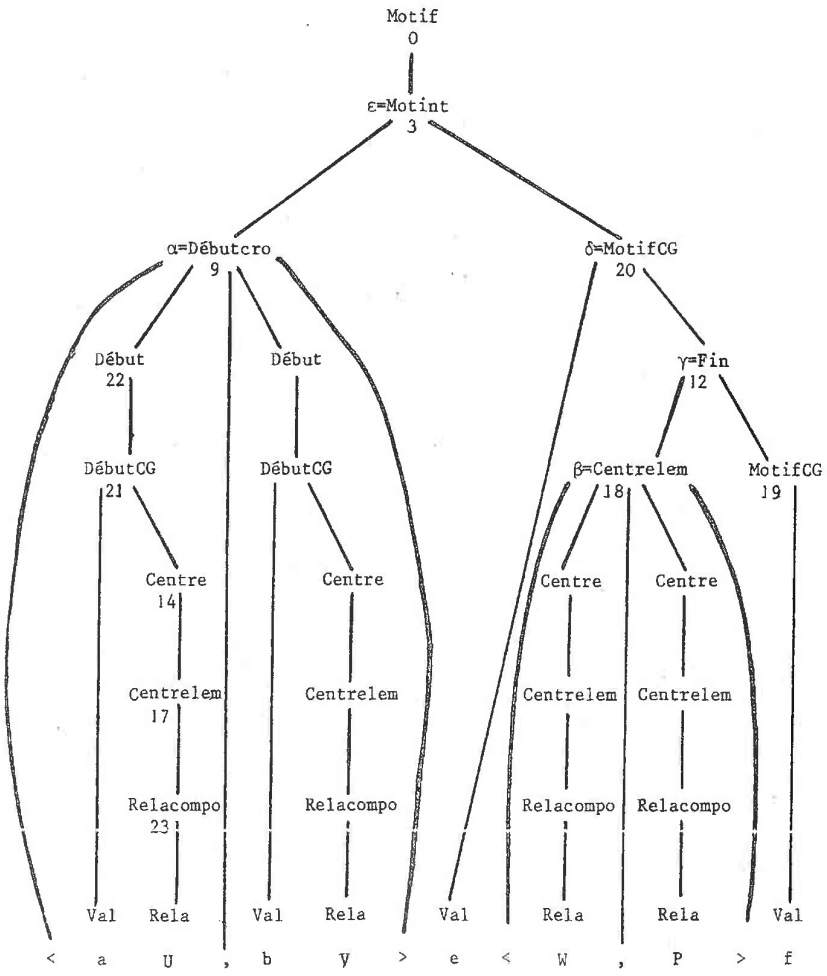
9	T4(Débutcro)
10	Rien
11	T1(Fin)
12	T2(Fin) puis T1(Fin)
13	T4(Fincro)
14	Si Centrelem est devenu Relacompo, Centre devient : Centre - Centresimple - Relacompo
15	T1(Centre)
16	T2(Centre) puis T1(Centre)
17	Centrelem devient Relacompo
18	T4(Centrelem) ; Centrelem devient Centrecro
19	Rien
20	T3(MotifCG) si MotifCG est précédé[1] de Centrecro ou Débutcro ou, pour le cas non dégénéré, de Relacompo.
21	T3(DébutCG) si DébutCG est précédé[1] de Centrecro ou Débutcro ou, pour le cas non dégénéré, de Relacompo.
22	T'1(Début)
23	Rien
24	Rien

Les règles définissant Val et Rela ne subissent aucune transformation.

(1) Plus exactement : "est le 2ème terme de la partie droite d'une règle dont le 1er terme est :".

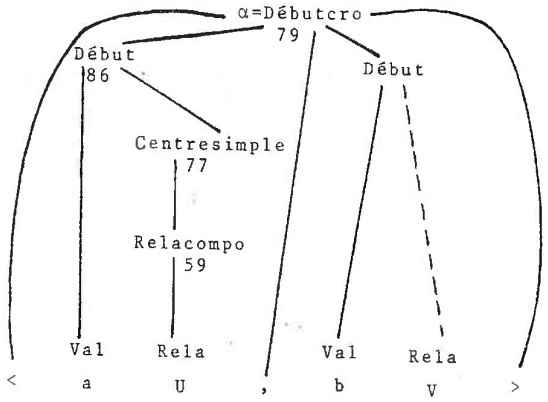
V.4.3.4. Exemple.

Cherchons l'effet de la transformation T sur le motif dont la ramification résultat de l'analyse est la suivante :

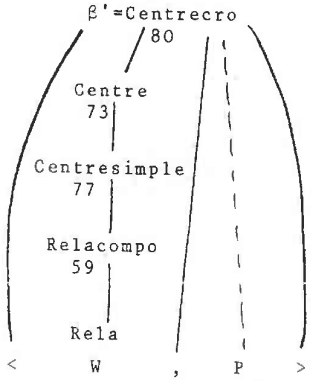


Le numéro inscrit sous certains noeuds désigne la règle de grammaire qui leur correspond.

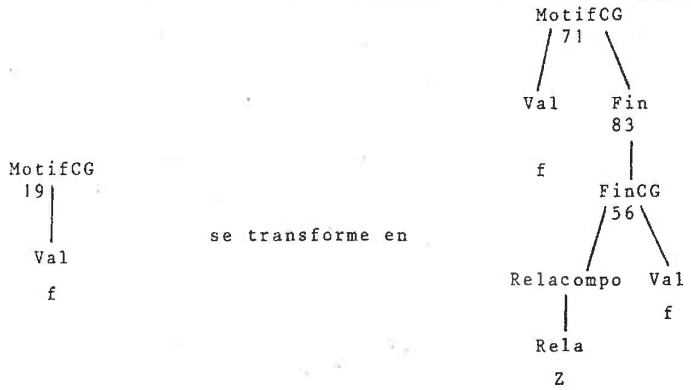
Après les transformations associées aux règles n° 23, 17, 14, 21 (ici, rien), 22 (T'1(Début)), la sous-ramification de racine α, dont le mot des feuilles ne change pas, devient :



Après les transformations associées aux règles 23, 17, 14, 18, la sous-ramification de racine β, dont le mot des feuilles ne change pas, devient :

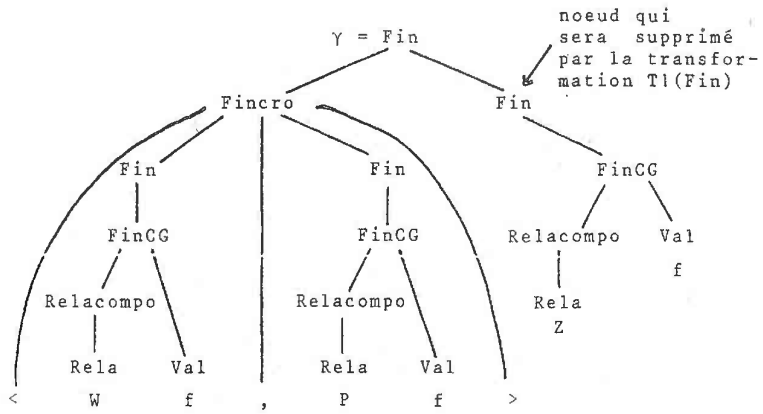


La sous-ramification suivante :



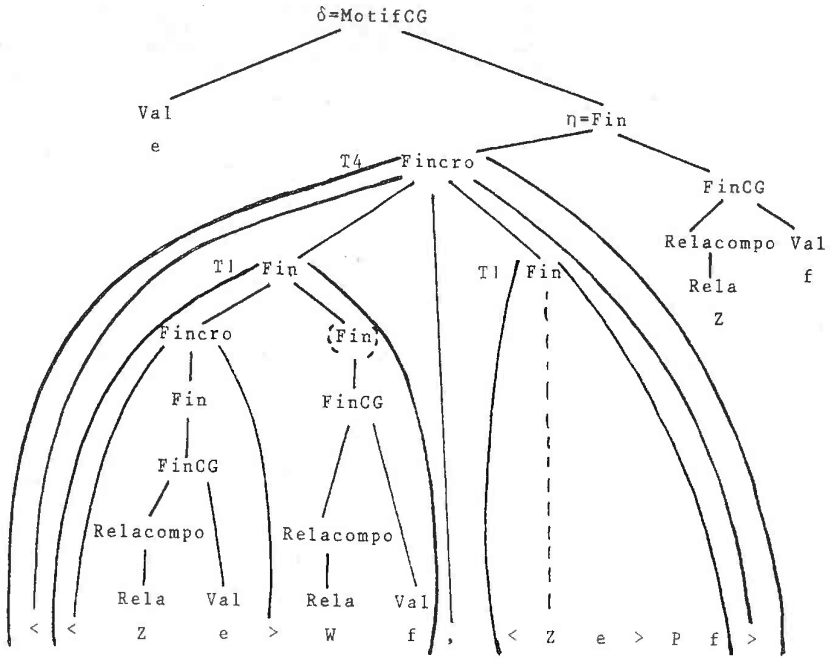
En effet, MotifCG est précédé de Centrecro.

Ensuite, la sous-ramification de racine γ devient, après transformation par T2(Fin) :



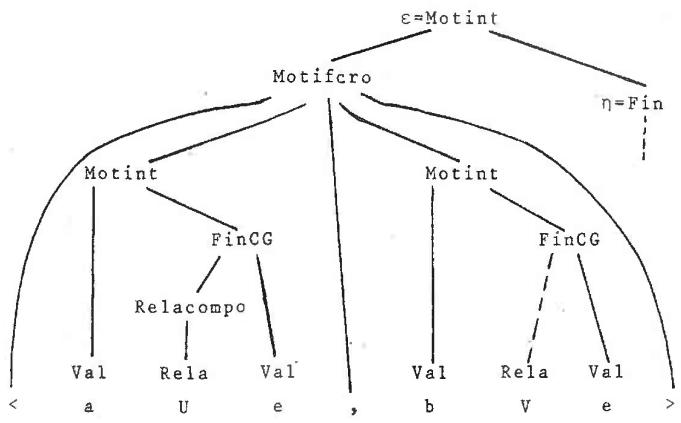
La transformation T1(Fin) est dégénérée et se contente de supprimer le noeud Fin de 1er niveau.

La transformation T3 (MotifCG) appliquée au noeud δ donne :



Les T1 sont dégénérés donc consistent seulement à supprimer le noeud Fin que nous avons mis entre parenthèses. T4 n'agit pas.

La transformation T2 (Motint), appliquée au noeud ε, donne :



On applique enfin la transformation T1 (Motint) à ce même noeud ε, ce qui donne comme mot des feuilles :

<<aUe,bVe><Ze>Wf,<aUe,bVe><Ze>Pf>Zf.

Aucune transformation n'intervient pour la règle n° 0.

V.4.4. LA TRANSFORMATION T TRANSFORME UN G-MOTIF M EN UN G'-MOTIF ET CONSERVE SA SIGNIFICATION.

Ce titre constitue un théorème dont voici la démonstration.

V.4.4.1. Principe de la démonstration.

Nous avons déjà dit que T est décrite sur la ramification engendrée par G dont M est le mot des feuilles. Pour démontrer le théorème, nous raisonnerons sur cette ramification et ses sous-ramifications, par récurrence sur leur nombre de points. Ainsi nous démontrerons le lemme suivant :

lemme : soit une G-ramification r. Si toutes les G-ramifications ayant moins de sommets que r sont transformées par T en G'-ramifications en conservant la signification de leur mot des feuilles, alors il en est de même pour r.

Si l'on suppose ce lemme démontré, le théorème l'est car T appliqué à Val et Rela est l'identité.

Pour démontrer le lemme, nous allons considérer des ramifications $r = a \times (s_1 + s_2 + \ldots)$ dont les sous-ramifications s_1, s_2, ... auront déjà été transformées en G'-ramifications (en ayant conservé leur signification). C'est ce qui nous a incitée à ne décrire les transformations Tk que dans ce cas (§ V.4.3.2.a).

Pour transformer chaque ramification r par T, il faut lui appliquer la transformation Tk associée à sa règle principale (cf. § V.4.3.1). En fait, nous grouperons ensemble les ramifications appelant les mêmes transformations et raisonnerons sur ces transformations et les règles principales des ramifications.

V.4.4.2. Démonstration du lemme pour les transformations T4 (Règles n° 5, 9, 13, 18).

Raisonnons par exemple sur la règle n° 13 : Fincro:<Fin$^{\times}$>.

Par hypothèse, chaque sous-ramification de racine Fin (qui a moins de sommets que la ramification considérée) a été transformée par T en une G'-ramification. Il lui correspond donc l'une des règles :

Fin : Fincro
Fin : Fincro FinCG
Fin : FinCG.

S'il lui correspond la 2ème ou la 3ème règle, T4 la laisse inchangée. S'il lui correspond la 1ère, T4 transforme simplement le Fincro en la suite de ses membres (qui ne sont pas eux-mêmes des Fincro puisque chaque sous-ramification a déjà été modifiée).

La ramification r considérée, de racine Fincro, est donc transformée en une ramification de racine Fincro, correspondant à la même règle dans G' que dans G, dont aucun membre n'est lui-même un Fincro seul. De plus, la définition même de la signification d'un Fincro montre que la signification de la sous-phrase $\varphi(r)$ reste inchangée.

Nous raisonnerions de façon tout à fait analogue sur les règles de n^{os} : 5 (Motifcro), 9 (Débutcro), 18 (Centrelem dans le cas où il signifie "Centrecro"). Le lemme est donc démontré dans ce cas.

V.4.4.3. Démonstration du lemme pour les transformations T1 (Règles n° 2, 7, 11, 15) et T'1 (règles n° 1 et 22).

Considérons d'abord la transformation T1.

Si l'on se trouve dans le cas dégénéré, le lemme est évident. Etudions le cas contraire, par exemple pour la règle n° 2.

Les ramifications telles que Motint ont moins de points que r. T1 les transforme donc en G'-ramifications en conservant leur signification.

Comme T4 transforme une G-ramification en G'-ramification, T1(r) est bien une G'-ramification et la règle de G' qui l'engendre est :

Motint:Motifcro[FinCG] (règle n° 53 ou 54).

Cherchons maintenant si T1 conserve la signification de r. Pour cela, cherchons la signification avant et après transformation, en considérant toutes les règles de G' en tant que cas particuliers de règles de G et en désignant les ramifications ①, ②, ... par leur numéro.

- $S(r) = E(1), S_{f(1)}(Fin)$ (cf. § V.3.4)

Or $S(Fin)^{(1)} = \begin{cases} \text{si 4 existe} & E(Fincro), S_{f(Fincro)}{}^{(4)} \\ \text{sinon} & S(Fincro) \end{cases}$

Or $S(Fincro) = E(2), E(3), \ldots, <f(2), f(3), \ldots>$

Donc $S(Fin) = \begin{cases} \text{si 4 existe} & E(2), E(3), \ldots, S_{<f(2),f(3),\ldots>}{}^{(4)} \\ \text{sinon} & E(2), E(3), \ldots, <f(2), f(3), \ldots> \end{cases}$

Donc $S(r) =$

$$\begin{cases} \text{si 4 existe} & E(1), E_{f(1)}(2), E_{f(1)}(3), \ldots, S_{<f(2),f(3),\ldots>}{}^{(4)} \\ \text{sinon} & E(1), E_{f(1)}(2), E_{f(1)}(3), \ldots, <f(2), f(3), \ldots> \end{cases}$$

- $S(T1(r)) =$

$$\begin{cases} \text{si 4 existe} & E(Motifcro), S_{f(Motifcro)}{}^{(4)} \\ \text{sinon} & S(Motifcro) \end{cases}$$

Or $S(Motifcro) = E(\text{1er Motint}), E(\text{2ème Motint}), \ldots, <f(\text{1er Motint}), f(\text{2ème Motint}), \ldots>$

En effet, nous ne tenons compte ni de T1 ni de T4 qui conservent la signification.

---

(1) Par abus de langage signalé au paragraphe V.3.2.

Or $S(\text{ième Motint}) = E(1), S_{f(1)}(i+1)$

Donc $S(Motifcro) = E(1), E_{f(1)}(2), E(1), E_{f(1)}(3), \ldots, <f(2), f(3), \ldots>$

$= E(1), E_{f(1)}(2), E_{f(1)}(3), \ldots, <f(2), f(3), \ldots>$

Donc $S(T1(r)) =$

$$\begin{cases} \text{si 4 existe} & E(1), E_{f(1)}(2), E_{f(1)}(3), \ldots, S_{<f(2),f(3),\ldots>}{}^{(4)} \\ \text{sinon} & E(1), E_{f(1)}(2), E_{f(1)}(3), \ldots, <f(2), f(3), \ldots> \end{cases}$$

- Donc $S(r) = S(T1(r))$.

La démonstration du lemme est donc établie pour la règle n° 2. Elle le serait de la même façon pour les règles n° 7, 11 et 15. Pour la transformation T'1 (règles n° 1 et 22), la démonstration est tout à fait analogue.

V.4.4.4. Démonstration du lemme pour les transformations T3 (Règles n° 20 et 21).

Les schémas de définition de T3 prouvent qu'une G-ramification est bien transformée en une G'-ramification si ses sous-ramifications le sont.

Voyons ce qu'il en est de la signification.

Cas 1°) : Dans le schéma de définition de T3, prenons par exemple, pour chaque noeud, la dernière version (règle n° 21, Centre de la forme Fincro Centre-simple).

- $S(r) = a, S_a(\text{Centre})$

  Or $S(\text{Centre}) = E(\text{Fincro}), S_{f(\text{Fincro})}(2)$

  Or $S(\text{Fincro}) = E(1), \ldots, <f(1), \ldots>$

  Donc $S(\text{Centre}) = E(1), \ldots, S_{<f(1), \ldots>}(2)$

  Donc $S(r) = a, E_a(1), \ldots, S_{<f(1), \ldots>}(2)$

- $S(T3(r)) = a, S_a(\text{Centre})$

  Or $S(\text{Centre}) = E(\text{Fincro1}), S_{f(\text{Fincro1})}(2)$

  $S(\text{Fincro1}) = E(\text{Fin1}), \ldots, <f(\text{Fin1}), \ldots>$

  $S(\text{Fin1}) = E(\text{Fincro2}), S_{f(\text{Fincro2})}(1)$

  $S(\text{Fincro2}) = E(\text{Fin2}), <f(\text{Fin2})>$

  $S(\text{Fin2}) = S^a(\text{Relacompo}), Ef(a)$

  $= (-, Z, a), a$

Donc en remontant dans la ramification :

$S(\text{Fincro2}) = (-, Z, a), <a>$

$S(\text{Fin1}) = (-, Z, a), S_a(1)$

$S(\text{Fincro1}) = (-, Z, a), E_a(1), \ldots, <f(1), \ldots>$

$S(\text{Centre}) = (-, Z, a), E_a(1), \ldots, S_{<f(1), \ldots>}(2)$

Donc $S(T3(r)) = a, (a, Z, a), E_a(1), \ldots, S_{<f(1), \ldots>}(2)$

$= a, E_a(1), \ldots, S_{<f(1), \ldots>}(2)$

d'après la propriété de Z.

- Donc $S(r) = S(T3(r))$.

La démonstration est donc établie dans ce cas pour la règle n° 21 avec Centre de la forme Fincro Centresimple. Elle serait analogue pour l'autre possibilité relative à la règle n° 21 et pour la règle n° 20.

Cas 2°) : la démonstration est immédiate grâce à la propriété de Z.

Le lemme est donc démontré pour T3.

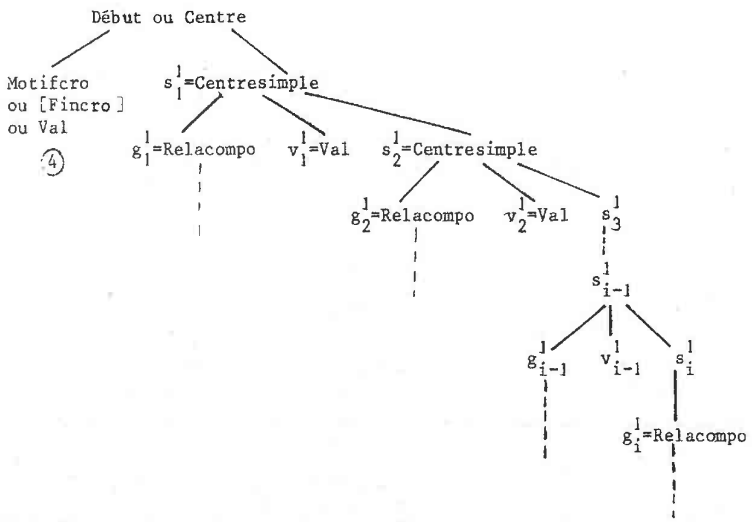
V.4.4.5. Démonstration du lemme pour les transformations T2 suivies de T1 (Règles n° 3, 8, 12, 16).

Supposons que la règle $a:\rho(s_1),\rho(s_2),\ldots$ soit l'une des quatre règles auxquelles on associe T2 (n°s 3, 8, 12, 16). T2 transforme la ramification $r = a \times (s_1 + s_2 + \ldots)$ en une G-ramification à laquelle on peut appliquer T1. Après avoir appliqué T1, on obtient donc une G'-ramification (d'après V.4.4.3).

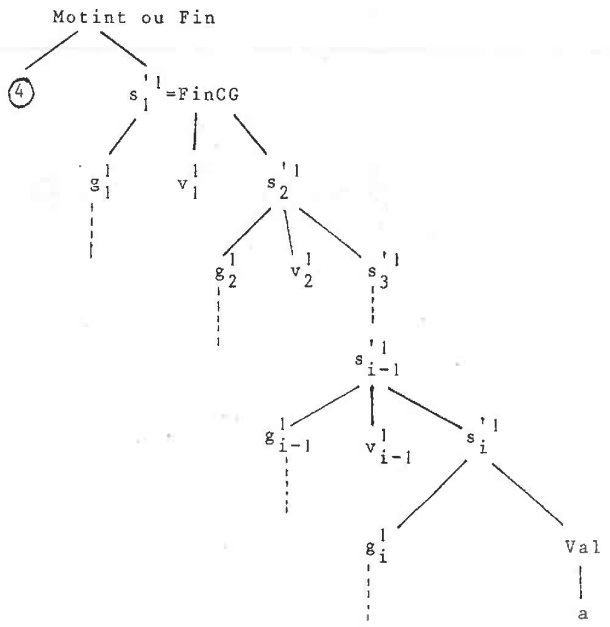
Reste à démontrer que T2 suivi de T1 ou, ce qui est suffisant, T2 seul conserve la signification.

- Reprenons le schéma de la transformation T2.

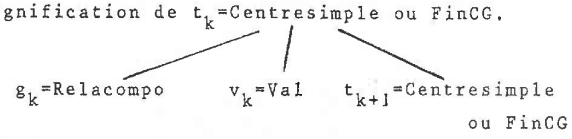
Rappelons la forme de 1 :



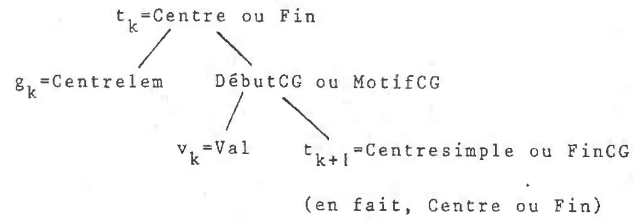
et celle de 1' :



- Avant d'évaluer la signification de 1 et de 1', évaluons la signification de t_k=Centresimple ou FinCG.



Cette G'-ramification est une contraction, dans un cas particulier, de la G-ramification suivante :



La signification de cette ramification est :

$$S(t_k)=S^{d(\text{DébutCG ou MotifCG})}(g_k),Ef(\text{DébutCG ou MotifCG})$$

Or $d(\text{DébutCG ou MotifCG}) = v_k$

$$Ef(\text{DébutCG ou MotifCG}) = S_{v_k}(t_{k+1})$$

Donc $S(t_k) = S^{v_k}(g_k),S_{v_k}(t_{k+1})$

- Evaluons maintenant la signification de 1 et de 1' .

$$S(1) = E(4),S_{f(4)}(s_1^1)$$

$$= E(4),S^{v_1^1}_{f(4)}(g_1^1),S_{v_1^1}(s_2^1)$$

$$= E(4),S^{v_1^1}_{f(4)}(g_1^1),S^{v_2^1}_{v_1^1}(g_2^1),S_{v_2^1}(s_3^1)$$

....

$$= E(4),S^{v_1^1}_{f(4)}(g_1^1),S^{v_2^1}_{v_1^1}(g_2^1),\ldots,S^{v_{i-1}^1}_{v_{i-2}^1}(g_{i-1}^1),S_{v_{i-1}^1}(s_i^1)$$

Posons $Q1 = E(4),S^{v_1^1}_{f(4)}(g_1^1),S^{v_2^1}_{v_1^1}(g_2^1),\ldots,S^{v_{i-1}^1}_{v_{i-2}^1}(g_{i-1}^1)$

Le calcul de S(1') jusqu'à ce stade donne rigoureusement le même résultat, en remplaçant les s par s'.

Or $S_{v_{i-1}^1}(s_i^1) = S_{v_{i-1}^1}(g_i^1)$

$S_{v_{i-1}^1}(s_i'^1) = S^a_{v_{i-1}^1}(g_i^1),a$ (cas particulier de la règle n° 12 de G)

Donc $S(1) = Q1,S_{v_{i-1}^1}(g_i^1)$

$$S(1') = Q1,S^a_{v_{i-1}^1}(g_i^1),a$$

- Cherchons enfin les significations S1 et S2 de r respectivement avant et après transformation par T2. Raisonnons par exemple sur la règle n° 8.

. $S(r) = S^{d(\text{DébutCG})}(\text{Débutcro}),E(\text{DébutCG})$

Or $S(\text{DébutCG}) = a,S_a(3)$

$S(\text{Débutcro}) = S(1),\ldots$

Donc $S(r) = S^a(1),\ldots,S_a(3)$

$$= Q1,S^a_{v_{i-1}^1}(g_i^1),\ldots,S_a(3)$$

/

$S(T2(r)) = E(\text{Motifcro}),S_{f(\text{Motifcro})}(3)$

Or $S(\text{Motifcro}) = E(1'),\ldots,<f(1'),\ldots>$

Donc $S(T2(r)) = E(1'),\ldots,S_{<f(1'),\ldots>}(3)$

$$= Q1,S^a_{v_{i-1}^1}(g_i^1),\ldots,S_{<a,a,\ldots>}(3)$$

Nous remarquons l'identité des significations au dernier terme près. Ces derniers termes paraissent comporter une contradiction car (3) semble "débuter" par des relations binaires dans S1 et n-aires (n>2) dans S2. Or la transformation T3 prépare les MotifCG et DébutCG de telle sorte que leur Fin ou Centre (un ou plusieurs membres) "débute" par Z qui admet un nombre d'arguments arbitraire. Nous pouvons grâce à cela écrire :

$$S_a(3) = S_{<a,a,\ldots>}(3)$$

donc $S(r) = S(T2(r))$.

Le lemme est donc démontré pour la règle n° 8. Il le serait de la même manière pour les autres règles de G appelant la transformation T2 suivie de T1, c'est-à-dire les règles n° : 3, 12, 16. La démonstration est évidente pour le cas dégénéré.

V.4.4.6. Conclusion.

Pour les autres règles que celles passées en revue aux paragraphes précédents, il est évident qu'elles transforment bien les G-ramifications en G'-ramifications et laissent invariante la signification de leur mot des feuilles.

Le lemme est donc complètement démontré. Donc aussi le théorème : la transformation T transforme tout G-motif en un G'-motif de même signification.

Mais G" est la grammaire G' réduite, c'est-à-dire à laquelle on enlève les règles "inutilisées". Tout G'-motif est en fait un G"-motif.

En conclusion, la transformation T transforme tout G-motif en un G"-motif. Voyons maintenant comment transformer tout G"-motif en un motif trivial ou G"' -motif.

V.4.5. TRANSFORMATION D'UN G"-MOTIF M" EN UN G"' -MOTIF Mo.

La différence entre un G"-motif et un G"' -motif est faible. Elle réside en deux points :

1°) Dans un G"-motif, l'origine d'un arc n-aire (n>2) est exprimée sous la forme de <Motint$^{\times}$>, chaque Motint pouvant être soit une simple valeur soit un véritable motif (de la forme Motifcro FinCG). Dans un G"' -motif au contraire, l'origine ne peut être constituée que de valeurs simples ; il n'y a plus que deux niveaux de crochets : le crochet extérieur et les crochets entourant les origines multiples d'arcs n-aires ; tous les Motint sont au niveau 1 ; nous les appelons ici "Elem" en raison du deuxième point de différence, que voici.

2°) Dans un G"-motif, deux ou plusieurs arcs peuvent être enchaînés, soit avec des valeurs intermédiaires (cas de FinCG : Relacompo Val FinCG) soit sans (cas de Relacompo). La transformation de M" en Mo va contenir un découpage de ces chaînes, avec création éventuelle des valeurs intermédiaires omises, sous la forme d' "inconnues locales".

Pour décrire cette transformation simple d'un G"-motif en G"' -motif, nous allons une fois encore raisonner sur les ramifications et, à chaque noeud et pour cela à chaque règle de G", associer une transformation élémentaire. Comme chaque règle de G" est un cas particulier de règle de G, nous pouvons lui associer une règle de définition de la signification du motif. C'est de cette règle que nous allons déduire la règle de transformation qui donc, par construction même, conservera la signification.

Nous emploierons les notations suivantes :

- C : ensemble des "Elem" qui constituent le G"' -motif. Celui-ci sera donc : < Ensemble C écrit sous forme de suite>. Pour chaque règle, nous écrirons la contribution propre de cette règle à l'ensemble C.

- Pour la règle de membre gauche A, nous appellerons F(A) ou FF(A) la fin de la sous-phrase dérivée de A qui sera utilisée pour former des "Elem" lors du traitement d'un prédécesseur (dans la G"-ramification) de A : F si cette fin comporte seulement l'origine d'un arc suivant éventuel, FF si elle comporte l'origine et le nom de relation.

  De même, D(A) et DD(A) désigneront le début de la sous-phrase dérivée de A pouvant servir de fin à un "Elem" produit par un prédécesseur de A : D si cette fin est le but de l'arc, DD si elle est ce but précédé du nom de relation.

<u>Exemple</u> : considérons le DébutCG : aRSbT.

$C = \{a\ R\ y_1\ ,\ y_1\ S\ b\}$
$D = a$
$FF = bT$

Le tableau suivant donne pour chaque règle de G" la transformation élémentaire correspondante avec, dans les cas intéressants, la règle de définition de la signification, dont elle est issue. Lorsque D(A) ou DD(A) (respectivement F(A) ou FF(A)) sera rigoureusement celui du premier (respectivement dernier) constituant de A, on ne le donnera pas dans ce tableau.

Règles de G"		Cas particulier de N°	Règles-de définition de la signification (éventuellement) -de transformation
N°	Texte		
51	Motint:Val	1 et 19	F(Motint)=Val
52	Motint:Val FinCG	1 et 20	- S(Motint)=Ef(MotifCG)=S_{Val}(FinCG) - C:=Cu{Val DD(FinCG)}
53	Motint: Motifcro	4	Rien
54	Motint: Motifcro FinCG	2	- S(Motint)=E(Motifcro), $S_{f(Motifcro)}$(Fin) - C:=Cu{F(Motifcro) DD(FinCG)}
55	Motifcro: <Motint$^{\times}$>	5	- S(Motifcro)=E(Motint1),E(Motint2), ...,<f(Motint1), f(Motint2),...> - Aucun nouveau terme dans C - F(Motifcro)=si plusieurs Motint alors <F(Motint)$^{\times}$> sinon F(Motint)

56	FinCG: Relacompo Val	12 et 19	- S(FinCG)=S_{Val}(Relacompo),Val - C:=Cu{FF$^{(1)}$(Relacompo) Val} -DD(FinCG)=Si Relacompo est Rela et est ≠ Z alors Rela Val sinon DD$^{(1)}$(Relacompo)$^{(2)}$ -F(FinCG)=Val
57	FinCG1: Relacompo Val FinCG2	12 et 20	- S(FinCG)=S_{Val}(Relacompo), Ef(Val FinCG2) - C:=Cu{FF$^{(1)}$(Relacompo) Val, Val DD$^{(1)}$(FinCG2)} -DD(FinCG1)=Si Relacompo est Rela et est ≠ Z alors Rela Val sinon DD(Relacompo)$^{(2)}$ -F(FinCG1)=F(FinCG2)
59	Relacompo: Rela	23	- S(Relacompo)=S(Rela)=(-,Rela,-) - Aucun nouveau terme dans C - DD(Relacompo) et FF(Relacompo) n'existent pas

(1) A condition que ces termes existent.

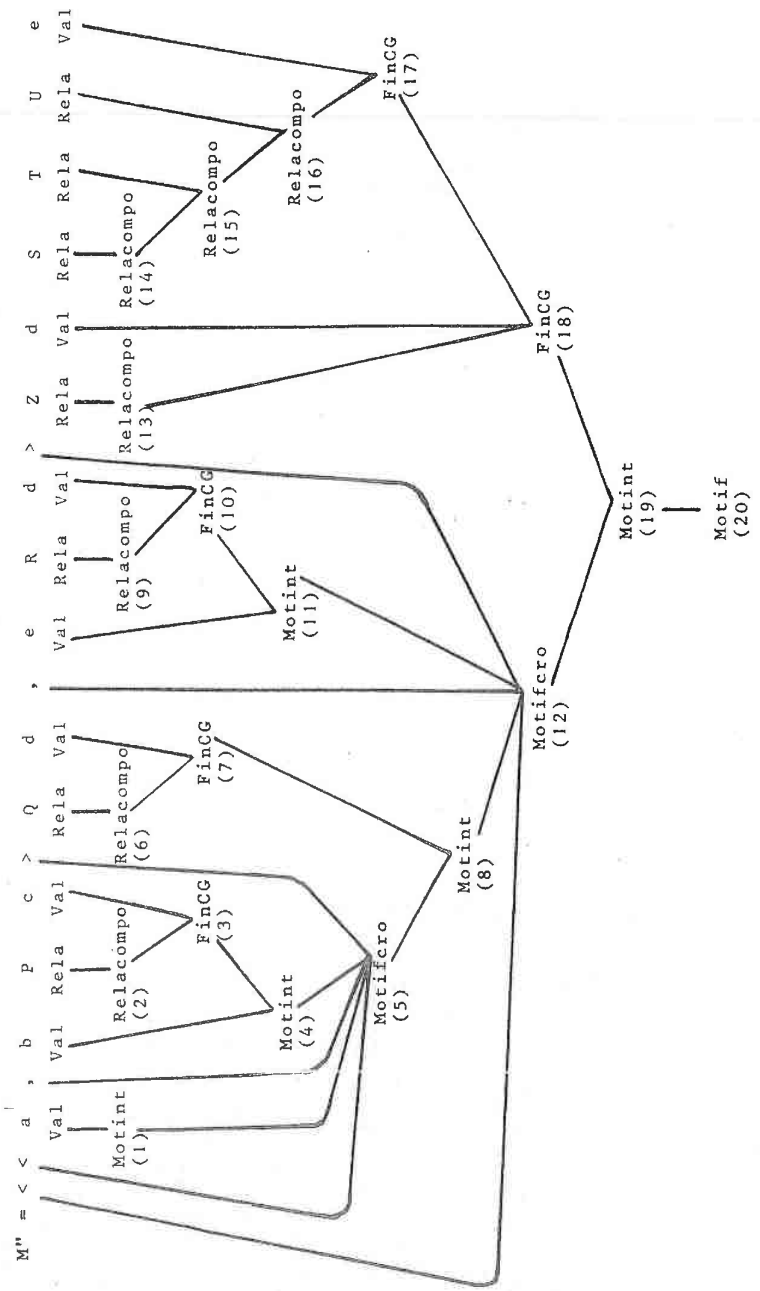
(2) Si Relacompo = Z alors DD(FinCG), comme DD(Relacompo), n'existe pas.

60	Relacompo: Relacompo1 Rela	24	- S(Relacompo)=$S^{y_j}$$^{(1)}$(Relacompo1), ($y_j$,Rela,-) - C:=Cu{FF(Relacompo1) y_j} - DD(Relacompo)=Si Relacompo1 est Rela' alors Rela' y_j sinon DD(Relacompo1) FF(Relacompo)=y_j Rela
50	Motif:Motint	0	- E(Motif)=E(Motint) - Mo = <Suite des éléments de C>

Exemple :

Soit le G"-motif M" suivant. Effectuons sa transformation en G"' -motif (Mo). Voici la G"-ramification résultat de l'analyse de M". Nous y avons numéroté les noeuds.

(1) Inconnue locale, jamais utilisée encore pour ce motif.



Voyons comment, étape par étape, s'effectue la transformation. Au début, C = ∅.

Noeud	Règle	Evolution éventuelle de C, F, FF, D, DD
1	51	F(1) = a
2	59	Rien
3	56	C est toujours vide car FF(2) n'existe pas DD(3) = Pc F(3) = c
4	52	C = {<u>bPc</u>}
5	55	F(5) = <a,c>
6	59	Rien
7	56	C ne change pas DD(7) = Qd F(7) = d
8	54	C = Cu{<u><a,c>Qd</u>}
9	59	Rien
10	56	C ne change pas DD(10) = Rd F(10) = d
11	52	C = Cu{<u>eRd</u>}
12	55	F(12) = <d,d>

13	59	Rien
14	59	Rien
15	60	C ne change pas car FF(14) n'existe pas DD(15) = S y_1 FF(15) = y_1 T
16	60	C = Cu{y_1 T y_2} DD(16) = S y_1 FF(16) = y_2 U
17	56	C = Cu{y_2 U e} DD(17) = S y_1 F(17) = e
18	57	C =Cu{d S y_1} DD(18) n'existe pas (à cause de Z) F(18) = e
19	54	C ne change pas car DD(18) n'existe pas
20	50	Mo = <bPc,<a,c>Qd,eRd,y_1Ty_2,y_2Ue,dSy_1>

V.4.6. CONCLUSIONS.

Etant donné un pochoir on peut trouver immédiatement des motifs qui l'expriment, les motifs triviaux. Par ailleurs, on sait, par des transformations qui sont inversi-

bles, étant donné un motif quelconque, trouver un motif trivial correspondant au même pochoir. On sait donc trouver, théoriquement du moins, tous les motifs qui traduisent un certain pochoir, quel qu'il soit. On peut donc dire, en conclusion des paragraphes V.3 et V.4, que la donnée d'un pochoir et la donnée d'un motif sont équivalentes.

Remarque 1 :

Dans un pochoir, nous avons supposé que tous les sommets ont un nom : on attribue aux sommets non nommés initialement des noms d' "inconnues locales". Or dans un motif il peut y avoir des sommets non nommés (Relacompo). Nous considérons que c'est là une simple question d'écriture, vu que les inconnues locales sont arbitraires.

Remarque 2 :

Nous ne donnons pas ici de procédé pratique d'écriture des motifs pour ne pas alourdir la rédaction. Ce n'est pas grave puisqu'on sait toujours écrire au moins un motif correspondant à un pochoir donné, son motif trivial.

Cependant, si écrire un motif non trivial présente un intérêt théorique faible, écrire un motif "compact" permet quand même d'une part d'éviter des répétitions inutiles et d'autre part d'écrire des demandes plus imagées, plus "parlantes". Ce point, assez important lors des utilisations classiques (nous ne parlons pas, ici, du cas où PIVOINES est utilisé comme langage pivot), le deviendrait encore plus lors d'utilisations conversationnelles ; or nous pensons que le

langage des motifs et PIVOINES en général se prêtent fort bien à cet emploi.

Nous avons donné au paragraphe V.1.2 de nombreux exemples qui doivent suffir à guider l'utilisateur. Mais, lorsque PIVOINES deviendra opérationnel, nous rédigerons un manuel d'utilisation.

V.5. LES FILTRES.

Nous avons brièvement présenté les filtres au paragraphe IV.1.5 et avons expliqué pourquoi et comment existe une certaine redondance entre les possibilités d'expression au niveau des filtres et au niveau des motifs. Nous allons préciser ici l'écriture des filtres et leur signification. Nous terminerons le paragraphe V.5 par un retour sur la théorie logique et des possibilités d'extension des filtres.

V.5.1. SYNTAXE DES FILTRES.

V.5.1.1. Grammaire.

Un filtre est construit avec pour matériaux :

- des motifs
- l'opérateur booléen et
- l'opérateur booléen ou
- des parenthèses (et).

Nous verrons plus loin (§ V.5.4) que l'on prévoit des extensions éventuelles, et en particulier l'utilisation de quantificateurs. Les négations peuvent déjà être exprimées au sein des motifs, dans les relations atomiques (cf. § V.1.1 Ensemble RELA des relations atomiques).

Un filtre revêt une forme d'expression booléenne (au sens des langages de programmation) mais nous avons hésité dans le choix de la priorité réciproque des opérateurs et et ou : ou bien, par cohérence avec les langages de programmation, désigner et comme prioritaire, ou bien, considérant qu'une question est principalement une conjonction de critères (chaque critère élémentaire au niveau du filtre étant un motif), désigner ou comme prioritaire. C'est ce que nous avons choisi.

Ainsi, la grammaire des filtres peut être écrite en complétant celle des motifs (§ V.2) par les règles de production suivantes :

Filtre : Facteur / Filtre et Facteur
Facteur : Terme / Facteur ou Terme
Terme : Motif / (Filtre)

Filtre est l'axiome de cette grammaire.

Exemple :

$F = aRx_1Sx_2TUb$ et cVx_2 ou x_2Wd et cPx_3Qx_1 est compris comme l'expression (qui est aussi un filtre correct) :

aRx_1Sx_2TUb et $(cVx_2$ ou $x_2Wd)$ et cPx_3Qx_1.

V.5.1.2. Contexte (et contexte total) à gauche d'un sous-filtre.

L'analyse d'un filtre F pour la grammaire précédente permet de le décomposer en sous-filtres et de définir le fils gauche, le fils droit et le connecteur d'un filtre. Plus préci-

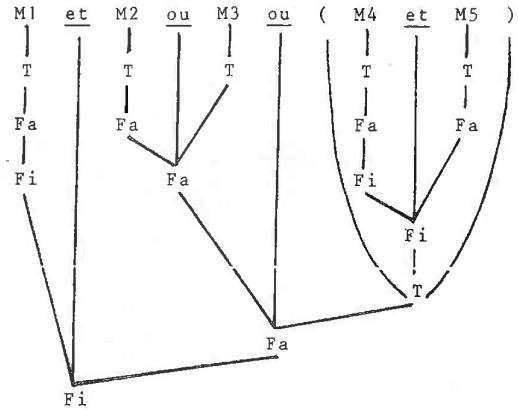
sément, la grammaire étant non ambiguë, ces notions peuvent être définies récursivement pour un filtre F par :

- si F est un motif, alors
  - . F est son unique sous-filtre ;
  - . F n'a ni fils ni connecteur.

- si F est (F'), alors
  - . les sous-filtres, fils et connecteur de F sont ceux de F'.

- si F est engendré par la règle Facteur : Facteur ou Terme (respectivement : Filtre : Filtre et Facteur), cette règle décompose F sous la forme F = F' ou F" (respectivement : F' et F") ; alors
  - . les sous-filtres de F sont ceux de F' et ceux de F" ainsi que F lui-même ;
  - . le fils gauche de F est F' ;
  - . le fils droit de F est F" ;
  - . le connecteur de F est ou (respectivement : et).

Exemple : F = M1 et M2 ou M3 ou (M4 et M5)



M2 ou M3 ou (M4 et M5) est un sous-filtre de F. Ses propres sous-filtres sont M2, M3, M2 ou M3, M4, M5, M4 et M5, (M4 et M5) et lui-même. Son connecteur est ou et ses fils gauche et droit sont respectivement M2 ou M3 et (M4 et M5).

Contexte à gauche d'un sous-filtre F' de F.

Cherchons le plus petit sous-filtre f de F, de connecteur et, tel que F' soit sous-filtre du fils droit de f.

Par définition, nous dirons que si f existe le contexte à gauche de F', CG(F'), est le fils gauche de f, et sinon CG(F') = ∧ (mot vide).

Dans l'exemple ci-dessus :

le contexte à gauche de :	est :
M2 ou M3 ou (M4 et M5)	M1
M2, M3, M2 ou M3, M4, (M4 et M5), M4 et M5	M1
M1	∧
M5	M4
F	∧

Contexte total à gauche d'un sous-filtre F' de F.

Nous avons défini le contexte à gauche de F' car son résultat sera caractéristique de l'état de l'automate "évaluateur de filtre" au moment de la prise en compte de F'.

Le contexte total à gauche de F', CTG(F'), rend compte de l' "historique" de cet état. Nous le définissons ainsi :

CTG(F') = si CG(F') = ∧ alors ∧
sinon si CTG(CG(F')) = ∧ alors CG(F')
sinon CTG(CG(F')) et CG(F').

Dans l'exemple, tous les CTG sont égaux au CG sauf celui de M5 qui est :

CTG(M5) = CTG(M4) et M4 = M1 et M4.

V.5.1.3. Inconnues, inconnues propres.

- Inconnues d'un filtre.

Les inconnues d'un filtre sont les inconnues globales (x_1, ..., x_n) qui figurent dans ce filtre (cf. § V.3.7.3). Lors de l'écriture d'un filtre, les noms des inconnues sont utilisés au fur et à mesure des besoins, en commençant par x_1 et de façon consécutive. L'inconnue x_0 peut être utilisée pour des besoins syntaxiques (cf. § V.3.7.3); elle ne sera pas condisérée comme une inconnue du filtre mais comme une inconnue locale. Le symbole US, lui, est considéré comme une inconnue (globale) du filtre, la dernière. Ainsi, un filtre contenant les inconnues x_1, x_2 et US est considéré comme ayant trois inconnues.

Les inconnues sont les seuls liens de dépendance entre les différents motifs d'un filtre : d'un motif à l'autre du même filtre, le même nom d'inconnue désigne la même inconnue. Nous pouvons dire que la portée de chaque inconnue est le filtre entier. x_0 a pour seule particularité

d'avoir comme portée chaque motif où elle figure : elle est locale à chacun d'eux.

- Inconnues propres d'un motif.

Les inconnues propres d'un motif M sont les inconnues globales qui figurent effectivement dans M mais pas dans son contexte total à gauche : lors de l'évaluation de F, c'est au cours de l'évaluation de M que ces inconnues seront considérées pour la première fois. Ainsi, dans l'exemple du paragraphe V.5.1.1, les inconnues propres du premier motif sont x_1 et x_2, les deux motifs suivants n'en ont pas, celle du dernier est x_3. Il se peut cependant que la notion d'inconnue propre soit un peu moins simple qu'il n'y paraît ici. C'est ce qui se produit avec les sous-filtres "hétéro-inconnue", que nous présentons au paragraphe V.5.2.

V.5.2. SEMANTIQUE DES FILTRES. DEFINITION DES RESULTATS D'UN FILTRE F.

Un filtre est l'expression d'une demande d'accès. Pour décrire sa sémantique, nous allons préciser quelles sont les réponses souhaitées à cette demande, en fonction de l'information I interrogée.

Au sein d'un filtre, les motifs, qui sont des atomes de l'expression booléenne, doivent avant tout être considérés comme des booléens. Pour pousser plus avant la comparaison avec les langages de programmation, disons que les motifs seront considérés comme des fonctions à résultat booléen ayant comme effet annexe l'évaluation des inconnues.

Cette définition est cohérente avec la définition des résultats d'un motif donnée au paragraphe V.3.7.3. Ainsi la recherche correspondant à un filtre F donnera naissance à deux résultats :

- RESU(F), qui est un ensemble de n-uples de valeurs pour les n inconnues globales de F ;
- BOOL(F), qui est un booléen prenant la valeur vrai si et seulement si la recherche est couronnée de succès.

Précisons maintenant en quoi consistent ces deux résultats et donnons-en une définition.

. Par définition, BOOL(F) est la valeur de l'expression booléenne de F, chaque motif Mi étant, rappelons-le, considéré comme une fonction à résultat booléen (appelons BOOL(Mi) ce résultat booléen).

. De RESU(F), donnons d'abord une définition intuitive : c'est l'ensemble des n-uples de valeurs des inconnues qui font que le résultat booléen soit vrai. Nous dirons que ce résultat est l'ensemble des n-uples de valeurs des inconnues pour lesquels "le filtre F a une occurrence dans I". Donnons maintenant une définition plus rigoureuse.

Considérons l'expression booléenne du filtre et interprétons-y chaque motif Mi comme l'ensemble RESU(Mi) et les opérateurs et et ou respectivement comme les opérateurs ensemblistes $\cap$ et $\cup$ (l'opérateur $\cup$ étant prioritaire par rapport à $\cap$). Ce que nous obtenons est par définition RESU(F).

Exemple : pour l'exemple du paragraphe V.5.1.1 :

$$F = \underbrace{aRx_1Sx_2TUb}_{M1} \ \underline{et} \ \underbrace{cVx_2}_{M2} \ \underline{ou} \ \underbrace{x_2Wd}_{M3} \ \underline{et} \ \underbrace{cPx_3Qx_1}_{M4}$$

RESU(F) = RESU(M1) ∩ RESU(M2) ∪ RESU(M3) ∩ RESU(M4)

qui se lit :

RESU(M1) ∩ (RESU(M2) ∪ RESU(M3)) ∩ RESU(M4).

. La définition de RESU(Mi) donnée dans le cas d'un motif seul au paragraphe V.3.7.3 doit être ici un peu modifiée : si Mi appartient au filtre F ayant n inconnues et contient lui-même p inconnues, le résultat défini en V.3.7.3 est un ensemble de p-uples ; RESU(Mi) sera l'ensemble de n-uples qui en est déduit en donnant aux n-p inconnues absentes dans Mi toutes les valeurs possibles. Pratiquement, nous noterons RESU(Mi) en donnant à ces n-p inconnues la valeur "non déterminé",que nous représenterons par un tiret. Les opérateurs ensemblistes seront interprétés en conséquence.

Si un motif Mi n'a pas d'inconnues, nous poserons que RESU(Mi) est un singleton de n-uple dont tous les constituants sont "non déterminés" : RESU(Mi) = {(-,-,...)}.

Sous-filtres "hétéro-inconnue".

Nous avions, au paragraphe V.5.1.3, annoncé un cas où la notion d'inconnue propre d'un motif n'est pas simple. Illustrons ce cas par l'exemple suivant.

Exemple : F = (M1 ou M2) et M3

Supposons que M1 contienne les inconnues x_1 et x_2 et que M2 contienne les inconnues x_1 et x_3. Si M3 contient les inconnues x_1 et x_4, il est clair que x_4 est une inconnue propre mais pas x_1. Si maintenant M3 contient l'inconnue x_2, la définition des inconnues propres donnée dans le paragraphe V.5.1.3 stipule que x_2 n'est pas une de ses inconnues propres. Mais si,lors d'une recherche, on ne trouve pas d'occurrence de M1 mais qu'on en trouve de M2, x_2 n'aura acquis aucune valeur lorsque commencera l'évaluation de M3. C'est donc au cours de la recherche correspondant à M3 que l'on attribuera des valeurs à x_2 pour la première fois.

Ce cas peut être gênant lors de la recherche d'une stratégie (§ VII - 4 - 4) ou pour la gestion des résultats de la recherche (§ VIII-1-4).

Définition :

Nous dirons qu'un sous-filtre F' est hétéro-inconnue si son résultat est susceptible de comporter deux n-uples au moins n'ayant pas les mêmes positions "non déterminées".

Exemple :

Dans l'exemple ci-dessus, M1 ou M2 est hétéro-inconnue même si le motif M1 n'a jamais d'occurrence.

Contre-exemple :

F = M4 et (M1 ou M2) et M3

M4 contient les inconnues x_2 et x_3
M1 contient les inconnues x_1 et x_2

M2 contient les inconnues x_1 et x_3

(M1 ou M2), dans le contexte de F, n'est pas un sous-filtre hétéro-inconnue car son résultat ne comportera que des p-uples dans lesquels seules seront occupées, mais toujours, les positions de x_1, x_2, x_3.

Exemple du mécanisme des opérateurs ensemblistes.

A cause de la présence de la valeur "non déterminée", les opérateurs ensemblistes ont une signification que nous précisons par l'exemple suivant.

E1 = {(4,7,-),(4,2,-),(2,3,-)}

E2 = {(3,-,-),(4,-,-)}

E3 = {(4,-,6),(4,-,9)}

E4 = {(-,-,-)}

E1 ∪ E2 = {(2,3,-),(3,-,-),(4,-,-)}

E1 ∩ E2 = {(4,7,-),(4,2,-)}

E1 ∪ E3 = {(4,7,-),(4,2,-),(2,3,-),(4,-,6),(4,-,9)}

E1 ∩ E3 = {(4,7,6),(4,7,9),(4,2,6),(4,2,9)}

E1 ∪ E4 = E4

E1 ∩ E4 = E1

. Cas de US . lorsqu'un filtre F contient le symbole US, chaque occurrence du filtre doit être trouvée à l'intérieur d'une même unité de sélection. US désigne le représentant de cette unité de sélection.

V.5.3. PREMIER COUP D'OEIL SUR L'EVALUATION D'UN FILTRE.

C'est au chapitre VIII que nous traiterons de l'évaluation d'un filtre puisque nous parlerons du programme objet d'une demande d'accès. Mais signalons dès maintenant quelques traits marquants de cette évaluation.

L'évaluation ne se fait pas comme le laisserait supposer la définition de RESU(F), c'est-à-dire en évaluant les motifs indépendamment les uns des autres. Il nous semble bien préférable, par raison d'économie et pour pouvoir bénéficier de l'aptitude qu'on aura de "bien" écrire les filtres (en commençant par les motifs les plus "sélectifs" par exemple), d'évaluer le filtre de gauche à droite en évaluant chaque motif Mi compte tenu de sa "source", SOURCE(Mi), qui est le résultat de son contexte total à gauche. Dans l'exemple du paragraphe V.5.1.1, il est inutile de chercher toutes les solutions de M4 : seules seront recherchées celles pour lesquelles x_1 a une valeur apparaissant dans le résultat du sous-filtre M1 et M2 ou M3 qui est son contexte total à gauche. Mais alors les valeurs de x_3 trouvées seront associées non seulement à des valeurs de x_1 mais aussi à des valeurs de x_2.

Ainsi, pour chaque motif Mi, ce n'est pas RESU(Mi) que l'on évaluera mais RESU(Mi et CTG(Mi)) que nous appellerons CRESU(Mi). La formule de définition de RESU(F) reste valable si l'on remplace les RESU(Mi) par les CRESU(Mi) mais l'opération d'intersection aura un effet nul.

Remarque : nous avons posé SOURCE(Mi) = RESU(CTG(Mi)).
Mais on peut aussi écrire que SOURCE(Mi) est CRESU(CG(Mi)).

Conséquence de l'évaluation de gauche à droite sur la gestion des inconnues et de US.

Pour chaque motif Mi, ses inconnues propres jouent véritablement un rôle d'inconnues mais ses autres inconnues jouent le rôle de données puisqu'elles ont déjà reçu des valeurs lors de l'évaluation de SOURCE(Mi). L'évaluation de Mi aura un rôle de sélection sur ces valeurs en plus du rôle d'évaluation des inconnues propres.

Le symbole US qui figurera quelquefois dans un filtre a de nombreux points communs avec une inconnue. Lors de la première recherche élémentaire où il figurera il acquerra une ou des valeurs, les autres recherches élémentaires utilisant (et éventuellement sélectionnant) ces valeurs.

Mais une différence entre US et une inconnue réside dans le fait que la présence de US influe sur toute l'évaluation du filtre. Ceci nous a poussée à décider que son utilisation serait sérielle et non globale (cf. IV.3), en ce sens que la recherche des occurrences du filtre dans sa totalité s'effectuerait d'abord pour la première information composante, puis pour la deuxième, ... et non pour toutes à la fois.

Une autre différence est que US a une signification par lui-même : ses valeurs successives lui seront données de l'extérieur et non évaluées grâce au filtre ; le mécanisme qui fera acquérir à US ces différentes valeurs successives sera spécifié dans la partie "cadrage". La première recherche élémentaire où figurera US actionnera ce mécanisme. Par exemple, celui-ci pourra comporter l'entrée en mémoire centrale de l'information composante à interroger.

A cause de son influence s'étendant à tout le filtre, nous imposerons que US, s'il figure dans un filtre, figure au moins dans son premier motif, ce qui est très naturel. La première étape de l'évaluation de ce motif sera la mise en oeuvre du mécanisme d'affectation d'une valeur à US.

Conclusion :

La définition du résultat d'un filtre F montre que ce résultat est indépendant de l'ordre des motifs dans F ; plus précisément, on peut inverser l'ordre des termes d'une conjonction ou d'une disjonction sans changer le résultat de F. Par contre, le mode d'obtention de ces résultats change.

L'utilisateur peut donc choisir l'ordre des motifs du filtre, ceci de façon à obtenir une exploitation la meilleure possible selon les critères qu'il juge prépondérants (cf. § IV.1.5 et VII 4.4).

210

V.5.4. REMARQUE : COMPARAISON DE PIVOINES ET DU CALCUL DU PREMIER ORDRE. EXTENSIONS ENVISAGEES.

Considérons l'ensemble des formules de logique du premier ordre construites à partir des relations de la structure et de leurs itérées (cf. par exemple [32]). Une donnée est un domaine d'interprétation pour ces formules. Une formule close, c'est-à-dire dans laquelle il n'y a pas de variable libre, peut ainsi prendre la valeur Vrai ou Faux pour une donnée.

Pour tout filtre F, il existe une formule close f telle que : f prend la valeur Vrai pour une donnée D si et seulement si le résultat booléen de F pour D est Vrai. Cette formule s'obtient en gros en remplaçant dans le filtre F chaque motif par sa signification (et en introduisant et entre les différents arcs), en faisant précéder F de $(\exists x_1)$ $(\exists x_2)$ ... $(\exists x_n)$ si F a n inconnues et de $(\exists u)$ pour chaque inconnue locale (on suppose que l'on rend différents les x_o qui interviennent dans les différents motifs et que deux motifs n'utilisent pas les mêmes y_j).

Exemple :

Pour $F = aRSx_o<Lx_1,Mx_2>$ et $x_2Tx_o<U,V>b$, la formule est :

$$(\exists x_1)\ (\exists x_2)\ (\exists x_o^1)\ (\exists x_o^2)\ (\exists y_1)\ (aRy_1 \text{ et } y_1Sx_o^1 \text{ et } x_o^1Lx_1 \text{ et } x_o^1Mx_2 \text{ et } x_2Tx_o^2 \text{ et } x_o^2Ub \text{ et } x_o^2Vb.$$

La réciproque est fausse : une formule close f quelconque étant donnée, on ne peut pas toujours trouver un filtre F qui lui corresponde, à cause de la présence possible dans f de quantificateurs universels. Autrement dit, PIVOINES est moins puissant que la logique du premier ordre. Cependant cette conclusion n'est valable que parce qu'on a placé dans les relations utilisées dans les formules du premier ordre, non seulement les relations R de la structure considérée, mais aussi leurs itérées R^*. Si l'on se limitait aux relations de la structure, on ne pourrait associer une formule close à tout filtre. L'opération * est du second ordre. Finalement, les puissances de PIVOINES et de la logique du premier ordre (sur les mêmes relations de base) ne sont pas comparables.

Pour accroître la puissance de PIVOINES, on peut envisager une extension consistant à introduire le quantificateur universel. Mais cela pose de nombreux problèmes. Considérons un motif dans lequel on autoriserait ce quantificateur. Supposons que l'on écrive ce motif de façon prénexe, c'est-à-dire en groupant les quantificateurs (le quantificateur $\exists$ ne sera plus implicite) au début ; le motif commencerait donc par une succession de termes du type $\exists x_j$ ou $\forall x_k$. La prise en compte de ce motif serait extrêmement alourdie par la présence de ces "préfixes". Elle le serait d'autant plus que nous avons choisi une exécution globale (cf. § IV.3) du programme objet indéterministe ; or cette optique globale est peu compatible avec la prise en compte des $\forall$.

Exemple simple :

$$M = (\exists x_1)\ (\forall x_2)\ aRx_1Sx_2Tb$$

Nous supposons que l'ensemble X2 des valeurs de x_2 aura été défini auparavant.

Il y a a priori deux façons de prendre en compte ce motif :

- ou, pour chaque valeur de x_1, chercher si toutes les valeurs de x_2 conviennent ; mais il ne s'agit plus d'une gestion globale ;

- ou chercher tous les couples (u_1,u_2) de valeurs de x_1 et x_2 puis chercher l'ensemble des valeurs u_1 telles qu'il existe un couple (u_1,u_2) pour tout $u_2 \in X2$; il s'agit d'une gestion globale mais qui risque d'être extrêmement longue (car on ne s'aperçoit d'un échec éventuel qu'à la fin).

<u>Exemple moins simple</u> :

$$M = (\forall x_1)\ (\exists x_2)\ (\forall x_3)\ (\exists x_4)\ \underbrace{a\ R\ \ldots\ldots\ldots\ldots}_{P(x_1,x_2,x_3,x_4)}$$

Là encore, on peut envisager le cas où la gestion n'est pas globale et celui où elle l'est. Dans ce dernier cas, la vérification à effectuer à la fin de la recherche, d'une part est compliquée, d'autre part risque d'avoir été précédée d'une bien longue recherche.

Or nous ne pouvons renoncer à la gestion globale car elle est la seule compatible avec une recherche de bonne stratégie (cf. § IV-3), et nous ne voulons pas pénaliser les cas fréquents. Une solution serait de gérer globalement les inconnues précédées de $\exists$ et sériellement celles précédées de $\forall$, mais l'ensemble risque d'être compliqué, d'autant plus que certaines inconnues seraient liées à d'autres (cf. § VIII-1-2-2).

La solution serait peut-être d'autoriser l'emploi de $\forall$ dans des cas bien limités, pour répondre aux besoins les plus courants, comme par exemple : chercher les entreprises qui fabriquent <u>toutes</u> les pièces demandées par tel client. La limitation des cas porterait d'une part sur le nombre et l'ordre des quantificateurs $\forall$ et d'autre part sur le fait que les inconnues figurant dans le motif considéré seraient des inconnues propres ou non.

Une autre solution enfin serait d'utiliser le langage de cadrage qui pourrait assurer (par des bouclages) les fonctions des quantificateurs depuis l'extérieur des filtres.

VI. REPRESENTATION D' UNE INFORMATION.

. STRUCTURE PHYLOG .

oOOo

VI.1. INDEPENDANCE.

L'utilisateur qui interroge une information connaît la structure logique de cette information, et c'est en ses termes qu'il formule les questions : les filtres sont écrits à base d'éléments de la structure logique.

Mais lorsque le filtre a été transformé en un programme d'exploitation, c'est à la représentation de l'information en mémoire que ce programme s'adresse.

Or il est très important qu'existe une certaine indépendance entre structure logique et représentation interne d'une information :

- il faut qu'à structure logique constante on puisse choisir et modifier la représentation interne d'une information, et en particulier changer d'ordinateur ou tenir compte de l'utilisation pour améliorer l'efficacité, sans que cela entraîne l'écriture de lourds programmes ; il faut de plus qu'on puisse considérer deux informations de représentations différentes comme ayant même structure logique et les interroger ensemble ;

- il faut qu'à une certaine information enregistrée en mémoire on puisse faire correspondre diverses structures logiques selon par exemple les utilisateurs qui y auront accès (cf. la notion de sous-schéma de Codasyl [10, 11, 12]).

Le programme de traduction des filtres en programmes doit être écrit une fois pour toutes, indépendamment de la structure logique et de la représentation interne. Pour que cela soit possible, il est nécessaire que ce programme dispose d'une description de la correspondance entre structure logique et représentation interne. Etudions cette correspondance et tout d'abord la représentation d'une structure de données dans une autre.

VI.2. REPRESENTATION D'UNE STRUCTURE DE DONNEES DANS UNE AUTRE.

La représentation d'une structure de données S dans une autre S' associe à chaque symbole relationnel ℓ de S une combinaison, par des opérations autorisées, de symboles relationnels de S' ; nous appelons cette combinaison la représentation (dans S') de ℓ. De plus, les éléments de V et les "repères privilégiés" (cf. § II.5.1) peuvent avoir une représentation dans S', sous la forme respectivement d'un élément du vocabulaire de S', et d'un repère privilégié ou d'un accès de S'.

On déduit de cette association la représentation dans S' de n'importe quelle combinaison de symboles de S (par des opérations autorisées). Lorsqu'on spécifie une représentation de S dans S', il peut cependant être utile de spécifier directement la représentation de certaines combinaisons de symboles de S (nous les appellerons "groupements traduisibles" ou "groupements" au paragraphe VII.4.2) . On peut ainsi tenir compte de simplifications dues, soit aux propriétés des opérations sur les relations, soit aux axiomes spécifiques de la structure S'.

EXEMPLE VI.2.1.

Dans l'exemple des dossiers médicaux (§ III.2-2), la relation D_1 est traduite par $A'B'_2$, la relation D_2 est traduite par ${}^1B'_2{}^{-1}B'_1$, la relation D_1 D_2 est traduite par $A'B'_1$.

EXEMPLE VI.2.2. Représentation inversée.

Soit un ensemble D de documents dans lequel chaque document comporte seulement un numéro d'identification et un ensemble de descripteurs. On considère une représentation inversée de D. Appelons S' la structure de cette représentation.

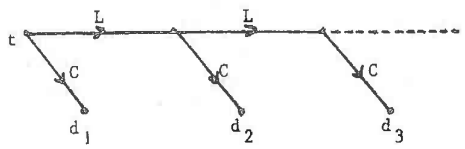
Dans la structure logique, une relation C relie chaque repère représentant de document à tous les descripteurs que ce document contient.

Dans la structure S', une relation E relie au contraire chaque descripteur à tous les repères des documents qui le contiennent.

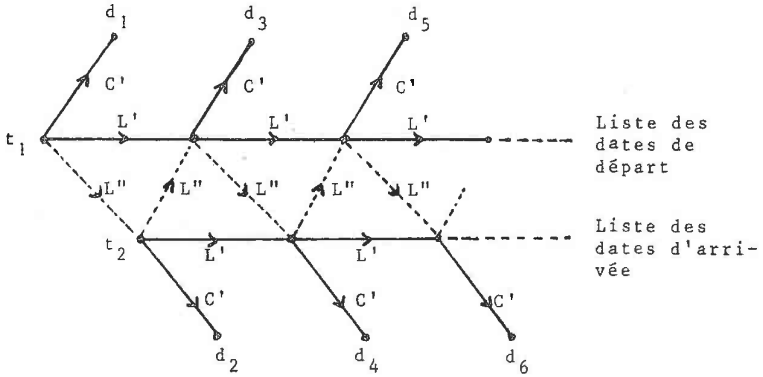
La représentation de C est E^{-1}. Mais on précisera aussi que la représentation de C^{-1} est E.

EXEMPLE VI.2.3.

S est une structure de liste sur des dates.



Mais il se trouve qu'une date sur deux est une date de départ et l'autre une date d'arrivée. Et l'on veut représenter la structure S dans une autre structure S' comportant : une liste des dates de départ, une liste des dates d'arrivée, une relation de succession L" reliant les deux listes et reproduisant L.



Nous avons supposé ici que les deux listes ont même fonction de succession L' et même fonction contenu C'.

L sera représenté par L".
Mais L^2 sera représenté par L'.

VI.3. REPRESENTATION EN MEMOIRE PAR L'INTERMEDIAIRE DE LA STRUCTURE PHYLOG.

VI.3.1. STRUCTURE INTERMEDIAIRE ET PROCEDURES.

La représentation d'une structure S dans une certaine mémoire pourrait être considérée comme la représentation de S dans la structure de mémoire M (cf. Exemples II.5.1.4 et II.2.4) ; cette représentation serait un cas particulier de ce qui vient d'être dit au paragraphe VI.2. Mais pour les besoins de l'interrogation, il serait ensuite nécessaire de faire correspondre à chaque relation de la structure de mémoire une procédure de recherche. En effet, la simple évocation des relations de la structure de mémoire d'un ordinateur n'est pas compréhensible par celui-ci.

Nous avons jugé plus commode de placer à un autre niveau la structure intermédiaire entre la structure donnée S et les procédures de recherche. C'est ainsi que nous avons introduit la structure PHYLOG, déjà évoquée au paragraphe IV.2.2.2 : c'est la structure logique de l'organisation physique interne des données.

La structure PHYLOG sera choisie pour être :

- suffisamment proche de la structure de mémoire pour qu'à chaque relation de PHYLOG corresponde une procédure de recherche simple à programmer - nous l'appellerons : procédure de recherche élémentaire - ;

- suffisamment éloignée de la structure de mémoire pour ne pas dépendre des détails d'implémentation et ne refléter vraiment que l'organisation des données en mémoire. Il se peut d'ailleurs que PHYLOG soit identique à la structure logique : c'est le cas où la représentation ne "déforme" pas la structure.

Exemple : reprenons l'exemple VI.2.3.

Supposons que l'on veuille représenter en mémoire la structure S sous la forme suivante : une liste de dates de départ rangées de façon consécutive, une liste de dates d'arrivée rangées de la même manière, chaque date étant accompagnée d'un pointeur vers son successeur par L.

C'est la structure S', que nous choisissons comme structure PHYLOG et à L' et L" nous faisons correspondre une procédure de recherche (liée à la contiguïté pour L' et au chaînage pour L").

La traduction d'un filtre en un programme de recherche (indéterministe) s'effectue en ignorant tous les détails d'implémentation : seules lui sont connues la structure logique et PHYLOG, ou plus exactement la correspondance entre les deux.

VI.3.2. CORRESPONDANCE ENTRE PHYLOG ET LA REPRESENTATION EN MEMOIRE.

Nous étudierons aux chapitres suivants comment la spécification de la représentation est utilisée :

- le programme "groupement traduction choix" traduit les relations ou groupements traduisibles (ou plus exactement les arcs de la signification des motifs, ou des groupements de ces arcs) en des relations de PHYLOG ou combinaisons de celles-ci (ou plus exactement en arcs ou groupements d'arcs de PHYLOG). En fait, il génère des appels de modules qui correspondent chacun à une relation de PHYLOG (modules de recherche élémentaires).

- lorsque le programme ainsi généré s'exécute, le texte des procédures qu'il appelle est, lui, l'expression de la représentation en mémoire.

On peut comparer ce processus à la programmation descendante ou modulaire [20, 30] : le programme "groupement traduction choix" ignore le contenu des procédures dont il génère des appels et est indépendant de celui-ci.

Remarque :

Le fait de spécifier en partie la correspondance entre structure logique et représentation en mémoire par des procédures de recherche est une solution commode pour les besoins de l'interrogation. Mais elle n'est pas tout à fait satisfaisante,à long terme, pour deux raisons principalement :

- la spécification de la représentation en mémoire devrait servir aussi pour les problèmes d'acquisition et de modification ; or dans l'état actuel des choses, il faudrait écrire pour chaque relation de PHYLOG des procédures pour ces usages. Il faudra donc essayer de trouver une spécification à double vocation mais cependant immédiatement ou automatiquement utilisable (par exemple une représen-

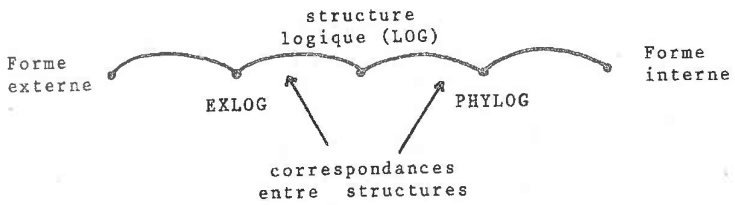
tation de PHYLOG dans la structure de mémoire et un outil permettant d'utiliser cette représentation pour l'acquisition et pour l'interrogation).

- dans un même ordre d'idées, nous désirons que la représentation en mémoire soit un élément "manipulable" d'un système de données, et même puisse dans une certaine mesure être conçue automatiquement (cf. I.1.2). Ce n'est pas le cas actuellement.

VI.4. PROBLEMES D'ACQUISITION.

Notre travail ne traite pas de ces problèmes que nous commençons à aborder cependant. Mais nous venons de dire que la façon dont nous spécifions les représentations n'est pas tout à fait satisfaisante. C'est dans un nouveau cadre élargi à l'acquisition et à la modification que nous pensons améliorer cette spécification. Nous désirons seulement ici donner une vue très schématique de ce cadre.

Il existe une certaine symétrie, par rapport à la structure logique, entre la représentation en mémoire et la forme d'acquisition. On peut en effet considérer qu'il existe une structure logique d'acquisition (appelons la EXLOG) - que l'on peut rapprocher de PHYLOG - , et une forme d'acquisition - que l'on peut rapprocher de la représentation en mémoire - . Ces différents éléments peuvent se visualiser ainsi :

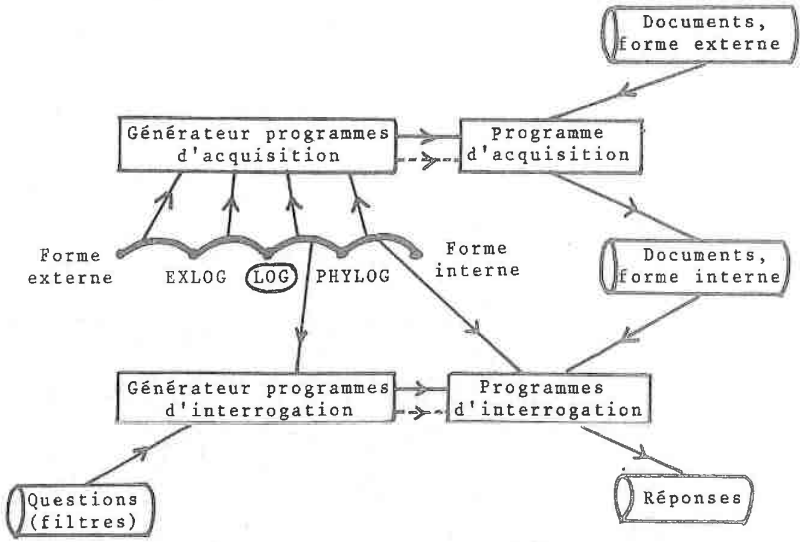


Le programme d'acquisition sera généré par un générateur de programmes qui utilisera les descriptions des correspondances exprimées par des arcs dans ce schéma, mais en sera lui-même indépendant.

Nous pensons qu'il serait même possible, grâce à la symétrie signalée plus haut, d'effectuer, par un procédé analogue à l'acquisition, des restructurations de données. Soit une donnée de structure logique LG, de structure PHYLOG PLG et ayant une certaine représentation interne RI ; on utiliserait le processus d'acquisition en prenant

- pour "forme externe" : RI
- pour structure EXLOG : PLG
- pour structure logique : la nouvelle structure logique LG'
- pour structure PHYLOG : la nouvelle structure PHYLOG : PLG'
- pour forme interne : la nouvelle forme interne: RI'.

Voici comment, très schématiquement, on peut concevoir l'utilisation des diverses descriptions de correspondances de structure pour l'acquisition et l'interrogation :



Les flèches pleines représentent des échanges d'informations.
Les flèches pointillées représentent des successions d'exécution.

VII . LA TRADUCTION D'UN FILTRE EN PROGRAMME MAQUETTE . SRATEGIE .

oOOo

La phase qui nous intéresse ici dans le traitement des questions exprimées en PIVOINES est celle qui transforme un filtre en un programme maquette (cf § IV-2-2-1-c où ce programme était appelé "programme indéterministe", et IV-3), constitué principalement d'appels de modules de recherche (il existe un tel module par relation de PHYLOG). Rappelons qu'après cette phase cette maquette donnera naissance à un programme exécutable (généré par le programme de traduction TRADID) qui, lui, comportera la gestion des informations : en particulier, pour chaque module de recherche du programme maquette, le programme exécutable contiendra un bouclage sur un appel de la procédure de recherche correspondante, appel précédé de la mise en place des paramètres et suivi de celle des résultats.

Avant d'expliquer l'organisation et le déroulement de la phase de traduction d'un filtre en programme maquette, examinons ce qu'est ce programme maquette.

VII-1- FORME DU PROGRAMME MAQUETTE.

VII-1-1- NIVEAU DES MOTIFS.

Nous avons déjà dit (§ IV-2-2-1-c) que le programme maquette résultant de la traduction d'un motif Mi est une suite d'appels de modules de recherche ; chacun de ces appels de module peut être écrit sous la forme $R(a_1,a_2,\ldots,a_u\ ;b)$ où :

- R est un nom de relation de la structure PHYLOG, c'est-à-dire un nom de procédure de recherche, éventuellement affecté d'un opérateur unaire ;

- $a_1,a_2,\ldots,a_u$ est une liste d'"arguments" qui sont les étiquettes de l'origine de l'arc considéré dans le graphe généralisé représentant le motif pour la structure PHYLOG, dans le cadre de la stratégie élaborée ; ces arguments sont soit des données élémentaires (arguments fixés) soit des inconnues ;

- b est le but qui est l'étiquette du but de l'arc considéré.

Nous appellerons EVAL(Mi) la suite d'appels de modules correspondant au motif Mi.

VII-1-2- NIVEAU DU FILTRE.

Le programme maquette doit contenir des renseignements reflétant l'expression booléenne qu'est le filtre. Ces renseignements sont de quatre types :

1) avant la séquence EVAL(Mi) correspondant au motif Mi, le programme maquette doit contenir une indication permettant de désigner le sous-filtre f_j qui est le contexte à gauche (cf § V-5-1-2) de Mi. Nous verrons dans la remarque située à la fin du présent paragraphe quelle est cette désignation. Le programme maquette contiendra donc ici : (désignation de f_j).

2) après la séquence qui, dans le traitement d'un sous-filtre de la forme f_1 ou f_2, correspond au sous-filtre f_2, le programme maquette doit contenir une instruction demandant la réunion des résultats de f_1 et de f_2. Cette instruction devra comporter la désignation du sous-filtre f_1. Elle s'écrira : REU (désignation de f_1). A l'exécution, elle sera traduite par un appel du module de réunion (cf § VIII-1-4).

3) après la séquence qui, dans le traitement d'un sous-filtre de la forme f_1 et f_2, correspond au sous-filtre f_2, le programme maquette n'a pas besoin de contenir d'instruction demandant une intersection car aucune intersection n'est jamais effectuée (cf § V-5-3) ; par contre, pour la gestion de l'espace des résultats, il est utile de savoir que f_1 ne peut plus être le contexte à gauche d'aucun sous-filtre : nous dirons qu'alors le résultat de f_1, qui était la "source" de certains motifs de f_2 (cf § V-5-3), est déchu de son rôle de source. Ceci sera exprimé par l'instruction :

DECHU(désignation de f_1).

4) enfin le programme maquette doit contenir certains jalons utiles pour le déroulement de la recherche en ce qui concerne les résultats booléens (cf § VIII-1-1-5). Ces jalons seront :

- des étiquettes :

  . après chaque EVAL(Mi), sera générée l'étiquette Fi qui, dans le programme exécutable, se retrouvera telle quelle ; son utilité est que si un module de EVAL(Mi) échoue (à l'exécution), le reste de EVAL(Mi) ne sera pas exécuté, un branchement étant effectué à l'étiquette Fi ;

  . si échoue l'évaluation du sous-filtre f_1, premier terme du sous-filtre f_1 et f_2 (c'est-à-dire fils gauche d'un sous-filtre dont le connecteur est et (cf § V-5-1-2)), alors il est inutile d'évaluer le fils droit du sous-filtre. Le programme maquette contiendra, à la fin de la séquence correspondant à l'évaluation de f_2, juste avant DECHU(-), une étiquette Ej qui sera reproduite dans le programme exécutable. Ces étiquettes seront créées dans l'ordre croissant des numéros (E1, E2,...) au fur et à mesure des besoins.

- des instructions :

  . dans le cas précédent, il faut qu'à l'exécution, en cas d'échec de f_1, ait lieu un branchement à Ej. Cet ordre doit être préparé dès la traduction du filtre. Ainsi, après la séquence correspondant à f_1, sera généré une instruction B(Ej) qui, dans le programme exécutable, sera remplacée par un ordre signifiant :

    "si échec aller à Ej".

  . enfin si un sous-filtre est de la forme f_1 ou f_2, alors à l'exécution le résultat booléen de f_1 devra être empilé sur PILBOOL (cf § VIII-1-1-5). Cet ordre d'empilement doit être préparé dès la traduction du filtre. Ainsi, après la séquence correspondant à f_1, sera générée une instruction EMPILER qui, dans le programme exécutable, sera remplacée par un ordre signifiant : "empiler le résultat booléen du motif sur PILBOOL".

Remarque: désignation des motifs et sous-filtres

Cette désignation n'a pas une très grande importance pourvu qu'elle ne soit pas ambiguë. Elle sera utile principalement dans le programme TRADID (cf § VIII-2) pour la gestion de la pile PILINC qui permet de préparer la gestion des données et résultats des appels de procédures de recherche.

Spécifier une désignation va consister à attribuer un numéro à chaque noeud de la ramification des motifs du filtre. Si un noeud n (correspondant au sous-filtre f) est un prédécesseur du noeud n' (correspondant à f'), alors les résultats de f' n'auront plus aucune utilité lorsque f aura été évalué (car alors f' ne pourra plus être contexte à gauche): nous dirons que f et f' ne sont pas d'actualité en même temps. Dans ce cas, on peut leur attribuer le même numéro.

Nous avons donc pris la décision suivante que nous décrivons par récurrence :

- chaque motif est désigné par son numéro d'ordre dans l'écriture du filtre ;
- si f est de la forme f_1 et f_2, ou bien f_1 ou f_2, alors : désignation de f = désignation de f_2 ;
- le sous-filtre vide est désigné par 0.

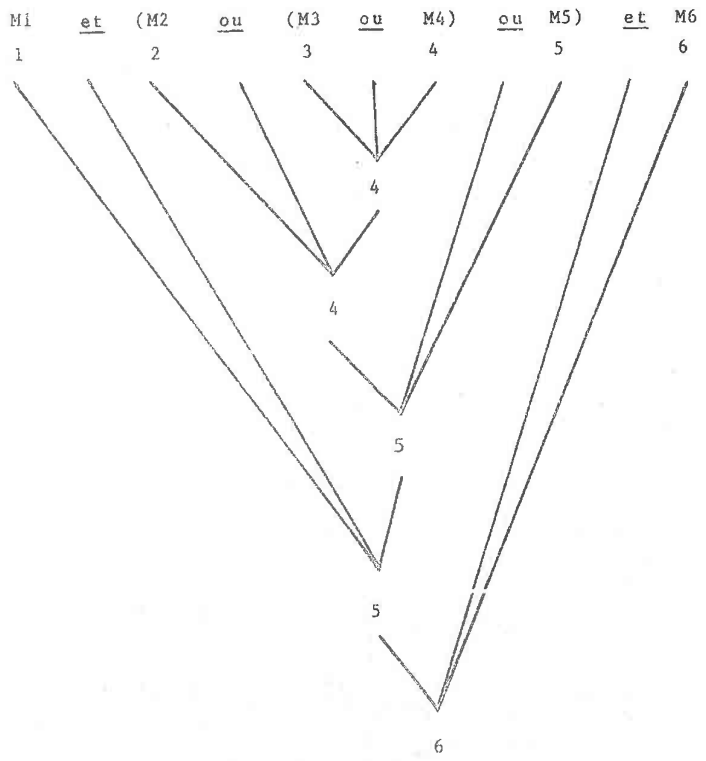
A chaque sous-filtre désigné par j correspondra le $j^{ème}$ "étage" de la pile PILINC.

VII-1-3- EXEMPLE.

Soit le filtre

F = M1 et (M2 ou (M3 et M4) ou M5) et M6 .

Ecrivons la ramification résultant de l'analyse de F (en supprimant les noeuds sans intérêt) et, à chaque noeud, attachons (comme attribut) le numéro désignant le sous-filtre correspondant :



Le programme maquette correspondant à F sera (en ne détaillant pas les EVAL(Mi)) :

(0), EVAL(M1), F1 : B(E1),
(1), EVAL(M2), F2 : EMPILER,
(1), EVAL(M3), F3 : B(E2),
(3), EVAL(M4), F4 : E2 : DECHU(3),
REU(2), EMPILER,
(1), EVAL(M5), F5 :
REU(4), E1 : DECHU(1), B(E3),
(5), EVAL(M6), F6 : E3 : DECHU(5) ;

A l'exécution, ce programme se déroulera selon les étapes suivantes :

- Evaluation de M1 avec comme CG (contexte à gauche) le sous-filtre désigné par 0. Si au cours de l'évaluation il se produit un échec, saut à F1.

- F1 : en cas d'échec de M1 aller à E1 (qui enverra tout de suite à E3 qui marquera l'échec définitif).

- Evaluation de M2...

- F2 : empiler le résultat booléen de M2 sur PILBOOL

- Evaluation de M3...

- F3 : en cas d'échec de M3, aller à E2 (on saute l'évaluation de M4)

- Evaluation de M4...

- F4 : E2 : réunion du résultat de M2 et de celui de M4 (c'est-à-dire de M3 et M4). On enlève le sommet de PILBOOL qui contient le résultat booléen de M2.

- On empile le résultat booléen de la réunion en prévision de la deuxième réunion.

- Evaluation de E5...

- F5 : réunion du résultat de M2 ou (M3 et M4) et de celui de M5. On enlève le sommet de PILBOOL.

- E1 : en cas d'échec de M1 et (M2 ou (M3 et M4) ou M5), aller à E3 (échec final).

- Evaluation de E6...

- F6 : E3 : fin du programme. BOOL contient le résultat booléen final.

VII-2- ORGANISATION DE LA TRADUCTION.

Au paragraphe IV-2-2-2, nous avons expliqué pourquoi l'on ne peut pas dissocier la traduction d'un motif en une succession d'appels de modules d'une part et la détermination d'une stratégie d'autre part, c'est-à-dire le choix d'un ordre sur ces appels : les modules eux-mêmes dépendent de la stratégie.

Dans le but d'étudier de façon modulaire la traduction d'un filtre en programme maquette, on peut considérer cette traduction comme constituée de deux parties :

- une partie purement syntaxique qui elle-même se décompose en deux sections :

  . traitement de tout ce qui doit être fait au niveau du filtre (analyse et mise en place des instructions et étiquettes) ;

  . construction des familles d'arborescences GD et DG associées aux motifs, que nous présenterons et justifierons au paragraphe VII-3-2.

- une partie liée à la structure PHYLOG et au choix d'une stratégie pour le traitement des motifs : la partie que nous avons appelée "Groupement, traduction, choix". Elle fait appel aux propriétés des relations de la structure logique (propriétés reflétées par la correspondance entre LOG et PHYLOG), alors que la première partie en est indépendante, sauf pour des vérifications éventuelles de conformité.

VII-3- PARTIE PUREMENT SYNTAXIQUE DE LA TRADUCTION.

VII-3-1- TRAITEMENT AU NIVEAU DU FILTRE.

L'analyse syntaxique du filtre est faite de façon classique pour une expression au moyen d'une pile, avec une relation de priorité. Mais en plus de cette gestion, il est nécessaire de savoir à chaque instant quel est le contexte à gauche du motif que l'on aura à traiter. Ceci nécessite une deuxième pile. Appelons PILCLA et PILET ces deux piles. La pile PILET (pile et) est modifiée chaque fois qu'entre ou sort dans la pile classique PILCLA l'opérateur et. Etudions le fonctionnement de ces deux piles.

Gestion des piles au fur et à mesure qu'on lit le filtre (en considérant les motifs comme s'ils étaient écrits sous la forme Mi).

Appelons s, s^-, s^{--} les fonctions qui à chaque instant associent à chacune des deux piles respectivement son sommet, son "sous-sommet", son "sous-sous-sommet". On entretient un compteur d'étiquettes E qu'on appelle CETI.

+ Au départ, les deux piles et CETI contiennent 0.

+ Lecture d'un élément u du filtre

. u est le motif Mi

(à ce moment, s(PILET) contient le numéro du sous-filtre fils gauche du dernier et entré, c'est-à-dire le numéro du contexte à gauche de Mi)

- i entre sur PILCLA
- on génère : (s(PILET))
- on traite Mi (pour générer EVAL(Mi))
- on génère : Fi :

. u est (:

(entre sur PILCLA

. u est), ou, et, signe de fin de filtre

Module TESTPRI :

si u $\lessdot^{(1)}$ s^-(PILCLA) alors s^-(PILCLA) doit être traité (Module TRAITS) sinon Module ENTRE.

---

(1) Tableau de la relation $\lessdot$:

u \ s^-	0	(	ou	et
)	erreur		$\lessdot$	$\lessdot$
ou			$\lessdot$	
et			$\lessdot$	$\lessdot$
fin	fin	erreur	$\lessdot$	$\lessdot$

Module TRAITS :

. si s^- (PILCLA) = et

- et sort de PILCLA ainsi que ses opérandes et on doit entrer le numéro du résultat, c'est-à-dire le numéro qui était au sommet : ainsi on ôte s, on ôte s- et on remplace s^{--} par s.

- on génère :
  s^-(PILET) : DECHU(s(PILET)).

- on ôte s(PILET) et s^-(PILET):
  s(PILET) contenait le numéro du sous-filtre "déchu"
  s^-(PILET) contenait le numéro d'étiquette où envoyer en cas d'échec de ce sous-filtre.
- Module TESTPRI

. si s^-(PILCLA) = ou

- on génère : REU(s^{--}(PILCLA))

- ou sort de PILCLA ainsi que ses opérandes : on ôte s, on ôte s^- et on remplace s^{--}par s.

- Module TESTPRI.

Module ENTRE :

. si u = et

- u entre sur PILCLA

- CETI:= CETI + 1

- on entre E suivi du contenu de CETI sur PILET

- on génère : B(s(PILET))

- on entre s^- (PILCLA) sur PILET : c'est le numéro qui désigne le contexte à gauche du prochain motif à traiter.

- on retourne à la lecture

. si u = ou

- u entre sur PILCLA

- on génère : EMPILER

- on retourne à la lecture

. si u =) c'est que s^-(PILCLA) = (

- on sort s(PILCLA) et on remplace s^-(PILCLA) par la valeur qu'on vient de sortir

- on retourne à la lecture

. si u = signe de fin de filtre

- on génère un signal de fin de programme maquette.

Exemple : reprenons l'exemple qui constitue le paragraphe VII-1-3 et montrons l'évolution des piles et les fragments générés.

u(lecture)	Etat de PILCLA après la lecture	PILET	Fragment généré
	0	0	
M1	0 1	0	(0) EVAL(M1) F1 :
et	0 1 et	0 E1 1	B(E1)
(	0 1 et (	-	
M2	0 1 et (2	-	(1) EVAL(M2) F2 :
ou	0 1 et (2 ou	-	EMPILER
(	0 1 et (2 ou (	-	
M3	0 1 et (2 ou (3	-	(1) EVAL(M3) F3 :
et	0 1 et (2 ou (3 et	0 E1 1 E2 3	B(E2)
M4	0 1 et (2 ou (3 et 4	-	(3) EVAL(M4) F4 :
)	0 1 et (2 ou (4	0 E1 1	E2 : DECHU(3)
	0 1 et (2 ou 4	-	
ou	0 1 et (4	-	REU(2)
	0 1 et (4 ou	-	EMPILER
M5	0 1 et (4 ou 5	-	(1) EVAL(M5) F5 :
)	0 1 et (5	-	REU(4)
	0 1 et 5	-	
et	0 5	0	E1 : DECHU(1)
	0 5 et	0 E3 5	B(E3)
M6	0 5 et 6	-	(5) EVAL(M6) F6 :
fin	0 6	0	E3 : DECHU(5)

VII-3-2- FAMILLES D'ARBORESCENCES GD ET DG.

Dans ce paragraphe, et toute la fin du chapitre VII d'ailleurs, il ne sera plus question de filtres mais uniquement de motifs.

La phase de traduction d'un motif en programme maquette commence nécessairement par l'analyse syntaxique du motif, par un procédé qui a été décrit au paragraphe V-3-5. Mais il nous a semblé utile d'alléger un peu plus la tâche de traduction proprement dite en poussant au maximum l'exploitation syntaxique du motif. C'est ainsi que nous avons pensé à présenter le résultat de l'analyse d'un motif sous la forme de deux familles d'arborescences : les arborescences "Gauche-Droite" (GD) et les arborescences "Droite-Gauche" (DG) ; ces familles d'arborescences ne contiennent pas plus d'"information" que les pochoirs eux-mêmes mais mettent en évidence l'enchaînement des différents arcs entre eux.

Avant de décrire ces arborescences, nous devons parler des "extrémités" d'un motif.

Extrémités d'un motif

Au paragraphe V-3-1(2°), nous avons défini les points d'entrée et les points de sortie d'un motif, en précisant qu'il était possible qu'il n'y en ait pas.

Lorsque l'on se demande comment va s'organiser la recherche relative à un motif, la première question qui se pose est de savoir quelles sont les premières étapes possibles ; et pour déterminer ces étapes, on doit connaître quels peuvent être les points de départ de ces étapes. Ce sont ces points de départ que l'on appelle les extrémités du motif.

Certaines extrémités (extrémités "naturelles") sont imposées par la nature du motif : ce sont ses points d'entrée, mais aussi ses points de sortie car nous verrons au paragraphe VII-4-3 que le motif peut être partiellement traité en recherchant des arcs réciproques des arcs du motif . Si US figure dans le motif, il est aussi une extrimité.

D'autres extrémités peuvent être ajoutées aux premières... en particulier cela est nécessaire lorsque celles-ci sont absentes. On peut imaginer des heuristiques de création d'extrémités :l'une peut être que chaque sommet étiqueté par un élément de Λ_1 soit désigné comme extrémité, d'autres peuvent utiliser des propriétés des relations. Pour l'instant, nous avons décidé de considérer comme extrémités tous les points origines donnés (c'est-à-dire qui ne sont pas des inconnues) des relations d'arité supérieure à 2. De plus, nous convenons que l'utilisateur peut imposer les extrémités qu'il désire en plus des extrémités obligatoires (naturelles et origines d'arcs n-aires). Il faut noter que le choix des extrémités a une grande influence sur le programme que l'on génère.

Définition de la famille d'arborescences GD dont nous donnons un exemple plus loin.

La famille d'arborescences GD ("gauche-droite") correspondant à un motif M exprimant un pochoir P est un graphe (généralisé : cf § II-3-2) qui a les propriétés suivantes :

- chaque arc de P se trouve une fois et une seule dans GD (sous la même forme que dans P) ;

- les racines de GD sont :

  . toutes les extrémités qui ne sont pas des points de sortie

  . tous les sommets (d'arcs) qui sont buts de plusieurs arcs

  . toutes les origines de relations d'arité supérieure à 2 ; il s'agit alors de racines "généralisées", constituées de plusieurs points origines (cf § V-3-1).

- pour chaque racine Z, la descendance de Z (qui est une arborescence, généralisée en ce sens que sa racine peut être une racine généralisée, comme défini ci-dessus) est déterminée de la façon suivante :

  . de Z partent tous les arcs de P ayant Z comme origine ; appelons b_i les buts de ces arcs ;

  . pour chaque b_i : s'il est ou fait partie d'une racine de GD (1), ou s'il est un point de sortie de P , alors il est une feuille pour la composante considérée ; sinon on peut dire pour b_i tout ce qu'on vient de dire pour Z.

- chaque feuille d'arborescence GD est munie d'un pointeur vers l'arborescence dont elle est racine (non généralisée), si cette arborescence existe.

---

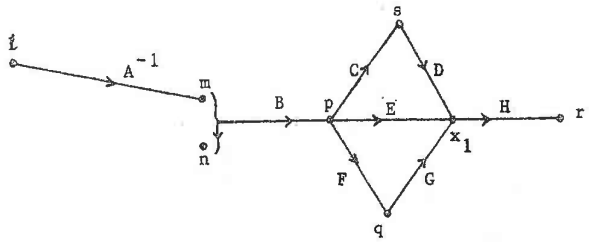
(1) en particulier s'il est but de plusieurs arcs ; dans ce cas, les divers arcs ne se rejoignent pas dans les arborescences : le but est dédoublé.

Exemple de ramification GD

Soit le motif :

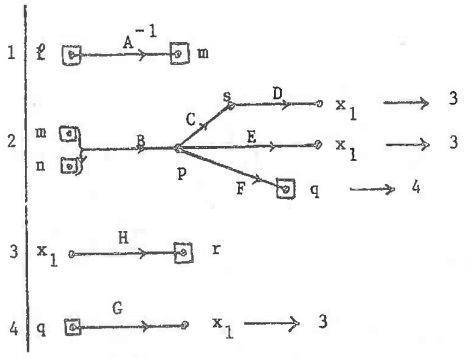
$$M = \langle \ell\ A^{-1} m, n \rangle B\ p \langle CsD,\ E,\ F\ q\ G \rangle\ x_1\ H\ r$$

représenté par le graphe(généralisé) :



Soit ℓ, m, n, q, r ses extrémités : ℓ, n et r sont des extrémités naturelles, m est point origine de B d'arité 3, q est choisi par l'utilisateur. Alors la famille d'arborescences GD est :

Numéro d'arborescence



Nous avons encadré les extrémités.
L'ordre des arborescences n'est pas significatif.

Définition de la famille d'arborescences DG

Considérons le pochoir P' obtenu à partir du pochoir P en :

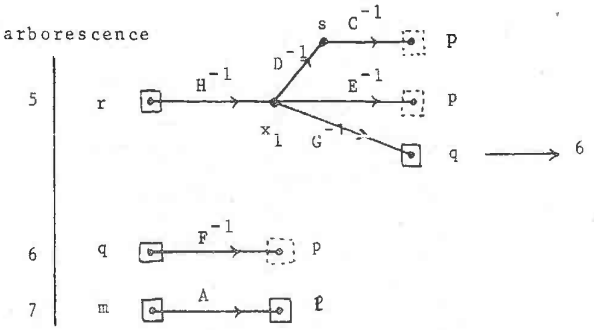
- supprimant tous les arcs dont la relation est d'arité supérieure à 2,
- inversant les rôles de l'origine et du but de chaque arc restant et remplaçant sa relation par la réciproque de celle-ci.

La famille d'arborescences DG est obtenue à partir de P' comme la famille GD l'est à partir de P.

Exemple de ramification DG :

Pour le motif faisant l'objet de l'exemple précédent, la famille d'arborescences DG est la suivante :

Numéro d'arborescence



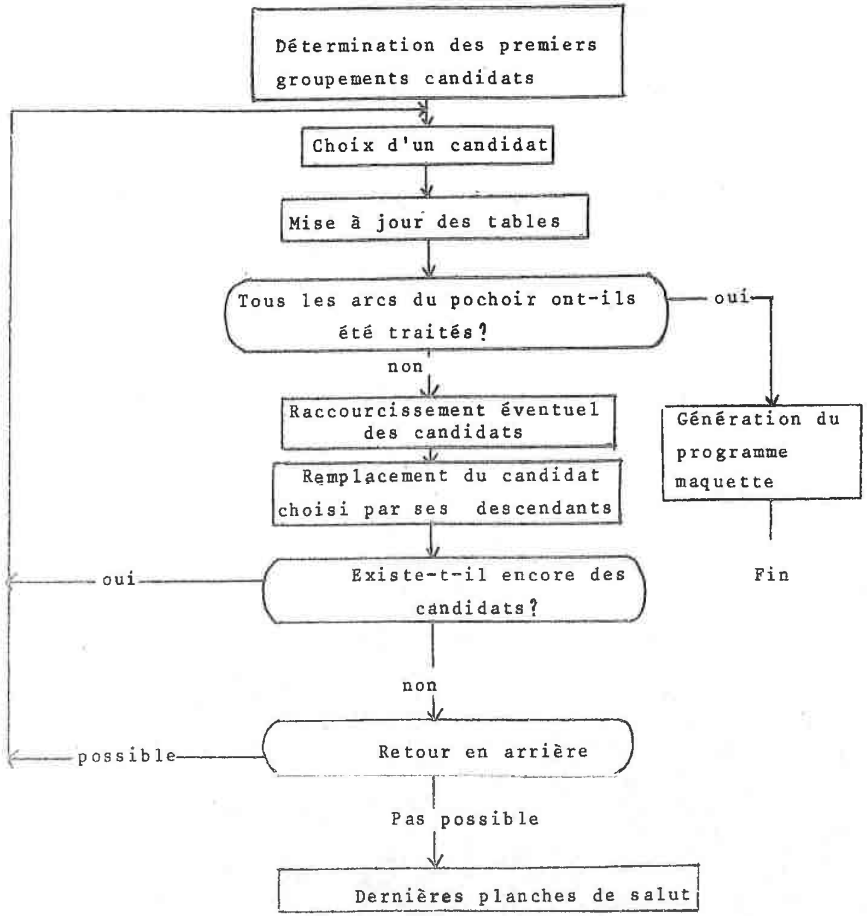
Nous avons encadré les feuilles étiquetées p par des traits discontinus car p est un point de sortie de P', créé par la suppression de l'arc correspondant à la relation 3-aire B.

VII-4- "GROUPEMENT, TRADUCTION, CHOIX".

Nous avons exposé au paragarphe IV-2 les grandes lignes de cette phase et en avons même donné un organigramme grossier au paragraphe IV-4. Elle a été réalisée sur ordinateur par un étudiant qui a rédigé à ce sujet son rapport de diplôme d'études approfondies (D.E.A.) [76] auquel on pourra se reporter pour plus de détails.

Nous allons maintenant donner un organigramme plus détaillé et expliquer rapidement chacun des modules qui y figurent en commentant les différents problèmes qui se posent, en justifiant les décisions prises et en proposant des améliorations ultérieures éventuelles.

VII-4-1- ORGANIGRAMME.



VII-4-2- COMMENT EST EXPRIMEE LA CORRESPONDANCE ENTRE LA STRUCTURE LOGIQUE ET PHYLOG ?

La correspondance entre la structure logique et PHYLOG est établie par la personne qui choisit la représentation ; ce peut être l'"administrateur des données" [6 et 28].

Il s'agit de la représentation d'une structure (appelons-la LOG) dans une autre, PHYLOG (cf § VI-2).

La représentation des termes du vocabulaire et des repères privilégiés (éléments de Λ_1) est donnée par une table utilisée dans le module de génération du programme maquette. N'en parlons plus maintenant.

La correspondance entre LOG et PHYLOG est une correspondance entre :

- presque tous les symboles relationnels de LOG (tous à l'exception des accès que l'on veut interdire) et certaines combinaisons de ceux-ci par des opérations de composition , réciproque, itération, négation (nous appelons ces combinaisons et, par extension, les symboles eux-mêmes, des groupements traduisibles ; appelons Q leur ensemble)

- et des symboles relationnels de PHYLOG ou combinaisons de ceux-ci.

<u>Exemple</u> de groupement traduisible : $T \neg S^{-1 *}$.

Remarque : nous verrons à la fin du paragraphe qu'en fait les groupements traduisibles peuvent comporter des sommets nommés (points intermédiaires).

Items :

Nous appelons item d'un groupement traduisible chaque symbole relationnel ou opérateur qui y figure. Nous considérons chaque groupement traduisible comme une liste d'items dans laquelle chaque nom de relation est suivi des opérateurs unaires qui le concernent, dans l'ordre où ils opèrent. Dans l'exemple ci-dessus, la liste d'items est : T, S, $^{-1}$, *, $\neg$. On peut considérer aussi une telle liste comme un mot sur le vocabulaire $K = \Lambda \cup \{-1, *, \neg\}$.

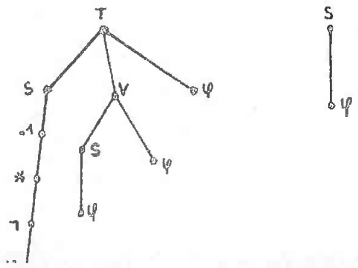
Ramification des groupements traduisibles :

L'ensemble Q est donc un ensemble de mots sur K. On peut le représenter par une ramification avec partage des débuts de mots (cf par exemple [40] et [51]).

Exemple très élémentaire :

soit $Q = \{ T \neg S^{-1} *, T, T V, T V S, S \}$

alors la ramification est :



φ est nécessaire pour matérialiser les branches correspondant à TV et T. L'ordre des noeuds de chaque famille est choisi comme étant l'ordre décroissant de la longueur du plus long chemin issu de chacun de ces noeuds (ordre quelconque entre les ex aequo).

Correspondance entre LOG et PHYLOG :

Spécifier une correspondance de LOG vers PHYLOG, c'est attacher, à chaque feuille de la ramification des groupements de LOG, une combinaison de symboles de PHYLOG (la plupart du temps, ce sera un simple symbole relationnel) : nous l'appellerons la "traduction" du groupement correspondant à la feuille.

C'est au cours du programme "Groupement, traduction, choix" que cette correspondance est utilisée, c'est-à-dire lorsque, étant donné un motif M, on cherche le programme maquette qui lui correspond. Elle rend compte implicitement de propriétés des relations de la structure LOG : les groupements traduisibles eux-mêmes reflètent certaines de ces propriétés, mais aussi les traductions, et même les indications d'efficacité dont il est question plus loin.
Donnons deux précisions importantes sur cette correspondance .

D'abord il se peut qu'on désire donner des traductions conditionnelles pour certains groupements: les conditions peuvent porter sur la nature de l'origine du groupement dans le motif M, sur l'environnement de ce groupement (autres groupements de même origine ou successeurs du groupement considéré, par exemple), etc... Une règle importante doit être respectée : les conditions exprimées doivent être "statiques", en ce sens que le fait qu'elles soient vérifiées ou non doit être indépendant du temps. Par exemple, la condition :

"l'origine du groupement doit être une inconnue" est une condition valable, tandis que "l'origine du groupement doit être une inconnue déjà évaluée au cours du motif" ne l'est pas. Ceci se justifie de la façon suivante : c'est au moment où le groupement traduisible devient candidat que l'on doit prendre en compte la condition, or la véracité de la condition pourrait changer entre cet instant et le moment où ce candidat est choisi donc "activé".

Pour exprimer les conditions, on peut envisager une solution du type suivant. Une condition serait une expression booléenne (avec des restrictions) dont les atomes seraient constitués d'un objet, un indicateur de relation, et une nature ou valeur :

- l'objet désignerait ce sur quoi porte la condition (point origine, point terminal,...),
- l'indicateur serait un symbole d'égalité ou d'inégalité, ou d'appartenance,
- la nature serait:"inconnue" ou "donnée" ou un nom d'ensemble, ou...

La deuxième précision à apporter est la suivante. Nous venons de voir qu'à chaque feuille de la ramification et pour chaque condition qui s'y rapporte (il y en aura rarement) on doit attacher une traduction dans PHYLOG. Mais en plus de cette traduction, il faut attacher des <u>indications "d'efficacité"</u> permettant d'effectuer le module "choix d'un candidat", c'est-à-dire d'appliquer les critères de choix ; parmi elles, figure le "degré d'indéterminisme" (cf § VII-4-4).

<u>Problème des points intermédiaires dans les groupements traduisibles.</u>

Nous avions primitivement jugé commode de ne donner dans la correspondance entre LOG et PHYLOG que la traduction de groupements traduisibles obtenus par les opérations citées plus haut, sans points intermédiaires (par exemple "F a G" n'était pas prévu). Ainsi, dans la traduction d'un motif, lorsque l'on comparait une succession d'arcs d'une arborescence GD ou DG avec l'ensemble des groupements traduisibles, on arrêtait la comparaison dès que l'on arrivait, dans l'arborescence, à un sommet d'arc nommé (par une valeur ou une inconnue). Or c'est gênant, comme l'illustre l'exemple suivant inspiré par le problème des dossiers médicaux (§ III-2-2).

On désire chercher les occurrences du motif US D_1 D_2 x_1 D_3 C coeur. Considérer séparément les deux parties US D_1 D_2 x_1 et x_1 D_3 C coeur serait vraiment compliqué puisque l'accès, sans que l'utilisateur ait besoin de le savoir, se fait directement de US à coeur. La correspondance entre LOG et PHYLOG, pour ce problème, doit donc comporter une traduction pour le groupement D_1 D_2 D_3 C <u>mais aussi pour D_1 D_2 point D_3 C</u> .

Il y a a priori deux façons d'exprimer ce cas : ou bien, à la feuille correspondant au groupement D_1 D_2 D_3 C, attacher deux traductions selon la condition "possède un point nommé après D_2" (il faudrait d'ailleurs attacher d'autres conditions à cette feuille), ou bien considérer, parmi les items des groupements traduisibles, les points intermédiaires; dans ce cas, la feuille de D_1 D_2 D_3 C ne serait pas la même que celle de D_1 D_2 point D_3 C.

Nous appelons points d'articulation d'un groupement les points - à part l'origine - où peuvent s'articuler d'autres groupements. Dans les motifs, tous les embranchements sont nommés. Les points d'articulation d'un groupement sont donc le point terminal de ce groupement et ses points inter - médiaires nommés. Ainsi, dans le groupement D_1 D_2 point D_3 C , les points d'articulation sont : "point" et la fin du groupement.

VII -4-3- CHEMINEMENT DANS LES FAMILLES GD et DG. GROUPEMENTS CANDIDATS

La recherche des groupements candidats s'effectue par une confrontation entre :

- les familles d'arborescences GD et DG, qui sont tirées du motif considéré
- et la ramification des groupements traduisibles qui, elle, est liée à la structure de données et à sa représentation.

Dans le premier module du programme "groupement, traduction, choix", les points à partir desquels on cherche les premiers groupements candidats sont: toutes les extrémités du motif M considéré. On initialise alors le cheminement dans les arborescences GD et DG au front constitué par ces extrémités. En fait, nous considérons ce front comme constitué

de deux fronts : le front GD et le front DG ; les cheminements à partir de chaque front se feront uniquement dans la famille correspondante (et dans le sens racine vers feuilles). Notons que certaines extrémités font partie des deux fronts, les autres pas. Ainsi, dans l'exemple du paragraphe VII-3-2, le front GD est constitué de l, m, n, q et le front DG de r, q, m.

A chaque étape, les groupements candidats sont des chemins des arborescences, issus des points du front ; lorsqu'une feuille d'une arborescence A1 renvoie à une autre arborescence A2, celle-ci est supposée prolonger A1 (et donc les chemins de A1 se poursuivent dans A2). Plus précisément, les groupements candidats sont, parmi ces chemins, ceux qui coïncident avec des éléments de Q (ensemble des groupements traduisibles) et ne comportent aucun arc déjà pris en compte (voir plus loin dans le présent paragraphe). Le cas de groupements dont l'un est le début de l'autre est étudié au paragraphe VII-4-4, à propos de la conjecture H.

Si US figure dans le motif considéré et si celui-ci est le premier du filtre, alors US lui-même est considéré comme un groupement (d'origine et but US) qui est choisi d'office en premier (il n'a même pas besoin d'être porté sur la liste des candidats).

Lorsqu'un groupement candidat (dont l'origine o fait donc partie du front GD ou DG) vient d'être choisi, o est remplacé dans le front par les points d'articulation du groupement. C'est ainsi que le front "avance". On détermine alors les nouveaux groupements candidats et ainsi de suite. On s'arrête lorsque tous les arcs du pochoir ont été pris en compte (que ce soit dans le sens GD ou dans le sens DG).

Cas des opérateurs unaires : soit par exemple la succession d'arcs $aR^{-1}S^{*}$. Si $R^{-1}S^{*}$ est un élément de Q, il sera traduit directement. Sinon, si R^{-1} est un élément de Q, il sera traduit directement ; dans le cas contraire c'est R qui sera traduit et l'on notera que la traduction doit être affectée de l'opérateur $^{-1}$. Puis, si S^{*} est un élément de Q, il sera traduit directement; sinon c'est Q qui sera traduit et l'on notera que sa traduction doit être affectée de l'opérateur * (ce qui d'ailleurs posera des problèmes).

Cas des arcs d'arité supérieure à 2.

Nous avons cité, parmi les extrémités d'un motif, les points origines donnés des arcs d'arité supérieure à 2. Pour chaque point origine qui est une inconnue x_i, deux cas peuvent se présenter : il se peut que ce soit une extrémité mais ce cas sera très rare, la plupart du temps il est sommet d'autres relations. Dans le premier cas, il sera considéré par le programme traducteur comme ayant une valeur (attribuée par ailleurs, que ce soit grâce aux propriétés de la relation ou grâce à la partie cadrage ; voir paragraphe VIII-1-3-2, module A2). Dans le deuxième cas, il ne sera considéré comme ayant une valeur qu'après que l'inconnue x_i ait été prise en compte dans un groupement choisi.

Un groupement commençant par une relation d'arité supérieure à 2 devient candidat dès que tous ses points origines inconnus sont considérés comme ayant reçu une valeur.

Remarquons qu'un tel groupement peut être déchu de son état de candidat si, au cours d'un retour en arrière, un de ses points origines inconnus ne peut plus être considéré comme ayant une valeur. Cette remarque a obligé à mettre en place un dispositif assez compliqué pour la candidature des groupements d'arité supérieure à 2.

Les groupements candidats ne peuvent-ils jamais contenir d'arc déjà pris en compte ?

Il est normal de penser qu'un arc déjà pris en compte dans un groupement choisi ne puisse pas faire l'objet d'une deuxième prise en compte au sein d'un autre groupement. De plus, cette décision a deux avantages importants : d'abord, accepter la décision contraire amènerait à avoir un beaucoup plus grand nombre de groupements candidats à chaque étape, en particulier les dernières ; ensuite le test de fin de programme, qui porte sur le fait que tous les arcs aient été traités, ne serait plus valable. Mais ce deuxième argument ne résiste pas à la réflexion car il suffit d'imposer que chaque nouveau groupement choisi ait au moins un arc non encore traité pour que le nombre d'arcs non traités diminue à chaque étape et ainsi que le programme puisse s'arrêter après un nombre fini d'étapes (nombre qui risque d'être assez grand cependant).

Nous avons adopté la décision d'obliger les groupements candidats, donc les groupements choisis, à ne contenir aucun arc déjà traité . D'où le module "raccourcissement des candidats" qui, après le choix d'un candidat C, supprime des autres candidats les arcs qui figurent dans C, ou supprime ces candidats s'ils ne possèdent pas de début traduisible ne contenant pas ces arcs.

Cependant, lorsque, malgré des retours en arrière (cf § VII-4-6), on ne peut parvenir à la fin de la traduction du motif, c'est-à-dire si l'on aboutit toujours à une impasse, on peut activer le module "planche de salut" où l'on supprime la contrainte d'unicité (cf § VII-4-6).

Nous avons, dans ce paragraphe, traité du but et du principe des modules suivants de l'organigramme :

- raccourcissement éventuel des candidats
- tous les arcs ont-ils été traités ?
- détermination des premiers groupements candidats
- remplacement du candidat choisi par ses descendants (c'est-à-dire les groupements issus de ses points d'articulation).

Nous ne décrirons pas ces modules, qui fonctionnent dans la réalisation effectuée. Les deux derniers utilisent le module "détermination des groupements candidats issus d'un point donné".

Disons simplement que l'on entretient une liste des groupements candidats, liste qui ne se modifie que par adjonction de nouveaux candidats ; en effet il est nécessaire de pouvoir revenir sur un choix antérieur et de pouvoir retrouver, pour ce choix, quels étaient tous les candidats. Dans cette liste, chaque groupement candidat est accompagné de son origine et de ses points d'articulation.

Nous allons maintenant aborder les autres modules de l'organigramme.

VII-4-4- CHOIX D'UN CANDIDAT.

Nous avons déjà, au paragraphe IV-2-3, donné le principe de ce choix. Mais d'un problème à l'autre le jugement de la qualité d'un programme peut reposer sur des critères

différents : les deux principaux sont le temps et l'encombrement. Ces deux critères, très souvent contradictoires, sont ici dans une certaine mesure concourants. En effet, notre problème, au point de vue de la stratégie, est principalement un problème de quantités de résultats à chaque étape, ces résultats devant servir de données pour des étapes ultérieures ; moins il y aura de résultats, meilleur sera l'encombrement mais aussi le temps.

Nous n'avons pas fait d'étude théorique sur l'influence de critères locaux de choix des candidats sur les facteurs encombrement et temps mais nous pouvons cependant livrer quelques réflexions à ce sujet. Nous mettrons ces réflexions en application en écrivant ce que l'on pourrait appeler des "heuristiques de choix" sous forme de modules de programme dont la place est prévue dans le module "choix d'un candidat"; puis nous effectuerons une étude expérimentale comparative des différentes heuristiques. Voici ces réflexions.

Degré d'indéterminisme :

Le critère local le plus important, vraisemblablement à quelque point de vue qu'on se place pour juger la qualité finale, est ce que nous avons déjà présenté (§ IV-2-1) sous le nom de "déterminisme". Nous avons décidé d'affecter à chaque groupement traduisible un "degré d'indéterminisme" sur lequel portera l'étape de choix ; ce n'est qu'à égalité d'indéterminisme que joueront les autres critères. Les groupements traduisibles qui sont des fonctions sont dits déterministes et se voient attribuer 0 comme degré d'indéterminisme. Pour les autres, il est difficile de fixer un "barême" et il serait intéressant d'établir des règles d'évaluation de ces degrés, en fonction de résultats statistiques d'emploi des diverses relations de PHYLOG dans l'information en particulier. Nous n'en sommes pas là et avons décidé d'effectuer une gradation très grossière en l'absence de moyens

d'attribution précis : les groupements traduisibles non déterministes se verront affecter, en même temps qu'une traduction en PHYLOG (par exemple par l'administrateur de données), un degré d'indéterminisme égal à 1 s'ils sont jugés peu indéterministes et à 2 s'ils sont jugés très indéterministes.

Même aussi peu précis, ce critère est très utile. Mais il est indispensable qu'il soit corrigé en tenant compte des propriétés de l'origine et des points d'articulation du groupement. Si la traduction même d'un groupement dépend de ces éléments, alors le degré d'indéterminisme aussi : par exemple, selon qu'un programme aura un point intermédiaire imposé ou non, sa traduction sera différente mais aussi son degré d'indéterminisme. Par contre, si un groupement est traduit de la même façon selon que son point terminal est une donnée ou une inconnue, il serait souhaitable que son degré tienne compte de cette différence, et éventuellement aussi du fait que l'inconnue soit propre ou non (ce qui n'est pas toujours un critère parfait à cause des filtres hétéro-inconnue). Il y a deux solutions pour effectuer cette amélioration:

- ou décider que l'on attache aux groupements non pas des degrés mais des procédures de calcul des degrés, tenant compte d'autres facteurs,

- ou, aux degrés liés aux groupements seuls, appliquer des procédures d'ajustement en fonction du contexte.

Ces deux solutions peuvent coexister. Les instaurer pousserait à enrichir la gradation en degrés d'indéterminisme.

Enfin, il faut augmenter le degré d'un groupement élémentaire s'il est affecté d'un opérateur unaire.

Conjecture H : Il nous semble vraisemblable que l'on puisse émettre la conjecture suivante : lorsqu'un groupement traduisible g est le début d'un groupement traduisible g', alors : $d^\circ\ g > d^\circ\ g'$.

Cette conjecture tend à favoriser le traitement de groupements les plus longs possibles. Ne seront jamais candidats ensemble deux groupements dont l'un est le début de l'autre, seul le plus long étant candidat. Cela raccourcit l'étape de choix.

Cependant, il est nécessaire d'avoir un accès facile à tous les groupements plus courts que les groupements candidats à cause de l'éventualité d'un raccourcissement à effectuer.

Il sera simple de donner à l'utilisateur une possibilité de choix entre l'adoption et le rejet de la conjecture H.

Autres critères :

Nous venons d'évoquer le critère qu'est le déterminisme de chaque groupement candidat, compte tenu éventuellement de la nature de son origine et de ses points d'articulation. Toujours dans le domaine du déterminisme, un critère intéressant peut être lié à l'environnement de chaque candidat :

- ce qui précède le candidat dans sa famille d'arborescences peut être pris en compte ; si deux candidats ont comme origine une inconnue, on préférera celui dont l'inconnue risque d'avoir l'ensemble de valeurs le plus petit au moment où le candidat sera activé ; mais ceci n'est pas souvent facile à déterminer. Un critère envisageable bien

qu'imparfait serait le moment où le groupement est devenu candidat : plus tôt il est devenu candidat plus son antécédent doit être déterministe.

- ce qui suit le candidat dans sa famille peut aussi être pris en compte : il n'est pas très intéressant de choisir un candidat si l'on sait que ses successeurs seront de mauvais candidats ou plus exactement que son "meilleur"successeur sera mauvais. Ceci est difficile à estimer car il faut anticiper d'une ou plusieurs étapes, ce qui non seulement est compliqué mais prend du temps.

Mis à part le déterminisme, un critère important peut être le <u>côut</u> -en temps, en place ou en une autre manière- du déroulement (à l'exécution) des procédures de recherche correspondant au groupement considéré. Mais ce critère doit être utilisé avec prudence car il faut rapporter le coût au "travail accompli". Si cependant on décide d'utiliser un tel critère, il faut attacher à chaque feuille de la ramification des groupements traduisibles non seulement un degré d'indéterminisme mais aussi un coût qui sera une autre "indication d'efficacité" (cf § VII-4-2).

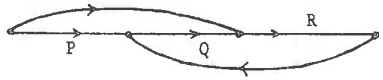
D'autres critères pourront être envisagés,...le hasard par exemple. Dans l'implémentation effectuée, à égalité d'indéterminisme on choisit le plus ancien groupement candidat.

<u>Changement de sens</u> : appuyons notre propos sur un exemple. Soit le motif aRcSTb . Les extrémités sont a et b ; supposons que seuls R, S, T, R^{-1}, S^{-1}, T^{-1} aient une traduction(peu importe que R^{-1}, S^{-1}, T^{-1} soient des groupements traduisibles, c'est-à-dire aient une traduction propre, ou qu'ils se traduisent par les réciproques des traductions de R, S, T). Au début, seront candidats les arcs $a \xrightarrow{R} c$ et $b \xrightarrow{T^{-1}}$. On

pourrait penser qu'il est dommage que $c \xrightarrow{R^{-1}} a$ ne soit pas candidat car il est possible qu'il ait de meilleures performances que sa réciproque (si R^{-1} est un groupement traduisible). Nous avons choisi de ne pas considérer ce cas : si l'on veut que $c \xrightarrow{R^{-1}} a$ soit candidat, il suffit d'imposer que c soit aussi une extrémité.

VII-4-5- "EXISTE-T-IL ENCORE DES CANDIDATS ?"

Il se peut que, bien que tous les arcs n'aient pas été traités, il ne reste plus de candidats après une phase de raccourcissement. Illustrons ce cas par un exemple élémentaire : il se peut qu'une relation de la structure logique n'ait pas,non combinée à une autre, de traduction en PHYLOG ; il en est ainsi quand on veut interdire un certain accès ; supposons que R soit dans ce cas, ainsi que R^{-1}, et qu'à un instant donné les candidats soient PQ et R^{-1} Q^{-1}.



Si R^{-1} Q^{-1} est choisi, l'arc étiqueté Q sera traité, donc PQ sera raccourci et deviendra P ; cela se passera bien. Mais si PQ est choisi, R^{-1} Q^{-1} ne pourra pas être raccourci et PQ n'aura pas de descendant. Il n'y aura donc plus de candidat.

Ce cas ne peut intervenir que si une relation n'a de traduction que lorsqu'elle est combinée à une autre.

VII-4-6- RETOUR EN ARRIERE . PLANCHES DE SALUT.

Le retour en arrière, d'une part pose des problèmes quant à son déroulement car il y a de nombreuses manières de concevoir celui-ci, mais aussi oblige à garder de nombreux renseignements sur l'historique des choix effectués. Nous parlerons de ces renseignements au paragraphe VII-4-7.

Examinons le problème du déroulement. Trois questions se posent : d'abord à quelle étape du choix revenir ? Ensuite, que remettre en cause ? Enfin, doit-on éventuellement continuer les remises en cause et jusqu'où ?

A quelle étape du choix revenir ?

Plusieurs solutions sont possibles : la plus simple est de revenir sur le choix effectué le dernier. Nous verrons, au sujet de la réitération des remises en cause, que ce procédé a le très gros avantage d'être exhaustif. Mais il existe certainement des procédés qui permettent, toujours ou dans certaines catégories de cas, d'avoir une meilleure probabilité d'atteindre vite une sortie d'impasse. On peut penser par exemple à revenir au dernier choix où c'est un prédécesseur du dernier groupement choisi qui avait été élu; ou au contraire au dernier choix où ce n'était pas un prédécesseur de ce groupement ; ou au premier choix où un groupement peu déterministe a été choisi ;.... Il conviendra de chercher et tester des heuristiques à ce niveau. Pour l'instant, c'est au dernier choix effectué que l'on revient.

Que remettre en cause ?

Lorsqu'on revient sur un choix, on peut se contenter de refaire un choix parmi les anciens candidats, en neutralisant celui qui avait été choisi. Mais on peut aussi penser que les

groupements qui sont des débuts de ce dernier n'ont plus de raison de n'être pas candidats : on rétablit, dans ce cas, comme candidat le plus long d'entre eux (en conservant donc la conjecture H).

Comment réitérer les remises en cause ?

Lorsqu'un retour en arrière aboutit de nouveau à un échec, on fait un nouveau retour en arrière. Nous avions songé à ne pas remettre en cause des choix déjà modifiés une fois, mais à remettre en cause nécessairement un choix précédent ; cela aurait eu l'avantage sans doute d'être la plupart du temps plus efficace mais n'aurait pas été exhaustif. Nous avons préféré une méthode plus sûre : ce n'est qu'après avoir épuisé toutes les possibilités d'une étape de choix (on pourrait cependant trouver des heuristiques permettant de prévoir que l'on n'a plus aucune chance de sortir de l'impasse en revenant sur le même choix) que l'on revient sur une étape précédente. Et l'échec sera complet lorsque même un retour sur la première étape de choix effectuée aboutira à une impasse. Dans ce cas, il y aura cependant la dernière "planche de salut".

Il se peut que la situation puisse être débloquée en autorisant un groupement à contenir un arc déjà traité ; c'est ce qui est fait dans cette "planche de salut". En fait, elle n'est activée que lorsque l'opération suivante a été tentée : on cherche quels sont les arcs non traités ; si leur origine ou leur but ne fait pas partie du front, on les ajoute a celui-ci, si bien que ces arcs deviennent candidats, ou même des groupements commençant par ces arcs ; on retourne à l'étape de choix.

VII-4-7- MISE A JOUR DES TABLES.

Nous avons déjà signalé l'existence d'une liste des groupements candidats. En raison de la conjecture H, nous avons adopté la structure suivante pour cette liste. C'est en fait une liste de piles de groupements. Chaque candidat est le sommet d'une pile constituée de ses débuts qui sont aussi des éléments de Q.

Ainsi les raccourcissements seront très aisés et un renoncement à la conjecture H changerait peu la gestion : tous les éléments de chaque pile seraient candidats à la place de son seul sommet. Chaque candidat est accompagné, dans la liste, de l'opérateur unaire qui l'affecte si c'est le cas, et de ses points d'articulation.

Par ailleurs, pour les nécessités du retour en arrière, il est utile de sauvegarder à chaque étape une "photographie" du contexte de cette étape. Ce contexte comprend principalement :

- la liste des candidats,
- la liste des arcs traités et des inconnues ayant reçu une valeur.

La "photographie" est exprimée principalement par :

- des pointeurs vers la liste de candidats,
- une ligne du tableau booléen EMPLOI, ligne qui contient, pour chaque arc du pochoir, "Vrai" s'il a été traité, "Faux" sinon, ceci au moment de l'étape considérée ; de même pour les inconnues.

L'ensemble des "photographies" est géré en une pile de sauvegarde, SAUV, dans laquelle une ligne (une "photographie") est ajoutée chaque fois qu'un nouveau choix est effectué. Cette ligne contient aussi l'indication du résultat du choix, sous la forme d'un pointeur vers le candidat choisi. Lors d'un retour en arrière, on fait redescendre la pile SAUV.

Ainsi, le dernier état de la pile SAUV contient la liste des groupements finalement choisis et donc sert de point de départ à la <u>génération du programme indéterministe</u>.

Ce module utilise, en plus de la pile SAUV et de la liste des groupements candidats, la traduction de LOG en PHYLOG (y compris la traduction des éléments de Λ_1).

Il génère, pour chaque groupement, le texte de sa traduction en PHYLOG sous la forme d'appels de modules (éventuellement affectés d'opérateurs unaires), après avoir mis en place dans ces appels les noms des arguments et but voulus.

VII-5- QUALITE DU PROGRAMME OBTENU.

Comme nous l'avons laissé entendre dans le paragraphe IV-2-3, on ne peut pas être sûr qu'après une succession de choix locaux la stratégie trouvée soit finalement la meilleure. Ce genre de problème n'a d'ailleurs pas été résolu pour des langages comme Snobol [27, 31] où il se pose pourtant de façon plus simple. Nous essayerons d'étudier quels sont les risques de laisser échapper les bonnes stratégies et si l'utilisation d'heuristiques de choix plus globales que celles utilisées, sans être trop coûteuse, peut diminuer ces risques (il y aura alors des compromis à chercher entre coût de la phase de recherche de stratégie et performances du programme exploitable).

Une autre façon d'aborder le problème sera de chercher quelles propriétés doivent posséder les motifs pour donner lieu à des stratégies efficaces et pouvant être trouvées par des choix locaux.

Si nous arrivons à des conclusions décevantes en ce qui concerne la stratégie trouvée, une solution sera peut-être de renoncer aux choix locaux et de traiter le problème de façon essentiellement globale par des méthodes inspirées du PERT.

VIII - EXECUTION DU PROGRAMME MAQUETTE

o00o

Dans ce chapitre, nous allons voir comment, à partir du programme maquette, on obtient un programme exécutable (nous l'appellerons quelquefois le programme objet déterministe).

Ce programme contient bien sûr des appels des procédures de recherche mais assure également la gestion des valeurs des inconnues ; c'est ce dernier point qui le fait essentiellement différer du programme maquette.

Comme nous l'avons annoncé au paragraphe IV-3, nous avons choisi le mode global pour l'exécution du programme. Ainsi donc, à chaque étape de la recherche, toutes les solutions sont recherchées. A chaque module de recherche de la maquette, correspond un bouclage sur un appel de la procédure de recherche homonyme, appel précédé de la mise en place des paramètres et suivi de celle des résultats. Si, cependant, dans la partie "cadrage", il est précisé que seule une solution est recherchée, ce n'est qu'à la dernière étape que l'on ne cherchera qu'une solution.

Nous allons voir en quoi consiste le programme objet exécutable (VIII-1) puis comment il est obtenu à partir du programme maquette (indéterministe) par le programme TRADID (TRAduction d'Indéterministe en Déterministe) (VIII-2).

VIII-1- LE PROGRAMME OBJET EXECUTABLE.

VIII-1-1- GENERALITES.

Dans ce paragraphe, nous allons donner un aperçu général du programme exécutable sans faire intervenir les problèmes techniques de gestion des résultats. Ces problèmes, que nous présenterons au paragraphe VIII-1-2, obligeront à revenir sur certains points du présent aperçu ; mais nous avons préféré séparer les difficultés. A partir du paragraphe VIII-1-3, nous décrirons plus précisément le programme.

Nous avons expliqué au paragraphe V-5-3 que l'évaluation d'un filtre s'effectuait de gauche à droite, chaque motif Mi s'évaluant compte tenu de sa source SOURCE (Mi) qui est le résultat de son contexte total à gauche, son CTG (cf § V-5-1-2). Cette évaluation de gauche à droite est d'ailleurs la raison d'être de la redondance que nous avons instituée entre les moyens d'expression offerts par les filtres d'une part et les motifs d'autre part, comme nous l'avions remarqué au paragraphe IV-1-5.

Dans tout ce chapitre, nous nous situons au point de vue de l'évaluation d'un filtre F ayant n inconnues. Pour chaque motif Mi, nous appelons Xi l'ensemble des inconnues figurant dans Mi ou dans son CTG; généralisons aux sous-filtres.

Signalons que dans la spécification du programme exécutable notre soucis constant a été de traiter les cas simples de façon simple, afin de ne pas les pénaliser par une complexité potentielle.

VIII-1-1-1- Source et résultat d'un motif.

Lorsque commence l'évaluation d'un motif Mi, certaines inconnues possèdent déjà des valeurs, qui constituent SOURCE(Mi). Ces valeurs sont considérées comme des données pour l'évaluation de Mi. Parmi ces valeurs, celles pour lesquelles on ne trouve aucune occurrence de Mi seront éliminées et ne figureront donc pas dans le résultat de Mi ; les autres y figureront ainsi que des valeurs des inconnues propres de Mi.

Nous venons de parler ici des valeurs des inconnues comme si elles étaient indépendantes les unes des autres, ce qui n'est évidemment pas le cas. Nous avons d'ailleurs bien dit au paragraphe V-5-2 que le résultat RESU(Mi) d'un motif Mi au sein du filtre F était un ensemble de n-uples dont la $k^{ème}$ position est occupée : par une valeur de x_k si x_k figure dans Mi ou son CTG (c'est-à-dire est un élément de Xi), et par "non déterminé" sinon. Cette forme sous-entend que toutes les valeurs des inconnues sont liées. Nous verrons cependant au paragraphe VIII-1-2-2 que l'on peut nuancer ce point de vue pour améliorer l'implémentation : certaines inconnues sont nécessairement liées, d'autres pas. Mais tel n'est pas notre propos ici.

VIII-1-1-2- Les inconnues locales.

Au cours de l'évaluation d'un motif, des inconnues locales peuvent intervenir : nous les avons appelées y_1, y_2,... et avons ajouté x_o à leur ensemble. Elles jouent exactement le même rôle que les autres inconnues à part que leur portée se réduit au motif, c'est-à-dire que le résultat final du motif ne les contient pas. Mais un résultat intermédiaire peut en contenir. Nous en tiendrons compte dans la gestion des résultats.

VIII-1-1-3- Constitution du programme objet exécutable.

La plupart des constituants du programme objet indéterministe donnent naissance à une séquence du programme objet déterministe. Cette séquence comprend : pour les appels de modules de recherche, des appels de la procédure correspondante séparés par des ordres de gestion des résultats ; pour les demandes de réunion, un appel de la procédure de réunion ; pour les fins de motifs, quelques ordres de gestion de l'espace des résultats ; aucune intersection n'est jamais effectuée. Nous parlerons au paragraphe VIII-1-1-5 de tout ce qui concerne les résultats booléens.

Nous avons déjà dit qu'un appel de module de recherche comprend :

- un nom de relation de PHYLOG (c'est-à-dire un nom de procédure de recherche), éventuellement affecté d'un opérateur ;

- une origine, formée d'un ou plusieurs arguments ; ceux-ci peuvent être soit des données élémentaires (éléments de $\wedge_1$), soit des inconnues globales ou locales (US est considéré comme une inconnue glabale) ; les inconnues peuvent ou non avoir été déjà évaluées ;

- un but, qui se présente comme un argument.

Voyons, selon la nature des arguments et du but de l'appel de module considéré, ce que devra comporter la séquence correspondante du programme objet déterministe.

. Selon les arguments :

- S'il y a des arguments qui sont des inconnues non encore évaluées, on devra leur "créer" des valeurs par ailleurs (cf plus loin § VIII-1-3-1 et 2, module A2) ; on se ramènera ainsi au cas des inconnues déjà évaluées ; un cas particulier est celui où R est une relation binaire et est affectée par l'opérateur "réciproque", le but étant soit une donnée soit une inconnue déjà évaluée ; dans les paragraphes VIII-1-1 et VIII-1-2, nous ne nous intéresserons plus à ces créations de valeurs que nous supposerons effectuées.

- la procédure sera appelée pour chaque jeu de valeurs possibles pour les arguments qui sont des inconnues ; pour tous ces appels, les autres arguments conserveront leur valeur donnée.

Exemple :

$R(x_2, 4, x_1$; but)

Supposons que le couple (x_1, x_2) ait les valeurs (2,a) et (5, b).

Les deux appels de la procédure R seront :
R(a,4,2 ; but) et R(b,4,5 ; but).

. Selon le but :

- si le but est donné, la procédure répondra seulement "succès" ou échec" à chaque appel (compte tenu de l'opérateur affectant éventuellement R) ; seuls seront conservés les jeux de valeurs des arguments pour lesquels la réponse est "succès".

- si le but est une inconnue x_k non encore évaluée, à chaque appel de la procédure donc à chaque jeu de valeurs des arguments correspondra un ensemble, éventuellement vide, de

valeurs de x_k ; les jeux de valeurs pour lesquels l'ensemble est vide seront rejetés ; pour les cas où R est affecté d'un opérateur, (cf § VIII-1-3-3), il faudra ici faire appel à des valeurs "par ailleurs" pour x_k (cf § VIII-1-3-1 et 2, module A2).

- si le but est une inconnue x_k déjà évaluée, chaque appel de la procédure donnera un ensemble de valeurs pour x_k, ensemble dont on ne gardera que l'intersection avec l'ensemble des valeurs de x_k compatibles avec celles des arguments.

Exemple :

$R(x_2,4,x_1 ; x_4)$

Supposons que (x_1,x_2,x_4) ait les valeurs

(2,a,7), (2,a,9), (5,b,1).

Si l'appel $R(a,4,2;x_4)$ donne les valeurs

1,7,10 comme résultats pour x_4 et si

l'appel $R(b,4,5 ; x_4)$ donne 8 et 9, alors le seul triplet de valeurs des inconnues qui soit retenu dans le résultat du module sera (2,a,7).

VIII-1-1-4- Vue simpliste de la gestion de l'espace des résultats.

Nous aurions pu adopter une gestion assez simple pour les résultats en considérant que tout résultat final de motif ou de sous-filtre est un ensemble de n-uples (si n est le nombre d'inconnues du filtre, y compris éventuellement US), et que tout résultat intermédiaire d'un motif Mi contenant n_i inconnues locales (y compris éventuellement x_o) est un ensemble de $n+n_i$-uples. Mais d'une part les n-uples et les $n+n_i$-uples auraient souvent été "creux" et d'autre part leur nombre aurait été inutilement grand dans le cas d'inconnues non liées (cf § VIII-1-2-1 et 2). Nous avons préféré adopter une gestion plus compliquée mais plus économique en place (et probablement en temps) qui consiste à scinder certains n-uples.

Avant d'étudier cette gestion en détail, voici ce que l'on peut dire dès maintenant au sujet de la durée de vie des résultats :

- l'élaboration du résultat de Mi ne doit pas modifier SOURCE(Mi) ; celle-ci doit subsister jusqu'à ce qu'elle soit "déchue" (cf § VII-1-2).

- par contre lorsqu'un résultat intermédiaire de motif a été utilisé, il peut être détruit car il ne sera pas réutilisé : ce qui reste utile en lui se retrouve dans le nouveau résultat intermédiaire ; en effet, l'exécution successive des différents modules d'un même motif ne fait qu'affiner (en éliminant les valeurs qui se révèlent non pertinentes) et enrichir (en évaluant de nouvelles inconnues)

la source du motif, de façon linéaire, séquentielle, jusqu'à arriver au résultat final du motif.

- lorsqu'une réunion est exécutée entre les résultats de deux sous-filtres, aucun des deux sous-filtres ne peut plus être contexte à gauche d'un motif ; leurs résultats peuvent donc être effacés.

- lorsqu'un résultat est déchu de son pouvoir de source, on peut aussi récupérer sa place ; de plus, rien de ce qui a été stocké après lui, sauf bien sûr le dernier résultat trouvé, ne peut être réutilisé ; toute la place correspondante peut donc également être récupérée.

VIII-1-1-5- Les résultats booléens.

Pour gérer les résultats booléens, le programme utilisera d'une part une variable booléenne BOOL et d'autre part une pile booléenne PILBOOL.

Chaque exécution de module de recherche se soldera par une valeur pour BOOL. Il comportera dans sa partie finale un test de cette variable : si BOOL est "vrai" (succès), alors on continuera à dérouler normalement le programme. Si BOOL est "faux",on pourra tout de suite affirmer que tout l'actuel EVAL(Mi) se solde par un échec. On fera donc un saut à l'étiquette Fi qui marque la fin de la séquence relative à Mi ; le résultat final de Mi sera tout naturellement BOOL : = "Faux".

En plus des étiquettes Fi qui sont utilisées pour les traitements booléens internes aux motifs, le programme indéterministe comporte des jalons (cf § VII-1-2) destinés aux traitements booléens au niveau du filtre. Ce sont des étiquettes Ep et des pseudo-intructions de la forme "B(Ep)" ou "Empiler". Les étiquettes seront reproduites fidèlement dans le programme exécutable.

La partie de programme exécutable relative à une pseudo-instruction "B(Ep)" sera :

si ¬BOOL alors allera Ep.

Cet ordre, rappelons-le, sert à déclarer en échec une conjonction (sous-filtres liés par et) si le premier se solde par un échec .

La partie de programme exécutable relative à une pseudo-instruction "Empiler" sera :

empiler BOOL sur PILBOOL.

Cette séquence interviendra au moment où l'on aura évalué le premier sous-filtre d'une disjonction (ou) et servira à préparer l'exécution de celle-ci. Chaque module exécutant une réunion (cf § VIII-1-4) testera le sommet de PILBOOL (pour le premier terme du ou) et BOOL (pour le deuxième) puis enlèvera le sommet de PILBOOL. De plus il placera son résultat booléen en BOOL.

VIII-1-2- GESTION DES VALEURS DES INCONNUES.

VIII-1-2-1- Principe.

Nous avons dit (§ V-5-2 et VIII-1-1) que le résultat RESU(Mi) d'un motif Mi figurant au sein d'un filtre F ayant n inconnues $x_1, \ldots, x_n$ (x_n étant éventuellement US) est un ensemble de n-uples. Dans chaque n-uple, la $k^{ème}$ position est occupée par une valeur de x_k, si x_k est un élément de X_i, ou par un tiret signifiant "toute valeur possible" (ou "non déterminé.") dans le cas contraire. Au cours de l'évaluation du motif, chaque résultat est même un ensemble de $n+n_i$-uples (si Mi a n_i inconnues locales, dont éventuellement x_o).

Il serait dommage de stocker véritablement des $n+n_i$-uples car de nombreuses positions seraient occupées par des tirets. Nous avons donc décidé de stocker au maximum des m_i+n_i-uples($m_i \leq n$) correspondant aux m_i éléments de X_i.

Mais surtout, les différentes inconnues d'un filtre, qu'elles figurent ou non dans un même motif, n'ont pas toujours besoin d'être associées en ce qui concerne leurs valeurs. Expliquons cela sur un exemple simple. Considérons le motif a $<R\ x_1, S\ x_2>$. Les valeurs trouvées pour x_1 sont indépendantes de celles trouvées pour x_2, contrairement à ce qui se passerait pour le motif a R x_1 S x_2. Stocker des couples plutôt que des valeurs isolées n'apporte ici aucune information; or il vaut mieux stocker par exemple 20 valeurs pour x_1 et 30 pour x_2, soit 50 valeurs, que 600 couples !

Nous avons donc introduit la notion d'inconnues liées et basé la gestion des résultats sur elle.

VIII-1-2-2- Inconnues liées.

Lorsque deux inconnues figurent dans un même appel de module du programme objet indéterministe, leurs valeurs deviennent dépendantes les unes des autres.

Exemple : F = M1 et M2 et M3

avec M1 = $a<R\ x_1, S\ x_2>$

M2 = $x_1 T x_2$

M3 = $x_1 V x_3$

Supposons que les relations R, S, T, V, appartiennent aussi à PHYLOG.

- Pour évaluer M1, deux modules seront appelés (peu importe dans quel ordre) : $R(a;x_1)$ et $S(a;x_2)$.

A la fin de l'évaluation de M1, les valeurs de x_1 et de x_2 n'auront aucun lien : il y aura seulement un ensemble de $\bar{x}_1$ valeurs de x_1 et un ensemble de $\bar{x}_2$ valeurs de x_2 pour lesquels M1 aura des occurrences dans la donnée. A ce stade, il est possible de stocker ces deux ensembles indépendamment (il y aura $\bar{x}_1 + \bar{x}_2$ valeurs à stocker alors qu'il y aurait $\bar{x}_1 \times \bar{x}_2$ couples).

- Pour évaluer M2, le module $T(x_1; x_2)$ sera appelé et, à chaque valeur de x_1, correspondra un ensemble de valeurs de x_2. Il sera donc, à partir de ce moment là, obligatoire de coupler les valeurs de x_1 et x_2.

- Cette obligation d'associer les inconnues est "contagieuse", transitive. Après l'évaluation de M3, non seulement x_3 sera associé à x_1 mais en plus il le sera à x_2 puisque celui-ci est lui-même lié à x_1.

Définition pour un filtre sans ou.

Nous définirons à chaque étape du programme indéterministe la relation L=" - est liée (à ce stade) à -" entre les inconnues comme la fermeture reflexo-transitive de la relation : "-figure dans le même appel de module précédent que-" ; précédent signifie ici : ou bien précédent dans la séquence correspondant au motif en cours ou bien figurant dans une séquence de son CTG.

Cette définition s'applique non seulement aux inconnues globales mais aussi aux inconnues locales. La relation L est une relation d'équivalence.

Cas de ou : considérons l'exemple suivant :

F = M1 ou M2

avec : M1 a deux inconnues non liées x_1 et x_2
M2 de même.

Supposons que le résultat de M1 soit :

x_1 peut prendre les valeurs 2, 7

x_2 a la valeur 12

et que le résultat de M2 soit :

x_1 peut prendre les valeurs 2, 9, 8

x_2 peut prendre les valeurs 15, 19.

Si l'on conservait l'indépendance de x_1 et x_2 après la réunion, cela reviendrait à dire que le résultat de F est le suivant :

pour x_1 : 2, 7, 8, 9

pour x_2: 12, 15, 19.

Mais si l'on prend x_1=7 et x_2 = 19 ni M1 ni M2 n'aura d'occurrences, donc a fortiori F n'en aura pas.

Il est aisé de généraliser, et nous devons poser la proposition suivante :

Proposition : Si f_1 et f_2 sont deux sous-filtres dont les inconnues (y compris celles de leur CTG) constituent respectivement les ensembles x_1 et x_2, alors les inconnues de $f_r=f_1$ ou f_2 constituent $Xr=X_1 \cup X_2$ et sont toutes liées entre elles après l'évaluation de f_r. Nous dirons qu'elles figurent dans la même disjonction (f_r).

Définition de L dans le cas général :

A chaque étape du programme indéterministe, L ="- est liée (à ce stade) à -" est la fermeture réflexo - transitive de la relation : " - figure précédemment dans un même appel de module ou dans une même disjonction que - ".

VIII-1-2-3- Base de nos choix concernant la gestion des valeurs.

La présentation et la gestion des résultats va être une conséquence des faits suivants :

-1- on a intérêt à stocker indépendamment les inconnues non liées pour qu'il n'y ait pas d'effet combinatoire inutile, d'une part au point de vue de l'encombrement, d'autre part en ce qui concerne les appels de procédures de recherche : il n'est pas facile de s'assurer que l'on n'appelle pas deux fois avec les mêmes arguments la procédure de recherche considérée (cf § VIII-1-6), et c'est d'autant moins facile que ces arguments sont davantage répétés en mémoire. On ne pourra cependant pas empêcher qu'il subsiste un certain effet combinatoire.

-2- Le nombre d'inconnues locales varie d'un motif à l'autre.

-3- Il y a un effet de pile sur les résultats, certains -le plus souvent les derniers créés- devenant caducs à certains moments et leur place pouvant donc être récupérée (cf § VIII-1-1-4).

-4- Les diverses inconnues ont des types différents dont nous n'avons guère parlé car nous n'avons pas traité du langage de cadrage (cf § IV-1-1-). Cela entraîne que les longueurs peuvent être très différentes.

VIII-1-2-4- Résultats final et intermédiaire d'un motif.

Plaçons-nous ici à un point de vue statique. Le résultat final de l'évaluation d'un motif Mi n'est pas nécessairement un ensemble de n-uples (si F a n inconnues) mais est constitué d'un certain nombre d'ensembles $\cup_i^j$ de n_i^j-uples : un par classe d'équivalence pour L (à la fin de l'évaluation de Mi) de l'ensemble Xi. Cette décision est une conséquence du point 1 du paragraphe précédent.

Exemple 1 :

Reprenons l'exemple du paragraphe VIII-1-2-2, en supprimant M2. Le résultat final de l'évaluation de M3 est composé de :

- l'ensemble $\cup_i^1$ constitué de couples ($n_i^1=2$) de valeurs de x_1 et x_3 qui sont liées ;

- l'ensemble $\cup_i^2$ constitué de singletons ($n_i^2=1$) de valeurs de x_2.

Pour un résultat intermédiaire de motif, c'est-à-dire pour le résultat d'un appel de module de ce motif, on peut affirmer la même chose sauf qu'aux inconnues globales s'ajoutent les inconnues locales déjà intervenues à cet instant dans l'évaluation du motif :

Exemple 2 :

soit F = M1 et M2 avec M1 = a P x_1 Q x_2

M2 = b < RSx_2,TV x_3 > ;

si la séquence de modules de M2 est T(b ; y_1), V(y_1;x_3), R(b ; y_2), S(y_2 ; x_2), alors avant l'exécution du troisième module le résultat est composé de :

- l'ensemble $\cup_i^1$ constitué de couples de valeurs de x_1 et x_2.

- l'ensemble $\cup_i^2$ constitué de couples de valeurs de x_3 et y_1.

Après l'exécution du dernier module correspondant à un motif, les places occupées dans les résultats par les valeurs d'inconnues locales n'ont plus de raison d'être. Nous verrons (§ VIII-1-2-7) quand et comment on récupère ces places ; c'est une question de gestion d'espace libre.

VIII-1-2-5- Mémorisation des ensembles $\cup_i^j$ de k-uples de valeurs.

Quatre questions se posent pour déterminer la façon de mémoriser chaque ensemble de k-uples :

. comment mémoriser chaque valeur ?
. comment mémoriser chaque k-uple ?
. comment mémoriser chaque ensemble de k-uples ?
. comment mémoriser les divers ensembles de k-uples (pour différents k) ?

Etudions chacune de ces questions, en commançant par la deuxième.

1) Comment mémoriser chaque k-uple ?

Deux possibilités se présentent a priori pour stocker les diverses positions d'un k-uple : ou bien de façon consécutive ou bien sous forme de liste chaînée. Or c'est par son rang dans le k-uple que se fera l'accès à chaque position :

en effet, étant donné un numéro d'inconnue, on cherchera à atteindre ses valeurs en accédant par son rang à la position lui correspondant dans chaque k-uple (d'un certain ensemble de k-uples). Il est donc naturel d'opter pour le stockage consécutif des différentes positions de chaque k-uple.

2) Comment mémoriser chaque valeur ?

Puisque les différentes inconnues ont des longueurs différentes (cf 4 de VIII-1-2-3), et puisque d'un ensemble de k-uples à un autre la même position ne correspond pas à la même inconnue (donc on ne peut pas avoir une correspondance fixe entre rang et longueur), il semble normal d'utiliser des adressages indirects. Le seul problème est de savoir si l'adressage sera complètement ou partiellement indirect, c'est-à-dire si toutes les valeurs seront remplacées par des pointeurs vers les valeurs ou si les valeurs les plus courtes seront placées directement dans les k-uples.

Cela sera résolu à l'usage (lorsqu'on écrira le programme, ce qui n'a pas encore été fait). Pour l'instant et pour uniformiser l'exposé, nous supposons que l'adressage est complètement indirect. De toute façon, toutes les positions de tous les k-uples auront la même longueur.

-3 et 4- Comment mémoriser chaque ensemble de k-uples et les différents ensembles de k-uples(pour divers k) ?

Comme les diffétents k-uples d'un ensemble doivent être considérés séquentiellement, il serait possible de les chaîner. Ce mode de stockage n'aurait d'intérêt que s'il permettait de récupérer des "trous" laissés par des informations devenues caduques. Or il est rare qu'un ensemble de k-uples après utilisation dans un module reste un ensemble de k-uples (qui serait alors inclus dans le précédent, laissant éventuellement des places libres) : cela n'arrive que lorsque le module ne fait intervenir que des inconnues impliquées dans cet ensemble de k-uples. Le plus souvent, l'ensemble de k-uples se transforme en un ensemble de k'-uples (k' > k). C'est donc plutôt entre les ensembles de k-uples (pour divers k) qu'à l'intérieur de chacun d'eux que se pose le problème de la récupération.

La mémorisation de chaque ensemble de k-uples et la mémorisation des ensembles de k-uples pour divers k doivent donc se résoudre de la même façon : ou bien être toutes deux effectuées par chaînage ou être toutes deux consécutives.

Dans le cas d'une mémorisation par chaînage,on constituerait un espace libre chaîné que l'on enrichirait quand une zone deviendrait caduque et où l'on prendrait de la place à chaque naissance d'information. Mais cette gestion serait lourde car les informations (k-uples pour différents k) à stocker ont des longueurs variées

(raisons 1, 2 et 4 de VIII-1-2-3). De plus, il n'y aurait pas de possibilités de partages de fins de listes.

Nous avons préféré ici aussi la solution d'un stockage consécutif des k-uples et des ensembles de k-uples. Pour récupérer de la place (point 3 de VIII-1-2-3), il faut donc généralement recopier les informations stockées. Nous aurons à choisir entre trois solutions pour la gestion de l'espace libre :

- ou ne jamais récupérer de place tant qu'on n'arrive pas à saturation et, là seulement, faire fonctionner un "ramasse-miettes" avec tassement ;

- ou récupérer de la place chaque fois que l'on peut mais cela exige de nombreuses recopies, ce qui finit par être trop onéreux en temps ;

- ou récupérer de la place de temps en temps, en particulier à la fin de l'évaluation de chaque motif (d'autant plus que les inconnues locales deviennent caduques à ce moment là et que leur place peut être récupérée) et/ou lors de l'exécution d'une réunion. C'est une solution de ce type que nous adopterons, combinée à la première : à chaque instant il pourra subsister des trous, c'est-à-dire des places devenues inutilisées mais non récupérées (sources devenues caduques) ; en cas de saturation, on fera appel à un ramasse-miettes.

VIII-1-2-6- Synthèse et conséquences des paragraphes précédents.

Nous avons donc choisi à tous les niveaux un rangement consécutif. Nous avons procédé ensuite de la façon suivante .

Nous avons organisé la mémoire disponible pour les résultats en un tableau unidimensionnel W. Ce tableau est viruellement décomposé au fil des besoins en sous-tableaux Wj à deux dimensions. Chaque Wj contient un ensemble de k-uples $\cup_i^\ell$, de la façon suivante :

- chaque ligne contient un k-uple
- chaque colonne correspond à une inconnue.

Cependant cette décomposition n'apparaîtra pas dans les adressages, qui se feront toujours dans W.

A chaque instant de l'évaluation d'un filtre, de nombreux Wj peuvent coexister. Plaçons-nous à un instant donné. Pour chaque motif Mi en cours d'évaluation ou dont le résultat est encore d'actualité, existent un certain nombre d'ensembles $\cup_i^\ell$ de k-uples, chacun stocké dans un Wj différent. Chaque inconnue de Xi a ses valeurs dans un (et un seul) des Wj.

Ce qu'il est intéressant de noter dès maintenant est que cette correspondance peut être établie dès la traduction, par le programme TRADID, du programme objet indéterministe en programme déterministe. La seule chose qui ne soit pas connue à la traduction est l'emplacement des Wj. Voici comment est exprimée la correspondance.

TRADID entretient une pile de tableaux, PILINC, qui a la forme d'un tableau à trois dimensions. Le $p^{ème}$ étage de la pile, PILINC [p,.,.] , contient à chaque instant les renseignements relatifs au stockage dans les différents Wj des valeurs des inconnues intéressant le sous-filtre de numéro p (cf "numéro de sous-filtre", § VII-1-2). En particulier, lors du traitement de EVAL(Mi), PILINC [i,.,.] contient les renseignements relatifs aux inconnues de Xi. Au cours de ce traitement, seul cet étage de PILINC évolue (au début du traitement, il est initialisé à la valeur de l'étage qui correspond à SOURCE(Mi)). La composition de l'étage PILINC [p,.,.] est la suivante :

- PILINC [p,m,1] contient le numéro j du tableau Wj où sont stockées les valeurs de x_m ; si à l'instant considéré x_m n'est pas encore intervenu, PILINC [p,m,1] =0.

- PILINC [p,m,2] n'est plus utilisé

- PILINC [p,m,3] et PILINC [p,m,4] contiennent respectivement le rang de x_m dans chaque ligne de Wj et la longueur de ces lignes (nombre d'inconnues ayant leurs valeurs dans Wj).

Nous reviendrons au paragraphe VIII-2 sur la gestion de PILINC.

VIII-1-2-7- Utilisation du tableau W par le programme déterministe.

Les arguments d'un appel de module de recherche :

Plaçons-nous dans EVAL(Mi), au début de la séquence correspondant à un appel de module de recherche, $R(a_1,\ldots,a_u;b)$. Chaque argument qui est une inconnue a ses valeurs dans un Wj dont le numéro j est connu dès la traduction par TRADID.

Appelons $J=\{j_1,\ldots,j_v,\ldots,j_\ell\}$ l'ensemble des indices des tableaux de W où se trouvent des valeurs d'inconnues arguments du module considéré. Certains j_v correspondent à plusieurs arguments : ou bien parce que ces arguments sont identiques (la même inconnue) ou bien parce que les inconnues qui les constituent sont liées. Si b est une inconnue liée à un des arguments, ses valeurs figurent dans un des Wj_v. Nous dirons des arguments ou du but correspondant à un tableau Wj_v qu'ils sont "délivrés" par Wj_v.

L'organisation générale du module est la suivante :

Pour chaque ligne du tableau Wj_1 faire

Pour chaque ligne du tableau Wj_2 faire

.....

Pour chaque ligne du tableau Wj_v faire

.....

- Appel de la procédure de recherche R, compte tenu de b
- Constitution de la partie correspondante du tableau de résultats.

Remarque :

Nous parlerons de l'ordre des j_v au paragraphe VIII-1-3-2, module B_v.

Le tableau de résultats d'un module de recherche :

Le tableau de résultats sera un amalgame des tableaux Wj_v car chaque ligne contiendra une valeur de chacune des inconnues figurant dans les tableaux Wj_v (les ensembles d'inconnues concernées par les différents Wj_v sont disjoints) ; de plus, si b est une inconnue non liée aux arguments mais figurant déjà dans un tableau, Ww, chaque ligne du tableau de résultats contiendra aussi une valeur de chacune des inconnues de Ww. Désormais, toutes les inconnues de tous les tableaux Wj_v et Ww sont liées.

Numéro du tableau de résultats d'un module de recherche :

Le numéro r qui sera affecté au tableau de résultats Wr est le numéro suivant le plus grand numéro de Wj occupé.

Place du tableau de résultats d'un module de recherche :

Pour trouver l'endroit où stocker Wr, on pourrait chercher à récupérer la place des derniers tableaux stockés qui sont des Wj_v ou Ww et peuvent être effacés. Mais cela obligerait à recopier Wr car on le forme petit à petit, à un moment où l'on a encore besoin des arguments. Nous préférons gagner du temps au détriment de la place, quitte à faire fonctionner un ramasse-miettes en cas de besoin (cf § VIII-1-2-5 et VIII-1-5-3).

Si le module considéré est le dernier de EVAL(Mi), par contre, nous accepterons de recopier le tableau de résultats pour gagner de la place. D'abord, avant sa mise en place durable, le tableau est dépouillé des colonnes correspondant aux inconnues locales. Ensuite, si une réunion doit être effectuée, la mise en place du tableau de résultats est évitée. Enfin, si EVAL (Mi) est suivi de la déchéance d'une ou plusieurs sources la mise en place du tableau en tient compte.

Remarque :

Si le résultat de Mi comporte plusieurs tableaux et si les tableaux autres que le dernier stocké comportent des inconnues locales, leurs places ne sont pas récupérées, dans l'état actuel des choses. Nous discuterons au paragraphe VIII-1-6 de l'utilité d'effectuer cette récupération : interviennent non seulement l'encombrement mais aussi la redondance des jeux d'arguments.

Exécution d'une réunion :

Puisque toutes les inconnues figurant dans une même disjonction sont liées, le résultat d'une réunion est constitué d'un tableau unique Wr qui, la plupart du temps, ne peut pas être fabriqué directement à sa place définitive.

TRADID générera un programme d'élaboration de ce résultat dans une zone libre de W puis un programme de transfert qui permettra de récupérer de la place libre.

Le numéro du tableau constituant le résultat pourra sans inconvénient prendre la valeur du numéro, t, du plus profond tableau dont la place aura été récupérée. L'emplacement de ce tableau sera celui de Wt.

VIII-1-2-8. Exemple :

Reprenons l'exemple 2 du paragraphe VIII-1-2-2, revu dans le paragraphe VIII-1-2-4.

F = M1 et (M2 ou M3) et M4.

Enoncé du motif	Séquence de modules
M1 = a P x_1 Q x_2	$P(a;x_1)$, $Q(x_1;x_2)$
M2 = b<R S x_2, TV x_3>	$T(b;y_1)$, $V(y_1;x_3)$,
	$R(b;y_2)$, $S(y_2;x_2)$
M3 = c W x_2 Z x_3	$W(c;x_2)$, $Z(x_2;x_3)$
M4 = <d N x_4, x_1> K e	$N(d;x_4)$, $K(x_4, x_1 ; e)$

Ecrivons ci-dessous le programme objet indéterministe en numérotant les étapes :

(Début de M1, CTG=∧) $P(a;x_1)$/1/$Q(x_1;x_2)$/2/
Fin de M1/3/ (Début de M2, CTG=M1) $T(b;y_1)$/4/
$V(y_1;x_3)$/5/$R(b;y_2)$/6/$S(y_2;x_2)$/7/Fin de M2/8/
(Début de M3, CTG=M1)$W(c;x_2)$/9/$Z(x_2;x_3)$/10/
Fin de M3, réunion M2 et M3, et M1 déchu/11/
(Début de M4, CTG=M2 ou M3) $N(d;x_4)$/12/$K(x_4,x_1;e)$/13/
Fin de M4 et fin du filtre/14/

Donnons pour chaque étape l'état de W (à l'exécution) et celui de PILINC (à la traduction). La colonne "R" (pour "récupérable") contient une croix si le tableau concerné peut être effacé (c'est-à-dire est devenu inutile mais n'a pas été récupéré) ; ces croix seraient exploitées par le ramasse-miettes.

Après l'étape..,		W contient....	R	PILINC contient.... p(étage)	m(inc)	N°W (1)	Rang(2)	Longueur (3)
M1	1	W1 à 1 colonne(x_1)		1	1	1	1	1
	2	W1 à 1 colonne(x_1)	x	1	1	2	1	2
		W2 à 2 colonnes (x_1,x_2)			2	2	2	2
	3	W2 à 2 colonnes (x_1,x_2)		idem				
M2	4(4)	W2 à 2 colonnes (x_1,x_2)		1	idem			
		W3 à 1 colonne (y_1)		2	5 (5)	3	1	1
	5	W2 à 2 colonnes(x_1,x_2)		1	idem			
		W3 à 1 colonne (y_1)	x	2	3	4	1	2
		W4 à 2 colonnes (x_3,y_1)			5	4	2	2
	6	W2 à 2 colonnes (x_1,x_2)		1	idem			
		W3 à 1 colonne (y_1)	x	2	3	4	1	2
		W4 à 2 colonnes(x_3,y_1)			5	4	2	2
		W5 à 1 colonne (y_2)			6 (5)	5	1	1
	7	W2 à 2 colonnes (x_1,x_2)	(6)	1	idem			
		W3 à 1 colonne (y_1)	x	2	1	6	1	3
		W4 à 2 colonnes(x_3,y_1)			2	6	2	3
		W5 à 1 colonne (y_2)	x		3	4	1	2
		W6 à 3 colonnes(x_1,x_2,y_2)			5	4	2	2
					6	6	3	3

(1) PILINC [p,m,1] (2) PILINC [p,m,3] (3) PILINC [p,m,4]

(4) A chaque début de motif, on crée un nouvel étage dans PILINC et on l'initialise au contenu de l'étage correspondant à sa source. Ainsi, après cette étape, l'étage 2 est garni pour m = 1, 2, 5.

(5) Comme F a quatre inconnues globales, y_1 est assimilé à x_5, y_2 à x_6.

(6) W2 ne devient pas caduc car il appartient à la source de M2.

	8	W2 à 2 colonnes(x_1,x_2)		1	idem			
		W3 à 1 colonne (y_1)	x	2	1	6	1	2
		W4 à 2 colonnes(x_3,y_1)			2	6	2	2
		W6 à 2 colonnes(x_1,x_2)			3	4	1	2
M3	9	W2 à 2 colonnes(x_1,x_2)		1	idem			
		W3 à 1 colonne (y_1)	x	2	idem			
		W4 à 2 colonnes(x_3,y_1)		3	1	7	1	2
		W6 à 2 colonnes(x_1,x_2)			2	7	2	2
		W7 à 2 colonnes(x_1,x_2)						
	10	W2 à 2 colonnes(x_1,x_2)		1	idem			
		W3 à 1 colonne (y_1)	x	2	idem			
		W4 à 2 colonnes(x_3,y_1)		3	1	8	1	3
		W6 à 2 colonnes(x_1,x_2)			2	8	2	3
		W7 à 2 colonnes(x_1,x_2)	x		3	8	3	3
		W8 à 3 colonnes(x_1,x_2,x_3)						
	11	W1 à 3 colonnes(x_1,x_2,x_3)		1	idem,	devenu	inutile	
				2	1	1	1	3
					2	1	2	3
					3	1	3	3
M4	12	W1 à 3 colonnes(x_1,x_2,x_3)		1	idem			
		W2 à 1 colonne (x_4)		2	idem			
				3	4	2	1	1
	13	W1 à 3 colonnes(x_1,x_2,x_3)		1	idem			
		W2 à 1 colonne (x_4)	x	2	idem			
		W3 à 4 colonnes(x_1,x_2,x_3,x_4)		3	1	3	1	4
					2	3	2	4
					3	3	3	4
					4	3	4	4
	14	Inutile de récupérer, puisque c'est fini.						

VIII-1-3- TRAITEMENT D'UN APPEL DE MODULE DE RECHERCHE $R(a_1, a_2, \ldots, a_u ; b)$

VIII-1-3-1- Organigramme de ce traitement dans le cas où R n'est affecté par aucun opérateur.

Chaque appel de module de recherche du programme indéterministe est traduit en une suite d'appels de la procédure de recherche correspondante. Comme il n'y a pas de problème de récursivité, le plus simple est de passer les paramètres de la procédure dans des mémoires fixes (à un adressage indirect près). D'ailleurs, tout ce que nous disons dans ce chapitre est à un adressage indirect près.

Nous avons convenu de passer :

. la valeur de chaque argument a_n en ARG [n]

. éventuellement la valeur du but en ARG [0] .

Les procédures de recherche seront écrites dans ce sens. De plus, si le but est donné (en ARG [0].), elles répondront "Succès" ou "Echec", sinon elles enverront leurs résultats dans un tableau fixe, RESU, de longueur théoriquement arbitraire, en répondant "Succès" si au moins un résultat existe et "Echec" sinon. Le nombre d'éléments de RESU sera envoyé en NRESU. La réponse booléenne sera exprimée dans la mémoire BRESU par "Vrai" pour "Succès" et "Faux" pour "Echec".

Remarque : nombre de résultats, c'est-à-dire de lignes du tableau de résultats, pour un jeu d'arguments (en cas de succès) :

- Si le but est fixé ou est une inconnue liée à des arguments, sa valeur sera unique pour chaque jeu : il y aura une ligne de résultats.

- Si le but est une inconnue x_k déjà évaluée mais non liée à des arguments, il y aura au plus une ligne de résultats par valeur de x_k ; figureront dans cette ligne, en plus des valeurs des arguments et des inconnues qui leur sont liées, la valeur de x_k et la valeur de chacune des inconnues liées à x_k.

- Si le but est une inconnue non encore évaluée, il y aura une ligne par résultat de la procédure de recherche pour le jeu d'arguments considéré.

A - INITIALISATION

A1) Initialisation à "rien" de ARG [0] et mise en place, dans le tableau ARG, des arguments et éventuellement du but fixes (c'est-à-dire n'ayant pas la forme d'une inconnue).

A2) Fabrication de valeurs pour les arguments qui sont des inconnues non encore évaluées lorsqu'on le sait dès la traduction (par TRADID) ; TRADID connaît alors le mode de calcul de ces valeurs.

A3) Initialisation de compteurs et mémoires de travail.

B - DEBUTS DE BOUCLAGE, un par tableau Wj_v impliqué dans le module, c'est-à-dire par élément de J (cf § VIII-1-2-7) : à j_v correspond le début de bouclage B_v.

B_v - Pour chaque ligne de Wj_v faire
$v=1,2,\ldots,$

début$_v$ B_v1) Mise en place éventuelle de NØUVIv

B_v2) Mise en place dans ARG des valeurs, pour la ligne considérée, des arguments et éventuellement du but délivrés par Wj_v (1).

C - (Ce module se trouve donc à l'intérieur de ℓ boucles)

ACTIVATION DE LA PROCEDURE DE RECHERCHE

A la fin de son déroulement,
- le résultat booléen se trouve en BRESU
- les éventuels résultats (valeurs du but) se trouvent en RESU et leur nombre en NRESU.

H - TEST DE L'EXISTENCE ET DE LA VALIDITE DES RESULTATS

(Si le but est une inconnue x_k déjà évaluée mais non liée à un argument (il est délivré par un tableau Ww avec $w \notin J$) alors

H1) Stocker dans un tableau, AD, les indices (dans W) des débuts des lignes de Ww pour lesquelles la valeur de x_k est égale à une valeur stockée dans RESU (1).

H2) Si cet ensemble est vide alors BRESU : ="Faux" sinon BRESU : = "Vrai").

H3) Si ¬BRESU alors Module B bis

I - PREPARATION DANS LE TABLEAU TRAV DES LIGNES A ENVOYER DANS LE TABLEAU DE RESULTATS Wr.

- BOOL : = "Vrai"
- on envoie dans TRAV les valeurs des inconnues

(1) Pour le cas d'un CG hétéro-inconnue, voir la fin du paragraphe

délivrées par les lignes actuelles des tableaux Wj_v si ces lignes ont changé depuis le dernier succès (c'est-à-dire si NØUVIv = "Vrai").

J - ENVOI DES NOUVELLES LIGNES DE RESULTATS DANS Wr

L'un des trois modules suivants est généré par TRADID selon les cas :

J1) (Si le but est imposé ou lié à un argument)

Il n'y a qu'une ligne à envoyer en Wr et elle est prête dans TRAV ; Wr : = Wr U TRAV (1)

J2) (Si le but est une inconnue x_k non liée aux arguments mais évaluée (délivrée par Ww))

Pour chaque ligne de Ww dont l'indice de début est dans AD faire

.Pour chaque inconnue délivrée par Ww envoyer sa valeur dans le tableau TRAV

.Wr : = Wr U TRAV (1)

J3) (Si le but est une inconnue x_k non évaluée)

Pour chaque élément de RESU faire

.l'envoyer à sa place dans TRAV
.Wr : = Wr U TRAV

Bbis-FINS DE BOUCLAGE

$Bbis_v$- fin$_v$
$v=\ell,\ldots,2,1$

G - FIN DU PROGRAMME POUR LE MODULE DE RECHERCHE CONSIDERE

Cas d'un contexte à gauche hétéro-inconnue (à éviter)

Supposons que dans la source de Mi l'inconnue x_m ait dans certains cas la valeur "non déterminée" (cf § V-5-2). Lors de la traduction de EVAL(Mi) par TRADID, cependant, x_m est considérée comme ayant une valeur puisqu'elle n'est pas une inconnue propre de Mi. C'est donc à l'exécution que ce cas doit être prévu, aux endroits suivants :

- Module B_v2 : si l'on s'aperçoit qu'un argument à mettre en place est "non déterminé", deux solutions sont possibles. La meilleure consiterait à activer un module de fabrication de valeurs pour x_m, comme cela est expliqué à propos du module A2 ; mais ce module, contrairement au cas de A2, ne peut être prévu par TRADID que si TRADID tient un état des sous-filtres hétéro-inconnue et des inconnues incertaines dans ces sous-filtres, ce qui serait lourd. La deuxième solution, que nous adoptons au moins provisoirement, est de signaler une erreur dans ce cas et de passer à la ligne suivante du tableau Wj concerné.

- Module H1 : si pour une ligne de Ww la valeur de x_k est "non déterminée", on met aussi l'indice de début de la ligne dans AD (voir plus loin, module J2).

- Module J1 : si le but est lié à un argument mais, pour la ligne de Wj considérée, a la valeur "non déterminée", alors on effectue le module J3 au lieu de J1.

- Module J2 : il faut en fait remplacer l'instruction

  Wr : = Wr U TRAV par :

  <u>si</u> x_k a la valeur "non déterminée" pour cette ligne <u>alors</u> Module J3 <u>sinon</u> Wr : = Wr U TRAV.

Pour simplifier la rédaction, nous n'envisageons plus dans le paragraphe VIII-1-3-2 le cas d'un contexte à gauche hétéro-inconnue.

VIII-1-3-2- Précisions sur les modules du traitement précédent.

- Tableaux, compteurs et mémoires de travail entretenus et utilisés par le programme déterministe.

Nous avons déjà parlé du tableau W, de BOOL et PILBOOL (§ VIII-1-1-5), et (dans le paragraphe VIII-1-3-1) des tableaux ARG, RESU(et son compteur NRESU), AD, TRAV ainsi que des booléens BRESU et NØUVIv pour chaque v (nous commenterons l'utilité de ces derniers au cours du présent paragraphe, module B_v1). Voici les autres principaux points de stockage utilisés.

IMPLANW : c'est un tableau qui, pour chaque numéro j, donne l'indice dans W du premier élément de Wj. ChaqueWj se termine par une marque de fin que nous noterons $\neq$.

RECUPW : c'est un tableau booléen qui, pour chaque numéro j, contient "vrai" si le tableau Wj est devenu caduc et peut donc être récupéré, et "faux" sinon. Ce tableau, dont le contenu à chaque instant peut être connu dès la traduction par TRADID (qui l'utilise pour effectuer les récupérations "normales"), à l'exécution sera utilisé par le ramasse-miettes.

NW : c'est un compteur qui à chaque instant, contient le plus grand numéro j de Wj utilisé à cet instant (à part le tableau en cours d'élaboration). Son contenu peut être connu dès la traduction. Il sert pour le ramasse-miettes.

MWLIB : contient l'indice de la première mémoire libre dans W.

NAD : contient le nombre d'éléments de AD.

- Les modules :

Rappelons que tout ce que nous disons ici est vrai à un adressage indirect près (nous ne nous occupons pas de la gestion des valeurs elles-mêmes). Lorsqu'il nous est commode d'indiquer le contenu d'une partie de programme, nous le donnons en "pseudo-algol". Rappelons que pour rendre la rédaction plus claire nous négligeons dans cette partie la possibilité d'un contexte à gauche hétéro-inconnue (cf fin du paragraphe précédent, VIII-1-3-1).

- A2 - Le programme TRADID constitue ce module d'après les indications données dans la partie cadrage et les renseignements sur les propriétés de R donnés avec la structure. C'est lui qui signale une erreur s'il n'a aucun moyen de générer un programme.

A l'issue du module A2, chaque inconnue x_q qui est un argument et qui n'avait pas encore été évaluée a des valeurs dans un tableau $W_{q'}$. Mais le module A2, pour ce qui concerne la liaison des inconnues, peut être assimilé à un module de recherche : si x_q est évaluée par A2 en fonction d'autres inconnues, ces inconnues deviennent liées entre elles et liées à x_q. Leurs valeurs seront donc toutes réunies dans un même tableau de W à l'issue de A2. Il sera rare que x_q soit évaluée en fonction de plusieurs inconnues (surtout non liées

entre elles) ; mais il sera plus fréquent qu'elle soit évaluée en fonction d'une inconnue, le plus fréquent étant toutefois qu'elle reste, à ce stade, non liée aux autres inconnues (c'est-à-dire qu'elle soit évaluée indépendamment d'elles).

Le module A2 doit comporter une mise à jour du tableau IMPLANW et du compteur MWLIB.

Il se peut que le module A2 échoue pour une inconnue argument, c'est-à-dire qu'il ne fournisse aucune valeur. Dans ce cas, il envoie directement au module G après avoir donné à BOOL la valeur "faux".

Remarque 1 :

Il n'y a de module A2 que si les x_q sont des extrémités, ce qui n'arrivera pratiquement jamais. Le seul cas fréquent où il y a un module A2 est le cas de US.

Remarque 2 :

Si R est une relation binaire et si le but de l'arc est donné, on peut envisager qu'il faille autoriser la recherche à s'effectuer dans le sens inverse (R^{-1}).

Remarque 3 :

Lorsqu'une inconnue argument x_q est, par A2, liée à d'autres arguments, on pourrait, au lieu de chercher toutes ses valeurs dans la phase d'initialisation, en chercher pour chacun des jeux d'arguments, au fur et à mesure que ces jeux sont traités dans le coeur du programme. Nous n'adopterons pas cette solution.

- A3-

IMPLANW [numéro du tableau de résultats] := MWLIB

RECUPW [numéro du tableau de résultats] := "Faux"

BOOL := "faux".

- B_v- Le bouclage peut être exprimé en pseudo-algol par :

<u>Pour</u> Iv := IMPLANW $[j_v]$, Iv + longueur des lignes de Wj_v

<u>tant que</u> W [Iv] ≠ "#" <u>faire</u>

j_v et longueur des lignes de Wj_v sont connus et donc générés par TRADID.

Remarque : ordre des j_v

Quel que soit l'ordre des j_v, éléments de J, la procédure de recherche sera appelée le même nombre de fois. Ce qui varie d'un ordre à un autre, c'est le temps passé en mise en place des arguments et résultats. Chaque ligne du tableau Wj_1 sera manipulée une seule fois ; chaque ligne de Wj_2 sera manipulée une fois pour chaque ligne de Wj_1, donc n_1 fois (si n_v est le nombre de lignes de Wj_v) ; ... chaque ligne de Wj_v sera manipulée $n_1 \times n_2 \times .. \times n_{j-1}$ fois ; .... On a donc intérêt à placer

en tête les tableaux dont les lignes à manipuler sont les plus longues: soit en ce qui concerne le nombre d'arguments (et but) délivrés, c'est-à-dire à placer dans ARG, soit en ce qui concerne le nombre total d'inconnues délivrées, c'est-à-dire à placer dans TRAV. L'importance de ce dernier facteur dépend beaucoup du pourcentage de succès de la recherche, celle du premier est constante. Nous avons choisi de classer les j_v dans l'ordre décroissant du nombre d'arguments (et but) délivrés par Wj_v.

- B_v1-Utilité de NØUVIv : lorsqu'on est au coeur du bouclage le plus profond (le ℓème), on ne sait pas par quels $\underline{fin}_v$ on est passé au tour précédent ; autrement dit, on ne sait pas quels sont les tableaux Wj_v pour lesquels on vient de changer de ligne. Ainsi on ne sait pas quels sont les éléments de TRAV qui sont encore valables pour figurer dans les lignes de résultats et ceux qui ne le sont pas. On entretient donc, pour chaque tableau Wj_v (sauf le dernier car il garderait la valeur "Vrai"), un indicateur booléen NØUVIv qui a la valeur "Vrai" si et seulement si l'on a changé de ligne dans Wj_v depuis l'élaboration du résultat précédent, c'est-à-dire si les valeurs dans TRAV des inconnues délivrées par ce tableau ne sont plus valables.

Le module B_v1 contient donc, si $v \neq \ell$ (ce qui est su par TRADID) :

NØUVIv : = "Vrai".

- B_v2- Pour chaque argument (soit q son numéro) et éventuellement le but (q=0) qui est une inconnue x_r délivrée par Wj_v, le programme TRADID fabrique l'instruction :

ARG [q] := W [Iv + déplacement de x_r par rapport au début de ligne]

(TRADID connaît, donc génère, q et le déplacement).

L'ensemble de ces instructions constitue le module B_v2.

-C- En fait, la procédure de recherche n'a besoin d'aucun paramètre car son but éventuel et ses arguments sont stockés en des emplacements fixes, le nombre des arguments étant connu pour chaque procédure.

-H- Nous avons placé, dans l'organigramme, le début de ce module entre parenthèses car il n'est généré par TRADID que si le but est une inconnue x_k déjà évaluée mais non liée à un argument. En effet, si le but est fixe ou lié à un argument (il y a une valeur en ARG [0]) alors la procédure elle-même tient compte de lui, et si le but n'a pas encore été évalué alors il n'y a pas de vérifications à effectuer.

-H1- L'inconnue x_k a ses valeurs dans Ww, éventuellement associées à des valeurs d'autres inconnues (qui ne sont pas des arguments). Il faut chercher quelles sont les lignes de Ww dans lesquelles la valeur de x_k est un résultat de la procédure ; on place les indices de début de ces lignes dans le tableau AD. Le module est :

```
NAD := 0 ;
Si  NRESU = 0 alors Module H2 ;
Pour t : = IMPLANW [w], t + longueur des lignes de Ww
     tant que  W [t] ≠  "#" faire
```



```
Début z : = 1 ;
     tant que z ≤ NRESU et  W [ t + déplacement  de x_k ]
                ≠ RESU [z] faire  z : = z + 1 ;
     NAD : = NAD + 1 ;
     AD [ NAD ]: = t

fin ;
```

TRADID connaît, donc génère, les valeurs de w, longueur de ligne de Ww, déplacement de x_k (par rapport au début de ligne dans Ww). Il est possible qu'au lieu de générer tout ce programme il génère un appel à une procédure jouant le même rôle, avec justement ces valeurs comme paramètres. Au cours de l'exécution, il est possible que pour un même élément de RESU plusieurs lignes conviennent dans Ww, correspondant à la même valeur de x_k mais à des valeurs différentes des inconnues qui lui sont liées.

- H2 - BRESU : = si NAD = 0 alors "Faux" sinon "Vrai".

- I- Donnons d'abord quelques détails sur le tableau TRAV. Si, à l'issue de l'appel du module de recherche considéré, les inconnues (globales ou locales) $x_i, x_j, \ldots, x_k, \ldots$ sont liées (avec $i < j < \ldots < k < \ldots$) alors on désire que chaque ligne du tableau de résultats soit composée de la juxtaposition de valeurs (à un adressage indirect près, comme toujours dans ce chapitre) respectivement des inconnues $x_i, x_j, \ldots, x_k, \ldots$ La correspondance entre

numéro d'inconnue et rang dans la ligne est connue par TRADID.

Chaque ligne à envoyer dans W est formée dans TRAV. Pourquoi pas directement dans W ? Parce que deux lignes différentes ont, presque toujours, beaucoup d'éléments communs ; les éléments constants d'une ligne à la suivante restent préparés dans TRAV au lieu d'avoir à être cherchés à nouveau dans leur tableau Wj_v.

Le module I, après l'affectation de "vrai" à BOOL, comporte en fait ℓ modules, un par élément j_v de J. Voyons ce que comporte le module I_v qui met, si besoin est, en place dans les cases convenables de TRAV les valeurs des inconnues délivrées par la ligne actuelle de Wj_v.

-I_v- On commence par tester la valeur de NØUVIv pour savoir s'il est utile d'effectuer la partie principale de I_v. Si NØUVIv est "vrai", il faut effectuer cette partie, décrite ci-dessous. Ce test n'est pas généré pour le tableau Wj_ℓ car NØUVIℓ est évidemment toujours "vrai".

<u>Partie principale de I_v</u> : le programme TRADID génère, pour chacune des inconnues x_k délivrée par Wj_v, l'instruction :

TRAV [rang de x_k dans TRAV] : = W [Iv +

déplacement de x_k dans sa ligne de Wj_v]

Rang de x_k et déplacement de x_k sont connus, donc générés, par TRADID. Iv est, rappelons-le, l'indice dans W du début de la ligne actuelle de Wj_v.

<u>Remarque</u> : pour éviter l'entretien et le test des NØUVIv , on aurait pu placer la préparation des lignes de TRAV dans les modules B_v. Mais cette préparation aurait été effectuée pour rien en cas d'échec.

-J1- <u>Pour</u> u : = 1 <u>pas</u> 1 <u>jusqu'à</u> longueur des lignes de TRAV <u>faire</u> W [MWLIB + u] : = TRAV [u] ;

MWLIB : = MWLIB + longueur des lignes de TRAV

Longueur des lignes de TRAV (c'est-à-dire leur nombre d'éléments) est connu, donc généré, par TRADID.

Remarque : On peut envisager que, pour chaque module de recherche qui est le dernier d'un motif, TRADID remplace la longueur des lignes de TRAV par la longueur de la partie occupée par des inconnues globales (elles sont groupées en début de ligne).

-J2- Appelons x_{w1}, x_{w2},... les inconnues délivrées par le tableau Ww. TRADID connaît leurs indices, leur déplacement par rapport à chaque début de ligne de Ww et le rang que doivent occuper leurs valeurs dans TRAV.
TRADID génère donc le module :

Pour v : = 1 pas 1 jusqu'à NAD faire

Début TRAV [rang de x_{w1}] : =W [AD [v] + déplacement de x_{w1}] ;
TRAV [rang de x_{w2}] : =W [AD [v] + déplacement de x_{w2}] ;
...
Module J1

Fin

-J3- Pour v:=1 pas 1 jusqua NRESU faire

Début TRAV [rang de x_k] := RESU [v] ;
Module J1

Fin

-G- Nous avons décidé (cf § VIII-1-2-7) de n'effectuer aucune récupération à ce stade. Les seules actions à effectuer ici sont donc d'une part (G1) de marquer dans RECUPW les indices de tableaux qui ne sont plus utiles (les tableaux résultats intermédiaires qui ont été utilisés dans ce module de recherche), de mettre à jour NW et de placer un "#" à la fin du tableau de résultats, et d'autre part (G2) de tester le résultat booléen du module.

-G1- Pour chacun des W_{j_v} qui sont des tableaux de résultats intermédiaires et ont été utilisés, pour fournir arguments ou but, par le présent module, TRADID génère l'ordre :

RECUPW [j_v] : = "Vrai" .

De plus, il génère les ordres :

NW : = numéro du tableau de résultats (connu à la traduction par TRADID) ;
RECUPW [NW] : = "Faux" ;
W [MWLIB] : = "#" ;
MWLIB : = MWLIB + 1.

-G2- Si BOOL = "Faux", c'est que l'on n'est jamais passé par le module I, donc que la recherche n'a jamais abouti. Comme nous l'avons dit au paragraphe VIII-1-1-5, le programme, dans ce cas, doit envoyer à la fin de la séquence correspondant à EVAL(Mi); or cette fin est matérialisée (dans le programme indéterministe mais aussi dans le programme exécutable) par l'étiquette Fi. Le programme TRADID connaît le numéro du motif en cours. Il génère tout simplement ici l'ordre :

Si ¬ BOOL alors allera Fi

avec la valeur convenable pour i.

Remarque : nous n'avons fait appel aux indications données dans la partie cadrage ou dans les spécifications de la structure que lorsqu'une inconnue pour laquelle on a besoin de valeurs n'en a pas encore. Mais il faudra aussi envisager le cas où ces indications interviennent pour valider (ou éliminer) des valeurs d'inconnues arguments ou but : ceci pourrait s'effectuer au cours des phases A ou B dans le premier cas, H dans le second. En fait, c'est souvent la procédure de recherche qui se chargera des vérifications liées aux spécifications de la structure.

VIII-1-3-3- Modifications à apporter aux modules pour pouvoir prendre en compte le rôle des opérateurs affectant R.

1°) Non : voyons, selon la nature du but, ce qui doit se produire lors de l'appel du module non R $(a_1, a_2, \ldots; b)$; appelons A l'ensemble des arguments.

α- b donné et unique pour chaque jeu de A (soit fixe, soit lié à des arguments de A) : à l'issue de la procédure, il faut simplement inverser le contenu de BRESU pour ne conserver le jeu (A,b) dans le tableau de résultats que si ¬ ARb ; ceci se fera par un module H'0 qui sera généré avant le module H3 (rappelons que dans le cas α les modules H1 et H2 n'existent pas).

- H'0 - BRESU : = ¬ BRESU

β- b inconnue x_k évaluée mais non liée aux arguments :

dans la phase H1, on devra noter dans AD non pas les lignes de Ww pour lesquelles ARb mais celles pour lesquelles ¬ ARb, c'est-à-dire pour lesquelles la valeur de $b(x_k)$ ne se retrouve pas dans RESU. En particulier, si RESU = Ø toutes les valeurs de b donc toutes les lignes de Ww conviennent. Le module H2 ne sera pas modifié. Seul, donc, H1 sera remplacé par le module H'1 :

```
- H'1 -  NAD : = 0 ;
Si NRESU = 0 alors pour t : = IMPLANW [w],
   t + longueur des lignes de Ww tant que
   W[t] ≠ "#" faire
   début NAD := NAD + 1 ; AD[NAD] : = t fin
sinon pour t : = IMPLANW [w] , t + longueur
   des lignes de Ww tant que W[t] ≠ "#" faire
   début  z : = 1 ;
   tant que  z  ≤NRESU et  W[t + déplacement de x_k]
            ≠ RESU[z] faire  z : = z + 1 ;
   NAD : =  NAD + 1 ;
   AD [NAD] : = t
   fin ;
```

-γ- b est inconnue non évaluée :

Si B est l'ensemble de toutes les valeurs possibles pour b et si, pour un jeu d'arguments A, l'ensemble des valeurs telles que ARb est noté R(A), alors ce que nous devons rechercher pour chaque jeu de A est $\subset_B$ R(A). C'est ce qui a été fait dans les deux cas α et β. Mais ici l'ensemble B est a priori inconnu.

- Si l'ensemble B peut être déduit d'indications données dans la partie cadrage ou de propriétés de la structure, le programme TRADID générera dans le module A2 une phase de programme créant cet ensemble. On sera ensuite ramené à l'un des cas α ou β .

-Si TRADID ne peut trouver de moyen de créer un module de fabrication de B, on peut songer, soit à signaler une erreur, soit à garder l'appel de module R(A;b) en réserve jusqu'à ce qu'il soit possible de trouver un ensemble B. Mais ceci est assez compliqué. En fait, ce cas se présentera très rarement, d'une part parce que les motifs seront rarement tels qu'on aboutisse à de telles difficultés, d'autre part parce que, lors de la recherche de stratégie, les arcs étiquetés par une relation affectée de l'opérateur de négation, surtout si le but est inconnu, auront la priorité minimale.

2°) Réciproque : la relation R ne peut être affectée de l'opérateur $^{-1}$ que si elle est binaire. Le programme TRADID, au lieu de traiter R^{-1} (a ;b), traite R(b;a) c'est-à-dire se ramène au cas "normal" en inversant les rôles de a et b. Il n'y a pas de difficulté particulière.

3°) Itération : la relation R ne peut, là encore, être affectée de l'opérateur * que si elle est binaire. L'itération doit s'arrêter soit lorsqu'on arrive à un succès qu'on sait unique, soit lorsque la procédure de recherche échoue. Cependant, pour être certain de ne pas avoir un bouclage "infini", on a décidé de fixer un nombre maximum d'itérations (MAXITER) pour chaque valeur de l'argument a.

α- b donné et unique pour chaque a :

b ne sera pas placé en ARG[0]. A chaque étape de l'itération, les arguments de R seront pris dans un tableau RARG (à NARG éléments) pour être tour à tour envoyés dans ARG, et les résultats de R, qui serviront

en général d'arguments pour le tour suivant, seront envoyés en RBUT (à NBUT éléments). Le module C est remplacé par le module C_α'' suivant :

```
-C''α- Initialisation α :  ITER : = 1 ;
                           RARG[1] : =  ARG [ 1 ] ;
                           NARG : = 1 ;
                           ARG [ 0 ]: = rien ;

Module Itération α :

      NBUT : = 0 ;
      Pour u : = 1 pas 1 jusqua NARG faire
         début ARG[1] : =  RARG [u];
               Appel de la procédure de recherche ;
               si b ∈ RESU alors allera Module I ;
               RBUT : = RBUT  U  RESU ;
               NBUT : = NBUT + NRESU (1)
         fin ;
     - Module fin itération α :
      ITER : = ITER + 1 ;
      Si  NBUT = 0 ou  ITER = MAXITER alors
               Module Bbis sinon
         début tableau RARG : = tableau RBUT ;
               NARG : = NBUT ;
               Module Itération α
         fin ;
```

(1) Si l'on élimine les redondances dans RBUT (ce qui est souhaitable), il faut diminuer NBUT d'autant.

β- Le but est une inconnue x_k non liée à a

On doit chercher toutes les valeurs de x_k telles que a R * x_k. Le module de programme ressemble au précédent mais, à chaque itération, on doit chercher quelles sont les lignes de Ww pour lesquelles $x_k \in$ RBUT. Seulement, si une ligne a déjà été trouvée, il est inutile de la considérer à nouveau aux tours suivants.

Les modules C, H1 et H2 sont remplacés par le module C_β'' suivant :

```
-C''β- Initialisation β : ITER : = 1 ;
                         RARG[1]: = ARG[1] ;
                         NARG : = 1 ;
                         NAD : = 0 ;

Module Itération β :

      NBUT : = 0 ;
      Pour u : = 1 pas 1 jusqua  NARG faire
         début    ARG  [1] : = RARG [u] ;
                  Appel de la procédure de recherche ;
                  pour t : = IMPLANW[ w], t +
                    longueur des lignes de Ww
                    tant que  W [t] ≠ "#" faire
                    si t ∉ AD et  W [t + déplacement
                    de x_k ] ∈ RESU faire
                    debut NAD : = NAD + 1 ;
                          AD [ NAD ]: = t
         fin ;
```

Si AD contient les indices de toutes les lignes de Ww alors allera Module I ;

RBUT := RBUT U RESU ;
NBUT : = NBUT + NRESU (1)

fin ;

Module fin itération β :

identique à fin itération α en remplaçant α par β .

γ- b est inconnue non évaluée

On peut reprendre ici les alinéa 2 et 3 de 2° γ (réciproque).

4°) Nous dévrions aussi examiner les cas où deux ou trois opérateurs affectent R. Mais il nous semble inutile d'alourdir la rédaction par ce développement qui, s'il est utile en pratique, n'aurait pas un grand intérêt dans ce texte.

(1) Si l'on élimine les redondances dans RBUT (ce qui est souhaitable), il faut diminuer NBUT d'autant.

VIII-1-4- TRAITEMENT D'UNE REUNION.

Lorsqu'une réunion doit être exécutée, c'est entre les résultats du dernier sous-filtre traité (appelons-le f_1) et d'un certain autre sous-filtre (f_2). Le programme TRADID connaît les noms des inconnues de ces deux sous-filtres (et de leur CTG) ainsi que les numéros j des Wj qui contiennent leurs valeurs. Appelons f_r le sous-filtre f_1 ou f_2 et appelons x_1, x_2, x_r les ensembles d'inconnues (y compris celles de leur CTG) respectivement de f_1, f_2, f_r. Si f_r n'est pas "hétéro-inconnue" (cf § V-5-2), alors X1 = X2 = Xr ; si f_r est "hétéro-inconnue", c'est que X1 ≠ X2, et on a alors Xr = X1 U X2. De toute façon, après l'évaluation de f_r, toutes les inconnues composant Xr sont liées (cf § VIII-1-2-2). Ainsi, le résultat de f_r est nécessairement constitué d'un tableau unique. Appelons-le Wr.

La constitution et la mise en place de Wr peut se décomposer en les étapes suivantes :

- M - Composition d'un tableau Wk unique de résultats de f_1.
- N - Composition d'un tableau Wℓ unique de résultats de f_2.
- P - Réunion des deux tableaux : on obtient le tableau de résultats mais en ayant gâché de la place.
- Q - Mise en place du tableau de résultats.

Donnons quelques détails sur ces modules.

- M - Si BOOL est "faux" (recherche de f_1 infructueuse), il n'y a pas de tableau Wk à préparer. Ou plutôt on forme Wk[1] = "#" (Wk vide). De plus si f_r est en fait un motif Mi, il faut activer le module V que nous décrivons ci-dessous.

Module V : il libère les tableaux qui auraient été libérés au cours du traitement des modules de recherche de Mi qui, à cause de l'échec de l'un d'entre eux, n'ont pas été effectués. Comme la gestion des tableaux est effectuée à la traduction, il n'est pas possible de savoir exactement quels sont les tableaux à récupérer car ils dépendent de l'endroit où s'est produit l'échec. Le programme TRADID génère donc un module, le "Module V", qui est activé si BOOL est "faux", et qui affecte la valeur "vrai" à toutes les cases de RECUPW correspondant à des résultats intermédiaires du motif Mi. Notons cependant que le module V, s'il permet de libérer plus de place au cours d'un ramasse-miettes éventuel, n'est pas indispensable.

Si BOOL est "vrai", chaque ligne du tableau Wk devra contenir une valeur pour chacune des inconnues constituant Xr (et non pas seulement X1). S'il existe des inconnues figurant dans Xr mais pas dans X1, elles auront dans Wk la valeur "non déterminée" (représenté dans notre texte par un tiret).

Soit Wk_1, Wk_2, ... les tableaux constituant le résultat de f_1. Si seul Wk_1 existe, que toutes les inconnues de Xr y figurent et qu'il ne contient pas d'inconnues locales, alors le module M n'est pas généré et k est en fait k_1.

Sinon, le module M constitue le tableau Wk (k est calculé par TRADID ; c'est le premier numéro libre) de la façon suivante (en "pseudo-algol"):

```
Pour chaque ligne de Wk1 faire
  Pour chaque ligne de Wk2 faire
    .....
        former une ligne de Wk par fusion ;
        incrémenter MWLIB ;

RECUPW [ k1 ]: = "vrai" ;
RECUPW [ k2 ]: = "vrai" ;

......

W [ MWLIB ] : = "/" ;
MWLIB : =  MWLIB + 1 ;
RECUPW [ k ]: = "faux" ;
NW : = k .
```

Le module de fusion cherche les valeurs des inconnues à leurs places dans les lignes considérées puis éventuellement crée des tirets pour les valeurs "non déterminées". Les valeurs d'inconnues locales ne sont pas transmises donc sont éliminées. Une solution simple pour réaliser ce module est d'utiliser le tableau TRAV comme pour le traitement des modules de recherche. Ainsi, la partie principale du module M aura plutôt la composition suivante :

Mettre des "tirets" dans les positions de TRAV correspondant aux inconnues absentes de x_1.

Pour chaque ligne de Wk_1 faire

```
début1 l'éclater dans TRAV (sans les inconnues locales)
   Pour chaque ligne de Wk2 faire
   début2 l'éclater dans TRAV
       ...
          début_n l'éclater dans TRAV
                Wk : = Wk U TRAV;
                MWLIB : = MWLIB + longueur des
                          lignes
          fin_n
       ...
   fin2
fin1
```

-N- Analogue à M mais au début c'est le sommet de PILBOOL que l'on teste (cf § VIII-1-1-5).

-P- La réunion des deux tableaux Wk et Wℓ est une simple réunion d'ensembles : on met en commun les lignes de Wk et celles de Wℓ, en supprimant les doubles. Tout ceci est simple à concevoir, mais peut être long à exécuter.

Un détail doit cependant retenir notre attention : celui des tirets. Ils ont, pourrait-on dire, un rôle absorbant. Illustrons cela par un exemple simple. Nous en avions d'ailleurs déjà donné un au cours du paragraphe V-5-2. Soit Wk et Wℓ les deux tableaux (ensembles) de triplets suivants:

Wk={ (5,15,-), (8, 12,-)}
Wℓ={ (1,12,25), (5,15,28),(-,12,29)}

alors Wr=Wk U Wℓ= { (5,15,-),(8,12,-),
(1,12,25), (-,12,29)}

Le triplet (5,15,28) est inclus dans (5,15,-) donc ne figure pas dans la réunion.

Mais attention ! Les triplets (8,12,-) et (-,12,29) ne sont pas comparables pour l'inclusion. Tous deux doivent donc subsister dans la réunion.

Le numéro r de Wr est le premier numéro disponible à la suite des autres. A la fin de la constitution de Wr, on fait suivre celui-ci du signe "#".

Si Wr est vide (c'est que BOOL ou PILBOOL(sommet) est "faux"), il faut faire BOOL : ="Faux", sinon BOOL : = "Vrai". De plus, de toute façon, il faut enlever le sommet de PILBOOL (cf VIII-1-1-5).

-Q- Le programme TRADID entretient un état des tableaux qui seront récupérables à l'exécution. D'ailleurs cet état est généralement tenu à l'exécution car il sert pour le ramasse-miettes qui, intervenant à des moments aléatoires, ne peut pas être informé dès la traduction.

En particulier, si une source devient caduque à l'issue d'une réunion, TRADID doit effectuer la mise à jour de cet état avant de générer la mise en place de Wr.

TRADID doit chercher dans cet état, en partant de r et de façon décroissante, quel est le premier numéro j de W non récupérable. C'est j + 1 qui est le numéro final du tableau de résultats de la réunion. Celui-ci sera implanté à partir du même indice que l'ancien Wj + 1.

Le module Q est donc un simple transfert du tableau Wr (y compris le signe terminal #) en Wj + 1. Le compteur MWLIB est mis à jour.

Si BOOL = "faux", la seule chose à faire dans ce module Q est de mettre à jour MWLIB à la valeur IMPLANW [j+1] .

De toute façon, NW est mis à jour à la valeur j + 1 et RECUPW [NW] est mis à la valeur "Faux". RECUPW [k] et RECUPW [ℓ] sont mis à la valeur "vrai".

VIII-1-5- ENVIRONNEMENT DES TRAITEMENTS PRECEDENTS.

Au cours des paragraphes précédents, nous avons étudié le déroulement d'un module de recherche, l'exécution d'une réunion et la gestion des résultats booléens ou non au cours de ces traitements. Pour compléter la description du programme exécutable, il reste à décrire brièvement l'initialisation en début de programme ainsi que la terminaison de celui-ci, les opérations à effectuer juste après le dernier module d'un motif, lorsqu'il n'y a pas de réunion à effectuer (car le cas de la réunion a été étudié au paragraphe VIII-1-4), et enfin le ramasse-miettes.

VIII-1-5-1- Initialisation en début de programme et terminaison de celui-ci.

Parmi les variables intervenant dans le programme exécutable et dont la liste est donnée au début du paragraphe VIII-1-3-2, certaines doivent être initialisées une seule fois au début du programme. Ce sont :

- la pile PILBOOL ou plutôt son compteur (à 0) ;
- NW qui est mis à 0 ;
- MWLIB qui est mis à 1 (1er indice libre dans W).

Après le dernier module du programme, il faut en général suivre les indications données dans la partie cadrage. Cependant, lorsque le filtre contient US, il faut boucler sur l'instruction donnant une valeur à US (car cette inconnue est gérée sériellement ; cf § V-5-3). Ce n'est qu'à la sortie de ce bouclage que l'on a vraiment terminé le programme et que l'on suit la partie cadrage.

VIII-1-5-2- Fin d'un motif

Au paragraphe VIII-1-2-7, nous avons indiqué (dans "place du tableau de résultats d'un module de recherche") qu'à la fin d'un motif on recopie le dernier tableau de résultats (qui très souvent sera le seul) pour gagner de la place. Nous avons déjà vu le cas où une réunion doit être effectuée ensuite ; étudions le cas contraire.

Si BOOL est "faux", il faut activer un module V (décrit à propos du module M du paragraphe VIII-1-4).

Ce que nous avons dit au sujet du module Q dans le paragraphe VIII-1-4 en ce qui concerne le numéro j + 1 du tableau récepteur est entièrement valable ici.

Il s'agit donc d'effectuer un transfert ligne par ligne du tableau W_{NW} dans le tableau W_{j+1} en éliminant les variables locales (TRADID préparera ce travail) et en supprimant les lignes redondantes ; on effectue également le transfert du signe terminal #.

A la fin, on effectue la mise à jour de MWLIB, NW est mis à la valeur j + 1 et RECUPW [j+1] est mis à "faux".

VIII-1-5-3- Ramasse-miettes.

Chaque fois que l'on prend de la place dans W, il faut qu'un débordement éventuel puisse être signalé.

Si ce cas se produit, on doit faire fonctionner un ramasse-miettes. Celui-ci consiste simplement à effectuer un tassement vers le début du tableau W pour supprimer les trous laissés par des Wj qui n'ont plus cours (leur RECUPW [j] est à "vrai").

Remarquons que ce ramasse-miettes n'aura en général pas beaucoup de place à récupérer car des récupérations sont effectuées au cours du traitement "normal". Ce point a été discuté au cours des paragraphes VIII-1-2-5 et VIII-1-2-7. Il pourra cependant être amélioré comme nous le dirons dans la remarque située en fin du présent paragraphe.

Au moment où est activé le ramasse-miettes, NW contient le numéro du dernier Wj strocké ; le tableau en cours d'élaboration n'est pas encore pris en compte dans NW. A ce même moment, le tableau RECUPW contient, pour chaque Wj (de 1 à (NW)) la valeur "vrai" si la place du tableau Wj peut être récupérée. Enfin le tableau IMPLANW contient pour chaque Wj l'indice dans W de son premier élément.

Le ramasse-miettes ne modifie pas les numéros de tableaux (et par conséquent ni NW ni RECUPW) mais modifie leur implantation (et par conséquent IMPLANW).

Les variables intervenant dans le ramasse-miettes, en plus de celles déjà introduites, sont :

RECEPT qui contiendra l'indice de la case de W prête à recevoir le prochain élément translaté.

DER qui contiendra le numéro qui suit celui du dernier tableau Wk translaté.

R qui contiendra le numéro qui suit celui du premier tableau récupérable.

Le ramasse-miettes peut se décomposer ainsi :

- Module R - Recherche du premier tableau récupérable.

```
k : = 1 ;
tant que  k ≤ NW-1 (1) et ( ¬ RECUPW[k ] ou IMPLANW [k] =
          IMPLANW [k+1]) (2) faire  k : = k +1 ;

          RECEPT : = IMPLANW [k] ;
          DER : = R : = k + 1
```

Si l'on n'a pas trouvé de Wk récupérable, échec du ramasse-miettes. Voir remarque, plus loin.

---

(1) Le dernier tableau stocké n'est sûrement pas récupérable.

(2) C'est-à-dire Wk non récupérable ou déjà récupéré au cours d'un ramasse-miettes précédent (cf Module S).

- Module S - Translation des tableaux Wk non récupérables

Pour k: = R pas 1 jusqua NW faire

si]RECUPW [k] alors

Début Pour j : = DER pas 1 jusqua k faire

IMPLANW [j]: = RECEPT
(tous les tableaux récupérés et de numéros inférieurs à k sont considérés comme ayant, comme Wk, leur premier élément dans la case de W dont l'indice est dans RECEPT) ;

Translater Wk pour que son premier élément vienne en W [RECEPT] ;

Mettre à jour RECEPT :

RECEPT : = indice du dernier élément de Wk (c'est-à-dire #) après translation + 1 ;

DERTRAN : = k + 1

fin ;

- Module T - Translation du tableau Wr en cours d'élaboration :

son numéro r est en NW.

Pour j : = DER pas 1 jusqua k faire

IMPLANW [j] = RECEPT ;

Translater la partie déjà élaborée de Wr pour que son premier élément vienne en W [RECEPT] ;

MWLIB : = indice du dernier élément translaté de Wr (celui dont l'indice était en NWLIB-1) ;

Remarque : le ramasse-miettes que nous venons de décrire ici pourrait être amélioré : il ne comporte pas la récupération des places des inconnues locales. Si l'on voulait effectuer cette récupération, il faudrait, à chaque instant, connaître à l'exécution la composition des lignes de tous les tableaux non caducs, ou au moins l'emplacement, dans ces lignes, des valeurs des inconnues locales. De plus, il faudrait savoir quels sont les tableaux non caducs relatifs au motif en cours de traitement car, pour eux, on n'a pas le droit de supprimer les inconnues locales. Tout ceci serait lourd mais réalisable. Nous verrons à l'usage si l'intérêt en est déterminant.

VIII-1-6- REDONDANCE DES JEUX D'ARGUMENTS.

Il est ennuyeux d'appeler une même procédure de recherche plusieurs fois pour le même jeu d'arguments. Mais c'est difficilement évitable. Illustrons ce problème par un exemple simple : soit à effectuer la séquence relative au module de recherche $R(x_1, x_3, x_4 ; b)$. Supposons que x_1 et x_3 soient liées entre elles et aux inconnues x_2 et x_5 (x_5 étant en fait l'inconnue locale y_1) et que leurs valeurs soient à prendre dans un tableau Wj résultat d'un motif précédent (y_1 était donc une variable locale à ce motif et n'a plus aucune utilité au cours de la recherche présente). Supposons que ce tableau Wj contienne :

x_1	x_2	x_3	x_5
1	4	5	2
1	6	5	4
1	6	5	9
3	4	8	2

Parmi les quatre jeux d'arguments x_1 et x_3 qui seront considérés (combinés à des jeux pour x_4), deux sont utiles. Or il est très lourd de rester des non-redondances, surtout lorsque le nombre d'inconnues délivrées par un tableau est grand (en plus, pour un module, il y a autant de tels tests à effectuer que de tableaux mis en jeu, et les nombres d'inconnues concernées sont divers).

Le fait d'introduire la notion d'inconnues liées et de stocker séparément les inconnues non liées diminue beaucoup la redondance mais ne l'élimine pas. Par contre, le fait de laisser des valeurs d'inconnues locales dans des résultats de motifs (rappelons que, dans les tableaux de résultats qui ne sont pas le dernier trouvé pour un motif, des inconnues locales peuvent subsister ; cf § VIII-1-2-7) peut créer des redondances ; c'est le cas ici pour x_5 : en l'absence de x_5, on aurait supprimé la troisième ligne du tableau Wj.

Pour l'instant, nous laisserons subsister les redondances résiduelles ; la première décision que nous prendrons pour les réduire sera de tester et supprimer les redondances dans le cas simple mais nettement le plus fréquent où la relation a un argument unique, en particulier lorsque le but ne sera pas une inconnue liée à cet argument ; la deuxième décision sera de supprimer les inconnues locales dans tous les tableaux de résultats des motifs.

VIII-2- LE PROGRAMME TRADID.

L'effet de ce programme a été longuement décrit au paragraphe VIII-1. Rappelons que sa donnée est un programme maquette et son résultat un programme exécutable (1) (à part les procédures de recherche elles-mêmes). Nous allons donner les grandes lignes de TRADID, en ne cherchant pas à économiser les itérations ou les tests, de façon à rendre ses phases les plus indépendantes possible. Nous n'explicitons l'évolution des compteurs que lorsqu'elle n'est pas évidente.

VIII-2-1- LES TABLEAUX ET COMPTEURS ENTRETENUS.

VIII-2-1-1- A l'échelle du filtre.

PILINC : nous l'avons présenté au paragraphe VIII-1-2-6 sous forme d'une pile de tableaux à deux dimensions, représentée par un tableau à trois dimensions. Pour alléger sa gestion, nous l'implémenterons sans doute sous la forme de trois tableaux à deux dimensions (trois parce que PILINC [.,.,2] n'est plus utilisé). Mais négligeons ce détail ici.

L'étage n° j de la pile, PILINC [j,.,.] , correspond au sous-filtre dont la désignation est j. Nous appelons ligne n° k de l'étage j le sous-tableau PILINC [j,k,.] ; la $k^{ème}$ ligne correspond à l'inconnue x_k. L'étage j a quatre "colonnes" mais en fait trois utiles parce que la deuxième est désormais inutilisée. PILINC [j, k, 1] contient le numéro j du tableau Wj qui délivre x_k (pour le sous-filtre considéré,

(1) Peu importe pour l'instant le langage dans lequel sera généré ce programme (vraisemblablement en langage proche du niveau machine) ; nous l'avons décrit en "pseudo-algol".

à l'instant actuel) ; PILINC [j, k, 3] et PILINC [j, k, 4] contiennent le rang de x_k dans chaque ligne de Wj et la longueur des lignes de Wj.

NI : numéro du dernier motif traité ou en cours. NI donne donc aussi le numéro du dernier étage de PILINC utilisé.

NBINC : nombre d'inconnues déjà évaluées à l'instant considéré (évaluées au sens statique, c'est-à-dire sans tenir compte du problème posé à l'exécution par les filtres hétéro-inconnue (§ VIII-1-3-1)). Ce compteur donne le nombre de lignes actuel de l'étage de PILINC en cours de traitement.

CRECUPW : ce tableau a le même but que RECUPW à l'exécution. Au point de vue de la récupération, un tableau Wj peut être dans trois états différents à un instant donné :

. récupérable ; alors CRECUPW [j] = 0 ;

. résultat d'un sous-filtre précédent qui peut encore être contexte à gauche ; alors CRECUPW [j] = 2 jusqu'à ce que le sous-filtre considéré soit déchu de son rôle de contexte à gauche ou soit l'objet d'une réunion : à ce moment CRECUPW [j]est mis à 0 ;

. resultat intermédiaire dans un EVAL (Mi) ; dans ce cas, CRECUPW [j] = 1 ; au cours de EVAL (Mi), si le tableau Wj est utilisé (pour fournir arguments ou but), alors il devient récupérable et CRECUPW[j] est mis à zéro ; à la fin de EVAL (Mi), si CRECUPW[j] = 1 c'est que Wj n'a pas été

réutilisé (1), donc qu'il fait partie du résultat final : on range 2 en CRECUPW [j] .

CNW : ce compteur a la même signification que NW à l'exécution ; il contient à chaque instant le plus grand numéro j de tableau Wj utilisé.

VIII-2-1-2- A l'échelle de l'appel de module de recherche.

Les tableaux et compteurs dont nous parlons ici sont locaux à chaque traitement d'appel de module de recherche. Certains sont également utilisés par les modules de réunion et de fin de motif.

TABW : ce tableau possède une ligne par tableau Wj_v impliqué dans l'appel considéré (pour fournir arguments ou but). La ligne v a la composition suivante :

. en TABW [v, 1] : le numéro j_v du tableau

. en TABW [v, 2] : le nombre d'arguments délivrés par Wj_v

. en TABW [v, 3] : la longueur de chaque ligne de Wj_v ($\geqslant$ TABW [v, 2])

. en TABW [v, 4 + q] : si l'argument a_q de l'appel n'est pas délivré par Wj_v alors 0, sinon le rang, dans chaque ligne de Wj_v, de la valeur à donner à a_q. Le but b correspond à q = 0.

---

(1) il s'agit bien entendu de la phase de traduction.

NTABW : contient le nombre de lignes occupées dans TABW.

NBARG : contient le nombre d'arguments de l'appel.

NTRAV : contient le nombre d'inconnues liées aux arguments et au but de l'appel, c'est-à-dire le nombre de cases qui seront remplies dans TRAV.

RANGTRAV : ce tableau donne pour chaque inconnue son rang dans les lignes du résultat (c'est-à-dire dans TRAV) si elle y figure, et 0 sinon.

VIII-2-2- LES MODULES DE TRADID.

Nous allons décrire, sans entrer dans les détails, ce que fait TRADID au fur et à mesure de la lecture du programme maquette.

VIII-2-2-1- Traitement de Ek, B(Ek), EMPILER.

Les étiquettes Ek sont tout simplement reproduites dans le programme objet. Les ordres B(Ek) sont traduits en : si ¬ BOOL alors allera Ek. L'instruction EMPILER se traduit par l'ordre d'empiler BOOL sur PILBOOL.

VIII-2-2-2- Traitement de (j).

Lorsque TRADID rencontre (j), il sait que l'évaluation d'un motif va commencer et que le numéro du sous-filtre qui en est le contexte à gauche est j (correspondant au $j^{ème}$ étage de PILINC).

TRADID ajoute 1 à NI, si bien que NI contient le numéro du motif que l'on va traiter. Puis il envoie tout le $j^{ème}$ étage de PILINC dans le $i^{ème}$: les inconnues évaluées par f_j (ou son CTG) seront donc utilisables par Mi.

VIII-2-2-3- Traitement d'un appel de module de recherche $R(a_1, \ldots, a_p, \ldots, a_u ; b)$

Ce traitement a pour but de générer le programme décrit aux paragraphes VIII-1-3-1 et 2. Il peut être décomposé en deux phases : la première consiste à garnir le tableau TABW pour les arguments inconnus et à générer les instructions correspondant aux arguments fixes ; la seconde utilise TABW pour générer les autres instructions relatives à cet appel. Auparavant, un ordre d'initialisation de ARG [0] est généré (cf module A1 dans VIII-1-3-1 et 2). Nous ne parlerons pas du cas où R est affecté d'un opérateur : TRADID fait ce qui est nécessaire pour que les modifications décrites en VIII-1-3-3 soient réalisées ; de même pour le cas des sous-filtres hétéro-inconnue.

1ère phase : formation de TABW, traitement des arguments fixes et préparation de RANGTRAV.

Ce module consulte la liste des arguments et but de l'appel. Pour chaque argument a_p, trois éventualités sont possibles :

- ou bien a_p est fixe. Dans ce cas, TRADID génère l'ordre de mise en place de sa valeur dans ARG[p]. Il en est de même pour le but: s'il est fixe, l'ordre d'envoi de sa valeur en ARG[0] est généré (cf module A1).

- ou bien a_p est une inconnue x_k qui "a déjà été évaluée" (c'est à l'exécution qu' à ce stade elle aura déjà été évaluée). Une remarque s'impose. Ce que le programme TRADID teste ici, c'est si l'inconnue x_k a une ligne dans PILINC (pour l'étage en cours) ; si elle n'en a pas, c'est qu'elle n'a jamais été évaluée ; si elle en a, il se peut cependant qu'elle possède la valeur "non déterminée" dans certains k-uples : ceci peut se produire si le contexte à gauche de Mi est hétéro-inconnue. Nous avons signalé ce cas à la fin du paragraphe VIII-1-3-1, et avons indiqué comment il fallait aménager le programme objet pour le traiter. Mais nous avons décidé de ne pas parler ici du rôle de TRADID dans cet aménagement car il va de soi.

Si donc a_p est une inconnue x_k ayant déjà été évaluée, on trouve dans PILINC [i, k, 1] le numéro j du tableau Wj où se trouvent les valeurs de x_k. Si le tableau Wj est déjà inscrit dans TABW, on se contente de noter dans la ligne correspondante que l'inconnue a_p est à prendre dans le tableau Wj, au rang indiqué par PILINC [i, k, 3] (dans chaque ligne). Si par contre Wj n'est pas encore dans TABW, on l'y met.

De même pour le but b.

- ou bien a_p est une inconnue x_k n'ayant pas encore été évaluée (inconnue propre pour Mi et pas encore rencontrée au cours de EVAL(Mi)). Dans ce cas, il faut que TRADID génère un module de programme pour lui créer une valeur par ailleurs, comme nous l'avons expliqué à propos du module A2 dans le paragraphe VIII-1-3-2. TRADID modifie en conséquence PILINC et TABW ainsi que leurs compteurs. Pour pouvoir en dire plus, il faudrait avoir décrit la partie cadrage et approfondi l'étude des propriétés des relations de la structure. Si c'est le but qui est dans ce cas, on ne fait rien ici.

<u>Fin de la 1ère phase</u> : <u>classement de TABW et préparation de RANGTRAV</u>.

A ce stade, TABW est complet. On cherche alors un ordre sur ses lignes (cf VIII-1-3-2, module B_v) : cet ordre est celui des valeurs de TABW [z, 2] en décroissant. Les lignes de TABW sont donc classées dans l'ordre où les tableaux vont entrer en jeu. Appelons, pour le $v^{ème}$ ligne,

j_v le contenu de TABW[v, 1] (numéro de W)

h_v le contenu de TABW[v, 3] (longueur des lignes de Wj_v)

d_v^q le contenu de TABW[v, 4 + q] (rang du $q^{ème}$ argument dans chaque ligne de Wj_v , ou 0 s'il n'y figure pas).

A la fin de cette phase, TRADID prépare dans RANGTRAV le dessin du tableau TRAV, c'est-à-dire du tableau de résultats de l'appel. Si par exemple les inconnues x_1, x_3, x_6 sont liées aux arguments ou but, alors TRAV sera constitué d'une valeur de x_1 (qui aura le rang 1), d'une valeur de x_3 (qui aura le rang 2), d'une de x_6 (qui aura le rang 3). Pour former RANGTRAV, TRADID passe en revue l'étage i de PILINC ; il effectue :

NTRAV : = 0 ;
<u>pour</u> q : = 1 <u>pas</u> 1 <u>jusqua</u> NBINC <u>faire</u>
<u>si</u> PILINC [i, q, 1]= 0 <u>ou</u> n'appartient pas à TABW (c'est-à-dire n'est pas impliqué dans cet appel) <u>alors</u>
RANGTRAV [q] : = 0 <u>sinon</u>
<u>début</u> NTRAV : = NTRAV + 1 ; RANGTRAV[q]: = NTRAV <u>fin</u>

<u>2ème phase : génération de la partie principale du programme relatif à l'appel.</u>

- <u>Génération de A3</u> :

TRADID génère le module A3. Le numéro du tableau de résultats est égal à NW + 1.

- <u>Génération des modules B_v</u> :

TRADID génère d'abord le début de bouclage :
<u>Pour</u> "I" v : = "IMPLANW"j_v ,"I"v + h_v <u>tantque</u> "W" ["I"v] ≠ "#" <u>faire</u>

Nous avons mis entre guillemets les symboles autres que les symboles de base qui se retrouvent tels quels dans le programme généré ; v, j_v, h_v sont remplacés par leur valeur.

Ensuite, si v ≠ NTABW, le module B_v1 est généré : "NOUVI" v : = <u>vrai</u>.

Enfin le module B_v2 est généré. Pour cela, TRADID exécute :

<u>Pour</u> q : = 0 <u>pas</u> 1 <u>jusqua</u> NBARG <u>faire</u>
<u>si</u> $d_v^q \neq 0$ <u>alors</u> générer ARG [q]:="W"["I"v + d_v^q].

- <u>Génération du module C</u> :

Il s'agit d'un simple appel de la procédure R.

- <u>Génération des modules H1 et H2</u> :

<u>Si</u>(b est une inconnue x_k <u>et</u> PILINC [i, k, 1] ≠ 0 <u>et</u> le tableau qui délivre x_k ne délivre aucun argument) <u>alors</u> TRADID génère les modules H1 et H2.

- <u>Génération du module H3</u> :

Cette génération ne soulève aucun problème.

- <u>Génération du module I</u> :

On génère d'abord l'affectation de "vrai" à BOOL.
Puis a lieu la génération des modules I_v.
Pour chaque v, TRADID effectue les opérations suivantes :

- si v ≠ NTABW , il génère : <u>si</u> "NOUVI" v <u>alors</u> <u>début</u>
- <u>Pour</u> h : = 1 <u>pas</u> 1 <u>jusqua</u> NBINC <u>faire</u>
<u>si</u> PILINC [i, h, 1] = j_v <u>alors</u> générer :
"TRAV" [RANGTRAV [h]] : ="W" ["I"v +PILINC [i, h, 3] -1]
La valeur de PILINC [i, h, 3] - 1 est calculée et c'est elle qui est placée après "I"v +.

- Génération du module J1 :

Si b est fixé ou est tel que le tableau qui le délivre délivre aussi des arguments, alors TRADID génère J1. La longueur des lignes de TRAV est donné par NTRAV.

- Génération du module J2 :

Si b est une inconnue x_k évaluée (PILINC [i, k, 1] ≠ 0) mais telle que le tableau Ww qui la délivre ne délivre aucun argument (cela a déjà été testé pour H), alors TRADID génère le module J2. En ce qui concerne les affectations à TRAV, il cherche dans TABW quelles sont les inconnues délivrées par le même tableau que x_k et, pour chacune, cherche dans RANGTRAV son rang dans TRAV, et dans PILINC son déplacement dans chaque ligne de Ww.

- Génération de J3 :

Dans le troisième cas, TRADID génère J3.

- Génération de $Bbis_v$:

Pour chaque valeur de v, TRADID génère : fin

- Génération de G et mise à jour du tableau CRECUPW :

+ TRADID passe en revue TABW. Pour chaque tableau qui y figure, si sa case dans CRECUPW contient 1 (tableau résultat intermédiaire), alors on peut noter sa récupération, à la traduction et à l'exécution. Ainsi, dans ce cas, TRADID effectue :

- affectation de 0 à la case considérée de CRECUPW
- génération de : RECUPW [numéro du tableau] : = "Vrai".

+ TRADID génère ensuite les derniers ordres de G1, puis G2.

- Mise à jour de PILINC [i,.,.] .

A la fin du traitement de l'appel du module de recherche, toutes les inconnues liées aux arguments ou but de l'appel sont liées entre elles et figurent dans le tableau d'indice (CNW) + 1, au rang indiqué par RANGTRAV. TRADID fait donc les mises à jour suivantes :

```
CNW := CNW + 1 ;
Pour k : = 1 pas 1 jusqua NTRAV faire
si  RANGTRAV [k] ≠ 0 alors
début  PILINC [i, k, 1] : = CNW ;
       PILINC [i, k, 3] : = RANGTRAV [k] ;
       PILINC [i, k, 4] : = NTRAV
fin ;
CRECUPW [CNW] : = 1 .
```

VIII-2-2-4- Traitement d'une étiquette Fi :

L'étiquette Fi marque la fin de la suite d'appels de modules de recherche constituant EVAL(Mi). On peut donc mettre à jour CRECUPW : tous les éléments qui sont à 1 correspondent à des tableaux qui constituent le résultat du motif Mi; on leur donne donc la valeur 2 (cf § VIII-2-1-1). De plus, l'étiquette Fi est reproduite dans le programme objet.

Ensuite intervient la phase de préparation du transfert éventuel du dernier tableau de résultats, pour gagner de la place. Les conditions de ce transfert ont été expliquées aux paragraphes VIII-1-4 et VIII-1-5-2. Nous n'entrerons pas dans les détails mais donnerons un plan général de cette phase.

Il est important à ce stade de savoir si le résultat de Mi va faire l'objet d'une réunion avant l'évaluation d'un autre motif ou pas. Pour cela, il faut effectuer des lectures à l'avance dans le programme maquette. Si une réunion intervient avant le prochain EVAL, on ne fait plus rien pour Fi ; on continue à traiter normalement la chaîne lue à partir de Fi. Sinon, on effectue ce qui est expliqué en VIII-1-5-2 (y compris pour le module V) en tenant compte des instructions DECHU qui interviennent avant la rencontre du prochain EVAL (les lectures à l'avance utiles ici peuvent être combinées avec les précédentes) : ces instructions DECHU sont exécutées avant que soit déterminé le numéro du tableau récepteur du transfert.

VIII-2-2-5- Traitement d'une instruction REU(i).

Il s'agit de la réunion des résultats des sous-filtres de numéros j et i, i étant le contenu de NI, tels que $j < i$.

Le paragraphe VIII-1-4 contient une description détaillée de l'objet de ce programme. Contentons-nous de donner quelques précisions sur les renseignements qu'il utilise pour mener à bien la génération du programme objet.

TRADID doit commencer par chercher Xr = Xi U Xj (rappelons que pour chaque k on appelle Xk l'ensemble des inconnues figurant dans le sous-filtre de numéro k ou son CTG). En classant les éléments de Xr, il constitue une image des lignes du tableau de résultats Wr, image qu'il exprime en garnissant le tableau RANGTRAV (et son compteur NTRAV). Pour chaque inconnue déjà évaluée, RANGTRAV contient son rang dans ces lignes si elle est élément de Xr et 0 sinon.

Ensuite, pour chacun des deux sous-filtres, TRADID doit fabriquer un tableau ayant la même forme que TABW. Appelons-les TABW (pour f_i) et TABWBIS (pour f_j). Il les construit en consultant respectivement l'étage i et l'étage j de PILINC. Ce sont ces tableaux et RANGTRAV qui lui servent pour générer les modules M et N. En même temps, il rend récupérables les tableaux qui ont été ainsi libérés.

Pour le module P, TRADID peut se contenter de générer un appel à une procédure de réunion.

La génération du module Q ne soulève pas de problème particulier à part le fait qu'une lecture à l'avance doit être pratiquée : ceci pour que les ordres DECHU existant éventuellement entre l'instruction REU(j) et une instruction amenant à élaborer un autre résultat soient exécutées avant que l'on détermine le numéro du tableau récepteur.

VIII-2-2-6- Traitement de DECHU(j).

Ce traitement consiste seulement à rendre récupérables à la traduction, et générer les ordres rendant récupérables à l'exécution, les tableaux constituant le résultat du sous-filtre f_j.

IX - APPLICATIONS, DEVELOPPEMENTS, CONCLUSION.

oOOo

Nous avons conçu ce méta-système à la lumière de l'expérience que nous avions de la réalisation de systèmes de données, avec un souci constant de formalisation et d'indépendance mais en ne perdant pas de vue l'aspect pratique.

En ce qui concerne l'indépendance entre les différents constituants d'un système de données, nous pouvons dire que le méta-système que nous proposons la réalise assez bien : on peut réfléchir à la structure logique sans faire intervenir de notions de représentation ou d'implémentation ; modifier une structure logique en laissant inchangées les informations enregistrées (seule est alors à modifier la correspondance entre LOG et PHYLOG) ou l'enrichir sans avoir à remettre en cause toute l'implémentation ; inversement, modifier la représentation, même dans sa logique, sans que cela apparaisse aux yeux de l'utilisateur et sans que l'effort de programmation soit important. Ainsi on peut faire voir une même donnée de façons différentes par des utilisateurs ou des applications différents, ou au contraire avoir un mode d'utilisation unique pour des informations hétérogènes. Cette indépendance s'étendra aux aspects d'acquisition et de modification.

Les objectifs que nous nous fixions au début de cette thèse seront en grande partie atteints grâce à cette indépendance et à la formalisation.

Notre objectif est maintenant de mettre notre système à l'épreuve.

- A l'épreuve de la réalisation effective, que nous avons conçue de près mais qui, pour une grande part, reste à mettre en oeuvre.

- A l'épreuve des applications. Nous avons commencé à appliquer ce système à nos anciennes réalisations ; mais nous voudrions aussi élargir le champ des essais et par exemple, au début, traiter des applications déjà effectuées en Socrate ou dans d'autres SGBD. Alors que, dans un premier temps, nous ne prendrons pas en compte l'acquisition ni les modifications ni les problèmes liés au "cadrage" (cf § IV-1-1), tous ces points essentiels étant alors traités de façon "ad hoc", ils seront petit à petit incorporés à l'ensemble ; nous avons commencé à aborder les problèmes d'acquisition. Avec l'aspect "cadrage" apparaîtront les problèmes de gestion de gros fichiers sur mémoires secondaires diverses.

  Les applications nous amèneront sans doute à préciser des notions telles que les ensembles d'appartenance de valeurs ou de repères, les prédicats, les contraintes d'intégrité. Mais il nous semble que ces notions entrent bien dans le cadre de notre formalisation.

  En ce qui concerne les "grosses" applications, nous espérons les étudier dans le cadre élargi de la prise en compte de gros problèmes informatiques ; un tel projet commence à se dessiner, conçu comme une généralisation de Civa [20] et de Remora [48,49]. Il stipule qu'on puisse décrire intrinsèquement l'univers d'un problème et que, dans l'utilisation d'un langage de très haut niveau, la description de l'objet d'un programme puisse comporter la spécification d'accès à cet univers.

- A l'épreuve de la comparaison avec d'autres systèmes et de tentatives pour situer notre système par rapport à des études systématiques ou comparatives menées par des

personnalités ou des groupes d'étude [2, 3, 13 à 16, 18, 19, 43].

  Nous pensons en particulier qu'il sera utile de situer le système PIVOINES par rapport à d'autres systèmes relationnels, comme ceux dont les langages de requête se nomment ALPHA [14,15], SQUARE [5], FQL [44] (cf [43]).

- A l'épreuve de tests d'efficacité de fonctionnement. Nous avons déjà évoqué, au paragraphe VII-5, le problème posé par la qualité de la stratégie obtenue à l'issue de la traduction d'un filtre. Lorsque l'information interrogée est très volumineuse et les recherches élémentaires longues ou coûteuses en place (1), la qualité du programme finalement obtenu est largement prépondérante (et cette qualité est essentiellement liée à celle de la stratégie). Mais souvent les phases de traduction seront très onéreuses et il y aura peut-être des compromis à établir entre complexité des phases de traduction et efficacité du produit.

  Nous avions tenté d'évaluer le nombre de tests et de transferts à effectuer lors du traitement d'un appel de module de recherche. Mais le nombre des paramètres et leurs marges de variation étaient tels que des approximations ou des moyennes étaient très difficiles à effectuer et trop sujettes à caution ; par conséquent aussi le résultat final. Nous y avons renoncé. Peut-être pourrons-nous avoir recours à la simulation sur ordinateur.

---

(1) Le programme exécutable est alors fortement répétitif : nombreux appels des mêmes procédures de recherche car les jeux de valeurs des arguments sont nombreux, nombreuses manipulations de résultats dans les opérations de réunion.

Au cours des réalisations, nous prendrons en tout cas soin de placer de nombreux compteurs afin de pouvoir observer les performances de chaque phase de l'élaboration puis de l'exploitation du programme objet.

- A l'épreuve des tentatives de perfectionnement. En plus des projets que nous venons d'exposer et qui, eux, sont pour la plupart nécessaires pour que notre système existe vraiment, nous avons des idées de perfectionnement de celui-ci, dont voici les principales.

Nous voudrions utiliser davantage les propriétés des relations de la structure logique et les exploiter de façon plus explicite. En effet, pour l'instant, elles interviennent dans la traduction de LOG en PHYLOG, dans les degrés d'indéterminisme, dans le fait de fournir éventuellement des valeurs aux inconnues lorsqu'elles en manquent -et encore n'avons-nous pas approfondi ce point-. En essayant d'utiliser davantage ces propriétés, nous nous inspirerons de systèmes inférentiels tels que Syntex [38] , Planner[29] et tels que les travaux de A. Waksman [55] et d'autres.

Nous voudrions aussi exprimer de façon plus mathématique que par des procédures la correspondance entre la structure PHYLOG et la représentation. En effet, cette façon d'exprimer la correspondance ne peut servir pour l'acquisition. Or il existe certainement un noyau de renseignements sur la représentation qui sont utiles aussi bien pour l'accès que pour l'acquisition.

De plus, effectuer cette formalisation permettrait de mieux étudier les problèmes de choix d'une bonne représentation, pour une structure logique donnée et des besoins d'accès exprimés. Peut-être même pourrait-on réaliser une aide automatisée au choix de cette représentation.

Que de travail encore...

UNIVERSITÉ POLYTECHNIQUE NANCY

B I B L I O G R A P H I E

oOOo

B I B L I O G R A P H I E

1- J.R.ABRIAL :"Projet SOCRATE, cours d'architecture des systèmes". -L'Alpe d'Huez- Décembre 1972.

2- J.R. ABRIAL : "Data Semantics" IFIP Working Conference- Cargèse (Corse) Avril 1974.

3- ANSI-SPARC "Study Group on Data Base Management Systems" -Second interim report- Février 1975.

4- F. BELLEGARDE : "FACE, langage d'écriture de compilateurs. Définition et implémentation"- Thèse de spécialité. Université de Nancy I- 1972.

5- R.F. BOYCE, D.D. CHAMBERLIN, MM. HAMMER, W.F. KING : "Specifying Queries as Relational Expressions : SQUARE". SIGPLAN Notices -Vol 10- N°1- Janvier 1975.

5bis(1)

6- C.E.T.E. Aix-en-Provence : "Etude comparative de classes de softwares de gestion et d'interrogation de fichiers" -Colloque IRIA- Banques de Données- Aix- Juin 1973.

7- C.E.T.E. MIISFIIT

8- Club Banques de Données (IRIA). Bulletins de liaison.

9- D.C. CHILDS : "Feasibility of a Set Theoretic Data Structure". Proc. of the IFIP congress 1968 -Vol 1 (420,430)- North Holland Publ.

10- CODASYL Data Base Task Group : Reports 1969 et 1971. ACM New York.

11- CODASYL Systems Committee Technical Report : "Feature Analysis of Generalized Data Base Management Systems". ACM New York 1971.

12- CODASYL Data Description Language Committee-Handbook n°113. National Bureau of Standards- Janvier 1974 (US Government Printing Office- C 13.6/2 : 113).

13- E.F. CODD : "A Relational Model of Data for Large Shared Data Banks". CACM. Vol 13. n°6. Juin 1970.

14- E.F. CODD : "A Data Base Sublanguage Founded on the Relational Calculus". Proc. 1971 ACM SIGFIDET Workshop on data description, access and control.

(1) 5bis-G. Bracchi, P. Paolini, G. Pelagatti : "binary logical associations in data modelling". Rapport 75-12 Lab. Calc. (Milan)

15- E.F. CODD : "Normalized Data Base Structure : A Brief Tutorial". Même référence que ci-dessus.

16- E.F. CODD : "Further Normalization of the Data Base Relational Model". Courant Computer Science Symposia 6 "Data Base Systems". Mai 71.

17- M. CREHANGE : "Description, représentation, interrogation, traitement des informations structurées. Langage PIVOINES". RAIRO. Septembre 1974. B-3 p. 5-43.

18- C. DELOBEL : "Contributions théoriques à la conception et l'évaluation d'un système d'informations appliqué à la gestion". Thèse d'état-Université Scientifique et Médicale de Grenoble. 1973.

19- C. DELOBEL : "Les systèmes de base de données". Ecole d'été de l'AFCET. Rabat 1975.

20- J.C. DERNIAME : "Le projet CIVA. Un système de programmation modulaire". Thèse d'Etat. Université de Nancy I. Janvier 1974.

21- J. EARLEY : "Toward an understanding of data structures". Communications of ACM. Octobre 1971 .

22- J.A. FELDMAN et P.P. ROVNER : "An Algol-based associative language, LEAP". CACM . Août 1969.

23- J.P. FINANCE et J.L. REMY : "Structure d'information et sémantique d'un langage de programmation". 3ème Ecole d'été de l'AFCET. Grenade 1973.

24- R.W. FLOYD : "Bounded Context Syntactic Analysis". Communications ACM. 7. n°2. Février 1964.

24bis- R.W. FLOYD : "Nondeterministic Algorithms" . JACM Vol 14 n°4. Oct 67. (636 à 644).

25- J.M. FOSTER : "Automatic Syntax Analysis"

26- J.C. GARDIN, R.C. CROS, F. LEVY : "L'automatisation des recherches documentaires. Un modèle général : SYNTOL". Gauthier-Villars.

27- R.E. GRISWOLD, J.F. POAGE, I.P. POLONSKY : "The SNOBOL 4 programming language". Prentice-Hall. 1971.

28- GUIDE-SHARE data base requirements group. Data Base Management System Requirements. Nov 1971.

29- C. HEWITT : "Description and Theoretical Analysis (using Schemata) of PLANNER : a language for proving theorems and manipulating models in a robot". National Technical Information Service. US Department of Commerce.

30- C.A.R. HOARE : "Proof of correctness of data representation". Acta Informatica 1 (1972). n°1.

31- J. JARAY : "Le langage Snobol 4. Ses applications, son implémentation". Thèse de Spécialité. Université de Nancy I. Juin 1975.

32- S.C. KLEENE : "Introduction to Metamathematics". Van Nostrand.

33- D.E. KNUTH : "Top Down Syntax Analysis". Acta Informatica-1 (79, 110). 1971.

34- D.E. KNUTH : "Semantics of Context- Free Languages". Mathematical Systems Theory. Vol 2. N°2 (127-145).

35- R.E. LEVIEN, M.E. MARON : "A computer System for Inference Execution and Data Retrieval". Comm. ACM 10, 715. Novembre 1967.

36- P. MARCHAND : "Etude et Classification des Bigrammaires. Applications à l'étude des systèmes tranformationnels". Thèse de Spécialité. Université de Nancy I. Novembre 1974.

37- J. MAROLDT : "Définition de FACE, langage pour l'écriture des compilateurs. Implémentation d'un sous-ensemble". Thèse de Docteur-Ingénieur. Université de Nancy I. 1972.

38- J. M. NICOLAS : "SYNTEX . Déduction automatique d'informations implicites". RAIRO. Septembre 74.B-3, p. 45-64. Suivi de : "LINUSS : langage d'interrogation naturel utilisé dans le système SYNTEX" (J.C. SYRE).

39- C. PAIR : "Cours sur les structures d'informations". 1ère Ecole d'été de l'AFCET (Alès 1971).

40- C. PAIR : "Cours de Compilation". Université de Nancy

41- C. PAIR et J.P. FINANCE : "Formalisation des notions de donnée, d'information et de structure d'information". Université de Nancy II. Octobre 1973.

42- D.L. PARNAS : "A technique for software module specification with examples". CACM. Mai 1972.

43- A. PIROTTE : "Comparaison de langages d'interrogation de bases de données relationnelles". Ecole d'été de l'AFCET. Rabat. 1975.

44- A. PIROTTE, P. WODON : "A comprehensive formal query language for a relational data base : FQL" MBLE Report R 283, Déc. 74, Revised June 1975.

45- A. QUERE : "Etude des ramifications et des bilangages". Thèse de Spécialité. Université de Nancy. 1969.

46- F.L. de REMER : "Simple LR(k) grammars". Comm. ACM 14 n°7. Juillet 1971.

47- W.P. de ROEVER : "Operational, mathematical and axiomatized semantics for recursive procedures and data structures". Mathematical Centre, Report ID/1. Amsterdam.

48- C. ROLLAND : "Le projet REMORA : un système de pilotage pour la conduite de projets informatiques". Informatique et gestion. Juillet-Août 1975.

49- C. ROLLAND : "Engineering of Management Informatic Systems". Congrès INTERNET-AFCET. Octobre 1974.

50- D.J. ROZENKRANTZ, R.E. STEARNS : "Properties of Deterministic Top Down Grammars". Information and Control. 17 (226, 256). 1970.

51- G. SALTON : "Manipulation of trees in information retrieval". CACM 1962.

52- M.E. SENKO, E.B. ALTMAN, M.M. ASTRAHAN, P.L. FEHDER : "Data structures and accessing in data-base systems". IBM Systems Journal. N°1. 1973.

53- A.J. STRNAD : "The Relational Approach of the Management of Data Bases". Information processing. 71. p. 801.

54- G. VEILLON : "Modèles et algorithmes pour la traduction automatique". Thèse d'Etat. Université de Grenoble. Avril 1970.

55- A. WAKSMAN : "Information Retrieval and the Query Languages". Proceedings of ACM SIGPLAN-SIGIR interface meeting (Sigplan notices. Volume 10. N°1. Janvier 1975).

56- N. WIRTH, H. WEBER : "Euler, a Generalization of Algol and its Formal Definition". CACM. 9. n°1, Janvier 1966 et 2, Février 1966.

Systèmes des constructeurs ou sociétés :

60- INFOL (Control Data)

61- TDMS (SDC-IBM)

62- IMS (IBM)

63- MAGIS (Philips)

64- IDS (Honeywell-Bull)

65- MISTRAL (CII)

66- SYSTEM 2000 (MRI)

....

Réalisations de notre équipe :

70- L. FOSSIER et M. CREHANGE : "Un essai de traitement sur ordinateur de documents diplomatiques du Moyen-Age". Annales (Janvier 1970).

71- J. MARTIN, P. GERMAIN, J.M. MARTIN, P. DROUIN, G. DEBRY, M. CREHANGE : "Une banque de dossiers médicaux à expression semi-libre". Journées Banques de Données. Aix-en-Provence. Juin 1971.

72- J.M. MARTIN : "Un mode d'exploitation du dossier médical: structure arborescente, système modulaire d'interrogation". Thèse de Spécialité . Université de Nancy.

73- P. GERMAIN : "Système de Gestion et d'Exploitation Documentaire d'un Corpus de Dossiers Médicaux". Thèse de Spécialité. Université de Nancy. 1972.

74- C. de BARY : "Conception et réalisation d'un logiciel de gestion et d'interrogation d'une banque de données géologiques". Thèse de spécialité. Université de Nancy I. 1974.

75- Y. SCHNEPF : "Système informatique de documentation pour l'assistance cardio-circulatoire". Thèse de spécialité. Université de Nancy I. 1974.

76- C. GRIMM : "Génération du programme maquette d'un motif de PIVOINES". Rapport de DEA. Université de Nancy I. 1975.

77- PH. GRANDCLAUDE : "Contribution à la méthodologie d'un système d'information en géologie. Application à la géochimie". Thèse d'Etat. Université de Nancy I. Mars 1974. (Cette thèse ne fait pas partie des travaux de notre équipe mais englobe les conclusions des travaux de C. de Bary [74]).

NOM DE L'ETUDIANT : Madame CREHANGE née CAEN Marion

NATURE DE LA THESE : DOCTORAT ES SCIENCES - THESE D'ETAT -

VU, APPROUVE

& PERMIS D'IMPRIMER

NANCY, le 10/11/1975

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

J.R. HELLUY