50376
1974
116

N° d'ordre : 465

50376
1974
116

THESE

présentée à

L'UNIVERSITÉ DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir le titre de

DOCTEUR DE SPÉCIALITÉ
(Informatique)

par

Bernard TOURSEL

BIBLIOTHÈQUE UNIVERSITAIRE
Section de SCIENCES
LILLE

APPLICATION DES FLOTS BOOLEENS CONTROLES A LA STRUCTURATION DES PROCESSUS ET A L'OPTIMISATION DES MICROPROGRAMMES

Soutenue le 21 Juin 1974, devant la COMMISSION D'EXAMEN

Membres du Jury :	MM.	P. BACCHUS	Président
		V. CORDONNIER	Examinateur
		C. CARREZ	Rapporteur

UNIVERSITE DES SCIENCES
ET TECHNIQUES DE LILLE.

DOYENS HONORAIRES

MM. H.LEFEBVRE, PARREAU.

PROFESSEURS HONORAIRES

M. ARNOULT, Mme BEAUJEU, MM. BEGHIN, BROCHARD, CAU, CHAPPELON, CHAUDRON, CORDONNIER, DEHEUVELS, DEHORNE, DEHORS, FAUVEL, FLEURY, P.GERMAIN, HEIM DE BALSAC, HOCQUETTE, KAMPE DE FERIET, KOURGANOFF, LAMOTTE, LELONG, Mme LELONG, MM. LIEBAERT, MARTINOT-LACARDE, MAZET, MICHEL, NORMANT, PARISELLE, PASCAL, PAUTHENIER, PEREZ, ROIG, ROSEAU, ROUBINE, ROUELLE, WIEMAN, ZAMANSKY.

PROFESSEURS TITULAIRES

M.	ANGRAND Jean Pierre	Géographie et Aménagement Spatial
M.	BACCHUS Pierre	Astronomie et Calcul
M.	BEAUFILS Jean Pierre	Chimie Générale
M.	BECART Maurice	I.U.T. Lille
M.	BLOCH Vincent	Psychophysiologie
M.	BIAYS Pierre	Géographie et Aménagement Spatial
M.	BONNEMAN Pierre	Chimie Industrielle
M.	BONTE Antoine	Géologie Appliquée
M.	BOUGHON Pierre	Mathématiques
M.	BOURIQUET Robert	Biologie Végétale
M.	CAPET Marcel-Francis	Institut de Préparation aux Affaires
M.	CELET Paul	Géologie Générale
M.	CONSTANT Eugène	Electronique
M.	CORSIN Pierre	Paléobotanique
M.	DECUYPER Marcel	Mathématiques
M.	DEDECKER Paul	Mathématiques
M.	DEFRETIN René	Biologie Animale - Directeur de l'Institut de Biologie Maritime de Wimereux
M.	DELATTRE Charles	Géologie Générale
M.	DURCHON Maurice	Biologie Animale
M.	FLATRES Pierre	Géographie et Aménagement Spatial
M.	FOURET René	Physique
M.	GABILLARD Robert	Electronique
M.	GEHU Jean Marie	Institut Agricole
M.	GLACET Charles	Chimie Organique
M.	GONTIER Gérard	Mécanique des Fluides
M.	GUILLAUME Jean	Biologie Végétale
M.	HEUBEL Joseph	Chimie Minérale
Mme	LENOBLE Jacqueline	Physique
M.	MONTREUIL Jean	Chimie Biologique
M.	POUZET Pierre	I.U.T. Lille

Mme	SCHWARTZ Marie Hélène	Mathématiques
M.	TILLIEU Jacques	Physique
M.	TRIDOT Gabriel	Chimie Minérale Appliquée
M.	VIDAL Pierre	Automatique
M.	VIVIER Emile	Biologie Animale
M.	WATERLOT Gérard	Géologie et Minéralogie
M.	WERTHEIMER Raymond	Physique

PROFESSEURS A TITRE PERSONNEL

M.	BOUISSET Simon	Physiologie Animale
M.	DELHAYE Michel	Chimie Physique et Minérale 1er Cycle
M.	LEBRUN André	Electronique
M.	LINDER Robert	Biologie Végétale
M.	LUCQUIN Michel	Chimie Physique
M.	PARREAU Michel	Mathématiques
M.	PRUDHOMME Rémy	Sciences Economiques et Sociales
M.	SAVARD Jean	Chimie Générale
M.	SCHALLER François	Biologie Animale
M.	SCHILTZ René	Physique

PROFESSEURS SANS CHAIRE

M.	BELLET Jean	Physique
M.	BODARD Marcel	Biologie Végétale
M.	BOILLET Pierre	Physique
M.	DERCOURT Jean Michel	Géologie et Minéralogie
M.	DEVRAINNE Pierre	Chimie Minérale
M.	LOMBARD Jacques	Sciences Economiques et Sociales
Mle	MARQUET Simone	Mathématiques
M.	MONTARIOL Frédéric	Chimie Minérale Appliquée
M.	PROUVOST Jean	Géologie et Minéralogie
M.	VAILLANT Jean	Mathématiques

MAITRES DE CONFERENCES (et chargés de fonctions)

M.	ADAM Michel	Sciences Economiques et Sociales
M.	ANDRE Charles	Sciences Economiques et Sociales
M.	AUBIN Thierry	Mathématiques Pures
M.	BEGUIN Paul	Mécanique des Fluides
M.	BILLARD Jean	Physique
M.	BKOUCHE Rudolphe	Mathématiques
M.	BOILLY Bénoni	Biologie Animale
M.	BONNEMAIN Jean Louis	Biologie Végétale
M.	BONNOT Ernest	Biologie Végétale
M.	BRIDOUX Michel	I.U.T. Béthune
M.	BRUYELLE Pierre	Géographie et Aménagement Spatial

M.	CAPURON Alfred	Biologie Animale
M.	CARREZ Christian	Calcul
M.	CHOQUET Marcel	I.U.T. Lille
M.	CORDONNIER Vincent	Calcul
M.	CORTOIS Jean	Physique
M.	COULON Jean Paul	Electrotechnique
M.	DEBRABANT Pierre	Sciences appliquées
M.	ESCAIG Bertrand	Physique
Mme	EVRARD Micheline	I.U.T. Lille
M.	FAIDHERBE Jacques	Biologie Animale
M.	FONTAINE Jacques	I.U.T. Lille
M.	FROELICH Daniel	Sciences Appliquées
M.	GAMBLIN André	Géographie et Aménagement Spatial
M.	GOBLOT Rémi	Mathématiques
M.	GOSSELIN Gabriel	Sciences Economiques et Sociales
M.	GOUDMAND Pierre	Chimie Physique
M.	GRANELLE	Sciences Economiques et Sociales
M.	GRUSON Laurent	Mathématiques
M.	GUIBAULT Pierre	Physiologie Animale
M.	HERMAN Maurice	Physique
M.	HUARD de la MARRE Pierre	Calcul
M.	JOLY Robert	Biologie (Amiens)
M.	JOURNEL Gérard	Sciences Appliquées
Mle	KOSMANN Yvette	Mathématiques
M.	LABLACHE-COMBIER Alain	Chimie Générale
M.	LACOSTE Louis	Biologie Végétale
M.	LANDAIS Jean	Chimie Organique
M.	LAURENT François	Automatique
M.	LAVAGNE Pierre	Sciences Economiques et Sociales
Mle	LEGRAND Solange	Mathématiques
M.	LEHMANN Daniel	Mathématiques
Mme	LEHMANN Josiane	Mathématiques
M.	LENTACKER Firmin	Géographie et Aménagement Spatial
M.	LEROY Jean Marie	ENSCL
M.	LEROY Yves	I.U.T. Lille
M.	LHENAFF René	Géographie et Aménagement Spatial
M.	LOCQUENEUX Robert	Physique
M.	LOUAGE Francis	Sciences Appliquées
M.	LOUCHEUX Claude	Chimie Physique
M.	MAES Serge	Physique
Mme	MAILLET Monique	Sciences Economiques et Sociales
M.	MAIZIERES Christian	Automatique
M.	MALAUSSENA Jean Louis	Sciences Economiques et Sociales
M.	MESSELYN Jean	Physique
M.	MIGEON Michel	Sciences Appliquées
M.	MONTEL Marc	Physique
M.	MONTUELLE Bernard	I.U.T. Lille
M.	MUSSCHE Guy	Sciences Economiques et Sociales
M.	NICOLE Jacques	E.N.S.C.L.
M.	OUZIAUX Roger	Sciences Appliquées
M.	PANET Marius	Electrotechnique
M.	PAQUET Jacques	Sciences Appliquées
M.	PARSY Fernand	Mécanique des Fluides
M.	PONSOLLE Louis	Chimie (Valenciennes)
M.	POVY Jean Claude	Sciences Appliquées
M.	RACZY Ladislas	Radioélectricité
Mme	RENVERSEZ Françoise	Sciences Economiques et Sociales
M.	ROUSSEAU Jean Paul	Physiologie Animale

M.	ROYNETTE Bernard	Mathématiques
M.	SALMER Georges	Electronique
M.	SEGUIER Guy	I.U.T. Béthune
M.	SIMON Michel	Sciences Economiques et Sociales
M.	SMET Pierre	Physique
M.	SOMME Jean	Géographie et Aménagement Spatial
M.	THOMAS Daniel	Chimie Minérale Appliquée
M.	TOULOTTE Jean Marc	Sciences Appliquées
M.	TREANTON Jean René	Sciences Economiques et Sociales
M.	VANDORPE Bernard	Sciences Appliquées
M.	VILETTE Michel	I.U.T. Béthune
M.	WATERLOT Michel	Géologie Générale
Mme	ZINN JUSTIN Nicole	Mathématiques.

Je tiens à exprimer toute ma gratitude à Monsieur le Professeur BACCHUS qui me fait l'honneur d'accepter la présidence du Jury.

Que Monsieur le Professeur CORDONNIER qui a bien voulu s'intéresser à ce travail et a accepté de le juger soit assuré de ma profonde reconnaissance.

Je tiens également à remercier vivement Monsieur le Professeur CARREZ, qui m'a orienté vers cette recherche, pour les précieux conseils et suggestions qu'il m'a prodigués au cours de la réalisation de ce travail.

Je remercie également tous les membres du Laboratoire, et en particulier, Monsieur Jean Pierre STEEN, pour les idées échangées et l'attention bienveillante qu'ils m'ont témoignée .

Enfin, je remercie Mademoiselle DRIESSENS, ainsi que Monsieur et Madame DEBOCK, qui avec leur compétence et leur diligence habituelles ont assuré la réalisation matérielle de cette thèse.

à mes parents,

à ma femme,

à ma fille.

TABLE DES MATIERES

CHAPITRE 0. MICRO-PROGRAMMATION : DEFINITIONS

CHAPITRE I. CONCEPTION DE LA MEMOIRE DE COMMANDE

1. Introduction
2. Codage des micro-opérations dans la micro-instruction
   2.1. Principes
   2.2. Micro-programmation horizontale
   2.3. Micro-programmation verticale
   2.4. Conclusion
3. Enchaînement des micro-instructions
   3.1. Introduction
   3.2. Enchaînements selon les différents types de micro-programmation
   3.3. Procédés d'accélération
4. Optimisation de la mémoire de commande
   4.1. Micro-programme optimisé
   4.2. Conception de la mémoire de commande
   4.3. Autre méthode
   4.4. Optimisation

CHAPITRE II. FLOT BOOLEEN GENERALISE SUR UN GRAPHE

1. Flot booléen contrôlé sur un graphe
   1.1. Définitions
   1.2. Types de flots booléens
2. Recouvrements
   2.1. Recouvrement conjonctif d'un ensemble de fonctions booléennes
   2.2. Recouvrement séquentiel d'un graphe

3. Flot booléen généralisé

3.1. Introduction

3.2. Définition

3.3. Flot booléen généralisé semi-homomorphe d'un flot booléen

3.4. Flot booléen généralisé homomorphe d'un flot booléen

3.5. Propriété

3.6. Recouvrement ϕ-compatible d'un graphe

3.7. Propriétés

3.8. Condition nécessaire et suffisante

3.9. Théorème

3.10 Algorithme de construction

3.11 Théorème

3.12 Théorème

3.13 Algorithme de construction

3.14 Exemple

4. Problèmes de flots booléens généralisés homomorphes d'un flot booléen simple

4.1. Construction d'un f.b.g.

4.2. Construction d'un f.b.g. réalisable, optimal

CHAPITRE III. PROCESSUS ELEMENTAIRE

1. Introduction

1.1. Notion de processus

1.2. Chaînes et chemins

2. Processus élémentaire

2.1. Réseau

2.2. Processus

2.3. Sous-processus

2.4. Processus élémentaire

3. Décomposition d'un processus élémentaire

3.1. Définitions

3.2. Processus simple

3.3. Processus élémentaire fermé

3.4. Processus élémentaire cyclique

3.5. Décomposition d'un processus élémentaire

4. Déroulement d'un processus élémentaire
4.1. Introduction
4.2. Fonction état
4.3 Déroulement d'un processus élémentaire simple
4.4. Déroulement d'un processus élémentaire cyclique
4.5. Déroulement d'un processus élémentaire fermé

CHAPITRE IV. CONSTRUCTION DES DÉROULEMENTS PRINCIPAUX D'UN PROCESSUS ELEMENTAIRE

1. Introduction
2. Contexte de processus
2.1. Définition
2.2. Contexte de déroulement
3. Théorème fondamental
3.1. Théorème
3.2. Algorithme de construction
3.3. Exemple
4. Propriétés du f.b.g. du théorème fondamental
4.1. Le processus A'(P)
4.2. Le processus A(P)
4.3. Chaînes déroulables, chaînes principales
4.4. Propriétés
5. Conclusion

CHAPITRE V. OPTIMISATION DES DÉROULEMENTS REALISABLES D'UN PROCESSUS ÉLÉMENTAIRE

1. Introduction
2. Déroulements réalisables
2.1. Ensembles compatibles
2.2. Déroulement par des ensembles compatibles
3. Déroulement minimal
3.1. Déroulements équivalents
3.2. Propriété
3.3. Conséquence

4. Flot booléen généralisé réalisable
   4.1. Définition
   4.2. Théorème
   4.3. Propriétés
5. Flot booléen généralisé réalisable et minimal
   5.1. Introduction
   5.2. Déroulements réalisables minimaux
   5.3. f.b.g. réalisable minimal
6. Flot booléen généralisé optimal
   6.1. Introduction
   6.2. Recherche du f.b.g. optimal
7. Exemple
8. Conclusion

INTRODUCTION

Les techniques de micro-programmation sont, depuis quelques années, utilisées fréquemment dans la réalisation des ordinateurs et sont appliquées à des domaines variés.

La micro-programmation a permis une structuration des circuits de commande, a apporté une certaine souplesse lors de la conception du matériel et a conduit à un abaissement des coûts de réalisation.

Du fonctionnement de la micro-machine dépend les performances de tout l'ordinateur ; aussi, le problème d'optimiser la mémoire de commande micro-programmée est-il fondamental.

Après le rappel dans un chapitre liminaire des principales définitions, nous essayons, dans le chapitre I, de dégager des réalisations actuelles, les notions essentielles à prendre en compte pour résoudre cette question.

Il revient à Grasselli ([6]) d'avoir, le premier, posé le problème en terme de machine asynchrone séquentielle, représentée par un graphe.

L'optimisation des micro-programmes, liée à la définition des micro-instructions en tant qu'ensemble de micro-opérations, a été essentiellement abordée jusqu'à maintenant ([1]) sous la forme de la recherche de partitions compatibles dans un graphe. Mais ces méthodes permettent mal de tenir compte des conditions booléennes qui régissent la succession des micro-instructions.

Nous allons, dans les sections suivantes, introduire certains concepts permettant de mieux examiner ces conditions d'enchaînement

et les appliquer à la recherche d'un optimum pour des contraintes variables.

Nous définissons, au chapitre II, la notion de flot booléen contrôlé sur un graphe, pour préciser les éventualités booléennes liées aux micro-opérations, puis nous généralisons ce flot à des relations entre sous-graphes pour traduire les regroupements de sommets effectués.

Les propriétés d'homomorphie et de semi-homomorphie de ce flot généralisé nous assure que ces regroupements ne contredisent pas les propriétés du flot booléen contrôlé et une CNS d'homomorphie, ainsi qu'un algorithme de construction sont présentés.

La définition d'un flot booléen contrôlé sur un réseau permet, au chapitre III,de formaliser le concept de processus et la prise en compte de spécifications particulières permet d'identifier un processus élémentaire et un micro-programme non codé et non cadencé.

Le regroupement des tâches d'un processus élémentaire pour former une micro-instruction, la détermination des conditions à tester et des fonctions booléennes d'enchaînement sont précisés par la définition d'un déroulement du processus qui est abordée de façon progressive selon les différents types de processus élémentaires.

Nous démontrons, au chapitre IV, en utilisant la notion de contexte de processus, qu'il existe un ensemble de déroulements d'un processus élémentaire qui définit un flot booléen généralisé semi-homomorphe du flot booléen contrôlé définissant le processus. Nous en élaborons un algorithme de construction et en déduisons un diagramme d'états particulier représentant le fonctionnement du processus.

Au chapitre V, nous vérifions que l'ensemble des déroulements défini précédemment, en respectant des contraintes de simultanéité, de séquentialité et des conditions externes, est minimal.

Cependant, en terme de micro-programmation, il est nécessaire de tenir compte des contraintes liées à l'environnement physique. L'introduction de ces contraintes physiques nous conduit à aborder le problème de la construction d'un diagramme d'états qui soit réalisable et l'optimisation de ce diagramme pour des contraintes variées (en particulier temps et codage) se ramène à une application particulière des méthodes d'exploration dans les graphes.

Enfin nous citons brièvement, en conclusion, quelques prolongements possibles, en particulier, relatifs à la notion de processus.

C H A P I T R E 0

MICRO-PROGRAMMATION : DEFINITIONS

1 - INTRODUCTION

Avant d'aborder les problèmes qui peuvent apparaître lors de la conception de blocs de commandes microprogrammés, il est utile de rappeler ce que l'on entend par micro-programmation et surtout de préciser le vocabulaire employé.

Les techniques de micro-programmation, peu connues il y a quelques années, se rencontrent de plus en plus fréquemment dans des domaines variés se rattachant à l'ordinateur, aussi bien à un niveau purement physique que dans une optique langage et programmation.

L'origine de la micro-programmation correspond à l'article de Wilkes, paru en 1951, qui présente celle-ci comme une méthode systématique de concevoir les circuits de contrôle d'un calculateur digital. Initialement, il s'agit donc, à un niveau uniquement technologique, de structurer l'unité de commande d'un calculateur en s'appuyant essentiellement sur l'emploi d'une mémoire de contrôle utilisée seulement en lecture. Progressivement cependant, la micro-programmation va occuper une place spécifique et s'intercaler entre le domaine du câblé et celui du programmé. La micro-programmation apparaît actuellement et parfois de façon essentielle, dans des domaines variés :

- circuits de contrôle-commande divers (unité de commande, canaux d'entrée-sortie, périphériques)
- systèmes opératoires (primitives de systèmes, traducteurs)
- aide à la conception (structuration, simulation, reconsidération de l'architecture dans un contexte microprogrammé)
- émulation
- utilisation en temps réel pour la conduite de processus industriels
- fiabilité et aide à la maintenance (microdiagnostic, reconfiguration)
- calculabilité et théorie des automates
- utilisation pédagogique.

2 - DEFINITIONS

Cependant, en dernier lieu, la micro-programmation agit sur les opérateurs de la machine à un niveau élémentaire et est donc un moyen d'assurer un contrôle direct des circuits d'un ordinateur pour l'exécution de certaines tâches.

2.1 Micro-opérations

On peut admettre que, dans un calculateur, l'exécution d'une instruction ne nécessite que des transferts (de registre à registre, de registre à mémoire, de mémoire à registre). Ces transferts peuvent être des transferts directs, ou bien, les informations circulent au travers d'opérateurs où elles subissent une transformation. Les circuits élémentaires étant statiques, la réalisation de ces transferts s'effectue par la mise en communication des registres entre eux et donc par l'ouverture de portes logiques. Dans un calculateur de structure classique, ces circuits sont câblés de manière définitive : ce sont des circuits ET dont une entrée est commandée par le décodage du code opération de l'instruction et l'autre par l'horloge du calculateur. A chaque cycle, l'impulsion émise par l'horloge provoque l'ouverture d'une ou plusieurs portes. L'ouverture d'une porte est la commande la plus élémentaire qui soit ; on dénommera **micro-opérations** ces commandes élémentaires.

2.2 Micro-instructions

L'exécution d'une instruction (en langage machine) peut se décomposer en plusieurs tâches successives (accès à l'instruction, décodage du code opération, accès aux opérandes, exécution) et chacune de ces tâches est une succession d'actions enchaînées entre elles et synchronisées, à l'aide des impulsions d'horloge, par le bloc de commande de l'ordinateur. Chacune de ces actions dont la durée est d'un cycle d'horloge (mais ce n'est pas toujours le cas) sera appelée **micro-instruction.** Une micro-instruction représente donc un ensemble de signaux émis simultanément pendant un cycle d'horloge ; c'est un ensemble de micro-opérations.

2.3 Micro-programme

La micro-programmation consiste donc en la décomposition d'une instruction en une succession de micro-instructions : on appellera **micro-programme** la séquence de ces micro-instructions. L'exécution d'une instruction de langage machine consiste alors en l'exécution du micro-programme correspondant, c'est-à-dire en une succession d'accés d'exécution de chacune des micro-instructions qui composent le micro-programme.

2.4 Mémoire de commande

Le temps supplémentaire nécessaire pour l'enchaînement et le décodage des micro-instructions peut être rendu presque négligeable si les micro-instructions sont rangées dans une mémoire permanente à lecture rapide. En fait, bien que la micro-programmation ait été définie dès 1951, il a fallu attendre une dizaine d'années avant que les progrès technologiques réduisant le coût des mémoires mortes et leur donnant un temps d'accès suffisamment court (de 50 à 400 ns selon les technologies employées) en permettent une application de façon industrielle ; on voit même apparaître actuellement le stockage des micro-programmes dans des mémoires ré-inscriptibles.

2.5 Conclusion

En résumé, nous dirons que la micro-programmation est un moyen d'assurer un contrôle direct des circuits élémentaires d'un calculateur pour l'exécution de certaines tâches. Ce contrôle consiste en l'enchaînement et la synchronisation des actions élémentaires (micro-instructions) qui composent une tâche (micro-programme) et les signaux de commande (micro-opérations) sont organisés en mots stockés dans une mémoire de contrôle.

C H A P I T R E 1

CONCEPTION DE LA MEMOIRE DE COMMANDE

I.1. Introduction.

I.2. Codage des micro-opérations dans la micro-instruction.

I.3. Enchaînement des micro-instructions.

I.4. Optimisation de la mémoire de commande.

1 - INTRODUCTION

Le champ des applications de la micro-programmation est, nous l'avons vu, très étendu. Cependant, qu'elle soit utilisée dans une optique hardware ou software, le concept fondamental est l'utilisation d'une mémoire de contrôle et, en conséquence, le problème essentiel de la micro-programmation est de définir la mémoire de commande, sa structure et son fonctionnement.

Pour préciser les différentes notions à prendre en compte lors de cette définition, il n'est peut-être pas inutile de rappeler le modèle initial qu'en présenta Wilkes. La micro-mémoire se décompose (cf. figure 1) en deux parties C et S de rôles différents. Ces mémoires C et S sont à sélection linéaire et peuvent être représentées comme deux réseaux matriciels sur lesquels la sélection d'une ligne détermine un mot en mémoire, mot dans lequel un bit égal à un signifie qu'il y a couplage entre la ligne et la colonne et un zéro une absence de jonction. L'adressage de C et S est commun et s'effectue par le décodage du contenu du registre R. Lorsqu'une ligne du réseau C.S est sélectionnée, celle-ci est parcourue par un signal qui est aiguillé vers les colonnes couplées à cette ligne. Chacune des colonnes du réseau C est reliée à un circuit réalisant une micro-opération élémentaire et l'ensemble de ces colonnes définit une micro-instruction. Simultanément à l'ouverture des portes commandant l'exécution des micro-opérations, des signaux sont émis sur les fils de sortie du réseau S qui déterminent la prochaîne micro-instruction à exécuter. De plus, des conditions extérieures (bascule C) peuvent intervenir à ce niveau et modifier le choix de la micro-instruction successeur. Ces signaux sont transmis au registre R avec un certain retard déterminé par le temps d'accès des circuits électroniques mis en jeu au cours de l'exécution des micro-opérations. Au cycle d'horloge suivant, le contenu du registre R est décodé, une autre ligne du réseau est sélectionnée et la répétition du processus permet l'exécution d'une séquence de micro-instructions.

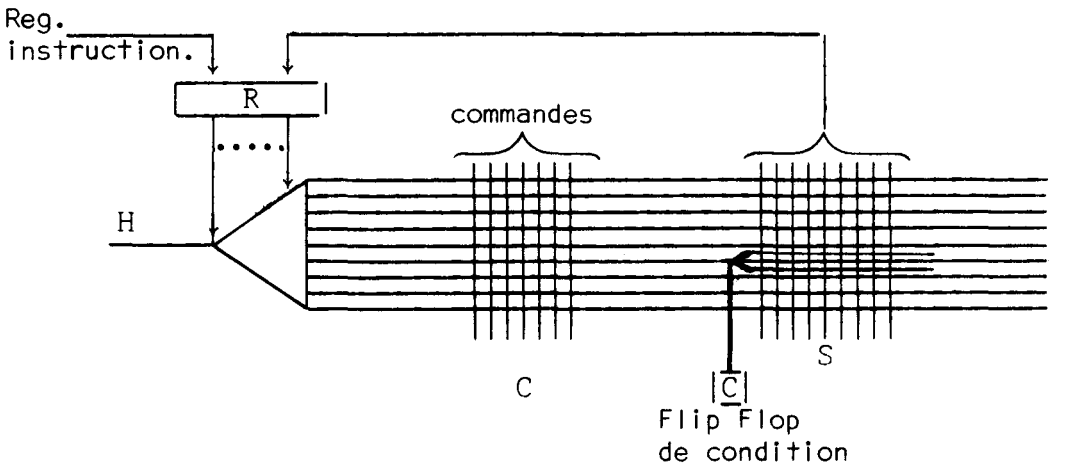


Figure 1

L'examen de ce modèle fait apparaître deux problèmes fondamentaux :

- D'une part, le problème de la longueur des mots de la micro-mémoire qui composent une micro-instruction et, en particulier, le choix du codage des micro-opérations élémentaires (matrice C).

- D'autre part, le problème de l'enchaînement des différentes micro-instructions (matrice S) et, en particulier, le choix et la réalisation des tests qui peuvent intervenir lors du déroulement du micro-programme.

2 - CODAGE DES MICRO-OPERATIONS DANS LA MICRO-INSTRUCTION

2.1 Principes

Dans le modèle original de Wilkes, à chaque micro-opération, correspond de façon biunivoque un bit du micro-mot. Dès que le nombre de micro-opérations du calculateur augmente, une telle définition de la micro-instruction devient impraticable et il faut recourir à un codage. Il est alors primordial d'effectuer un choix judicieux du degré de codage à utiliser dans les micro-instructions ainsi que de la partition des micro-opérations en classes. De nombreuses stratégies peuvent être mises en oeuvre et le critère d'optimalité est multiple. Les effets de ce choix apparaissent dans deux domaines,

taille de la mémoire de contrôle et vitesse d'exécution :

a) Le degré de codage utilisé est en relation étroite avec la longueur des mots de la mémoire de commande. De façon générale, un degré de codage élevé rétrécit la "largeur" de la mémoire de contrôle. Cependant, le gain ainsi réalisé est bien souvent perdu à cause du point suivant.

b) Le degré de codage a une influence directe sur le degré de parallélisme des micro-opérations. Dans le cas du modèle de Wilkes, si l'on dispose de n bits pour représenter n micro-opérations, il est possible d'exécuter simultanément jusqu'à n opérations élémentaires (dans l'hypothèse où l'exécution simultanée de n micro-commandes serait possible). Dans le cas opposé où les n bits de la micro-instruction servent à coder 2^n commandes élémentaires, la configuration, à un instant donné, de ces bits ne permet la sélection que d'une seule micro-opération parmi 2^n. Entre ces deux cas extrèmes, toutes les possibilités intermédiaires sont réalisables en effectuant des codages partiels de sous-groupes de micro-opérations. En général, la perte de simultanéité due à un degré de codage élevé se traduit par la réalisation en séquence de ces micro-opérations dans des micro-instructions successives ; ceci provoque non seulement une augmentation de la taille de la mémoire de contrôle due à l'allongement des micro-programmes, mais aussi un accroissement du temps d'exécution à cause des accès multiples à la micro-mémoire.

c) Enfin, le degré de codage influe également sur le temps d'exécution d'une micro-instruction. Boland a montré que, sur un calculateur conventionnel, 35% du temps d'un cycle d'horloge était utilisé pour le décodage et 65% du temps pour l'exécution des opérations élémentaires. Dans le cas d'un codage élevé, le temps de décodage n'est donc pas insignifiant, d'autant plus qu'il s'ajoute à une perte de micro-parallélisme.

Le degré de codage des micro-opérations dans les micro-instructions est donc un point important qui est surtout fonction du nombre de bits disponibles dans la micro-instruction. Il est d'usage en micro-programmation de distinguer micro-programmation verticale et micro-programmation horizontale et cette distinction est assez bien

caractérisée par la longueur des mots de la mémoire de commande, tant cette contrainte est astreignante. De façon générale, on parlera de micro-programmation à mots courts ou verticale pour des mémoires de contrôle contenant des mots dont la longueur varie de 16 à 32 bits et dont la plupart des combinaisons de bits ont un sens, et de micro-programmation horizontale pour des mots de longueur 40 à 120 bits selon les ressources du calculateur, avec un degré de simultanéité important. Pour cette raison, on nomme parfois micro-programmation parallèle ce type de micro-programmation et micro-programmation série, la micro-programmation verticale dont la faible largeur de champ interdit un degré de simultanéité élevé.

2.2 Micro-programmation horizontale

Dans ce type de micro-programmation, les mots de format long permettent d'utiliser une densité de codage assez faible.

A l'origine, dans le modèle de Wilkes, les signaux des micro-opérations ne sont pas codés. Ce cas limite, coûteux en largeur du mot de mémoire morte, n'a guère trouvé de réalisation que sur un ordinateur expérimental, l'ILLIAC IV, où la mémoire morte contrôlant un quadrant de 64 processeurs, est composé de mots de 280 bits. Pour cet ordinateur, le critère essentiel retenu a été celui de rapidité, d'où l'élimination de tout temps de décodage des micro-instructions.

En général, et essentiellement pour des raisons de coût, les micro-opérations qui composent chaque micro-instruction sont codées. Le découpage de la micro-instruction en champs de bits s'effectue en fonction du nombre total d'opérations élémentaires, de leur degré de simultanéité (ou d'incompatibilité), du débit de commandes recherché et de critères plus arbitraires comme l'orientation que l'on veut donner aux possibilités de micro-programmation (orientation micro-opérations ou orientation micro-instructions comme l'on parle de langages orientés respectivement machines ou problèmes). Le codage des champs peut être direct (chaque champ a une signification propre) ou indirect (un champ est interprété en fonction de la signification d'un autre champ) et l'optimisation des champs peut être effectuée en fonction de diverses stratégies que nous allons étudier sur des exemples.

2.2.1 l'Honeywell H4200

Sur cet ordinateur, l'utilisation de la micro-programmation apparaît essentiellement comme une forme de réalisation technologique de la mémoire de commande, étroitement intégrée dans la structure de la machine, et non comme des programmes de micro-instructions (firmware). Une micro-instruction comporte 120 bits : 34 sont utilisés pour l'enchaînement des micro-instructions et les 86 autres bits servent à coder plus de 300 actions élémentaires. Le nombre de micro-opérations est important dans une telle structure car un grand nombre de variations d'une même micro-fonction sont permises. Ces différentes variations peuvent apparaître dans la même séquence de micro-programme sans que des branchements soient nécessaires, de façon à accélérer la vitesse de l'ensemble. La technique utilisée est un composé de codage par champs directs et indirects. On repère les diverses utilisations des micro-opérations dans le jeu de micro-instructions du H4200 :
65 commandes élémentaires apparaissent potentiellement dans toutes les classes de micro-instructions. Ces micro-commandes de base sont codées dans un champ de 34 bits. Les autres micro-opérations sont organisées en 12 classes mutuellement exclusives de 50 micro-opérations. Un champ de 12 bits de la micro-instruction, chargé dans un registre (registre de groupe) permet de sélectionner un groupe de micro-opérations non de base et agit sur le décodage de ce champ (fig. 2). On utilise dans cette micro-instruction un codage direct pour les micro-commandes de base et un codage indirect pour les autres micro-opérations. Par ailleurs, dans chacun de ces champs, la densité de codage est faible. On parle ici de microprogrammation orientée hardware ou micro-opérations.

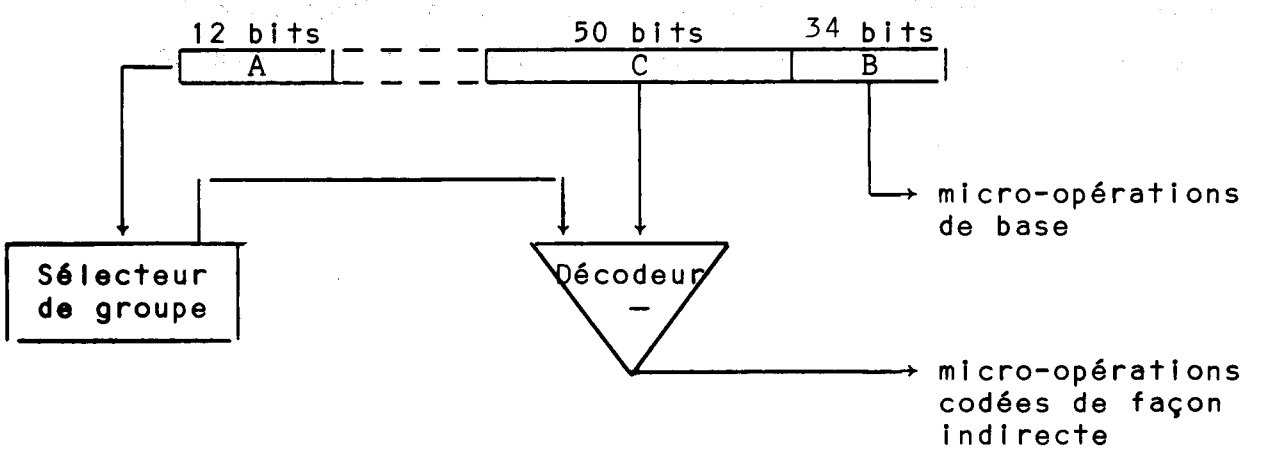


Figure 2

2.2.2 IBM System/360. modèles 40 à 85

De façon générale, les micro-instructions de l'IBM System/360 peuvent se décomposer en 3 parties :

- une première partie composée d'un certain nombre de champs a pour rôle d'assurer la détermination de la micro-instruction suivante en fonction de diverses conditions

- un champ particulier ("emit field") contient une constante ou des bits de significations particulières

- les autres champs de la micro-instruction servent à coder l'ensemble des micro-opérations.

La longueur des mots et le nombre de champs dans ces mots varient suivant le nombre de ressources du calculateur : le modèle 40 utilise des micro-mots de 56 bits codés en 18 champs, le modèle 50 des mots de 90 bits codés en 25 champs et le modèle 85 des micro-instructions de 108 bits codés en 27 champs. Examinons plus particulièrement de quelle façon les micro-opérations ont été partionnées en champs sur l'IBM/360-40. Onze champs, de 1 à 5 bits chacun, sont utilisés pour coder les différentes opérations agissant sur l'unité centrale et les canaux multiplexés. Les micro-opérations mutuellement exclusives sont regroupées pour former un seul champ (puisqu'une seule de ces opérations peut se produire à la fois) et chacun de ces champs est *codé de façon directe*. Les différentes opérations définies par les configurations de chacun des champs peuvent donc être simultanées. Le constructeur s'est attaché à ne pas restreindre les possibilités des micro-instructions, en ne faisant que les regroupements d'actions physiquement incompatibles. *On parlera dans ce cas d'optimisation de conjonction et de micro-programmation orientée micro-instructions.*

Codage des micro-opérations dans les micro-instructions IBM System/360 modèle 40 :

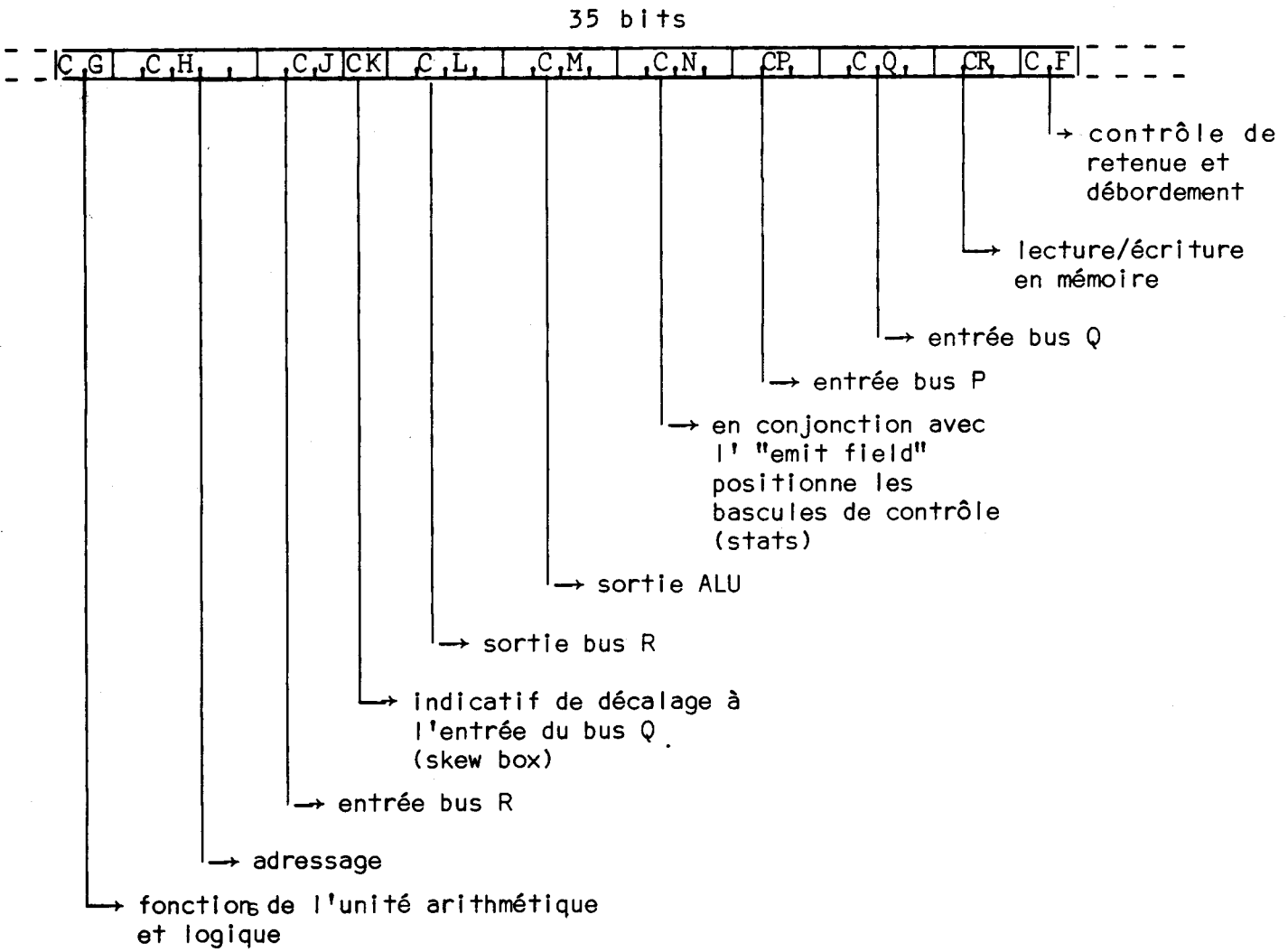


Figure 3

BUS LILLE

2.2.3 BULL M40

Les micro-instructions dans IBM-360 ont une structure qui rappelle celles du BULL M40 qui a été un des premiers ordinateurs microprogrammés commercialisés. Sur ce calculateur, les micro-mots ont une longueur de 50 bits : 11 bits (champ W) servant à définir l'adresse de la micro-instruction suivante, 28 bits décomposés en 7 champs de 4 bits servent à coder les micro-opérations et un dernier champ (champ N) contient 11 bits particuliers. Les micro-opérations sont donc regroupées en classes exclusives et 7 micro-opérations peuvent être exécutées de façon simultanée. Cependant à la différence des IBM System 360, le champ N (analogue à l'emit field) joue un rôle beaucoup plus important : il intervient dans l'interprétation de la micro-instruction dans plus du tiers des cas, est utile à la mémorisation d'un certain nombre de conditions et joue ainsi un rôle dans l'enchaînement des micro-instructions.

Par exemple les micro-instructions WYZ et TGY sont définies respectivement par :

$$\text{WYZ} : (N_{0-7} \cap Y_{0-7}) \cup (\bar{N}_{0-7} \cap W_{0-7}) \rightarrow Z_{0-7}$$

$$\text{TGY} : (G) \rightarrow Y_2$$

$$\begin{array}{ll} \text{Si } N_8 = 1 & Q_5 \rightarrow Y_1 \\ \text{Si } N_9 = 1 & Y_0 \rightarrow G \\ \text{Si } N_{10} = 1 & \bar{K}_5 \rightarrow K_5 \end{array}$$

2.2.4 Raytheon RAC-251

Cet ordinateur occupe une place particulière du fait de la technique utilisée pour réduire la taille de la mémoire morte. Le principe de cette méthode est le suivant : les micro-instructions sont groupées en 4 classes :

groupe 1 : accès aux instructions

groupe 2 : contrôle

groupe 3 : manipulation de données

groupe 4 : opérations arithmétiques et logiques.

On regroupe alors les micro-opérations de la façon suivante :

a) celles qui interviennent dans la réalisation des instructions des groupes 1 et 2 mais pas dans la réalisation des instructions des groupes 3 et 4 ;

b) celles qui sont utilisées pour l'exécution des micro-instructions des groupes 3 et 4 et non dans l'exécution des groupes 1 et 2;

c) enfin, les micro-opérations qui sont communes aux micro-instructions des différents groupes.

Ces divers regroupements apparaissent figure 4

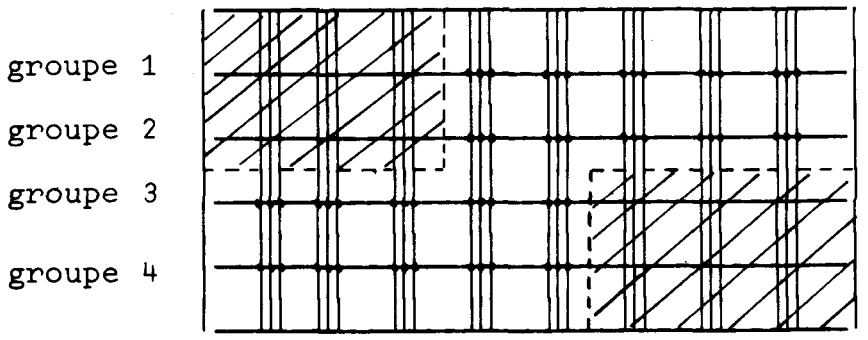


Figure 4

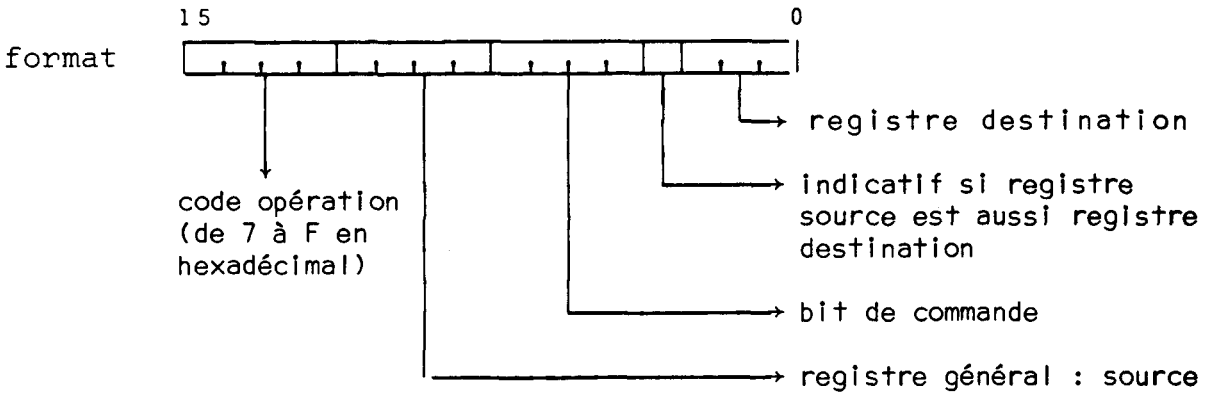
L'utilisation de cette technique permet d'implanter un nombre restreint de bits dans une mémoire morte de taille apparente beaucoup plus importante. En contre-partie, la disjonction systématique d'un certain nombre de micro-fonctions, basée uniquement sur l'utilisation des micro-instructions, peut restreindre les possibilités de la micro-programmation. En effet, les incompatibilités de fonctionnement ne sont pas basées, dans ce cas, sur une réelle impossibilité physique de fonctionnement simultané, mais sur une décision plus arbitraire : l'utilisateur n'aura pas besoin d'utiliser en simultanée des micro-fonctions des groupes a) et b). La contraction du micro-mot de sa taille apparente à sa taille réelle fait perdre également un certain nombre de simultanéités. Par ce fait, la micro-programmation occupe ici une place particulière, à la limite des micro-programmation horizontale et verticale. Il est d'ailleurs à

remarquer que cette conception de la micro-programmation a été utilisée sur le RAC-251 qui est un calculateur uni-bus, donc offrant peu de possibilités de travaux simultanés.

2.3 Micro-programmation verticale

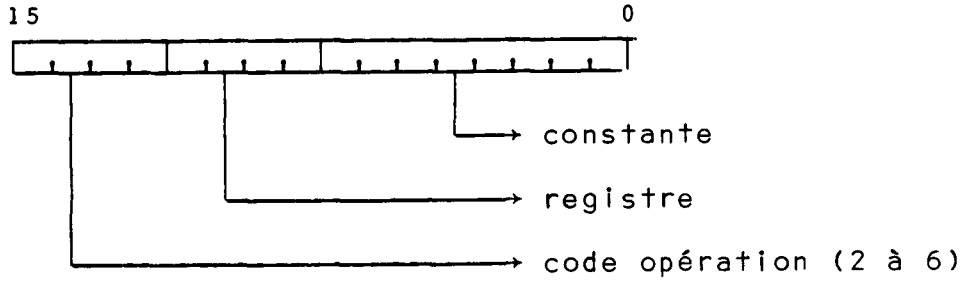
Ce type de micro-programmation utilise des micro-instructions de format court (16 à 32 bits en général) et apparaît pour le micro-programmeur sous une forme analogue à la programmation classique ; par analogie avec le software, on parle souvent à ce propos de firmware. La structure de la micro-instruction se rapproche du format d'une macro-instruction classique : un champ sert de code opération et sa valeur permet d'interprêter les bits suivants de diverses manières. Ceci permet une micro-programmation relativement simple, en particulier sur les mini-calculateurs. A titre d'exemple nous allons examiner les micro-instructions utilisées sur un de ces mini-calculateurs : le multi-20 fabriqué par Intertechnique. Les 72 micro-instructions sont des mots de 16 bits dont le premier champ sert de code opération ; selon la nature des formats, on peut les classer en 5 classes :

- classe 1 : micro-instructions opérationnelles



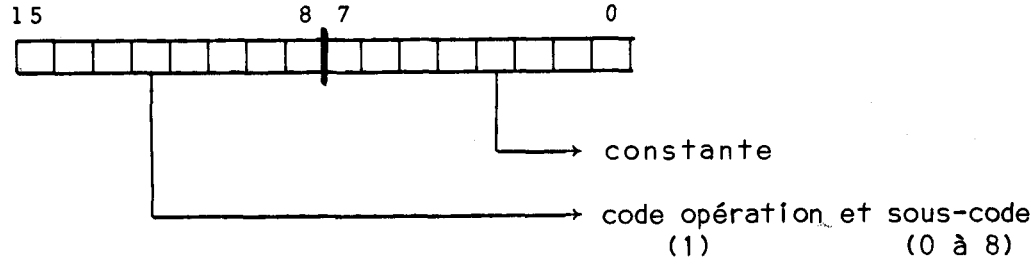
Ces micro-instructions réalisent les opérations arithmétiques et logiques, les transferts de registre à registre, les décalages, les lectures et écritures en mémoire et les entrées-sorties.

- classe 2 : micro-instructions à constante associée



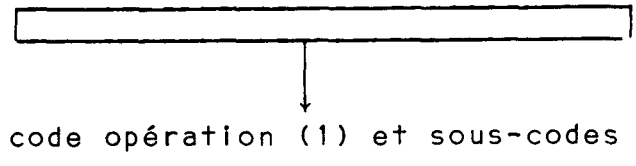
Ces micro-instructions effectuent les opérations associant registre et constante et un certain nombre de branchements, par comparaison avec la constante.

- classe 3 : micro-instructions sur registre fonctionnel

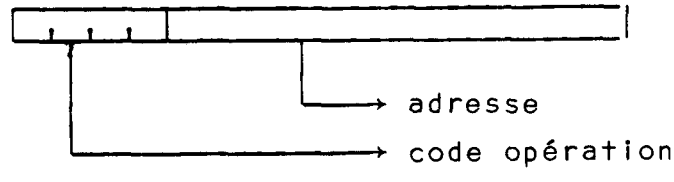


Ces micro-instructions effectuent des chargements de constantes dans les registres fonctionnels et les branchements ainsi que les diverses opérations suivantes : armement et désarmement d'interruption, départ et arrêt horloge etc...

- classe 4 : micro-instructions fonctionnelles :



- classe 5 : micro-instructions spéciales



Les micro-instructions de ces 2 dernières classes permettent d'effectuer un certain nombre d'opérations particulières telles que : NOP, sélection de jeu de registres, opérations sur pile, modification de micro-instruction, altération de micro-programme, retour de sous-programme avec ou sans sélection d'un jeu de registres, saut étendu (classe 5).

Des micro-instructions définies de telle façon sont donc extrêmement simples à manipuler (en comparaison de la micro-programmation horizontale tout au moins) et l'écriture des micro-programmes en est facilitée d'autant. Cependant une telle conception ne permet pas un important degré de simultanéité et son emploi reste réservé à des calculateurs de taille et performances modestes.

2.4 Conclusion

En conclusion, on peut dire que le degré de parallélisme est avant tout fonction des possibilités du calculateur, de sa taille et de sa puissance. Sur les mini-calculateurs ainsi que sur les machines unibus, qui possèdent peu de possibilités de travaux simultanés, une micro-programmation de type vertical suffit et a, de plus, l'avantage d'être aisément manipulable lorsque les mémoires de contrôle sont ré-inscriptibles, permettant ainsi d'orienter le processus vers des applications spécifiques. Par contre, pour des ordinateurs plus puissants, pour lesquels, d'une part, la rapidité d'exécution est souvent primordiale et qui nécessitent d'autre part un nombre important d'opérations à coder, on s'oriente naturellement vers des micro-mots de format plus long. Toutes les possibilités de codage sont alors permises : codage direct par champs, ou codage indirect et dans ce cas un peu plus lent, ou même à la limite, lorsque la rapidité est le but principal, absence totale de codage comme sur ILLIAC IV.

3 - ENCHAINEMENT DES MICRO-INSTRUCTIONS

3.1 Introduction

Le second point à préciser, en ce qui concerne la définition des mémoires de contrôle, est la manière dont sera effectué, à l'exécution, l'enchaînement des différentes micro-instructions d'un

micro-programme. En fonction des contraintes hardware (taille de la mémoire de contrôle, nombre de conditions, contraintes de temps) et du but que l'on cherche à atteindre (micro-programmation orientée hardware ou orientée micro-instructions, machines à émuler) des tactiques et des astuces de réalisation nombreuses peuvent être utilisées.

3.2 Enchaînements dans les différents types de micro-programmation

3.2.1 Principe de modification d'adresse

En général, les micro-instructions dans la mémoire de contrôle sont adressées par le décodage du contenu du registre adresse de la mémoire de contrôle. Parfois, l'adressage de la mémoire peut se faire à l'aide d'un registre à décalage, permettant l'accès à un nombre multiple de mots dans la micro-mémoire. C'est la méthode qui a été retenue pour réaliser le bloc de contrôle de chacun des quatre quadrants de 64 processeurs du calculateur ILLIAC IV.

Dans le cas où les micro-instructions sont repérées par le contenu du registre adresse mémoire, deux options peuvent, à priori, être choisies :

- ou les micro-instructions sont mémorisées en séquence,
- ou les micro-instructions, exécutées de façon séquentielle, ne sont pas implantées à la suite les unes des autres.

Dans le premier cas, la méthode la plus simple et la plus couramment utilisée est d'adjoindre au registre adresse de la micro-mémoire un incrémenteur qui, à chaque cycle d'horloge, augmente d'une unité le contenu de ce registre.

Dans le second cas, et également lorsqu'il faut se brancher à une adresse déterminée à l'issue d'un test d'une condition, il est nécessaire de spécifier une nouvelle adresse. Cette adresse peut être définie de diverses façons :

a) Tout d'abord, la nouvelle adresse peut être un champ particulier de la micro-instruction en cours d'exécution, cette

nouvelle adresse pouvant apparaître soit entièrement, soit partiellement. Par exemple, sur le MITRA 15, les micro-instructions ne sont pas rangées en séquence et chacune d'elle contient un champ de 6 bits déterminant un déplacement dans la micro-mémoire, le registre adresse mémoire étant un registre de 10 bits. C'est en fait la réalisation du modèle proposé par Wilkes où chaque micro-instruction contient l'adresse de l'instruction suivante à exécuter (matrice S).

b) Le chargement du registre adresse de la mémoire de commande peut également être effectué à partir d'un registre spécifié par un champ de la micro-instruction. Il est à noter, par ailleurs, qu'au début du micro-programme, l'adresse de la micro-instruction initiale est, en général, obtenue par chargement du registre adresse mémoire, à partir du registre instruction contenant le code opération de la macro-instruction et au travers d'un circuit générant cette adresse.

c) La nouvelle adresse de la micro-instruction à exécuter peut également être définie par une opération logique entre l'adresse courante et le contenu d'un registre, l'opération ou le registre étant spécifié par la micro-instruction courante.

d) Enfin, de façon générale, tout un réseau modificateur ou générateur d'adresse peut apparaître devant le registre adresse de la mémoire de commande, pouvant tenir compte de l'adresse courante, du type de la micro-instruction en cours, du contenu de registres et de bascules de condition.

3.2.2. Enchaînements selon conditions

Lors du déroulement d'un micro-programme, diverses actions peuvent souvent être choisies en fonction d'un certain nombre de conditions. Il est donc à chaque fois nécessaire de spécifier la ou les conditions à tester et, à cette occasion, se pose un problème de découpage et de définition proche de celui du codage des actions élémentaires composant une micro-instruction. En particulier, la longueur du mot micro-instruction devient alors déterminante.

Un autre sujet de préoccupation est de préciser quel est le moment d'intervention dans le temps de (ou des) condition(s). Trois niveaux d'intervention peuvent apparaître :

a) La condition intervient au moment de la détermination de l'adresse de la nouvelle micro-instruction à exécuter ; c'est le cas le plus courant. Cependant, comme il sera vu plus loin, dans le cas d'overlap des micro-instructions, le fait que la condition dépende ou non de l'exécution de la micro-instruction en cours a son importance.

b) Il peut également être possible d'altérer dans certains cas le mot micro-instruction qui vient d'être lu en mémoire de contrôle, avant son décodage et l'envoi des signaux de commande. Par exemple, sur le Multi-20 Intertechnique, il est possible d'effectuer une union logique des poids forts de la micro-instruction (code opération de la micro-instruction) avec le contenu d'un registre déterminé.

c) Enfin dans le cas d'une micro-programmation totalement orientée hardware, des variations dans l'exécution d'une micro-instruction peuvent apparaître par câblage en fonction de conditions automatiquement générées par le système.

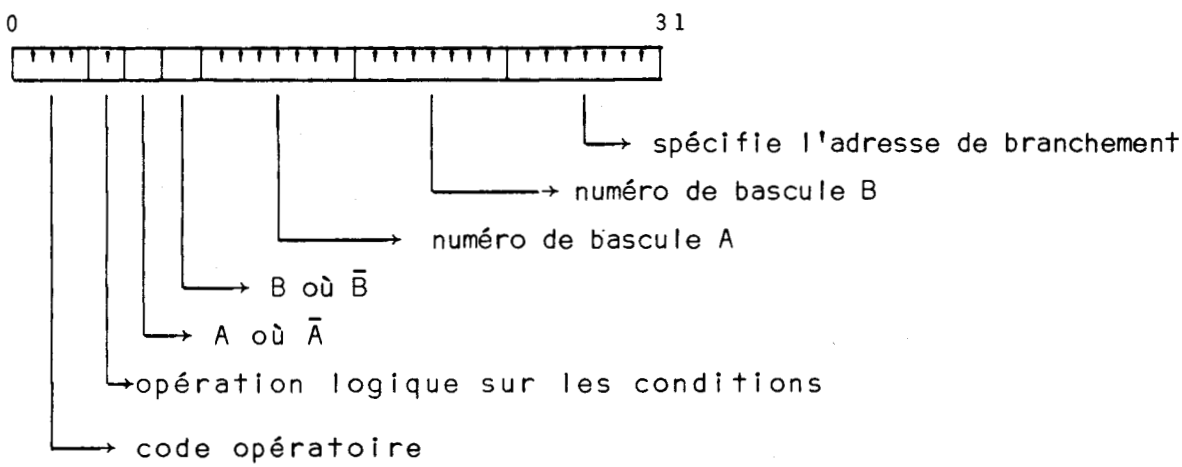
Le problème essentiel reste cependant : comment spécifier le ou les conditions à tester et de quelle façon le résultat du test va-t-il modifier le contenu du registre adresse de la mémoire de contrôle ? Nous allons pour cela examiner les divers types de micro-programmation rencontrés, en tenant compte, en particulier, de la longueur des micro-mots.

3.2.3 Micro-programmation verticale

ⓐ Dans le cas de la **micro-programmation verticale**, les micro-instructions comportent peu de bits. En général, le mot de la micro-instruction est trop court pour pouvoir définir à la fois les opérations élémentaires et l'adresse d'une micro-instruction successeur. Dans ce cas, le nombre de micro-instructions est accru et chacune d'elles est spécialisée : ou bien les micro-instructions sont des instructions fonctionnelles exécutant véritablement une

tâche et elles sont enchaînées de façon séquentielle par incrémentation du registre adresse de la mémoire de contrôle, ou bien les micro-ordres sont des instructions de contrôle, spécifiant un branchement ; le test est alors défini par le code opération (+sous-code si nécessaire) et le reste des bits de la micro-instruction est considérée comme définissant une adresse de branchement en micro-mémoire dans le cas d'une réponse positive. Dans le cas contraire, il y a incrémentations du compteur ordinal. C'est en particulier le fonctionnement adopté sur Multi-20 qui utilise des micro-instructions de 16 bits. Il s'agit donc à chaque fois d'un codage indirect, le code opératoire de la micro-instruction définissant l'interprétation du reste de la micro-instruction. La micro-programmation est, dans ce cas, proche de la programmation classique et est particulièrement aisée.

(b) En conservant toujours la partition des micro-ordres en micro-opérations fonctionnelles et micro-instructions de contrôle mais en allongeant le format des mots de la mémoire morte, il est possible d'apporter plus de puissance aux instructions de branchement. Ainsi le MLP 900 utilise des micro-mots de 32 bits ; les micro-instructions sont réparties en micro-instructions fonctionnelles et de contrôle en fonction du premier bit du code opératoire ; de plus les différents types de micro-instructions sont traités par des organes différents (cf.§3.3.3). Le MLP 900 dispose d'une batterie de 256 bascules, testables lors des branchements, qui sont positionnées soit par micro-programme, soit par le système lui-même. Lors de chaque micro-instruction de branchement, il est possible de tester deux bascules A et B ; plus précisément le résultat d'une opération logique (ET, OU, OU exclusif) sur deux des quatre variables A, $\bar{A}$, $\bar{B}$ et $\bar{B}$ est testé pour préciser l'adresse de branchement. Une micro-instruction de branchement est du type ci-dessous :



3.2.4 Micro-programmation horizontale

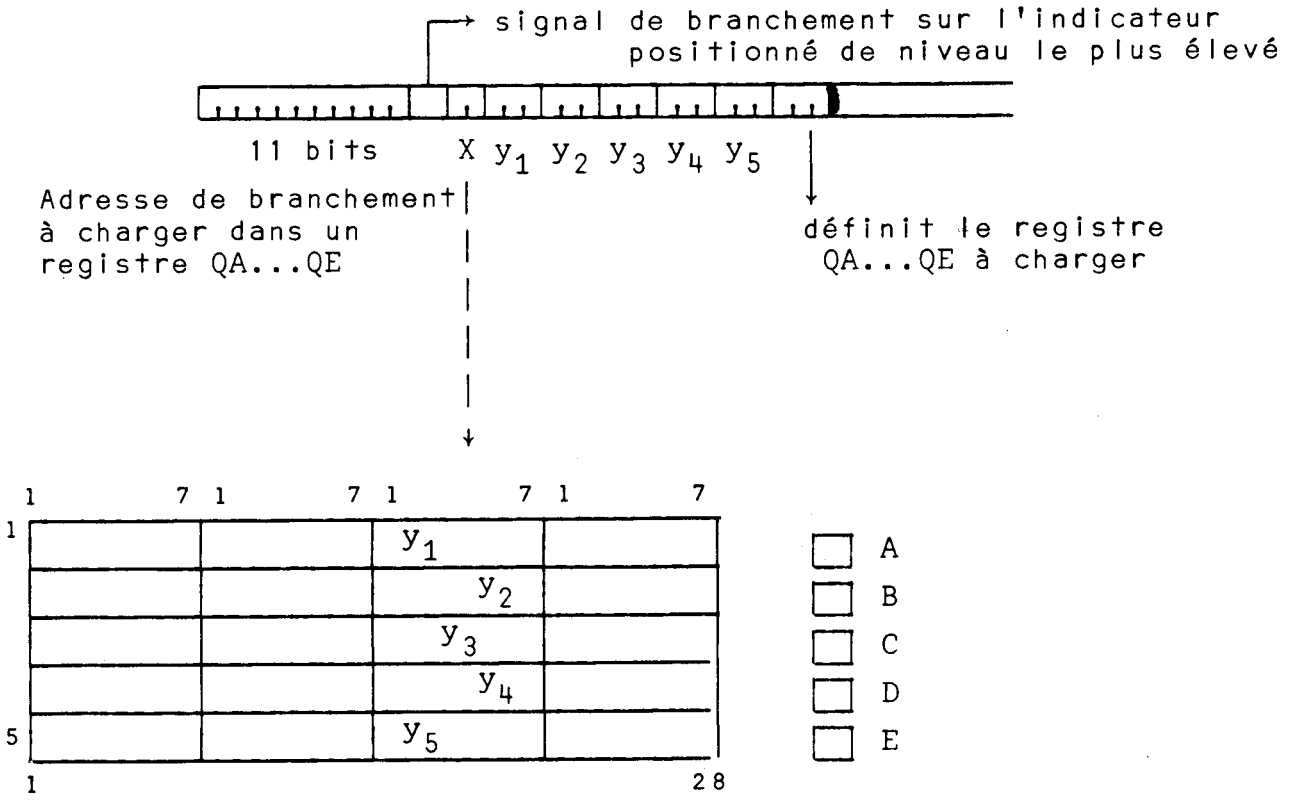
Dans le cas de la micro-programmation horizontale, le format plus long des micro-mots permet à la fois le codage des opérations élémentaires, le codage des conditions à tester et la spécification de la micro-adresse suivante. Sur l'IBM 360-40, les micro-mots sont de 56 bits et, nous l'avons vu, 34 d'entr'eux servent à coder les micro-opérations ; 16 bits serviront à définir les branchements (2 bits servent au contrôle de parité et 4 bits servent dans l'emit field à définir des constantes particulières). Le registre adresse de la micro-mémoire est de 12 bits. Il peut être chargé à partir de registres (undumped format) ou bien tous ses bits ou une partie peuvent être modifiés à l'aide des différents champs de la micro-instruction. Deux champs de 4 bits CB et CC servent à définir deux tests dont le résultat positionne deux bits. Le champt CD (2 bits) peut être, ou les poids forts du registre adresse de la micro-mémoire, ou servir à interprêter les autres champs. En particulier sa valeur précise quelle partie du registre adresse de la mémoire de contrôle sera remplacée par le contenu du champ CA de 4 bits de la micro-instruction. 16 bascules groupées par ensemble de 4 peuvent être testées par micro-programme. D'autres conditions sont testées automatiquement par les circuits selon la fonction exécutée.

3.2.5 Micro-programmation orientée hardware

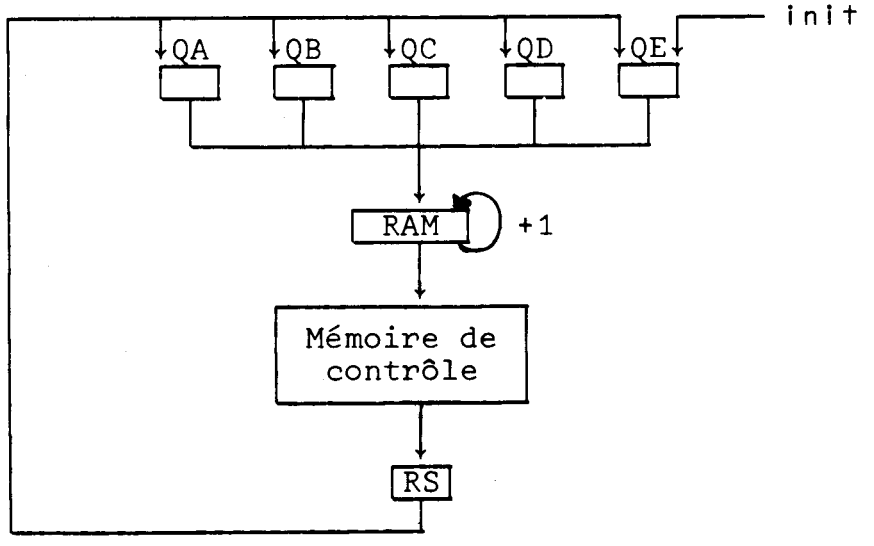
Dans la micro-programmation orientée micro-opérations, le problème se complique. Dans ce cas, en effet, la micro-programmation est partie intégrante du hardware et n'en est qu'une représentation ; aussi le nombre d'indicateurs à tester est-il élevé. Dans l'Honeywell 4200, 140 indicateurs peuvent ainsi être testés. Avec un codage maximal, ces 140 indicateurs pourraient être codés à l'aide de 8 bits mais, dans ce cas, il est impossible de réaliser plusieurs tests simultanément. La méthode adoptée sur le H4200 est la suivante : les 140 indicateurs sont partitionnés en 5 groupes de 28 bascules, de façon à rendre possible 5 tests au plus en simultanée. Chacun de ces groupes est divisé en 4 sous-groupes de 7 bits. Les sous-groupes sont sélectionnés à l'aide des 2 bits du champ X de la micro-instruction (pour tous les groupes en même temps) et pour chacun des groupes un champ y_i de 3

bits sélectionne un des 7 indicateurs dans le sous-groupe. Ceci permet de tester jusqu'à $4 \times 7^5 = 67\ 228$ possibilités. Les résultats des tests sont mémorisés dans des bascules A,B,C,D et E.

Les micro-instructions du H4200 comportent 120 bits dont 32 sont utilisés pour la détermination de la micro-instruction suivante



Une priorité fixe est attachée à chacune des bascules A...E. A chaque bascule est associée un registre QA...QE servant à charger le registre adresse de la micro-mémoire. Dans le cas d'exécution en séquence, le registre d'adressage est augmenté par incrémentation.



3.3 Procédés d'accélération

3.3.1 Introduction

Dans la partie 3.2, nous avons examiné les différentes possibilités de définition de l'adresse de la micro-instruction suivante à exécuter lors du déroulement d'un micro-programme. On peut concevoir que la définition de cette nouvelle adresse n'intervient qu'à la fin de la micro-instruction en cours. On parle alors de micro-programmation série ; aucun problème n'apparaît en ce qui concerne les branchements conditionnels mais le temps total d'exécution d'une micro-instruction est la somme du temps d'accès et du temps d'exécution véritable. On peut fort bien concevoir que se recouvre dans le temps les phases définition de l'adresse suivante et exécution. On parle alors de micro-programmation avec anticipation (ou overlap).

3.3.2 Dédoublement de la mémoire de contrôle

Une première solution a été proposée par Wilkes lui-même et a été adoptée pour la réalisation des circuits de commande du RCA spectra 70/45. Elle consiste en l'utilisation de deux mémoires de contrôle travaillant en bascule et les adresses des micro-instructions sont entrelacées entre les deux mémoires. L'accès à une micro-mémoire est effectué pendant qu'est exécutée une micro-instruction de l'autre micro-mémoire, ce qui permet un gain de temps de moitié. Cependant dans le cas d'un branchement sous condition, l'adresse de la même mémoire peut être incrémentée, sans utilisation du basculement, provoquant une pénalité de temps due au calcul de l'adresse et à l'accès à la fin de la micro-instruction en cours.

3.3.3 Anticipation avec une seule micro-mémoire

La technique de recouvrement des temps d'exécution est également utilisée pour un calculateur ne disposant que d'une seule mémoire de contrôle physique.

Dans le cas d'enchaînement séquentiel aucun problème n'apparait: pendant l'exécution, la nouvelle adresse peut être générée et est

toujours correcte.

Dans le cas d'enchaînement en fonction de conditions, plusieurs cas peuvent se produire :

- Lorsque la (ou les) condition(s) à tester ne dépend(ent) pas du résultat de la micro-instruction en cours d'exécution, la nouvelle adresse peut être générée compte tenu du résultat du test et est toujours exacte.

- Si la ou les conditions à tester sont, par contre, fonction du résultat de la micro-instruction en cours d'exécution, la nouvelle adresse générée peut être valable ou non selon qu'une hypothèse correcte ou incorrecte a été faite sur le résultat du test. Dans le cas d'une mauvaise hypothèse, il est alors nécessaire de générer une nouvelle adresse ce qui provoque une pénalité par allongement du temps total d'exécution. Dans une telle conception, le "doigté" du micro-programmeur intervient dans la réduction ou l'allongement du temps de déroulement d'un micro-programme.

3.3.4 Différentiation des fonctions exécution et contrôle des micro-instructions.

Dans le cas d'une micro-programmation verticale, nous avons vu qu'il était possible de partitionner l'ensemble des micro-instructions en deux classes : les micro-instructions fonctionnelles et les micro-instructions de contrôle. Dans cette optique, on peut alors imaginer de traiter ces 2 types de micro-instructions séparément dans des organes différents et par conséquent simultanément.

Ainsi sur le M.L.P.900, à chaque cycle, deux micro-instructions sont lues simultanément dans la mémoire de micro-programmes. Si l'une des instructions est un micro-ordre fonctionnel et si l'autre est un micro-ordre de contrôle chacun d'eux est traité simultanément avec l'autre, dans deux organes séparés : l'unité fonctionnelle et l'unité de contrôle. Sinon seule la première micro-instruction fonctionnelle est exécutée pendant que le registre adresse mémoire est incrémenté.

A cette occasion trois remarques peuvent être faites :

- D'une part accéder à deux instructions dont l'une spécifie l'action à exécuter et l'autre l'enchaînement revient, en fait, à utiliser les deux parties C et S du réseau matriciel proposé par Wilkes.

- D'autre part, une telle simultanéité suppose que la définition de l'adresse n'est pas fonction de la micro-instruction en cours d'exécution dans l'unité fonctionnelle.

- Enfin, la séparation physique en deux unités fonctionnelles et de contrôle n'est avantageuse que si l'unité de contrôle est à même d'exécuter des tâches multiples ; c'est d'ailleurs ce qui est réalisé sur la M.L.P.900.

Poussant plus loin dans cette voie, il est tentant de scinder la mémoire en 2 mémoires indépendantes mais interactives.
Le problème est alors de gérer la synchronisation entre ces 2 mémoires et d'autre part de partager les ressources communes à chacun de ces contrôles. Ceci suppose l'existence de circuits et de verrouillages câblés (car rapides) équivalents aux opérations P et V sur les sémaphores de Dijkstra. A l'extrême, il est imaginable d'éclater les différents contrôles en les attachant à chacune des ressources.

3.3.5 Accès multiples et union de micro-instructions

Dans le paragraphe précédent, seules des micro-instructions de types différents (fonctionnelle ou de contrôle) peuvent être exécutées simultanément. Sur le calculateur ILLIAC IV, le parallélisme peut apparaître de façon plus générale. La mémoire morte qui contrôle un quadrant de 64 processeurs d'ILLIAC IV est composée de mots de 280 bits et les signaux de contrôles ne sont pas codés. Jusqu'à dix mots de la micro-mémoire (2 instructions de 5 mots au plus) sont simultanément accessibles par adressage à l'aide de registres à décalage et la réunion logique de ces différents mots est effectuée. Pratiquement, cela se traduit par le fait que les signaux de contrôle correspondant à des micro-mots différents sont

envoyés simultanément, permettant l'exécution parallèle d'un certain nombre d'opérations.

L'utilisation d'une telle technique suppose cependant que l'on dispose d'un degré de simultanéité élevé et que le but principal soit la vitesse d'exécution justifiant seule l'utilisation de micro-mots aussi longs.

3.3.6 Apports de la programmation à la micro-programmation

Divers procédés ont été apportés à la micro-programmation en utilisant à ce niveau des techniques mises en oeuvre dans la programmation classique. En particulier la notion de sous-programme peut être utilisée en micro-programmation pour réduire la taille de la mémoire de contrôle ; il est alors nécessaire de disposer d'une pile d'adresses de retour . Par exemple, sur le Multi-20 (Intertechnique), une pile de 16 registres, contrôlée par un pointeur de pile géré par micro-programme, est disponible pour la mémorisation des adresses de retour à l'issue de l'exécution d'un sous-micro-programme.

La notion d'interruption peut également être utilisée en micro-programmation : le Mitra 15 (CII) dispose de 32 niveaux de micro-interruptions ("suspension") hiérarchisés pour le couplage par micro-programme avec certains périphériques. Une pile de profondeur 4 permet de mémoriser les adresses de retour.

Enfin le MLP 900, qui dispose d'une unité distincte pour traiter les micro-instructions de contrôle, possède, de plus, des avantages pour l'utilisation d'itérations. L'unité de contrôle comprend en particulier 2 groupes de 16 registres. Un des groupes est utilisé comme pile mémorisant les adresses de retour des sous-micro-programmes. Les autres registres servent à mémoriser des compteurs (compteur de bytes pour opérations arithmétiques, transferts de blocs), à gérer la pile et l'adressage. Des micro-instructions particulières permettent, sous conditions, le branchement à un sous-programme avec mémorisation d'adresse, ou le retour, ainsi que le branchement conditionnel avec décrémentation de compteur pour le contrôle des boucles.

Il faut cependant noter que l'utilisation de telles techniques ne sont performantes que dans le cas de micro-programmation se rapprochant de la programmation classique, c'est-à-dire de micro-programmation verticale.

4 - OPTIMISATION DE LA MEMOIRE DE COMMANDE

4.1 Micro-programme optimisé

Compte tenu des diverses techniques qui peuvent être mises en oeuvre pour définir et enchaîner les micro-instructions, de quelle façon un micro-programme peut-il être optimisé ?

Deux critères peuvent être pris en compte : D'une part, on peut chercher à maximiser la rapidité d'exécution du micro-programme; d'autre part, on peut rechercher une occupation en mémoire qui soit minimale.

Plus précisément, la vitesse d'exécution peut être accrue à deux niveaux différents : en tentant d'obtenir un degré maximum de simultanéité des micro-opérations qui composent chaque micro-instruction, on diminue le nombre de micro-instructions nécessaires et, en conséquence, le nombre d'accès à la micro-mémoire ; d'autre part, l'étude précédente a montré l'extrême diversité des solutions et astuces utilisées pour accélérer l'enchaînement des micro-instructions.

Le nombre de bits nécessaires à la mémorisation du micro-programme en mémoire de commande peut également être minimisé de diverses manières : il est possible de minimiser la longueur des micro-mots de la mémoire de commande (codage) et de minimiser la longueur du micro-programme en recherchant une simultanéité maximale dans l'exécution des micro-opérations.

Ces différents critères peuvent, par ailleurs, s'opposer et la longueur du micro-mot est, en particulier, un facteur très contraignant. Enfin, une optimisation de la mémoire de commande, c'est-à-dire de l'ensemble des micro-programmes, peut être effectuée,

au niveau de l'analyse, en tentant de maximiser le nombre de parties communes aux divers micro-programmes (sous-micro-programmes).

4.2 Conception de la mémoire de commande

Une fois effectué le choix de principe initial (type de micro-programmation, buts recherchés) en fonction de la taille du système et de sa destination, le problème principal est de se définir les micro-instructions pour la structure machine donnée. Cette définition porte sur trois points essentiels :

- longueur du micro-mot,
- codage des micro-opérations,
- enchaînement.

Pour l'enchaînement des micro-instructions, toute l'ingéniosité du concepteur hardware est requise pour accélérer la détermination et l'accès aux micro-instructions.

Pour définir le codage des micro-opérations, on cherchera le plus souvent à réaliser des micro-instructions assez générales, à éviter autant que possible la réunion dans une même micro-instruction, de micro-opérations qui n'ont que peu de chance d'être souvent réalisées simultanément. Pour effectuer le codage, on tiendra compte des incompatibilités d'exécution simultanée des micro-opérations, de façon à définir des classes de micro-opérations, des ordres de transferts qui sont habituellement utilisés dans un ordinateur et des micro-programmes que l'on cherchera à réaliser, surtout si ces micro-programmes sont de caractères particuliers.

Pour l'enchaînement des micro-instructions, et plus encore, pour le codage des micro-opérations, il faudra tenir compte, à chaque instant, de la longueur du micro-mot que l'on s'autorise.

Une fois définies les micro-instructions, il est nécessaire de rédiger les micro-programmes. Pour faciliter cette écriture, est en général mis au point un micro-assembleur offrant les possibilités d'adressage symbolique , d'utilisation de pseudo-instructions et de

codes mnémoniques pour les micro-instructions. La mise au point peut ensuite être effectuée :

- soit par simulation (simulateur général, suffisamment flexible pour s'adapter aux spécifications particulières de chaque micro-processeur mis au point à la compagnie Honeywell Bull)

- soit de façon intéractive sur la micro-machine elle-même (à l'aide d'un Mitra 15 à l'IRIA, sur IBM 370 aux USA...) ou sur une machine émulant le micro-processeur (à l'étude).

Une telle méthode n'offre guère de grandes possibilités d'optimisation ; celle-ci se réduisant à un codage à priori des micro-opérations et il faut ensuite écrire le moins mal possible les micro-programmes à l'aide des micro-instructions préalablement définies.

4.3 Autre méthode

Une autre stratégie peut être choisie. Elle consiste à définir les micro-instructions en même temps que les micro-programmes et compte tenu des micro-programmes. Si les micro-programmes sont particuliers (par exemple, contrôle en temps réel d'un processus), on aura ainsi particularisé les micro-instructions. Si, par contre, on se propose de n'utiliser que des micro-programmes classiques, la définition des micro-instructions, conjointement à celle des micro-programmes de base formant le noyau du calculateur (transferts, lecture-écriture en mémoire, opérations arithmétiques et logiques...), amènera la définition de micro-instructions assez générales.

Ceci suppose, cependant, que les micro-programmes soient décrits en termes d'opérations élémentaires ; ceci peut être rendu possible par l'utilisation de langages de description adéquats soit dérivé d'Algol 68 (utilisation des définitions "collatéral" et "parallèle") soit proche des langages de description d'automatismes séquentiels. Un micro-programme se présente alors comme un ensemble d'opérations élémentaires munies de relations de précédence,

susceptible de traitement automatique. A notre connaissance, un seul projet (développé à la Compagnie Thomson - CSF) met en jeu une stratégie analogue. Le but de ce projet est de permettre une description algorithmique des micro-programmes portant sur des noms de variables, puis d'associer la description de la micro-machine pour générer des micro-programmes (sous forme logique) en partie optimisée.

4.4 Optimisation

Dans l'optique du §4.3, optimiser un micro-programme, c'est se définir les micro-instructions fournissant le micro-programme le plus rapide et le moins encombrant ou plutôt réalisant un compromis entre vitesse et occupation en mémoire, entre performance et coût.

La définition de l'enchaînement de ces micro-instructions, la détermination des adresses physiques des micro-programmes n'est, pour l'instant, guère soluble dans le cas le plus général. Des types divers d'enchaînement des micro-instructions peuvent être utilisés (selon le nombre et le niveau d'intervention des conditions) et les nombreuses astuces d'accélération mises en oeuvre par les concepteurs du câblé (accès multiples, overlap, différentiation des fonctions exécution et contrôle) rendent difficile toute formalisation.

Par contre, la définition des micro-instructions en tant qu'ensemble de micro-opérations est plus abordable et il est classique de représenter un micro-programme comme un ensemble de micro-opérations reliées entre elles par des relations variées.

La suite de ce travail consiste en la formalisation de ces notions et la recherche d'algorithmes permettant une optimisation automatique de la micro-mémoire ; on s'intéressera, en particulier, à la recherche de simultanéité, permettant à la fois l'accélération du micro-programme et la réduction de son occupation en mémoire.

En fait, il est apparu que les notions utilisées dans la formalisation d'un bloc de commande micro-programmé étaient plus générales, une telle formalisation n'étant qu'une application d'un cas particulier de la notion de flot booléen contrôlé sur un graphe. Dans le chapitre suivant, nous allons tout d'abord définir ce qu'est un flot booléen contrôlé et en étudier quelques propriétés.

C H A P I T R E 2

FLOT BOOLEEN GENERALISE SUR UN GRAPHE

II.1. Flot booléen contrôlé

II.2. Recouvrements

II.3. Flot booléen généralisé

II.4. Problèmes de fbg homomorphe d'un flot booléen

Dans ce chapitre, après la définition d'un flot booléen contrôlé sur un graphe, nous nous intéressons aux propriétés des flots booléens généralisés entre sous-graphes d'un graphe.

En particulier, nous recherchons une caractérisation des flots booléens généralisés qui respectent les propriétés du flot booléen contrôlé simple du graphe à partir duquel il a été construit (semi-homomorphie, homomorphie) de façon à assurer la validité des regroupements que l'on peut effectuer sur ce graphe.

1 - FLOT BOOLEEN CONTROLE SUR UN GRAPHE

1.1 Définitions

1.1.1 Introduction

La notion de flot sur un graphe permet de guider l'étude ou l'exécution d'une circulation de matière (quantifiable) au travers des arcs du graphe.

Etant donné un graphe $G = (X,U)$

- X ensemble des sommets
- $U \subset X \times X$ ensemble des arcs,

on appelle flot défini sur le graphe G, la donnée d'une fonction ϕ, application de U dans $\mathbb{R}$ qui vérifie la condition de conservation :

$$\forall\, x \in X \qquad \sum_{\substack{u=(y,x)\\ u\in U}} \phi(u) = \sum_{\substack{u=(x,y)\\ u\in U}} \phi(u)$$

c'est-à-dire que pour tout sommet du graphe, la somme des quantités entrantes est égale à la somme des quantités sortantes.

En procédant d'une démarche analogue, on considère une fonction ϕ qui associe à chaque arc du graphe une fonction booléenne et on s'intèresse non plus à une circulation de matière, mais à l'existence d'une circulation : le passage dans un arc donné sera permis ou non selon que la valeur de la fonction booléenne associée est vraie ou fausse.

1.1.2 Définition

Etant donnés :

- un graphe $G = (X,U)$
- un ensemble E de n variables booléennes
- un ensemble $\mathcal{E}$ de fonctions booléennes dont les variables appartiennent à E et à valeurs dans $\{0,1\}$.

On appelle flot booléen contrôlé défini sur ce graphe, une application ϕ de U dans $\mathcal{E}$ telle que :

$$\forall\, x \in X \qquad \sum_{(y,x)\in U} \phi(y,x) = \sum_{(x,y)\in U} \phi(x,y)$$

(où $\sum$ représente la somme logique).

1.1.3 Justification : Le terme de flot est-il acceptable pour une telle fonction ϕ ?

L'ensemble $\mathcal{E}$ est muni de deux opérations :

- une opération "addition" associative, commutative, qui possède un élément neutre
- une opération "multiplication" qui est associative et distributive par rapport à l'addition.

$\mathcal{E}$ possède donc les mêmes caractéristiques que l'ensemble N des entiers naturels sur lequel on définit des flots entiers.

De plus, C. BERGE a défini un flot sur un graphe comme une application de l'ensemble des arcs dans un anneau et dans tout anneau, l'addition définit un groupe abélien et possède donc les propriétés de $\mathcal{E}$, plus l'existence d'un inverse.

Aussi, le terme de flot est-il justifié pour une telle fonction .

D'autre part, notons $\bar{E} = \{0,1\}^n$. Les valeurs des fonctions ϕ dépendent des points de $\bar{E}$ envisagés et les arcs sélectionnés sont donc fonctions de ces points. En conséquence le passage entre deux sommets du graphe est régi par des conditions externes au graphe, qui sont exprimées à l'aide de variables booléennes, d'où l'appellation de flot contrôlé.

1.1.4 Extension

Dans la définition précédente, la condition de conservation précise que si la valeur de la fonction booléenne associée à un arc entrant en un sommet quelconque est vraie en un point de $\bar{E}$, alors il existe nécessairement une fonction booléenne vraie au même point de $\bar{E}$ et permettant de quitter ce sommet. Le rôle de paramètre des éléments de $\bar{E}$ peut être étendu, en permettant que la fonction booléenne de sortie soit vraie en un point différent.

Définition 2 : On appelle flot booléen contrôlé défini sur un graphe $G = (X,U)$, une application ϕ de U dans l'ensemble $\mathcal{E}$ précédent telle que:

$$\forall\, x \in X,\ \forall\, e_1 \in \bar{E}\ \ \exists\, e_2 \in \bar{E} \text{ tel que} \left(\sum_{(y,x)\in U} \phi(y,x)(e_1) = 1 \Longleftrightarrow \sum_{(x,y)\in U} \phi(x,y)(e_2) = 1 \right)$$

Les transitions entre deux sommets du graphe sont donc fonctions de différentes éventualités (les éléments de $\mathcal{E}$) et la valeur de ces éventualités dépend de conditions externes (les éléments de E).

On peut, par exemple, supposer que chaque sommet du graphe représente une tâche à effectuer, la succession de 2 tâches se faisant sous une éventualité, fonction d'un certain nombre de conditions.

1.1.5 Remarques

L'absence de conditions est représentée par la fonction booléenne identiquement égale à 1.

Inversement, le fait qu'une éventualité soit la fonction booléenne identiquement nulle signifie que l'on ne passera jamais par l'arc auquel cette éventualité est attachée et qu'il peut en conséquence être supprimé. Une autre façon de présenter ce cas, est de remarquer que $\mathcal{E}$ est un dioïde dont 0 est l'élément blanc et que ϕ peut être considéré comme un renseignement attaché à la relation binaire représentée par le graphe (cf. J. KUNTZMANN). Dans ce cas $\forall (x,y) \in X \times X$
$(x,y) \notin U \Longleftrightarrow \phi(x,y) \equiv 0$.

On supposera donc dans toute la suite que :

$$\forall\ u \in U \qquad \phi(u) \neq 0.$$

De même dans la suite on ne s'intèressera qu'à la manière de sortir d'un sommet, une fois qu'on y est entré. La condition de conservation d'un flot booléen contrôlé (que l'on nommera souvent flot booléen par souci d'écourter) peut donc être notée :

$$\forall\ x \in X \qquad \forall\ e_1 \in \bar{E}$$

$$\left(\sum_{(y,x)\in U} \phi(y,x)(e_1) = 1 \Longrightarrow \exists\ e_2 \in \bar{E} \text{ tel que } \sum_{(x,y)\in U} \phi(x,y)(e_2) = 1 \right)$$

1.2 Types de flots booléens

Les conditions booléennes qui régissent la circulation au travers des noeuds d'un graphe peuvent varier au cours du temps. Selon la façon dont elles varient et les contraintes qui peuvent exister sur ces variations, on est amené à distinguer différents types de flots booléens contrôlés.

1.2.1 Flot booléen complet

Définition : Etant donné un flot booléen ϕ défini sur un graphe G et à valeurs dans $\bar{E}$, ce flot est dit complet si et seulement si :

$$\forall\ x \in X \qquad \left[\sum_{\substack{u\in U \\ u=(x,y)}} \phi(u) \equiv 1 \right]$$

Il s'agit du type de flot booléen le plus simple ; la somme logique des éventualités en sortie d'un sommet étant la fonction booléenne identiquement vraie, dès que l'on entre en un sommet du graphe, on est assuré d'en sortir sans attente.

1.2.2 Flot booléen non complet

Si le flot booléen défini sur un graphe n'est pas complet, il est alors nécessaire que des éventualités deviennent vraies "au bon moment" de façon à permettre le passage du flot. Le positionnement des éventualités peut être indépendant ou non des sommets traversés ; d'autre part une éventualité vraie peut être prise en compte à différents moments dans le temps. Ceci oblige à se définir une loi de cheminement dans le graphe, en fonction de ces éventualités. On supposera une échelle de temps discrète.

a) De la façon la plus générale, on peut supposer que tout temps d'attente est permis en n'importe quel sommet ; il est alors possible de n'avoir, pendant un moment, aucun mouvement dans le graphe, en attendant qu'une éventualité soit positionnée.

b) Une façon plus restrictive d'envisager un flot booléen est d'imposer qu'à tout instant un déplacement ait lieu dans le graphe. On s'autorise alors la possibilité d'une attente du positionnement d'une éventualité en un sommet à condition qu'ailleurs dans le graphe, un mouvement ait lieu, c'est-à-dire que l'ensemble ne soit pas inactif à un moment donné.

c) Enfin de façon stricte, on peut considérer qu'aucun temps d'attente n'est admis en n'importe quel sommet ; il faut alors qu'à la fin de la traversée d'un sommet, une éventualité vraie permette la mise en relation avec le sommet suivant. Un cas particulier d'un tel flot est le flot booléen complet.

En tenant compte du temps, la condition de conservation s'écrira :

$$\forall\, x \in X,\ \forall\, t,\ \forall\, e_1 \in \bar{E}$$

$$\sum_{\substack{u=(y,x)\\ u\in U}} \phi_t(u)(e_1) = 1 \Longrightarrow \exists\, t'>t,\ \exists\, e_2 \in \bar{E} \sum_{\substack{u=(x,y)\\ u\in U}} \phi_{t'}(u)(e_2)=1$$

et des contraintes sur la définition de t' permettent de définir les différents types de flots.

2 - RECOUVREMENTS

2.1 Recouvrement conjonctif d'un ensemble de fonctions booléennes

2.1.1 Eventualités conjonctives

Soit $\mathcal{E}$ un ensemble de fonctions booléennes définies sur un ensemble E de variables booléennes ; deux éventualités f et g appartenant à $\mathcal{E}$ seront dites conjonctives si et seulement si leur produit booléen est différent de la fonction identiquement nulle.

$$\forall\ f, g \in \mathcal{E} \qquad f \text{ et } g \text{ sont conjonctives} \iff f.g \neq 0$$

Cette relation est réflexive et symétrique, mais n'est pas transitive.

2.1.2 Ensemble conjonctif

Tout sous-ensemble $\mathcal{E}'$ de $\mathcal{E}$ sera dit conjonctif si et seulement si le produit logique des fonctions booléennes qu'il contient n'est pas identiquement nul :

$$\mathcal{E}' \subset \mathcal{E} \text{ est conjonctif} \iff \prod_{f \in \mathcal{E}'} f \neq 0$$

2.1.3 Recouvrement conjonctif

a) Définition : $R_{\mathcal{E}} = \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_n\}$ est un recouvrement de $\mathcal{E}$ si et seulement si

$$\forall\ i \in \{1,2,\ldots,n\} \qquad \mathcal{E}_i \subset \mathcal{E} \text{ et } \bigcup_{i=1}^{i=n} \mathcal{E}_i = \mathcal{E}.$$

b) Définition : $R_{\mathcal{E}}$ sera dit conjonctif si et seulement si $\mathcal{E}_i$ est un ensemble conjonctif pour tout i appartenant à $\{1,2,\ldots,n\}$.

c) Définition : On appellera éventualité généralisée associée à $\mathcal{E}_i$, la fonction booléenne $g_i = \prod_{f \in \mathcal{E}_i} f$.

2.2 Recouvrement séquentiel d'un graphe

2.2.1 Ensembles séquentiels de sommets

2.2.1.1 Définition : Soient un graphe $G = (X,U)$ et X_1 et X_2 deux sous-ensembles quelconques de sommets de ce graphe. Le couple (X_1,X_2) est dit séquentiel si et seulement si

- $X_1 \cap X_2 = \emptyset$
- $\forall (x,y) \in X_1 \times X_2$
  $[(x,y) \in U \Longrightarrow \forall\, y' \in X_2 \;\; \exists\, x' \in X_1$ tel que $(x',y') \in U]$

Cette seconde condition signifie que si un sommet de X_2 est accessible depuis X_1, alors tous les sommets de X_2 sont accessibles depuis ceux de X_1.

2.2.1.2 Propriétés :

- $\left.\begin{array}{l} \langle Y,X\rangle \text{ séquentiel} \\ \langle Y,X'\rangle \text{ séquentiel} \end{array}\right\} \Longrightarrow \langle Y, X \cup X'\rangle$ séquentiel
- $\forall\, X' \subset X$, $\langle Y,X\rangle$ séquentiel $\Longrightarrow \langle Y,X'\rangle$ séquentiel
- $\forall\, Y' \supset Y$, $\langle Y,X\rangle$ séquentiel $\Longrightarrow \langle Y',X\rangle$ séquentiel.

2.2.2 Entrées et sorties dans un sous-graphe

Soit un graphe $G = (X,U)$. On considère un sous-graphe partiel $G' = (X',U')$ de G

- $X' \subset X$
- $U' \subset U \cap (X' \times X')$

<u>Définition</u> : On appelle ensemble des entrées et ensemble des sorties de G' les ensembles $D(G')$ et $S(G')$ suivants :

$$D(G') = \{x \in X' \mid \exists\, y \in X - X',\ (y,x) \in U-U'\}$$
$$S(G') = \{x \in X' \mid \exists\, y \in X - X',\ (x,y) \in U-U'\}$$

2.2.3 Sous-graphes séquentiels

On considère deux sous-graphes partiels $G_1 = (X_1,U_1)$ et $G_2 = (X_2,U_2)$ d'un graphe G.

Définition : G_1 et G_2 sont dits séquentiels si et seulement si

$$\langle S(G_1), D(G_2)\rangle \text{ est séquentiel.}$$

2.2.4 Recouvrement séquentiel d'un graphe

Soit un graphe $G = (X,U)$. Une famille $R_G = \{G_1,G_2,\ldots,G_p\}$ de sous-graphes partiels $G_i = (X_i,U_i)$ de G ($U_i \subset U \cap (X_i \times X_i)$) forme un recouvrement de G si et seulement si $X = \bigcup_{i=1}^{i=p} X_i$.

Définition : Un recouvrement R_G de G est dit séquentiel si et seulement si $\forall\, G_i, G_j \in R_G \quad \langle S(G_i), D(G_j)\rangle$ est séquentiel.

2.2.5 Notations

Dans ce qui suit, pour tout sous-ensemble U' de U, on notera org(U') et extr(U') les ensembles suivants :

$$org(U') = \{x \in X \mid \exists\, y \in X \quad (x,y) \in U'\}$$

$$extr(U') = \{y \in X \mid \exists\, x \in X \quad (x,y) \in U'\}$$

ensembles origine et extrémité d'un groupe d'arcs.

3.3 Flot booléen généralisé semi-homomorphe d'un flot booléen

Soit un flot booléen ϕ défini sur un graphe $G = (X,U)$ et à valeurs dans un ensemble E. On considère un recouvrement $R_G = \{G_1, G_2, \ldots, G_n\}$ de G par des sous-graphes $G_i = (X_i, U_i)$ sur lequel est défini un f.b.g. Φ, application de $\bar{U}$ dans $\bar{E}$.

Définition :

Le f.b.g. Φ est dit semi-homomorphe du flot booléen ϕ si et seulement si

$$\forall\, U_{i,j} \in \bar{U},\ \forall\, e \in \bar{E}$$

$$[\Phi(U_{i,j})(e) = 1 \Longrightarrow \forall\, x_j \in D(G_j)/X_i\ \exists\ (x_i, x_j) \in U_{i,j}\ \ \phi(x_i, x_j)(e) = 1]$$

Par conséquent, chaque fois qu'une éventualité généralisée $\Phi(G_i, G_j)$ est vraie en un point e, il existe un ensemble d'arcs de $U_{i,j}$ dont les extrémités recouvrent $D(G_j)$ et pour lesquels les éventualités simples sont vraies au même point ; le f.b.g. Φ ne contredit pas le flot booléen ϕ.

3.4 Flot booléen généralisé homomorphe d'un flot booléen

Définition :

Un f.b.g. Φ est dit homomorphe d'un flot booléen ϕ si et seulement si (avec les notations précédentes)

i) $\forall\, U_{i,j} \in \bar{U},\ \forall\, e \in \bar{E}$

$$[\Phi(U_{i,j})(e) = 1 \Longrightarrow \forall\, x_j \in D(G_j)/X_i\ \exists\ (x_i, x_j) \in U_{i,j}\ \ \phi(x_i, x_j)(e) = 1]$$

ii) $\forall\, U_{i,j} \in \bar{U},\ \forall\, e \in \bar{E},\ \forall\, (x_i, x_j) \in U_{i,j}$

$$[\phi(x_i, x_j)(e) = 1 \Longrightarrow \Phi(U_{i,j})(e) = 1]$$

Le flot booléen généralisé Φ respecte les propriétés du flot booléen ϕ.

3.5 Propriété

Si Φ est un flot booléen généralisé homomorphe de ϕ, les propriétés a), b) et c) sont équivalentes $\forall\ U_{i,j} \in \bar{U}$, $\forall\ e \in \bar{E}$:

a) $\exists (x_i,x_j) \in U_{i,j} \quad \phi(x_i,x_j)(e) = 1$

b) $\Phi(U_{i,j})(e) = 1$

c) $\forall\ x_j \in D(G_j)/X_i\ \exists (x_i,x_j) \in U_{i,j} \quad \phi(x_i,x_j)(e) = 1$

et réciproquement.

Démonstration : Ce résultat résulte immédiatement des points i) et ii) de la définition :

a) $\Longrightarrow$ b) d'après ii) ; b) $\Longrightarrow$ c) d'après i) et
c) $\Longrightarrow$ a).

3.6 Recouvrement ϕ compatible d'un graphe

3.6.1 Notation

Soient un graphe $G = (X,U)$ et $\phi : U \to E$ un flot booléen sur ce graphe.

Pour toute fonction f de $\bar{E}$ on note :

$$U(f) = \{u \in U \mid \forall\ e \in \bar{E}(f(e) = 1 \Longrightarrow \phi(u)(e) = 1)\}$$

et pour tout sous-ensemble U' de U on pose :

$$U'(f) = U(f) \cap U'.$$

3.6.2 f-Séquentialité

3.6.2.1 Définition : Soit $\phi : U \to E$ un flot booléen sur un graphe $G = (X,U)$. On considère deux sous-ensembles X_1 et X_2 de X. Pour toute fonction f de $\bar{E}$, le couple (X_1,X_2) est dit f-séquentiel si et seulement si :

$$\forall (x,y) \in X_1 \times X_2$$
$$[(x,y) \in U(f) \Longrightarrow \forall\ y' \in X_2\ \exists\ x' \in X_1 \text{ tel que } (x',y') \in U(f)]$$

3.6.2.2 Propriété

$$\forall\ X_1,X_2 \subset X,\ \forall\ f,g \in \bar{E}\ \left[\left.\begin{matrix}(X_1,X_2) \text{ g-séquentiel} \\ f \to g\end{matrix}\right\} \Longrightarrow (X_1,X_2) \text{ f-séquentiel}\right]$$

3.6.2.3 Ensemble g-satisfaisant

<u>Définition</u> : Soit g une fonction de $\bar{E}$. Un sous-ensemble H de $\bar{E}$ est g-satisfaisant si et seulement si :

$$- \forall e \in \bar{E} \; [g(e) = 1 \Longrightarrow \exists f \in H \quad f(e) = 1]$$

$$- \forall f \in H, \forall e \in \bar{E} \; [f(e) = 1 \Longrightarrow g(e) = 1]$$

Il existe toujours un ensemble g-satisfaisant : {g}.

<u>Propriété</u> : Si $H \subset \bar{E}$ est g-satisfaisant et si $\forall e \in \bar{E}(f(e) = 1 \Longrightarrow g(e) = 1)$ alors $H \cup \{f\}$ est g-satisfaisant.

<u>3.6.2.4 Propriété</u> : Soient X_1 et X_2 deux sous-ensembles de X et g une fonction de $\bar{E}$. On considère un sous-ensemble H de $\bar{E}$ qui soit g-satisfaisant. Une condition nécessaire et suffisante pour que le couple (X_1,X_2) soit g-séquentiel est qu'il soit f-séquentiel pour toute fonction f de H.

Démonstration :

. Soit (X_1,X_2) g-séquentiel. Si $H \subset \bar{E}$ est g-satisfaisant alors

$$\forall f \in H \left(\begin{array}{l} f \to g \\ (X_1,X_2) \text{ g-séquentiel} \end{array} \Longrightarrow (X_1,X_2) \text{ f-séquentiel} \right)$$

d'après la propriété 4.6.2.2.

. Soit $H \subset \bar{E}$ un ensemble g-satisfaisant.
Supposons (X_1,X_2) f-séquentiel, $\forall f \in H$.
On considère un arc (x,y) appartenant à $U(g) \cap (X_1 \times X_2)$.
Soit $e \in \bar{E}$ tel que $g(e) = 1$

$$\left. \begin{array}{l} H \text{ g-satisfaisant} \Longrightarrow \exists f \in H \text{ telle que } f(e) = 1 \\ (x,y) \in U(g) \subset U(f) \\ (X_1,X_2) \text{ f-séquentiel} \end{array} \right\} \Longrightarrow$$

$\forall y' \in X_2 \; \exists x' \in X_1$ tel que $(x',y') \in U(f)$
et donc $\phi(x',y')(e) = 1$.

En conséquence si $(x,y) \in U(g) \cap (X_1 \times X_2)$, $\forall e \in \bar{E}$ on a

$$g(e) = 1 \Longrightarrow \forall y' \in X_2 \ \exists x' \in X_1 \text{ tel que } \phi(x',y')(e) = 1$$

c'est-à-dire $\forall (x,y) \in X_1 \times X_2$

$$(x,y) \in U(g) \Longrightarrow \forall y' \in X_2 \ \exists x' \in X_1 \text{ tel que } (x',y') \in U(g)$$

ou encore (X_1,X_2) est g-séquentiel.

3.6.3 Définition

Soit un graphe $G = (X,U)$ muni d'un flot booléen ϕ. Un recouvrement R_G de G est compatible avec le flot ϕ (ou ϕ-compatible) si et seulement si $\forall G_i, G_j \in R_G$, $\forall f \in \bar{E}$

. ou $\langle org(U_{i,j}), D(G_j)\rangle$ est f-séquentiel

. ou $U_{i,j}(f) = \emptyset$.

3.7 Propriétés

Propriété 3.7.1

Si R_G est ϕ-compatible, alors pour tout couple (G_i,G_j) de R_G, $\langle org(U_{i,j}), D(G_j)\rangle$ est séquentiel, mais la réciproque est fausse.

Démonstration : Ceci résulte du fait que $\langle org(U_{i,j}),D(G_j)\rangle$ est f-séquentiel pour toute fonction f, donc en particulier pour la fonction identiquement nulle pour laquelle $U(f) = U$.

La réciproque est fausse car la séquentialité pour un ensemble d'arcs $U_{i,j}$ peut être perdue si l'on ne considère qu'un sous-ensemble $U_{i,j}(f)$ de $U_{i,j}$.

Conséquence : Si R_G est ϕ-compatible, alors R_G est un recouvrement séquentiel.

Propriété 3.7.2

Si R_G est ϕ-compatible, alors $E_{i,j}(f) = \{\phi(u) \mid u \in U_{i,j}(f)\}$ est un ensemble conjonctif ou vide pour toute fonction f.

Démonstration : Si $E_{i,j}(f)$ n'est pas vide, en un point e de $\bar{E}$, toutes les fonctions $\phi(u)$ de $E_{i,j}(f)$ sont vraies dès que f est vraie en ce point et leur produit n'est pas nul.

3.8 Condition nécessaire et suffisante

Dans ce paragraphe, nous allons vérifier que, pour qu'un recouvrement soit ϕ-compatible, il n'est pas nécessaire de vérifier la relation de f-séquentialité pour toutes les fonctions f de $\bar{E}$, mais que l'on peut se restreindre à certaines classes de fonctions.

3.8.1 Notation

Etant donné un recouvrement R_G d'un graphe G, pour tout couple (G_i, G_j) de sous-graphes de R_G on pose :

$$F_{i,j} = \{\phi(u) \mid u \in U_{i,j}\}$$

3.8.2 Remarque

Pour tout sous-graphe G_i et G_j de R_G et pour toute fonction f de $\bar{E}$, les deux notations a) et b) sont équivalentes :

a) $\langle \text{org}\,(U_{i,j}), D(G_j)\rangle$ f-séquentiel

b) $\text{extr.}(U_{i,j}(f)) = D(G_j)$

3.8.3 C.N.S.

Soit $\phi : U \to E$ un flot booléen sur un graphe $G = (X,U)$. Une condition nécessaire et suffisante pour qu'un recouvrement R_G de G soit ϕ-compatible est que :

$\forall\, G_i, G_j \in R_G$. ou $F_{i,j} = \emptyset$

. ou $\forall\, f \in F_{i,j}$ $\exists\, H_{i,j}(f) \subset \bar{E}$ tel que

i) $H_{i,j}(f)$ est f-satisfaisant

ii) $\forall\, g \in H_{i,j}(f)$ $\text{extr.}(U_{i,j}(g)) = D(G_j)$

Démonstration :

. Soit R_G recouvrement ϕ-compatible de G.

On considère G_i, $G_j \in R_G$ et une fonction f de $F_{i,j}$ ($F_{i,j} \neq \emptyset$).

On est assuré qu'il existe toujours un sous-ensemble $H_{i,j}(f)$ de $\bar{E}$ qui soit f-satisfaisant : $H_{i,j}(f) = \{f\}$.

Soit $H_{i,j}(f) \subset \bar{E}$ qui est f-satisfaisant

$\langle org(U_{i,j}), D(G_j)\rangle$ est f-séquentiel par hypothèse.

De la propriété 4.6.2.4 on déduit que

$\forall\ g \in H_{i,j}(f)$ $\langle org(U_{i,j}), D(G_j)\rangle$ est g-séquentiel

$\forall\ g \in H_{i,j}(f)$, $\forall\ x_j \in D(G_j)$ $\exists (x_i,x_j) \in U_{i,j}(g)$

c'est-à-dire $extr.(U_{i,j}(g)) = D(G_j)$.

. Supposons que $\forall\ f \in F_{i,j}(\neq\emptyset)$ $\exists\ H_{i,j}(f) \subset \bar{E}$ tel que

1/ $H_{i,j}(f)$ est f-satisfaisant

2/ $\forall\ g \in H_{i,j}(f)$ $extr.(U_{i,j}(g)) = D(G_j)$

c'est-à-dire $\langle org(U_{i,j}),D(G_j)\rangle$ est g-séquentiel.

On désire montrer que

$\forall\ f \in \bar{E}$ $\langle org(U_{i,j}),D(G_j)\rangle$ est f-séquentiel (ou $U_{i,j}(f)\neq\emptyset$).

Soit $f \in \bar{E}$

1er cas : $f \in F_{i,j}$

$$\left.\begin{array}{l} \exists\ H_{i,j}(f) \text{ f-satisfaisant} \\ \forall\ g \in H_{i,j}(f)\ \langle org(U_{i,j}),D(G_j)\rangle \text{ g-séquentiel} \end{array}\right] \Longrightarrow$$

d'après propriété 4.6.2.4 : $(org(U_{i,j}),D(G_j))$ f-séquentiel.

2ème cas : $f \notin F_{i,j}$

a) Supposons que : $\exists\ f' \in F_{i,j}$ telle que $\forall\ e \in \bar{E}(f(e)=1 \Longrightarrow f'(e) = 1)$ alors $\exists\ H_{i,j}(f') \subset \bar{E}$ qui est f'-satisfaisant.

D'après 4.6.2.3 $H_{i,j}(f') \cup \{f\}$ est f' satisfaisant et $\forall$ g $\in H_{i,j}(f') \cup \{f\}$ $\langle org(U_{i,j}),D(G_j)\rangle$ est g-séquentiel,

ce qui implique (d'après 4.6.2.4) que

$(org(U_{i,j}),D(G_j))$ est f'-séquentiel

or $f \to f'$ et on en déduit (d'après 4.6.2.2) que

$(org(U_{i,j}),D(G_j))$ est f-séquentiel

b) Supposons $\nexists$ f' $\in F_{i,j}$ tel que $\forall$ e $\in \bar{E}$ (f(e)=1 $\Longrightarrow$ f'(e) = 1).

Soit (x,y) $\in U_{i,j}(f)$.

Ceci implique que $\phi(x,y) \in F_{i,j}$ et $\forall$ e $\in \bar{E}$(f(e) = 1 $\Longrightarrow$ $\phi(x,y)(e) = 1$) ce qui contredit l'hypothèse b).

On en déduit que $U_{i,j}(f) = \emptyset$.

3.8.4 Corollaire

Une condition nécessaire et suffisante pour qu'un recouvrement R_G d'un graphe G soit compatible avec un flot booléen ϕ défini est que :

$\forall$ $G_i,G_j \in R_G$. ou $F_{i,j} = \emptyset$

. ou $\forall$ f $\in F_{i,j}$ extr.$(U_{i,j}(f)) = D(G_j)$

Démonstration : cas particulier du théorème précédent avec

$\forall$ G_i,G_j $H_{i,j}(f) = \{f\}$.

3.9 Théorème

Soient - $\phi : U \to E$ un flot booléen sur un graphe G et

- R_G un recouvrement de G.

Alors, il existe un flot booléen généralisé Φ défini sur R_G qui est semi-homomorphe de ϕ.

Démonstration : Soit $F_{i,j} = \{\phi(u) \mid u \in U_{i,j}\}$.

On considère les fonctions de $\bar{E}$ vraies en un seul point de $\bar{E}$

$\forall\ e \in \bar{E} \quad e \rightsquigarrow \bar{e} \in \bar{E}$ telle que $\bar{e}(e) = 1$

$\bar{e}(e') = 0$ si $e' \neq e$

et pour toute fonction f de $F_{i,j}$ on écrit f comme somme de telles fonctions $f = \Sigma\ e^k$.

On note $\overline{F_{i,j}}(f)$ l'ensemble de ces monômes associés à une fonction f de $F_{i,j}$.

On pose $\overline{F_{i,j}} = \bigcup_{f \in F_{i,j}} \bar{F}_{i,j}(f)$.

Soit $H_{i,j} = \{f \in \bar{F}_{i,j} \mid D(G_j) = \text{extr.}(U_{i,j}(f)\}$.

On définit Φ par :

$$\Phi(U_{i,j}) \equiv \sum_{f \in H_{i,j}} f \quad \text{si } H_{i,j} \neq \emptyset$$
$$\equiv 0 \quad \text{sinon.}$$

Soit $e \in \bar{E}$ tel que $\Phi(U_{i,j})(e) = 1$ (ce qui suppose $H_{i,j} \neq \emptyset$)
ceci implique $\exists\ f \in H_{i,j}$ $f(e) = 1$
$D(G_j) = \text{extr.}(U_{i,j}(f))$

donc $\forall\ x_j \in D(G_j)\ \exists\ (x_i, x_j) \in U_{i,j}(f)$
c'est-à-dire $\forall\ x_j \in D(G_j)\ \exists\ (x_i, x_j) \in U_{i,j}$ tel que $\phi(x_i, x_j)(e) = 1$
ce qui montre que Φ est semi-homomorphe de ϕ.

3.10 Algorithme de construction

La démonstration du théorème 3.9 fournit un algorithme permettant de construire un f.b.g. semi-homomorphe d'un flot booléen ϕ qui est basé sur la décomposition systématique de toute éventualité $\phi(u)$ comme somme de ses facteurs premiers. Cette décomposition peut être longue et surtout coûteuse en occupation mémoire si le nombre de variables de E est important.

La démonstration suivante fournit un algorithme qui se borne à écrire toute éventualité sous forme de somme de monômes et de n'en

retenir que les implicants essentiels.

$\forall f \in F_{i,j}$, on écrit f sous la forme d'une somme de monômes $f = \Sigma f^k$ et on note $\overline{F}_{i,j}(f)$ l'ensemble de ces monômes.

On pose $\overline{F}_{i,j}(f) = \{f^k \in \overline{F}_{i,j}(f) \mid \nexists f^\ell \in \overline{F}_{i,j}(f)$ telle que $f^\ell \to f^k\}$ et $\overline{F}_{i,j} = \bigcup_{f \in F_{i,j}} \overline{F}_{i,j}(f)$.

On recherche alors un recouvrement de $\overline{F}_{i,j}$ par des sous-ensembles conjonctifs ; on est assuré qu'il en existe toujours un, car le recouvrement discret $R_{\overline{F}_{i,j}} = \{\{f\} \mid f \in \overline{F}_{i,j}\}$ est nécessairement conjonctif.

On note $R_{\overline{F}_{i,j}} = \{\overline{F}^1_{i,j},\ldots,\overline{F}^\ell_{i,j},\ldots,\overline{F}^L_{i,j}\}$ ce recouvrement conjonctif. Soit $\overline{F}^\ell_{i,j} \in R_{\overline{F}_{i,j}}$. On considère les sous-ensembles H de $\overline{F}^\ell_{i,j}$ tels que $\bigcup_{f \in H} \text{extr.}(U_{i,j}(f)) = D(G_j)$. La relation d'inclusion sur les ensembles définit pour les parties de $\overline{F}^\ell_{i,j}$ un demi-treillis.

On note $\mathcal{H}_\ell$ l'ensemble $\{H_1,\ldots,H_{q_\ell}\}$ des sous-ensembles minimaux de $\overline{F}^\ell_{i,j}$ vérifiant $D(G_j) = \bigcup_{f \in H_q} \text{extr}(U_{i,j}(f))$ (s'il existe de tels sous-ensembles) et

on pose $$\Phi(U_{i,j}) = \sum_{\overline{F}^\ell_{i,j} \in R_{\overline{F}_{i,j}}} \sum_{H \in \mathcal{H}_\ell} \prod_{f \in H} f.$$

Soit $e \in \bar{E}$ tel que $\Phi(U_{i,j})(e) = 1$

$\Longrightarrow \exists\ \overline{F}^\ell_{i,j} \in R_{F_{i,j}}$, $\exists\ H \in \mathcal{H}_\ell$ tels que $\prod_{f \in H} f(e) = 1$

$\Longrightarrow \forall f \in H \quad f(e) = 1$

$\Longrightarrow \forall x_j \in D(G_j)\ \exists\ (x_i,x_j) \in U_{i,j}$ tel que $\phi(x_i,x_j)(e) = 1$
c'est-à-dire que Φ est semi-homomorphe de ϕ.

3.11 Théorème

Soit Φ un f.b.g. sur un graphe G, défini à l'aide d'un recouvrement R_G de G. Si Φ est homomorphe d'un flot booléen ϕ défini sur G, alors le recouvrement R_G est ϕ-compatible.

Démonstration :

Soit $f \in \bar{E}$ $\quad f \neq \Phi(U_{i,j})$

. Si $\exists$ e $\in \bar{E}$ tel que $f(e) = 1$ et $\Phi(U_{i,j})(e) = 0$ alors, d'après le point ii) de la définition de l'homomorphie :

$\forall\ (x_i,x_j) \in U_{i,j} \quad \phi(x_i,x_j)(e) = 0$
ce qui implique $U_{i,j}(f) = \emptyset$

. Si $\nexists$ e $\in \bar{E}$ tel que $f(e) = 1$ et $\Phi(U_{i,j})(e) = 0$

alors $\quad f(e) = 1 \Longrightarrow \Phi(U_{i,j})(e) = 1 \quad (\forall\ e \in \bar{E})$

comme d'après le point i) de la définition de l'homomorphie, $\langle org.(U_{i,j}), D(G_j)\rangle$ est $\Phi(U_{i,j})$-séquentiel, la propriété 4.6.2.2 assure que $\langle org(U_{i,j}),D(G_j)\rangle$ est f-séquentiel.

Ce qui démontre que R_G est ϕ-compatible.

3.12 Théorème

Soit R_G un recouvrement d'un graphe G = (X,U) muni d'un flot booléen $\phi(\phi : U \to E)$.

Si R_G est ϕ-compatible, alors il existe un f.b.g. Φ construit sur R_G qui est homomorphe de ϕ.

Démonstration :

Soit Φ le f.b.g. semi-homomorphe de ϕ construit sur le recouvrement R_G lors de la démonstration du théorème 3.9.

Soient $(x_i,x_j) \in U_{i,j}$ et $e \in \bar{E}$ tels que $\phi(x_i,x_j)(e) = 1$

$\Longrightarrow$ $\exists\ \bar{e} \in \bar{F}_{i,j}$ telle que $\bar{e}(e) = 1$. On suppose alors que $U_{i,j} \neq \emptyset$ comme le permet la remarque 1.1.4).

a) Si $extr(U_{i,j}(\bar{e})) \neq D(G_j)$ alors $\bar{e} \notin H_{i,j}$.
D'autre part $\forall\ f \in F_{i,j}$ $f(e) = 0$ (à cause de la décomposition des fonctions de $F_{i,j}$), donc $\Phi(U_{i,j})(e) = 0$ et Φ n'est pas homomorphe de ϕ.

b) Si $extr(U_{i,j}(\bar{e})) = D(G_j)$ alors $\bar{e} \in H_{i,j}$ et $\Phi(U_{i,j})(e) = 1$ ce qui assure que Φ est homomorphe de ϕ.

Or si le recouvrement R_G est ϕ-compatible

ou $U_{i,j}(\bar{e}) = \emptyset$ (ce qui n'est pas le cas puisque $(x_i, x_j) \in U_{i,j}(\bar{e})$)

ou $\langle org(U_{i,j}), D(G_j)\rangle$ est $\bar{e}$-séquentiel

c'est-à-dire $extr(U_{i,j}(\bar{e})) = D(G_j)$
ce qui assure le point b).

Donc $\forall (x_i, x_j) \in U_{i,j}$

$$\phi(x_i, x_j)(e) = 1 \Longrightarrow \Phi(U_{i,j})(e) = 1$$

c'est-à-dire que Φ est homomorphe de ϕ.

3.13 Algorithme de construction

Comme dans le paragraphe 3.10 nous allons vérifier que, si R_G est ϕ-compatible, on peutconstruire un f.b.g. Φ homomorphe de ϕ à partir d'une écriture des éventualités sous forme de somme d'implicants essentiels. Reprenons la démonstration du § 3.10.
$\forall\ G_i, G_j \in R_G$, $\forall\ f \in F_{i,j}$ l'ensemble $\bar{F}_{i,j}(f)$ est f-satisfaisant.
Si R_G est ϕ-compatible, d'après le théorème 3.8.3 on a :

$$\forall\ g \in \bar{F}_{i,j}(f)\ extr(U_{i,j}(g)) = D(G_j)$$

En conséquence $card(H) = 1$ et le produit $\prod_{f \in H} f$ n'est plus nécessaire pour obtenir un recouvrement des sommets.

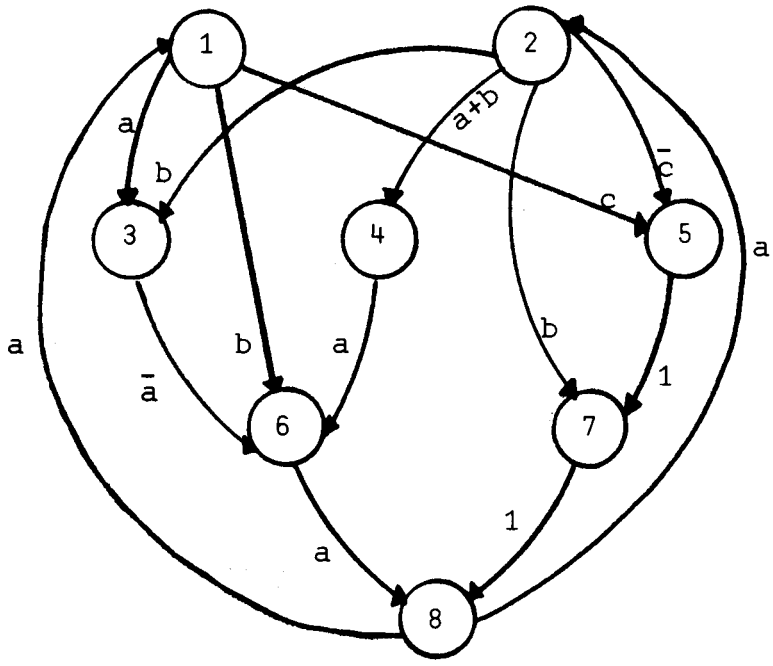
(De plus, d'après 3.7.2. $F_{i,j}(f)$ est conjonctif ou vide pour tout f dès que R_G est ϕ-compatible).

On pose $\Phi(U_{i,j}) = \sum_{f \in F_{i,j}} f$.

Soient $(x_i, x_j) \in U_{i,j}$ et $e \in \bar{E}$ tels que $\phi(x_i, x_j)(e) = 1$ ceci implique $\exists$ $f \in F_{i,j}$ telle que $f(e) = 1$ et par conséquent $\Phi(U_{i,j})(e) = 1$. D'autre part, comme R_G est ϕ-compatible, $D(G_j) = \text{extr}(U_{i,j}(f))$ c'est-à-dire que $\forall$ $x_j \in D(G_j)$ $\exists$ $(x_i, x_j) \in U_{i,j}$ tel que $\phi(x_i, x_j)(e) = 1$. Ce qui démontre que Φ est un f.b.g. homomorphe de ϕ.

3.14 Exemple

On considère le flot booléen ϕ défini sur un graphe G comme ci-dessous



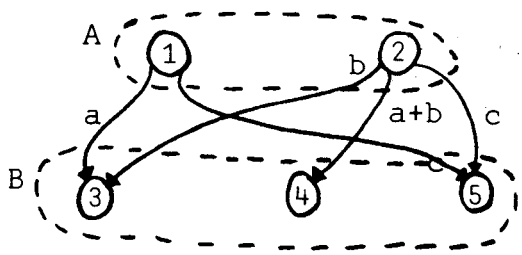
	1	2	3	4	5	6	7	8
1			a		c			
2			b	a+b	$\bar{c}$			
3						$\bar{a}$		
4						a		
5							1	
6								a
7								1
8	a	a						

Considérons le recouvrement suivant

$$R_G = \{\{1,2\}, \{3,4,5\}, \{6,7,8\}\}$$

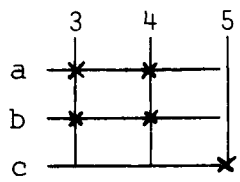
et construisons sur ce recouvrement un flot booléen généralisé à partir de ϕ comme dans le théorème précédent.

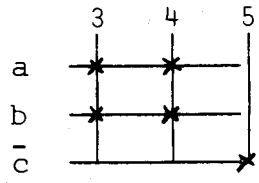
. Soient A = {1,2} et B = {3,4,5}



$\bar{F}_{AB} = \{a,b,c,\bar{c}\}$

$R_{\bar{F}_{A,B}} = \{\{a,b,c\}, \{a,b,\bar{c}\}\}$

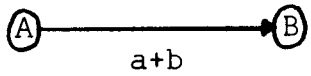




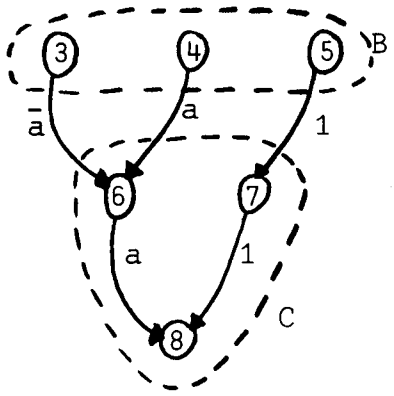
Recouvrements de {3,4,5} minimaux par

{a,c} et {b,c} et par $\{a,\bar{c}\}$ et $\{b,\bar{c}\}$

$\Phi(A,B) \equiv (a.c + b.c) + (a.\bar{c} + b.\bar{c}) = a+b$

on a donc 

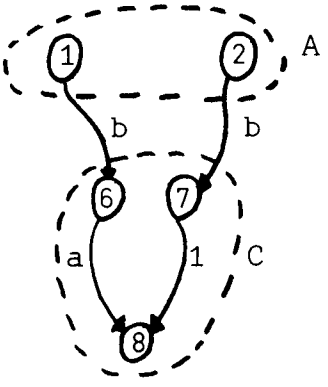
. Soit C = {6,7,8}



$\bar{F}_{BC} = \{a,\bar{a},1\}$

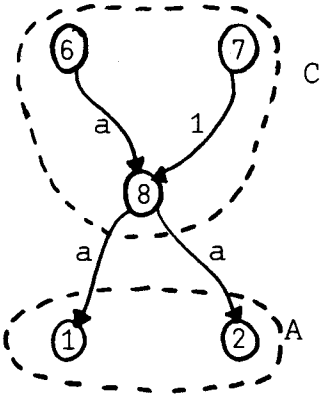
$R_{\bar{F}_{BC}} = \{\{a,1\}, \{\bar{a},1\}\}$

$\Phi(B,C) = a.1 + \bar{a}.1 = a + \bar{a} = 1$

. 

$R_{\bar{F}_{AC}} = \{b\}$

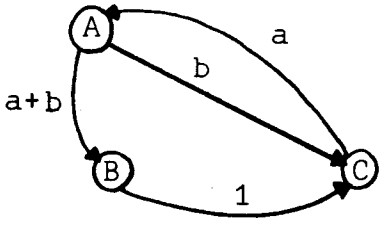
$\Phi(A,C) = b$

. 

$R_{\bar{F}_{CA}} = \{a\}$

$\Phi(C,A) = a$

Le flot booléen généralisé Φ sur G est donc :



BUS LILLE

Ce f.b.g. Φ n'est pas homomorphe de ϕ.

En effet $\phi(1,5)(\bar{a}\bar{b}c) = 1 \not\Longrightarrow \Phi(A,B)(\bar{a}\bar{b}c) = 1$. On remarque qu'à cet

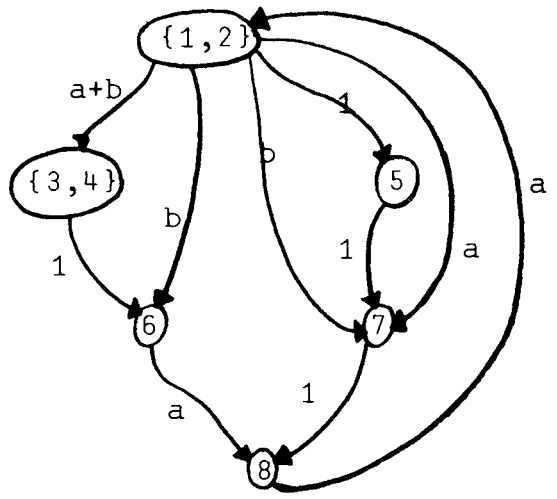
endroit le recouvrement de G n'est pas ϕ-compatible :

$$<org(U_{AB}), D(B)> = (\{1\},\{3,4,5\})$$

n'est pas C-séquentiel.

Considérons le recouvrement $R_G = \{\{1,2\}, \{3,4\}, \{5\}, \{6\}, \{7\}, \{8\}\}$ qui est ϕ-compatible, alors la construction d'un f.b.g. d'après le théorème précédent donne un f.b.g. Φ' qui est homomorphe de ϕ.

f.b.g. Φ' homomorphe de ϕ

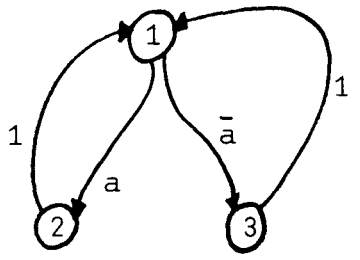


4 - PROBLEMES DE FLOTS BOOLEENS GENERALISES HOMOMORPHES D'UN FLOT BOOLEEN SIMPLE

4.1 Construction d'un f.b.g.

La démonstration du théorème 3.10 fournit un algorithme qui permet de construire un flot booléen généralisé Φ homomorphe d'un flot booléen ϕ à partir d'un recouvrement ϕ-compatible de G et le théorème 3.4 permet de ramener toute construction d'un f.b.g. à une construction d'un recouvrement ϕ-compatible de G. Le problème posé est alors : comment construire un recouvrement R_G ϕ-compatible sur un graphe G ?

Notons tout d'abord qu'il n'existe pas nécessairement une partition propre ϕ-compatible sur un graphe, comme le montre l'exemple ci-après :



{{1}, {2,3}}, {{3}, {1,2}} et

{{2}, {1,4}} ne sont pas des recouvrements ϕ-compatibles.

La construction d'un f.b.g. suppose d'abord que l'on puisse trouver deux ensembles de sommets localement ϕ-compatibles. Ensuite un examen des successeurs ou des prédécesseurs permettra d'effectuer des regroupements ou des éclatements d'ensembles de sommets et si l'itération du procédé converge on obtiendra un recouvrement ϕ-compatible.

Une autre méthode pourraît être d'effectuer un recouvrement des arcs du graphe en fonction d'un recouvrement des implicants des éventualités et de rechercher un recouvrement ϕ-compatible à partir de ce recouvrement.

Nous n'examinerons pas de façon générale le problème de construction d'un f.b.g. homomorphe ou semi-homomorphe d'un flot booléen, mait traiterons dans les chapitres suivants un cas particulier applicable aux problèmes de conception d'un bloc de commande micro-programmé.

4.2 Construction d'un f.b.g. réalisable, optimal

La notion de flot réalisable apparaît lorsque sont introduites un certain nombre de contraintes et le flot est dit réalisable s'il respecte ces contraintes. De plus, la construction d'un f.b.g. réalisable peut être effectuée en recherchant une optimisation du flot selon des critères divers : longueur de chemins généralisés, simultanéité maximale, temps, codage...

Dans la suite, noús introduirons des contraintes entre les différents sommets du graphe en nous attachant, en particulier, au cas où les sommets représentent des tâches à enchaîner selon certaines conditions et où les contraintes portent sur des restrictions de simultanéité entre ces tâches.

C H A P I T R E 3

PROCESSUS ELEMENTAIRE

III.1. Introduction

III.2. Processus élémentaire

III.3. Décomposition d'un processus élémentaire

III.4. Déroulement d'un processus élémentaire

1 - INTRODUCTION

1.1 Notion de processus

La notion de processus est maintenant chose courante ; elle correspond, de façon générale, à l'idée de progression ou de développement de quelque chose, par exemple, d'une maladie ou d'un travail. Cette progression peut être soit continue, soit discrète et nous ne considérerons, par la suite, que le cas d'un développement discret. Ceci implique le découpage du processus en un certain nombre d'étapes et, par extension, c'est l'ensemble et l'enchaînement de ces étapes que recouvre le mot processus. En particulier, un processus peut représenter un travail à effectuer, plus ou moins complexe, décomposé en tâches distinctes et enchaînées les unes aux autres (par exemple, opérations diverses d'une chaîne de fabrication) ; un processus peut également être un ensemble de directives ordonnant un travail : commande d'une machine outil, unité de commande micro-programmée d'un calculateur, programme d'ordinateur.

Les diverses tâches qui composent un processus, ne s'enchaînent pas nécessairement de façon séquentielle. D'une part, certaines d'entre-elles, intéressant des organes différents, peuvent s'exécuter de façon simultanée. D'autre part, toutes ne sont pas nécessairement exécutées : un certain nombre de conditions peuvent influer sur les divers enchaînements.

De façon générale, un flot booléen paramétré sur un graphe, tel que nous l'avons défini au chapitre II, permet donc de formaliser le concept de processus : les sommets du graphe représentent les diverses tâches à exécuter, les arcs, les enchaînements de ces tâches et les éventualités associées aux arcs, les conditions qui régissent ces enchaînements en fonction d'indicateurs exprimés par des variables booléennes.

De même, un flot booléen généralisé permet de représenter des regroupements de tâches et les propriétés du flot booléen généralisé assure le respect des enchaînements des tâches lorsque sont effectués des regroupements.

Ces notions permettent donc une approche mathématique de la notion de processus et l'assimilation d'un bloc de commande micro-programmé à un processus dit "élémentaire" permettra d'en formaliser les problèmes.

Le déroulement d'un processus est une succession de travaux qui est fonction d'une suite de conditions externes ; un certain nombre de définitions préalables sont nécessaires pour préciser ces notions et sont données dans les paragraphes suivants.

1.2 Chaînes et chemins

Soit ϕ un flot booléen défini sur un graphe $G = (X,U)$ et à valeurs dans un ensemble E de fonctions booléennes.

1.2.1 Chaîne

Définition : on appelle chaîne d'éventualités, tout élément du monoïde libre E^* engendré par l'opération de concaténation sur E.

1.2.2 Chemin

- Définition : on appelle chemin, toute suite ordonnée $(u_1,u_2,\ldots,u_n)$, finie ou non, d'arcs de U tels que :

$$\forall\ i \in \{1,2,\ldots,n-1\} \quad extr(u_i) = org(u_{i+1})$$

- $x_0 = org(u_1)$ est dit sommet initial et $x_n = extr(u_n)$ sommet final. Tout chemin $c = (u_1,u_2,\ldots,u_n)$ tel que $x_0 = org(u_1)$ et $x_n = extr(u_n)$ est dit "menant de x_0 à x_n" et l'on note $C_{x,y}$ l'ensemble des chemins menant de x à y.

- Remarque : tout chemin $c = (u_1,u_2,\ldots,u_n)$, il est possible d'associer une chaîne d'éventualités $\phi(u_1) \circ \phi(u_2) \circ \cdots \circ \phi(u_n)$, mais la réciproque est fausse.

1.2.3 Chemins exclusifs

- Définition : Etant donnés deux chemins $c = (u_1, u_2, \ldots, u_p)$ et $d = (v_1, v_2, \ldots, v_q)$ dans un graphe $G = (X,U)$ muni d'un flot booléen ϕ, c et d sont dits exclusifs si et seulement si

$$\text{i)} \quad org(u_1) = org\ (v_1)$$

$$\text{ii)} \quad \phi(u_1).\phi(v_1) \equiv 0$$

- Cette relation binaire sur l'ensemble des chemins du graphe G est symétrique et transitive ; elle n'est cependant pas réflexive.

2 - PROCESSUS ELEMENTAIRE

2.1 Réseau

2.1.1 Définition

On appelle réseau, un triplet $R = \langle G,D,T \rangle$ où

- $G = (X,U)$ est un graphe
- $D \subset X$ est un ensemble non vide, dit ensemble des sommets de départ
- $T \subset X$ est un ensemble non vide, dit ensemble des sommets terminaux

tel que :

$$\forall\ y \in X \quad \exists\ x \in D \quad \text{tel que } C_{x,y} \neq \emptyset.$$

2.1.2 Flot booléen sur un réseau

Soient un réseau $R = \langle G,D,T \rangle$ (avec $G = (X,U)$), E un ensemble de fonctions booléennes et ϕ une application de U dans E.

On considère le graphe $G' = (X,U')$ défini par

$$U' = U \cup \{(\tau,d) \ / \ \forall\ \tau \in T,\ \forall\ d \in D \quad (\tau,d) \notin U\}$$

et la fonction ϕ' défini sur G' à partir de ϕ par

- $\forall\ u \in U \qquad \phi'(u) \equiv \phi(u)$
- $\forall\ u \in U'-U \quad \phi'(u) \equiv 1$

Définition : L'application ϕ de U dans E est dit flot booléen sur le réseau R si et seulement si l'application ϕ' est un flot booléen sur le graphe $G' = (X,U')$.

2.2 Processus

Définition : On appelle processus un triplet $P = \langle R,E,\phi\rangle$

où
- $R = \langle G,D,T\rangle$ est un réseau
- E un ensemble de fonctions booléennes
- ϕ un flot booléen sur R, à valeurs dans E

avec les significations suivantes :

- l'ensemble des sommets représente l'ensemble des tâches
- l'ensemble des arcs représente l'enchaînement de ces tâches
- l'ensemble des éventualités représente les conditions régissant les enchaînements
- et le passage sur un sommet représente l'exécution de la tâche représentée par ce sommet.

Cette définition très générale doit être précisée pour chaque application en tenant compte surtout du facteur temps et, en particulier, pour être rendue apte à la description d'un bloc de commande micro-programmé.

2.3 Sous-processus

2.3.1 Sous-graphe convexe

Soit un graphe $G = (X,U)$.

Un sous-graphe $G' = (X',U')$ de G est convexe si et seulement si

$$\forall (x,y) \in X' \times X' \quad C_{x,y}(G) \subset C_{x,y}(G').$$

2.3.2 Sous-Réseau

$R' = \langle G',D',T' \rangle$ est un sous-réseau d'un réseau $R = \langle G,D,T \rangle$ si et seulement si

- R' est un réseau
- G' est un sous-graphe de G.

2.3.3 Sous-processus

On considère un processus $P = \langle R,E,\phi \rangle$ avec $R = \langle G,D,T \rangle$.

On appelle sous-processus, la restriction du flot booléen ϕ à un sous-réseau $R' = \langle G',D',T' \rangle$ de R tel que

- G' est un sous-graphe convexe de G
- $D' = D(G') = \{x \in X' \mid \exists\, y \in X-X' \text{ tel que } (y,x) \in U\}$

2.4 Processus élémentaire

2.4.1 Définition

On appelle processus élémentaire un processus $P = (R,E,\phi)$ qui possède les propriétés suivantes :

i) toute composante fortement connexe de graphe G induit un sous-processus ;

ii) toutes les tâches ont une même durée d'exécution, égale à l'unité de temps et sont ininterruptibles ;

iii) le passage sur un arc est instantané.

2.4.2 Processus élémentaire et micro-programmation

Considérons un bloc de commande micro-programmé d'un calculateur. En dehors de toute représentation technique, on peut considérer un micro-programme comme un ensemble de micro-opérations (les sommets

d'un réseau) qui sont enchaînées les unes aux autres (les arcs) en fonction de conditions (les éventualités). Ces micro-opérations sont ininterruptibles et exécutées en un cycle d'horloge ; le passage sur un arc n'est qu'une représentation logique de l'enchaînement et ne prend aucun temps.

A chaque cycle d'horloge (à chaque unité de temps) un ensemble de micro-opérations (formant une micro-instruction) est exécutée. Il est donc nécessaire de décrire de quelle façon le réseau peut être parcouru, mais la prise en compte des phénomènes de bouclage oblige à étudier préalablement une décomposition d'un processus élémentaire.

3 - DECOMPOSITION D'UN PROCESSUS ELEMENTAIRE

3.1 Définitions

Soit $C_{x,y}$ l'ensemble des chemins de x à y dans un graphe $G = (X,U)$. Pour tout chemin c de $C_{x,y}$ on note :

- $S_c^+ = \{x \in X \mid \exists\, u \in c \text{ tel que } x = extr(u)\}$
- $S_c^- = \{x \in X \mid \exists\, u \in c \text{ tel que } x = org(u)\}$
- $S_c = S_c^+ \cup S_c^-$
- $\bar{S}_c = S_c^+ \cap S_c^-$.

3.2 Processus simple

Définition : Un processus est simple si et seulement si le graphe sur lequel il est défini ne comporte aucune composante fortement connexe (propre ou non).

3.3 Processus élémentaire fermé

Définition : Un processus élémentaire est fermé si et seulement si le graphe sur lequel il est défini est fortement connexe.

3.4 Processus élémentaire cyclique

3.4.1 Définition

Etant donné un processus élémentaire $P = \langle R, E, \phi \rangle$, P est dit cyclique si et seulement si les 2 conditions suivantes sont vérifiées:

i) P est un processus fermé

ii) $\forall\ x,y \in X,\ \forall\ c \in C_{x,y}$

$$\bar{S}_c \cap D = \emptyset \implies \text{card}\ (c) < \infty$$

c'est-à-dire que tout chemin dans G qui ne passe pas par D est fini ou encore que tout chemin de cardinalité infinie passe nécessairement par D.

3.4.2 Lemme

a) Etant donné un processus élémentaire P quelconque,

$\forall\ x \in X-D,\ \exists\ y \in D,\ \exists\ c \in C_{y,x}$ tels que $\begin{cases} - \text{card}\ c < \infty \\ - D \cap S_c^- = \{y\} \end{cases}$

Etant donné un processus élémentaire fermé P

$\forall\ x \in X-D,\ \exists\ y \in D,\ \exists\ c \in C_{x,y}$ tels que $\begin{cases} - \text{card}\ c < \infty \\ - D \cap S_c^+ = \{y\}. \end{cases}$

Démonstration :

a) Il suffit, à partir de x, de parcourir les arcs du réseau dans le sens inverse de leur orientation sans passer deux fois sur le même arc et de s'arrêter au premier sommet rencontré dans D.
On est assuré qu'il en existe toujours un car la définition d'un processus précise que : $\forall\ x\ \ \exists\ y \in D$ tel que $C_{y,x} \neq \emptyset$.

b) Si P est fermé, $\forall\ C_{x,y} \in X,\ C_{x,y} \neq \emptyset$. On met en évidence un chemin possédant la propriété ci-dessus en parcourant les arcs du réseau dans le sens de leur orientation, sans parcourir deux fois le même arc et en s'arrêtant au premier sommet rencontré dans D.

3.4.3 Théorème

Dans la définition 3.4.1, on peut remplacer le point ii) par :

ii bis) $\forall\ x,y \in D,\ \forall\ c \in C_{x,y}$

$$\bar{S}_c \cap D = \emptyset \implies \text{card } c < \infty$$

et les 2 définitions sont équivalentes.

Démonstration :

ii) $\implies$ iibis) car $D \subset X$

a) Soient x' et $y' \in X-D$, $c \in C_{x',y'}$ tels que

- card $c = \infty$
- $D \cap \bar{S}_{c'} = \emptyset$

b) Lemme 3.4.2 a) $\implies$

$\exists\ x \in D,\ c_1 \in C_{x,x'}$ tels que

- card $c_1 < \infty$
- $D \cap S^-_{c_1} = \{x\}$

c) Lemme 3.4.2 b) $\implies$

$\exists\ y \in D,\ c_2 \in C_{y',y}$ tels que

- card $c_2 < \infty$
- $D \cap \{org(u_i) \mid u_i \in c\} = \{y\}$
- $D \cap S^+_{c_2} = \{y\}$

a,b,c) $\implies c = c_1\, c' c_2 \in C_{x,y}$ $\quad x,y \in D$ est tel que

- card $c = \infty$
- $\bar{S}_c \cap D = \emptyset$

ce qui montre que ii) $\implies$ ii bis)
et l'équivalence est démontrée.

3.4.4 Propriétés

propriété 3.4.4.1 :

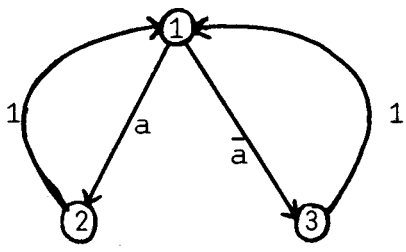
Si un processus élémentaire $P = \langle R, E, \phi \rangle$ ne contient pas de sous-processus cyclique et si le praphe G sur lequel il est défini est fortement connexe, alors P est cyclique.

Démonstration : Si P ne contient pas de sous-processus cyclique, le sous-processus P' obtenu en supprimant les arcs entrant en D est simple ; donc, pour tout chemin c' dans P', card c' $< \infty$ et par conséquent, pour tout chemin c dans P et s'arrêtant en D, card c $\leq$ card c'+1 $< \infty$. D'autre part G étant fortement connexe, P est fermé, ce qui démontre que P est un processus cyclique.

propriété 3.4.4.2 :

Un processus élémentaire cyclique peut contenir des sous-processus cycliques.

Exemple : Considérons le processus suivant :



D = {1}

T = {2,3}

Le processus de graphe {1,2,3} est cyclique et il contient 2 sous-processus de graphes respectifs {1,2} et {1,3} qui sont cycliques.

propriété 3.4.4.3 :

Une condition nécessaire et suffisante pour qu'un processus élémentaire fermé $P = \langle R, E, \phi \rangle$ défini sur un réseau $R = \langle G, D, T \rangle$ de graphe $G = (X, U)$ soit cyclique est que le processus P', défini à partir de P sur le réseau R', obtenu en supprimant de U les arcs de $\{u \in U \mid extr(u) \in D\}$, soit un processus simple.

Démonstration :

- P cyclique ==> P' simple est une conséquence du théorème 4.3.3
- réciproquement
  P' processus simple ==>

$$\forall\ x,y \in X \text{ tel que } y = org(u),\ \forall\ c \in C_{x,y}\ \text{card}\ c < \infty$$
$$extr(u) \in D$$

  Si de plus P est fermé, alors P est cyclique.

c.q.f.d.

propriété 3.4.4.4 :

Les éléments minimaux du demi-treillis défini dans l'ensemble des sous-processus fermés d'un processus élémentaire par la relation d'inclusion sont des sous-processus élémentaires cycliques.

Démonstration : Dans le cas contraire, un processus cyclique pourrait contenir un chemin de cardinalité infini ne passant pas par D, ce qui est contraire à la définition et assure la propriété.

3.5 Décomposition d'un processus élémentaire

3.5.1 Niveau de connexité d'un sommet

On considère une application n de l'ensemble X des sommets d'un processus dans $\mathbb{N}$, qui mesure, pour chaque sommet, le nombre de sous-processus fermés qui le contient.

On appelle niveau de connexité d'un sommet, la valeur de l'application n en ce sommet.

3.5.2 Recherche des sous-processus cycliques

On se propose de mettre en évidence les sous-processus cycliques d'un processus élémentaire en utilisant les propriétés 3.3.4.3 et 3.3.4.4.

Considérons un processus élémentaire $P = \langle R, E, \phi \rangle$ défini sur un réseau de graphe $G = (X,U)$.

La fermeture transitive de la matrice booléenne représentant le graphe G, permet de mettre en évidence les composantes fortement connexes du graphe.

Le processus P étant élémentaire, ces composantes fortement connexes déterminent des sous-processus fermés.

On note $P_i = \langle R_i, E, \phi \rangle$ ces sous-processus fermés , l'indice i variant dans un ensemble I. Soit $R_i = \langle G_i, D_i, T_i \rangle$ le réseau sur lequel est défini le sous-processus fermé P_i, avec $G_i = (X_i, U_i)$.

Considérons un des sous-processus P_i ($i \in I$).

On supprime de U_i, les arcs de l'ensemble $\{u \in U_i \mid extr(u) \in D_i\}$ on a alors deux possibilités :

- ou le processus ainsi créé est simple ; alors P_i est cyclique d'après la CNS 3.4.4.3 et on peut abandonner tout examen plus approfondi de P_i

- ou le processus n'est pas simple ; il contient alors une ou plusieurs composantes fortement connexes définissant des sous-processus P_{j_i} ($j \in J_i$).

On effectue ce travail pour tous les sous-processus P_i d'indice I ; puis on itère le procédé pour tous les sous-processus fermés trouvés lors de cette étape. Cette itération s'arrêtera nécessairement car le nombre de sommets est fini et l'on aura mis en évidence une classe de sous-processus cycliques. De plus, si à chaque détection d'un sous-processus fermé on incrémente d'une unité le niveau de connexité des sommets de ce sous-processus (préalablement initialisé à zéro) on calcule par la même occasion les niveaux de connexité des sommets de R.

3.5.3 Définition

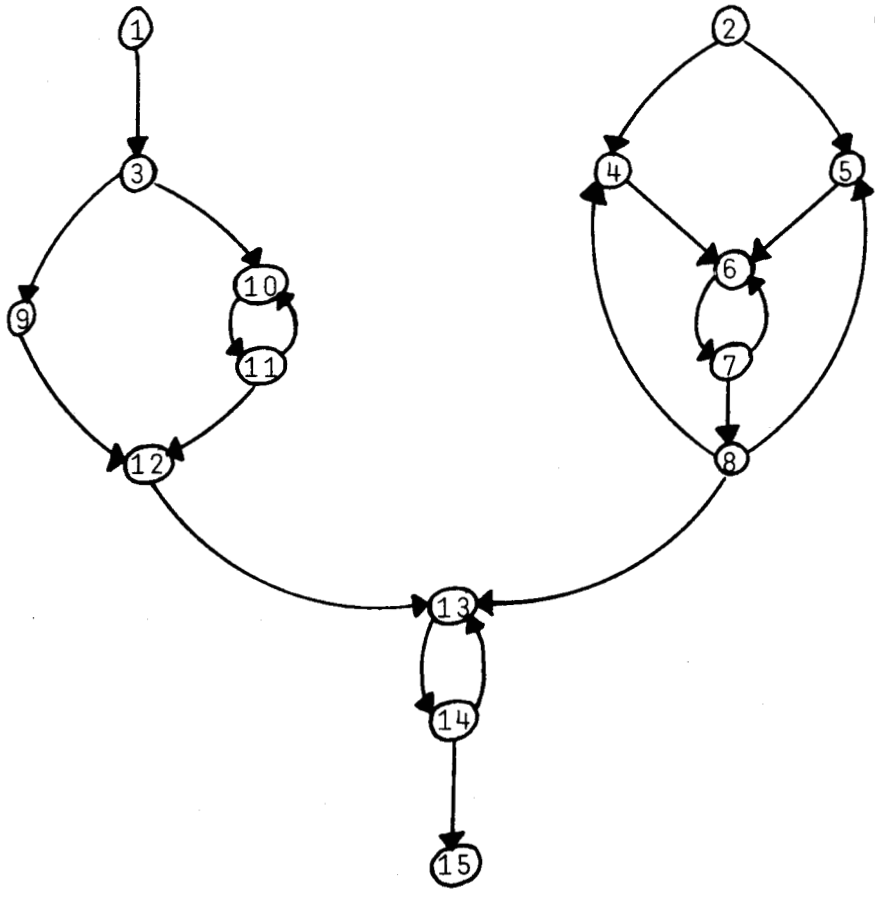
On pose, pour tout i appartenant à N :

$$Y_i = \{x \in X \mid n(x) = i\}$$

3.5.4 Exemple

Considérons un processus élémentaire quelconque P défini sur un réseau R de graphe ci-dessous :

D = {1,2}
T = {15}



La fermeture transitive de la matrice de succession associée à ce graphe met en évidence l'existence de trois composantes fortement connexes :

P_1 de sommets {4,5,6,7,8}

P_2 de sommets {10,11}

P_3 de sommets {13,14}

Pour chacun de ces sommets n(x) prend la valeur 1.

Pour chacun des sous-processus P_1, P_2, P_3 la suppression des arcs internes entrant dans D_i, suivie de la fermeture transitive des matrices correspondantes montre que P_2 et P_3 sont cycliques et que P_1 possède un sous-processus fermé P_4 de sommets {6,7}

alors on a $n(6) = n(7) = 2$.

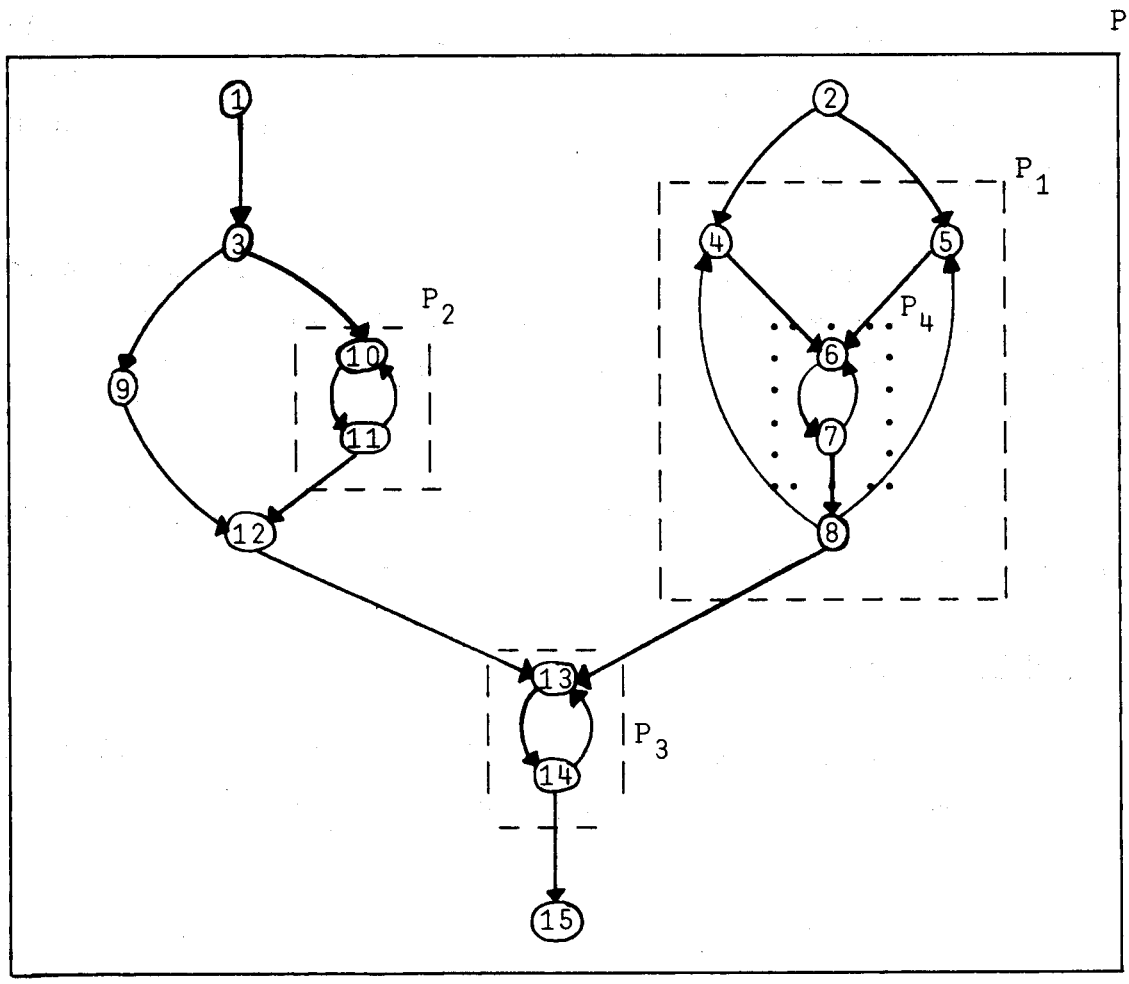
Le même procédé montre que P_4 est cyclique.

On décompose ainsi le processus P en plusieurs sous-processus fermés et les sommets en trois ensembles :

y_2 = {6,7} de niveau de connexité 2

y_1 = {4,5,8} ∪ {10,11} ∪ {13,14} de niveau de connexité 1

y_0 = {1,2,3,9,12,15} de niveau de connexité 0.



4 - DEROULEMENT D'UN PROCESSUS ELEMENTAIRE

4.1 Introduction

Considérons un processus élémentaire P = <R,$\mathcal{E}$,ϕ> défini sur un réseau R = <G,D,T> de graphe G = <X,U>.

Dans la définition d'un processus élémentaire, nous avons précisé que les tâches du processus étaient symbolisées par les sommets et que l'exécution d'une tâche consistait à passer par le sommet correspondant.

D'autre part, un ensemble de tâches peut être exécuté simultanément par passage simultané sur tous les sommets représentant ces tâches.

Enfin, l'exécution des différentes tâches est fonction de conditions extérieures représentées à l'aide des éventualités.

Nous allons définir dans ce paragraphe la notion de déroulement d'un processus pour une chaîne d'éventualités, c'est-à-dire la définition des exécutions successives d'ensembles de sommets en fonction d'éventualités.

Cette définition sera abordée progressivement en considérant tout d'abord l'exécution d'un seul sommet dans un processus simple puis en prenant en compte plusieurs sommets et des processus cycliques et fermés.

On s'aidera pour ces définitions d'un marqueur dynamique associé à chaque sommet, l'état d'un sommet .

4.2 Fonction état

On considère un processus élémentaire P = <R,$\mathcal{E}$,ϕ> défini sur un réseau R = <G,D,T> de graphe G = <X,U>.

4.2.1 Etat d'un sommet

On appelle fonction état, une application notée e de l'ensemble X dans un ensemble ξ de 3 variables booléennes telles que :

$$\forall\ u,v,w \in \xi \qquad u = \neg v \ \underline{\text{et}}\ \neg w$$

On notera passif, bloqué et actif les éléments de ξ.

4.2.2 Etat d'un sous-processus élémentaire

Soit P' un sous-processus du processus élémentaire P. On note X' l'ensemble de ses sommets et D' l'ensemble des sommets de départ.

Définition

La fonction e définie sur X peut être étendue à un sous-processus P' de la façon suivante (on conserve la même écriture de la fonction) :

$$- \ e(P') = \text{passif} \iff \begin{cases} \forall\ x \in X' & e(x) = \neg\,\text{actif} \\ \forall\ x \in D' & e(x) = \text{passif} \end{cases}$$

$$- \ e(P') = \text{bloqué} \iff \begin{cases} \forall\ x \in X' & e(x) = \neg\,\text{actif} \\ \exists\ x \in D' & e(x) = \text{bloqué} \end{cases}$$

$$- \ e(P') = \text{actif} \iff \exists\ x \in X' \quad e(x) = \text{actif.}$$

Remarque :

On vérifie que l'on a toujours

$$\text{actif} \iff \neg\,\text{bloqué}\ \underline{\text{et}}\ \neg\,\text{passif}$$

(de même en permutant les rôles respectifs des variables bloqué, passif et actif).

4.3 Déroulement d'un processus élémentaire simple

On considère un processus élémentaire simple $P = \langle R,E,\phi \rangle$ défini sur un réseau $R = \langle G,D,T \rangle$ de graphe $G = \langle X,U \rangle$.

4.3.1 Exécution d'un sommet

La fonction état du paragraphe précédant permet de représenter l'exécution d'un sommet :

Un sommet à exécuter se trouve nécessairement à l'état actif.

L'exécution de ce sommet consiste à faire passer dans un état non passif les successeurs de ce sommet. L'exécution d'un sommet s'identifie donc à une transition puisque l'exécution d'une tâche est de durée élémentaire et la transition dans un arc instantanée.

Cette exécution (ou transition) ne peut cependant avoir lieu que si l'éventualité correspondante est vraie. On parlera alors d'exécution du sommet x avec une éventualité f et on décrit l'exécution du sommet x, lorsque e(x) = actif, et le changement possible de l'état des sommets successeurs lorsqu'une éventualité f devient vraie, par la micro-primitive notée Exec(x;f) ci-dessous. Cet ensemble d'opérations est rédigé en un pseudo-algol dans lequel on conserve une écriture mathématique, notamment pour les ensembles dont on ne précise pas l'organisation ni la fonction d'accès.

Pour tout sommet y de X, on note W'_y l'ensemble des prédécesseurs de y : $W'_y = \{z \in X \mid C_{z,y} \neq \emptyset\}$ et la micro-primitive Exec(x;f) s'écrit :

Exec(x;f) : <u>début</u>
$Y := \{y \in X \mid \forall e \in E\ (f(e) = 1 \Longrightarrow \phi(x,y)(e) = 1)\}$
<u>Si</u> $Y \neq \emptyset$ <u>alors</u> (e(x) := passif ;
$\forall y \in Y$ <u>faire</u> e(y) := bloqué) ;
$\forall y \in X$ <u>tel que</u> e(y) = bloqué <u>faire</u>

```
Si ∃ z ∈ W'_y tel que e(z) = ¬ passif
        alors e(y) := bloqué
        sinon e(y) := actif
        fin
```

La dernière boucle "faire" de la micro-primitive exec(x;f) sert à distinguer les sommets qui sont à l'état "actif" de ceux qui se trouvent à l'état "bloqué", un sommet ne pouvant devenir actif que s'il ne possède pas de prédécesseur non passif. Il s'agit d'un phénomène de déblocage et on note debloq la micro-primitive suivante :

```
debloq  : début ∀ x ∈ X tel que e(x) = bloqué faire
                si ∃ z ∈ W'_x tel que e(z) = ¬ passif
                        alors e(x) := bloqué
                        sinon e(x) := prêt
          fin
```

La micro-primitive exec(x;f) s'écrit alors

```
exec(x;f) : début
            Y := {y ∈ X | ∀ e(f(e) = 1 ==> φ(x,y)(e)=1)};
            Si Y≠∅ alors (e(x):=passif; ∀ y∈Y faire
                                          e(y):=bloqué) ;
            debloq ;
            fin
```

L'exécution de cette micro-primitive associe donc à un sommet x et une éventualité f, un ensemble de sommets de X qui se trouvent tous à l'état "actif" et qui ne possèdent aucun prédécesseur qui est à l'état "¬ passif". Il est alors naturel de l'étendre à un ensemble de sommets qui se trouvent tous à l'état "actif" à un instant donné.

4.3.2 Exécution d'un ensemble de sommets pour une éventualité f

Soit X_0 un ensemble de sommets d'un processus élémentaire simple tel que : $\forall x \in X_0$ e(x) = actif.

On appelle exécution de l'ensemble X_0 avec une éventualité f, l'exécution de la micro-primitive suivante, que l'on notera $exec(X_0;f)$.

$Exec(X_0;f)$: début
$U(f):=\{u\in U \mid \forall e\ (f(e)=1 \Longrightarrow \phi(u)(e)=1)\}$;
$X_0:=X_0 \cap org(U(f))$; $Y:=\{y\in X \mid \exists x\in X_0$ tel que $(x,y)\in U\}$;
$\forall\ x \in X_0$ <u>faire</u> $e(x)$:= passif ;
$\forall\ y \in Y$ <u>faire</u> $e(y)$:= bloqué ;
debloq
<u>fin</u>

L'exécution de cette micro-primitive fait passer d'un ensemble X_0 de sommet à l'état "actif" à un instant t, à un nouvel ensemble de sommets à l'état "actif" à l'instant t+1, pour une éventualité f vraie en un point. On notera $EX(X_0,f)$ ce nouvel ensemble.

4.3.3 Déroulement d'un processus élémentaire simple pour une chaîne α

4.3.3.1 Pas associé à une éventualité f

Définition : On appelle pas associé à l'éventualité f, tout couple (Y,Z) d'ensembles de X tel que $Z = EX(Y,f)$.

Notation : $p = (Y,Z)_f$.

4.3.3.2 Déroulement associé à une chaîne α

Définition : On appelle déroulement associé à une chaîne $\alpha=f_0f_1\ldots f_n$ ($f_i \in \bar{E}\ \forall i \in \{0,1,\ldots n\}$), toute suite ordonnée de pas $p_0,p_1,\ldots,p_n$ tel que

$$\forall\ i \in \{0,1,\ldots,n\} \quad p_i = (X_i,X_{i+1})_{f_i}$$

Notation : $(p_0,p_1,\ldots,p_n)_\alpha$.

4.3.3.3 Déroulement d'un processus élémentaire simple pour une chaîne α

On considère un processus élémentaire simple $P = \langle R, \mathcal{E}, \phi \rangle$ défini sur un réseau $R = (G,D,T)$ de graphe $G = (X,U)$ et une chaîne α, $\alpha = f_0\ f_1 \ldots f_n$ d'éventualités de $\bar{\mathcal{E}}$.

On appelle déroulement du processus élémentaire simple P pour la chaîne α, tout déroulement $(p_0, p_1, \ldots, p_n)_\alpha$ tel que :

- $\forall\ i \in \{0,1,\ldots,n\}$ $\quad X_i \subset X \quad (p_i = (X_i, X_{i+1}))$
- $X_0 \subset D$
- $X_n \subset T$
- $X_{n+1} \cap X \neq \emptyset$.

Ce dernier point $X_{n+1} \cap X = \emptyset$ signifie

- si P est un processus : $X_{n+1} = \emptyset$
- si P est un sous-processus d'un processus P' : $X_{n+1} \subset X' - X$.

Dans les deux cas, après exécution de $\mathrm{Exec}(X_n, f_n)$ on a

$\forall\ x \in X \quad e(x) = \text{passif}.$

4.4 <u>Déroulement d'un processus élémentaire cyclique</u>

Dans un processus cyclique, tout sommet est successeur et prédécesseur de n'importe quel autre sommet. L'application des micro-primitives du paragraphe 4.2 est donc impossible car les sommets resteraient à l'état bloqué. Comme d'après la condition nécessaire et suffisante 3.3.4.3, il suffit de supprimer les arcs entrant en D pour obtenir un processus simple, les micro-primitives précédentes pouvant être encore valables sous réserve de ne prendre en compte que les chemins ne passant pas par D.

On pose $W_y(P) = \{z \in X \mid \exists\ c \in C_{z,y}$ tel que $\forall\ u \in c$ $\mathrm{extr}(u) \notin D\}$.

On définit une micro-primitive debloq à partir de la précédente en remplaçant W'_y par W_y

$debloq_P(X)$: <u>début</u>
$\forall\ y \in X$ <u>tel que</u> $e(y)$ = bloqué <u>faire</u>
<u>Si</u> $\exists_z \in W_y(P)$ <u>tel que</u> $e(z) = \neg$ passif
<u>alors</u> $e(y)$:= actif
<u>fin</u>

4.4.1 Exécution d'un ensemble de sommets pour une éventualité

Soit X_0 un sous-ensemble de X tel que : $\forall\ x \in X_0\ e(x)$ = actif. Comme dans le paragraphe 4.2.2 l'exécution de cet ensemble de sommets pour une éventualité f peut être décrite par la micro-primitive suivante que l'on note $Exec(X_0;f)$.

Exec $(X_0;f)$: <u>début</u>
$Y := \{y \in X \mid \exists\ x \in X_0$ tel que $(x,y) \in U$ et $f \rightarrow \phi(x,y)\}$
$\forall\ x \in X_0$ <u>faire</u> $e(x)$:= passif ;
$\forall\ y \in Y$ <u>faire</u> $e(y)$:= bloqué ;
$Z = \{x \in X \mid e(x) = \neg$passif$\}$
<u>Si</u> $Z \neq \emptyset$ <u>alors</u>
<u>début</u>
<u>Si</u> $Z \subset D$ <u>alors</u> $debloq_P(D)$
<u>sinon</u> $debloq_P(X/D)$
<u>fin</u>
<u>fin</u>

Remarque :

Si le processus élémentaire est simple, on a

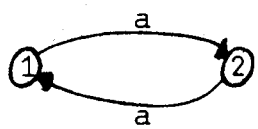
. $D \cap Y = \emptyset$
. $W_y / W'_y = \emptyset$.

En conséquence la micro-primitive du paragraphe 4.2 est un cas particulier de celle-ci (ce qui justifie l'emploi de la même notation).

On considèrera dans la suite que la micro-primitive Exec(X_0;f) décrite dans ce paragraphe est valable pour les processus élémentaires simples et cycliques.

Cas particulier :

Considérons le processus cyclique suivant



avec D = {1,2}
T = {1,2}

Soit X_0 = {1,2} avec e(1) = e(2) = actif.

Si l'on exécute la micro-primitive exec(X_0;a) on obtient

Y = {1,2}

puis e(1) = e(2) = bloqué

et comme $W_1 = W_2 = \emptyset$

on aboutit à e(1) = e(2) = actif.

Il n'y a pas dans un processus élémentaire d'interblocage interne de ce type.

4.4.2 Déroulement d'un processus cyclique pour une chaîne α

La définition d'un pas donné au paragraphe 4.2.3 reste valable sous réserve d'utiliser la définition du paragraphe précédent pour la micro-primitive exec et la définition du déroulement, donnée au paragraphe 4.2.3.3 dans le cas d'un processus élémentaire simple, reste inchangée pour un processus élémentaire cyclique.

4.5 Déroulement d'un processus élémentaire fermé

Pour un processus élémentaire fermé, l'exécution d'un ensemble quelconque de sommets se complique par le fait que des sommets peuvent appartenir à plusieurs composantes fortement connexes.

4.5.1 Processus élémentaires fermés de type séquentiel

On considère un processus élémentaire fermé $P = \langle R, E, \phi \rangle$ défini sur un réseau $R = \langle G, D, T \rangle$ de graphe $G = (X, U)$.

On note $P_1, P_2, \ldots, P_n$ les sous-processus élémentaires fermés de P.

Définition : Le processus élémentaire fermé P est dit séquentiel si et seulement si : $\forall$ P_i sous-processus élémentaire fermé de P

1) $\forall\ d \in D,\ \forall\ t \in T,\ \forall\ c \in C_{d,t}$ tel que card $c < \infty$

$$\left.\begin{array}{l} \exists\ u \in c \\ \exists\ x_i \in D_i \end{array}\right\} \text{tels que } x_i = \text{org}(u) \text{ ou } x_i = \text{extr}(u)$$

2) P_i est séquentiel.

A la limite, on considèrera un processus élémentaire cyclique comme séquentiel.

Définition : Le processus élémentaire fermé P est dit séquentiel inclusif si et seulement si

- P est séquentiel
- $P_n \subset P_{n-1} \subset \ldots \subset P_1 \subset P$ (à un isomorphisme près de l'ensemble des indices).

Propriété : Soit P un processus élémentaire fermé.

P séquentiel inclusif $\Longrightarrow$ $\forall$ P_i sous-processus élémentaire fermé de P.
P_i est séquentiel inclusif.

4.5.2 Déroulement d'un processus élémentaire fermé de type séquentiel

Dans un processus élémentaire fermé de type séquentiel, il est nécessaire de préciser à chaque instant quel sous-processus fermé est concerné par le déroulement.

4.5.2.1 L'ensemble W_x

On considère un processus élémentaire fermé $P = \langle R, E, \phi \rangle$ défini sur un réseau $R = \langle G, D, T \rangle$ de graphe $G = \langle X, U \rangle$.

Soit x un sommet de X. On note P_x le sous-processus fermé minimal contenant x, c'est-à-dire le sous-processus fermé de P dont les sommets de l'ensemble de départ D_x ont même niveau de connexité que celui de x.

On définit alors l'ensemble W_x de la façon suivante :

$$W_x = \{y \in X \mid \exists\ c \in C_{x,y}(P) \text{ tel que } \forall\ u \in c\ \text{extr}(u) \notin D_x\}$$

4.5.2.2 Primitive déblocage

L'écriture de la primitive debloq(Y) reste identique à la définition du paragraphe 4.3 en utilisant la définition ci-dessus de W_x qui généralise l'ancienne.

```
debloq(Y) : début
        ∀ y ∈ Y tel que e(y) = bloqué faire
   si ∄ z ∈ W_y tel que e(z) = ¬ passif
                alors e(y) := actif
                   fin
```

L'action de déblocage des sommets se trouvant à l'état bloqué sera décrite, pour un processus élémentaire fermé séquentiel, par la primitive "déblocage" dans laquelle il est nécessaire de tenir compte des différents sous-processus imbriqués, représentés par les différents niveaux de connexité des sommets.

On rappelle (cf. § 3.4) que l'on note

$$Y_i = \{x \in X \mid n(x) = i\} \quad \forall\, i \in N$$

micro-primitive "déblocage" :

déblocage : <u>début</u>

$Z = \{x \in X \mid e(x) = \neg \text{passif}\}$;

$\forall\, i \in N$ <u>faire</u> $Z_i := Z \cap Y_i$;

$i_{max} := \max\limits_{Z_i \neq \emptyset} (i)$;

P_k := le sous-processus élémentaire fermé de P, minimal contenant $Z_{i_{max}}$;

<u>Si</u> $Z_{i_{max}} \subset D_k$ <u>alors</u> debloq (D_k)

<u>sinon</u> debloq $(X_k - D_k)$;

<u>fin</u>

4.5.2.3 Exécution d'un ensemble X_0 avec une éventualité f

Soit X_0 l'ensemble des sommets de X tel que $\forall\, x \in X_0$ $e(x)$ = actif ; l'exécution de cet ensemble de sommets pour une éventualité f est décrite par la micro-primitive suivante (notée $\text{Exec}(X_0;f)$).

Exec $(X_0;f)$: <u>début</u>

$Y := \{y \in X \mid \exists\, x \in X_0$ tel que $(x,y) \in U$ et $f \rightarrow \phi(x,y)\}$

$\forall\, x \in X_0$ <u>faire</u> $e(x)$:= passif ;

$\forall\, y \in Y$ <u>faire</u> $e(y)$:= bloqué ;

déblocage ;

<u>fin</u>

Cette définition de la micro-primitive $\text{Exec}(X_0;f)$ est valable pour un processus élémentaire cyclique, cas particulier d'un processus élémentaire fermé de type séquentiel, puisque dans

ce cas il n'existe qu'un seul niveau de connexité et donc le déblocage n'a lieu que dans le processus élémentaire cyclique.

Elle est aussi vraie pour un processus élémentaire simple, cas particulier d'un processus élémentaire cyclique, pour lequel on est assuré de ne jamais passer par l'ensemble de départ une fois qu'on l'a quitté.

4.5.2.4 Pas et déroulement

L'exécution de micro-primitive exec(X_0;f) fait passer d'un ensemble de sommets à l'état actif à un instant donné, à un autre ensemble de sommets tous à l'état actif à l'instant suivant, lorsqu'une éventualité f est vraie en un point. On appellera pas un tel couple d'ensembles et l'on note $p = (X,Y)_f$.

Soit $\alpha = f_0 \ldots f_n$ une chaîne d'éventualités f_i de $\bar{E}$; on appelle déroulement associé à la chaîne α toute suite ordonnée de pas $p_0\ p_1 \ldots p_n$ telle que

$$\forall\ i \in \{1,\ldots,n\} \quad p_i = (X_i, X_{i+1})_f$$

Notation : $\delta_\alpha = (p_0, p_1, \ldots, p_n)_\alpha$.

4.5.2.5 Déroulement d'un processus élémentaire fermé de type séquentiel

On considère un processus élémentaire fermé de type séquentiel $P = \langle R, E, \phi \rangle$ défini sur un réseau $R = (G,D,T)$ de graphe $G = (X,U)$ et une chaîne $\alpha = f_0 f_1 \ldots f_n$ d'éventualités de $\bar{E}$.

Définition : On appelle déroulement du processus élémentaire fermé de type séquentiel P (a fortiori processus cyclique ou simple) associé à la chaîne d'éventualités α, tout déroulement $\delta_\alpha = (p_0 \ldots p_n)$ tel que:

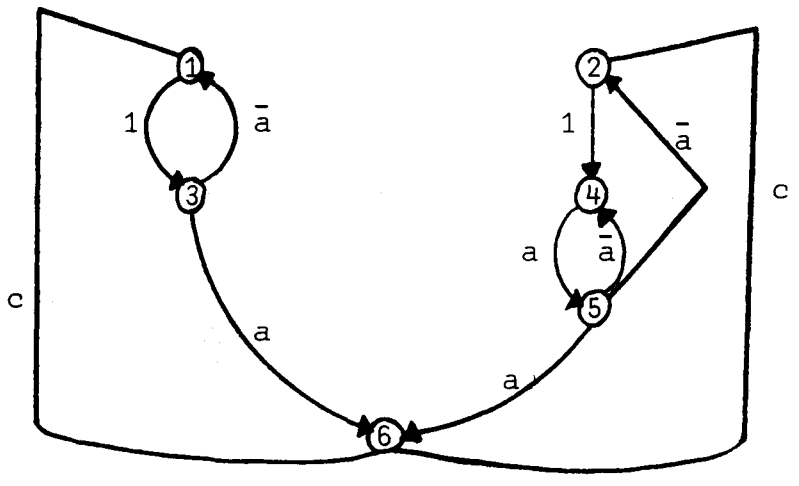
- $\forall\ i \in \{0,\ldots,n+1\}\ X_i \subset X$
- $X_0 \subset D$
- $X_n \subset T$
- $X_{n+1} \cap X = \emptyset$

4.5.3 Déroulement d'un processus élémentaire

Dans le déroulement des processus élémentaires décrit précédemment, il n'y a pas de simultanéité à l'exécution entre des sommets appartenant à un sous-processus élémentaire fermé et des sommets qui n'appartiennent pas à ce sous-processus. A chaque exécution, les sommets exécutés ont tous un même niveau de connexité et appartiennent tous au même sous-processus élémentaire fermé, minimal pour chacun d'eux.

Dans le cas d'un processus élémentaire fermé de type non séquentiel et, a fortiori, dans le cas d'un processus élémentaire quelconque, ces deux propriétés peuvent être mises en défaut.

Ainsi dans l'exemple ci-après d'un processus élémentaire fermé de type non séquentiel, l'ensemble $X_0 = \{1,2\}$ est composé de deux sommets de même niveau de connexité égal à 2 mais qui n'appartiennent pas au même sous-processus élémentaire fermé, minimal pour chacun d'eux. De même, l'ensemble $X_1 = \{3,4\}$ est composé des sommets 3 et 4 de niveaux de connexité respectifs 2 et 3.



$D = \{1,2\}$

$T = \{6\}$

$Y_0 = \emptyset$

$Y_1 = \{6\}$

$Y_2 = \{1,2,3\}$

$Y_3 = \{4,5\}$

processus élémentaire fermé de type non séquentiel

Les différents sous-processus élémentaires fermés contenus dans un processus élémentaire fermé de type non séquentiel peuvent être exécutés séparément ou non.

Dans certains cas (en particulier lorsque les réseaux sont isomorphes et les chaînes d'éventualités identiques) il est très avantageux d'effectuer un déroulement simultanée de processus. Dans d'autres cas, ces déroulements simultanés sont possibles mais on risque, ignorant lequel sera terminé en premier, de multiplier les regroupements de sommets.

Dans d'autres occasions, il est préférable de dérouler ces sous-processus en série mais le choix de l'ordonnancement dans le réseau contracté suppose des critères précis.

Enfin, ces sous-processus peuvent être toujours exclusifs l'un de l'autre et il est intéressant, dans ce cas, de le distinguer au départ.

C H A P I T R E 4

CONSTRUCTION DES DEROULEMENTS PRINCIPAUX D'UN PROCESSUS ELEMENTAIRE

IV.1. Introduction.

IV.2. Contexte de processus.

IV.3. Théorème fondamental.

IV.4. Propriétés du f.b.g. du théorème fondamental.

IV.5. Conclusion.

1 - INTRODUCTION

Au chapitre précédent, nous avons défini le déroulement d'un processus élémentaire simple ou fermé de type séquentiel pour une chaîne d'éventualités donnée. Ce déroulement se caractérise par un enchaînement de procédures EXEC.

Cependant, un même ensemble de sommets peut être traité à l'aide de la procédure EXEC, pour la même fonction booléenne, mais lors de déroulements différents et il peut se faire, dans cette hypothèse, que le résultat soit fonction du déroulement.

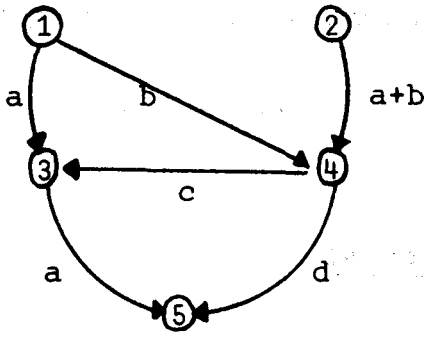
Plus précisément, lors de la phase de déblocage de la procédure EXEC, l'ensemble des sommets à prendre en compte peut être différent selon le déroulement.

Ainsi si l'on considère le processus ci-dessous, aux chaînes $\alpha_1 = b \circ \bar{c}d$ et $\alpha_2 = a \circ \bar{c}d \circ 1$, on peut associer les déroulements respectifs suivants :

$$\delta_{\alpha_1} = (\{\{1,2\},\{4\}\}_b, (\{4\},\{5\})_{\bar{c}d})$$

$$\delta_{\alpha_2} = (\{\{1,2\},\{4\}\}_a, (\{4\},\{3\})_{\bar{c}d}, (\{3\},\{5\})_1)$$

Dans δ_{α_1} et δ_{α_2}, exec $(\{4\};\bar{c}d)$ définit des pas différents.



D = {1,2}

T = {5}

Il y a là une ambiguïté puisque l'exécution d'un même sommet pour une même éventualité conduit à des ensembles différents. Au niveau de la micro-programmation, cela signifie que l'exécution d'une même micro-instruction, fournissant les mêmes conditions d'enchaînement,

peut donner des micro-instructions successeurs différentes.

Il est donc nécessaire de lever cette ambiguïté en considérant deux occurences différentes d'un même ensemble de sommets (de la même micro-instruction).

Pour cela nous utiliserons la notion de contexte de processus, puis nous donnerons un algorithme permettant de générer un flot booléen généralisé semi-homomorphe du flot définissant le processus et construit à partir d'un recouvrement par des contextes, et enfin nous examinerons les propriétés de ce f.b.g.

2 - CONTEXTE DE PROCESSUS

2.1 Définition

On considère un processus élémentaire $P = \langle R, E, \phi \rangle$. Chacun des sommets du réseau peut se trouver dans l'un des 3 états, passif, bloqué ou actif.

On appelle contexte d'un processus à un instant donné, l'ensemble des états de tous les sommets du processus.

En fait, compte tenu de la relation logique qui lie les états des sommets, on peut se borner à ne définir un contexte que par le couple d'ensembles de sommets qui sont respectivement en l'état actif ou bloqué

$$C = (A,B) \text{ avec } A = \{x \in X \mid e(x) = \text{actif}\}$$
$$B = \{x \in X \mid e(x) = \text{bloqué}\}.$$

2.2 Contexte de déroulement

Dans ce qui suit, on ne considère plus que les processus élémentaires simples ou fermés de type séquentiel, seuls processus pour lesquels un déroulement est défini.

2.2.1. Définitions

Un contexte $C = (A,B)$ du processus élémentaire P est dit contexte de déroulement s'il existe un déroulement δ_α de ce processus et un pas $(X,A)_f$ de ce déroulement tel que, à l'issue de l'exécution de la procédure exec(X;f), A soit l'ensemble des sommets en l'état actif et B l'ensemble des sommets en l'état bloqué.

On parlera alors d'exécution d'un contexte Exec(C;g) où $C = (A,B)$:

Exec (C;g) : <u>début</u>

$Y := \{y \in X \mid \exists\, x \in A$ <u>tel que</u> $(x,y) \in U$ <u>et</u>
$\forall\, e(g(e) = 1 \Longrightarrow \phi(x,y)(e)=1)\}$

$\forall\, x \in A$ e(x) := passif ;

$\forall\, y \in Y$ e(y) := bloqué ;

déblocage (avec $Z := B \cup Y$)

<u>fin</u>

2.2.2. Propriétés des contextes de déroulement

L'ensemble des propriétés ci-après est une conséquence de la régulation du processus P à l'aide de la procédure exec.

Propriété 2.2.2.1.

Soit $C = (A,B)$ un contexte de déroulement du processus P.

a) $\forall\, y \in B \quad \exists\, x \in A$ tel que $y \in W_x$
où W_x est l'ensemble défini au paragraphe III 4.5.2.1.

b) $\forall\, x,y \in A \quad x \notin W_y$.

c) $\forall\, x,y \in A \quad n(x) = n(y)$.

Propriété 2.2.2.2.

Soient $C_1 = (A_1,B_1)$ et $C_2 = (A_2,B_2)$ deux contextes de déroulement du processus P.

Si $A_1 \subset A_2$ alors $A_2 \cap B_1 = \emptyset$ et $A_1 \cap B_2 = \emptyset$

Si $B_1 \subset B_2$ alors $A_2 \cap B_1 = \emptyset$

Démonstration :

Soit $A_1 \subset A_2$: si $A_2 \cap B_1$ était non vide, on pourrait trouver un sommet x de $B_1 \cap A_2$ et un sommet y de A_1, donc appartenant à A_2, tel que $x \in W_y$, ce qui contredit la propriété 2.2.1.b.

D'autre part $A_1 \cap B_2 = \emptyset$ car $A_2 \cap b_2 = \emptyset$.

Enfin si $B_1 \subset B_2$, $A_2 \cap B_1 = \emptyset$ car $A_2 \cap B_2 = \emptyset$.

Propriété 2.2.2.3.

Soient $c_1 = (A_1,B_1)$ et $c_2 = (A_2,B_2)$ deux contextes de déroulement distincts du processus P, alors $A_1 \cup B_1 \neq A_2 \cup B_2$.

Démonstration :

Supposons que $A_1 \cup B_1 = A_2 \cup B_2$.

1er cas : si $A_1 \subset A_2$, $A_2 \cap B_1 = \emptyset$ d'après 2.2.2.
et $B_1 \subset B_2$ d'après l'hypothèse .

. soit $y \in B_2 - B_1$: l'hypothèse implique $y \in A_1$
ce qui est impossible car $A_1 \cap B_2 = \emptyset$
donc $B_2 = B_1$

. soit $x \in A_2 - A_1$: l'hypothèse implique $x \in B_1$ et
donc $x \in B_2$
ce qui est impossible car $A_2 \cap B_2 = \emptyset$
et donc $A_2 = A_1$.

En conséquence si $A_1 \subset A_2$ alors $c_1 = c_2$.

2ème cas : si $A_1 \cap A_2 = \emptyset$

alors $C_1 = (A,B)$ et $C_2 = (B,A)$
ce qui implique que C détermine un processus cyclique
tel que $\forall x \in A \ \exists y \in B \ (x,y) \in U$
$\forall x \in B \ \exists y \in A \ (x,y) \in U$
$U \cap A \times A = \emptyset \qquad U \cap B \times B = \emptyset$

On a alors $A \cup B = D$ = ensemble des éléments d'entrée dans le processus et on ne peut avoir une partie des sommets de D en l'état actif et l'autre en l'état bloqué (résulte de la procédure exec).

En conséquence $A_1 \cap A_2 \neq \emptyset$.

3ème cas : $C_1 = (A_1 \cup A, A_2)$

$C_2 = (A_2 \cup A, A_1)$ avec $\begin{cases} A_1 \cap A_2 = \emptyset \\ A \neq \emptyset \end{cases}$

$C_1 = (A_1 \cup A, A_2) \Longrightarrow \forall\, x \in A_1, \nexists\, y \in A \quad x \in W_y$

$C_2 = (A_2 \cup A, A_1) \Longrightarrow \forall\, z \in A_2, \nexists\, y \in A \quad z \in W_y$

En conséquence $\forall\, x \in A_1 \;\exists\, y \in A_2$ tel que $y \in W_x$ et inversement. Ce qui implique que $A_1 \cup A_2$ détermine un processus cyclique ce qui est impossible (cf.cas 2).

En conséquence si deux contextes de déroulement distincts $C_1 = (A_1,B_1)$ et $C_2 = (A_2,B_2)$ d'un processus élémentaire fermé de type séquentiel sont toujours tels que $A_1 \cup B_1 \neq A_2 \cup B_2$, un contexte de déroulement est entièrement défini par un ensemble de sommets qui se trouvent tous en état non passif. Tout contexte de déroulement $C = (A,B)$ est défini par la donnée de l'ensemble AUB et de cet ensemble, on pourra toujours déduire les ensembles A et B en examinant l'existence d'un antécédant dans le sous-graphe de G de sommets AUB.

3 - THEOREME FONDAMENTAL

3.1 Théorème

Soit $P = \langle R,E,\phi \rangle$ un processus élémentaire simple ou fermé de type séquentiel. Il existe un recouvrement de ce processus par des contextes de déroulement sur lequel est défini un flot booléen généralisé semi-homomorphe du flot ϕ.

Démonstration :

A. Dans cette première partie de la démonstration, nous allons construire simultanément le recouvrement du processus et le flot booléen généralisé.

Supposons que l'on ait déjà mis en évidence un certain nombre de contextes de déroulement $c_o, c_1, \ldots, c_i, \ldots, c_j$.

On s'intéresse au contexte $C_i = (A_i, B_i)$ $(i \leq j)$ après avoir déjà examiné les contextes $c_o, c_1 \ldots c_{i-1}$.

Initialement on pose $C_0 = (D, \emptyset)$ (D = ensemble de départ du processus).

On note X_i l'ensemble des successeurs possibles de A_i

$$X_i = \{y \in X \mid \exists\, x \in A_i \text{ tel que } (x,y) \in U\}.$$

1/ Pour toute éventualité $\phi(u)$, telle que $org(u) \in A_i$, on écrit $\phi(u)$ sous la forme d'une somme d'implicants premiers comme dans la démonstration du théorème II.3.9.

On note H_i l'ensemble de tous ces implicants pour toutes les éventualités de l'ensemble $\{\phi(u) \mid u \in (A_i \times X_i) \cap U\}$.

2/ Pour toutes les fonctions f de H_i, on exécute la procédure Exec $(C_i;f)$.

On note $c'_1, c'_2, \ldots, c'_m$ les contextes <u>distincts</u> obtenus à l'issue de l'exécution des procédures exec $(C_i;f)$ et on pose

$$\forall\, k \in \{1,\ldots,m\} \quad \mathcal{O}(c'_k) = \{f \in H_i \mid c'_k = \text{exec } (C_i;f)\}$$

$$\nu(c'_k) = \sum_{f \in \mathcal{O}(c'_k)} f.$$

3/ Soient $c_o, c_1 \ldots c_i \ldots c_j$ les contextes déjà mis en évidence. Parmi les contextes $c'_1, c'_2, \ldots, c'_m$ obtenus, on note

$c_{j+1}, c_{j+2}, \ldots, c_{j+m_j}$ les contextes c'_k qui n'apparaissent pas déjà dans la liste $c_o, \ldots, c_j$ et $\forall\ h \in \{0,1,\ldots,j+m_j\}$ on pose :

si $\exists\ \ell \in \{1,2,\ldots,m\}$ tel que $c_h = c'_\ell$ alors

$$U_{i,h} = (c_i, c_h) \text{ et } \Phi(U_{i,h}) = \nu(c'_\ell)$$

4/ On a alors établi les différents contextes c_k successeurs de c_i et pour chacun d'eux on a établi une fonction booléenne $\phi(c_i, c_k)$.

On itère alors le procédé avec c_{i+1}, c_{i+2}, ... tant qu'il y a des contextes à examiner (qui sont nécessairement en nombre fini).

A l'issue du traitement on aboutit à un ensemble de contextes $\mathcal{C} = \{c_o, c_i, \ldots, c_n\}$ et à un ensemble de couples $\mathcal{U} = \{U_{i,h} = (c_i, c_h)\}$ auxquels sont associées les fonctions booléennes non identiquement nulles $\Phi(U_{i,h})$.

B. Vérifions que l'application Φ de $\mathcal{U}$ dans $\bar{E}$ est un f.b.g. sur le graphe $\mathcal{G} = (\mathcal{C}, \mathcal{U})$ qui est semi-homomorphe du flot ϕ définissant le processus.

Soient c_i et c_j, deux contextes et e un élément de $\bar{E}$ tels que $\Phi(c_i, c_j)(e) = 1$.

$$\phi(c_i, c_j)(e) = 1 \Longrightarrow \exists\ f \in \mathcal{O}(c_j) \text{ telle que } f(e) = 1.$$

Soit x un sommet de C_j

- ou x se trouvait déjà en l'état bloqué dans C_i :
  $x \in C_j \cap B_i$
- ou x vient de passer en l'état bloqué et ne s'y trouvait pas : $(x \in B_j - B_i)$. Alors, de la définition de la procédure exec $(C_i; f)$ on déduit qu'il existe un sommet y de A_i tel que :

- $(y,x) \in U$

- $f(e) = 1 \Longrightarrow \phi(y,x)(e) = 1$

- ou x vient de passer en l'état actif ($x \in A_j - B_i$).
  Alors, comme précédemment, on déduit de la définition de la procédure exec qu'il existe un sommet y de A_i tel que

  - $(y,x) \in U$

  - $f(e) = 1 \Longrightarrow \phi(y,x)(e) = 1$

En conséquence, si $\Phi(C_i,C_j)(e) = 1$,

$\forall\ x \in C_j - B_i \quad \exists\ y \in C_i$ tel que $(y,x) \in U$ et $\phi(y,x)(e) = 1$

et si cette propriété est vraie pour tout x de $C_j - B_i$ elle est à fortiori vraie pour tout x de $D(C_j) - C_i$ et donc Φ est semi-homomorphe de ϕ.

C. Vérifions que le recouvrement de contextes est constitué de contextes de déroulement ou, plus généralement, vérifions la propriété suivante :

Propriété : A tout chemin dans G menant de $C_o = (D,\emptyset)$ à un contexte $C' = (A',B')$ avec $A' \subset T$ et $B' = \emptyset$, on peut associer un déroulement du processus P.

Démonstration :

Considérons un chemin $u_o,\ldots,u_n$ de G avec

$$\forall\ i \in \{0,\ldots,n\}\ u_i = (c_i,c_{i+1}) \text{ et } c_i = (A_i,B_i)$$

$\forall\ i \in \{0,\ldots,n\}$ considérons une fonction f_i, vraie en un seul point, de $\mathcal{O}(C_{i+1})$.

Alors C_{i+1} = exec $(C_i;f_i)$ et la suite $A_0 \ldots A_n$ constitue un déroulement du processus pour la chaîne $f_0 \ldots f_n$.

3.2 Algorithme de construction

La démonstration du théorème précédent permet de construire le f.b.g. cherché en décomposant systématiquement toute éventualité sous la forme d'une somme de ses implicants premiers. Cette décomposition peut être longue et coûteuse en occupation mémoire si le nombre de conditions de E est important. Comme au paragraphe II.3.10, on peut se borner à n'examiner que les combinaisons d'implicants des éventualités qui permettent de passer d'un A_i à un X_i. Nous allons détailler ci-après un algorithme de construction.

Soit $C = \{c_o, \ldots, c_{j_{max}}\}$ les contextes déjà mis en évidence.

On s'intéresse au contexte $C_i = (A_i, B_i)$ $(i \leq j_{max})$ en supposant que les contextes $c_o \ldots c_{i-1}$ ont déjà été examinés.

Initialement, pour i=0, $C_i = (D, \emptyset)$.

Au contexte C_i on fait correspondre l'ensemble X_i

$$X_i = \{y \in X \mid \exists\, x \in A_i \text{ tel que } (x,y) \in U\}.$$

De façon analogue au début de la démonstration II.3.10, on note

$$F_i = \{\phi(u) \mid u \in (A_i \times X_i) \cap U\}$$

et pour toute fonction f de F_i, on écrit f sous la forme d'une somme de monômes $f = \Sigma\, f^k$. On note $\bar{\bar{F}}_i(f)$ l'ensemble de ces monômes.

On pose $\bar{F}_i(f) = \{f^k \in \bar{\bar{F}}_i(f) \mid \nexists\, f^\ell \in \bar{\bar{F}}_i(f) \text{ telle que } f^\ell \to f^k\}$

$$\text{et } \bar{F}_i = \bigcup_{f \in F_i} \bar{F}_i(f)$$

Notations : On considère les sous-ensembles de fonctions de $\bar{F}_i$.

$\forall\, F' \in \mathcal{P}(\bar{F}_i)$ on pose :

$$X_i(F') = \bigcup_{f \in F'} \{y \in X_i \mid \exists\, x \in A_i \text{ tel que } \forall\, e\ (f(e) = 1 \Longrightarrow \phi(x,y)(e) = 1)\}$$

$$H(F') = F' \cup \{\bar{g} \mid X_i(g) \cap (X_i - X_i(F')) \neq \emptyset\}$$

$$\psi(F') = \prod_{f \in H(F')} f.$$

L'organigramme se décompose en 2 phases :

Pour le contexte C_i en cours d'examen, on élabore, dans la phase 1, l'ensemble de fonctions $\bar{F}_i$ en repérant celles qui sont exclusives.

Dans la phase 2, on examine les recouvrements de sommets de X_i que l'on peut déduire de chacun des éléments de $\mathcal{P}(\bar{F}_i)$.

Pour chacun des contextes mis en évidence, on s'aidera des suites de terme général C_i, $\mu(C_i)$, $\nu(C_i)$ et $m(C_i)$ $(i=0,1,\ldots,j_{max})$:

- la suite (C_k) représente les contextes déjà mis en évidence à un moment donné

- la suite $(\mu(C_k))$ est l'ensemble des fonctions de $F' \in \mathcal{P}(F_i)$ tel que
  - . $C_k = \text{EXEC}\ (C_i, \psi(F'))$
  - . F' conjonctif
  - . $\forall\ F' \in \mu(C_k),\ \forall\ F'' \in \mathcal{P}(\bar{F}_i)$

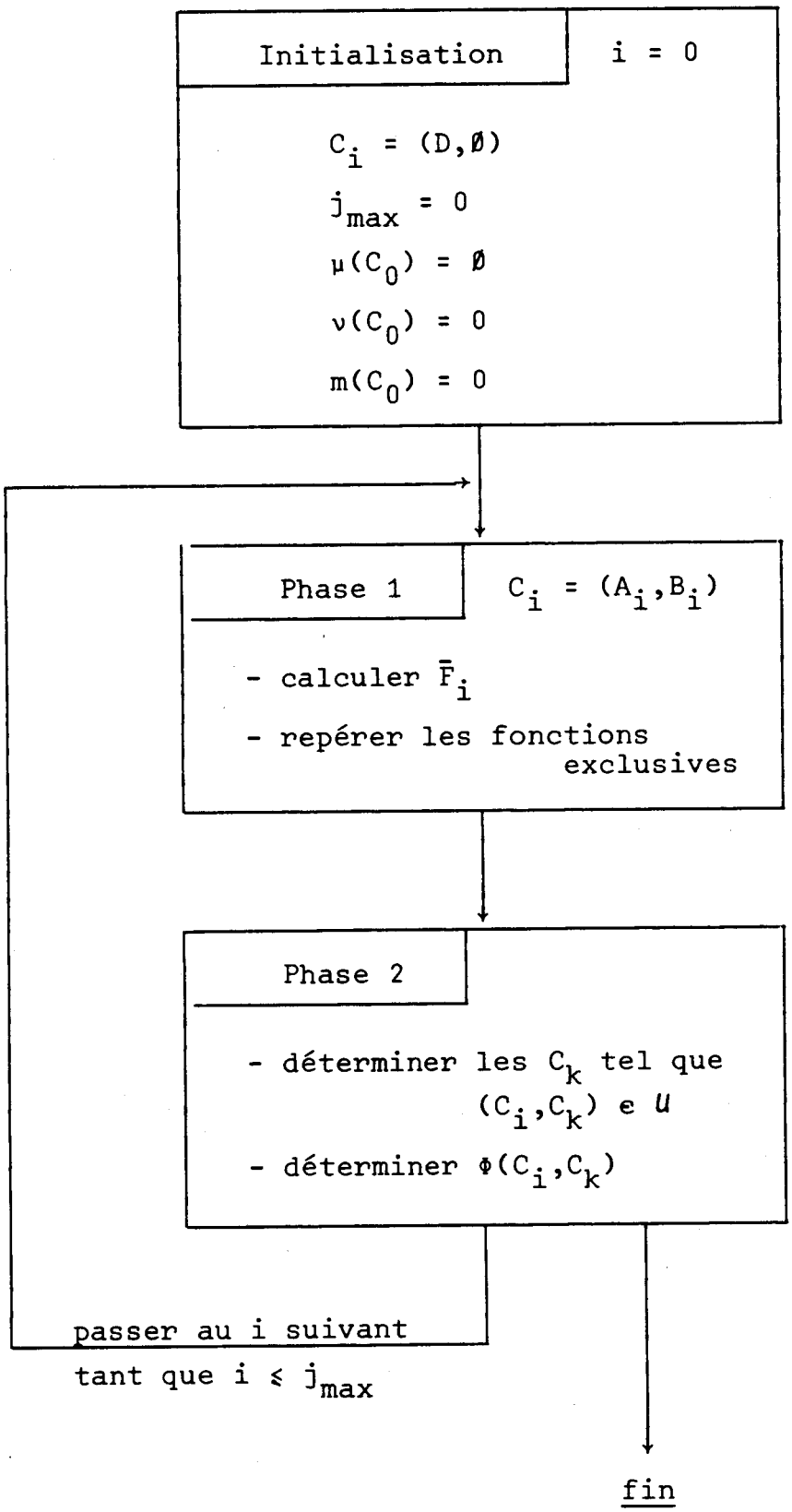
    $$F' \subset F'' \Longrightarrow F'' \notin \mu(C_k)$$

- $\nu(C_k) = \{\psi(F') \mid F' \in \mu(C_k)\}$ sert à élaborer l'éventualité généralisée $\phi(C_i, C_k)$.

- $m(C_k)$ est un marqueur qui repère l'existence d'une transition entre C_i et C_k.

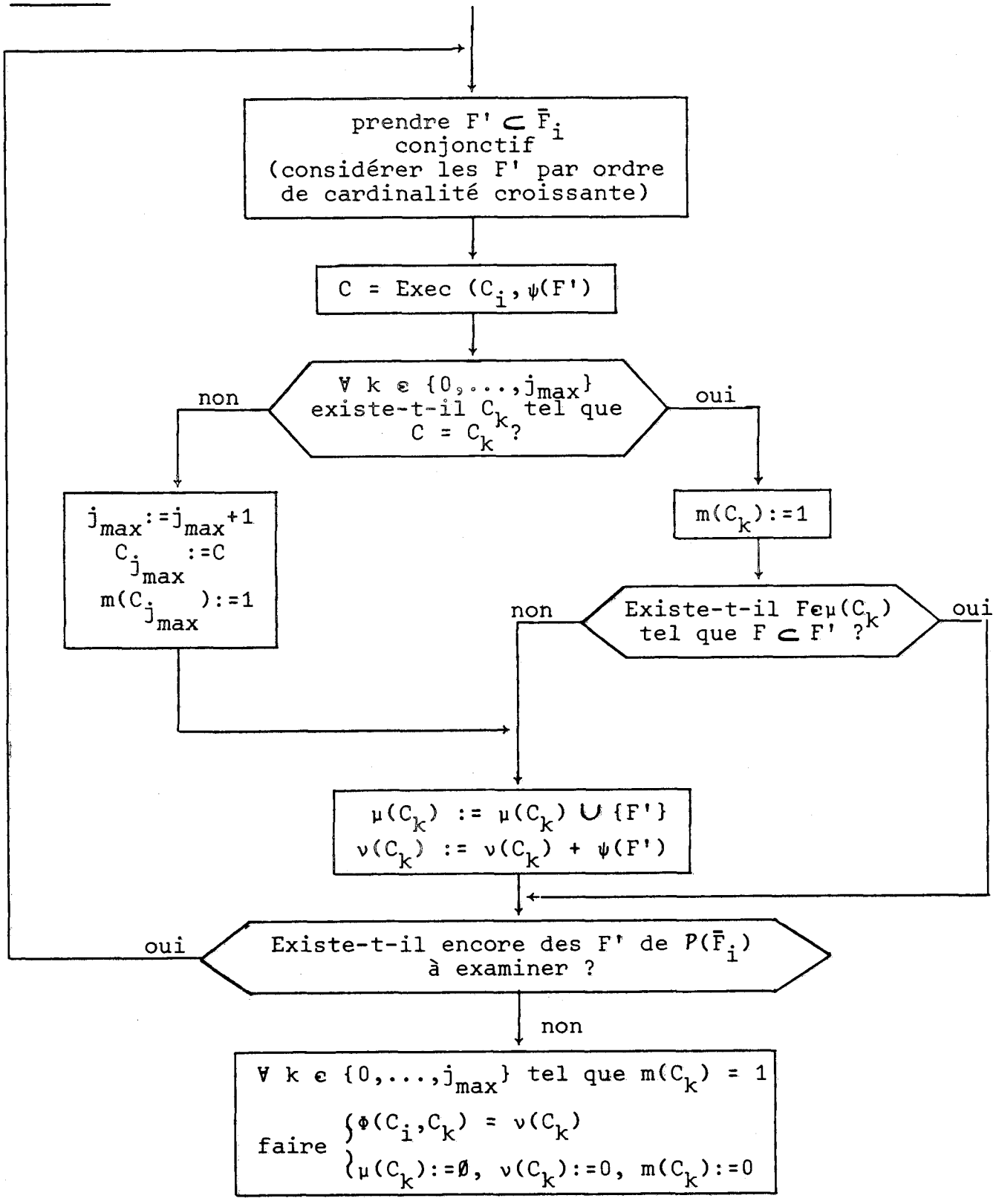
L'organigramme général est donné ci-après, puis nous détaillons le déroulement de la phase 2.

On prendra garde dans cette phase de ne pas oublier d'examiner l'ensemble $F' = \emptyset$ si $\bar{F}_i$ est conjonctif <u>et</u> $B_i \neq 0$.


Initialisation
i = 0
$C_i = (D,\emptyset)$
$j_{max} = 0$
$\mu(C_0) = \emptyset$
$\nu(C_0) = 0$
$m(C_0) = 0$
Phase 1
$C_i = (A_i,B_i)$
- calculer $\bar{F}_i$
- repérer les fonctions exclusives
Phase 2
- déterminer les C_k tel que $(C_i,C_k) \in \mathcal{U}$
- déterminer $\Phi(C_i,C_k)$
passer au i suivant
tant que $i \leq j_{max}$
fin

BUS LILLE

Phase 2

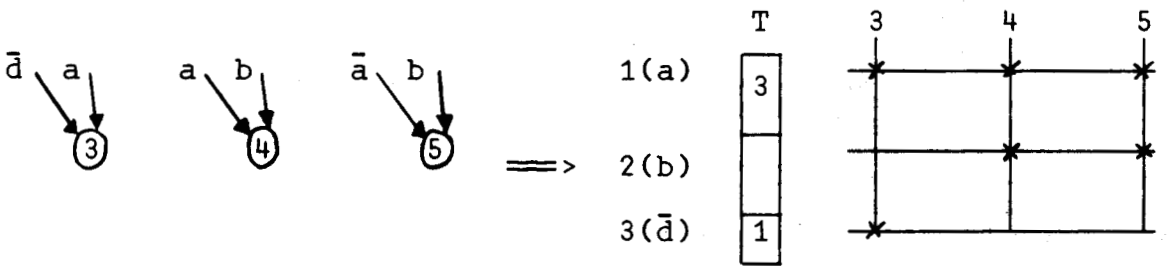


BCS ULB

Détails de la phase 2.

Les différents manipulations et calculs de la phase 2 sont grandement facilités si, pour chaque $\bar{F}_i$, on s'aide d'un tableau indicé en ligne par les fonctions de $\bar{F}_i$ et en colonnes par les sommets de X_i.

Exemple.



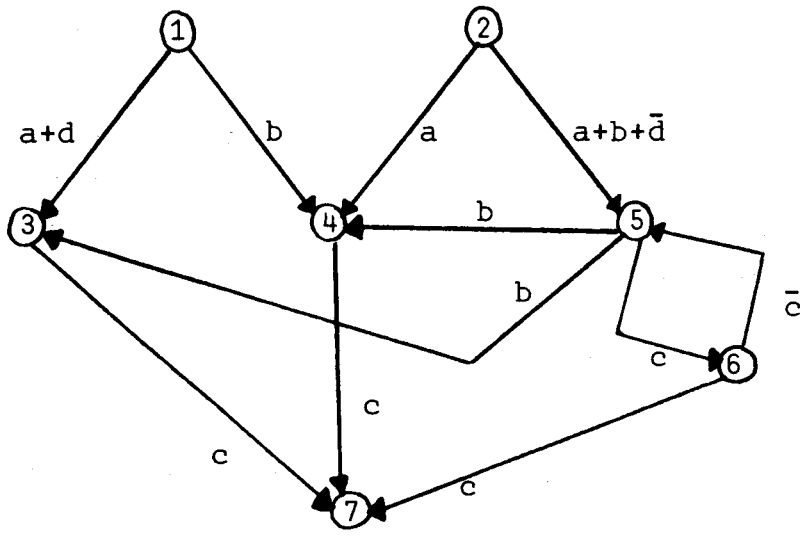
Les différents ensembles F' sont créés par bouclage sur les indices des lignes, en éliminant à l'aide de T les ensembles non conjonctifs.

Par consultation de la matrice, $X_i(F')$ est immédiatement déterminé.

Les fonctions de $\bar{H}(F') = \{\bar{g} \mid X_i(g) \cap (X_i - X(F')) \neq \emptyset\}$ apparaissent en consultant les colonnes du tableau qui n'appartiennent pas à $X_i(F')$ et en complémentant les fonctions correspondant aux lignes pour lesquelles on trouve une croix à l'intersection.

3.3 Exemple

Par cet exemple, nous allons illustrer l'algorithme précédent. Considérons le processus suivant :

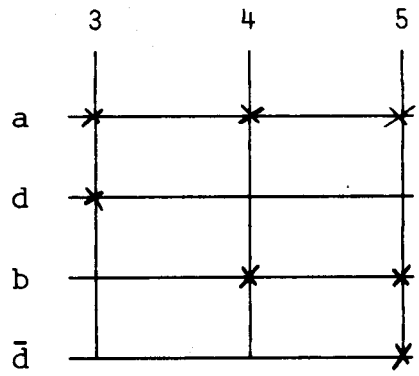


D = {1,2}
T = {7}

1) $C_0 = (\{1,2\}, \emptyset) \Longrightarrow X_i = \{3,4,5\}$

$\bar{F}_i = \{a,d,b,\bar{d}\}$

0	1
4	2
0	3
0	4



k	C_k	$\mu(C_k)$	$\nu(C_k)$
0	1,2 / ∅		
1	5 / 3,4	$\{a\} \cup \{bd\}$	$a + bd$
2	3 / ∅	$\{d\}$	$\bar{a}\bar{b}d$
3	5 / ∅	$\{b\}$	$\bar{a}b\bar{d}$
4	5 / ∅	$\{\bar{d}\}$	$\bar{a}\bar{b}\bar{d}$

$\{d,\bar{d}\}$, $\{d,b,\bar{d}\}$, $\{a,d,\bar{d}\}$, $\{a,d,b,\bar{d}\}$ ne sont pas pris en compte car ils sont non conjonctifs.

BUS LILLE

$F' = \{a,d,b\}$, $\{a,d\}$, $\{a,b\}$, $\{a,\bar{d}\}$, $\{a,b,\bar{d}\}$ ne donne rien car
$X_i(F') = C_1$ et $\{a\} \subset F'$

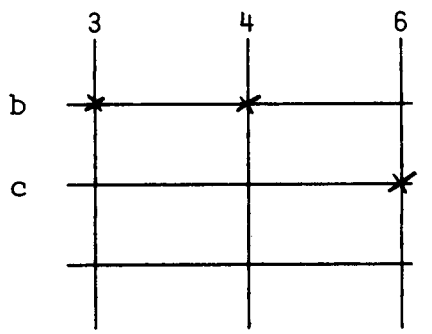
$F' = \{b,\bar{d}\}$ ne donne rien car $X_i(F') = C_3$ et $\{b\} \subset \{b,\bar{d}\}$.

On a donc

$$\Phi(C_0,C_1) = a + bd$$
$$\Phi(C_0,C_2) = \bar{a}\bar{b}d$$
$$\Phi(C_0,C_3) = \bar{a}b\bar{d}$$
$$\Phi(C_0,C_4) = \bar{a}\bar{b}\bar{d}.$$

2) Recommençons avec $C_1 = (\{5\},\{3,4\})$

$\bar{F}_i = \{b,c\}$



C_k	$\mu(C_k)$	$\nu(C_k)$
$C_5 = (\{3,4\} / \emptyset)$	$\{b\} \cup \emptyset$	$b\bar{c} + \bar{b}\bar{c} = \bar{c}$
$C_6 = (\{6\} / \{3,4\})$	$\{c\} \cup \{b,c\}$	$\bar{b}c + bc = c$

$\Longrightarrow \quad \Phi(C_1,C_5) = \bar{c}$

$\Phi(C_1,C_6) = c$

3) $C_2 = (\{3\},\emptyset)$

$C_7 = (\{7\} / \emptyset) \qquad \Phi(C_3,C_7) = C$

4) $C_3 = (\{5\}, \{4\})$

C_K	$\mu(C_k)$	$\nu(C_k)$
$C_5 = (\{3,4\} / \emptyset)$	$\{b\}$	$b\bar{c} = \Phi(C_3,C_5)$
$C_8 = (\{6\} / \{4\})$	$\{c\}$	$\bar{b}c = \Phi(C_3,C_8)$
$C_6 = (\{6\} / \{3,4\})$	$\{b,c\}$	$bc = \Phi(C_3,C_6)$
$C_9 = (\{4\} / \emptyset)$	$\emptyset$	$\bar{b}\bar{c} = \Phi(C_3,C_9)$

5) $C_4 = (\{5\}, \emptyset)$

C_k	$\mu(C_k)$	$\nu(C_k)$
$C_{10} = (\{6\},\emptyset)$	$\bar{b}c$	$\bar{b}c = \Phi(C_4,C_{10})$
$C_5 = (\{3,4\},\emptyset)$	$b\bar{c}$	$b\bar{c} = \Phi(C_4,C_5)$
$C_6 = (\{6\}, \{3,4\})$	bc	$bc = \Phi(C_4,C_6)$

6) $C_5 = (\{3,4\},\emptyset)$ $\Phi(C_5,C_7) = c$

7) $C_6 = (\{6\}, \{3,4\})$

C_k	$\mu(C_k)$	$\nu(C_k)$
$C_1 = (\{5\}, \{3,4\})$	$\{\bar{c}\}$	$\bar{c} = \Phi(C_6,C_1)$
$C_{11} = (\{3,4\}, \{7\})$	$\{c\}$	$c = \Phi(C_6,C_{11})$

8) $C_7 = (\{7\},\emptyset)$

9) $C_8 = (\{6\} / \{4\})$

C_k	$\mu(C_k)$	$\nu(C_k)$
$C_3 = (\{5\}, \{4\})$	$\{\bar{c}\}$	$\Phi(C_8,C_3) = \bar{c}$
$C_{12} = (\{4\}, \{7\})$	$\{c\}$	$\Phi(C_8,C_{12}) = c$

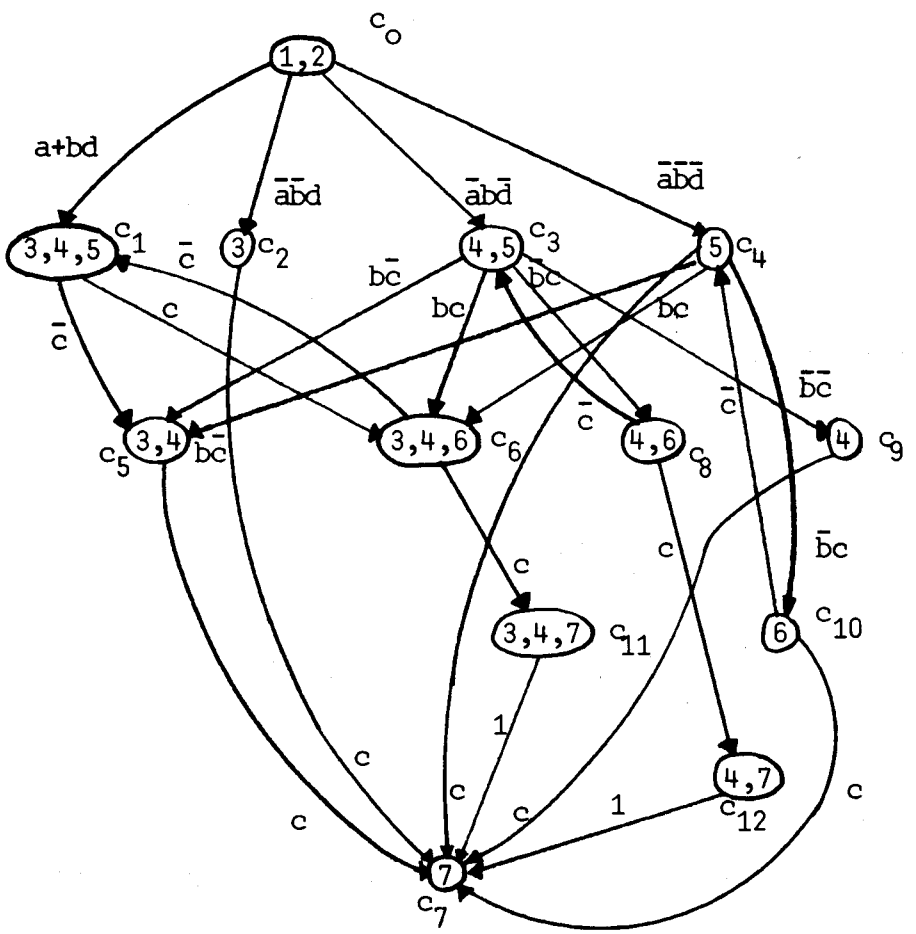
10) $C_9 = \{4,\emptyset\}$ $\Phi(C_4,C_7) = c$

11) $C_{10} = \{6,\emptyset\}$ $\Phi(C_{10},C_4) = \bar{c}$
$\Phi(C_{10},C_7) = c$

12) $C_{11} = (\{3,4\}, \{7\})$ $\Phi(C_{11},C_7) = c + \bar{c} = 1$

13) $C_{12} = (\{4\}, \{7\})$ $\Phi(C_{12},C_7) = c + \bar{c} = 1$

Ce qui donne le f.b.g. suivant



4 - PROPRIETES DU F.B.G. DU THEOREME FONDAMENTAL

4.1 Le processus A'(P)

4.1.1 Définition

A partir d'un processus élémentaire P = <R,E,ϕ>, simple ou fermé de type séquentiel, le théorème du paragraphe précédent permet de construire un flot booléen généralisé semi-homomorphe du flot du processus P et qui est défini sur un recouvrement du réseau par des contextes de déroulement.

En utilisant les notations du théorème 3.1, on définit le processus A'(P) de la façon suivante :

$$A'(P) = (R', \bar{E}, \Phi)$$

avec $- R' = (\mathcal{G}, \mathcal{D}, \mathcal{T})$ $\begin{cases} . \ \mathcal{G} = (\mathcal{C}, \mathcal{U}) \\ . \ \mathcal{D} = C_o \\ . \ \mathcal{T} = \{C_i \in \mathcal{C} \mid C_i = (A_i, B_i) \\ \qquad \text{tel que } A_i \subset T \text{ et } B_i = \emptyset\} \end{cases}$

- $\Phi : \mathcal{U} \to \bar{E}$ est le f.b.g. défini dans la démonstration du théorème 3.1.

4.1.2 Propriété

Dans le processus A'(P), $\forall\ C_i, C_j, C_k \in \mathcal{C}$

$$\left.\begin{matrix} (C_i, C_j) \in \mathcal{U} \\ (C_i, C_k) \in \mathcal{U} \end{matrix}\right) \Longrightarrow \Phi(C_i, C_j) \ . \ \Phi(C_i, C_k) \equiv 0$$

Démonstration :

Considérons l'ensemble H_i du début de la démonstration 3.1.

Par définition $\forall\ f, g \in H_i$ $\quad f.g \equiv 0$.

Soit $e \in \bar{E}$ tel que $\Phi(C_i, C_j)(e) = 1$.

Supposons que $\Phi(C_i, C_k)(e) = 1$.

Ceci implique que : $\exists\ f \in \mathcal{O}(C_k) \subset H_i$ telle que $f(e) = 1$ or $\Phi(C_i, C_j)(e) = 1 \Longrightarrow \exists\ g \neq f \quad g \in \mathcal{O}(C_j) \subset H_i$ telle que $g(e) = 1$ ce qui est impossible car H_i n'est pas conjonctif et assure que $\Phi(C_i, C_k)(e) = 0$.

cqfd.

4.1.3 Corollaire

Dans le processus A'(P),

$\forall\ C_i, C_j \in C,\ \forall\ f \in \bar{E}$

$f \rightarrow \Phi(C_i, C_j) \Longrightarrow \forall\ C_k \in C \quad C_k \neq C_j \qquad f \not\rightarrow \Phi(C_i, C_k)$

4.1.4 Conséguence

Du corollaire précédent, on déduit que dans tout déroulement du processus A'(P), un seul contexte est pris en compte à chaque pas ; il n'y a pas de cheminements simultanés sur des arcs différents et parallèles. On peut donc considérer le processus A'(P) comme un automate, pour lequel les variables d'entrée de chaque transition (C_i, C_j) appartiendraient à l'ensemble des fonctions booléennes f telles que $f \rightarrow \Phi(C_i, C_j)$.

Partant d'une description des relations entre sommets à l'aide du processus P, on a donc obtenu un diagramme d'états, le processus A'(P), en effectuant des regroupements de sommets et une mise à niveau de ces regroupements.

D'autre part, pour toute fonction booléenne $\bar{e}$ de $\bar{E}$, vraie en seul point e de $\bar{E}$, c'est-à-dire pour toute combinaison des valeurs des conditions de E, et pour tout contexte C_i, il existe au plus un contexte C_j tel que $C_j = \text{exec}(C_i; \bar{e})$ et la formulation, pour tout contexte C_i, des éventualités généralisées $\Phi(C_i, C_j)$ fournit pour ce contexte C_i, l'ensemble des conditions à tester pour déterminer la transition.

4.2 Le processus A(P)

Dans un déroulement du processus A'(P), à chaque pas, tous les sommets d'un contexte C_i ne sont pas exécutés ; seuls ceux de l'ensemble $A_i \subset C_i$ des sommets tous en l'état actif sont pris en compte. En ne considérant que les différents ensembles A_i, on peut donc déduire un diagramme d'états qui précise les sommets à exécuter à chaque pas.

On note A(P), le processus déduit de A'(P) en remplaçant chaque contexte $C_i = (A_i, B_i)$ par l'ensemble A_i et en conservant le même flot booléen généralisé.

4.3 Chaînes déroulables, chaînes principales

Définition : Une chaîne α de E^* est dite déroulable dans P si et seulement s'il existe un déroulement δ_α de P pour α.

Définition : Une chaîne $\alpha = f_o, f_1 \ldots f_n$ de E^* est dite chaîne principale de P si et seulement si

- α est déroulable dans P
- $\forall$ i $\in \{0,\ldots,n\}$ f_i n'est vraie qu'en un seul point de $\bar{E}$.

Définition : Un déroulement d'un processus P pour une chaîne α est dit principal si la chaîne α est une chaîne principale de P.

4.4 Propriétés

Propriété 4.4.1.

Tout déroulement pour une chaîne α du processus A(P) est un déroulement de P pour une chaîne principale p.

Démonstration : cf. propriété du point (c) de la démonstration du théorème 3.1.

Propriété 4.4.2.

Si δ_α est un déroulement principal de P pour une chaîne α. Alors δ_α est un déroulement de A(P).

Démonstration :

Soit $C_i = (A_i, B_i)$ le contexte de A'(P) associé à A_i. f_i, fonction vraie en un seul point de $\bar{E}$, appartient à $H_{i,j}$ (avec les notations de la démonstration du théorème 3.1), ceci implique

$$\exists\ C_j \subset \{y \mid \exists\ x \in C_i \text{ tel que } (x,y) \in U \text{ et } f_i \rightarrow \Phi(C_i, C_j)\}$$

$$\text{et } C_j = \text{exec}\ (C_i; f_i).$$

$(A_i, A_j)_{f_i}$ est donc un pas dans A(P).

5 - CONCLUSION

Pour un processus élémentaire P, simple ou fermé de type séquentiel, nous pouvons donc construire grâce au théorème 3.1 un diagramme d'états (processus A(P)) isomorphe d'un f.b.g. (processus A'(P)) qui est semi-homomorphe du flot ϕ définissant P.

Dans ce processus A(P), tout chemin menant de C_o à un contexte terminal de T est un déroulement du processus P.

Inversement, tout déroulement principal de P tel que, à chaque pas, l'ensemble de tous les sommets à l'état actif est exécuté, détermine un chemin menant de C_o à un contexte terminal dans A(P).

De plus l'algorithme détermine la fonction booléenne de transition dont la formulation précise les conditions booléennes à tester.

Dans une optique micro-programmation, les sommets du processus A(P) représentent les micro-instructions, ensemble de micro-opérations et le f.b.g. précise les conditions à vérifier pour exécuter la micro-instruction successeur.

Partant d'une représentation du micro-programme sous forme de micro-opérations liées par des relations de successions auxquelles étaient attachées des conditions, on a donc abouti, dans le processus A(P), à une image du micro-programme en termes de micro-instructions.

Chaque micro-instruction est parfaitement définie comme un ensemble de micro-opérations et les micro-instructions successeurs sont déterminées, ainsi que les fonctions d'enchaînements, de manière non ambiguë.

Dans tout ce qui précède, nous ne nous sommes préoccupés que des contraintes logiques apparaissant dans le micro-programme, dans le chapitre suivant, nous prendrons en compte les contraintes physiques qui peuvent exister entre les micro-opérations.

C H A P I T R E 5

OPTIMISATION DES DEROULEMENTS REALISABLES D'UN PROCESSUS ELEMENTAIRE

V.1. Introduction.

V.2. Déroulements réalisables.

V.3. Déroulement minimal.

V.4. f.b.g. réalisable.

V.5. f.b.g. réalisable minimal.

V.6. f.b.g. optimal.

V.7. Exemple.

V.8. Conclusion.

1 - INTRODUCTION

Dans le chapitre précédent, un algorithme a été présenté, qui construit, à partir d'un processus simple ou fermé de type séquentiel, un flot booléen généralisé semi-homomorphe du flot définissant le processus. Ce f.b.g. définit un ensemble de déroulements principaux du processus P tels que, à chaque pas, l'ensemble de tous les sommets se trouvant en l'état actif est exécuté.

Cependant, si l'on considère un ensemble dont tous les sommets se trouvent à un instant donné, à l'état actif, il peut se faire que des contraintes physiques interdisent une exécution simultanée de tous ces sommets.

En particulier, deux tâches différentes nécessitant des ressources communes et non partageables ne peuvent être exécutées simultanément et, par conséquent, deux occurrences de la même tâche ne peuvent non plus être exécutées simultanément.

Le problème est alors de construire un flot booléen généralisé, semi-homomorphe du flot ϕ servant à définir le processus, qui soit réalisable, c'est-à-dire qui définisse un ensemble de déroulements principaux du processus tels que, à chaque pas, les sommets soient physiquement exécutables de façon simultanée. Le paragraphe suivant va préciser ces notions.

2 - DEROULEMENTS REALISABLES

2.1 Ensembles compatibles

2.1.1. Ensemble logiquement compatible

Un ensemble de sommets d'un processus élémentaire est dit logiquement compatible si et seulement si tous ses sommets peuvent se trouver simultanément en l'état actif.

La reconnaissance des ensembles logiquement compatibles est effectuée lors de l'élaboration du f.b.g., puisque seules sont prises en compte les contraintes logiques de succession. Il n'en est plus de même si l'on introduit les contraintes d'origine physique qui doivent

être connues à priori.

2.1.2. Nature des sommets

Chaque sommet du processus représente une tâche, mais une même tâche peut apparaître un certain nombre de fois dans le processus.

A chaque sommet, on associe la nature de ce sommet qui est en fait l'appellation de la tâche dont le sommet représente une occurrence.

On peut donc considérer tout sommet du processus comme un couple (x_1,x_2) où x_1 représente le numéro du sommet dans le réseau et x_2 la nature de ce sommet.

2.1.3. Sommets incompatibles

Les incompatibilités d'exécution simultanée de deux tâches qui sont d'origines physiques ne peuvent être découvertes par un algorithme et doivent être données lors de la description du processus.

Au processus P on associe donc une relation I définie sur l'ensemble des natures des sommets et telle que $I(x,y)$ si et seulement si x et y sont physiquement incompatibles lors de l'exécution.

Deux sommets sont dits physiquement incompatibles si leurs natures sont incompatibles.

2.1.4. Ensemble compatible

Un ensemble de sommets est dit physiquement compatible s'il ne contient pas deux sommets physiquement incompatibles.

Un ensemble de sommets est dit compatible s'il est logiquement et physiquement compatible.

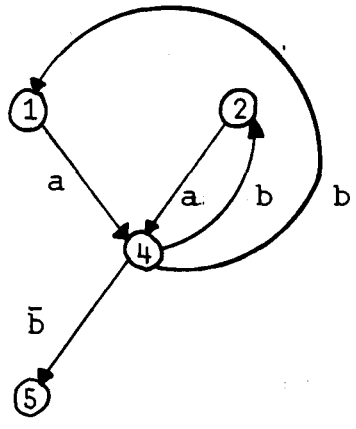
2.2 Déroulement par des ensembles compatibles

2.2.1 Introduction et problème

Considérons un ensemble X_o de sommets d'un processus P tel que X_o soit logiquement compatible et physiquement incompatible.

Les sommets qui ne peuvent être exécutés simultanément doivent être exécutés de façons successives : un sous-ensemble compatible de X_o est tout d'abord exécuté (à l'aide de la procédure exec précédente), puis un autre sous-ensemble compatible disjoint doit être exécuté ; il peut l'être seul ou avec les successeurs du premier sous-ensemble et ceci va poser un problème du fait de la régulation du processus par l'intermédiaire des ensembles de départ.

En effet, considérons un processus cyclique (ou fermé de type séquentiel) tel que D ne soit pas physiquement compatible, comme dans l'exemple ci-après :



D = {1,2}

T = {5}

1 et 2 incompatibles

Si 1 et 2 sont incompatibles, on ne peut effectuer Exec({1,2};a). Supposons que l'on effectue Exec ({2};a) en considérant que e(1) = bloqué.

Au début de la procédure déblocage, les sommets 1 et 4 sont à débloquer ; le sommet 1 appartient à D, et, par conséquent seul 4 est débloqué. Si l'on effectue alors exec({4},b), à l'issue de cette procédure on se retrouve dans le cas initial sans que {1} n'ait été exécuté. Si l'on considère la chaîne ababab̄1, on obtient le déroulement 2-4-2-4-2-4-1-5, c'est-à-dire que les sommets 2 et 4 sont bien exécutés 3 fois lorsque l'on tourne dans la boucle mais que 1 n'est exécuté qu'une fois, bien que 1 et 2 jouent des rôles symétriques.

La procédure exec se révèle donc insuffisante dans le cas où tous les sommets en l'état actif ne sont pas exécutés simultanément : il faut tenir compte du fait que les sommets non exécutés ne se trouvaient pas tous en l'état bloqué et par conséquent la partition d'un contexte entre sommets actifs et bloqués est insuffisante.

2.2.2. Contexte de processus

On appelle contexte de processus lors d'une exécution le triplet $C = (A,\bar{A},B)$ tel que

- A représente l'ensemble des sommets en l'état actif et à exécuter (ensemble compatible)

- $\bar{A}$ représente les autres sommets en l'état actif

- B représente les sommets se trouvant en l'état bloqué.

La définition d'un contexte donné au paragraphe IV.2 correspond au cas particulier $C = (A,\emptyset,B)$, mais avec la définition ci-dessus, un contexte de processus d'une exécution n'est plus caractérisé par l'ensemble de ses sommets.

2.2.3. Exécution d'un contexte

Considérons un contexte $C = (A,\bar{A},B)$. On définit l'exécution du contexte C sous une éventualité f comme indiqué par la procédure ci-dessous et on note exec(C;f) cette procédure (celle donnée au paragraphe IV.2.2.1 en étant un cas particulier).

On rappelle (§ IV.3.5.3) que l'on note Y_i l'ensemble des sommets de X de niveau de connexité i.

```
Exec (C;f) : début
     Y := {y ∈ X | ∃ x ∈ A tel que (x,y) ∈ U et f → φ(x,y)} ;
     ∀ x ∈ A faire e(x) := passif ;
     ∀ y ∈ Y faire e(y) := bloqué ;

  déblocage :
     Z := {x ∈ X | e(x) := bloqué} ;
     ∀ i ∈ ℕ faire Z_i := Z ∩ Y_i
     Si Ā ≠ ∅ Alors
        début j:= n(Ā) ; D := ensemble de départ du sous-processus
                              fermé minimum contenant A s'il existe,
                              ∅ sinon ;
              Z_j := Z_j / D ;
              Z_j := Z_j ∪ Ā ;
              debloq (Z_j) ;
        fin        sinon
        début      i_max := max (i) ;
                         Z_i≠∅
                   P_k := le sous-processus élémentaire fermé de P
                          minimal contenant Z_i_max
                   Si Z_i_max ⊂ D_k alors debloq (D_k)
                                  sinon debloq (X_k / D_k) ;
        fin
                   fin
```

2.2.4. Déroulement réalisable

Soit un contexte $C = (A,\bar{A},B)$. L'exécution de l'ensemble A pour une éventualité f, à l'aide de la procédure exec fait correspondre à A un ensemble de contextes $C' = (A',\bar{A}',B')$ où $A' \cup \bar{A}'$ représente l'ensemble des sommets en l'état actif.

Comme précédemment, on appelle pas un couple $p = (A,A')$ et déroulement une succession de pas $(p_o,p_1,\ldots,p_n)$ telle que $p_i = (A_i,A_{i+1})$ pour tous les indices i de $\{0,\ldots,n\}$.

On dira que ce déroulement est réalisable si et seulement si l'ensemble A_i est compatible pour tout indice i de $\{0,\ldots,n+1\}$.

Etant donné un processus élémentaire P, il est toujours possible de construire un déroulement réalisable de ce processus : il suffit, étant donné un déroulement par des ensembles logiquement compatibles, d'exécuter en série les sommets qui ne peuvent être exécutés de façon simultanée ; à la limite on peut n'exécuter qu'un seul sommet à chaque pas.

Une telle méthode est évidemment aberrante puisque l'on allonge de façon démesurée le déroulement du processus. Dans une optique micro-programmation, ceci reviendrait à n'exécuter qu'une seule micro-opération à chaque cycle !

Le problème est donc de construire des déroulements réalisables qui répondent à certains critères d'optimalité, en particulier en tenant compte de la longueur du déroulement.

3 - DEROULEMENT MINIMAL

3.1 Déroulements équivalents

On considère l'ensemble des déroulements d'un processus élémentaire P et on définit sur cet ensemble une relation binaire $\mathcal{R}$ de la façon suivante :

Soient δ et γ, deux déroulements de P pour des chaînes α et β,

$\delta = p_o \ldots p_n$ tel que $\forall\ i \in \{0,\ldots,n\}\ p_i = (X_i, X_{i+1})$

$\gamma = q_o \ldots q_m$ tel que $\forall\ i \in \{0,\ldots,m\}\ q_i = (Y_i, Y_{i+1})$

On note, pour tout sommet $x \in X$, $x(n)$ la $n^{\text{ième}}$ occurrence de x, en numérotant selon l'ordre de rencontre dans le déroulement.

δ et γ vérifient la relation $\mathcal{R}$ ($\mathcal{R}(\delta,\gamma)$) si et seulement si

1. Chaque sommet de P a le même nombre d'occurrences dans chacun des déroulements δ et γ.

2. $\forall$ x,y tels que $(x,y) \in U$ et $y \notin W_x$ (où W_x est l'ensemble défini au paragraphe III.4.5.2) et $\forall$ X_i et X_j dans δ

$$\left.\begin{array}{l} x(n) \in X_i \\ y(p) \in X_j \\ \quad j > i \end{array}\right\} \Longrightarrow \left(\begin{array}{l} \exists\ Y_k,\ Y_\ell \text{ dans } \gamma \\ \text{tels que } - \ x(n) \in Y_k \\ \qquad\qquad\quad - \ y(p) \in Y_\ell \\ \qquad\qquad\quad - \ \text{et } k > \ell \end{array}\right.$$

Cette relation est symétrique : ceci se démontre à l'aide de la propriété suivante de la procédure exec

$$\left.\begin{array}{l} x_n \in X_i \\ y_p \in X_i \\ (x,y) \in U \end{array}\right\} \Longrightarrow y_p \in W_x$$

et d'autre part en tenant compte du point 1.

Elle est d'autre part réflexive et transitive. C'est donc une relation d'équivalence sur l'ensemble des déroulements d'un processus élémentaire.

Dans la suite, on s'intéressera aux classes d'équivalence restreintes aux déroulements principaux d'un processus élémentaire.

3.2 Propriété

Etant donnée une classe d'équivalence de déroulements principaux d'un processus élémentaire P, tout déroulement δ_α de cette classe dont les ensembles de sommets X_i de chaque pas appartiennent à des contextes $C_i = (A_i,\emptyset,B_i)$ et un déroulement de longueur minimale dans la classe d'équivalence.

Démonstration :

Soient δ_α et δ_β deux déroulements principaux équivalents d'un d'un processus P :

$\delta_\alpha = p_o \ldots p_n$ tel que $\forall$ i $\in \{0,\ldots,n\}$

$p_i = (X_i, X_{i+1})_{f_i}$ et, $X_i = \mathrm{Exec}(C_i; f_i)$ et $C_i = (X_i, \emptyset, B_i)$

$\gamma_\beta = q_o \ldots q_m$ tel que $\forall$ i $\in \{0,\ldots,m\}$

$q_i = (Y_i, Y_{i+1})_{g_i}$, $Y_i = \mathrm{Exec}(C'_i; g_i)$ et $C'_i = (Y_i, \bar{Y}_i, B'_i)$.

On suppose que m<n, c'est-à-dire que le déroulement γ_β est de longueur inférieure à celle de δ_α.

Soit i le premier indice tel que $\bar{Y}_i \neq \emptyset$.

$$\Longrightarrow \forall\ k < i \quad C_i = C'_i$$
$$\Longrightarrow X_i = Y_i \cup \bar{Y}_i \text{ et } B_i = B'_i$$

Nous allons raisonner par récurrence.

Supposons (hypothèse de récurrence) qu'il existe k < m tel que

$$\forall\ j \leq k \qquad Y_j \cup \bar{Y}_j \cup B'_j \subset \bigcup_{\ell=0}^{j} (X_\ell \cup B_\ell)$$

Initialement, cette relation est vérifiée pour $k \leq i$.

Vérifions qu'elle est vraie pour l'indice j+1 :

Soit y une occurrence apparaissant dans le contexte C'_{j+1}.

- Si $y \in Y_{j+1}$ ou $\exists\, x \in Y_j$ tel que $(x,y) \in U$ et $g_j \Longrightarrow \phi(x,y)$

  et comme $x \in \bigcup_{\ell=0}^{j} (X_\ell \cup B_\ell)$,

  $y \in \bigcup_{k=0}^{j+1} (X_k \cup B_k)$

  ou $y \in \bar{Y}_j$

  ou $y \in B'_j$

  dans ces trois cas $y \in \bigcup_{\ell=0}^{j+1} (X_\ell \cup B_\ell)$

- Si $y \in \bar{Y}_{j+1}$ ou $\exists\, x \in Y_j$ tel que $(x,y) \in U$ et $g_j \rightarrow \phi(x,y)$

  ou $y \in \bar{Y}_j$

  ou $y \in B'_j$

  et comme précédemment $y \in \bigcup_{\ell=0}^{j+1} (X_\ell \cup B_\ell)$

- Si $y \in B'_{i+1}$ ou $\exists\, x \in Y_j$ tel que $(x,y) \in U$ et $g_j \rightarrow \phi(x,y)$

  ou $y \in B'_j$

La relation ci-dessous est donc vérifiée pour tout indice $j \leq m$.

En conséquence, $\bigcup_{j=0}^{m} (Y_j \cup \bar{Y}_j \cup B'_j) \subset \bigcup_{j=0}^{m} (X_j \cup B_j)$.

Or le point 1 de la définition de l'équivalence de deux déroulements précise que chaque sommet du graphe a le même nombre d'occurrences dans les deux déroulements et par conséquent l'hypothèse de départ est erronée.

La longueur du déroulement γ_β est donc supérieure ou égale à celle du déroulement δ_α.

3.3 Conséquence

Considérons un processus élémentaire P. Le théorème fondamental permet de construire un processus A'(P) (f.b.g. semi-homomorphe de ϕ) dont tous les sommets sont des contextes $C = (A,\emptyset,B)$.

La propriété précédente assure donc que tout chemin menant du contexte de départ à un contexte terminal, dans le processus A(P), détermine un déroulement principal de P de longueur minimale dans sa classe d'équivalence.

Cependant, dans l'élaboration de ces cheminements minimaux, on n'a pas tenu compte des contraintes physiques : un ensemble de sommets logiquement compatible est exécuté à chaque pas.

En prenant en compte les incompatibilités physiques, nous allons rechercher des déroulements réalisables de longueur minimale dans chaque classe d'équivalence.

4 - FLOT BOOLEEN GENERALISE REALISABLE

4.1 Définition

Considérons un processus élémentaire P défini à l'aide d'un flot booléen ϕ sur un réseau R et un f.b.g. Φ défini sur un recouvrement de ce réseau.

Le f.b.g. Φ est dit réalisable si et seulement si chacun de ses sommets est un sous-graphe du réseau R dont l'ensemble des sommets est compatible.

Une conséquence de cette définition est que les sous-graphes, sommets du f.b.g., ne contiennent pas d'arc, sinon ils ne formeraient pas des ensembles compatibles.

De façon analogue à l'algorithme du chapitre précédent, est-il possible de construire un f.b.g. Φ semi-homomorphe du flot ϕ servant à définir le processus qui soit réalisable et dont le déroulement par une chaîne principale du processus associé ne présente pas de parallélisme ?

4.2 Théorème

Soit $P = \langle R, E, \phi \rangle$ un processus élémentaire. Il existe un recouvrement de ce processus par des contextes de déroulements réalisables sur lequel est défini un flot booléen généralisé semi-homomorphe du flot ϕ.

Démonstration :

La démonstration du théorème IV.3.1 repose sur l'utilisation de contextes du type $C = (A, \emptyset, B)$ où A est un ensemble logiquement compatible. Si A n'est pas physiquement compatible, il devient nécessaire de n'exécuter qu'un sous-ensemble A' de A à l'aide de la procédure Exec.

Supposons que l'on ait déjà mis en évidence un certain nombre de contextes de déroulements réalisables $C_0\ C_1 \ldots C_i\ C_{i+1} \ldots C_j$.

On s'intèresse au contexte $C_i = (A_i, \bar{A}_i, B_i)$ - $(i \leq j)$ - après avoir déjà examiné les contextes $C_0\ C_1 \ldots C_{i-1}$.

Initialement on pose $C_0 = (D', \bar{D}', \emptyset)$ avec $D' \cup \bar{D}' = D$ et D' compatible.

On note X_i l'ensemble des successeurs possibles de A_i et H_i l'ensemble d'implicants d'éventualités analogue à celui de la démonstration IV.3.1.

Pour toutes les fonctions f de H_i, on exécute la procédure Exec $(C_i;f)$ et on note $C'_0, C'_1, \ldots, C'_m$ les contextes distincts obtenus à l'issue de l'exécution des procédures Exec $(C_i;f)$ pour les fonctions f de H_i, avec $\forall\ k \in \{0, \ldots, m\}$ $C'_k = (A'_k, \emptyset, B'_k)$.

On note pour tous les indices k de $\{0, \ldots, m\}$:

$$\mathcal{O}(C'_k) = \{f \in H_i \mid C'_k = \text{exec}\ (C_i;f)\}$$

$$\nu(C'_k) = \sum_{f \in \mathcal{O}(C'_k)} f$$

On considère chaque $C'_k = (A'_k, \emptyset, B'_k)$ parmi les contextes $C'_0\ C'_1 \ldots C'_m$ à traiter. S'il existe un contexte $C_\ell = (A_\ell, \bar{A}_\ell, B_\ell)$ parmi les contextes $C_0, C_1 \ldots C_j$ déjà mis en évidence

tel que $\begin{cases} A'_k = A_\ell \cup \bar{A}_\ell \\ B'_k = B_\ell \end{cases}$

(et s'il existe, il est unique par construction),

on pose $U_{i,\ell} = (C_i, C_\ell)$ et $\Phi(U_{i,\ell}) = \nu(C'_k)$.

Sinon si A'_k est compatible on pose $C''_k = C'_k$

si A'_k n'est pas compatible, on prend un sous-ensemble compatible A''_k de A'_k et on considère le contexte $C''_k = (A''_k, A'_k/A''_k, B'_k)$.

On note $C_{j+1}\ C_{j+2} \ldots C_{j+m_i}$ ces nouveaux contextes C''_k et $\forall\ \ell \subset \{1,\ldots,m_i\}$, si $\exists\ k \in \{1,\ldots,m\}$ tel que $C''_k = C_{j+\ell}$

on pose $U_{i,j+\ell} = (C_i, C_{j+\ell})$ et $\Phi(U_{i,j+\ell}) = \nu(C''_k)$.

Puis on itère le procédé avec les contextes $C_{i+1}\ C_{i+2} \ldots$ et la suite de la démonstration est analogue à celle du théorème IV.3.1.

On vérifie que le f.b.g. ainsi construit est semi-homomorphe du f.b.g. ϕ servant à définir le processus P.

4.3 Propriétés

Ce f.b.g. Φ semi-homomorphe de ϕ mais construit sur des ensembles compatibles possède des propriétés similaires à celles du f.b.g. du chapitre IV.

On peut lui associer un processus A(P) qui est un automate pour les chaînes principales puisqu'il n'y a alors ni parallélisme, ni attente en un sommet, et qui est tel que, à tout chemin menant du

contexte initial C_0 à un contexte terminal, on peut associer un déroulement principal réalisable de P.

Sous réserve de ne prendre en compte que des ensembles compatibles, l'algorithme de construction du paragraphe IV.3.2 peut être utilisé pour construire un f.b.g. réalisable convenable.

5 - F.B.G. REALISABLE MINIMAL

5.1. Introduction

Dans le paragraphe V.3, nous avons démontré que, dans le processus A(P) construit sur des ensembles logiquement compatibles, les chemins menant du contexte initial à un contexte terminal déterminaient des déroulements principaux de longueur minimale dans leur classe d'équivalence.

Un problème analogue se pose pour les processus A(P) construits à l'aide d'ensembles compatibles : peut-on construire un processus A(P) à l'aide d'un f.b.g. réalisable tel que tout chemin menant du contexte initial à un contexte terminal soit un déroulement principal, réalisable et de longueur minimale dans la classe d'équivalence.

5.2. Déroulements réalisables minimaux

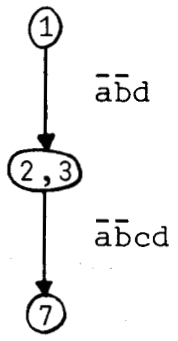
Considérons le processus A(P) obtenu dans l'exemple IV.3.3 et supposons que les sommets (1) et (2) du processus élémentaire P soient incompatibles.

Considérons, dans ce processus A(P), le chemin $((C_0,C_2),(C_2,C_7))$. Ce chemin détermine un déroulement

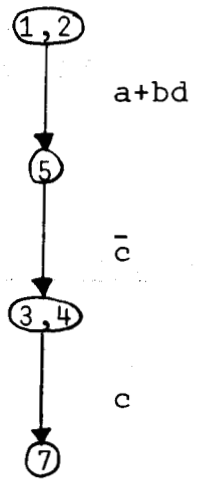
$$\delta = p_0\ p_1 \qquad \text{avec} \qquad p_0 = (\{1,2\},\ \{3\})_{\bar{a}\bar{b}d}$$
$$p_1 = (\{3\},\ \{7\})_c$$

mais ce déroulement δ n'est pas réalisable car les sommets 1 et 2 sont physiquement incompatibles.

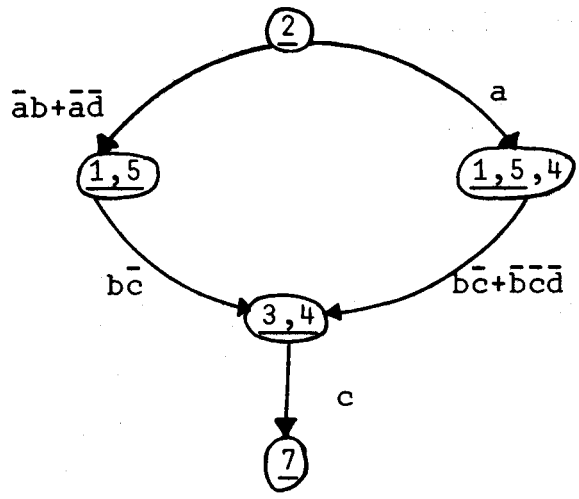
En utilisant une procédure d'exploration du réseau, combinée avec l'utilisation des grilles d'éventualités analogues à celle de l'algorithme du paragraphe IV.3.2, on peut mettre en évidence les déroulements réalisables minimaux du graphe. Un tel algorithme fournirait pour cet exemple le déroulement suivant comme seul déroulement principal qui soit réalisable et minimal



Considérons maintenant dans le même processus A(P), le chemin $((C_0,C_1),(C_1,C_5),(C_5,C_7))$ auquel on associe le déroulement minimal logiquement compatible



La même méthode fournit les déroulements de contextes réalisables de longueur minimale suivants :



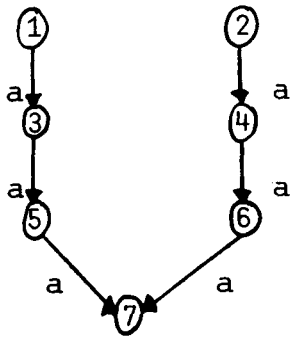
Dans ces deux exemples, les déroulements principaux, réalisables et minimaux commencent par des contextes différents. Il n'existe donc pas nécessairement un processus A(P) dont les chemins déterminent des déroulements réalisables minimaux.

5.3. f.b.g. réalisable minimal

En conséquence, nous allons recherche un processus A(P) qui ne soit pas "trop mauvais", c'est-à-dire tel que la somme des longueurs des déroulements réalisables soit minimale.

Une première méthode pourrait consister en la construction progressive d'un déroulement réalisable, sur la base de règles de décision locale. On pourrait penser, en première approche, que l'exécution du nombre maximal de sommets à chaque pas oriente la construction vers un déroulement réalisable de longueur minimale. Cependant les décisions prises à chaque stade ne tiennent pas compte des conséquences sur les décisions à venir, comme le montre l'exemple ci-après

Considérons le processus :



D = {1,2}

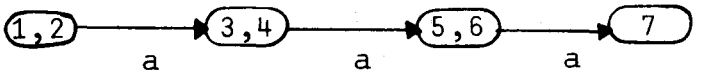
T = {7}

I	2	3	4	5	6	7
1	0	|		|		|
2		+	|		|	|
3			+	|		|
4				+	|	|
5					+	|
6						|

+ incompatibilités physiques ou logiques

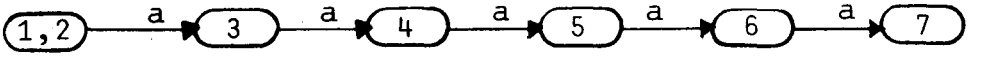
| incompatibilités logiques (par fermeture transitive)

On obtient le déroulement logiquement compatible suivant

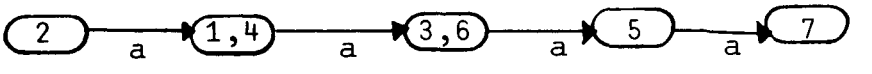


longueur = 4

Si l'on exécute au premier pas l'ensemble maximal {1,2}, on aboutit à un déroulement réalisable de longueur 6



Par contre si l'on exécute au premier pas uniquement le sommet 2, on obtient le déroulement réalisable minimal (longueur 5)



Ne pouvant décider localement si l'on s'oriente vers un déroulement réalisable minimal, il est nécessaire de recourir à une méthode plus systématique d'exploration des différents processus A(P) réalisables, pour n'en conserver qu'un, évalué comme minimal. L'algorithme à mettre en oeuvre étant important en temps et occupation

mémoire, il est intéressant d'utiliser une fonction d'évaluation faisant intervenir d'autres contraintes.

6 - F.B.G. OPTIMAL

6.1 Introduction

Considérons un processus élémentaire P défini par un flot booléen ϕ sur un réseau R. Chacun des sommets du réseau est un couple (x_1,x_2) où x_1 représente le numéro du sommet dans le réseau et x_2 la nature de ce sommet. Une relation binaire I traduit les incompatibilités physiques entre ces natures.

On appellera nature d'un ensemble de sommets physiquement compatibles, l'ensemble des natures de ces sommets.

Lors de la construction du processus A(P) décrivant les déroulements principaux réalisables du processus P, on peut rechercher, lorsque l'on regroupe les sommets à créer, le moins de natures d'ensemble possibles. En terme de micro-programmation, cela revient à se définir le moins de types de micro-instructions possibles.

Lors de la construction du processus A(P), une telle recherche pourrait se faire en tenant compte à priori du nombre d'occurrences de chacune des natures de tâches et lors de la construction des déroulements, des regroupements déjà créés.

Cependant, ces deux critères ne sont pas des conditions nécessaires d'optimalité et il n'est pas possible de savoir localement, comme pour la recherche d'un déroulement minimal, si l'on s'oriente vers un déroulement optimal.

6.2 Recherche du f.b.g. optimal

Si l'on applique l'algorithme du paragraphe IV.3.2, avec la procédure Exec définie au paragraphe V.2.2.3, à tous les sous-ensembles compatibles de chaque contexte mis en évidence, on peut

générer tous les processus A(P) réalisables et le théorème V.4.2 assure que ces processus possèdent les propriétés requises (semi-homomorphie, déroulements principaux réalisables, pas de parallélisme). Il est alors possible de choisir un de ces processus A(P), optimal pour le ou les critères choisis.

Cependant, par son aspect combinatoire, cette énumération devient rapidement coûteuse dès que la taille du processus P augmente et pour un critère donné, il est intéressant de pouvoir abandonner suffisamment tôt la construction d'un processus A(P). Encore est-il nécessaire de disposer d'une fonction d'évaluation permettant de décider si la construction d'un processus A(P) peut être interrompue ou reprise.

Sous une telle hypothèse, la recherche d'un f.b.g. réalisable optimal peut être traitée à l'aide d'une méthode du type séparation et évaluation progressive.

Si l'on adopte les critères de longueur totale des déroulements minimale et nombre de natures de regroupements minimal, tels qu'ils ont été présentés aux paragraphes 5 et 6.1 précédents, le problème se pose de la façon suivante :

$$\text{minimiser une fonction } f = k\,d + \ell\,\text{nat} \qquad (k,\ \ell \in \mathbb{N})$$
$$d \leq d_{max}$$
$$\text{nat} \leq \text{nat}_{max}$$

où d mesure la "longueur" totale de A(P) et
nat le nombre de natures de regroupements créés.

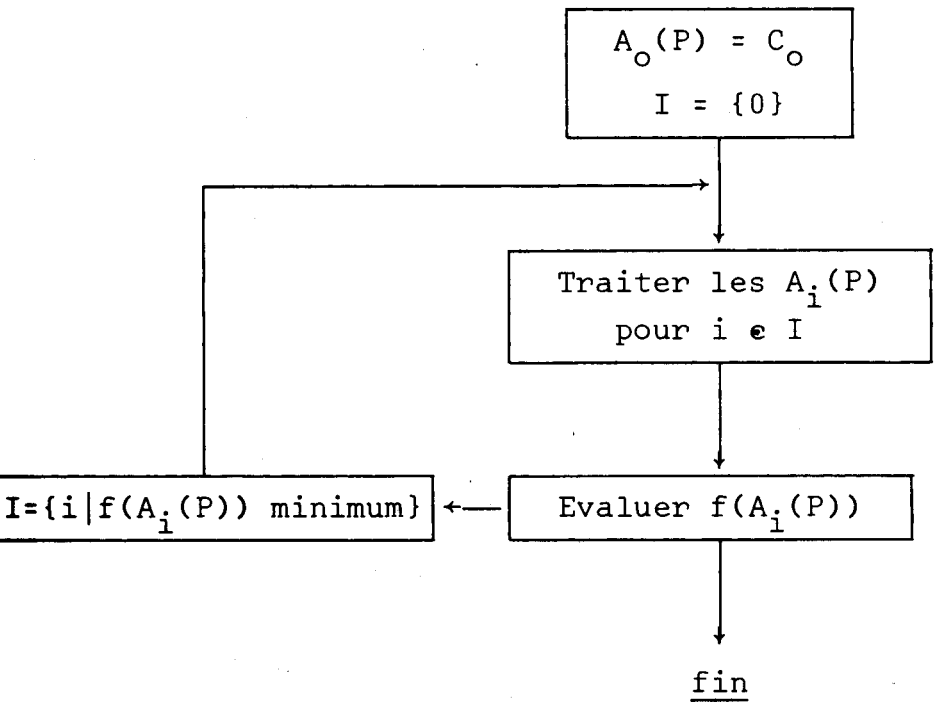
La longueur totale des processus réalisables A(P) peut être évaluée par comparaison avec le processus A(P) construit sur des ensembles logiquement compatibles, qui est minimal. On notera $\bar{A}(P)$ ce processus qui représente une borne inférieure pour les totaux des longueurs de déroulements. On effectue une mise à niveau par les prédécesseurs du processus $\bar{A}(P)$ (en regardant les sous-processus fermés maximaux comme formant un seul niveau, que l'on affine ensuite en

sous-niveaux lors de leur examen).

Soit $A_i(P)$, une étape de la construction d'un processus réalisable A(P), mis à niveau par les prédécesseurs dont le niveau maximal est i.

On considère les sommets de P apparaissant dans les contextes de niveau i et on note j le minimum des niveaux de ces sommets dans le processus $\bar{A}(P)$. La différence d = i-j donne une idée du "retard" pris dans la construction de A(P) par rapport au processus minimum $\bar{A}(P)$, donc de l'allongement moyen du processus réalisable.

L'organigramme général de la méthode est alors le suivant :

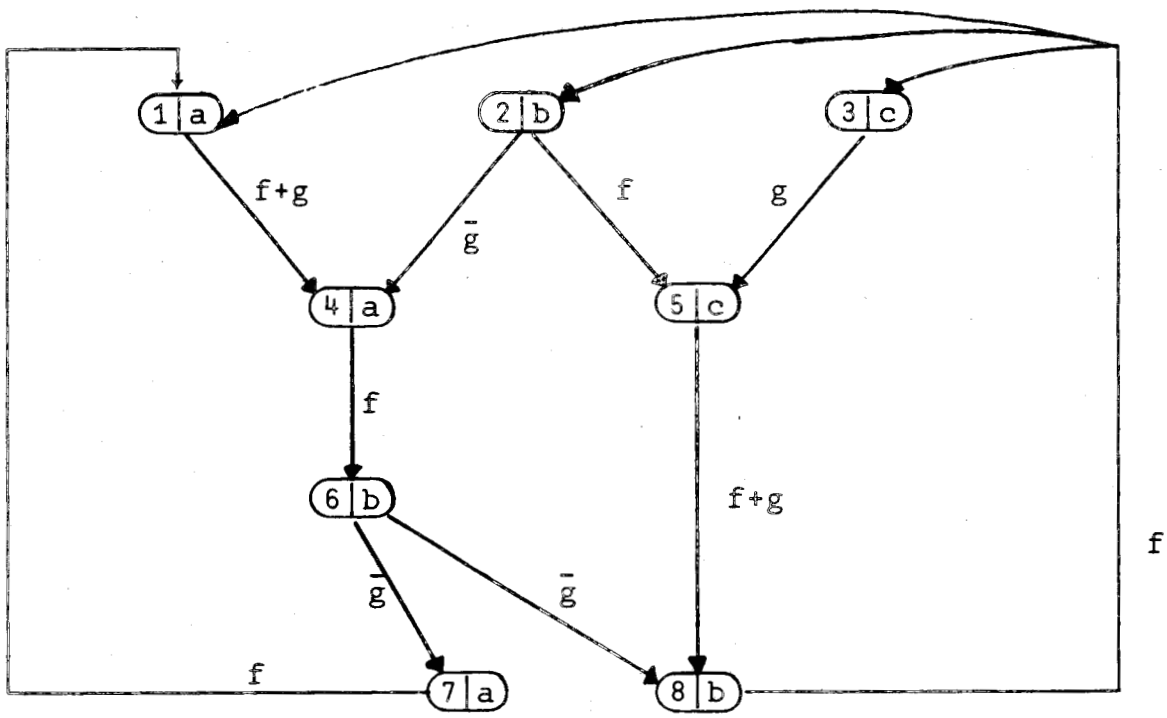


Traiter un $A_i(P)$ consiste à appliquer une itération de l'algorithme de construction du f.b.g. à tous les sous-ensembles compatibles des contextes de niveau maximum.

On construit ainsi une arborescence de solutions partielles conduisant à une solution optimale pour le critère choisi.

7 - EXEMPLE

Considérons le processus élémentaire ci-dessous :

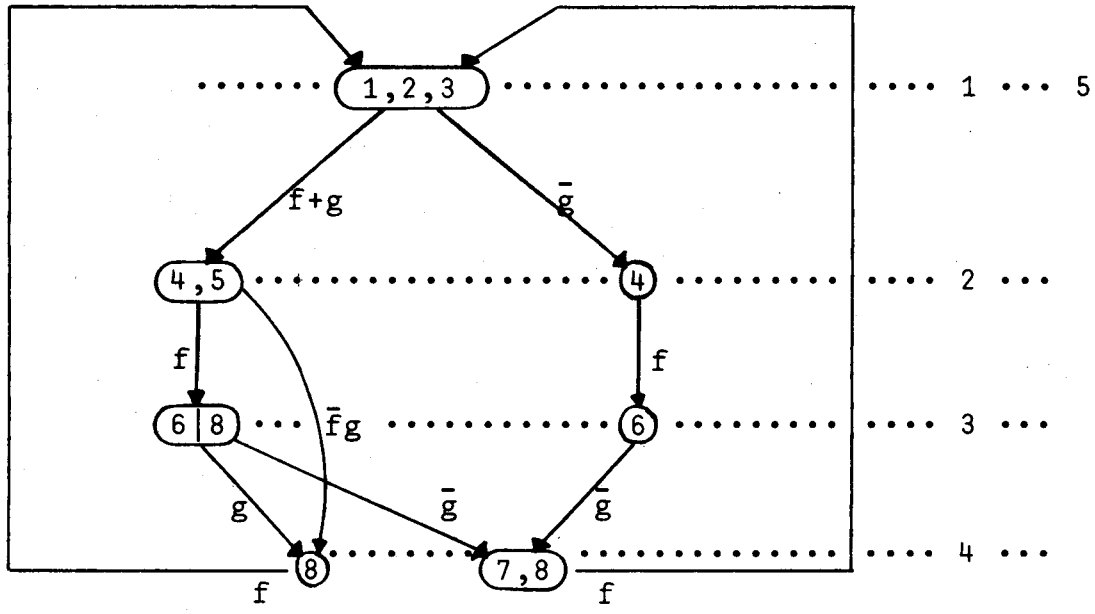


avec D = {1,2,3}
T = {7,8}

Incompatibilité physique : I(a,c)

Contrainte particulière :
si e(7) = e(8) = actif alors exécuter simultanément (7) et (8).

L'algorithme du paragraphe IV.3.2, utilisant les ensembles logiquement compatibles, donne le processus A(P) suivant, dont les déroulements principaux sont minimaux



Les sommets de ce processus ne sont pas compatibles :

{1,2,3} a comme nature de sommets a, b et c et I(a,c).

Nous allons appliquer l'algorithme d'optimisation, en prenant comme critère minimiser d+n (d indicateur de retard, n nombre de types de regroupements de natures).

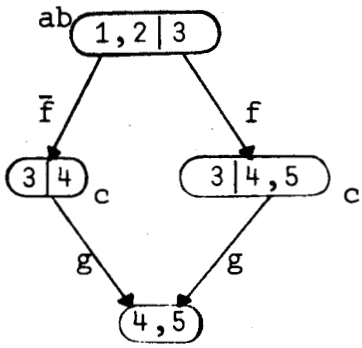
Etape 1 : {1,2,3} contient les sous-ensembles compatibles suivants :
{1,2}, {2,3}, {a}, {b}, {c}.

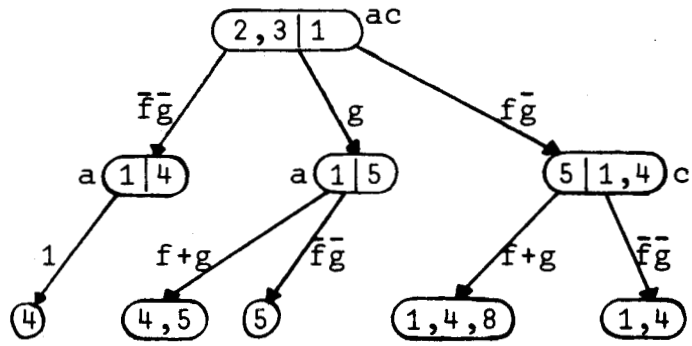
Appliquer une itération à chacun des contextes de sommets actifs ci-dessus.

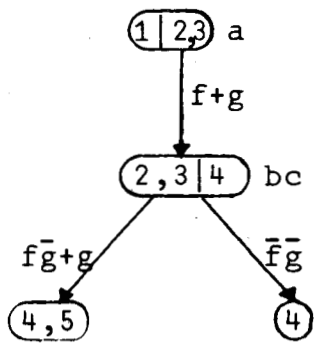
d+n = 2 pour chacun des éléments construits.

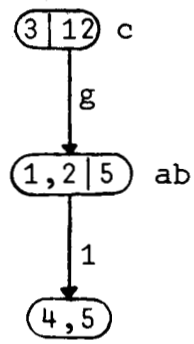
Etape 2 : On applique une nouvelle itération aux contextes mis en évidence.

L'évaluation, pour chaque élément de d+n, sélectionne 4 éléments sur les 22 construits (d=1, n=2)







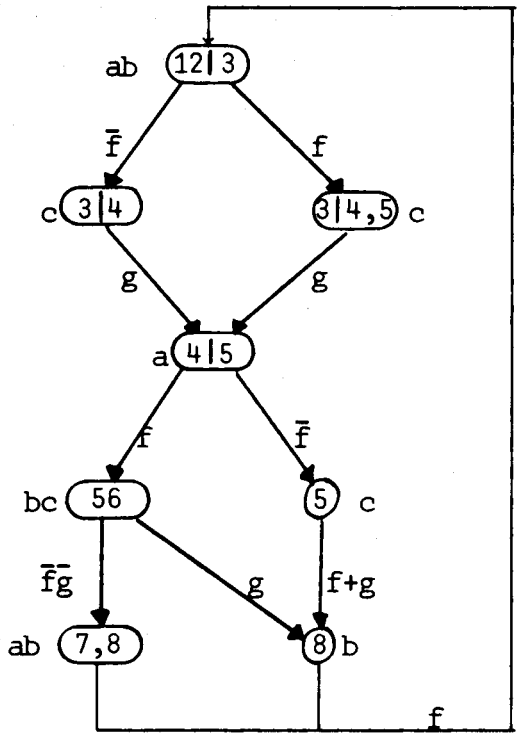


Etape 3 : On effectue un pas de construction sur ces 4 éléments. Sur les 8 éléments construits lors de cette étape, 2 ont une fonction d'évaluation minimale et égale à 4 et, d'autre part, 7 des éléments construits lors d'une étape précédente ont également la même valeur de fonction d'évaluation.

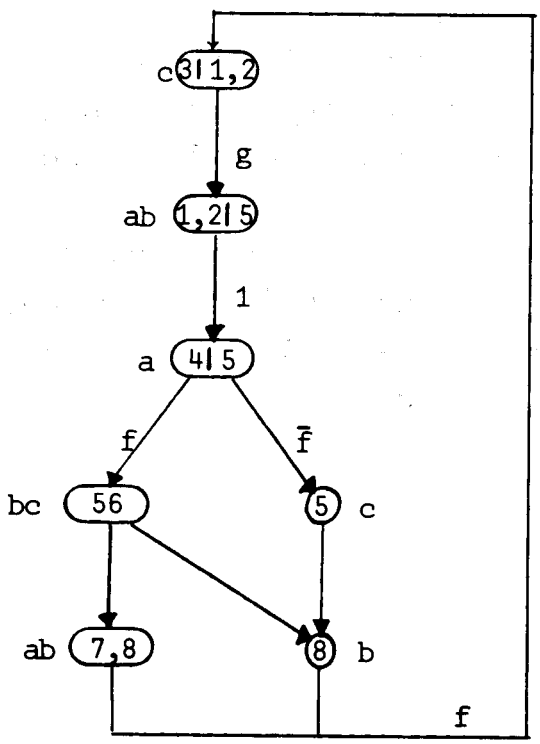
Etape 4 : Les 9 éléments précédents sont traités et donnent 28 nouveaux éléments. 2 éléments ont une fonction d'évaluation minimale (d+n=5) ainsi que 5 éléments de l'étape précédente et 11 des autres étapes antérieures.

Etape 5 : A l'issue de l'étape 5, 4 éléments ont une fonction d'évaluation de valeur minimale.

Etape 6 : Ces 4 éléments donnent au cours de cette étape, 4 nouveaux éléments dont 2 ont une fonction d'évaluation minimale et dont tous les contextes ont été examinés. L'algorithme est donc terminé.



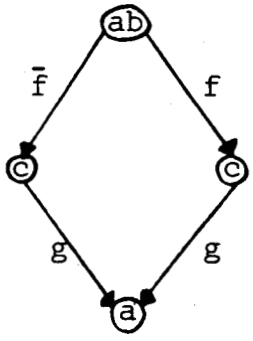
III



IV

BUS LILLE

On remarque que les deux processus réalisables mis en évidence se déduisent l'un de l'autre par inversion des exécutions de {1,2} et {3}. Des calculs peuvent donc être communs dans la construction. D'autre part, certaines améliorations sont encore possibles : l'écriture du début de I en terme de natures.



montre que l'on peut réunir les contextes (3|4) et (3|4,5) pour donner

De même pour les contextes (7,8) et (7,8|1,2,3) dans cet exemple. Leurs constructions sont cependant nécessaires dans la méthode si l'on veut ne pas perdre certaines transitions, dans le cas général.

Enfin, le nombre d'éléments examinés à chaque étape est relativement important, car la fonction évaluation augmente ici par pas de 1. Pour une fonction du type d+10n le nombre d'éléments à traiter à chaque étape chute rapidement.

8 - CONCLUSION

On a donc ramené le problème d'optimisation des déroulements principaux et réalisables d'un processus P, c'est-à-dire d'optimisation d'un micro-programme, à un problème d'exploration sur un graphe muni de propriétés particulières.

De façon générale, on peut tenter de mettre en oeuvre de nombreuses méthodes de recherche dans les graphes, sous réserve de remplacer les chemins habituels par des successions d'exécutions d'ensembles de sommets compatibles à l'aide de la procédure Exec et de manipulations d'éventualités.

En particulier, l'optimisation peut être rendue beaucoup plus fine, si l'on associe une probabilité de réalisation à chacune des conditions prise en compte dans le contrôle des déroulements du processus. On peut alors porter son effort vers la minimisation des déroulements les plus fréquents du processus, en se permettant des allongements jugés tolérables des autres déroulements.

D'autre part, des contraintes d'ordre divers peuvent être introduites (exécution simultanée obligatoire de plusieurs sommets, succession immédiate sans blocage...) et les méthodes d'exploration sont suffisamment puissantes (malgré leurs coûts élevés en temps et mémoire) et souples pour les prendre en compte.

C O N C L U S I O N

Le concept de processus a été précisé à l'aide de la notion de flot booléen contrôlé et, partant de la représentation d'un micro-programme non codé et non cadencé sous la forme d'un processus élémentaire, nous avons élaboré une méthode qui construit un processus A(P) réalisable et optimal pour des critères variables.

Ce processus A(P) est défini sur un réseau dont les sommets représentent l'ensemble des micro-opérations à exécuter à chaque cycle lors du déroulement du micro-programme et le f.b.g. qui définit A(P) précise les conditions à tester et les fonctions d'enchaînements entre les micro-instructions. Une justification de cette construction a été apportée à l'aide des propriétés des flots booléens contrôlés et des déroulements.

Nous disposons donc d'une image, optimale pour un critère, d'un micro-programme cadencé. Cependant l'exemple du dernier chapitre a montre qu'à cette étape du travail, de nouvelles améliorations peuvent être encore apportées pour minimiser la mémoire de commande et les méthodes classiques de minimisation des machines séquentielles peuvent être adoptées.

Il reste ensuite à effectuer le codage des micro-opérations et des conditions dans chaque micro-instruction. Ceci peut être réalisé par un examen systématique des sommets et des conditions dans le processus A(P) construit au cours de la phase précédente. Certaines options, comme le niveau d'intervention des conditions (cf. chapitre I) doivent cependant être prises à priori.

De même, pour décider de l'implantation du micro-programme en mémoire de contrôle, on peut concevoir une implantation automatique pour un environnement physique donné et des choix à priori, mais il semble difficile de concevoir une méthode générale.

Dans un domaine autre que la micro-programmation, ce travail est susceptible d'un certain nombre d'extensions :

D'une part, on peut considérer le processus Exec comme un cas particulier d'une loi de transition entre sous-graphes et étudier les propriétés de cette loi dans un cadre plus général.

D'autre part, on peut étendre la formulation des processus élémentaires à des processus plus généraux (entrées multiples, temps d'exécution variables, partage des ressources) et en rechercher des propriétés.

B I B L I O G R A P H I E

MICRO-PROGRAMMATION ET OPTIMISATION

[1] S.R. DAS, D.K. BANERJI, A. CHATTOPADHYAY
On control memory minimization in microprogrammed digital computers.
IEEE Trans. on comp. vol. C-22, n°9, sept. 1973 p.845

[2] R.L. KLEIR, C.V. RAMAMOORTH Y.
Optimisation strategies for microprograms.
IEEE Trans. on comp. vol. C-20, n°7, juillet 1971 p.783

[3] P. MARTIN, H. DU MANOIR
Réduction du volume des micro-programmes par une méthode d'optimisation de l'allocation des registres.
R.A.I.R.O. février 1974, B-1, p.19

[4] G. BLAIN, NGUYEN XUAN HONG, PERRONE, RUGGIU
Génération automatique de micro-programmes optimisés à partir d'une description algorithmique des macro-codes et d'une description formelle de la structure câblée.
Thomson-CSF - rapport C.R.I. 70 - 098

[5] A. GRASSELLI, F. LUCCIO
A method for minimizing the number of internal states in incompletely speciefied sequential networks.
IEEE Trans. on comp. juin 1965 p.350

[6] A. GRASSELLI
Control units for sequencing complex asynchronous operations.
IEEE Trans. on comp. août 1962 p.483

[7] G. SAUCIER
State assignment of asynchronous sequential machines using graph techniques.
IEEE Trans. on comp. vol. C-21, n°3, mars 1972 p.282

[8] B. TOURSEL
Flot booléen et micro-programmation : formalisation de la notion de processus élémentaire.
Ecole d'été d'informatique de l'AFCET 1973

[9] G. BOULAYE
Graphes contrôlés et instructions.
R.I.R.O. B-2, 1971 p.43

[10] G. SAUCIER
Next-state equations of asynchronous sequential machines.
IEEE Trans. on comp. avril 1972 p.397

MICRO-PROGRAMMATION

[11] S.S. HUSSON
Micro-programming : principles and practices.
Prentice-Hall, INC 1970

[12] International Advanced Summer institute on micro-programming.
Hermann 1972

[13] G. BOULAYE
La micro-programmation
Dunod 1971

[14] Chicago, Ind. Sci. Conf. manag. 1971 p.373-420.

[15] AFIPS Conf. proc. 1970 p.191-205.

[16] AFIPS Conf. proc. 1971 p.163.

[17] IEEE Trans. on el. computers vol. C-20 n°7 juillet 1971
(numéro spécial micro-programmation).

[18] P.M. DAVIES
Readings in micro-programming.
IBM system journal 1972 p.16

[19] C.V. RAMAMOORTHY, L.C. CHANG
System modeling and testing procedures for micro-diagnostics.
IEEE Trans. on el. comp. vol. C-21 n°11 Novembre 1972 p.1169

[20] IEEE Trans. on computers vol. C-22 n°3 mars 1973.

[21] R.M. GUFFIN
Micro-diagnostics for the standard computer MLP-900 processor.
IEEE Trans. on comp. vol. C-20 n°7 juillet 1971 p.803

[22] A.B. TUCKER, M.J. FLYNN
Dynamic micro-programming : processor organisation and programming.
CACM vol. 14, n°4, avril 1971 p.240

[23] M.J. FLYNN, R.W. COOK
System design of c dynamic micro processor.
IEEE Trans. on comp. vol. C-19 n°3 mars 1970

[24] J.R. JUMP, D.R. FRITSCHE
Micro-programmed arrays.
IEEE Trans. on comp. vol. C-21, n°9, septembre 1972 p.974

[25] M.V. WILKES, J.B. STRINGER
Micro-programming and the design of the control circuits in an electronic digital computer.
Proc. Cambridge phil. Soc. vol.49 avril 1953 p.230

[26] R.F. ROSIN, G. FRIEDER, R.H.ECKHOUSE
An environment for research in micro-programming and emulation.
CACM vol.15, n°8 août 1972 p.748

GRAPHES

[27] C. BERGE
Graphes et hypergraphes.
Dunod 1970

[28] B. ROY
Algèbre moderne et théorie des graphes.
Dunod 1970

[29] J. KUNTZMANN
Théorie des réseaux.
Dunod 1972.

[30] J.P. STEEN
Théorie des graphes.
UST Lille 1970 Cours de DEA

[31] J.P. STEEN
Séparation et évaluation progressive.
UST Lille 1973

[32] B. ROY
Prise en compte des contraintes disjonctives dans les méthodes de chemin critique.
RFRO n°38 1966 p.69

[33] B.W. KERNIGHAN
Optimal sequential partitions of graphs.
JACM Vol. 18, n°1, janvier 1971 p.34

[34] Von WOLFGANG SCHALLEHN, HANS-JÜRGEN SEBASNAN
Ein Reihenfolgeproblem in der Netzplantechnik.
Elektronische Informationsverarbeitung und Kybernetik
Eik 8 (1972) 1,p.3

[35] J. PAIR
Mille et un algorithmes pour les problèmes de cheminement dans les graphes.
R.I.R.O. B-3 1970 p.125

[36] G. AILLAUD, J.P. STEEN
Armature d'un graphe.
IBM-France. Développement scientifique. Etude n° FF2 - 0149

3 - FLOT BOOLEEN GENERALISE

3.1 Introduction

Un sommet peut, pour une éventualité donnée, avoir plusieurs successeurs et chacun d'eux peut avoir également un ou plusieurs successeurs. Il est donc possible d'avoir plusieurs cheminements simultanés dans le graphe et l'on est amené à étendre le flot booléen entre les sommets d'un graphe à un flot booléen généralisé (f.b.g.) entre sous-graphes d'un graphe.

3.2 Définition

On considère

- un graphe $G = (X,U)$ et
- un recouvrement $R_G = \{G_1, G_2, \ldots, G_n\}$ de G par des sous-graphes partiels avec $G_i = (X_i, U_i)$ $(i \in \{1,2,\ldots,n\})$.

Pour tout couple G_i, G_j de sous-graphes de R_G, on pose $U_{i,j} = U \cap ((X_i/X_j) \times X_j)$ qu'on notera parfois par analogie évidente $U_{i,j} = (G_i, G_i)$.

On note également $\bar{U} = \{U_{i,j} \mid G_i, G_j \in R_G\}$.

On appelle $\bar{E}$ l'ensemble des 2^n fonctions booléennes que l'on peut définir sur un ensemble E de n variables booléennes (moins la fonction identiquement nulle).

Par définition on dira qu'une application Φ de $\bar{U}$ dans $\bar{E}$ détermine un flot booléen généralisé sur G si et seulement si

$$\Phi \text{ est un flot booléen sur } \mathcal{G} = (R_G, \bar{U}).$$

Un flot booléen généralisé sur un graphe régit donc le passage d'un sous-graphe à un autre sous-graphe du graphe. Si sur ce graphe a été défini un flot booléen, peut-on construire un flot booléen généralisé qui "respecte" les conditions de transition du flot simple ?