Test et testabilité de systèmes informatiques

Chantal Robach

THESE

présentée à

Institut National Polytechnique de Grenoble

pour obtenir le grade de
DOCTEUR es SCIENCES
«Informatique»

par

Chantal ROBACH

TEST ET TESTABILITE DE SYSTEMES INFORMATIQUES

Thèse soutenue le 14 Juin 1979 devant la commission d'examen

L. BOLLIET — Président

M. CAMUS
W.C. CARTER
A. COSTES
J. KUNTZMANN
G. SAUCIER
— Examinateurs

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Année universitaire 1982-1983

Président de l'Université : D. BLOCH

Vice-Président : René CARRE
Hervé CHERADAME
Marcel IVANES

PROFESSEURS DES UNIVERSITES :

ANCEAU François	E.N.S.I.M.A.G.
BARRAUD Alain	E.N.S.I.E.G.
BAUDELET Bernard	E.N.S.I.E.G.
BESSON Jean	E.N.S.E.E.G.
BLIMAN Samuel	E.N.S.E.R.G.
BLOCH Daniel	E.N.S.I.E.G.
BOIS Philippe	E.N.S.H.G.
BONNETAIN Lucien	E.N.S.E.E.G.
BONNIER Etienne	E.N.S.E.E.G.
BOUVARD Maurice	E.N.S.H.G.
BRISSONNEAU Pierre	E.N.S.I.E.G.
BUYLE BODIN Maurice	E.N.S.E.R.G.
CAVAIGNAC Jean-François	E.N.S.I.E.G.
CHARTIER Germain	E.N.S.I.E.G.
CHENEVIER Pierre	E.N.S.E.R.G.
CHERADAME Hervé	U.E.R.M.C.P.P.
CHERUY Arlette	E.N.S.I.E.G.
CHIAVERINA Jean	U.E.R.M.C.P.P.
COHEN Joseph	E.N.S.E.R.G.
COUMES André	E.N.S.E.R.G.
DURAND Francis	E.N.S.E.E.G.
DURAND Jean-Louis	E.N.S.I.E.G.
FELICI Noël	E.N.S.I.E.G.
FOULARD Claude	E.N.S.I.E.G.
GENTIL Pierre	E.N.S.E.R.G.
GUERIN Bernard	E.N.S.E.R.G.
GUYOT Pierre	E.N.S.E.E.G.
IVANES Marcel	E.N.S.I.E.G.
JAUSSAUD Pierre	E.N.S.I.E.G.
JOUBERT Jean-Claude	E.N.S.I.E.G.
JOURDAIN Geneviève	E.N.S.I.E.G.
LACOUME Jean-Louis	E.N.S.I.E.G.
LATOMBE Jean-Claude	E.N.S.I.M.A.G.

.../...

LESSIEUR Marcel	E.N.S.H.G.
LESPINARD Georges	E.N.S.H.G.
LONGEQUEUE Jean-Pierre	E.N.S.I.E.G.
MAZARE Guy	E.N.S.I.M.A.G.
MOREAU René	E.N.S.H.G.
MORET Roger	E.N.S.I.E.G.
MOSSIERE Jacques	E.N.S.I.M.A.G.
PARIAUD Jean-Charles	E.N.S.E.E.G.
PAUTHENET René	E.N.S.I.E.G.
PERRET René	E.N.S.I.E.G.
PERRET Robert	E.N.S.I.E.G.
PIAU Jean-Michel	E.N.S.H.G.
POLOUJADOFF Michel	E.N.S.I.E.G.
POUPOT Christian	E.N.S.E.R.G.
RAMEAU Jean-Jacques	E.N.S.E.E.G.
RENAUD Maurice	U.E.R.M.C.P.P.
ROBERT André	U.E.R.M.C.P.P.
ROBERT François	E.N.S.I.M.A.G.
SABONNADIERE Jean-Claude	E.N.S.I.E.G.
SAUCIER Gabrielle	E.N.S.I.M.A.G.
SCHLENKER Claire	E.N.S.I.E.G.
SCHLENKER Michel	E.N.S.I.E.G.
SERMET Pierre	E.N.S.E.R.G.
SILVY Jacques	U.E.R.M.C.P.P.
SOHM Jean-Claude	E.N.S.E.E.G.
SOUQUET Jean-Louis	E.N.S.E.E.G.
VEILLON Gérard	E.N.S.I.M.A.G.
ZADWORNY François	E.N.S.E.R.G.

PROFESSEURS ASSOCIES

BASTIN Georges	E.N.S.H.G.
BERRIL John	E.N.S.H.G.
CARREAU Pierre	E.N.S.H.G.
GANDINI Alessandro	U.E.R.M.C.P.P.
HAYASHI Hirashi	E.N.S.I.E.G.

PROFESSEURS UNIVERSITE DES SCIENCES SOCIALES (Grenoble II)

BOLLIET Louis
Chatelin Françoise

PROFESSEURS E.N.S. Mines de Saint-Etienne

RIEU Jean
SOUSTELLE Michel

CHERCHEURS DU C.N.R.S.

FRUCHART Robert	Directeur de Recherche
VACHAUD Georges	Directeur de Recherche

.../...

ALLIBERT Michel	Maître de Recherche
ANSARA Ibrahim	Maître de Recherche
ARMAND Michel	Maître de Recherche
BINDER Gilbert	
CARRE René	Maître de Recherche
DAVID René	Maître de Recherche
DEPORTES Jacques	
DRIOLE Jean	Maître de Recherche
GIGNOUX Damien	
GIVORD Dominique	
GUELIN Pierre	
HOPFINGER Emil	Maître de Recherche
JOUD Jean-Charles	Maître de Recherche
KAMARINOS Georges	Maître de Recherche
KLEITZ Michel	Maître de Recherche
LANDAU Ioan-Dore	Maître de Recherche
LASJAUNIAS J.C.	
MERMET Jean	Maître de Recherche
MUNIER Jacques	Maître de Recherche
PIAU Monique	
PORTESEIL Jean-Louis	
THOLENCE Jean-Louis	
VERDILLON André	

CHERCHEURS du MINISTERE de la RECHERCHE et de la TECHNOLOGIRE (Directeurs et Maîtres de Recherches, ENS Mines de St. Etienne)

LESBATS Pierre	Directeur de Recherche
BISCONDI Michel	Maître de Recherche
KOBYLANSKI André	Maître de Recherche
LE COZE Jean	Maître de Recherche
LALAUZE René	Maître de Recherche
LANCELOT Francis	Maître de Recherche
THEVENOT François	Maître de Recherche
TRAN MINH Canh	Maître de Recherche

PERSONNALITES HABILITEES à DIRIGER des TRAVAUX de RECHERCHE (Décision du Conseil Scientifique)

ALLIBERT Colette	E.N.S.E.E.G.
BERNARD Claude	E.N.S.E.E.G.
BONNET Rolland	E.N.S.E.E.G.
CAILLET Marcel	E.N.S.E.E.G.
CHATILLON Catherine	E.N.S.E.E.G.
CHATILLON Christian	E.N.S.E.E.G.
COULON Michel	E.N.S.E.E.G.
DIARD Jean-Paul	E.N.S.E.E.G.
EUSTAPOPOULOS Nicolas	E.N.S.E.E.G.
FOSTER Panayotis	E.N.S.E.E.G.

.../...

GALERIE Alain	E.N.S.E.E.G.
HAMMOU Abdelkader	E.N.S.E.E.G.
MALMEJAC Yves	E.N.S.E.E.G. (CENG)
MARTIN GARIN Régina	E.N.S.E.E.G.
NGUYEN TRUONG Bernadette	E.N.S.E.E.G.
RAVAINE Denis	E.N.S.E.E.G.
SAINFORT	E.N.S.E.E.G. (CENG)
SARRAZIN Pierre	E.N.S.E.E.G.
SIMON Jean-Paul	E.N.S.E.E.G.
TOUZAIN Philippe	E.N.S.E.E.G.
URBAIN Georges	E.N.S.E.E.G. (Laboratoire des ultra-réfractaires ODEILLON)
GUILHOT Bernard	E.N.S. Mines Saint Etienne
THOMAS Gérard	E.N.S. Mines Saint Etienne
DRIVER Julien	E.N.S. Mines Saint Etienne
BARIBAUD Michel	E.N.S.E.R.G.
BOREL Joseph	E.N.S.E.R.G.
CHOVET Alain	E.N.S.E.R.G.
CHEHIKIAN Alain	E.N.S.E.R.G.
DOLMAZON Jean-Marc	E.N.S.E.R.G.
HERAULT Jeanny	E.N.S.E.R.G.
MONLLOR Christian	E.N.S.E.R.G.
BORNARD Guy	E.N.S.I.E.G.
DESCHIZEAU Pierre	E.N.S.I.E.G.
GLANGEAUD François	E.N.S.I.E.G.
KOFMAN Walter	E.N.S.I.E.G.
LEJEUNE Gérard	E.N.S.I.E.G.
MAZUER Jean	E.N.S.I.E.G.
PERARD Jacques	E.N.S.I.E.G.
REINISCH Raymond	E.N.S.I.E.G.
ALEMANY Antoine	E.N.S.H.G.
BOIS Daniel	E.N.S.H.G.
DARVE Félix	E.N.S.H.G.
MICHEL Jean-Marie	E.N.S.H.G.
OBLED Charles	E.N.S.H.G.
ROWE Alain	E.N.S.H.G.
VAUCLIN Michel	E.N.S.H.G.
WACK Bernard	E.N.S.H.G.
BERT Didier	E.N.S.I.M.A.G.
CALMET Jacques	E.N.S.I.M.A.G.
COURTIN Jacques	E.N.S.I.M.A.G.
COURTOIS Bernard	E.N.S.I.M.A.G.
DELLA DORA Jean	E.N.S.I.M.A.G.
FONLUPT Jean	E.N.S.I.M.A.G.
SIFAKIS Joseph	E.N.S.I.M.A.G.
CHARUEL Robert	U.E.R.M.C.P.P.
CADET Jean	C.E.N.G.
COEURE Philippe	C.E.N.G. (LETI)

.../...

DELHAYE Jean-Marc	C.E.N.G. (STT)
DUPUY Michel	C.E.N.G. (LETI)
JOUVE Hubert	C.E.N.G. (LETI)
NICOLAU Yvan	C.E.N.G. (LETI)
NIFENECKER Hervé	C.E.N.G.
PERROUD Paul	C.E.N.G.
PEUZIN Jean-Claude	C.E.N.G. (LETI)
TAIEB Maurice	C.E.N.G.
VINCENDON Marc	C.E.N.G.

LABORATOIRES EXTERIEURS

DEMOULIN Eric	C.N.E.T.
DEVINE	C.N.E.T. (R.A.B.)
GERBER Roland	C.N.E.T.
MERCKEL Gérard	C.N.E.T.
PAULEAU Yves	C.N.E.T.
GAUBERT C.	I.N.S.A. Lyon

L'ORDRE

Je mets beaucoup d'ordre dans mes idées.
Cela ne va pas tout seul. Il y a des idées
qui ne supportent pas l'ordre et qui
préfèrent crever. A la fin, j'ai beaucoup
d'ordre et presque plus d'idées.

NORGE

Avant-propos,

Le travail présenté dans cette thèse a été effectué au sein de l'équipe "Conception et Sécurité des Systèmes Logiques" du Laboratoire de Mathématiques Appliquées de Grenoble.

Nous tenons à remercier Monsieur L. BOLLIET, professeur à l'Institut Universitaire de Technologie, pour l'honneur qu'il nous fait de présider notre jury.

Nous sommes très honorée de la présence de Monsieur M. CAMUS, directeur du Centre de Télécommunications de Grenoble, et de Monsieur W.C. CARTER, responsable de recherche au Centre Thomas J. Watson, d'IBM à New-York.

Que Monsieur le professeur J. KUNTZMANN, et Monsieur A. COSTES, Maître de Conférences à l'Institut National Polytechnique de Toulouse, trouvent ici l'expression de notre gratitude pour leur présence à cette commission, l'intérêt qu'ils ont bien voulu accorder à nos travaux et les discussions critiques qui nous ont permis d'améliorer ce travail.

Nous voudrions exprimer notre reconnaissance à Madame G. SAUCIER, Maître de Conférences à l'ENSIMAG, pour ses nombreux conseils, sa disponibilité constante et sa contribution à l'élaboration de cette thèse.

De plus, nous ne saurions dissocier le contenu de cette étude de l'environnement humain qui a permis de la mener à bien : que soient en particulier remerciés Madame C. BELLON, Messieurs L. CASEI et M. DELAUNAY, pour leurs critiques et leur collaboration.

Pour clore cet avant-propos, nous exprimerons nos très sincères remerciements à Mesdames G. DUFFOURD et M.J. DOREL pour leur gentillesse et leur compétence, ainsi qu'à Monsieur D. IGLESIAS et le Service de Reprographie qui ont assuré la réalisation matérielle de cette thèse.

INTRODUCTION

La conception, le développement et la vie opérationnelle d'un système informatique ou automatique induisent plusieurs phases de vérification caractérisées par des moyens et des objectifs de test spécifiques.

On peut schématiser le processus comme suit :

Phase	Vérifications
Etude produit	
. phase de définition ↔ cahier des charges . conception fonctionnelle ↔ synoptique	Validation fonctionnelle de la conception
. conception matérielle ↔ spécifications de définition	Validation structurelle et technologique
. réalisation des prototypes - prototype de définition - prototype de fabrication ↔ spécifications de réalisation	- vérification des dossiers fonctionnels - mise à jour des dossiers matériels
Etude système	
. développement du logiciel	- mise au point du logiciel
Production série	
composants ↔ stockage magasin	Inspection d'entrée des composants
sous-ensembles - interchangeables minimaux (cartes) - fonctionnels	- test de cartes - test des sous-ensembles fonctionnels
intégration ensemble	- banc de test - test de mise au point - banc de recette

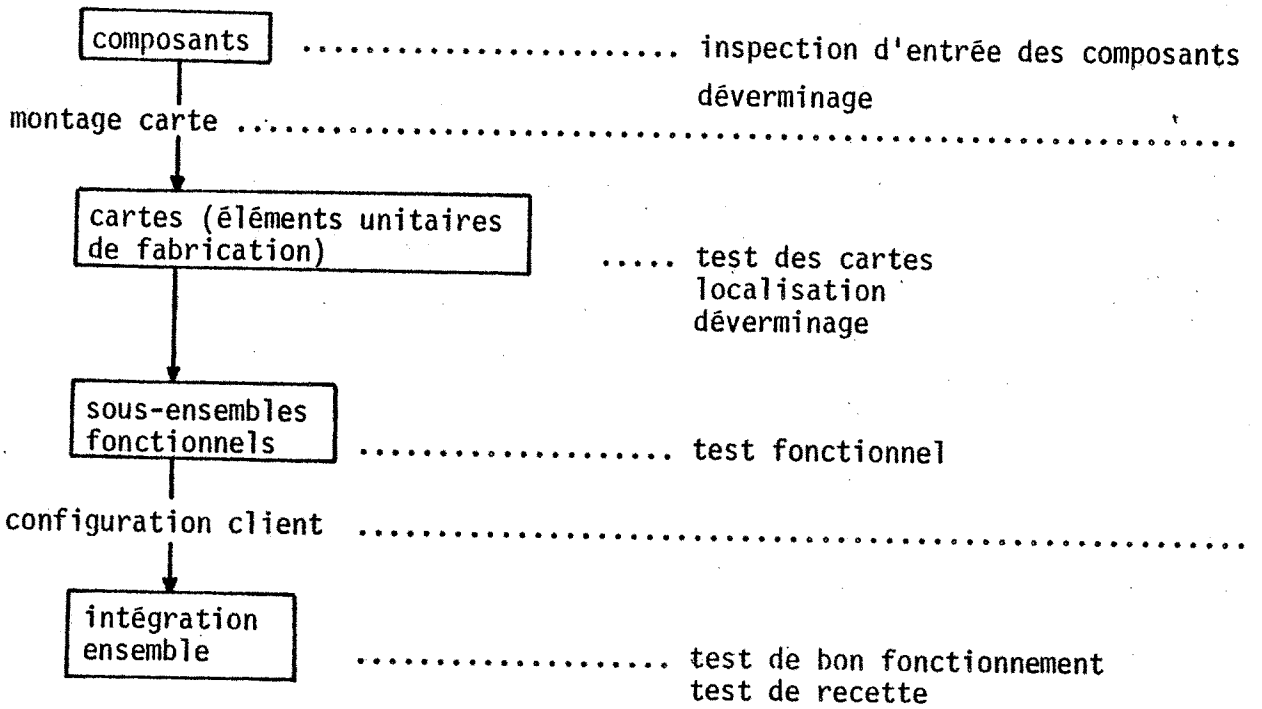
Exploitation	
Première phase : phase opérationnelle	- maintenance en ligne : verdict de bon ou mauvais fonctionnement - prélocalisation éventuelle
Deuxième phase : localisation du sous-ensemble interchangeable ↔ dépose	- maintenance hors ligne - banc de test de sous-ensemble
Troisième phase : déstockage sous-ensemble ↔ rechange	- réparation - vérification sous-ensemble en stock

. Etude du produit : la phase de définition est fondamentale et conduit à établir trois types de dossiers :

- *les dossiers fonctionnels* : manuels d'utilisation, spécifications de réalisation,...
- *les dossiers matériels* qui décrivent les constituants du système et leurs interconnexions,
- *les dossiers commerciaux* : le produit doit correspondre à un besoin; le cahier des charges doit prendre en compte des contraintes d'exploitation, sécurité, fiabilité, maintenabilité, disponibilité, environnement et coût, déterminées par le type d'application.
  Le but de la validation fonctionnelle est de vérifier que la conception réalise les objectifs définis par le cahier des charges; la validation matérielle doit qualifier la conception technologique.

. Etude système : l'équipement est intégré dans son environnement : utilisation des entrées/sorties, et le logiciel d'application est développé : production des programmes, mise au point du logiciel en environnement réel.

. Production : les différentes étapes de la fabrication peuvent être représentées par l'organigramme suivant :



. Phase d'exploitation : cette phase comporte plusieurs étapes :

- *La maintenance en ligne*, en phase opérationnelle, doit donner un verdict de bon ou mauvais fonctionnement.
  Elle peut être assurée :
  . soit en cours d'exécution du processus normal par une vérification logicielle; les pannes détectées appartiennent à une classe donnée de pannes; la détection est dite continue,
  . soit en oisiveté : la détection est dite discontinue.

  Les caractéristiques de la maintenance en ligne sont les suivantes :
  * le programme de test doit s'insérer dans les cycles du programme opérationnel (contraintes de temps);
  * c'est un auto-test : le système se teste lui même;
  * le programme doit être résident dans le matériel opérationnel (contrainte de place mémoire).

- *La maintenance hors ligne* :
  . le premier niveau doit confirmer le verdict de la maintenance en ligne ou être une vérification préventive périodique. Il est exécuté sur le site opérationnel;

Ses caractéristiques sont les suivantes :

- c'est un programme d'auto-test
- le programme peut être résident ou chargé de l'extérieur
- il y a peu de contraintes en temps d'exécution ou en volume mémoire.

. au deuxième niveau, le système défectueux est envoyé sur le site de test (qui peut être le site opérationnel dans certains cas). Le programme de maintenance doit localiser le sous-ensemble interchangeable défectueux si cette localisation n'est pas faite au premier niveau, puis vérifier le bon fonctionnement de l'ensemble réparé. Ce niveau induit des moyens de test extérieurs au système à tester.
Le déstockage : le programme de test doit faire une localisation à l'intérieur de cet élément interchangeable et vérifier le bon fonctionnement du sous-ensemble réparé ou déstocké.

La conception d'un système facilement testable induit donc une harmonisation entre ces différentes phases. L'utilisation de composants à large échelle d'intégration (LSI) réduit considérablement la résolution de diagnostic requise. En outre, la réduction des points d'accès à l'intérieur de ces composants complexes rend le processus de test et de localisation plus difficile. Dans une telle situation, il est nécessaire d'intégrer les caractéristiques de la maintenance dans la conception du matériel. On fait alors appel à des techniques de tolérance aux pannes "intégrée", c'est-à-dire des techniques qui sont appliquées pendant la conception et non après la fabrication.

Un aspect important de ces techniques est constitué par la modélisation des systèmes en vue du diagnostic avec une évaluation de la testabilité. C'est cet aspect qui est étudié dans le Chapitre I.

Parmi les différents constituants d'un système il en est qui posent des problèmes particuliers et peu abordés : ce sont les organes de contrôle des systèmes considérés. Alors que les méthodes de test pour les parties opératives ont été largement développées et utilisées avec succès, les parties contrôle posent des problèmes certains. Le chapitre II donne une solution au test d'organes de contrôle à travers le système.

Enfin, parmi les composants complexes utilisés dans la fabrication des systèmes logiques, les microprocesseurs jouent un rôle particulier : de par leur utilisation de plus en plus importante d'une part, de par leur complexité d'autre part. Leur haut niveau d'intégration ainsi que les nouvelles technologies qu'ils utilisent, font que les méthodes de test classiques de circuits ne sont guère utilisables. Le problème du test des microprocesseurs et des systèmes à microprocesseurs est étudié dans le chapitre III.

CHAPITRE I

MODÈLE DE TESTABILITÉ D'UN SYSTÈME LOGIQUE

INTRODUCTION

Le besoin se fait sentir d'évaluer a priori pour un matériel sa testabilité. On entend par testabilité la facilité d'assurer la maintenance du système, c'est-à-dire la facilité que l'on aura de réaliser le test complet et la localisation des pannes à l'unité remplaçable près.

De même que les autres paramètres de conception, la testabilité doit être abordée en formalisant le système de ce point de vue, autrement dit en définissant un modèle de testabilité.

Le modèle de testabilité proposé permet :

- de faire l'inventaire des chemins de test possibles pour le test complet de chaque module, pour un modèle de pannes donné ;
- de proposer des stratégies de test et de localisation ;
- de classer les modules en blocs d'indiscernabilité, c'est-à-dire en ensembles dont les pannes ne pourront être distinguées par les stratégies ;
- de proposer des modifications de type implantation et points de test en entrée ou en sortie, pour permettre le test de certains modules ou pour améliorer éventuellement la finesse du diagnostic, sachant que le but à atteindre est la localisation d'une panne à l'unité remplaçable près.

Ce modèle est complété par une analyse de la quantité d'information dans le réseau. Cette analyse s'appuie sur une théorie des réseaux porteurs d'information qui permet de définir des mesures de :

- *commandabilité* c'est-à-dire la quantité d'information maximum qu'un module peut recevoir par un chemin de test à partir des entrées du système.
- *observabilité* c'est-à-dire la quantité d'information maximum qu'un module peut transmettre aux points de sortie du système.
- *testabilité* c'est-à-dire une mesure informationnelle de la facilité de génération du test d'un module par un chemin de test donné. Cette mesure est étendue à l'ensemble des modules appartenant à un chemin de test, puis à l'ensemble des chemins de test du réseau, pour une stratégie de test donnée.

Ces mesures permettent de discriminer des stratégies de test qui apparaissent équivalentes au terme de l'analyse sur le modèle non paramétré, donc de choisir la politique qui paraît a priori optimale et de proposer des modifications pour faciliter le diagnostic dans le système.

On aboutit ainsi à un ensemble d'outils d'évaluation de la testabilité d'un système : programmes de mise en oeuvre automatique de recherche de chemins de test dans un réseau et de calcul de la testabilité d'un système.

SECTION 1 : L'ETAT DE L'ART

I - POSITION DU PROBLÈME

Lors de la conception d'un système informatique ou automatique, un paramètre qui prend de l'importance du fait de la complexité des systèmes envisagés est sa testabilité, c'est-à-dire la facilité de détecter et de localiser les pannes matérielles pouvant survenir dans le système opérationnel.

I - 1. NECESSITE D'UN MODELE

Deux cas sont à envisager :

- soit le système est en cours de conception : le test doit être pris en compte alors qu'on ne connaît, par exemple, du système que ses propriétés architecturales ; il faut alors pouvoir représenter les relations de test du système encore incomplètement précisé ; le modèle, à ce niveau, doit permettre de définir la meilleure stratégie de test, compte tenu d'autres critères tels que coût, performances, ... [BEL,78]
- soit le système est déjà réalisé (il pourra être figé ou modifiable) : en général, sa complexité est telle que la réalisation d'un programme de maintenance doit passer par une phase où l'on s'abstrait du détail. On a besoin d'une simplification caractéristique du niveau où l'on se trouve dans le test : politique de maintenance, méthodes de diagnostic, élaboration des vecteurs de test proprement dits [ROB,76].

I - 2. QU'EST-CE QU'UN MODELE DE TESTABILITE ?

C'est une représentation simplifiée du système qui ne met en évidence que les relations entre éléments du système qui sont :

- soit des contraintes à respecter
- soit des choix possibles

dans l'élaboration d'un programme de maintenance, à savoir :

- contraintes de fonctionnement : cheminement des données et séquencement des opérations
- contraintes et possibilités de génération d'une information en entrée d'un élément donné (commandabilité)
- contraintes et possibilités d'observation d'une information à la sortie d'un élément donné (observabilité).

Selon le niveau auquel on se place, le degré de simplification sera différent : introduction de renseignements supplémentaires lors du passage d'un niveau donné à un niveau plus fin.

I - 3. OBJECTIFS D'UN MODELE DE TESTABILITE

Le modèle, aux différents niveaux, doit servir à prévoir les performances du produit final (le programme de maintenance, la testabilité du système réel) et son coût, sans que celui-ci et le système sur lequel il porte soient entièrement spécifiés. Par performances, on entend des qualités telles que la finesse de localisation, le taux de couverture de pannes, le comportement dans le cas de pannes multiples ; le coût faisant référence à la difficulté de génération du programme et de ses arguments (séquences de test).

L'utilité de modèles atteignant de tels objectifs est évidente : ils permettent d'une part d'organiser de façon descendante le programme de maintenance de façon à obtenir des performances et des coûts optimaux - et ce, au cours de la conception même du système - et d'autre part d'infléchir cette conception ou de proposer des modifications lorsque les performances ou les coûts attendus s'éloignent de ceux imposés par le cahier des charges.

Il convient, pour ce faire, que les modèles aient une bonne capacité prédictive et que les renseignements négligés à un certain niveau ne remettent pas en cause ultérieurement les conclusions que fournit le modèle à ce niveau. En général, ce sera cependant partiellement le cas : au fur et à mesure que le système se précise, on devra remonter au niveau précédent pour modifier le modèle.

I - 4. PLAN DU CHAPITRE

On considère la modélisation des systèmes logiques face au test (au sens le plus général de test : mesures de testabilité, politiques de maintenance et de diagnostic, élaboration des séquences de test) en supposant que certains choix d'architecture sont déjà induits par d'autres paramètres lors de la conception.

En Section 1, on rappellera d'abord les niveaux de description d'un système (§ II) : en effet, il apparaît évident de lier le niveau de description d'un système à son niveau de modélisation face au test, puisque le modèle de testabilité ne peut être construit qu'à partir des données descriptives connues du système. On fera ensuite (§ III) l'analyse et la critique des modèles de testabilité connus avec les objectifs qu'ils atteignent.

En Section 2, on définira un modèle de testabilité, utilisant des graphes bipartis ; on montrera en quoi il diffère des modèles connus et ce qu'il apporte par rapport à ces derniers.

La Section 3 indique comment ce modèle est utilisé dans le contexte de la maintenance ; on indiquera d'abord comment il permet de choisir une stratégie de diagnostic ; ensuite, comment l'adjonction au modèle d'une information structurelle permet de prévoir la testabilité du système.

II - TYPES DE DESCRIPTION ET DE MODÉLISATION DES SYSTÈMES

II - 1. CLASSIFICATION DES TYPES DE DESCRIPTION [BAR,76a]

On peut considérer trois types de description d'un système :

- une *description de comportement* où l'on ne connaît du système que les relations d'entrée/sortie, le système lui-même étant considéré comme une "boîte noire" ; il est alors caractérisé par une propriété sur ses entrées/sorties, l'intention étant seulement de montrer son comportement par rapport à une caractéristique donnée (file d'attente, test)

- une *description structurelle* où le système est donné en termes composants matériels réels ; ce type de description est proche de l'implémentation physique : les éléments et les interconnexions entre ces éléments sont connus. Un élément, appelé aussi module, est une entité physique qui peut aller de la porte logique à l'armoire

- une *description fonctionnelle* (niveau intermédiaire) où le système est décrit en termes de registres et composants avec leur relation fonctionnelle ou algorithmique.

Pour un type de description donné, on peut considérer essentiellement quatre niveaux de représentation d'un système :

- le *niveau système* pour lequel les éléments sont des processeurs ou ordinateurs
- le *niveau sous-système* où sont seulement connus les éléments macroscopiques du système : mémoires, unités périphériques, unités arithmétiques et logiques, ...
- le *niveau transfert registre* (niveau RT) pour lequel les éléments sont des registres, ... et où les fonctions sont les transferts entre registres et les opérations sur les données (fonctions de décalage, d'addition, ...)
- le *niveau logique* où la structure du système est donnée par l'ensemble des portes logiques, bascules, ... et son comportement est exprimé par des équations booléennes.

II - 2. RELATIONS ENTRE LE TYPE DE DESCRIPTION ET LE NIVEAU DE REPRESENTATION

On peut noter une relation entre le type de description et le niveau de représentation d'un système :

- une description de comportement est particulièrement adaptée pour une machine considérée aux niveaux système ou sous-système puisque ne sont mises en jeu que des caractéristiques de comportement
- une description fonctionnelle pourra être donnée au niveau transfert registre pour lequel la fonction ou algorithme met en jeu des éléments sans que soit connue la structure interne de ces éléments
- une description structurelle pourra être considérée :
  - α) soit au niveau Transfert Registre : la description est proche du matériel et les interconnexions entre modules sont connues ; en outre, une propriété structurelle peut être donnée sur les modules
  - β) soit au niveau logique lorsque la structure interne et l'implémentation physique sont connues en détail (circuits élémentaires ou portes et leurs interconnexions physiques).

Le tableau suivant récapitule les relations entre le type de description et le niveau de représentation :

Description / Représentation	de comportement	fonctionnelle	structurelle
Système (ordinateur)	Architecture : détermination d'un ensemble cohérent d'actions selon les objectifs : type de calcul (gestion, calcul scientifique, graphique, ...)	Algorithmes et décomposition en fonctions exécutables	- jeu d'instructions - organisation : mémoires, unités de traitement, type de contrôle, communications
Sous-sytème (UAL)	Traitement des données : analyse et transformation	Différents types d'opérations (addition, soustraction, décalage,...) avec les besoins en registres et composants	Organisation : nombre de registres, additionneur, contrôle local ; interconnexions entre ces éléments; format (registres, bus,...)
Transfert registre (séquenceur de multiplication)	Multiplication entre 2 registres : $A \times B \rightarrow C$	- algorithme de multiplication - précision sur le registre résultat (C = A ou B)	Structure interne : bascules, compteur de cycle ... définition des registres A, B, C

II - 3. TYPES DE MODELISATION UTILISES POUR LE TEST

Un nombre important de travaux a été proposé ces dernières années sur la modélisation de systèmes logiques vis-à-vis du test.

Deux aspects sont pris en compte lors de la modélisation :

- l'aspect qualitatif, c'est-à-dire le modèle de testabilité proprement dit
- l'aspect quantitatif, c'est-à-dire une mesure sur le modèle.

Un modèle de testabilité d'un système est donné par un graphe

- dont les noeuds sont des modules (entités matérielles ou fonctionnelles)
- dont les arcs représentent les liaisons entre ces modules (liaisons de test, liaisons topologiques, ...).

L'aspect quantitatif se traduit par une mesure qui est associée soit au module, soit à la liaison, soit aux deux.

Parallèlement aux types de description d'un système, on peut définir trois types de modélisation :

- *modélisation structurelle* : un modèle structurel est obtenu à partir d'une description structurelle du système en définissant un module comme un circuit logique ayant une fonction bien définie (ou un ensemble de circuits logiquement ou physiquement dépendants).
Les liaisons entre modules sont des liaisons purement topologiques et représentent les lignes physiques entre les composants du système.

- *modélisation fonctionnelle* : dans un modèle fonctionnel, un module est une entité matérielle et la liaison est une liaison topologique renseignée
  . soit par le fonctionnement du système : ce renseignement est fourni par une description fonctionnelle du système (séquencement, algorithme)
  . soit par une propriété de test.

- *modélisation de comportement* : dans ce type de modèle
  . un module est une entité physique de haut niveau (niveau système : processeur, ... ; niveau sous-système : mémoire, ...), fourni par une description de comportement du système
  . une liaison représente une propriété par rapport au test et peut être éloignée de la signification topologique.

On peut noter que, lorsqu'il n'y a pas interaction de niveau, le passage d'un modèle de haut niveau à un modèle de niveau plus fin se caractérise par la suppression d'un certain nombre de liaisons, non significatives pour le niveau de test considéré (en particulier, perte de renseignements topologiques).

II - 4. LE TYPE DE DESCRIPTION ET LE NIVEAU DE REPRESENTATION D'UN SYSTEME FACE AU TEST

Le tableau suivant montre le type de description et le niveau de représentation d'un système mis en jeu aux différents stades du test (génération des vecteurs de test, politiques de diagnostic, testabilité, politiques de maintenance : réparation, reconfiguration).

Description / Représentation	de comportement	.fonctionnelle	. structurelle,
Système	. Etude de la testabilité	. Validation fonctionnelle . Politique de diagnostic (multiple-clue)	(trop complexe)
Sous-système	. Mesure de testabilité	. Test fonctionnel aléatoire (fonction activée n fois) . Politique de test (multiple-clue)	. Génération de vecteurs de test (définition de méthodes de test) . Mesure de testabilité . Politique de test (start, small, multiple-clue
Transfert registre ou logique	. Sans intérêt (pas de propriété architecturale définie à ce niveau)	. Test fonctionnel (identification d'automates)	. Méthodes de test analytiques ; génération de vecteurs de test

III - CLASSIFICATION ET ÉTUDE DES MODÈLES EXISTANTS

III - 1. MODELES DE COMPORTEMENT [AGN,65 - BAR,75,76b - FOR,65 - FRI,75 - KIM,70 - PRE,67,68 - RAN,67 - RUS,73a-b,75a-b - VAN,72]

Un nombre important d'études ont porté sur la "diagnosticabilité" des systèmes, c'est-à-dire la capacité de détecter et localiser un nombre donné de modules défectueux (simultanément ou non) parmi les n modules du système, sachant qu'il y a au plus t < n modules défectueux.

Ces études ont été motivées par le besoin de systèmes hautement fiables qui pourraient continuer à fonctionner, même avec des performances réduites, quand des pannes matérielles surviennent. Cette approche vers de tels systèmes - via la reconfiguration et la réparation - exige que la présence d'éléments défectueux soit détectée et que la localisation soit faite à l'élément remplaçable près.

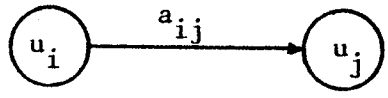
. La plupart des modèles de ce type font une partition du système en unités indépendantes telles que chaque unité puisse être testée par une combinaison d'autres unités. Le test complet suppose donc que l'ensemble qui teste peut contrôler et générer les vecteurs de test du sous-système testé de même qu'analyser les résultats de test, afin de conclure au bon ou mauvais fonctionnement de l'unité testée.

Cependant, si certaines des unités qui testent une autre unité sont elles-mêmes défectueuses, alors l'unité testée peut être déclarée correcte alors qu'elle ne l'est pas et vice versa.

Les modèles de ce type sont tous basés sur le modèle de Preparata [PRE,67] qui considère uniquement des systèmes où chaque unité est capable, à elle seule, de tester une autre unité. Des extensions de cette étude ont été données dans [PRE,68 - SES,71] ; des études similaires [VAN,72] ont également été proposées pour des systèmes où plus d'une unité est nécessaire pour tester une autre unité.

Ces modèles représentent le système par un graphe dont

- les noeuds sont des unités indépendantes
- les arcs représentent les relations de test entre les unités.



u_i teste u_j

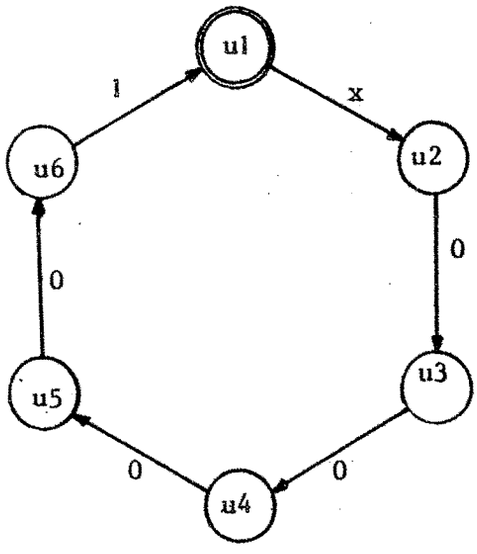
Dans l'hypothèse où u_i est correct

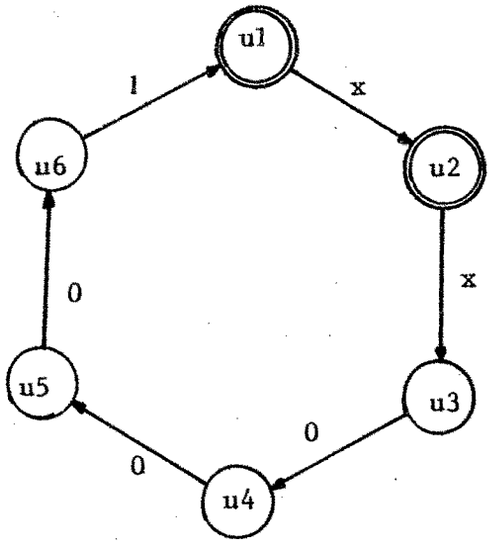
$a_{ij} = 0$ indique que u_j est correct

$a_{ij} = 1$ indique que u_j est défectueux

Dans le cas où u_i est défectueux, le résultat de test n'est pas un renseignement sûr ($a_{ij} = x$, avec $x = 0$ ou 1).

Exemple:





où  dénote l'unité défectueuse.

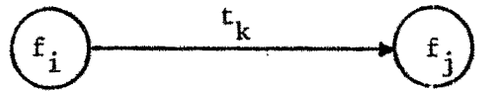
. Le modèle de Kime et Russel [RUS,73a-b, 75a-b] étend et précise les relations de test entre unités en faisant intervenir la notion d'invalidation du test ; plutôt que de décrire le système en termes d'unités testant d'autres unités, le système est décrit en termes de fautes, de tests et de relations entre eux.

Le terme de "faute" est défini comme toute condition causant le mauvais fonctionnement d'une unité du système ; le terme de "test" est entendu comme toute combinaison de procédures matérielles ou logicielles détectant une faute.

L'hypothèse est faite qu'il ne peut y avoir qu'une seule faute par unité (autrement dit à chaque faute est associée une unité ou module).

Le système est alors modélisé par un graphe dont

- les noeuds représentent les fautes possibles
- une liaison entre un noeud f_i et un noeud f_j, indicée par un test t_k, indique que la présence de la faute f_i dans le système invalide le test t_k (t_k est un test complet de f_j)



Le noeud d'un graphe, défini comme une "faute possible" est en fait considéré comme un "module défectueux".
Le modèle suppose qu'un test t_k est un test complet : ceci n'est pas toujours vrai en pratique et affaiblit le renseignement sur la diagnosticabilité du système (elle risque d'être surévaluée).

. Un modèle plus récent proposé par Hayes [HAY,76] est un modèle de comportement élargi. Un noeud du graphe représente une ressource (au sens large), matérielle ou logicielle et les arcs représentent les relations d'accès. L'auteur étudie, à l'aide de méthodes de théorie des graphes, la possibilité pour le système de continuer à exécuter des algorithmes après occurrence d'une panne.

III - 2. MODELES FONCTIONNELS [CHA,74]

Le modèle proposé dans [CHA,74] est représentatif de ce type de modélisation.

Le système est représenté par un graphe paramétré dont

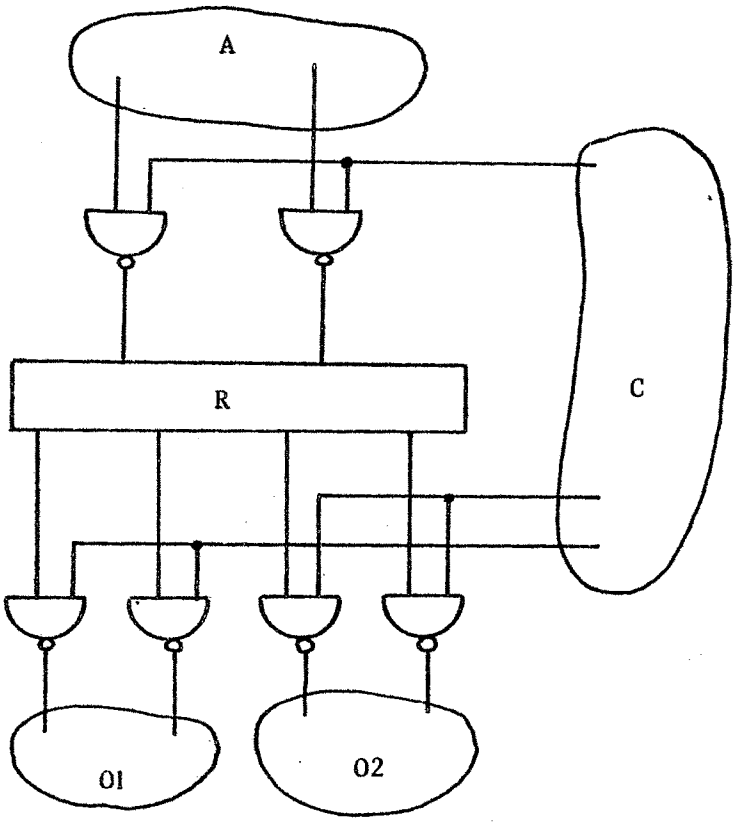
- les noeuds sont des entités fonctionnelles, c'est-à-dire des circuits logiques (ou ensembles de circuits) ayant une fonction bien définie : additionneur, circuit de décalage, ...
- les arcs sont des liaisons topologiques renseignées par le fonctionnement du système ; on distingue
  - . les relations de contrôle et d'accès
  - . les relations d'observation.

Un ensemble de relations est alors défini sur l'ensemble des modules du système :

. le noeud X contrôle le noeud Y, noté X $\Rightarrow$ Y si Y requiert le contrôle ou les données de X pour être totalement testé

. le noeud X' observe le noeud Y', noté X' $\rightarrow$ Y' si X' est requis pour observer les résultats de test de Y'.

Exemple



relations de contrôle et d'accès:

$A \Rightarrow R$

$C \Rightarrow R$

relations d'observation:

$O1 \rightarrow R$

$O2 \rightarrow R$

Cet ensemble de relations est représenté par un graphe ; au moyen d'outils de théorie des graphes, un algorithme a été développé pour permettre d'organiser (ou réorganiser) la conception du système considéré afin d'augmenter sa testabilité ; il met en jeu les points suivants :

- l'ordonnancement des modules vis-à-vis d'une politique de diagnostic donnée
- l'implémentation physique : procédure d'organisation des modules fonctionnels en unités de remplacement
- l'adjonction de points de test, c'est-à-dire de liaisons supplémentaires d'accès et/ou d'observation.

Ce modèle est proche de la structure matérielle du système considéré et introduit deux notions importantes :

- la notion d'implantation physique des modules en unités remplaçables permettant une bonne politique de maintenance (problème de la réparation)
- la notion de distinction entre contrôle-accès et observation, induisant des choix sur la génération des séquences de test et sur la politique de diagnostic. On notera toutefois les inconvénients suivants :

. quoique proche du matériel, le modèle ne prend pas en compte de façon satisfaisante le problème des fan-in/fan-out en entrée/sortie des modules

. les lignes de données et les lignes de commande ne sont pas différenciées ; ceci implique, sur le graphe de testabilité, une relation $\Rightarrow$ du module de contrôle vers l'ensemble des autres modules, ce qui rend difficile l'analyse du graphe

. au niveau du test proprement dit, la solution de déconnexion physique ou logique induit de fortes contraintes (test impossible en fonctionnement normal).

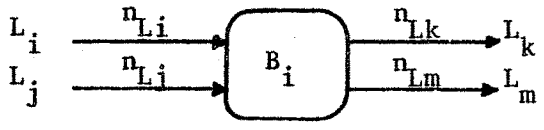
III - 3. MODELES STRUCTURELS [AKE,76 - STE,76]

Le graphe de testabilité d'un modèle structurel est défini par :

- les noeuds qui sont des entités matérielles (quel que soit le niveau)
- les arcs qui sont des liaisons topologiques entre les modules (fils d'information).

Deux études sont représentatives de ce type de modèle ; elles utilisent un graphe paramétré où le paramètre est un renseignement de test qui dépend de la structure matérielle du module (niveau logique).

A) Le modèle proposé par Stephenson et Grason [STE,76] représente le système par un ensemble de circuits (combinatoires ou séquentiels : additionneur, multiplexeur, contrôle ...) interconnectés par des lignes d'information.
L'objectif du modèle est de calculer une mesure de testabilité du système considéré, c'est-à-dire une mesure de la facilité de générer les séquences de test du système, basée sur la structure du système et le comportement des modules. Ce modèle peut être appliqué au niveau transfert registre ou au niveau logique.
Le système est donc décrit par des modules B_i et des lignes L_i auxquelles sont attachés les nombres de fils de ces lignes soit n_{Li}, reliant ces modules :



Pour chaque module B_i, les auteurs définissent deux coefficients calculés à partir de la fonction réalisée par B_i :

- le facteur de commande CFT_i qui représente la difficulté de générer une valeur en sortie de B_i à partir des valeurs d'entrées ; le CFT_i est donné comme moyenne pondérée de l'écart entre le nombre de valeurs d'entrée produisant une valeur de sortie donnée et la moyenne de ce même nombre. Ce facteur est compris entre 0 et 1, 1 représentant une commandabilité parfaite.

- le facteur d'observation OTF_i qui représente de même la difficulté de discerner si une valeur erronée est présente en entrée de B_i, à partir des valeurs de sortie.

La commandabilité C_i est obtenue par moyenne pondérée des commandabilités des lignes d'entrées C_{Li}.

Pour l'exemple de la figure précédente, on obtient:

$$C_i = \frac{n_{Li}\ C_{Li} + n_{Lj}\ C_{Lj}}{n_{Li} + n_{Lj}}$$

La commandabilité des lignes de sortie est considérée comme égale pour toutes les lignes et s'écrit, pour l'exemple donné :

$$C_{Lk} = C_{Lm} = C_i . CTF_i$$

On a pour l'observabilité des relations analogues :

$$- O_i = \frac{n_{Lk}\ O_{Lk} + n_{Lm}\ O_{Lm}}{n_{Lk} + n_{Lm}}$$

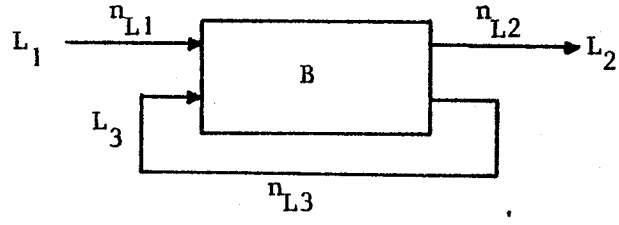
$$- O_{Li} = O_{Lj} = OTF . O_i$$

Des formules spéciales permettent de traiter les divergences de lignes.
Soit un système comportant N modules B_i, i = 1, N.
On cherche à calculer les commandabilités C_i et les observabilités O_i de tous les modules. Ces valeurs s'obtiennent en résolvant deux systèmes linéaires de N équations à N inconnues.

Les commandabilité C et observabilité O du système complet sont les moyennes des C_i et des O_i. Enfin, la testabilité T du système global est définie par $T = \sqrt{C.O}$.

α) Exemple de calcul

Nous allons donner un exemple simple de tels calculs, exemple qui illustre certaines insuffisances de cette méthode ; soit le circuit séquentiel suivant :



B est défini par ses CTF et OTF.
Sa commandabilité C est donnée par

$$C = \frac{n_{L1}.C_{L1} + n_{L3}.C_{L3}}{n_{L1} + n_{L3}}$$

$C_{L1} = 1$ puisqu'il s'agit d'une entrée primaire.
C_{L3} est donnée par $C_{L3} = C.CTF$.
C est donc la solution du système à une équation et une inconnue :

$$C = \frac{n_{L1} + n_{L3}.C.CTF}{n_{L1} + n_{L3}}$$

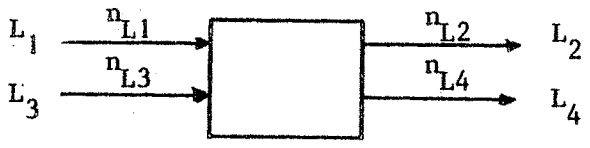
De même O est la solution de l'équation

$$O = \frac{n_{L2} + n_{L3}.O.OTF}{n_{L2} + n_{L3}}$$

Un exemple de valeurs numériques qui posent un problème peut être obtenu en posant CFT = OTF = 1, correspondant à un boîtier B réalisant une fonction bijective. En appliquant ces valeurs, on trouve

$$C = O = T = 1.$$

Il s'ensuit que le circuit doit être considéré comme très facile à tester et en tout cas aussi facile à tester que le circuit combinatoire correspondant donné ci-après:



Or, il semble clair que le test du circuit séquentiel sera plus coûteux que celui du circuit combinatoire.

Il y a une faiblesse dans cette approche de la testabilité, faiblesse sans doute due au caractère empirique de la formulation adoptée.

β) Résultats expérimentaux

Les auteurs ont procédé à une étude expérimentale de leur coefficient de testabilité, étude qui montre son intérêt pratique : ils ont considéré un certain nombre de circuits de complexité variée pour lesquels ils ont calculé la testabilité T_i. Ils ont procédé à la génération automatique de séquences de test en notant le temps de calcul t_i (CPU) pour chaque circuit. Enfin, ils ont approximé le nuage (t_i, T_i) par régression linéaire à l'aide de la fonction $\text{Log } t_i' = p \text{ Log } T_i + \text{Log } a$, trouvant un coefficient de corrélation significatif au seuil de 10 %. Le résultat de cette régression est donné dans la figure 1.1.

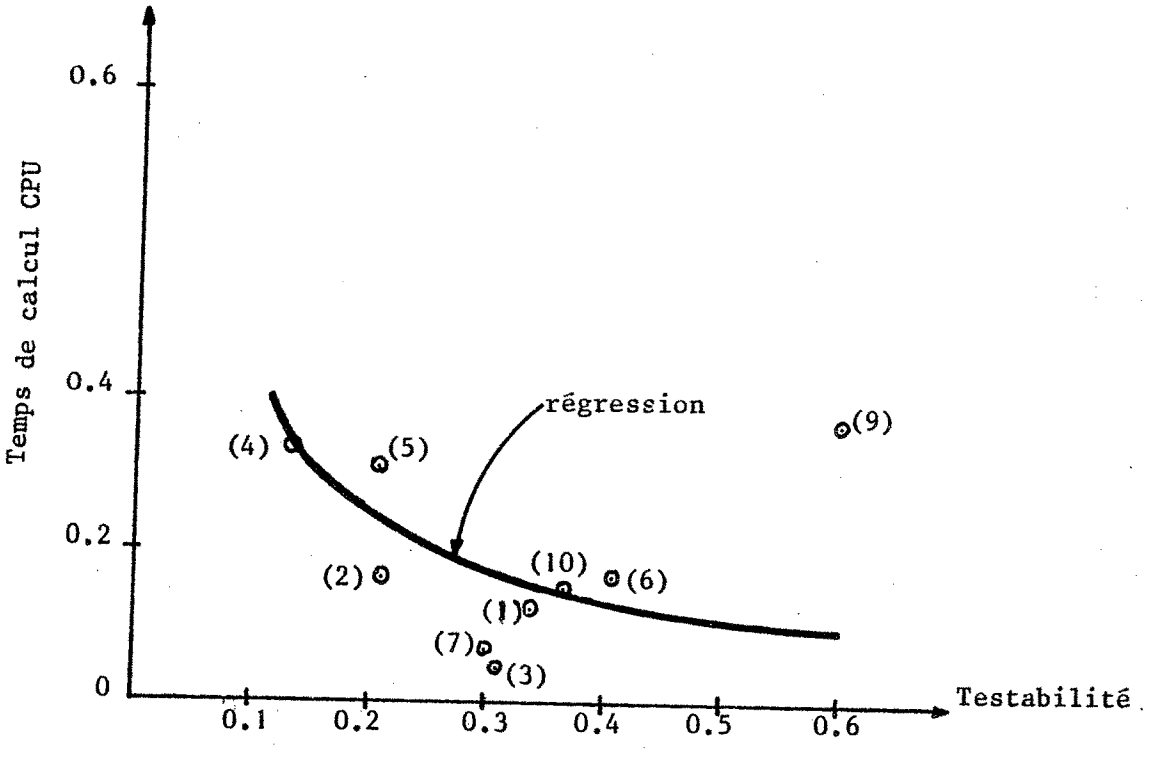


Figure.1.1. Résultats expérimentaux

Ce résultat montre donc que l'on peut utiliser ce coefficient de testabilité pour prévoir la difficulté d'élaboration d'un test, chercher des modifications (points de test) améliorant la testabilité, et même guider des algorithmes du type chemin sensible en les aiguillant vers les voies les plus commandables ou les plus observables.

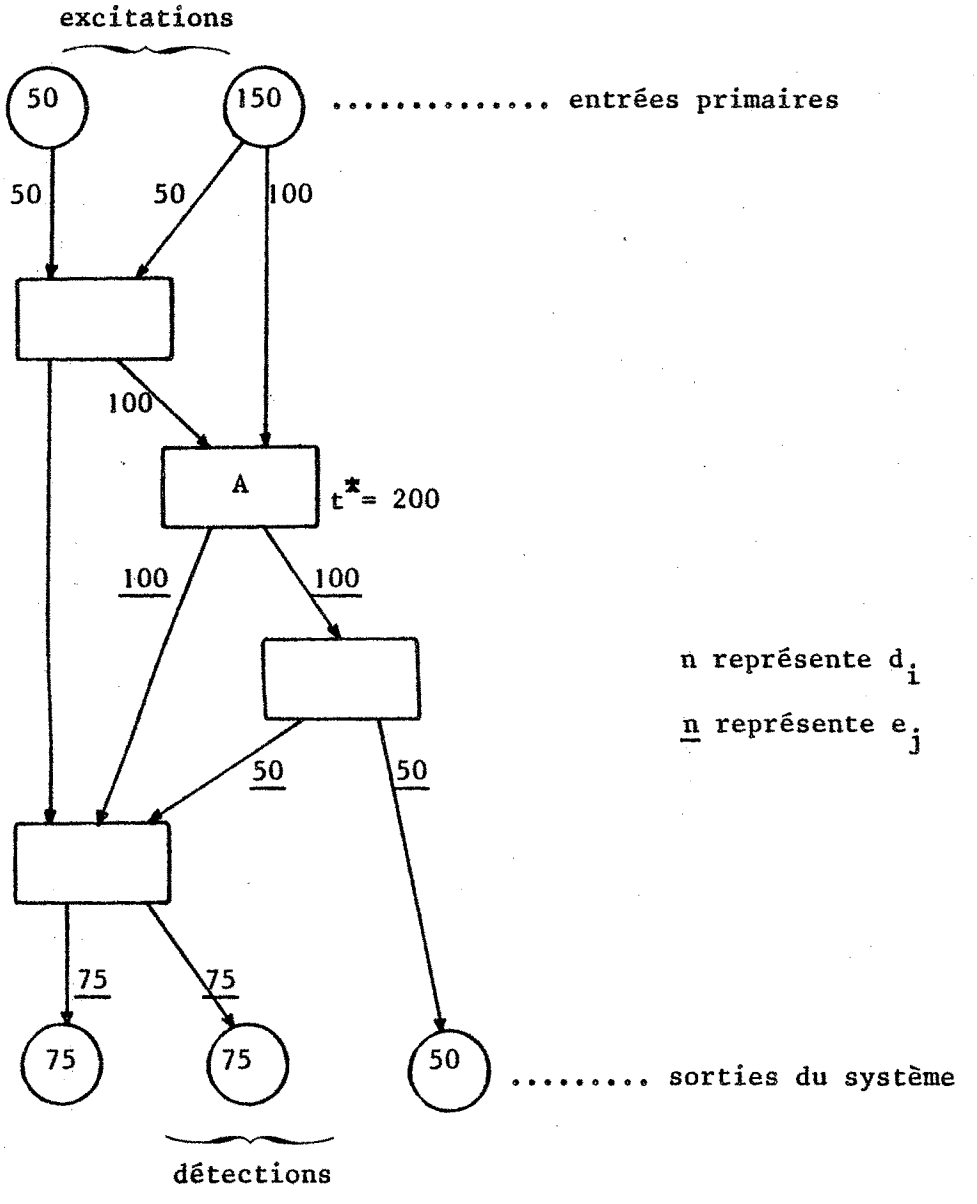
B) Le modèle présenté par Akers [AKE,76] représente le système par un graphe paramétré dont les modules sont des entités matérielles de complexité quelconque (allant de la porte à l'armoire).

A chaque module est associée une mesure t^* qui reflète sa testabilité : l'auteur considère que cette mesure peut être aussi bien le nombre de vecteurs de test nécessaires pour tester ce module avec un pourcentage de détection donné que le simple décompte de composants ou de portes dans le module.

Le graphe a pour but de représenter le processus de test : pour chaque module, un ensemble de t^* entrées de test ou excitations sont générées à l'entrée de ce module à partir des entrées primaires du système et un ensemble de t^* sorties de test ou détections sont propagées vers les sorties du système pour être analysées.

Chaque liaison est alors indicée par deux coefficients d_i et e_j qui indiquent respectivement le nombre de détections (issues de m_i) et le nombre d'excitations (en entrée de m_j) déduits des mesures t_i^* et t_j^*.

Exemple :



Ce processus est répété pour chacun des modules du système ; l'analyse du graphe permet alors de partitionner le système en sous-systèmes (ensembles de modules) ayant une charge de test (nombre de détections ou d'excitations) à peu près équivalente de manière à améliorer le test des ensembles résultants ; cette partition se fait au moment de la conception en vue d'une bonne implantation : les ensembles seront implantés sur des cartes différentes et testés individuellement, hors fonctionnement normal.

Cette modélisation permet aussi l'insertion de points de test pour répartir la charge de test globale aussi uniformément que possible entre les entrées primaires et les sorties primaires des ensembles.

Néanmoins les hypothèses de cette étude:

- graphe acyclique,
- répartition uniforme des détections sur les sorties des modules,
- répartition uniforme des excitations sur les entrées des modules,
- répartition uniforme du flot de test entre les modules,

sont très restrictives et rendent l'applicabilité de ce modèle réduite (niveau grossier de représentation). D'autre part, la mesure de testabilité d'un module donnée comme le nombre de vecteurs de test du module, ne réflète pas réellement la facilité de tester ce module (accessibilité et observabilité de l'information de test négligées, problème de masquage, perte d'information).

SECTION 2 : PROPOSITION D'UN MODELE DE TESTABILITE

I - INTRODUCTION

I - 1. POSITION DU PROBLEME

On se place dans le contexte général de la validation matérielle d'un système : vérification en fin de fabrication, maintenance hors ligne.
La validation matérielle d'un système a pour but de détecter la présence de pannes dans ce système et de les localiser, au minimum à l'unité remplaçable près (boîtier, carte, ...).

Le système est considéré soit comme étant réalisé (produit fini et non modifiable), soit en cours de conception (à l'état d'avant-projet).

L'objet du modèle est de fournir des paramètres de testabilité du système permettant :

- de comparer différentes stratégies de test en vue d'un choix optimal, prenant en compte :
  - . des critères de difficulté de mise en oeuvre (sans avoir à réaliser exhaustivement cette mise en oeuvre, c'est-à-dire sans atteindre le stade de la génération des séquences de test)
  - . des critères de résolution : hypothèses de pannes, taux de détection, degré de localisation

- de proposer des modifications (adjonction de matériel supplémentaire, points de test, ...) améliorant sa maintenance ultérieure (le système doit être en cours de conception ou modifiable)

- de proposer une implémentation en unités de remplacement par une partition du système.

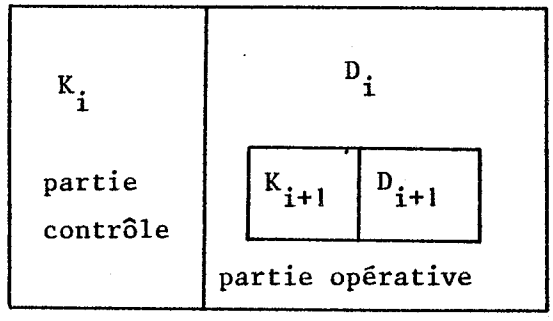
Il s'ensuit que le modèle doit prendre en compte la complexité du système par une modélisation récursive (effet de Zoom) permettant une étude applicable du niveau de description structurelle à celui de comportement.

I - 2. LA PARTIE CONTROLE ET LA PARTIE OPERATIVE

Tout système peut être décomposé en deux parties (processeur discret d'information):

- une partie contrôle générant des ordres
- une partie opérative : modules de traitement et chemins de données.

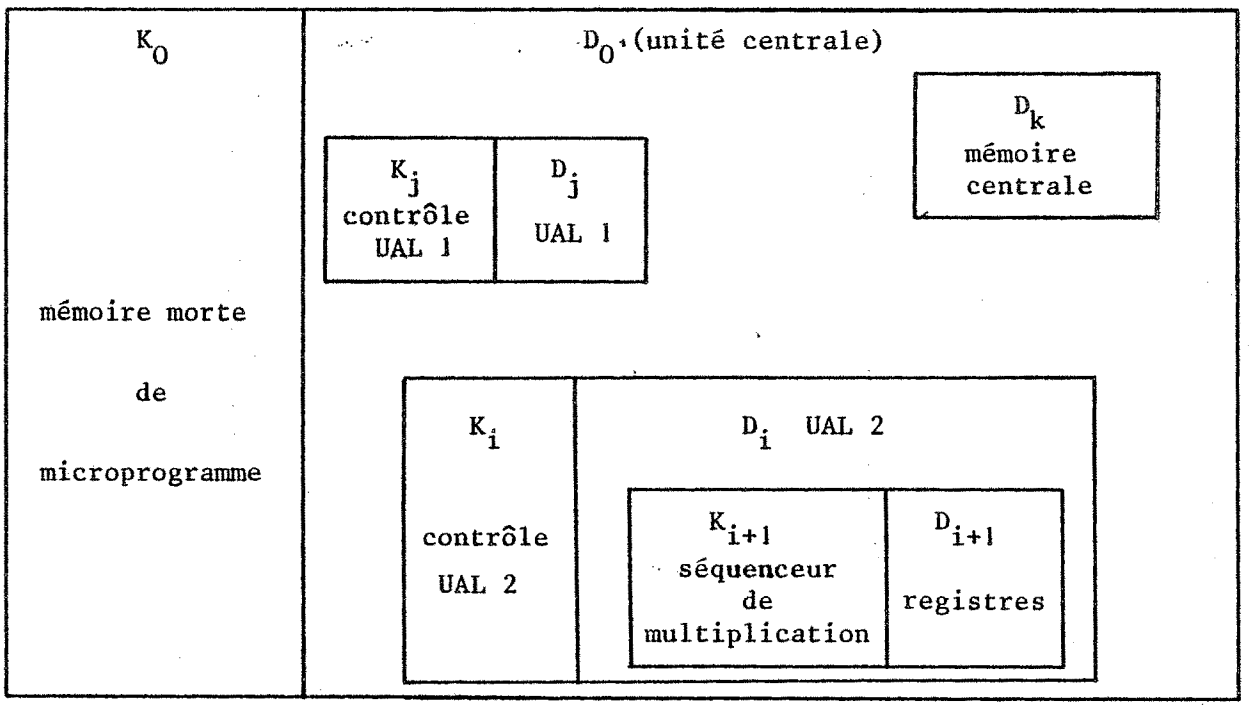
Selon le niveau de spécification, on obtient un schéma récursif du type suivant :



Pour un système donné, deux niveaux de contrôle au moins peuvent être considérés : un contrôle central K_i (mémoire morte par exemple) et un contrôle local K_{i+1} (séquenceur câblé local) à l'intérieur de la partie opérative D_i.

Pour un niveau donné j, la partie contrôle K_j peut être inexistante : D_j est alors gérée par un contrôle de niveau supérieur $K_{j-\alpha}$.

Exemple [ROB, 75]



On considère que la maintenance d'un tel système comportera trois étapes :

α) Le test de la partie minimale du contrôle qui assure le déroulement des (micro)programmes de test du système global (hardcore, autotest, test préalable)

β) Le test de l'ensemble du système à l'exclusion de la partie contrôle ; cette étude fait l'objet de la suite de ce chapitre.

γ) Le test complet de la partie contrôle par des méthodes qui lui sont propres et qui sont exposées au chapitre II.

I - 3. OUTILS NECESSAIRES A L'ETUDE DE LA TESTABILITE DES SYSTEMES LOGIQUES

a) *Rappels élémentaires* [BER, 63]

. Un graphe G = (X,U) est un couple constitué par :

- un ensemble de sommets $X = \{x_1, x_2, \ldots, x_n\}$
- une famille d'arcs $U = \{u_1, u_2, \ldots, u_p\}$ éléments du produit cartésien $X \times X$.

. Pour un arc u = (x,y), x est son extrémité initiale et y son extrémité finale.

- y est dit <u>successeur</u> de x ; l'ensemble des successeurs de x est dénoté $\Gamma_G^+(x)$

- x est dit <u>prédécesseur</u> de y ; l'ensemble des prédécesseurs de y est dénoté $\Gamma_G^-(y)$.

- on définit l'ensemble des sommets voisins d'un sommet x comme $\Gamma_G(x) = \Gamma_G^+(x) + \Gamma_G^-(x)$.

- $p \in X$ est un <u>puits</u> si $\Gamma_G^+(p) = \emptyset$. L'ensemble des puits est dénoté P.

- $s \in X$ est une <u>source</u> si $\Gamma_G^-(s) = \emptyset$. L'ensemble des sources est dénoté S.

. <u>Un graphe est biparti</u> si l'ensemble X de ses sommets est partitionné en deux classes Z et T telles que :

$\forall z \in Z$, $\Gamma_G(z) \subseteq T$; Z est l'ensemble des <u>places</u>
$\forall t \in T$, $\Gamma_G(t) \subseteq Z$; T est l'ensemble des <u>transitions</u>.

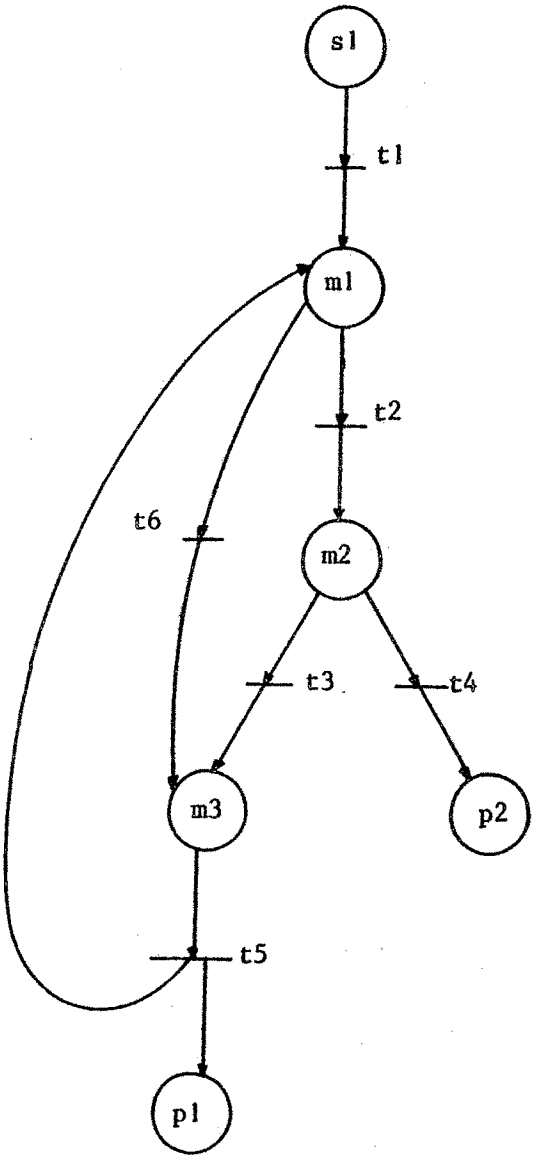
On définit un <u>module</u> m comme une place qui n'est ni source ni puits
$m \in Z$; $\Gamma_G^+(m) \neq \emptyset$ <u>et</u> $\Gamma_G^-(m) \neq \emptyset$.

L'ensemble des modules est dénoté M.

On a donc $Z = M \cup S \cup P$.

. <u>Représentation graphique - Exemple</u> (Figure 1.2)

Les places sont représentées par des cercles et les transitions par des barres.



$S = \{s1\}$

$P = \{p1, p2\}$

$M = \{m1, m2, m3\}$

$\Gamma_G^-(m1) = \{t1, t5\}$

$\Gamma_G^+(m1) = \{t2, t6\}$

$\Gamma_G^-(t5) = \{m3\}$

$\Gamma_G^+(t5) = \{p1, m1\}$

Figure 1.2 - Représentation graphique

b) *Représentation d'un système logique pour la testabilité*

La modélisation des systèmes logiques pour le test est fondée sur une représentation du système par un graphe.

On veut distinguer :

- les modules matériels du système
- les conditions de fonctionnement qui autorisent le transfert d'information entre modules matériels.

Un système logique peut donc être représenté par un graphe biparti, dans lequel :

- les places seront les modules matériels
- les transitions seront les conditions de fonctionnement.

Les graphes bipartis considérés ont alors la propriété :

$\forall t \in T, \Gamma^{-}(t) \neq \emptyset$ et $\Gamma^{+}(t) \neq \emptyset$.

c) *Structure mathématique de travail*

Le test d'un système logique se fait en véhiculant une information de test à travers le système, à partir des sources qui sont génératrices de l'information et en observant les résultats à la sortie des puits, qui sont émetteurs d'information vers l'extérieur.

On a donc besoin de définir l'ensemble des chemins d'information ou "chemins généralisés" dans le graphe biparti associé.

La définition de ces chemins obéit aux conditions suivantes :

. tout arc issu de ou entrant dans un module matériel porte toute l'information nécessaire à ce module.

. les arcs issus d'une transition se partagent l'information entrant sur cette transition.

. la condition de fonctionnement associée à une transition n'est valide que si chaque arc entrant sur cette transition porte une information.

Il s'ensuit que :

. si une transition appartient à un chemin, toutes les places voisines de cette transition y appartiennent

. si une place appartient à un chemin, une seule des transitions successeurs et une seule des transitions antécédentes y appartiennent, sauf dans le cas de cycles.

Exemple :

. si t5 appartient à un chemin, m3, p1 et m1 appartiennent à ce chemin

. à partir de m1, on peut définir quatre sous-ensembles :

{...t1 m1 t2 .....}
{...t1 m1 t6 .....}
{...t5 m1 t2 .....}
{...t5 m1 t6 .....}

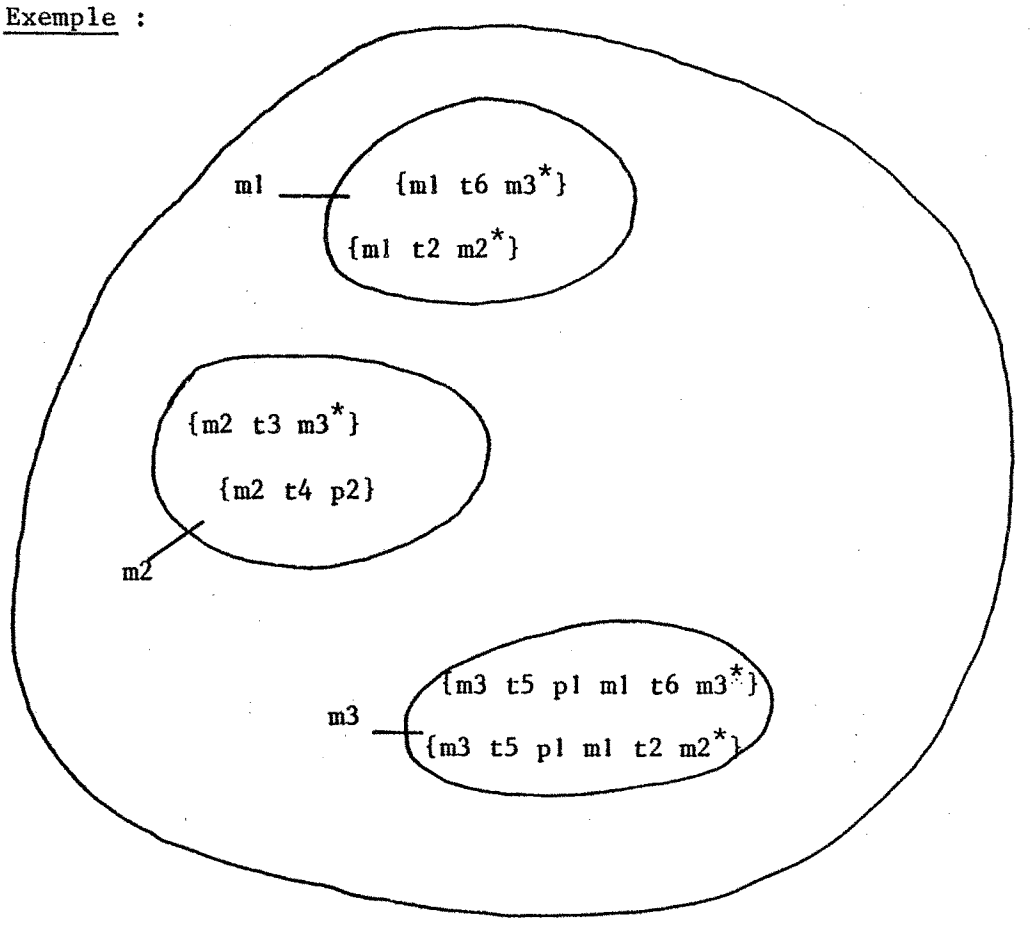
générateurs de quatre chemins.

Ceci conduit donc à introduire deux opérations qui sont associées aux deux types de noeuds :

- une fonction Somme qui est appliquée sur les noeuds du type "place"
- une fonction Produit qui est appliquée sur les noeuds du type "transition".

Ces deux fonctions définissent pour chaque noeud du graphe un ensemble de sous-ensembles de sommets, appelé "hyperchemin", élément de $P(P(X))$.

Exemple :



où m_i^* est une variable du système.

Il faut alors résoudre le système d'équations induit par ces fonctions.

II - DÉFINITIONS

. *D1 : Définition d'un hyperchemin*

Un hyperchemin a ∈ H est un élément de $P(P(X))$ où X = Z ∪ T, et $P(X)$ est l'ensemble des parties de X

$$a = \{A_i \in P(X),\ 1 \leq i \leq p\}.$$

Nous noterons par la suite

$$I_p = \{i \in N \ /\ 1 \leq i \leq p\}.$$

Exemple (figure 1.2)

a1 = {{m2 t3 m3}, {m2 t4 p2}}

a2 = {{m3 t5 p1 m1}}.

Remarque : L'hyperchemin vide de H sera noté 0.

. *D2 : Somme de deux hyperchemins*

Soient deux hyperchemins $a = \{A_i \in P(X),\ i \in I_p\}$ et $b = \{B_j \in P(X),\ j \in I_q\}$. La somme de a et b est définie comme leur union :

$$a \nabla b = a \cup b = \{C_k \in P(X) \ /\ C_k \in a \text{ ou } C_k \in b\}.$$

Exemple formel : a = {{124}, {34}}

b = {{123}, {2}}

a ∇ b = {{124}, {34}, {123}, {2}}

Propriétés :

Les propriétés de l'union d'ensembles induisent que

- la somme de deux hyperchemins est un hyperchemin
- $k \in I_r$, $r \leq p+q$
- la somme est commutative, associative et idempotente
- l'élément neutre de la somme est l'hyperchemin vide 0 : on l'appellera hyperchemin nul .

. *D3 : Produit de deux hyperchemins*

Soient deux hyperchemins $a = \{A_i \in P(X),\ i \in I_p\}$ et $b = \{B_j \in P(X),\ j \in I_q\}$ le produit de a par b est défini par :

$$a \Delta b = \{C_k \in P(X) \ /\ \exists i \in I_p, \exists j \in I_q\ ;\ C_k = A_i \cup B_j\}.$$

Exemple formel : $a = \{\{124\}, \{34\}\}$

$b = \{\{123\}, \{2\}\}$

$a \Delta b = \{\{1234\}, \{124\}, \{234\}\}$.

Propriétés :

Les propriétés de l'union de sous-ensembles induisent que :

- le produit de deux hyperchemins est un hyperchemin
- $k \in I_r$, $r \leq p \times q$
- le produit est commutatif et associatif
- l'hyperchemin nul est élément neutre du produit.

Autre propriété : Le produit est distributif par rapport à la somme

$a \Delta (b \nabla c) = (a \Delta b) \nabla (d \Delta c)$.

Démonstration :

$b \nabla c = \{X_\alpha \ ; \ X_\alpha = B_j \text{ ou } X_\alpha = C_k\}$

$a \Delta (b \nabla c) = \{Y_\beta \ ; \ Y_\beta = A_i \cup X_\alpha\}$

$= \{Y_\beta \ ; \ Y_\beta = A_i \cup B_j \text{ ou } Y_\beta = A_i \cup C_k\}$ (1)

$a \Delta b = \{Z_\gamma \ ; \ Z_\gamma = A_i \cup B_j\}$

$a \Delta c = \{W_\delta \ ; \ W_\delta = A_i \cup C_k\}$

$(a \Delta b) \nabla (a \Delta c) = \{T_\varepsilon \ ; \ T_\varepsilon = Z_\gamma \text{ ou } W_\delta\}$

$= \{T_\varepsilon \ ; \ T_\varepsilon = A_i \cup B_j \text{ ou } T_\varepsilon = A_i \cup C_k\}$ (2)

(1) et (2) $\Rightarrow a \Delta (b \nabla c) = (a \Delta b) \nabla (a \Delta c)$.

. *D4 : Extension d'un hyperchemin*

On appelle extension d'un hyperchemin le cas particulier du produit $a \Delta b$ où b comporte un seul élément :

$b = \{\{x\}, x \in X\}$.

On note l'extension d'un hyperchemin $a = \{A_i \in P(X), i \in I_p\}$ par x

$a \Delta \{\{x\}\} = \{B_k \in P(X) \ / \ \exists i \in I_p, B_k = A_i \cup \{x\}\}$.

Propriétés : Les propriétés du produit de deux hyperchemins induisent que

- l'extension d'un hyperchemin est un hyperchemin
- $k \in I_r$, $r \leq p$
- l'extension est commutative, associative et distributive par rapport à la somme.

. *D5 : Hyperchemin élémentaire*

1. Relation d'équivalence sur H

Soit l'hyperchemin $a = \{A_i \in P(X),\ i \in I_p\}$.

Soit l'application SU : $a \longrightarrow a_e$ telle que

$$a_e = \{A_i \in P(X) \ / \ \forall i', \forall i'', A_i \neq A_{i'} \cup A_{i''},\ i' \neq i'' \neq i\}.$$

Cette application permet de définir une relation sur H :

$$a \equiv b \iff SU\ (a) = SU\ (b).$$

C'est une relation d'équivalence car l'égalité de deux sous-ensembles est une relation d'équivalence.

Exemple: $c = a \Delta b = \{\{1234\}, \{124\}, \{234\}\}$

$SU\ (c) = \{\{124\}, \{234\}\}$

2. Hyperchemin élémentaire

Un élément a_e de H_e, ensemble quotient de H par cette relation d'équivalence ($H_e = H/\equiv$) est appelé hyperchemin élémentaire.

3. Propriété

Cette relation d'équivalence est compatible avec les deux opérations Δ et ∇.

$$\left.\begin{array}{l} a \equiv a' \\ b \equiv b' \end{array}\right\} \Rightarrow \left\{\begin{array}{l} SU(a \Delta b) = SU\ (a' \Delta b') \\ SU\ (a \nabla b) = SU\ (a' \nabla b'). \end{array}\right.$$

Il suffit d'expliciter les deux membres de l'égalité pour prouver l'identité.

. *D6 : Somme et produit d'hyperchemins élémentaires*

Des deux lois de composition internes ∇ et Δ, on déduit les deux lois de composition internes + et × selon le procédé classique.

Propriétés :

. La somme est commutative, associative, idempotente et O est l'élément neutre.

. Le produit est commutatif, associatif, distributif par rapport à la somme et O est l'élément neutre.

Ces propriétés sont directement déduites des propriétés des opérations $\vee$ et Δ.

Autre propriété : $b \times b = b$: le produit d'hyperchemins élémentaires est idempotent.

$$b \Delta b = \{C_k \ / \ C_k = B_i \cup B_j\}$$
$$b \times b = SU\ (b \Delta b) = \{B_i\} = b.$$

Remarque : H_e n'est pas un pseudo-treillis distributif puisque $a + ab \neq a$.

Exemple : $a = \{\{124\}, \{34\}\}$
$b = \{\{123\}, \{2\}\}$
$ab = \{\{124\}, \{234\}\}$
$a+ab = \{\{124\}, \{234\}, \{34\}\} \neq a.$

Note :
On considère dans la suite uniquement des hyperchemins élémentaires.

. *D7 : Somme et produit généralisés*

Soit a_z un hyperchemin élémentaire.

La somme généralisée est définie par :

$$\begin{cases} \sum\limits_{\substack{z \in Y \cup \{x\} \\ Y \in P(X)}} a_z = \left(\sum\limits_{z \in Y} a_z \right) + a_x \\ \sum\limits_{z=x \in X} a_z = a_x \end{cases}$$

Le produit généralisé est défini par :

$$\begin{cases} \prod\limits_{\substack{z \in Y \cup \{x\} \\ Y \in P(X)}} a_z = \left(\prod\limits_{z \in Y} a_z \right) \times a_x \\ \prod\limits_{z=x \in X} \end{cases}$$

Propriétés : Le produit et la somme généralisés sont commutatifs et associatifs.

Ces propriétés découlent des propriétés des opérations + et $\times$.

III - RECHERCHE D'ÉCOULEMENTS ÉLÉMENTAIRES DANS UN GRAPHE BIPARTI

III - 1. OBTENTION DU SYSTEME D'EQUATIONS

A chaque noeud x du graphe on associe un hyperchemin élémentaire h_x. On considère le vecteur $v = (h_x)_{x \in X}$ où $v \in H_e^n$, n étant la cardinalité de X.

. α) Pour toute application ψ de X dans $P(X)$

- on définit une fonction somme $\sigma_{x,\psi}$

$$\sigma_{x,\psi} : H_e^n \longrightarrow H_e$$

$$(h_y) \longrightarrow \begin{cases} . \{\{x\}\} \text{ si } \psi(x) = \emptyset \\ . \left(\sum_{y \in \psi(x)} h_y \right) \times \{x\} \end{cases}$$

- on définit une fonction produit $\Pi_{x,\psi}$

$$\Pi_{x,\psi} : H_e^n \longrightarrow H_e$$

$$(h_y) \longrightarrow \begin{cases} . \{\{x\}\} \text{ si } \psi(x) = \emptyset \\ . \left(\prod_{y \in \psi(x)} h_y \right) \times \{x\} \end{cases}$$

. β) Pour les fonctions $\psi = \begin{cases} \psi 1 : Z \to P(T) \\ \psi 2 : T \to P(Z) \end{cases}$

on définit une fonction F_ψ

$$F_\psi : H_e^n \longrightarrow H_e^n$$

telle que

$$\text{I} \qquad \boxed{F_\psi = [(\sigma_{x,\psi_1})_{x \in Z}, (\Pi_{x,\psi_2})_{x \in T}]}$$

Du fait qu'on travaille sur un graphe biparti, ceci revient à faire une fonction somme sur les places et une fonction produit sur les transitions, c'est-à-dire :

$$\text{II} \qquad \boxed{\begin{array}{l} \forall x \in Z,\ h_x = x \times \Sigma h_t \text{ soit } h_x = \sigma_{x,\psi_1}\ h_t \\ \forall t \in T,\ h_t = t \times \Pi h_x \text{ soit } h_t = \Pi_{x,\psi_2}\ h_x \end{array}}$$

. γ) Le problème de recherche des écoulements se ramène au problème de la résolution de l'équation v = F(v) où $v = ((h_x)_{x\in Z}, (h_t)_{t\in T})$. Il s'agit de démontrer que cette équation

1) admet une solution

2) que cette solution est unique.

Si cette solution existe, on peut l'écrire :

$g = (g_x)_{x\in X}$ où $g_x = \{G_{x,k}\}$.

Un élément $G_{x,k}$ de la solution g_x est appelé pré-écoulement.

Notation : Un pré-écoulement s'écrira sous la forme

$E = \{S, M, P ; T\}$

où $S \subset S$, $M \subset M$, $P \subset P$ et $T \subset T$.

Propriétés des pré-écoulements

. un pré-écoulement n'est pas vide

. un pré-écoulement est entièrement défini par ses transitions : l'ensemble des transitions du pré-écoulement définit de façon unique l'ensemble des places

. deux pré-écoulements G1 et G2 sont égaux si et seulement si $T1 = T2$: $T1$ et $T2$ étant égaux définissent le même ensemble de places

. un pré-écoulement G3 est l'union de deux pré-écoulements G1 et G2 si et seulement si $T3 = T2 \cup T1$: ceci est également déduit du fait que Ti définit entièrement les places du pré-écoulement.

III - 2. EXISTENCE D'UNE SOLUTION

α) *Rappels*

. Définition d'un demi-treillis supérieur complet [DUB,61]

Un demi-treillis supérieur ou sup demi-treillis est un ensemble ordonné dans lequel tout couple d'éléments admet un plus petit majorant.
Un sup demi-treillis est complet si tout sous-ensemble de ce treillis admet un plus petit majorant.

. Définition d'une fonction monotone non décroissante

Soit E un ensemble ordonné par une relation d'ordre R et f une fonction :

$f : x \longrightarrow z = f(x)$, $x \in E$, $z \in E$

f est monotone non décroissante si et seulement si

$$\forall x, \forall y, \; x \; R \; y \Rightarrow f(x) \; R \; f(y).$$

. Forme générale du théorème du point fixe [DUB,61-TAR,55]

Si E est un sup-demi-treillis complet et f une fonction monotone non décroissante : f : E → E, l'ensemble des points fixes de f (solutions de l'équation X = f(X)) est un sup-demi-treillis non vide.

Construction d'une solution :

Soit O l'élément nul du treillis et soit U le plus petit majorant

$$X_0 = 0$$
$$X_1 = f(X_0)$$
$$X_n = f(X_{n-1})$$

alors $\bigcup_{i \in N} f(X_i)$ est un point fixe.

β) *Relation d'ordre sur* H_e

On définit une relation d'ordre sur H_e, notée $<.$, comme suit :

$$a <. b \iff a+b = b.$$

- Réflexivité : $a <. a \iff a+a = a$
  puisque la somme est idempotente.

- Transitivité : $\left.\begin{array}{l} a <. b \\ b <. c \end{array}\right\} \Rightarrow a <. c$

$a <. b \iff a+b = b \quad (1)$

$b <. c \iff b+c = c \quad (2)$

$$(2) \quad a+b+c = a+c \quad \left.\begin{array}{l} \\ (1) \end{array}\right\} \iff b+c = a+c \quad \left.\begin{array}{l} \\ (2) \end{array}\right\} \iff c = a+c$$

- Antisymétrie : $\left.\begin{array}{l} a <. b \\ b <. a \end{array}\right\} \Rightarrow a = b$

$$\left.\begin{array}{l} a <. b \iff a+b = b \\ b <. a \iff b+a = a \\ a+b = b+a \end{array}\right\} \iff a = b$$

C'est une relation d'ordre partiel (l'élément neutre 0 est inférieur à tout élément de H_e).

γ) *La somme + et le produit × sont isotones* (compatibles avec la relation d'ordre)

. Soit $a <\cdot b \Rightarrow a+x <\cdot b+x \quad \forall x \in H_e$.

Démonstration :

$$a <\cdot b \Rightarrow a+b = b \quad (1)$$

$$a+x+b+x = a+b+x+x = a+b+x$$

$$\overset{(1)}{=} b+x$$

$$\Rightarrow (a+x) + (b+x) = b+x$$

$$\Rightarrow a+x <\cdot b+x$$

. Soit $a <\cdot b \Rightarrow a\times x <\cdot b\times x \quad \forall x \in H_e$.

Démonstration :

$$a <\cdot b \Rightarrow a+b = b \quad (1)$$

$$a\times x+b\times x = (a+b) \times x$$

$$\overset{(1)}{=} b\times x$$

$$\Rightarrow (a\times x) + (b\times x) = b\times x$$

$$\Rightarrow a\times x <\cdot b\times x.$$

Il s'ensuit que les fonctions somme σ et produit π sont isotones (composition de fonctions isotones).

δ) *Plus petit majorant de deux éléments*

Le plus petit majorant de deux éléments est défini comme la somme de ces éléments

$$\text{Maj } (a,b) = a+b.$$

C'est un majorant : il suffit de démontrer que $a <\cdot a+b$

$$a + (a+b) = (a+a) + b = a+b \Rightarrow a <\cdot a+b$$

C'est le plus petit majorant : supposons qu'il en existe un plus petit soit h1. On a alors

$$a <\cdot h1 <\cdot a+b$$

$$b <\cdot h1 <\cdot a+b$$

$$\left.\begin{array}{l} a <\cdot h1 \Rightarrow a+h1 = h1 \\ b <\cdot h1 \Rightarrow b+h1 = h1 \end{array}\right\} \Rightarrow a+b+h1 = h1 \Rightarrow a+b <\cdot h1$$

$$\Rightarrow h1 = a+b.$$

ε) *H_e est un sup-demi-treillis complet*

C'est un sup-demi-treillis puisque tout couple d'éléments admet un plus petit majorant.

Il est complet puisque tout sous-ensemble H' de H_e admet comme plus petit majorant $\Sigma h'_i$, $\forall h'_i \in H'$.

Il s'ensuit que H_e^n est un sup-demi-treillis complet pour la relation d'ordre $v_1 <\cdot\ v_2 \Rightarrow \forall x,\ h_x^1 <\cdot\ h_x^2$ où $v_1 = (h_x^1)_{x\in X}$ et $v_2 = (h_x^2)_{x\in X}$

ν) *Monotonie de F*

F est monotone si et seulement si $v_1 <\cdot\ v_2 \Rightarrow F(v_1) <\cdot\ F(v_2)$.

Ceci résulte du fait que les fonctions σ et π sont isotones car F est de la forme I (§ 3.1).

Conclusion :

> La forme générale du théorème du point fixe nous assure alors que l'équation v = F(v) admet une solution non vide.

III - 3. RESOLUTION DE L'EQUATION $v = F_\psi(v)$

L'équation $v = F_\psi(v)$ où $v = (h_x)_{x\in X}$ fournit un système de n équations à n inconnues où n est la cardinalité de X.

Ces équations sont du type :

$$\begin{cases} z^* = z \times \underset{i}{\Sigma}\, t_i,\ z \in Z,\ t_i \in \psi_1(z) \\ t^* = t \times \underset{j}{\Pi}\, z_j,\ t \in T,\ z_j \in \psi_2(t). \end{cases}$$

Ces équations sont du type II (§ 3.1) où les inconnues sont marquées d'une astérisque *.

Etape 1 : Réduction des inconnues

Nous cherchons à nous ramener à un système d'équations où les seules inconnues sont les places z. Pour cela, on remplace les t^* par leurs valeurs, dans les équations z^*

$$z^* = z \times \sum_i (t_i \times \prod_j z_j^*)$$

Etape 2 : Résolution du système

On considère donc le sous-système d'équations dans lequel le premier terme z^* est un module ($z \in M$).
On choisit l'un des z_i^* :

- si z_i^* est de la forme $z_i^* = a+b\times z_i^*$ où a et b sont des hyper-chemins élémentaires, alors $z_i^* = a+b + a\times b$
- sinon on substitue z_i^* dans les autres équations
- on itère cette étape 2 avec une nouvelle inconnue z_j^*.

Démonstration :

hypothèse : $z_i^* = a+b+a\times b$

$$\begin{aligned} a+b\times z_i^* &= a+b\times(a+b+a\times b) \\ &= a+b\times a+b\times b+b\times a\times b \\ &= a+a\times b+b \\ &= z_i^* \end{aligned}$$

Cette forme de z_i^* est dérivée de la forme de la solution du théorème du point fixe.

Construction :

$$\begin{aligned} z_{i,0}^* &= 0 \\ z_{i,1}^* &= f(z_{i,0}^*) = a+b\times z_{i,0}^* = a+b \\ z_{i,2}^* &= f(z_{i,1}^*) = a+b\times z_{i,1}^* = a+a\times b+b \\ z_{i,3}^* &= f(z_{i,2}^*) = z_{i,2}^* \end{aligned}$$

Le théorème du point fixe donne la solution comme le plus petit majorant des itérés successifs ; le plus petit majorant est la somme

$$\sum_{n\in N} z_{i,n}^* = z_{i,2}^* = a+b+ab.$$

Le procédé de construction montre que la solution est unique.

Le nombre maximum de substitutions est le nombre de modules du graphe : la substitution d'un z_i^* dans les autres équations se termine parce qu'il existe une solution non vide.

L'ensemble des pré-écoulements G_i est obtenu en remplaçant dans les équations des sources et des puits les z_i^* par leurs valeurs.

Définitions :

- Un sous-écoulement commandable est un pré-écoulement g_s, $s \in S$ pour lequel $\psi(x) = \Gamma_G^+(x) \quad \forall x \in X$.
- Un sous-écoulement observable est un pré-écoulement g_p, $p \in P$, pour lequel $\psi(x) = \Gamma_G^-(x) \quad \forall x \in X$.
- Un écoulement E est un pré-écoulement g_s, $s \in S$, pour lequel :

$$\psi(x) = \begin{cases} \Gamma_G(x) & \forall x \in T \\ \Gamma_G^+(x) & \forall x \in Z \end{cases}$$

Ceci revient à dire qu'un écoulement est un sous-graphe biparti tel que :

- si un module m appartient à E, l'une des transitions prédécesseurs et l'une des transitions successeurs appartiennent à E
- si une transition t appartient à E, toute place prédécesseur et toute place successeur appartiennent à E
- pour tout module de l'écoulement, il existe un chemin (au sens théorie des graphes) appartenant à l'écoulement, allant d'une source $s \in E$ à un puits $p \in E$ via ce module.

L'ensemble des écoulements du système est défini comme l'ensemble des pré-écoulements qui sont "sans impasse", c'est-à-dire des pré-écoulements qui comportent au moins une source et au moins un puits.

III - 4. EXEMPLE : RECHERCHE D'ECOULEMENTS

Soit le graphe décrit en Figure I.2:

. *Obtention du système d'équations*

Nous prenons chaque sommet du graphe et nous appliquons

- la fonction Somme sur chaque place
- la fonction Produit sur chaque transition

Nous obtenons le système suivant:

$$\begin{cases} s1^* = s1 \times t1^* \\ t1^* = t1 \times m1^* \\ m1^* = m1 \times (t6^* + t2^*) \\ t2^* = t2 \times m2^* \\ t6^* = t6 \times m3^* \\ m2^* = m2 \times (t3^* + t4^*) \\ m3^* = m3 \times t5^* \\ t3^* = t3 \times m3^* \\ t4^* = t4 \times p2^* \\ t5^* = t5 \times (p1^* \times m1^*) \\ p2^* = p2 \\ p1^* = p1 \end{cases}$$

Pour la commodité de l'écriture on écrira

$$a \times b \times c = abc$$

. *Etape 1 : réduction des inconnues*

$$\text{A} \begin{cases} s1^* = s1\ t1\ m1^* \end{cases}$$

$$\text{B} \begin{cases} m1^* = m1\ t6\ m3^* + m1\ t2\ m2^* \\ m2^* = m2\ t3\ m3^* + m2\ t4\ p2 \\ m3^* = m3\ t5\ p1\ m1^* \end{cases}$$

. *Etape 2 : résolution du sous-système* B

1) On choisit $m1^*$; on substitue $m1^*$ dans les autres équations

$$\begin{cases} m1^* = m1\ t6\ m3^* + m1\ t2\ m2^* \\ m2^* = m2\ t3\ m3^* + m2\ t4\ p2 \\ m3^* = m3\ t5\ p1.(m1\ t6\ m3^* + m1\ t2\ m2^*) \end{cases}$$

2) $m3^*$ est de la forme $m3^* = a + m3^* b$

où $a = m3\ t5\ p1\ m1\ t2\ m2^*$

$b = m3\ t5\ p1\ m1\ t6$

La solution est donc $m3^* = a + b + ab$

$m3^* = m3\ t5\ p1\ m1\ t2\ m2^* + m3\ t5\ p1\ m1\ t6 + m3\ t5\ p1\ m1\ t2\ t6\ m2^*$

3) On substitue $m3^*$ dans $m2^* = m2\ t3\ m3^* + m2\ t4\ p2$

On obtient:

$$\begin{aligned} m2^* &= m2\ t3\ m3\ t5\ p1\ m1\ t2\ m2^* \\ &+ m2\ t3\ m3\ t5\ p1\ m1\ t6 \\ &+ m2\ t3\ m3\ t5\ p1\ m1\ t2\ m2^*\ t6 + m2\ t4\ p2 \end{aligned}$$

$m2^*$ est de la forme $m2^* = a + m2^* b$

où $a = m2\ t3\ m3\ t5\ p1\ m1\ t6 + m2\ t4\ p2$

$b = m2\ t3\ m3\ t5\ p1\ m1\ t2 + m2\ t3\ m3\ t5\ p1\ m1\ t2\ t6$

La solution est donc $m2^* = a + b + ab$, ce qui donne après développement:

$$\begin{aligned} m2^* &= m1\ m2\ m3\ p1\ t3\ t5\ t6 \\ &+ m2\ p2\ t4 \\ &+ m1\ m2\ m3\ p1\ t2\ t3\ t5 \\ &+ m1\ m2\ m3\ p1\ t2\ t3\ t5\ t6 \\ &+ m1\ m2\ m3\ p1\ t2\ t3\ t5\ t6 \\ &+ m1\ m2\ m3\ p1\ t2\ t3\ t4\ t5 \\ &+ m1\ m2\ m3\ p1\ p2\ t2\ t3\ t4\ t5\ t6 \end{aligned}$$

soit $m2^* = a + b + c + d + e + f + g$

avec $e = d = c \cup a$

$g = d \cup f$

$f = b \cup c$

Note: La réduction peut se faire par observation des seules transitions qui caractérisent de façon unique un hyperchemin.

Après réduction on obtient donc:

$$\begin{aligned} m2^* &= m1\ m2\ m3\ p1\ t3\ t5\ t6 \\ &+ m2\ p2\ t4 \\ &+ m1\ m2\ m3\ p1\ t2\ t3\ t5 \end{aligned}$$

4) Solution de $m3^*$: on remplace la valeur de $m2^*$ dans l'équation de $m3^*$ obtenue en 2).

Après développement et réduction, on obtient:

$$m3^* = m1\ m3\ p1\ t5\ t6 + m1\ m2\ m3\ p1\ t2\ t3\ t5 + m1\ m2\ m3\ p1\ p2\ t2\ t4\ t5$$

5) Solution de $m1^*$: on substitue les valeurs de $m2^*$ et $m3^*$ dans l'équation de $m1^*$.

Après développement et réduction, on obtient:

$$m1^* = m1\ m3\ p1\ t5\ t6 + m1\ m2\ p2\ t2\ t4 + m1\ m2\ m3\ p1\ t2\ t3\ t5$$

Les écoulements sont obtenus en portant les valeurs de $m1^*$, $m2^*$, $m3^*$ dans les équations des sources, soit $s1^* = s1\ t1\ m1^*$

On obtient donc trois écoulements (Figure 1.3):

$E1$ = (s1 m1 m3 p1; t1 t5 t6)

$E2$ = (s1 m1 m2 p2; t1 t2 t4)

$E3$ = (s1 m1 m2 m3 p1; t1 t2 t3 t5)

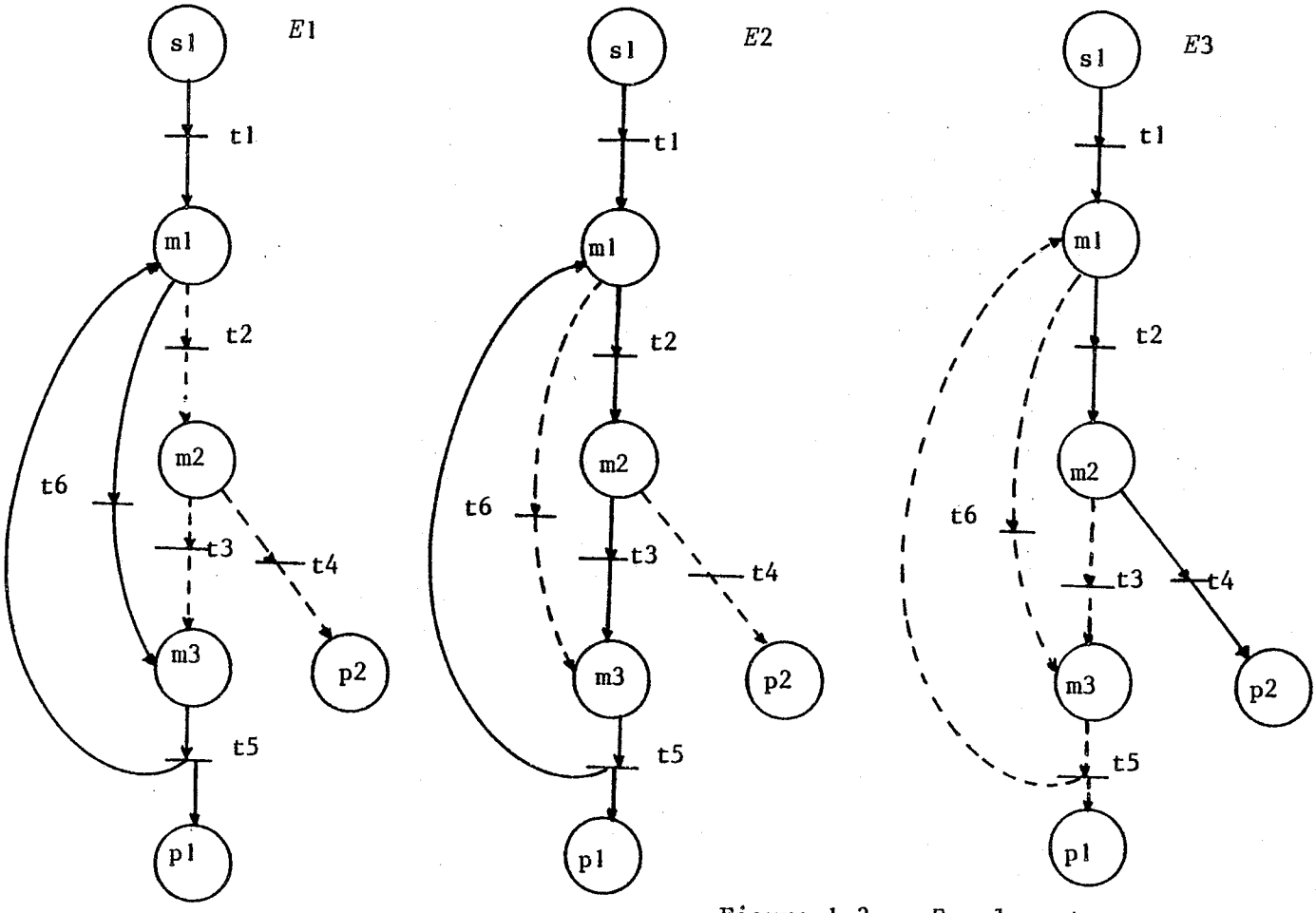


Figure 1.3 - Ecoulements

III - 5. STRUCTURE DU PROGRAMME DE RECHERCHE DES ECOULEMENTS

Ce schéma de programme de recherche des écoulements est écrit dans un langage inspiré de PL/1 et PASCAL; le programme réel a été écrit en Algol W sous CP 67/CMS.

A - DECLARATIONS GENERALES

```
type nature_écoulements=Enumération(commandable, observable, sous_commandable)
var  nature_écoulements NATURE_TRAITEMENT;
var entier      NB_TRANS;
var entier      NB_SOURCES, NB_MODULES, NB_PUITS;
```

B - LECTURE DES PARAMETRES DU MODELE

```
lire (NB_TRANS, NB_SOURCES, NB_MODULES, NB_PUITS);
lire (NATURE_TRAITEMENT);
lire_noms_places_transitions;
var entier NB_PLACES, D_MODULES, F_MODULES;
D_MODULES := NB_SOURCES +1
F_MODULES := NB_SOURCES + NB_MODULES;
NB_PLACES := F_MODULES + NB_PUITS;
```

C - REPRESENTATION DU MODELE

```
type repr_ens_trans = bits (NB_TRANS);
type repr_ens_places = bits (NB_PLACES);
type ens_écoulements =
      struct (ptr (écoulement) enbl_écoulement,
              ptr (ens_écoulements) frère_ens_écoulements);
type écoulement =
      struct (repr_ens_trans  enbl_trans
              repr_ens_places enbl_places
              repr_ens_places enbl_places_inconnues);
var  tableau   (NB_TRANS) repr_ens_places GAMMA_PLUS_TRANS,
                                          GAMMA_MOINS_TRANS;
var  tableau   (NB_PLACES) repr_ens_trans GAMMA_PLUS_PLACES,
                                          GAMMA_MOINS_PLACES;
```

D) ALGORITHME

```
lire_graphe_biparti;
var  tableau (nb_places) ptr (ens_écoulements) SYSTEME_EQUATIONS;
var                      ptr (ens_écoulements) SOLUTION_SYSTEME;
SYSTEME_EQUATIONS :=
            construire_système_équations (NATURE_TRAITEMENT);
SOLUTION_SYSTEME := résoudre (SYSTEME_EQUATIONS);
imprimer  (SOLUTION_SYSTEME);
```

E) CONSTRUCTION DU SYSTEME

```
fonction  construire_système_équations
     (const  nature_écoulements  FCT_PSI)
     retourne  tableau (NB_PLACES) ptr (ens_écoulements)
début
     commentaire
       à l'aide de FCT.PSI et des tableaux gamma,
          on construit le système  d'équations à résoudre
     fin commentaire
fin

fonction   résoudre(const tableau (NB_PLACES)  ptr (ens_écoulements) système)
     retourne  ptr (ens_écoulements)
début
     pour NO_MODULE depuis D_MODULES jusqu à F_MODULES
     faire
       si recherche_A_B (NO_MODULE, A, B)
       alors système (NO_MODULE) :=
             SU (PLUS(A, PLUS(B, MULT (A,B))))
       finsi;
       pour N_MOD depuis D_MODULES jusqu à E_MODULES
       faire
         si N_MOD  ≠  NO_MODULE
         alors
               substituer (système (N_MOD), NO_MODULE)
         finsi
       finfaire
     finfaire;
```

```
        pour N_S depuis 1 jusqu à NB SOURCES
        faire
            pour N_MOD depuis D_MODULES jusqu à F_MODULES
            faire
                substituer (système (N_S), N_MOD)
            finfaire
        finfaire;
        pour N_P depuis F_MODULES +1 jusqu à NB_PLACES
            pour N_MOD depuis D_MODULES jusqu à F_MODULES
            faire
                substituer (système (N_P), N_MOD)
            finfaire
        finfaire;
fin;
```

F) FONCTIONS ALGEBRIQUES

```
fonctions
        PLUS (const ptr (ens_écoulements) A, B)
        retourne ptr (ens_écoulements);
début
        commentaire
          . additionne A et B, ie concatène les deux listes A et B, en
            éliminant les éléments communs, après copie des 2 listes para-
            mètres.
        fincommentaire
fin
fonction MULT (const ptr (ens_écoulements) A, B)
        retourne ptr (ens_écoulements)
début
        commentaire
          Après copie des deux listes paramètres, on clacule le produit
          cartésien des deux listes
        fincommentaire
fin;
```

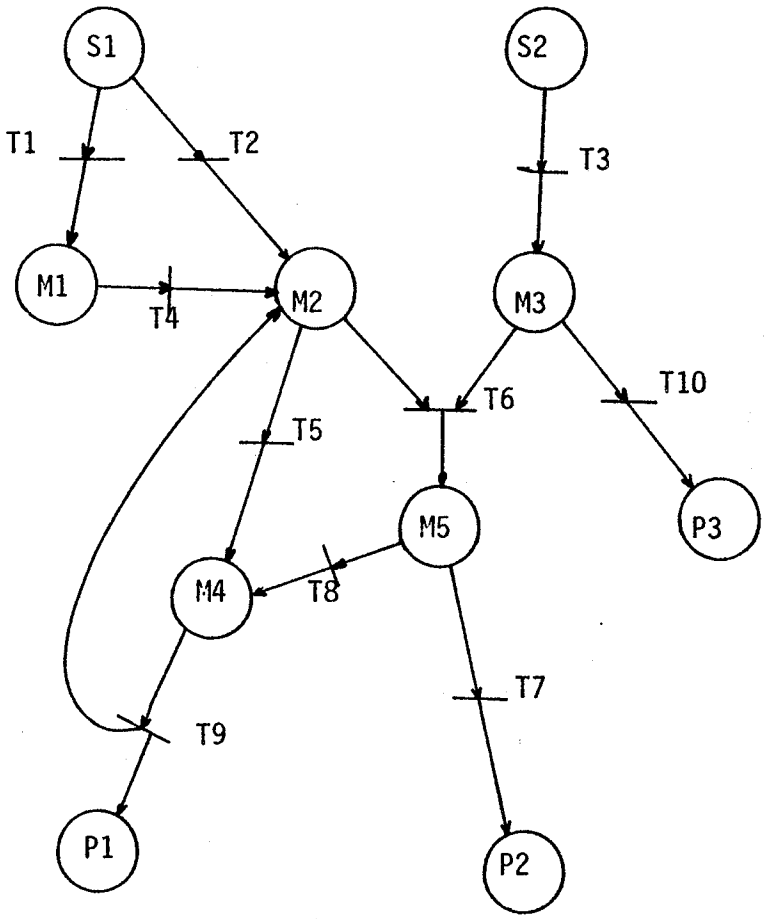
```
fonction SU (const ptr (ens_écoulements) A)
    retourne ptr (ens_écoulements)
début
    commentaire
      construction d'un représentant de la classe d'équivalence de A
      nous utilisont la propriété suivante :
        un élément a de A est union de deux autres éléments b et c de A
        si et seulement si l'ensemble des transitions de a est l'union
        des ensembles de transition de b et c
      Dans le programme réel écrit en Algol W, la complexité de cette
      fonction est
                              O(n ** ** 3)
      où n est le nombre d'éléments de A
    fincommentaire
```

G) FONCTION DE RESOLUTION DU SYSTEME

```
fonction recherche_A_B (const entier NO_MODULE,
                        result ptr (ens_écoulements) A, B)
    retourne booléen
début
    commentaire
      on cherche a établir si l'équation dont le numéro est NO MODULE
      est de la forme z** = a + b**z**
    fincommentaire
fin

fonction substituer
    const ptr (ens_écoulements) EQUATION
    const entier NO_MODULE)
  retourne ptr (ens_écoulements)
début
  commentaire
    on élimine dans un écoulement équation la variable de numéro NO MODULE
    si cette variable est présente dans EQUATION.
  fincommentaire
fin
```

EXEMPLE:



```
 NOMBRE DE TRANSITIONS= 10
 LISTE DES TRANSITIONS
      T1
      T2
      T3
      T4
      T5
      T6
      T7
      T8
      T9
      T10
 NOMBRE DE SOURCES=  2
 LISTE DES SOURCES
      S1
      S2
 NOMBRE DE MODULES=  5
 LISTE DES MODULES
      M1
      M2
      M3
      M4
      M5
 LISTE DES PUITS
      P1
      P2
      P3
 NOMBRE DE PUITS=  3
 NUMERO DE LA DERNIERE PLACE= 10
 MATRICE_DU_RESEAU=
  -1    0    1    0    0    0    0    0    0    0
  -1    0    0    1    0    0    0    0    0    0
   0   -1    0    0    1    0    0    0    0    0
   0    0   -1    1    0    0    0    0    0    0
   0    0    0   -1    0    1    0    0    0    0
   0    0    0   -1   -1    0    1    0    0    0
   0    0    0    0    0    0   -1    0    1    0
   0    0    0    0    0    1   -1    0    0    0
   0    0    0    1    0   -1    0    1    0    0
   0    0    0    0   -1    0    0    0    0    1
 SOUS_ECOULEMENTS COMMANDABLES=
 ( S2 M2 M3 M4 M5 P1 T3 T5 T6 T8 T9 )
 ( S2 M3 M5 P2 T3 T6 T7 )
 ( S2 M2 M3 M4 M5 P1 T3 T6 T8 T9 )
 ( S2 M3 P3 T3 T10 )
 ( S1 M2 M5 P2 T2 T6 T7 )
 ( S1 M2 M4 M5 P1 T2 T6 T8 T9 )
 ( S1 M2 M4 P1 T2 T5 T9 )
 ( S1 M1 M2 M4 P1 T1 T4 T5 T9 )
 ( S1 M1 M2 M4 M5 P1 T1 T4 T6 T8 T9 )
 ( S1 M1 M2 M5 P2 T1 T4 T6 T7 )
 ECOULEMENTS COMMANDABLES=
 ( S1 M1 M2 M4 P1 T1 T4 T5 T9 )
 ( S2 M3 P3 T3 T10 )
 ( S2 M2 M3 M4 M5 P1 T3 T6 T8 T9 )
 ( S1 M2 M4 P1 T2 T5 T9 )
 ( S1 S2 M1 M2 M3 M4 M5 P1 T1 T3 T4 T6 T8 T9 )
 ( S1 S2 M2 M3 M5 P2 T2 T3 T6 T7 )
 ( S2 M2 M3 M4 M5 P1 T3 T5 T6 T8 T9 )
 ( S1 S2 M2 M3 M4 M5 P1 T2 T3 T6 T8 T9 )
```

```
( S1 S2 M1 M2 M3 M5 P2 T1 T3 T4 T6 T7 )

ECOULEMENTS P_MINIMAUX
( S1 S2 M2 M3 M5 P2 T2 T3 T6 T7 )
( S2 M2 M3 M4 M5 P1 T3 T5 T6 T8 T9 )
( S1 M2 M4 P1 T2 T5 T9 )
( S2 M3 P3 T3 T10 )
ECOULEMENTS T_MINIMAUX
( S1 S2 M1 M2 M3 M5 P2 T1 T3 T4 T6 T7 )
( S1 M1 M2 M4 P1 T1 T4 T5 T9 )
( S2 M2 M3 M4 M5 P1 T3 T6 T8 T9 )
( S1 S2 M2 M3 M5 P2 T2 T3 T6 T7 )
( S1 M2 M4 P1 T2 T5 T9 )
( S2 M3 P3 T3 T10 )
ECOULEMENTS OBSERVABLES=
( S2 M3 P3 T3 T10 )
( S1 S2 M2 M3 M4 M5 P2 T2 T3 T5 T6 T7 T9 )
( S1 S2 M1 M2 M3 M4 M5 P2 T1 T3 T4 T5 T6 T7 T9 )
( S1 S2 M1 M2 M3 M5 P2 T1 T3 T4 T6 T7 )
( S1 S2 M2 M3 M5 P2 T2 T3 T6 T7 )
( S2 M2 M3 M4 M5 P2 T3 T5 T6 T7 T8 T9 )
( S2 M2 M3 M4 M5 P2 T3 T6 T7 T8 T9 )
( S1 M2 M4 P1 T2 T5 T9 )
( S1 M1 M2 M4 P1 T1 T4 T5 T9 )
( S2 M2 M3 M4 M5 P1 T3 T5 T6 T8 T9 )
( S2 M2 M3 M4 M5 P1 T3 T6 T8 T9 )
( S1 S2 M2 M3 M4 M5 P1 T2 T3 T6 T8 T9 )
( S1 S2 M1 M2 M3 M4 M5 P1 T1 T3 T4 T6 T8 T9 )

003.71 SECONDS IN EXECUTION
```

III - 6. ECOULEMENTS PARTICULIERS

- Un écoulement minimum est un écoulement qui ne contient pas d'autre écoulement que lui-même.

- Un écoulement élémentaire est un écoulement qui n'est pas l'union de deux écoulements distincts entre eux et distincts de lui-même.

E_i élémentaire si $\forall i, \forall j, \forall k, E_i \neq E_j \cup E_k$

avec $E_j \neq E_i \neq E_k$

- Un écoulement simple est un écoulement dans lequel tout module a un seul prédécesseur et un seul successeur.

$\forall z \in M, |\Gamma_G^+(z)| = 1$ et $|\Gamma_G^-(z)| = 1$

Propriétés:

- Un écoulement minimum est élémentaire.

Cette propriété résulte de la définition d'un écoulement élémentaire et de la définition d'un écoulement minimum.

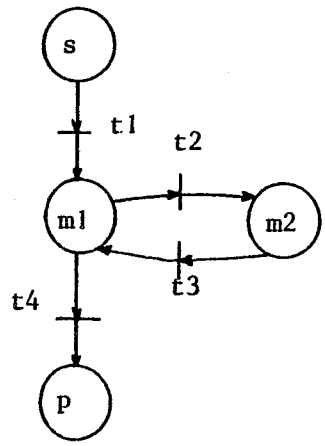
Exemples:

Il existe deux écoulements:

*E*1 = (t1, t4)

*E*2 = (t1, t2, t3, t4)

*E*1 est minimum et simple



Les écoulements sont:

*E*1 = (s1, m1, m2, m3, p; t1, t3, t4)

*E*2 = (s1, s2, m1, m2, m3, p; t1, t2, t3, t4)

*E*3 = (s1, s2, m1, m2, m3, m4, p; t1, t2, t3, t5, t6)

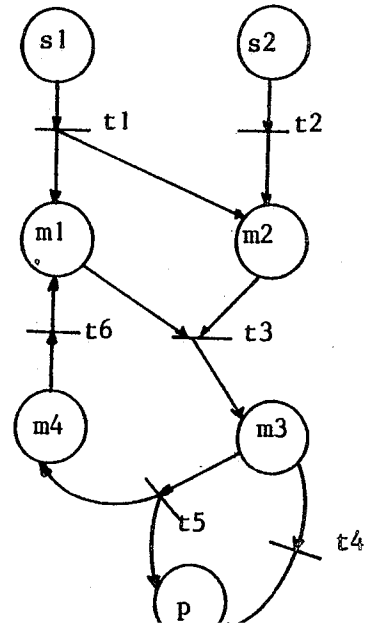
*E*4 = (s1, m1, m2, m3, m4, p; t1, t3, t5, t6)

*E*5 = (s2, m1, m2, m3, m4, p; t2, t3, t5, t6)

*E*1, *E*2, *E*3, *E*5 sont des écoulements élémentaires.

*E*1, *E*3, *E*5 sont des écoulements minimaux.

*E*1 et *E*5 sont des écoulements simples.



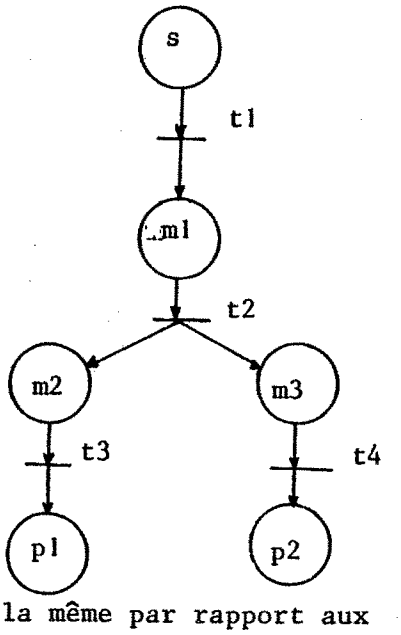
Sous-écoulement observable: un sous-écoulement observable E'_i de E, associé au puits p_i, est le plus grand sous-réseau de E contenant un chemin de chacune de ses places à p_i (toute transition a un seul successeur).

Exemple:

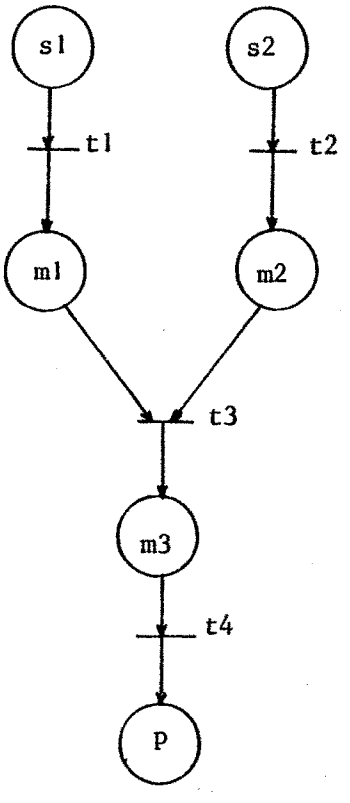
Cet écoulement comporte deux sous-écoulements observables:

E'_1 = (s, m1, m2, p1; t1, t2, t3)

E'_2 = (s, m1, m3, p2; t1, t2, t4)



Sous-écoulement commandable: la définition est la même par rapport aux sources (toute transition a un seul prédécesseur)



Cet écoulement comporte deux sous-écoulements commandables:

E'_1 = (s1,m1,m3,p; t1,t3,t4)

E'_2 = (s2,m2,m3,p; t2,t3,t4)

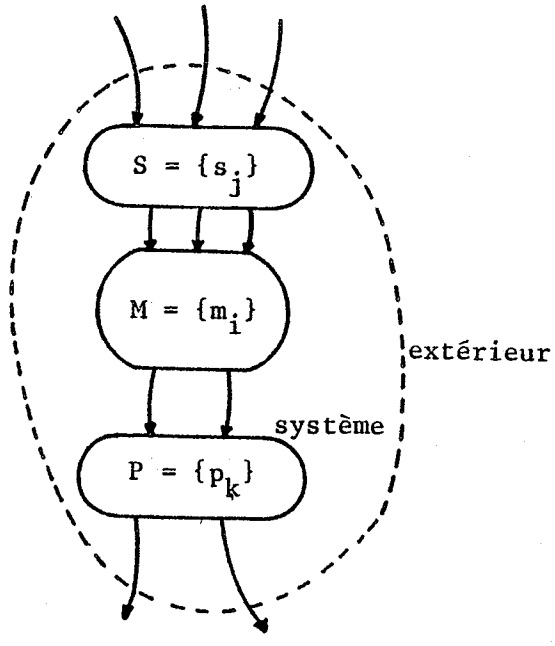
SECTION 3 : APPLICATION AU TEST DES SYSTEMES LOGIQUES [ROB.77a-77b]

I - INTERPRÉTATION

Un système logique est donc modélisé par un graphe biparti tel qu'il a été défini en Section 2, dans lequel :

. Les places $z_i \in Z$ sont des parties matérielles ou fonctionnelles du système (par exemple : additionneur, registre, mémoire,...) c'est-à-dire des ensembles de circuits logiques ayant une fonction bien définie.

Les sources $s_j \in S$ sont des modules d'accès à partir de l'extérieur du système; les puits $p_k \in P$ sont des modules d'observation vers l'extérieur du système (interface avec l'extérieur). Autrement dit, toute information (en particulier les données de test) en provenance de l'extérieur entre dans les sources $s \in S$ du graphe; toute information (en particulier les résultats de test) recueillie à l'extérieur du système sort des puits $p \in P$ du graphe.

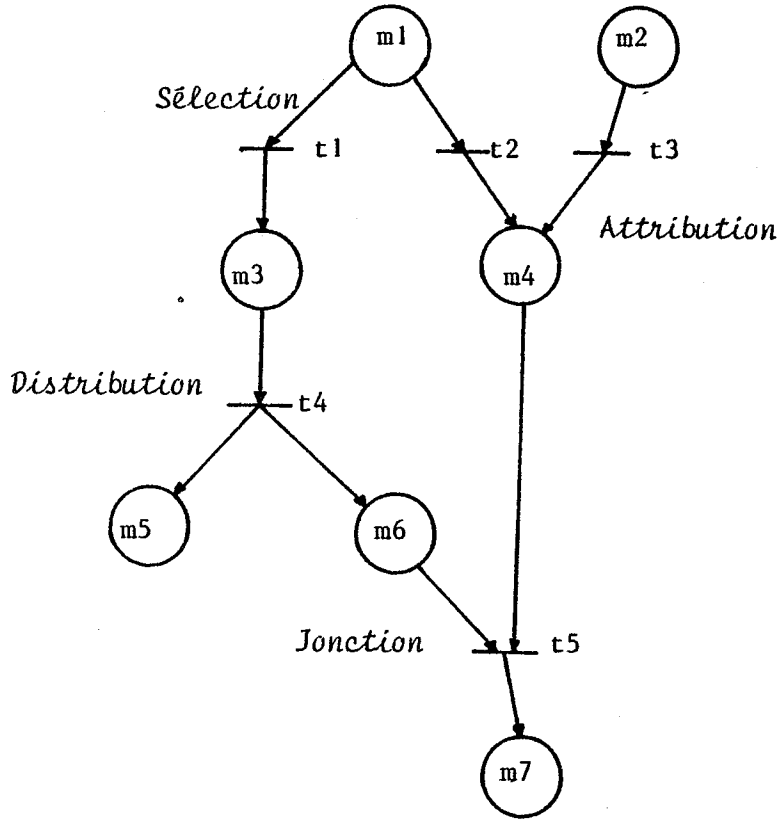


. Les transitions $t_i \in T$ représentent les conditions (fonctions, opérations, commandes) autorisant le transfert d'information d'un module m_i vers un module m_j; ces conditions sont issues de l'organe de contrôle ou de gestion.

. Les arcs $u_j \in U$, connectant places et transitions représentent les supports de l'information à travers le système.

La modélisation proposée est donc une représentation topologique et fonctionnelle du système. Le graphe peut être obtenu directement à partir des fonctions du système, des circuits mis en jeu pour chacune de ces fonctions et de l'information traitée.

I - 1. MODES DE TRANSFERT



On peut considérer quatre modes de transfert, avec l'interprétation suivante :

Le mode sélection : m1 envoie toute l'information soit à m3 (si t1 vérifiée) soit à m4 (si t2 vérifiée).

Le mode attribution : m4 reçoit toute son information soit de m1 (si t2 vérifiée) soit de m2 (si t3 vérifiée).

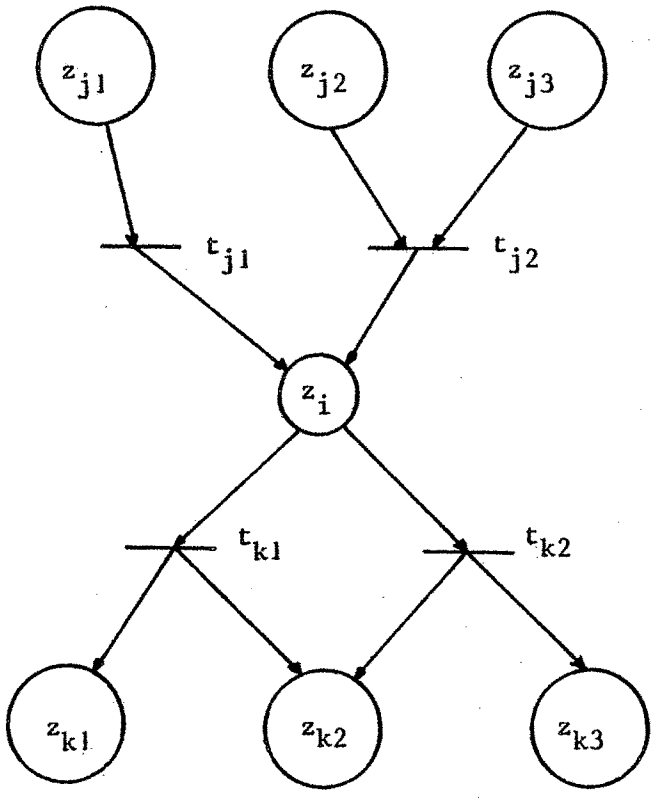
Le mode distribution : m3 envoie son information en partie à m5 et en partie à m6, simultanément (si t4 vérifiée).

Le mode jonction : m7 reçoit son information en partie de m6 et en partie de m4, simultanément (si t5 vérifiée).

Note : Les noeuds de sélection ou d'attribution sont appelés "noeuds OU" et les noeuds de jonction ou distribution sont appelés "noeuds ET".
De manière générale :

- Une place z_i reçoit une information de l'ensemble des places z_j telles que
  $\{z_j\} = \Gamma_G^- (t_j)$ où $t_j \in \Gamma_G^- (z_i)$
- Une place z_i envoie une information à l'ensemble des places z_k telles que
  $\{z_k\} = \Gamma_G^+ (t_k)$ où $t_k \in \Gamma_G^+ (z_i)$

On dit que z_i est observable à travers $\Gamma_G^+ (t_k)$, $t_k \in \Gamma_G^+ (z_i)$ et commandable à travers $\Gamma_G^- (t_j)$, $t_j \in \Gamma_G^- (z_i)$

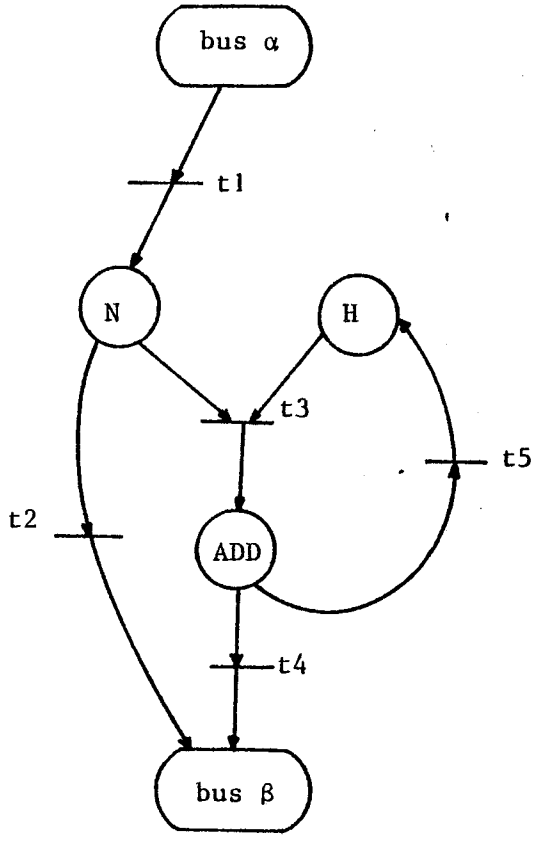


z_i est commandable à travers z_{j1} ou (z_{j2} et z_{j3})

z_i est observable à travers (z_{k1} et z_{k2}) ou (z_{k2} et z_{k3})

Exemple

Soit une unité arithmétique et logique simplifiée comportant deux registres H et N et un additionneur ADD; deux bus de données α et β sont respectivement générateur de données et observateur de résultats. L'unité est modélisée comme suit :



S = {bus α}
M = {H, N, ADD}
P = {bus β}

Les transitions (conditions issues d'une partie contrôle : mémoire morte par exemple) peuvent avoir la signification suivante :

t1 : charger le contenu du bus dans le registre N

t2 : faire un décalage à droite du contenu de N et sortir le résultat sur le bus

t3 : additionner H et N

t4 : sortir le résultat de l'addition sur le bus

t5 : décaler à gauche le résultat et le charger dans H.

- L'additionneur est commandable à travers H et N : si la commande d'addition est envoyée (t3 vrai), ADD reçoit une information à la fois de H et N

$$\{H, N\} = \Gamma_G^- (t3)$$

- L'additionneur est observable soit directement (sur le bus β) soit à travers le registre H si la commande de décalage est envoyée (t5 vrai)

$$\{H\} = \Gamma_G^+ (t5)$$

$$\{\text{bus } \beta\} = \Gamma_G^+ (t4)$$

$$\text{avec } \{t4, t5\} = \Gamma_G^+ (ADD)$$

I - 2. INCIDENCE DU CONTROLE SUR LE MODELE

On a vu que les transitions dans le graphe représentent les conditions issues de l'organe de gestion autorisant le transfert d'information d'un module vers un autre.

Le graphe est donc paramétré par la partie contrôle; ces renseignements doivent permettre de définir l'ensemble des écoulements "possibles" dits écoulements k-compatibles autrement dit les séquences de transitions autorisées.

A toute transition t_k du réseau est associé un prédicat $P_k = P' \cap P''$

P' : prédicat de séquencement

P'' : prédicat de contrainte

La transition t_k est activée (commandes validées) si son prédicat associé est vrai c'est-à-dire si P' est vrai et P'' est vrai.

Le prédicat de séquencement est une somme de fonctions de commande : $P' = \Sigma C_k$; si l'une de ces fonctions de commande est validée, le prédicat de séquencement est vrai.

Le prédicat de contrainte P'' exprime les contraintes dûes à des transitions t_j activées antérieurement dans l'écoulement considéré :

$$P'' = \prod_j f_{j \to k}(C).$$

Le prédicat de contrainte est vrai si toutes les contraintes $f_{j \to k}(C)$ sont vérifiées ($f_{j \to k}(C)$ peut se décomposer également en une somme de fonctions de commande).

Remarque : il n'existe de contraintes sur l'enchaînement des transitions que s'il existe des noeuds OU dans le graphe considéré.

Exemple : Soit un système informatique réalisant deux types d'instructions

1er type : instruction à une adresse : r op M' → M'

2e type : instruction à deux adresses : M op M' → r

L'instruction comporte trois champs :

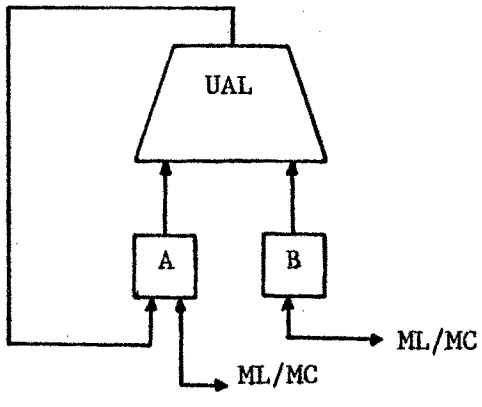
Le champ RI_L : adresse de r dans la mémoire locale ML

Le champ RI_C : adresse de M en mémoire centrale MC

Le champ $RI_{C'}$: adresse de M' en mémoire centrale MC

qui sont les sources du système.

op désigne une opération arithmétique et logique effectuée dans l'UAL entre deux opérandes contenus dans les registres d'entrée A et B; le résultat de l'opération est dans A.

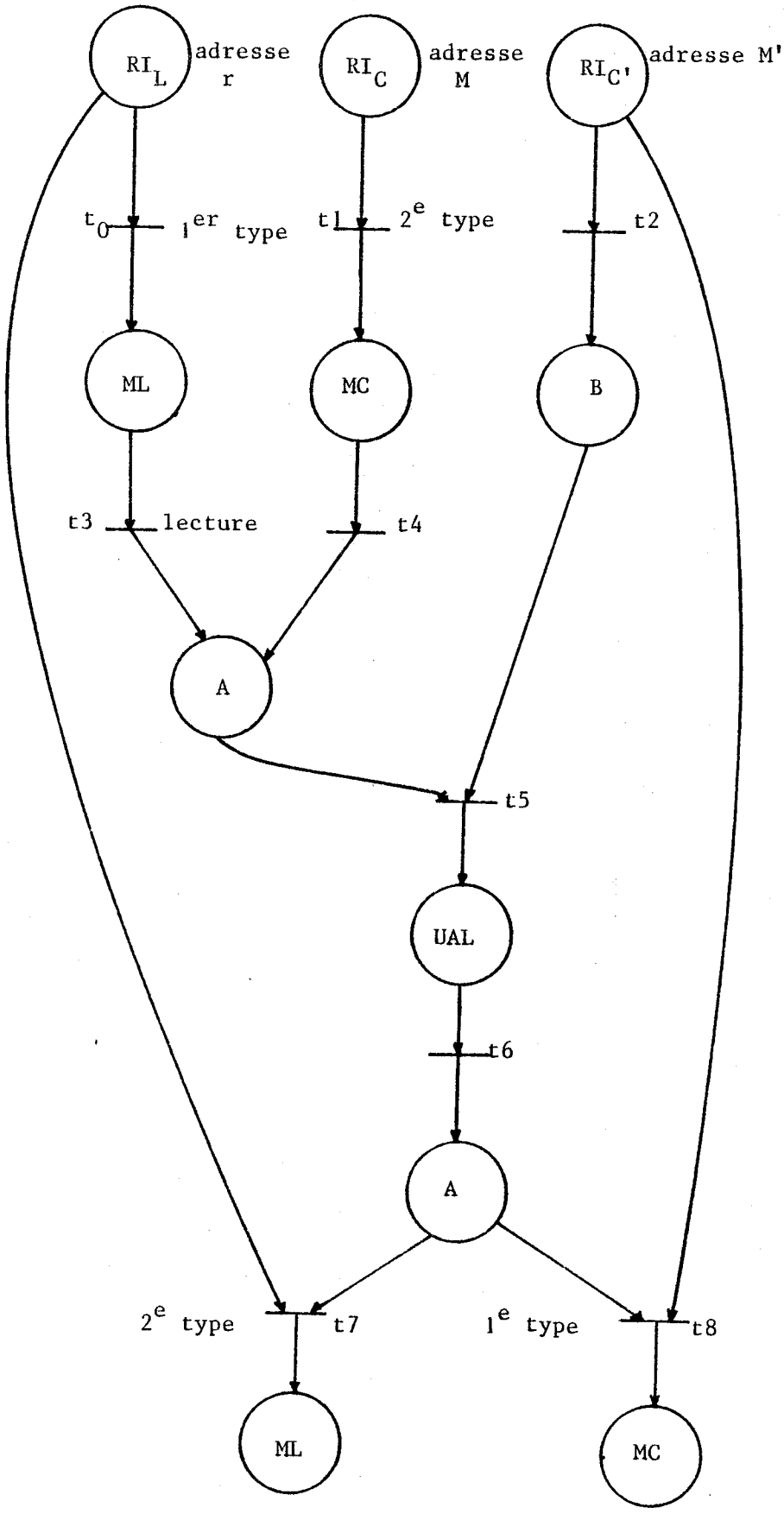


Le code opération CO peut prendre deux valeurs :

CO = 0 instruction de 1er type

CO = 1 instruction de 2e type

Le système est modélisé comme suit (Figure 1.4)



t0: chargement de r si CO=0

t1: chargement de M si CO=1

t2: chargement de M' dans B

t3: lecture de r et chargement dans A

t4: lecture de M et chargement dans A

t5: opération entre A et B

t6: chargement du résultat dans A

t7: écriture de A dans la mémoire locale si instruction de 2ème type

t8: écriture de A dans la mémoire centrale si instruction de 1er type

Figure 1.4. Exemple

Organigramme des opérations

1er type : r op M' → M'	2e type : M op M' → r
r → A (t_0,t_3)	M → A (t_1,t_4)
M' → B (t_2)	M' → B (t_2)
A op B → A (t_5,t_6)	A op B → A (t_5,t_6)
A → MC (t_8)	A → ML (t_7)
adressée par RI_C	adressée par RI_L

Les écoulements, en termes de transitions sur le modèle non renseigné, sont les suivants :

$E_1 = (t_0, t_2, t_3, t_5, t_6, t_7)$

$E_2 = (t_0, t_2, t_3, t_5, t_6, t_8)$

$E_3 = (t_1, t_2, t_4, t_5, t_6, t_7)$

$E_4 = (t_1, t_2, t_4, t_5, t_6, t_8)$

La représentation du contrôle implique que t_7 ne peut avoir lieu que pour une instruction de 2e type soit si t_1 appartient à l'écoulement; de même t_8n'appartient à un écoulement que si t_0 y appartient.

Il s'ensuit qu'on a seulement 2 écoulements K-compatibles, soient E_2 et E_3.

La représentation du contrôle permet donc de définir l'ensemble des écoulements fonctionnellement possibles dans le système. Un écoulement K-compatible caractérise donc une fonction élémentaire (ou une séquence de fonctions élémentaires) du système.

Dans la suite, on ne considère que des transitions sans prédicat de contraintes (tous les écoulements sont considérés comme K-compatibles).

I - 3. ECOULEMENTS, SOUS-ECOULEMENTS ET TEST

Un écoulement correspond à la notion intuitive d'ensemble minimal de matériel pouvant fonctionner complètement et isolément du reste du matériel :

- C'est un sous-réseau isolable : pour toute transition appartenant au sous-réseau et qui définit une opération, les modules d'où provient et où va l'information nécessaire à cette opération, doivent tous appartenir au sous-réseau;
- Il peut fonctionner complètement : il doit contenir une source et un puits (introduction de l'information initiale, sortie de l'information transformée).

α) *Les écoulements et le test*

On a vu que les deux premières étapes de la maintenance sont :
a) vérification du contrôle minimal pour assurer ou faciliter le test
b) vérification de la partie opérative du système.

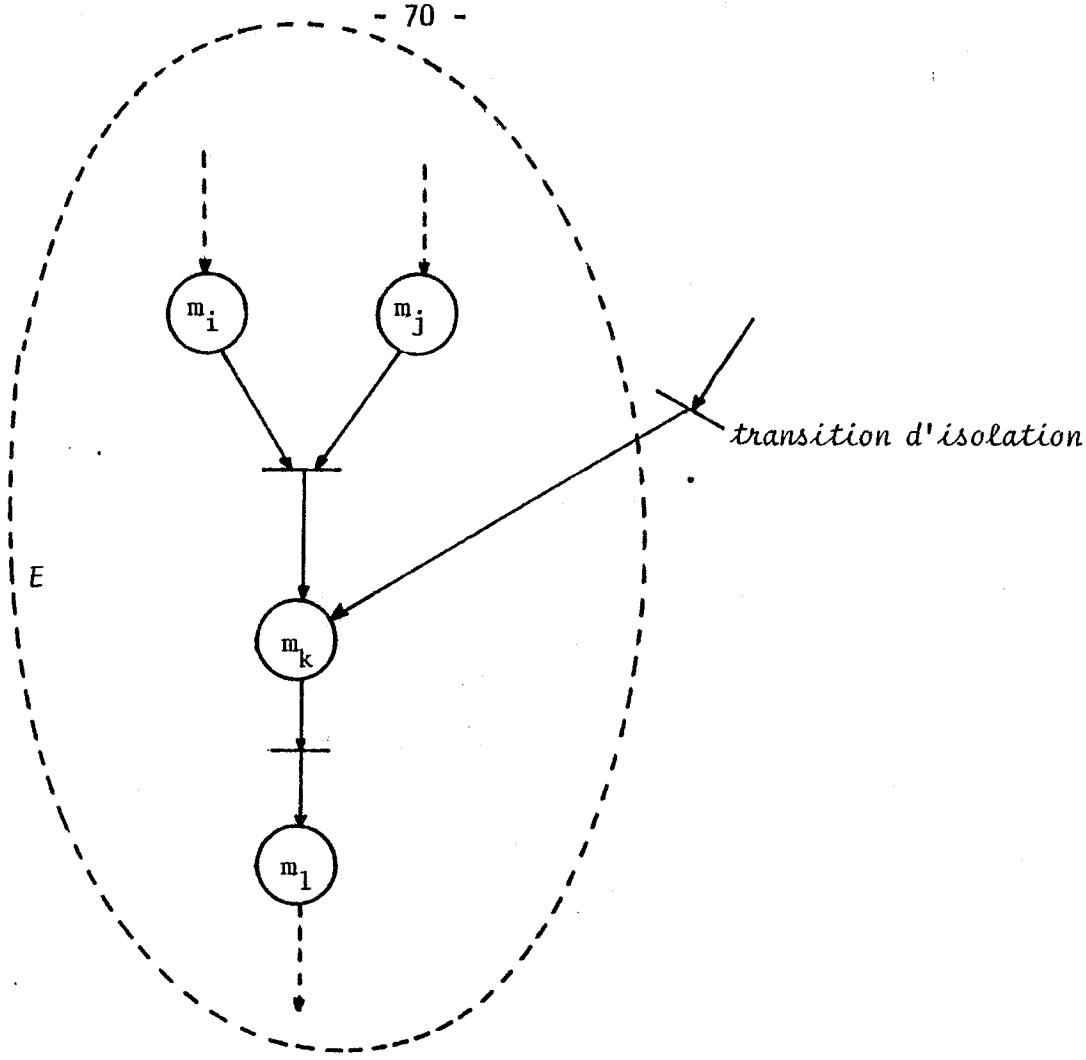
. Le point b) consiste en la détermination d'un ensemble -minimal- d'écoulements permettant de tester -et localiser- les modules défectueux. Ce point est étudié en détail au §II.

. Le point a) peut être assuré comme suit :
Soit $E_1 \ldots E_p$ l'ensemble des écoulements du système assurant un test complet.

On définit l'ensemble d'isolation I d'un écoulement E comme l'ensemble des transitions antécédentes des places de E qui n'appartiennent pas elles-mêmes à E.

$$I = \{\Gamma_G^-(z),\ \forall z \in E\ ;\ \Gamma_G^-(z) \notin E\}$$

On appelle transitions d'isolation les éléments de cet ensemble et degré d'isolation la cardinalité de cet ensemble.



Remarque : L'ensemble d'isolation est relatif aux entrées sur un noeud attribution.

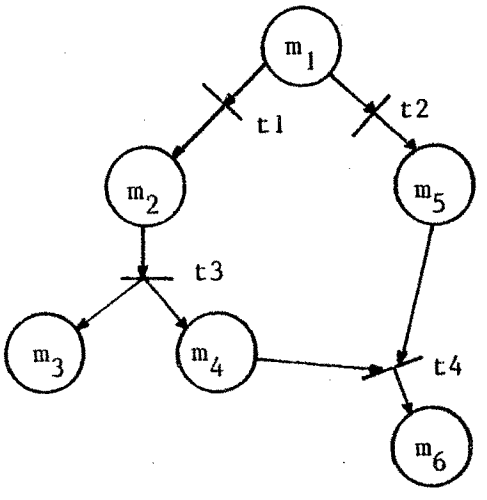
Si l'on inhibe les transitions d'isolation, l'écoulement est dit "isolé" : aucune information ne peut venir de l'extérieur de l'écoulement dans l'écoulement considéré (pas de perturbation). Le contrôle minimal sera alors la partie du contrôle permettant d'assurer l'isolation des écoulements $E_1 \ldots E_p$: on vérifiera les transitions d'isolation au préalable.

Ce contrôle minimal, très restrictif, peut être étendu comme suit : vérification du contrôle assurant l'ensemble des transitions activées par les écoulements $E_1 \ldots E_p$; ainsi, en cas d'erreur, toute accusation ne porte que sur les modules eux-mêmes (c'est-à-dire la partie opérative).

β) *Les sous-écoulements et le test*

Définition : un sous-écoulement E' de E est un sous-ensemble de E tel que toute transition ait un seul successeur et un seul prédécesseur : sous-écoulement commandable et observable.
Un sous-écoulement peut être entièrement défini par ses places.
On définit l'ensemble de complétude d'un sous-écoulement E' de E comme l'ensemble des arcs connectant une place de E' à une place de $(E - E')$ et vice versa.
On appelle arcs de complétude les éléments de cet ensemble et degré de complétude la cardinalité de cet ensemble.

Exemple : Soit l'écoulement E suivant :



Les sous-écoulements sont :

$E'_1 = (m_1, m_2, m_4, m_6)$

$E'_2 = (m_1, m_2, m_3)$

$E'_3 = (m_1, m_5, m_6)$

Soit le sous-écoulement E'_1 :
$t_3 - m_3$ et $m_5 - t_4$ sont les arcs de complétude de E'_1 par rapport à E.

Remarque : L'ensemble de complétude est relatif aux arcs entrants et sortants d'un noeud ET (distribution, jonction).

Si l'on peut forcer une valeur quelconque sur les arcs de complétude (points de test ou de forçage) le sous-écoulement se ramène à un écoulement. La notion de sous-écoulement sera utilisée lorsque les écoulements ne fournissent pas assez de renseignements pour le diagnostic.

I - 4. DEFINITIONS

. Hypothèse : dans un premier temps, on fait l'hypothèse suivante :
- tout arc arrivant sur une place permet d'acheminer toutes les données nécessaires au test de cette place,
- tout arc partant d'une place permet d'observer tous les résultats de test issus de cette place.

. Il s'ensuit que tout module m_i appartenant à un écoulement E_j peut être potentiellement testé par E_j. On dit que m_i est couvert par E_j.

On définit la matrice de couverture $M = |m_{ij}|$
à m lignes $E_1 \ldots E_m$ et n colonnes $m_1 \ldots m_n$

telle que $m_{ij} = 1$ si m_j est couvert par E_i
$m_{ij} = 0$ sinon.

. Soient deux modules m_i et m_j couverts par le même ensemble d'écoulements (colonnes identiques dans M); m_i et m_j sont dits <u>indiscernables</u>; au niveau du test, ceci implique que la localisation ne pourra pas se faire à m_i ou m_j près.

. Soient deux écoulements E_i et E_j couvrant le même ensemble de modules (lignes identiques dans M); E_i et E_j sont dits <u>équivalents</u>.

. Une <u>matrice réduite</u> R est déduite de la matrice de couverture
- en supprimant tout module m_j indiscernable d'un module m_i
- en supprimant tout écoulement E_j équivalent à un écoulement E_i.

<u>Exemple</u> :

Soit le graphe d'un système donné en Figure 1.5.

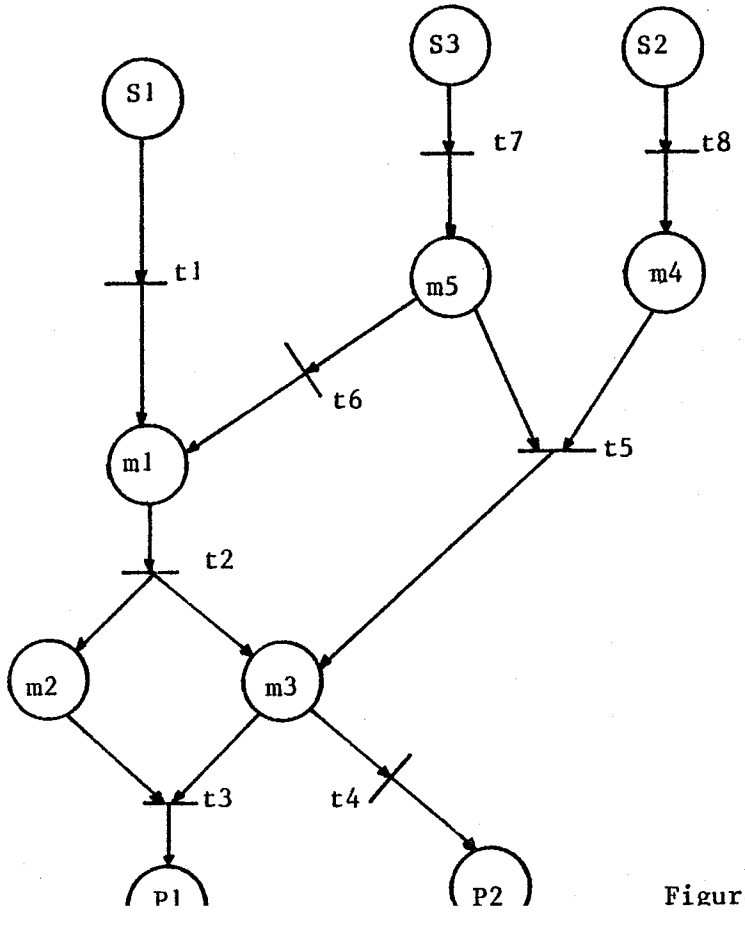


Figure 1.5. Exemple

Le système comporte 3 sources (S1,S2,S3) qui génèrent l'information de test et deux puits (P1,P2) où peuvent être observés les résultats de test; il est par ailleurs constitué de cinq modules (m1...m5).

. Le réseau comporte 6 écoulements :

E_1 = (S1,m1,m2,m3,P1; t1,t2,t3)

E_2 = (S1,m1,m2,m3,P1,P2; t1,t2,t3,t4)

E_3 = (S2,S3,m4,m5,m3,P2; t7,t8,t5,t4)

E_4 = (S1,S2,S3,m1,m2,m3,m4,m5,P1; t1,t2,t3,t5,t7,t8)

E_5 = (S3,m5,m1,m2,m3,P1; t7,t6,t2,t3)

E_6 = (S3,m5,m1,m2,m3,P1,P2; t7,t6,t2,t3,t4)

Les écoulements minimum sont E_1,E_3 et E_5

Les écoulements simples sont E_1,E_3 et E_5

Les écoulements sont tous élémentaires.

Sous-écoulements :

Par exemple E_1' = (S1,m1,m2,P1)

E_1'' = (S1,m1,m3,P1)

Matrice de couverture

M	m1	m2	m3	m4	m5
E_1	1	1	1	0	0
E_2	1	1	1	0	0
E_3	0	0	1	1	1
E_4	1	1	1	1	1
E_5	1	1	1	0	1
E_6	1	1	1	0	1

Les modules m1 et m2 sont indiscernables.

Les écoulements E_1 et E_2 d'une part, E_5 et E_6 d'autre part sont équivalents.

On en déduit la matrice réduite

R	m1	m3	m4	m5
E_1	1	1	0	0
E_3	0	1	1	1
E_4	1	1	1	1
E_5	1	1	0	1

I - 5. APPLICATION AU TEST

Le modèle est utilisé en deux étapes :

Etape 1 : analyse du modèle non paramétré, dit modèle fonctionnel; le réseau est étudié en terme d'écoulements et permet d'explorer les possibilités
- de test complet
- de localisation

pour les diverses stratégies de test connues, avec leurs limites (degré de résolution maximum,...).

Etape 2 : l'introduction de renseignements sur le modèle, dit modèle structurel, permet
- de définir la politique de test optimale quand aucune modification n'est apportée
- de proposer des points d'accès supplémentaires permettant d'accroître la testabilité du système (réduction du coût du test, résolution plus fine).

Les renseignements peuvent être de deux types :
- . structure des modules : combinatoire/séquentiel, complexité, nombre de pannes possibles,...
- . capacité d'information des lignes de données matérialisant les arcs du réseau.

Les renseignements, utilisés au moyen d'outils de théorie de l'information, permettent de définir trois paramètres : la commandabilité, l'observabilité, la testabilité.

II - ANALYSE DU MODÈLE FONCTIONNEL

Au niveau fonctionnel, on s'intéresse au choix d'une politique de diagnostic, avec comme objectif :

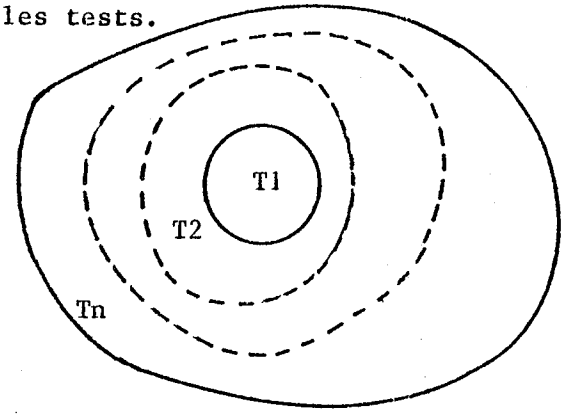
- la vérification du bon fonctionnement du système,
- la localisation de la partie matérielle défectueuse.

On considèrera plus particulièrement l'approche structurelle progressive ("start-small" ou boule de neige) et l'approche par recoupements ("multiple clue") [DEN,68].

II - 1. L'APPROCHE STRUCTURELLE PROGRESSIVE

A - Définition

L'approche structurelle progressive ou approche start-small consiste à tester au préalable une petite partie de matériel, en général la plus petite quantité permettant de dérouler les tests ultérieurs (hardcore). Pour chaque test supplémentaire, on considère une partie du matériel ne dépendant que des parties précédemment testées (accès et observation de l'information de test).
Lorsqu'un test détecte une erreur, on en déduit que l'élément défectueux appartient au matériel rajouté pour ce test; une erreur est réparée avant de poursuivre les tests.



Un test Ti est défini par un écoulement E et un ensemble de données (opérandes) [ROB,78]; son application doit permettre de localiser la panne à l'un des modules composant E.

Cette stratégie est une méthode ascendante fondée sur une analyse des chemins de données dans le système; elle est particulièrement bien adaptée au cas où une très grande liberté est donnée au niveau contrôle (utilisation d'activations fonctionnelles élémentaires).
Cette approche nécessite donc un ordonnancement des modules en couches ordonnées par rapport au test : $C_1 \ldots C_n$.

B - Algorithme d'ordonnancement

Soit la matrice réduite R

pas i : création de la couche C_i

1. Choisir un écoulement E_j de cardinalité minimale, c'est-à-dire la ligne ayant le nombre minimal de "1".
   Les modules couverts par E_j constituent un bloc et appartiennent à C_i.
2. Marquer d'une astérisque les écoulements couvrant les modules trouvés précédemment, soit $\{E_k^*\}$;
   Supprimer les écoulements trouvés en (1) ainsi que les modules couverts par ces écoulements.
3. Dans la matrice R^* ainsi obtenue
   Existe-t'il un écoulement de cardinalité minimale qui soit non marqué ?
   Si oui, le choisir et classer les modules couverts dans C_i; retourner en 2,
   Sinon, démarquer la matrice et aller au pas (i+1) de l'algorithme avec la matrice obtenue.

L'algorithme se termine lorsque tous les modules ont été classés.

C - Mise en oeuvre du test

- L'ensemble des modules d'une couche C_i appartenant au même écoulement constitue un bloc B_{ij} : $C_i = \{B_{ij}\}$.
- On teste l'ensemble des blocs d'une couche C_i avant ceux de C_{i+1}.
- Le test des blocs d'une même couche peut se faire simultanément si les écoulements associés à ces blocs sont disjoints (condition suffisante mais non nécessaire); sinon l'ordre de test est indifférent à l'intérieur d'une couche.

. La localisation se fait au bloc près à l'intérieur d'une couche.

. Tout bloc constitué de plus d'un module est dit bloc d'indiscernabilité : la localisation ne peut se faire au module près à l'intérieur d'un tel bloc.

D - Exemple

Considérons l'exemple de la figure 1.5 dont la matrice réduite R est la suivante :

R	m1	m3	m4	m5
E_1	1	1	0	0
E_3	0	1	1	1
E_4	1	1	1	1
E_5	1	1	0	1

avec m1 et m2 indiscernables

pas 1 : 1. Choisir E_1 = {m1,m3} ; (m1,m3) $\in C_1$

2. Marquer E_3, E_4 et E_5 qui couvrent m1 et m3; supprimer la ligne E_1, les colonnes m1 et m3.

3. Dans la matrice obtenue il n'existe pas d'écoulement non marqué; démarquer la matrice.

pas 2 : La nouvelle matrice est

	m4	m5
E_3	1	1
E_4	1	1
E_5	0	1

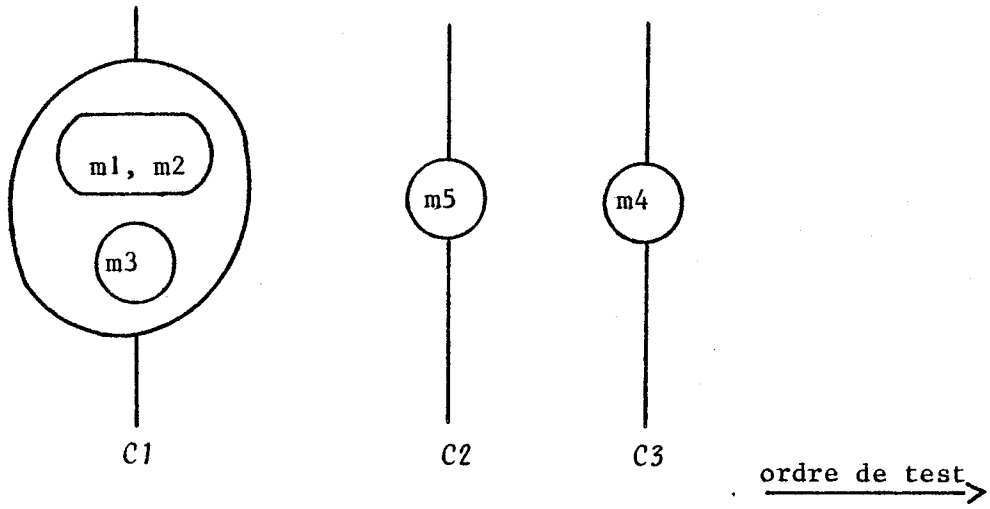
1. Choisir E_5 = {m5} ; (m5) $\in C_2$

2. Marquer E_3 et E_4 qui couvrent m5; supprimer la ligne E_5 et la colonne m5.

3. Dans la matrice obtenue il n'existe pas d'écoulement non marqué; démarquer la matrice.

pas 3 : 1. Choisir E_3 ou E_4 = {m4} ; (m4) $\in C_3$

On obtient le schéma de test suivant

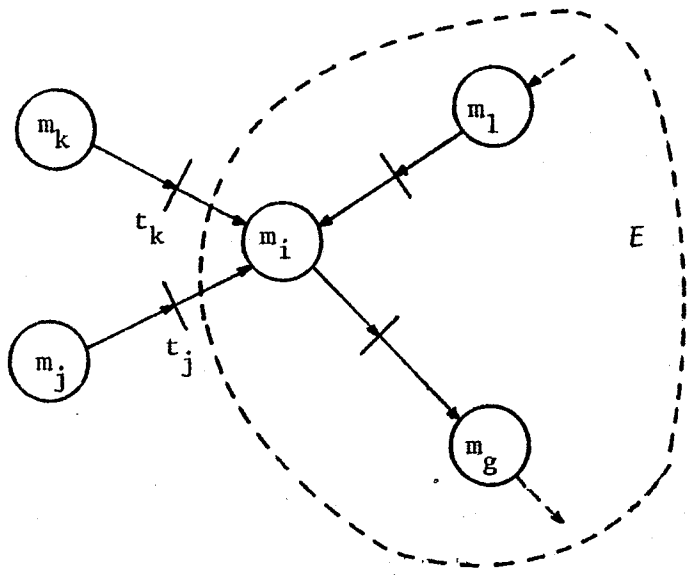


On peut noter que m1, m2 et m3 sont indiscernables : m1 et m2 le sont de façon intrinsèque, c'est-à-dire quelle que soit la méthode choisie; m3 est indiscernable de m1 et m2, du fait de la méthode.

E - Isolation - Sous-écoulements

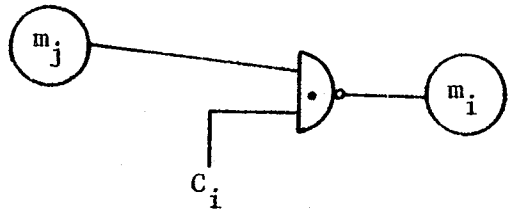
Le problème qui se pose au cours du test est le suivant :
éviter la propagation d'une erreur d'un module non testé vers un module sous test.

Soit un écoulement E dont le degré d'isolation est non nul; il s'agit d'assurer l'isolation par rapport aux modules m_k et m_j.



Dans le cas où la partie contrôle K est testée, l'isolation peut être assurée par affectation d'une valeur bloquante aux transitions t_j et t_k; les transferts $m_k \rightarrow m_i$ et $m_j \rightarrow m_i$ seront bloqués (valeurs neutres en entrée de m_i) et une panne de m_k ou m_j ne se propagera pas vers m_i.

Exemple :



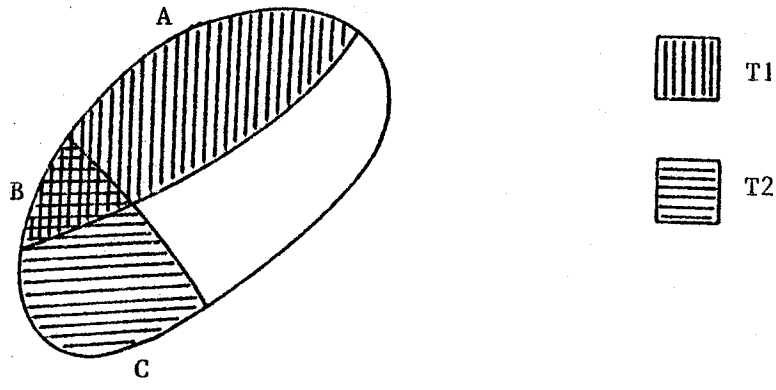
Une panne dans m_j ne peut se propager vers m_i si $C_i = 0$ (transition inhibée).

Utilisation des sous-écoulements et points de forçage : si les écoulements du système ne permettent pas une résolution du diagnostic suffisante, on utilise des sous-écoulements appropriés et en conséquence un ensemble de points de forçage. Si l'on n'a pas de possibilité de forçage, on aura recours à des tests partiels.

II - 2. APPROCHE PAR RECOUPEMENTS

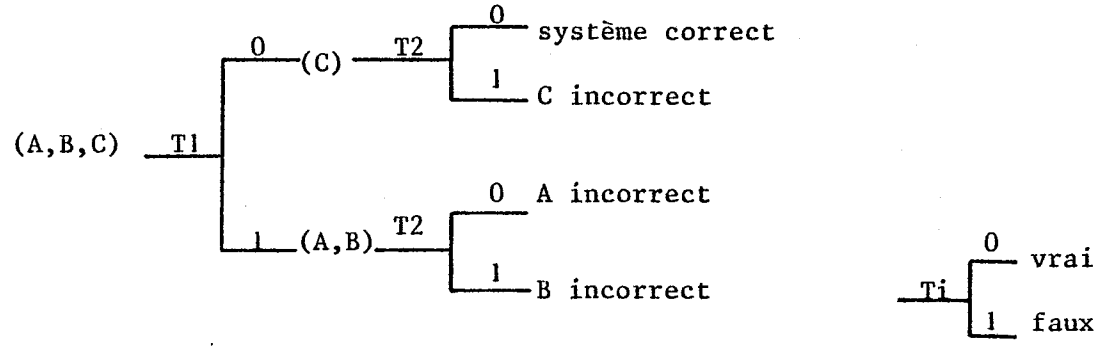
A - Définition

L'objectif d'une telle approche est toujours la détection et la localisation du module (ou unité de remplacement) défectueux; on fait l'hypothèse, dans cette approche, qu'un module au plus est en panne. Cette méthode base son diagnostic sur l'analyse d'une série de tests; une erreur ne met pas fin au test : l'information d'erreur est stockée et le diagnostic se termine par une analyse des données d'erreur.



Dans l'hypothèse de panne simple

si T1 et T2 vrais, les parties A, B, C sont correctes
si T1 vrai et T2 faux, la partie C est défectueuse
si T1 faux et T2 vrai, la partie A est défectueuse
si T1 et T2 faux, la partie B est défectueuse



B - Algorithmes de test et de localisation [RAM,72]

Soit $M = \{m_1, \ldots m_n\}$ l'ensemble des modules
$E = \{E_1, \ldots E_m\}$ l'ensemble des écoulements

α) *Définitions*

. L'ensemble de test est l'ensemble minimal d'écoulements $E1 .. Ep$ tel que tout module mi soit couvert par un écoulement au moins
$\exists E1 \ldots Ep;\ E1 \cup \ldots \cup Ep = \{mi\}$

. L'ensemble de localisation est l'ensemble minimal d'écoulements permettant de localiser une panne au module près.
$\forall$ mi, $\exists Ei1 \ldots Ein;\ \tilde{E}i1 \cap \ldots \cap \tilde{E}im = mi$
$\tilde{E}$ représente soit l'écoulement E soit l'écoulement complémentaire (places non couvertes par E).

β) *Algorithme de recherche de l'ensemble de test* E_T : soit la matrice réduite R

1) Dans la matrice réduite R on cherche les colonnes ayant un seul "1" : modules couverts par un seul écoulement; un tel écoulement est dit obligatoire et appartient à E_T. Supprimer :
- les lignes correspondant aux écoulements obligatoires
- les colonnes ayant un "1" dans ces lignes : modules couverts par un écoulement obligatoire.

Si toutes les colonnes ont été supprimées l'ensemble de test est l'ensemble des écoulements obligatoires; sinon aller en 2.

2) Supprimer :
   - les colonnes mj telles que mj $\supseteq$ mi
   - les lignes Ej telles que Ej $\subseteq$ Ei

   S'il existe dans cette nouvelle matrice des colonnes ayant un seul "1" aller en 1), sinon aller en 3).

3) Chercher une couverture minimale des modules restants par les écoulements restants (méthode de Petrick [PET,56]).

γ) *Algorithme de recherche de l'ensemble de localisation* E_L

Il s'agit de trouver un ensemble minimal d'écoulements tel que tout module (ou tous sous-ensemble de modules indiscernables) défectueux puisse être localisé. On suppose qu'une panne a été détectée par E_T, on applique ensuite E_L pour la localiser. On propose une procédure préfixée : l'ensemble des écoulements de E_L est déterminé indépendamment des résultats de test obtenus après exécution de chacun des écoulements de E_L.

Soit la matrice réduite R; on construit la <u>matrice différentielle</u> *D* déduite de R comme suit :

R |k lignes, p colonnes |→ *D*|k lignes, C_2^P colonnes|; les colonnes représentent la somme modulo 2 de tous les couples de colonnes de R.

A tout couple (mi,mj) de R on associe mij dans *D* tel que mij = mi ⊕ mj

Pour un écoulement *E*k, mij = 1 => *E*k distingue mi de mj

mij = 0 => *E*k ne distingue pas mi de mj

On cherche ensuite, par la méthode de Petrick, une couverture des mij par les *E*k.

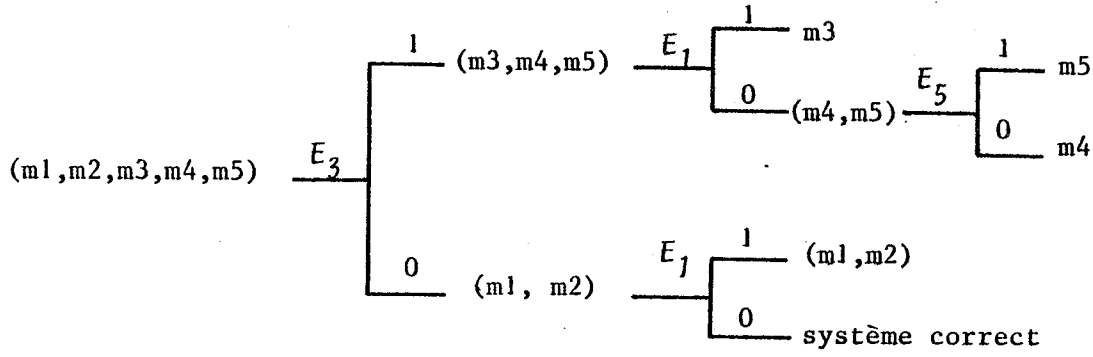
L'ensemble de diagnostic (test + localisation) peut être trouvé par couverture des matrices R et *D*.

C - Exemple

Considérons le système de la figure 1.5. La matrice R réduite et la matrice différentielle D sont alors les suivantes :

	m1	m3	m4	m5	m13	m14	m15	m34	m35	m45
E_1	1	1	0	0	0	1	1	1	1	0
E_3	0	1	1	1	1	1	1	0	0	0
E_4	1	1	1	1	0	0	0	0	0	0
E_5	1	1	0	1	0	1	0	1	0	1

$$E_L = E_3 \cdot E_1 \cdot E_5$$



D - Minimisation

On peut minimiser les ensembles de test et de localisation comme suit : un écoulement est une fonction élémentaire du système à laquelle on peut associer une information d'entrée (opérandes).

Définition : Soit un écoulement E couvrant les modules m1... mq; on appelle écoulement précisé E_p l'écoulement associé à l'ensemble des opérandes de test nécessaires pour tester {m1... mq}

Si Φ_i est l'ensemble des opérandes nécessaires pour tester mi, $E_p = (E, \bigcup_i \Phi i)$.

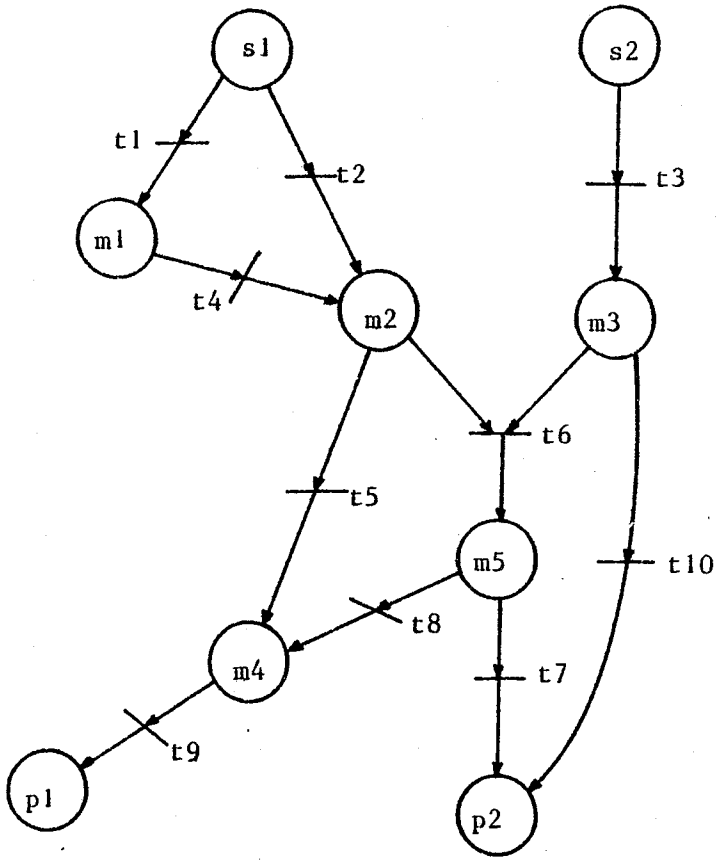
On note $T(E_p)$ le temps requis pour l'exécution d'un écoulement précisé.

Remarque : si n opérandes sont nécessaires pour tester {m1... mq}, $T(E_p) = n.T(E)$ où $T(E)$ est le temps d'exécution de la fonction élémentaire définie par E.

On peut alors faire intervenir ce temps $T(E_p)$ pour choisir un ensemble de diagnostic.

II - 3. EXEMPLE

Soit le système suivant :



Le système comporte deux sources (s1,s2) qui génèrent l'information de test et deux puits (p1,p2) où l'on peut observer les résultats de test.

Le graphe comporte 7 écoulements :

E_1 = {s1,m2,m4,p1; t2,t5,t9}
E_2 = {s1,m1,m2,m4,p1; t1,t4,t5,t9}
E_3 = {s1,s2,m2,m3,m5,p2; t2,t3,t6,t7}
E_4 = {s1,s2,m1,m2,m3,m5,p2; t1,t4,t3,t6,t7}
E_5 = {s1,s2,m2,m3,m5,m4,p1; t2,t3,t6,t8,t9}
E_6 = {s1,s2,m1,m2,m3,m5,m4,p1; t1,t4,t3,t6,t8,t9}
E_7 = {s2,m3,p2; t3,t10}

La matrice de couverture est la suivante

M	m1	m2	m3	m4	m5
E_1	0	1	0	1	0
E_2	1	1	0	1	0
E_3	0	1	1	0	1
E_4	1	1	1	0	1
E_5	0	1	1	1	1
E_6	1	1	1	1	1
E_7	0	0	1	0	0

La matrice réduite R est identique à M.

A - L'approche boule de neige

Pas 1 : couche $C1$

(1). choisir E_7 = {m3}; m3 $\in C1$

(2) . marquer E_3, E_4, E_5 et E_6 (qui couvrent m3); supprimer la ligne E_7 et la colonne m3

(3) . la matrice R^* est la suivante :

R^*	m1	m2	m4	m5
E_1	0	1	1	0
E_2	1	1	1	0
E_3^*	0	1	0	1
E_4^*	1	1	0	1
E_5^*	0	1	1	1
E_6^*	1	1	1	1

choisir E_1 = {m2,m4}; (m2,m4) $\in C1$

(2) . marquer E_2 (qui couvre m2 et m4)

supprimer la ligne E_1 et les colonnes m2 et m4

(3) . il n'existe pas d'écoulement non marqué.

Pas 2 : couche $C2$

La matrice est :

	m1	m5
E_2	1	0
E_3	0	1
E_4	1	1
E_5	0	1
E_6	1	1

(1) . choisir E_2 = {m1}; (m1) $\in C2$

(2) . marquer E_4 et E_6

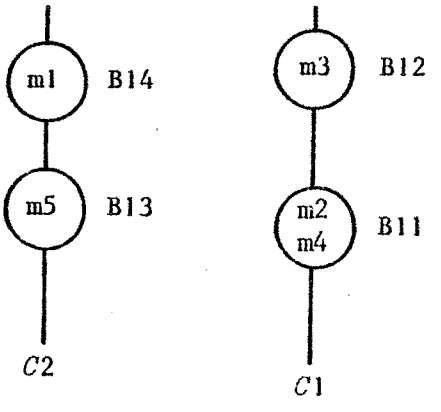
supprimer la ligne E_2 et la colonne m1

(3) . la matrice R^* est

	m5
E_3	1
E_4^*	1
E_5	1
E_6^*	1

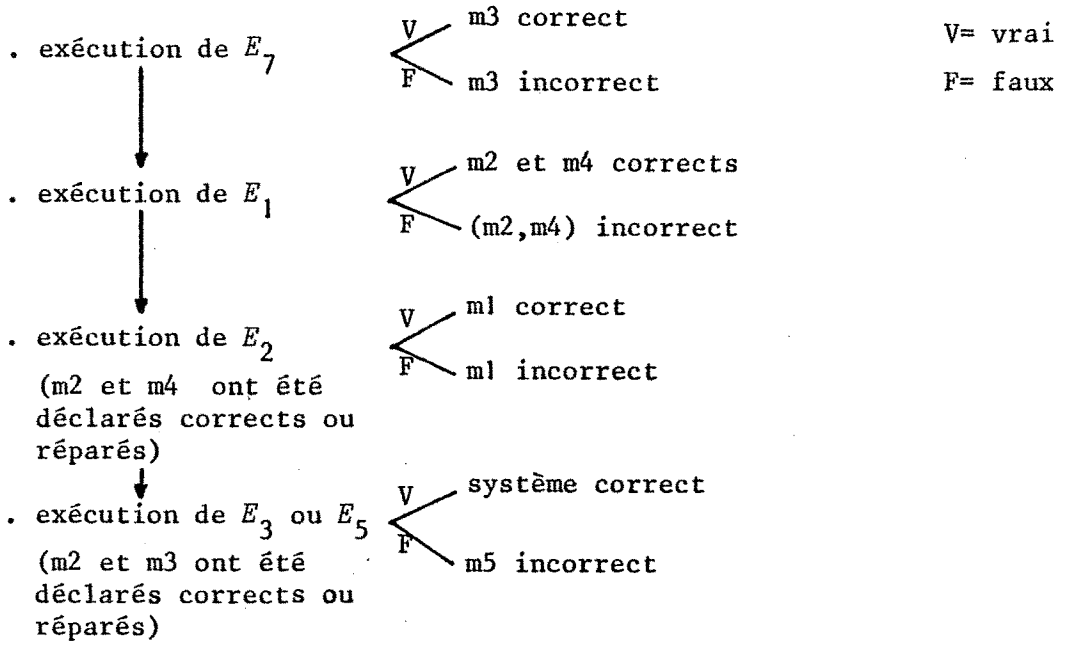
choisir E_3 ou E_5 = {m5}; (m5) $\in C_2$

L'ordre de test est donc le suivant :



Les modules m2 et m4 sont indiscernables.

Dans l'hypothèse où, dès qu'une panne est détectée, le module défectueux est réparé, on peut avoir l'algorithme de test suivant:



B - L'approche par recoupements

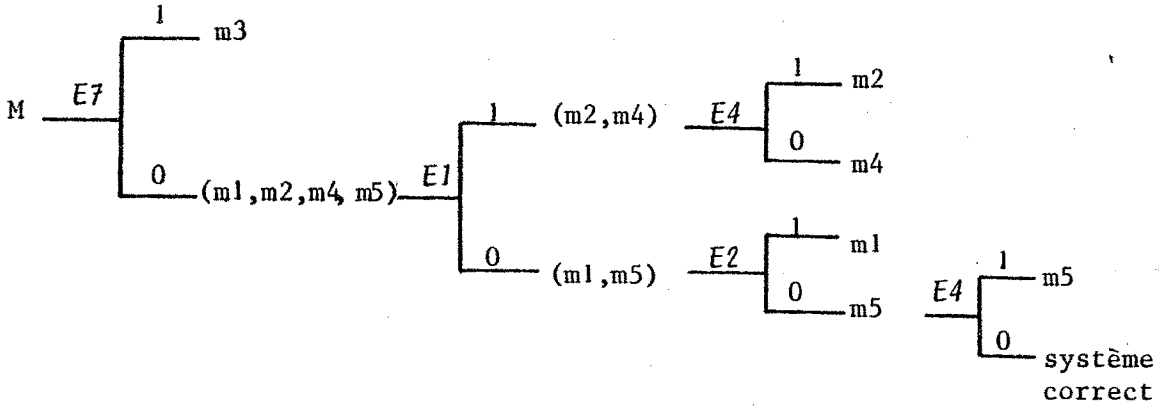
	m1	m2	m3	m4	m5	m12	m13	m14	m15	m23	m24	m25	m34	m35	m45
E_1	0	1	0	1	0	1	0	1	0	1	0	1	1	0	1
E_2	1	1	0	1	0	0	1	0	1	1	0	1	1	0	1
E_3	0	1	1	0	1	1	1	0	1	0	1	0	1	0	1
E_4	1	1	1	0	1	0	0	1	0	0	1	0	1	0	1
E_5	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0
E_6	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
E_7	0	0	1	0	0	0	1	0	0	1	0	0	1	1	0

$$E_L = E_7 \cdot (E_2+E_4+E_6)(E_1+E_3+E_5)(E_1+E_4+E_5)(E_2+E_3+E_5)(E_3+E_4)(E_1+E_2)$$

$$\text{soit } E_L = E_7(E_1\,E_2\,E_3+E_1\,E_2\,E_4+E_1\,E_3\,E_4+E_2\,E_3\,E_4+E_1\,E_4\,E_5+E_2\,E_3\,E_5+E_2\,E_4\,E_5+E_1\,E_3\,E_6)$$

Une solution est $E_L = E_7\,E_1\,E_2\,E_4$

L'algorithme de test est le suivant :



Remarque : On peut voir qu'en appliquant cette approche on obtient une meilleure localisation qu'avec l'approche structurelle progressive : m2 et m4 ne sont pas indiscernables.

On peut noter que :

- l'approche structurelle progressive a l'avantage d'être utilisable en cas de pannes multiples, à l'encontre de l'approche par recoupements;
- par contre, l'approche par recoupements donne la meilleure localisation possible (elle peut distinguer des pannes trouvées indiscernables par l'approche structurelle progressive).

II - 4. EXEMPLE D'APPLICATION : LE SYSTEME E10

On montre ici l'application pratique de cette modélisation sur deux sous-ensembles de l'autocommutateur E10 : le GUR [ANS,77]-[BEL,78] et un organe microprogrammé : l'ELS 48 [KUB,76].

A - Le GUR : Groupe d'Unités de Raccordement

a) Description [ANS,77]

Le groupe d'unités de raccordement s'insère dans le système E10 selon le schéma de la figure 1.6.

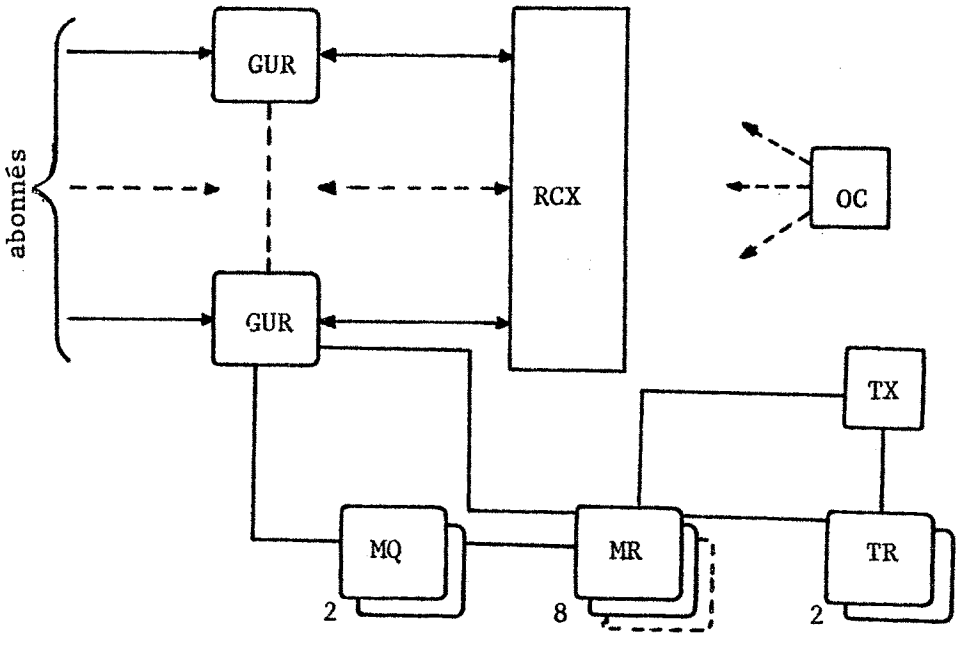


Figure 1.6. Représentation d'un centre E 10.

GUR: Groupe d'Unités de Raccordement
MQ: Marqueur
MR: Multienregistreur
TR: Traducteur
TX: Taxeur
OC: Organe de Contrôle
RCX: Réseau de Connexion

Le GUR réalise la jonction entre des unités de raccordement et le reste de l'autocommutateur (marqueurs, multienregistreurs, etc) par l'intermédiaire de liaisons spécialisées.

La commande est réalisée par l'ensemble MCP-MLE (module de commande programmée - module de liaison et d'échange), les modules MLE étant plus particulièrement chargés des liaisons avec l'autocommutateur. Le raccordement avec les abonnés et les circuits est réalisé par les modules MRX (module de raccordement), dont le nombre maximal est 16.

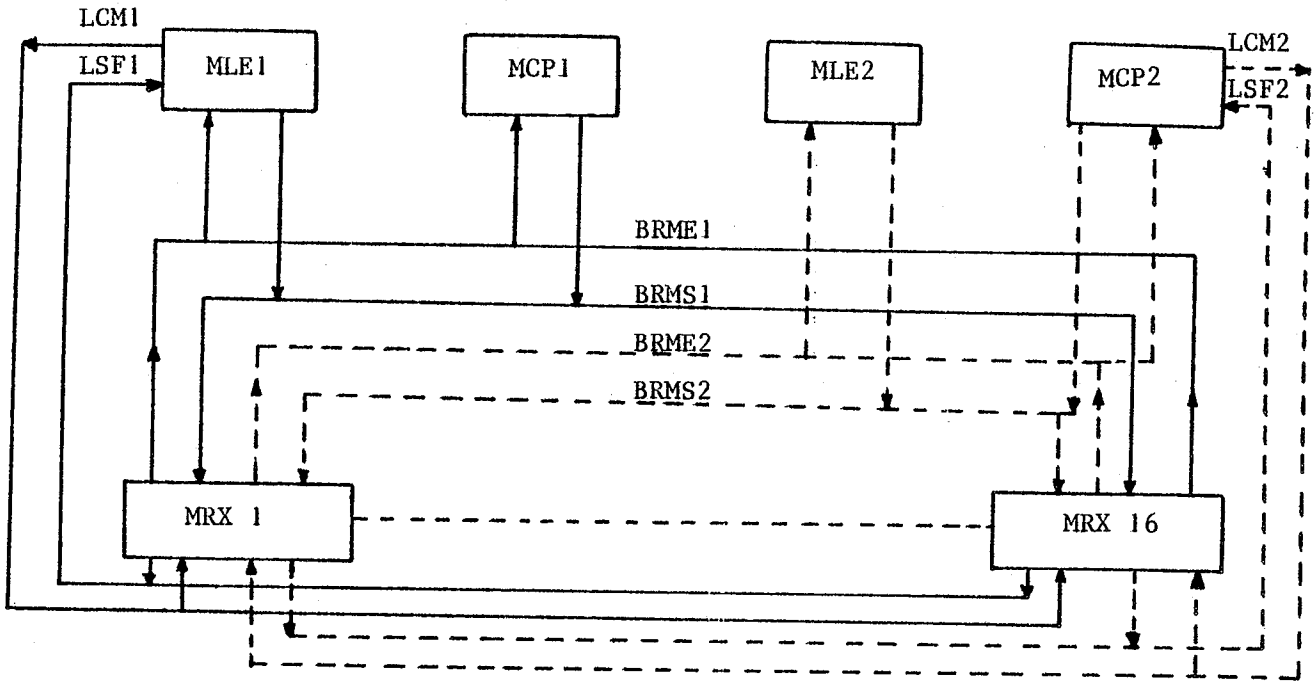


Figure 1.7. Description du groupe d'unités de raccordement

MRX: Module de raccordement
MLE: Module de liaison et d'échange
MCP: Module de commande programmé
LCM: Liaison de commande de module
LSF: Liaison de signalisation de faute
BRME: Bus d'entrée
BRMS: Bus de sortie

α) *Les unités centrales ou modules de commande programmés (MCP)*

Ces unités sont des processeurs microprogrammés, qui ne peuvent être interrompus; le nombre de tâches étant très grand et leur durée faible (< 1 ms), ces unités travaillent en temps réel. La prise en compte d'événements extérieurs se fait par exploration systématique des unités de raccordement dont le MCP a la charge. Le processus que doit exécuter le MCP est écrit dans une mémoire morte (mémoire programme).

β) *Les modules de liaison et d'échange (MLE)*

Chacun de ces modules d'échange est associé à un MCP donné. Chaque MLE est constitué d'automates câblés qui réalisent la jonction entre le MCP qui lui est associé et les autres organes du système E10 (marqueur, enregistreur, réseau de connexion et organe de contrôle). Chaque MLE est connecté à tous les modules de raccordement par les liaisons LCM et LSF.

γ) *Les modules de raccordement (MRX)*

Le type des modules de raccordement est fonction de la liaison à raccorder. Il comportent tous des fonctions communes, en particulier les fonctions non liées à la signalisation et celles concernant les liaisons aux MLE et MCP.

δ) *Les bus d'entrée et de sortie (BRME, BRMS)*

Chaque MCP est connecté au MLE qui lui est associé et à tous les modules de raccordement par deux bus unidirectionnels :

BRMS (MCP → MRX) et BRME (MRX → MCP)

ε) *Les liaisons de commande et de signalisation (LCM et LSF)*

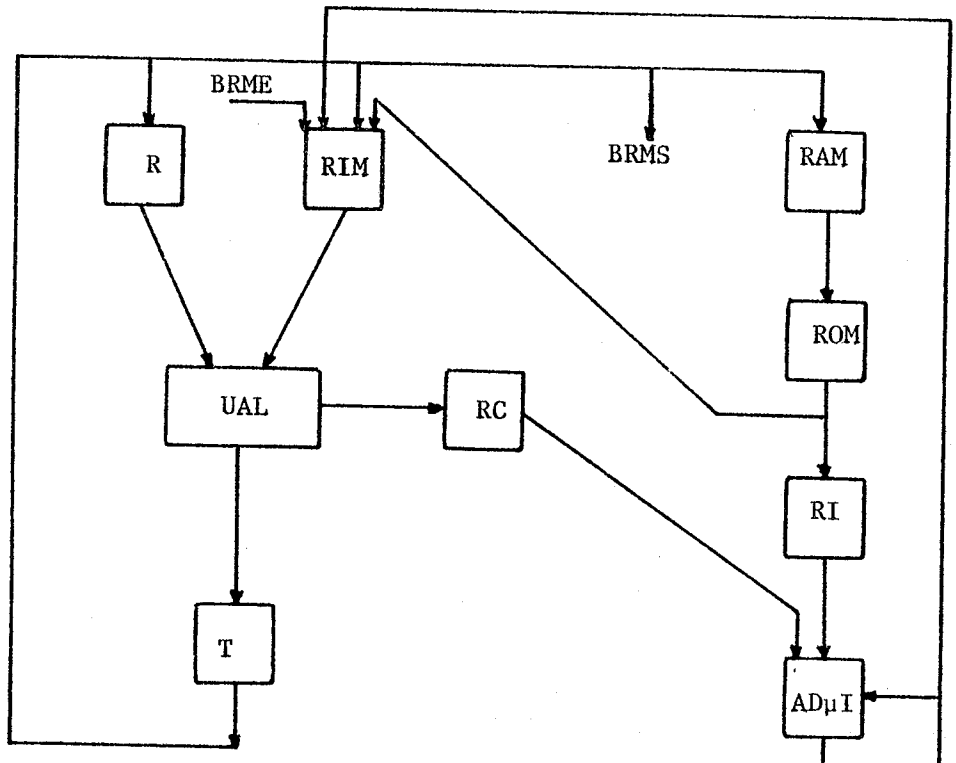
Un module de raccordement est connecté à un MCP -et un seul- par une liaison de commande LCM, positionnée par le MLE correspondant, sur ordre du reste du système E10.
La liaison de signalisation de faute (LSF) est positionnée par les MRX et signale au MLE la présence d'une faute.

b) *Modélisation*

. Description

Considérons le MCP ou Module de Commande Programmé. Il est constitué (Fig.1.8) :

- d'une mémoire de micro-programmes (μI) adressée par le registre ADμI et contrôlée par le registre RI,
- d'une mémoire (ROM) adressée par le registre RIM, lui-même chargé à partir du BUS BRME,
- d'une unité arithmétique et logique UAL et 16 registres généraux notés R,
- le registre RIM peut être chargé à partir du bus interne, du bus BRME ou de la ROM,
- d'un registre de condition RC et un registre T qui mémorise le résultat de l'UAL.



R: Registres généraux
RIM: Registre d'information mémoire
UAL: Unité arithmétique et logique
T: Tampon de sortie de l'UAL
RC: Registre de condition
RAM: Registre d'adresse mémoire
ROM: Mémoire morte
RI: Registre d'instruction
ADμI: Adresse de microinstruction
μI: Micromachine

Figure i.8. Organisation du Module de commande programmé

. Modèle (Fig.1.9)

Les bus BRME et BRMS sont respectivement les places source et puits.

Les écoulements sont :

E_1 = (BRME, R, RIM, UAL, T, BRMS),

E_2 = (BRME, R, RIM, UAL, T, RAM, ROM, BRMS),

E_3 = (BRME, R, RIM, UAL, T, RC, ADμI, BRMS),

E_4 = (BRME, R, RIM, UAL, T, RAM, ROM, RI, ADμI, μI, BRMS).

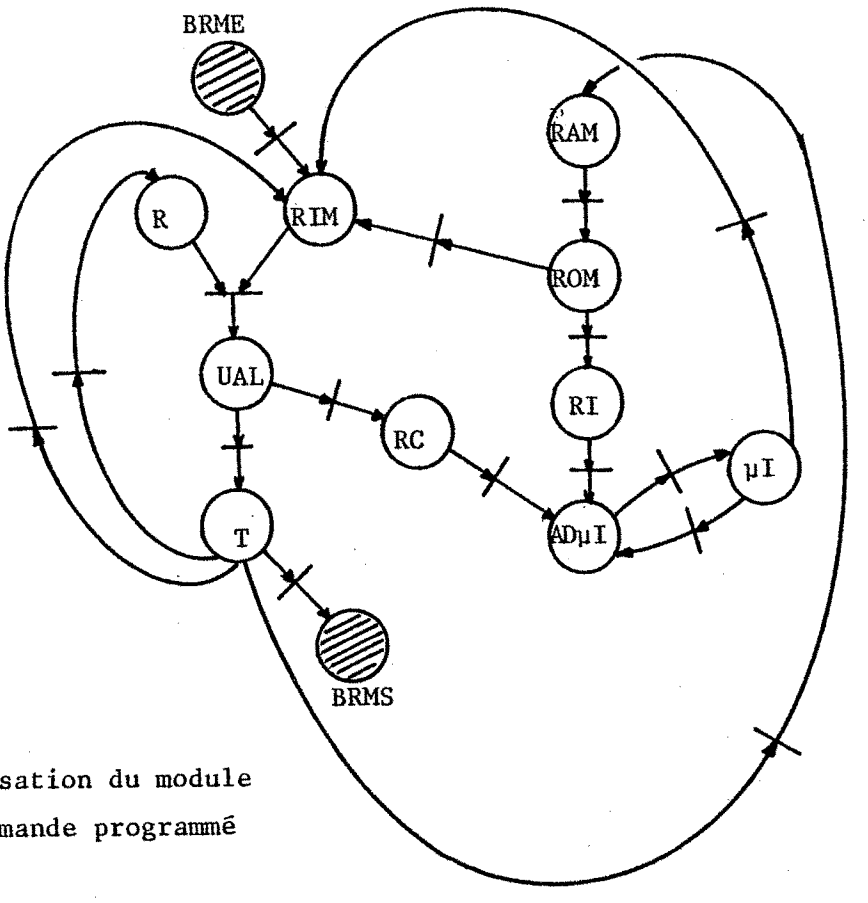


Figure 1.9. Modélisation du module de commande programmé

La matrice de couverture est alors :

$M =$

	R	RIM	UAL	T	RAM	ROM	RI	$AD_\mu I$	μI	RC
E_1	1	1	1	1	0	0	0	0	0	0
E_2	1	1	1	1	1	1	0	0	0	0
E_3	1	1	1	1	0	0	0	1	1	1
E_4	1	1	1	1	1	1	1	1	1	0

La matrice réduite R est :

$R =$

	R RIM UAL T	RAM ROM	RI	$AD_\mu I$ μI	RC
E_1	1	0	0	0	0
E_2	1	1	0	0	0
E_3	1	0	0	1	1
E_4	1	1	1	1	0

. Choix d'une politique de diagnostic

Considérons le système de la figure 1.9 et sa matrice de réduction R; l'algorithme d'ordonnancement se déroule comme suit :

1. Choisir E_1 = {R, RIM, UAL, T}.
L'ensemble de ces modules appartient à C_1 et ils sont indiscernables c'est-à-dire que la localisation ne pourra pas se faire au niveau d'un de ces modules.

2. Dans la nouvelle matrice (E_1, R, RIM, UAL et T sont supprimés), choisir E_2 = {RAM, ROM} $\in C_2$.

3. Deux solutions sont possibles :
   soit 3' : choisir E_3 ={ADμI, μI, RC} $\in C_3$,
   soit 3" : choisir E_4 = {RI, ADμI, μI} $\in C_3$,

4. Selon le choix précédent on obtient :
   soit 4' : E_4 = {RI} $\in C_4$,
   soit 4" : E_3 = {RC} $\in C_4$.

La figure 1.10 représente un ordonnancement de test.
La localisation sera faite au niveau d'une couche, qui doit correspondre à l'implantation physique en unités remplaçables. Cette implantation idéale n'est pas toujours possible, lorsqu'une couche devient trop importante. Il devient nécessaire d'ajouter des points d'accès ou d'observation : soit en tenant compte des volumes de matériel impliqué, soit en utilisant des renseignements supplémentaires (capacité d'information).

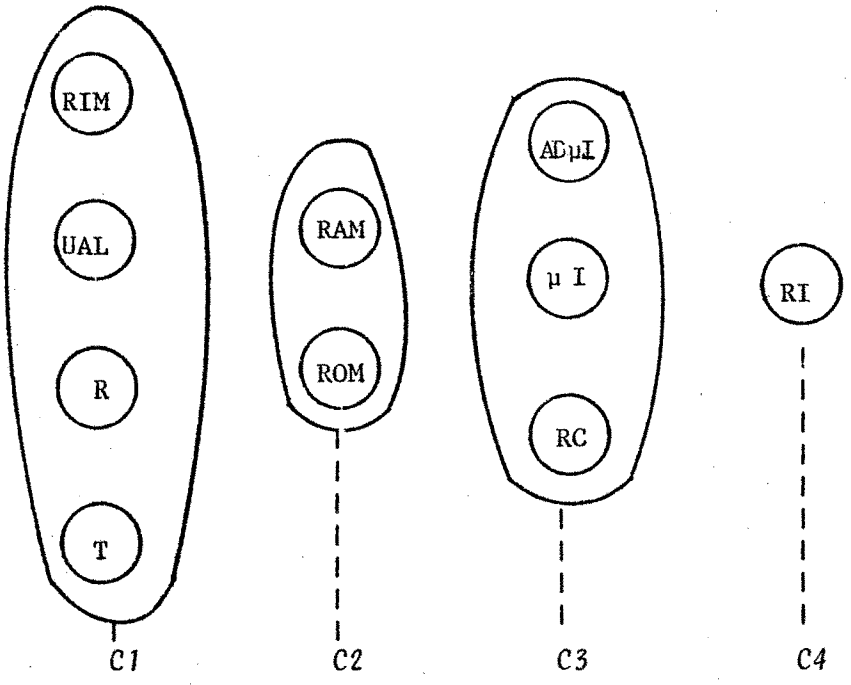
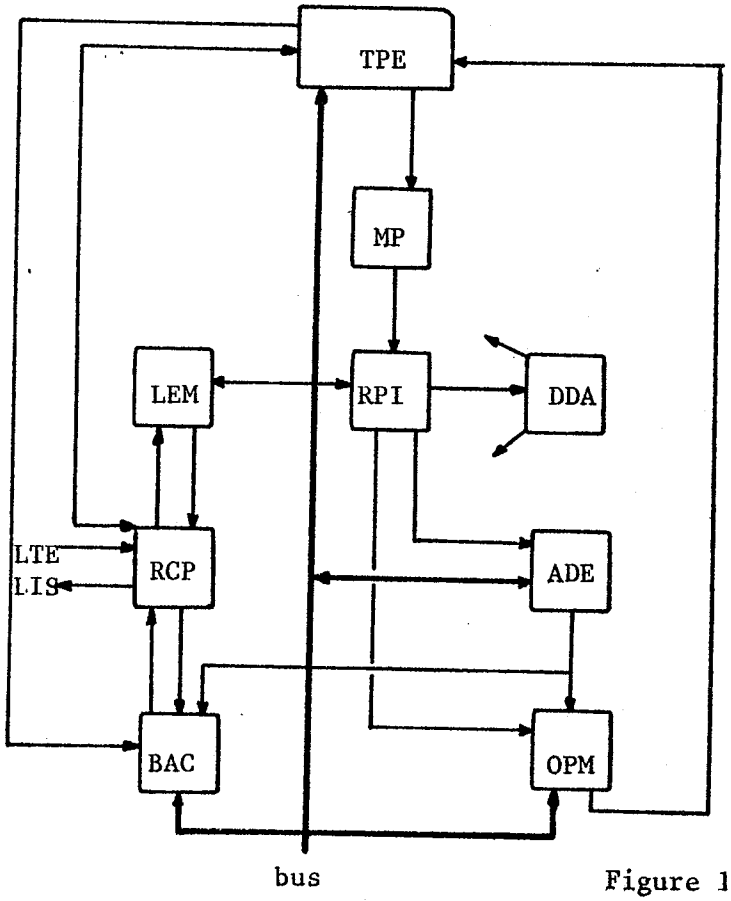


Figure 1.10. Ordonnancement du test Ordre de test: C1 →C2 →C3 →C4

B - L'ELS48

L'ELS 48 est un Equipement Logique Standard qui commande diverses unités de raccordement du centre. C'est un processeur microprogrammé dont la structure est donnée en Figure 1.11.



TPE: traitement des phases
MP: mémoire programme
RPI: regroupement programme
DDA: décodage d'adresse
ADE: accumulateur/ décalage
OPM: opérateur de l'ELS
BAC: bus et accumulateur
LEM: lecture mot programme
RCP: accès pupitre et télépupit
LTE: liaison de test d'entrée
LIS: liaison de sortie

Figure 1.11. Structure de l'ELS 48

La carte DDA décode une partie du mot mémoire et envoie des commandes vers les périphériques d'une part, et vers les autres cartes de la machine d'autre part. Il n'est pas possible d'observer les commandes sortant de DDA. Les cartes ADE et OPM effectuent des opérations arithmétiques et logiques. TPE permet d'adresser la mémoire. BAC permet principalement de lire le BUS ou les accumulateurs de ADE. LEM peut lire un mot de la mémoire programme ou se substituer à celle-ci pour injecter un mot connu dans la machine. Les cartes LEM, RCP, BAC sont destinées au test de la machine. La carte RCP reçoit l'information de test (transmission en mode série). Cette information est distribuée dans la machine. RCP génère des commandes destinées à superviser le test. Ces commandes ne sont pas non plus observables.

L'ELS 48 est modélisé comme suit :

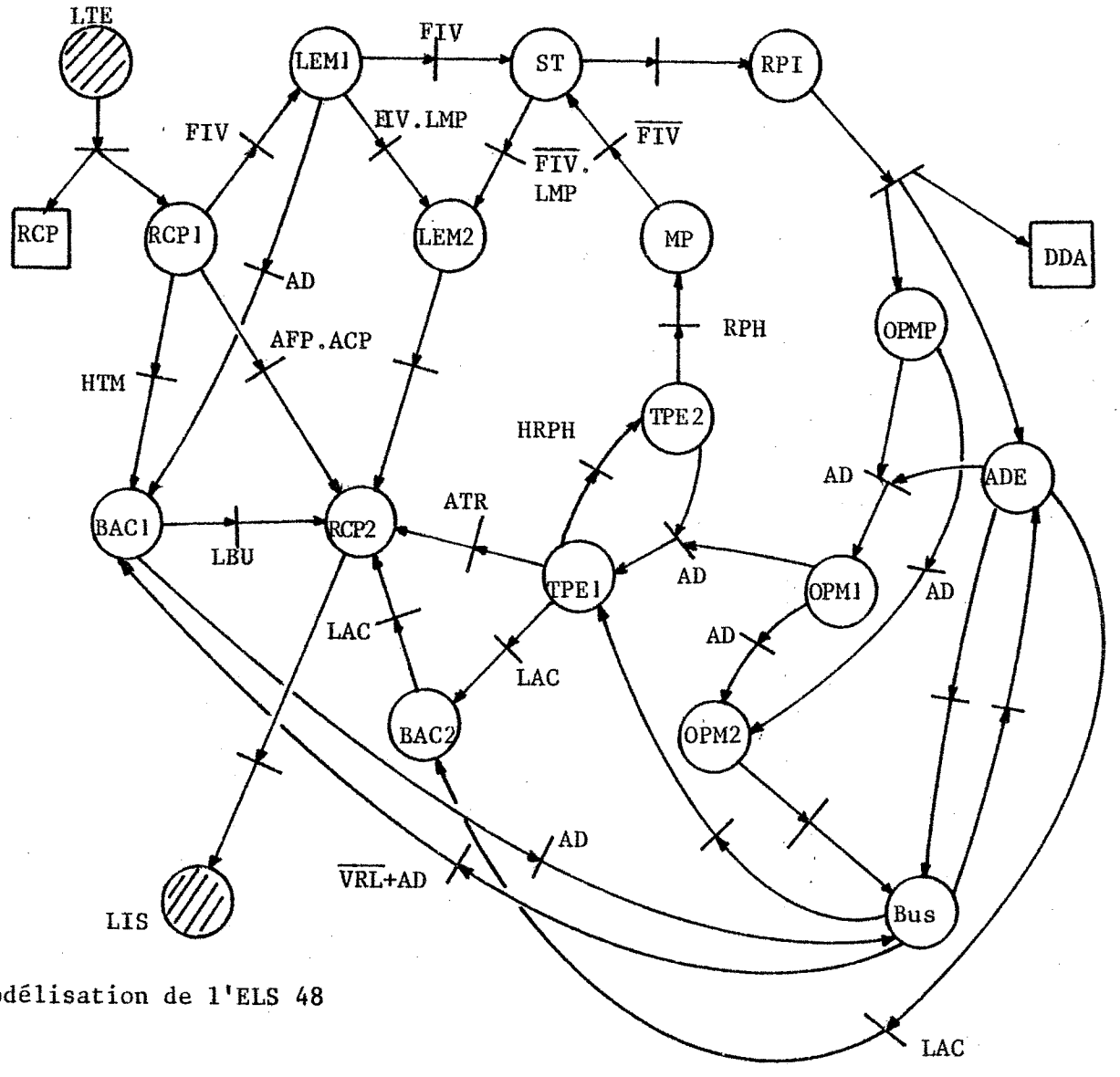


Figure 1.12. Modélisation de l'ELS 48

α) *Liste des écoulements*

E_1 = {RCP1, LEM1, MEM2, RCP2}

E_2 = {RCP1, TEP1, RCP2}

E_3 = {RCP1, BAC1, RCP2}

E_4 = {RCP1, TPE1, BAC2, RCP2}

E_5 = {RCP1, LEM1 ,ST, RPI, ADE, DDA, OPMP, OPM2, BUS, BAC1, RCP2}

E_6 = {RCP1, LEM1, ST, RPI, ADE, OPMP, DDA, OPM1, OPM2, BUS, BAC1,RCP2}

E_7 = {RCP1, LEM1, ST, RPI, OPMP, DDA, ADE, OPM1, TPE1, TPE2, RCP2}

E_8 = {RCP1, TPE1, TPE2, MP, ST, LEM2, RCP2}

β) *Matrice de couverture*

	RCP1	RCP2	LEM1	LEM2	BAC1	BAC2	BUS	TPE1	TPE2	OPMP	OPM1	OPM2	ADE	RPI	ST	MP	DDA
E1	1	1	1	1													
E2	1	1						1									
E3	1	1			1												
E4	1	1				1		1									
E5	1	1	1		1		1			1		1	1	1	1		1
E6	1	1	1		1		1			1	1	1	1	1	1		1
E7	1	1	1					1	1	1	1		1	1	1		1
E8	1	1		1				1	1						1	1	

γ) *Algorithme d'ordonnancement*

Choix de E2 ou E3; par exemple E2

B1 = {RCP1,TPE1,RCP2} ∈ C1

choix de E3 B2 = {BAC1} ∈ C2

" E4 B3 = {BAC2} ∈ C3

" E1 B4 = {LEM1,LEM2} ∈ C4

" E8 B5 = {TPE2,ST,MP} ∈ C5

" E7 B6 = {OPMP,OPM1,ADE,RPI,DDA} ∈ C6

" E6 ou E5 B7 = {BUS,OPM2} ∈ C7

C1 ↔ 2 cartes (unités de remplacement)

C2 ↔ 1 carte

C3 ↔ 1 carte (la même que la précédente)

C4 ↔ 1 carte

C5 ↔ 2 cartes + cartes mémoire

C6 ↔ 4 cartes

C7 ↔ 1 carte + bus

δ) *Amélioration de la localisation*

Couche C1 : l'utilisation des 2 écoulements E1 et E2 permet de lever l'indiscernabilité entre RCP(RCP1+RCP2) et TPE1

→ localisation à 1 carte.

Couche C5 : utilisation de sous-écoulements pour dissocier ST de (TPE2,MP)

Couche C6 : utilisation de sous-écoulements pour distinguer (OPM1,OPMP) de ADE

ε) *Impact sur la conception*

- L'algorithme de diagnostic a permis de mettre en évidence le rôle particulier de la carte DDA. Cette carte, n'étant pas observable, ne peut être testée préalablement. Elle sera donc mise en cause chaque fois qu'un test sera propagé dans un écoulement contrôlé par DDA. Une solution consisterait à partager DDA en deux parties : l'une auto-testable, l'autre observable.

- On n'obtient pas de localisation de faute au niveau de RPI et TPE2. Cela provient de l'impossibilité de les inclure dans des écoulements de taille réduite, dûe à la présence de noeuds ET en entrée ou en sortie. On résoud ce problème :
  . par des points de forçage en entrée : création de sous-écoulements utilisables pour le diagnostic
  . par des points d'observation en sortie : observabilité vers l'extérieur ou vers un autre module.

- Une fois testée, la mémoire programme n'est pas utilisée pour des tests ultérieurs. Sa vérification peut donc être reportée. Une autre solution consisterait à tester la partie minimale de la MP, où seraient stockés lesprogrammes de test du reste de l'ELS.

III - ANALYSE DU MODÈLE STRUCTUREL

Le niveau structurel a pour but l'analyse de la testabilité du système, c'est-à-dire une évaluation de la difficulté de génération de test complet pour obtenir une résolution donnée; cette difficulté peut être par exemple mesurée par le temps de calcul du test et est fonction du choix de la politique de diagnostic.

Dans le modèle fonctionnel, on fait les hypothèses de commandabilité et observabilité équivalentes, soit :

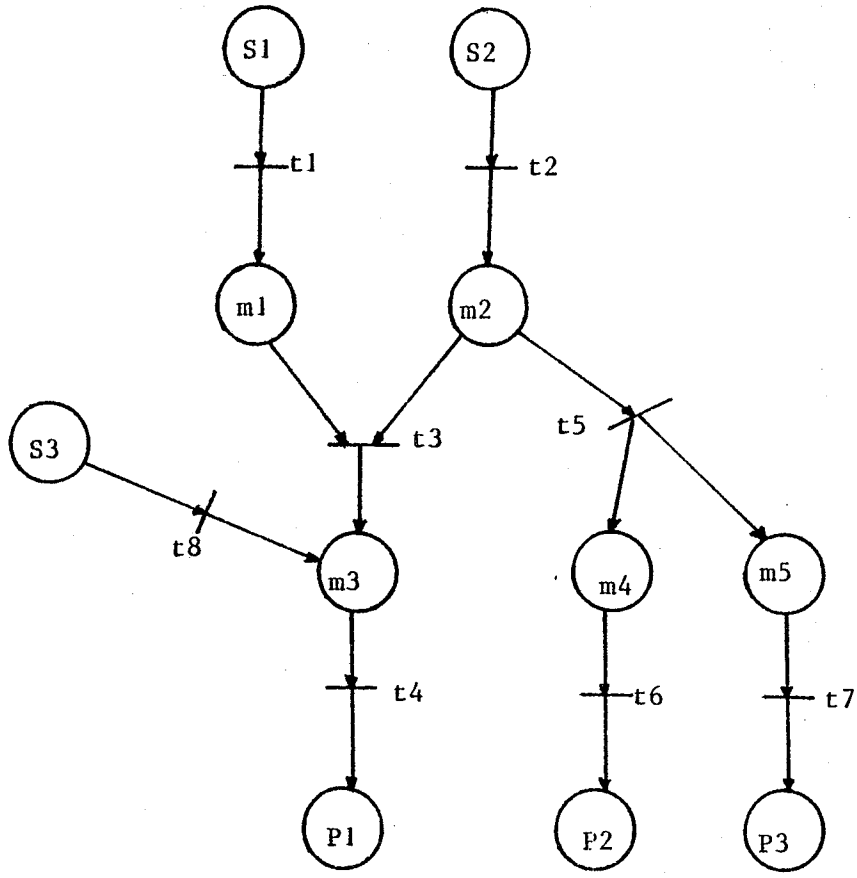
- il est possible d'envoyer des ensembles équivalents de données de test à un module, à travers tous les écoulements partageant ce module,
- de même, à travers tous ces écoulements, on peut obtenir des ensembles équivalents de résultats de test aux sorties du système.

Autrement dit, tous les écoulements sont considérés comme équivalents pour un module donné, par rapport à la capacité de test complet.
Mais cette hypothèse n'est pas vraie en pratique; en effet, il se peut qu'on ne puisse pas :

- acheminer, à partir des sources d'un écoulement donné, toute la quantité de données suffisante au test du module,
- acheminer intégralement vers les puits de l'écoulement, tous les résultats de test issus du module.

Il s'ensuit qu'un module, appartenant à plusieurs écoulements, peut avoir une commandabilité et/ou une observabilité différente pour chacun des écoulements et son test complet peut ne pas être réalisable par un seul écoulement.

Exemple : Soit le système modélisé comme suit :



On suppose que les modules m1... m5 sont implantés sur une même unité remplaçable (pas de problème de localisation).
Le graphe comporte trois écoulements :

$E1$ = (S1,S2,m1,m2,m3,P1; t1,t2,t3,t4)
$E2$ = (S2,m2,m4,m5,P2,P3; t2,t5,t6,t7)
$E3$ = (S3,m3,P1; t8,t4)

Le test complet du système peut se faire par les seuls écoulements $E1$ et $E2$.

La transition t3 permet de commander complètement le module m3 à partir des deux modules m1 et m2; d'autre part, les sources S1 et S2 permettent d'acheminer toutes les données de test aux modules m1 et m2. Il se peut cependant que dans cet écoulement $E1$ on ne puisse obtenir en entrée de m3 toutes les données de test qui lui sont nécessaires : transfert et traitement des données par m1 et m2 diminuent la commandabilité.

La relation de commandabilité complète n'est pas transitive :

(S1 et S2 commandent complètement m1 et m2) et (m1 et m2 commandent complètement m3) n'implique pas (S1 et s2 commandent complètement m3).

On conçoit bien dans cet exemple, que les données que l'on peut acheminer vers m3 par l'écoulement $E3$ (commandabilité complète de m3 à partir de la source S3) peuvent être plus variées que celles acheminées par l'écoulement $E1$.

On pourrait montrer de la même manière que la relation d'observabilité complète n'est pas transitive.

La mise en oeuvre de la méthode des écoulements exige donc que l'on puisse comparer des écoulements partageant un module, par rapport aux paramètres de commandabilité et d'observabilité.

L'introduction, sur le modèle fonctionnel, d'une information structurelle (capacité d'information) va permettre de définir pour chaque module dans chaque écoulement :

- une mesure de commandabilité,
- une mesure d'observabilité

en vue de discriminer ces écoulements entre eux.

D'autre part, après discrimination, il peut rester plusieurs choix d'ensembles d'écoulements pour une stratégie de test donnée; le problème est alors de choisir un ensemble d'écoulements permettant la politique de diagnostic la plus économique en même temps que la plus facile à mettre en oeuvre.

Cette étude est fondée sur l'analyse de la charge de test de chaque module dans chaque écoulement et conduit à la définition d'une mesure de testabilité

Il s'ensuit que:

- Pour une politique de diagnostic donnée on peut choisir l'ensemble d'écoulements rendant la testabilité du système maximale, compte tenu des écoulements fournis par cette méthode (choix d'écoulements, adjonction d'écoulements afin d'assurer un test complet).
- Dans le cas où plusieurs politiques de test peuvent être envisagées, leur testabilité propre permettra de choisir la plus économique ou celle donnant le meilleur degré de résolution.

-Dans le cas où la testabilité s'avère trop faible, les paramètres de commandabilité et observabilité permettent de définir des points d'accès supplémentaires en vue d'augmenter la testabilité globale.

III - 1. NOTATIONS - DEFINITIONS [CAS,77b]

Les problèmes à résoudre sont étroitement liés à la propagation d'une information dans un réseau et on utilisera les concepts de théorie de l'information.

α) *Nomenclature des variables*

Soit Xi la variable entrant dans le module i et Yi la variable sortant de ce module.

X_{Si} représente la variable sortant de la source S_i et Y_{Pj} la variable entrant dans le puits P_j; X_S représente la concaténation de toutes les variables X_{Si} et Y_P la concaténation des Y_{Pj}. (X_S représente un vecteur de test de l'écoulement et Y_P le résultat d'un test).

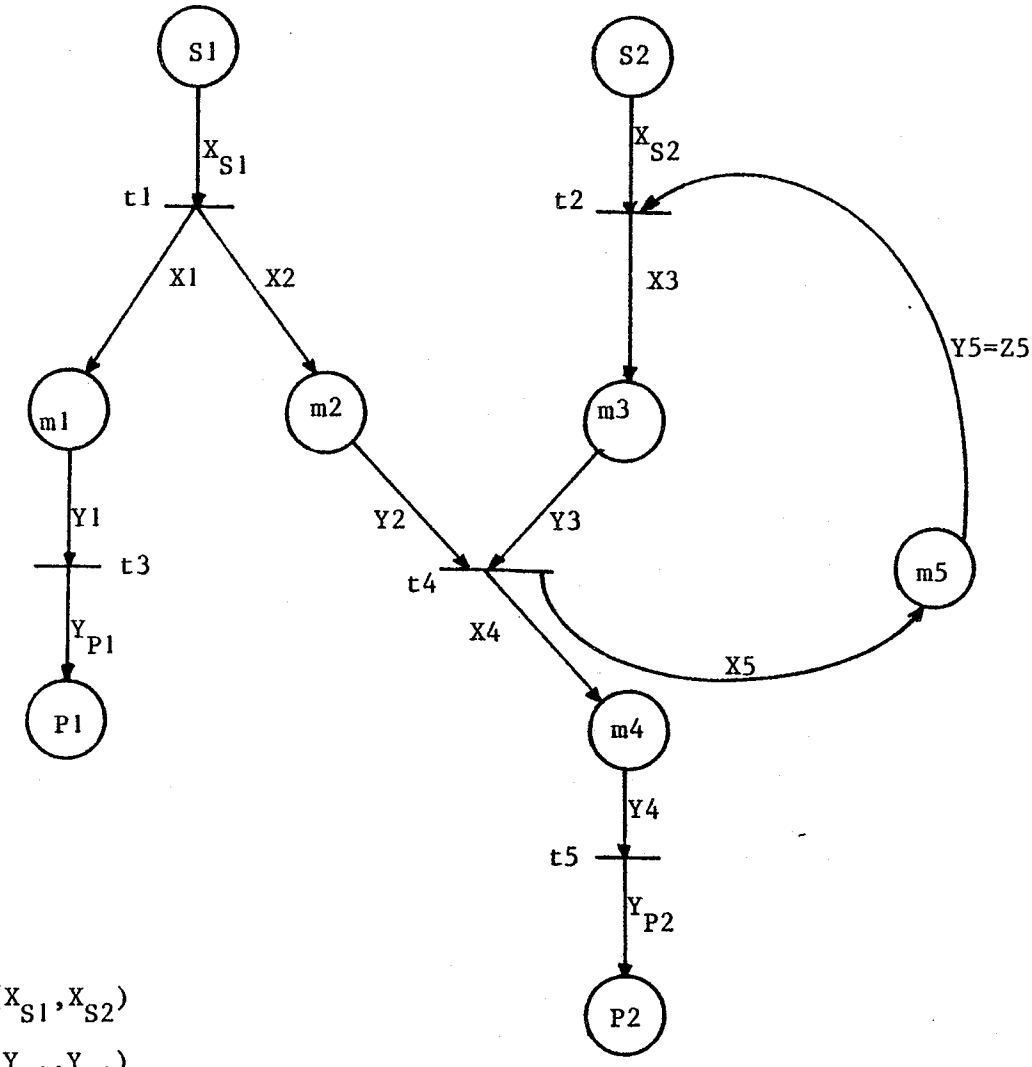
β) *Variables internes*

Dans le cas d'écoulements comportant des boucles, lorsqu'on cherche à transférer des données des sources vers les puits, on rencontre des variables qui ne proviennent pas directement des sources. Les variables internes sont représentées en remplaçant la lettre Y par la lettre Z. Z (sans indice) représente la concaténation de toutes les variables internes de l'écoulement.

Les différentes notations sont représentées dans la figure de la page suivante.

γ) *Capacité d'un arc*

On peut noter chaque arc d'un écoulement par le nom de la variable véhiculée. Les arcs, en tant que canaux d'information, sont caractérisés par leur capacité; cette capacité sera en général le nombre de fils de la ligne physique correspondante, sauf dans le cas de codage redondant de la variable.



$X_S = (X_{S1}, X_{S2})$

$Y_P = (Y_{P1}, Y_{P2})$

$Z = (Z5)$

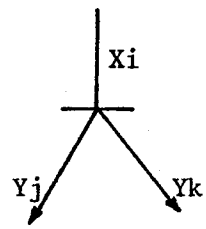
On notera C_u la capacité d'information de la ligne u et $C'_u \geq C_u$ le nombre de fils de cette ligne.

Chaque arc du graphe est alors étiqueté par sa capacité; au niveau des transitions on a les règles suivantes :

transition simple

$C_{Xi} = C_{Yj}$

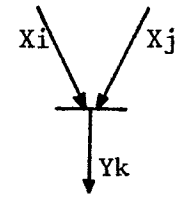
divergence



$C_{Yj} \leq C_{Xi}$

$C_{Yk} \leq C_{Xi}$

convergence



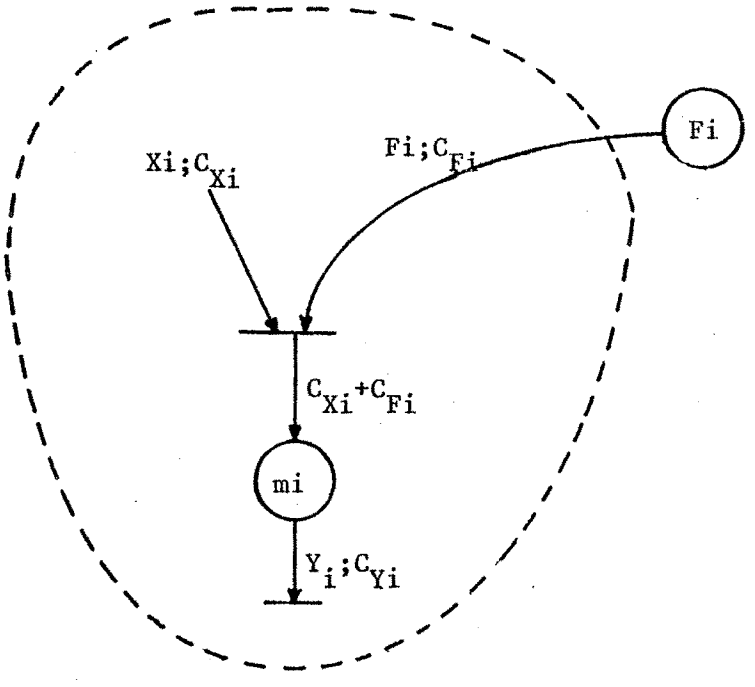
$C_{Yk} \leq C_{Xi} + C_{Xj}$

δ) *L'état de santé d'un module*

Une autre information structurelle utile pour la définition d'une mesure de testabilité est la quantité de matériel d'un module ou, de façon équivalente, le nombre de pannes possibles n_{i-1} qui peuvent survenir dans le module i.

L'ensemble des états faux et l'état sain du module sont représentés par une entrée additionnelle Fi dans le module (méthode du réseau d'injection de pannes [OGU,74] : cette entrée Fi est appelée l'état de santé du module i et peut prendre n_i valeurs possibles. La capacité de la ligne Fi est alors $C_{Fi} = \log_2 n_i$.

La figure suivante représente un module m_i muni de son entrée Fi :



III - 2. MESURE DE COMMANDABILITE

α) *Définition* : la mesure de commandabilité d'un module dans un écoulement représente la quantité d'information maximum qu'un module peut recevoir dans cet écoulement E à partir des sources de E.

Une formulation mathématique de la commandabilité d'un module i dans un écoulement E_j est donc

$$C_i^{E_j} = \sup [H(Xi)]$$

sur l'ensemble des distributions possibles de Xs et des variables internes Z de l'écoulement E_j, où H(X) représente la quantité d'information (ou incertitude) de X.

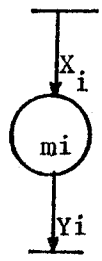
β) *Evaluation pratique* : l'évaluation exacte d'une telle mesure nécessiterait la connaissance des fonctions des modules en même temps qu'un calcul très long, ce qui n'est pas compatible avec l'objectif d'évaluation de la testabilité (évaluation moins coûteuse que le calcul proprement dit des vecteurs de test).
Une solution à ce problème consiste à calculer une entropie modifiée H' [CAS,77a] qui est une majoration de la mesure de commandabilité initiale.
On définit alors :
$C_{Xi}^{E_j}$ = H'(Xi) = capacité minimale d'une coupe qui déconnecte le module m_i des sources et des variables internes de l'écoulement E_j. La capacité de la coupe est la somme des capacités des arcs supprimés par cette coupe.

Dans les cas complexes, la mesure de commandabilité peut être évaluée au moyen des algorithmes classiques de "coupe minimale - flot maximal" [BER,63]; dans les cas simples, le calcul est facilement réalisable à l'aide des règles de propagation vérifiées par H' :

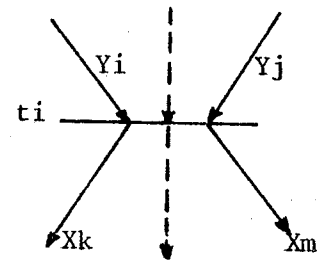
- l'information d'une source est égale à sa capacité; les sources sont indépendantes,
- l'information de sortie d'un module est le minimum de l'information d'entrée et de la capacité de la ligne de sortie,
- l'information sur une ligne de sortie de transition est le minimum de l'information conjointe des lignes d'entrée et de la ligne de sortie,
- L'information conjointe sur l'ensemble des lignes de sortie d'une transition est égale au minimum de l'information conjointe de l'ensemble des entrées et de la somme des capacités des lignes de sortie.

Ces règles sont illustrées dans la figure suivante

a)



b)



a) pour un module combinatoire

$$H'(Yi) = \min \lceil H'(Xi), C_{Yi} \rceil$$

b) propagation par les transitions

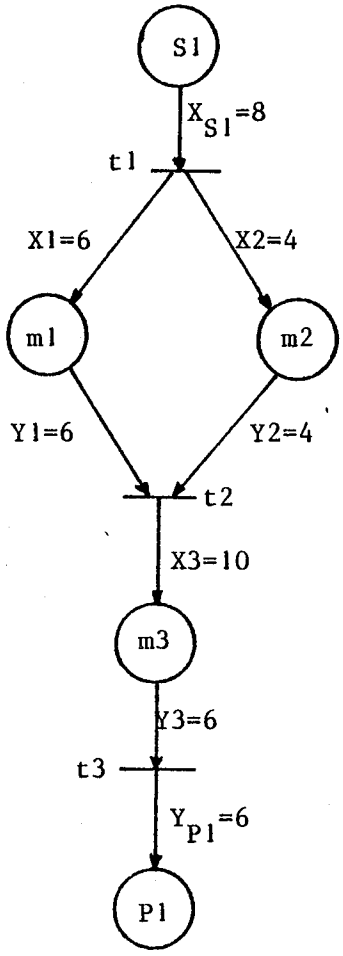
$$H'(Xk) = \min \lceil H'(Yi,\ldots Yj), C_{Xk} \rceil$$

$$H'(Xk,\ldots Xm) = \min \lceil H'(Yi,\ldots Yj), C_{Xk} + \ldots + C_{Xm} \rceil$$

On peut noter que la fonction H' permet d'évaluer la dépendance entre variables : ceci apparaît lorsque $H'(U,V) < H'(U) + H'(V)$.

γ) *Exemples*

Considérons l'écoulement suivant, renseigné par la capacité des arcs :



Etapes du calcul

$H'(X_{S1}) = C_{X_{S1}} = 8$

$H'(X1) = \min(8,6) = 6$

$H'(X2) = \min(8,4) = 4$

$H'(X1,X2) = \min(8,10) = 8$

$H'(Y1) = \min(6,6) = 6$

$H'(Y2) = \min(4,4) = 4$

$H'(Y1,Y2) = \min(8,6+4) = 8$

$H'(X3) = \min(8,10) = 8$

$H'(Y3) = \min(8,6) = 6$

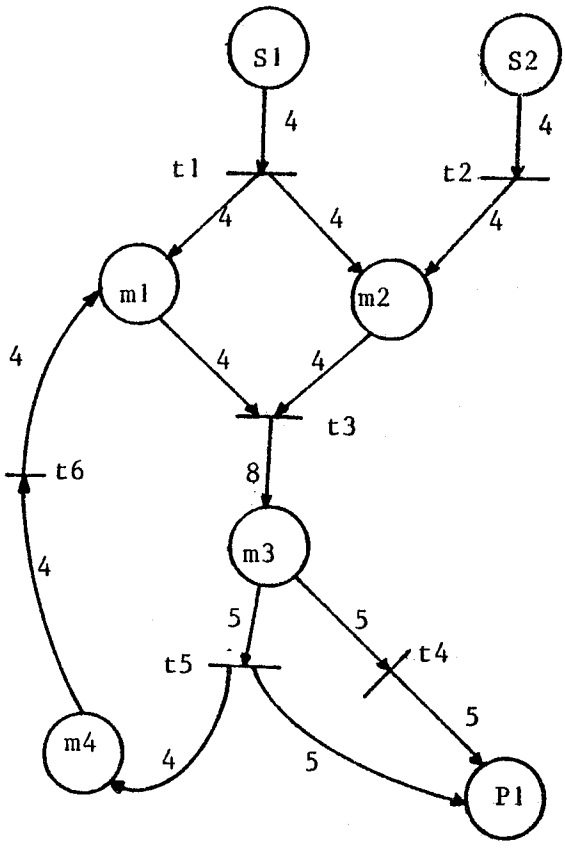
$H'(Y_{P1}) = H'(Y3) = 6$

La commandabilité du module m3 est donnée par:

$C^{E}_{m3} = H'(X3) = 8$

Le module m3 n'est pas complètement commandable par cet écoulement puisque la commandabilité de ce module isolé est égale à $C_{X3} = 10$.

Comparaison de la commandabilité d'un module dans deux écoulements : Considérons l'exemple donné en Figure 1.6. pour lequel on veut comparer la commandabilité du module m3 dans chacun des écoulements $E1$ et $E2$ (en utilisant "coupe minimale - flot maximal").



dans $E1$

H'(XS1) = 4

H'(X1) = min (4,4) = 4

H'(X2) = min (4,4) = 4

H'(X1,X2) = min (4,8) = 4

H'(Y1) = min (4,4) = 4

H'(Y2) = min (4,4) = 4

H'(Y1,Y2) = min (4,8) = 4

H'(X3) = min (4,8) = 4

dans $E2$: $E2$ n'étant pas un écoulement simple, on utilise l'algorithme de "coupe-minimale - flot maximal".

Figure 1.6.

$E1$ = (S1, m1, m2, m3, P1; t1, t3, t4)

$E2$ = (S1, S2, m1, m2, m3, P1; t1, t2, t3, t4).

Dans $E1$, S1 est la seule source et il n'y a pas de variable interne; la suppression de l'arc (S1, t1) déconnecte m3 de la source S1. Puisque cet arc a une capacité 4 et que c'est une coupe minimale, on peut écrire que

$$C_3^{E_1} = 4$$

Dans $E2$, les sources sont S1 et S2 et il n'y a pas de variable interne; il existe plusieurs coupes donnant une capacité minimale, par exemple la suppression de l'arc (t3, m3) de capacité 8. Ainsi

$$C_3^{E_2} = 8.$$

Cette mesure représente la commandabilité maximale de m3 puisque c'est aussi la commandabilité de ce module lorsqu'il est isolé.
Ces résultats montrent que le test de m3 à travers l'écoulement $E1$ est plus difficile qu'à travers $E2$: un certain nombre de vecteurs de test qui détectent une panne dans m3 ne pourront pas être propagés à travers $E1$ ou conduiront à des contradictions lorsqu'on fera la consistence vers S1 (méthode de sensibilisation de chemin).

III - 3. MESURE D'OBSERVABILITE

α) *Définition* : La mesure d'observabilité représente la quantité d'information maximum qu'un module peut transmettre aux puits d'un écoulement E.
La mesure d'observabilité d'un module i dans un écoulement Ej peut être formulée comme suit :

$$\theta_i^{E_j} = \sup H\,[Yp, Z^+/Xs, Z^-]$$

sur l'ensemble des distributions possibles de (Xs, Z^-)

où Z^- représente les valeurs des variables internes avant de passer par m_i et Z^+ les valeurs de ces variables après passage par m_i.
Les mêmes difficultés apparaissent pour l'évaluation pratique de cette mesure.

β) *Evaluation pratique* : On utilisera de même une entropie modifiée et on définit alors

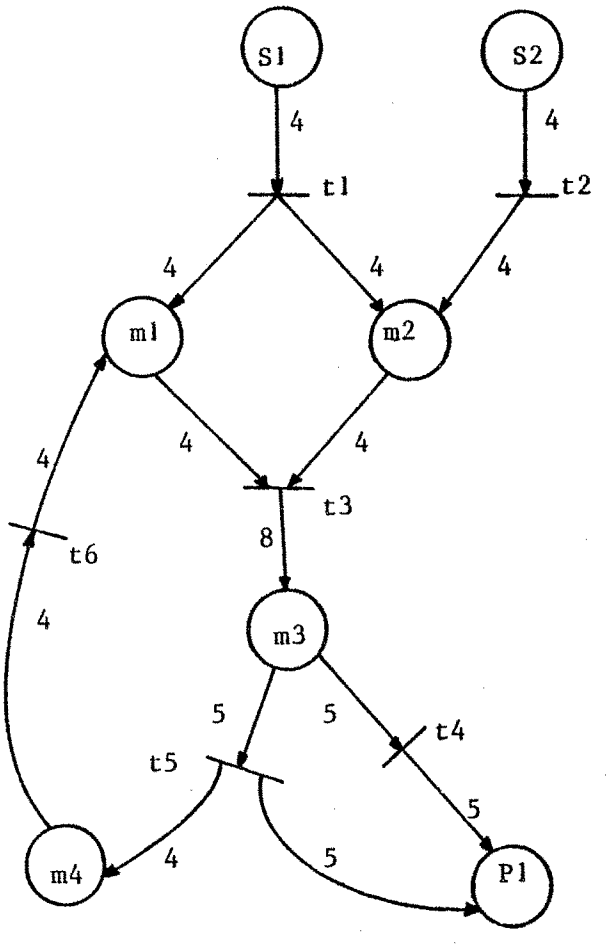
$\theta_i^{E_j} = H'(Yp, Z^+/Xs, Z^-)$ = capacité minimale d'une coupe déconnectant le module m_i de tous les puits et des variables internes de l'écoulement Ej.

Dans les cas simples, les règles de propagation définies pour la mesure de commandabilité permettent de calculer la mesure d'observabilité, avec les règles additionnelles suivantes

$$H'(Xs, Z^+/Xs, Z^-) = 0$$
$$H'(Fi/Xs, Z^-) = H'(Fi) = C_{Fi}$$

Dans les cas complexes, on utilisera un algorithme de "coupe minimale - flot maximal".

γ) *Exemple* : Considérons l'exemple déjà donné en Figure 1.6. et rappelé ci-dessous; comparons l'observabilité du module m2 dans chacun des écoulements $E1$ et $E2$.



$E1$ = (t1,t3,t4)

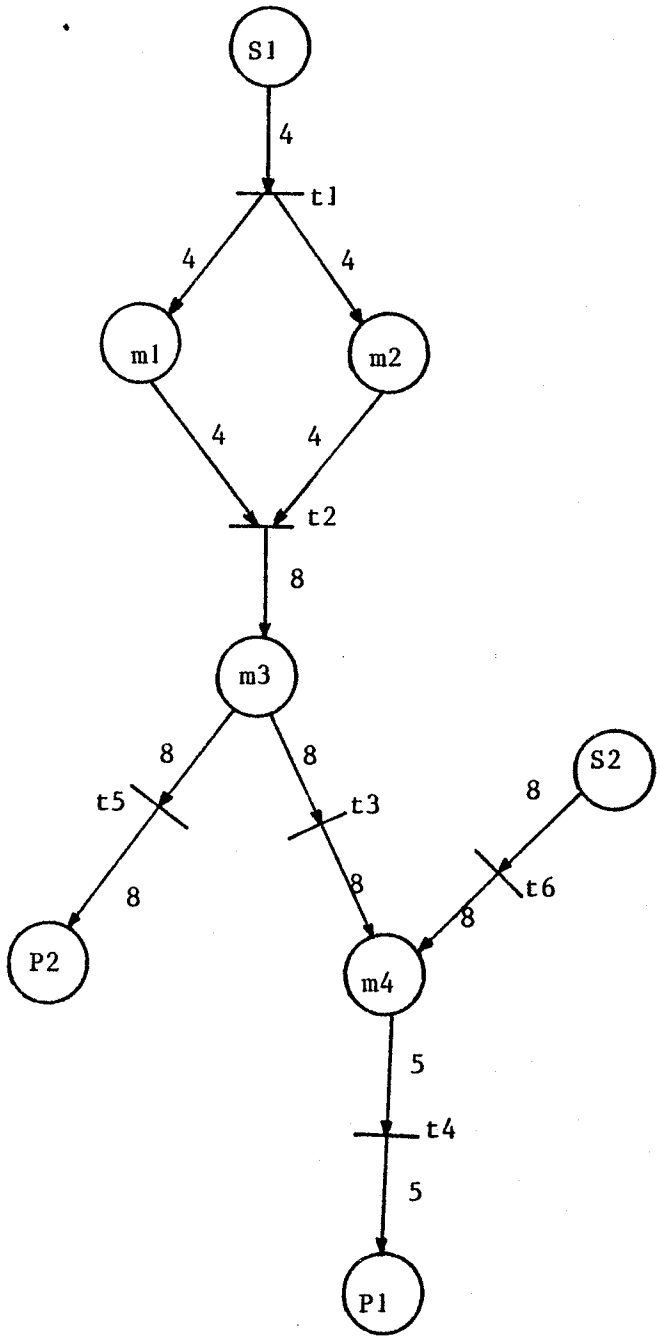
$E2$ = (t1,t2,t3,t4)

Dans $E1$, S1 est la seule source et il n'y a pas de variable interne; la suppression de l'arc (m2,t3) déconnecte m2 du puits P1.
Puisque cet arc a une capacité 4 et que c'est une coupe minimale, on peut écrire que $\theta_2^{E_1} = 4$

Dans $E2$, on obtient le même résultat $\theta_2^{E_2} = 4$.

III - 4. INTERPRETATION

Soit le système modélisé comme suit :



Les écoulements sont :

$E1$ = (S1,m1,m2,m3,P2;t1,t2,t5)

$E2$ = (S1,m1,m2,m3,m4,P1;t1,t2,t3,t4)

$E3$ = (S2,m4,P1;t6,t4)

Trois ensembles d'écoulements permettent de couvrir de façon équivalente, l'ensemble des modules :

D1 = {$E1$, $E2$}

D2 = {$E2$, $E3$}

D3 = {$E1$, $E3$}

Chacun de ces ensembles Di permet de détecter une panne simple et de la localiser soit dans m4, soit dans le bloc (m1,m2,m3).
L'apport d'une information structurelle permet d'évaluer la commandabilité de chacun des modules dans chacun des écoulements; on obtient ainsi :

. pour les modules m1 et m2

$$C^{E_1}_{m1} = C^{E_1}_{m2} = C^{E_2}_{m1} = C^{E_2}_{m2} = 4$$

$$\theta^{E_1}_{m1} = \theta^{E_1}_{m2} = \theta^{E_2}_{m1} = \theta^{E_2}_{m2} = 4$$

Pour ces modules on peut donc choisir indifféremment les écoulements $E1$ ou $E2$.

. pour le module m3 :

$$C^{E_1}_{m3} = C^{E_2}_{m3} = 4$$

$\theta^{E_1}_{m3}$ = 8 (suppression de l'arc t5 - P2)

$\theta^{E_2}_{m3}$ = 5 (suppression de l'arc t4 - P1)

Il s'ensuit que le test de m3 est meilleur par l'écoulement $E1$ que par l'écoulement $E2$ (observabilité complète par $E1$).

. pour le module m4 :

$C^{E_2}_{m4} = 4$ (suppression de l'arc S1 - t1)

$C^{E_3}_{m4} = 8$ (suppression de l'arc S2 - t6)

$\theta^{E_2}_{m4} = \theta^{E_3}_{m4} = 5$

Il s'ensuit que le test de m4 est meilleur par l'écoulement $E3$ que par l'écoulement $E2$ (commandabilité complète par $E3$).

Ceci implique qu'un test complet de m3 et m4 par l'écoulement $E2$ est soit impossible, soit difficile et que l'ensemble D3 = ($E1$, $E3$) constitue la meilleure politique de test.

D'une manière générale on peut dire :

. si un module m_i appartient à deux écoulements E_j et E_k et $\theta_i^{E_j} < \theta_i^{E_k}$ alors le test de m_i à travers E_k est plus complet qu'à travers E_j

. de même si un module m_i appartient à deux écoulements E'_j et E'_k et $C_i^{E'_j} < C_i^{E'_k}$ alors le test de m_i à travers E'_k est plus complet qu'à travers E'_j.

Un problème se pose lorsqu'on a la situation suivante :

$C^{E_j}_i < C^{E_k}_i$

$\theta_i^{E_j} > \theta_i^{E_k}$

Un examen plus approfondi du système et une information supplémentaire sur son fonctionnement sont alors nécessaires.

III - 5. ETUDE DE LA TESTABILITE

On a vu que les concepts de commandabilité et d'observabilité sont à la fois des outils de comparaison d'écoulements par rapport au critère de test complet d'un module et des outils permettant d'approcher la difficulté de génération du test d'un module dans un écoulement donné.

De la même manière que Stephenson et Grason [STE,76] on pourrait construire un coefficient empirique de testabilité, à partir des mesures de commandabilité et observabilité. Mais ces coefficients sont plus représentatifs de la difficulté d'acheminement et de propagation des données de test que de la difficulté elle-même du test.

En effet, la mesure de commandabilité exprime la quantité d'information que l'on peut acheminer vers l'entrée d'un module donné; mais elle n'indique pas si cette information vient plutôt des sources de l'écoulement (module directement commandable) ou si elle provient en majeure partie des variables internes (l'information est alors plus difficilement commandable).

Le même problème se pose au niveau de la mesure d'observabilité : cette mesure indique si l'information de sortie d'un module se conserve dans l'écoulement ou disparaît; mais elle n'indique pas la proportion de cette information qui se dirige vers les puits de l'écoulement (information directement observable) et celle qui va affecter les variables internes du système (difficilement observable).
Ces ambiguités sont d'ailleurs la cause des difficultés rencontrées dans la méthode de Stephenson et Grason, dans le traitement des systèmes séquentiels.

D'autre part, les paramètres de commandabilité et observabilité éliminent certaines solutions (ensembles d'écoulements); s'il reste plusieurs solutions équivalentes ou non comparables, il est alors nécessaire d'introduire une nouvelle mesure: la testabilité, pour départager les solutions possibles.

Il faut alors faire un compromis entre la charge de test totale, c'est-à-dire le nombre de vecteurs de test nécessaires pour vérifier le système et le travail de test, c'est-à-dire le travail d'élaboration de ces vecteurs.

Exemple : Soit un ensemble de modules qui peut être couvert soit par un écoulement $E0$, soit par deux écoulements $E1$ et $E2$. Il apparaît évident que le nombre de vecteurs de test associé à $E0$ -soit n0 - sera inférieur au nombre de vecteurs de test associés à $E1$ et $E2$ -soient n1 et n2.

$$n0 < n1 + n2$$

La charge de test de la solution $E0$ est inférieure à la charge de test de la solution ($E1$, $E2$).
Par contre, le travail d'élaboration des vecteurs de test de $E0$ sera certainement plus complexe que celui de la solution ($E1$, $E2$) : perte d'information à travers les modules, propagation et consistence,...
Il s'ensuit que le travail de test de la solution $E0$ est plus grand que le travail de test de la solution ($E1$, $E2$).

Il s'agit donc, en fait, de faire un compromis entre

- le coût de la maintenance C_M : ce coût est fonction de la charge de test

$$C_M(E0) < C_M(E1, E2)$$

- le coût d'élaboration du programme de test C_E : ce coût est fonction du travail de test

$$C_E(E0) > C_E(E1, E2)$$

Pour résoudre ce compromis une mesure de testabilité a été développée dans [MIL,78] comme suit :

. On élabore un paramètre propre à chaque module qui constitue la charge de test du module, c'est-à-dire le nombre de vecteurs de test qui lui sont nécessaires.

. A partir des mesures de commandabilité et d'observabilité, on définit des paramètres :

  - de rendement
  - de charge de travail

- le coefficient de rendement amont est une fonction de la probabilité qu'une valeur s affichée en S se transforme en une valeur x donnée (vecteur de test) en entrée du module considéré,
- le coefficient de rendement aval est une fonction de la probabilité qu'un vecteur de test x du module m considéré donne une valeur différente aux puits du système selon que m est ou non défectueux,
- la charge de travail représente le coût nécessaire pour vérifier qu'un vecteur de test x a une image aux sources de l'écoulement et que sa transformation aux puits P donne une valeur différente suivant l'état de santé de m.

IV - CONCLUSION

Le travail du concepteur d'un système logique facilement testable, consistera donc à étudier le modèle de son système et à analyser les points suivants :

- les possibilités des diverses stratégies de test,
- l'évaluation des paramètres de commandabilité, d'observabilité et de testabilité,
- l'introduction de points de test ou la définition d'écoulements spécifiques pour le test.

La procédure générale peut se représenter par l'organigramme de la page suivante.

Cette approche à deux niveaux a été proposée comme aide au concepteur d'un système facilement testable et comme outil de recherche de la politique de test optimale.

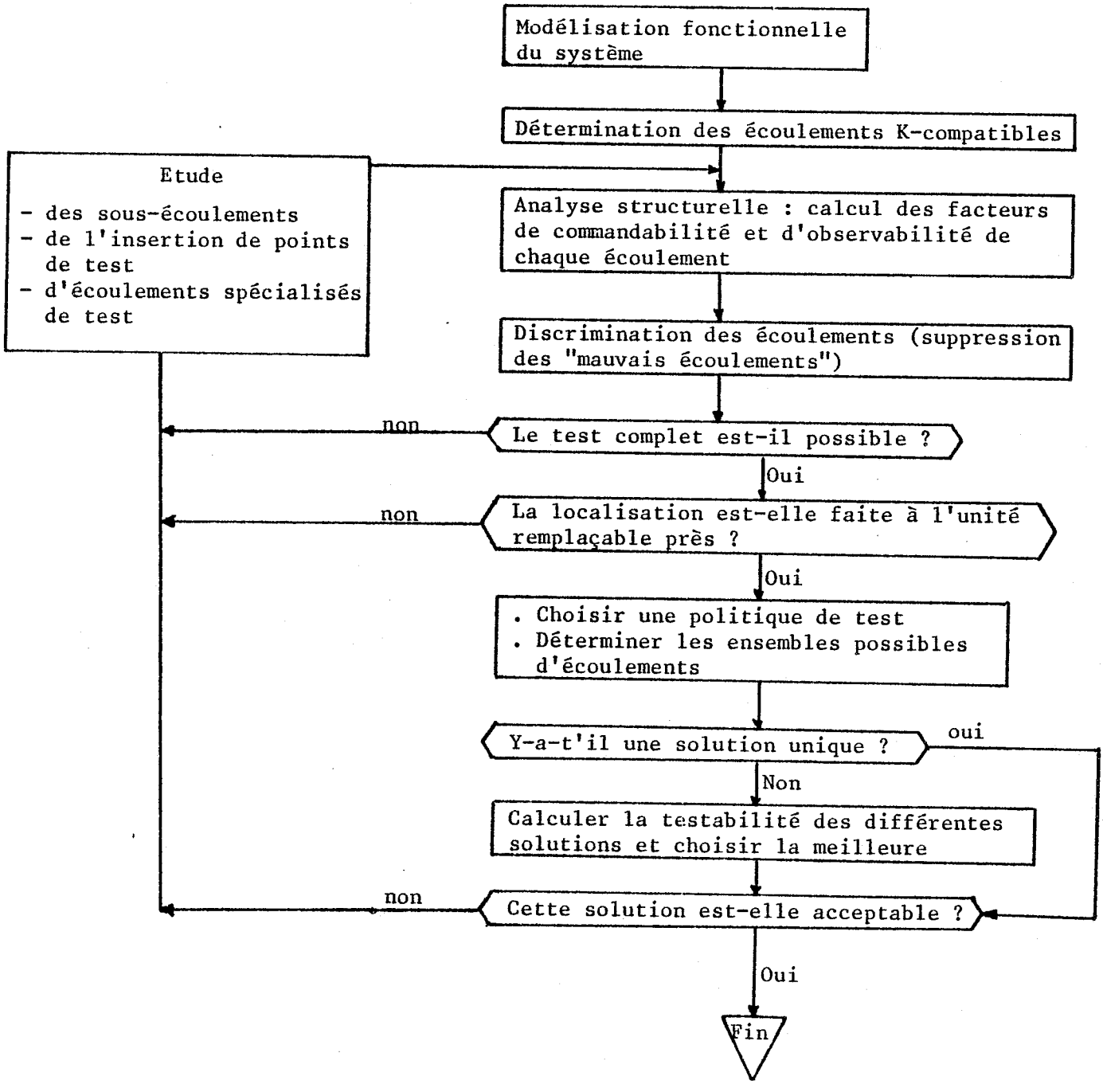
. Le premier niveau est caractérisé par une modélisation fonctionnelle; le modèle n'a besoin que d'une information topologique sur les lignes de données et les activations du contrôle élémentaires. L'analyse permet l'énumération de "chemins de test" qui peuvent être utilisés pour la mise en oeuvre du test, la conception d'une stratégie de test et donne une estimation des performances possibles de cette stratégie : modules non testables complètement, pannes non localisables.
Lors de la conception du système des procédures d'insertion de points de test, des algorithmes de test spécifiques, de partition du système en unités remplaçables sont alors utilisées.

. Le second niveau (niveau structurel) nécessite une information supplémentaire sur la structure du système. Il permet :

- de définir la politique de test optimale parmi celles fournies par le premier niveau,
- de proposer des points d'accès supplémentaires pour améliorer la diagnosabilité du système (réduction du coût de diagnostic, meilleure résolution)

à partir des critères de commandabilité, observabilité et testabilité; il s'ensuit que :

- pour une stratégie donnée, un ensemble d'écoulements peut être choisi qui rend maximum la testabilité du système,
- dans le cas de stratégies équivalentes, leur mesure de testabilité permet de choisir la plus économique et/ou celle donnant le meilleur degré de résolution,
- dans le cas où la testabilité est trop faible, ces critères induisent une insertion de points de test pour augmenter la testabilité.



CHAPITRE II

LE TEST DES UNITÉS DE CONTRÔLE

INTRODUCTION

Lors de l'élaboration de [micro]programmes de test de systèmes complexes, ce sont les unités de contrôle qui présentent le plus de difficultés [NOR,77 - ROB,76a] dûes :

- à leur rôle de gestion de l'ensemble des organes constituant le système
- au fait que leur bon fonctionnement ne peut être vérifié qu'à travers leurs actions sur les autres unités du système.

Généralement deux approches sont utilisées :

- soit l'utilisation de techniques d'auto-test permettant d'éviter les problèmes d'écriture d'un programme de test, en particulier lorsque l'unité de contrôle fait partie d'un système à haute sécurité [COO,73]
- soit l'élaboration de méthodes de test pour une unité de contrôle considérée hors de son contexte ; ces méthodes s'appliquent surtout pour des unités de contrôle de petite taille et câblées ; ces méthodes sont de deux types :
  - . extension de méthodes analytiques : l'unité de contrôle est testée comme un circuit séquentiel (D-algorithme) [ROT,68]
  - . méthodes de test fondées sur une représentation de l'unité de contrôle par un automate : méthode d'identification d'automates (utilisation de séquences de distinction) [KOH,70].

L' approche présentée ici est sensiblement différente. L'objectif est le test des unités de contrôle - câblées ou microprogrammées - dans les conditions suivantes :

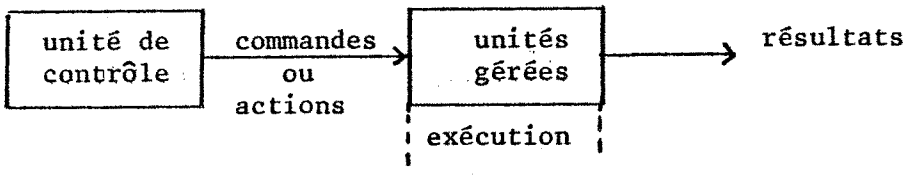
- . le système dans lequel l'unité de contrôle est intégrée, est opérationnel
- . le test se fait dans les conditions de fonctionnement dynamique normal de l'unité : séquencement temporel normal

. on ne considère pas de moyens d'accès spéciaux pour le test : on ne pourra faire d'observation directe et exhaustive des états du contrôle (lecture des microinstructions dans un contrôle microprogrammé, lecture des sorties de bascules dans un contrôle câblé).

Si de tels moyens existent, ils peuvent être inutilisables par suite de contraintes de temps et/ou de stockage :

- vérification rapide d'un processeur oisif dans une structure multiprocesseurs [BEL,76]
- télémaintenance de centres de commutation téléphonique [BEL,78] dans lesquels la distance entre l'organe testé et le centre de maintenance exige une minimisation de l'information de test.

En conséquence, la méthode de test proposée vérifie le bon fonctionnement de l'unité de contrôle à travers les unités commandées, supposées correctes : le principe est donc de vérifier une action par le résultat de son exécution ; un tel test est dit dynamique.



La première partie de cette étude expose le modèle utilisé : l'unité de contrôle est représentée par les algorithmes qu'elle réalise et est associée aux unités fonctionnelles qu'elle commande.

Dans la deuxième partie, les moyens d'identification des états de l'unité de contrôle à travers la partie opérative sont étudiés.

La troisième partie expose un algorithme de diagnostic qui, par un choix adéquat d'opérandes et d'algorithmes, permet

- une observabilité suffisante à travers la partie opérative
- une bonne efficacité du test pour les hypothèses de pannes considérées.

La dernière partie montre l'application de cette méthode sur une partie contrôle câblée d'un processeur de moyenne importance.

I - DEFINITIONS

I -1. MODELE GENERAL

L'ensemble {unité de contrôle, unités commandées} réalise des algorithmes comportant un état final.

Pour chaque algorithme on peut considérer :

- la fonction calculée par l'algorithme
- la méthode de calcul de cette fonction c'est-à-dire la définition de la séquence d'actions à exécuter pour réaliser la fonction.

Afin d'utiliser les concepts d'automates pour la description de processus algorithmiques, il est nécessaire d'interpréter les signaux d'entrée et de sortie de l'automate, respectivement comme :

- les signaux concernant l'information à traiter
- les actions élémentaires définies par l'algorithme.

Un automate ayant un état final est appelé processeur discret d'information si une telle interprétation a été donnée.

Le système peut être décomposé en deux parties [GLU,69] selon le schéma suivant :

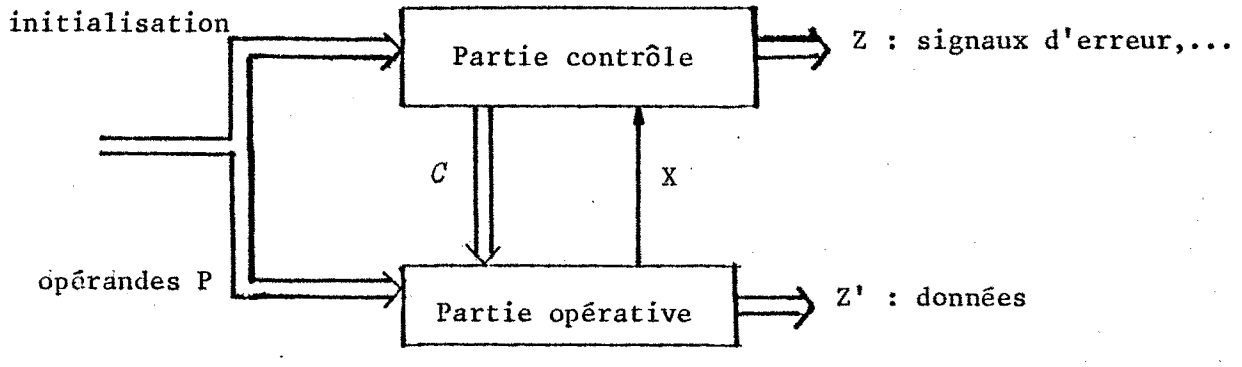


Figure 1. Modèle général

La partie contrôle envoie des signaux vers un ensemble de registres, compteurs, unités arithmétiques et logiques, mémoires ... . Ces unités ren-

voient des réponses à l'unité de contrôle, lui permettant de s'orienter sur une séquence de fonctions particulière.

α) *L'unité de contrôle*

L'unité de contrôle (K) est donnée par un automate de Moore :

$$K = \{X, O, Q, \delta, \lambda\}$$

où

- X est un ensemble fini d'entrées provenant de la partie opérative : compte-rendu, code condition ...
- $O = \{Z, \mathcal{C}\}$ est un ensemble fini non vide de sorties
  Z est le sous-ensemble des sorties qui sont observables : signal de fin d'algorithme, alarme,...
  $\mathcal{C}$ est l'ensemble des commandes élémentaires $\{c_1, \ldots c_n\}$ envoyées vers la partie opérative.
- Q est l'ensemble des états du contrôle : microinstructions dans un contrôle microprogrammé, bascules dans un contrôle câblé.
- $\delta : X \times Q \rightarrow Q$ la fonction de transition qui, à l'état présent et aux entrées fait correspondre un état suivant .
- $\lambda : Q \rightarrow O$ la fonction de sortie qui fait correspondre une sortie à un état donné.

A tout état Qi du contrôle est associé un ensemble de commandes $\mathcal{C}_i$: $\mathcal{C}_i$ est le sous-ensemble de commandes: $\mathcal{C}_i \subset \mathcal{C}$, générées par Q_i, qui activent des opérations dans la partie opérative.

On définit un temps élémentaire θ_i comme le temps de validation d'un état Q_i soit le temps pendant lequel les commandes activées par Q_i sont validées.

Remarques :

- dans le cas de systèmes synchrones, θ_i est le même pour tous les états Q_i : $\forall_i$, $\theta_i = \theta$ (cycle d'horloge)
- on considère qu'il n'y a pas de parallélisme dans l'unité de

contrôle : un seul état est valide pendant un temps élémentaire .

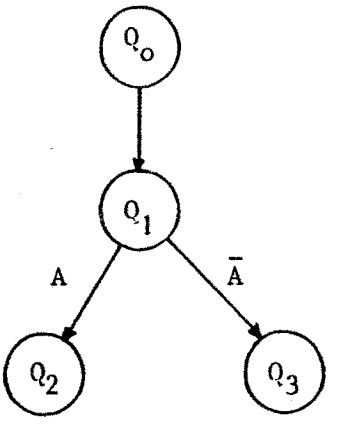
β) *Représentation algorithmique du contrôle*

L'ensemble {unité de contrôle + partie opérative} réalise des algorithmes ayant un état initial et un état final.

Pour chaque algorithme, le fonctionnement de la partie contrôle est représenté par un organigramme dont :

- les noeuds sont les différents états du contrôle mis en jeu par l'algorithme considéré
- les arcs représentent les différents séquencements possibles entre ces états ; les arcs sont indicés par les conditions de séquencement (prédicats).

Exemple



- Le contrôle est dans l'état initial Q_o au temps to.
- la transition $Q_o \rightarrow Q_1$ est réalisée sans condition ; le contrôle est dans l'état Q_1 au temps t_1 $(to+\theta)$
- si le prédicat A est vrai, la transition $Q_1 \rightarrow Q_2$ est réalisée : le contrôle est dans l'état Q_2 au temps t_2 $(t_o+2\theta)$; sinon, il passe dans l'état Q_3 au temps t_2.

γ) *La partie opérative*

La partie opérative est un ensemble d'unités réalisant les fonctions définies par l'algorithme. Elle reçoit en entrée :

- un ensemble d'opérandes P sur lesquels elle effectue un traitement
- un ensemble de commandes générées par la partie contrôle.

Elle génère comme sorties

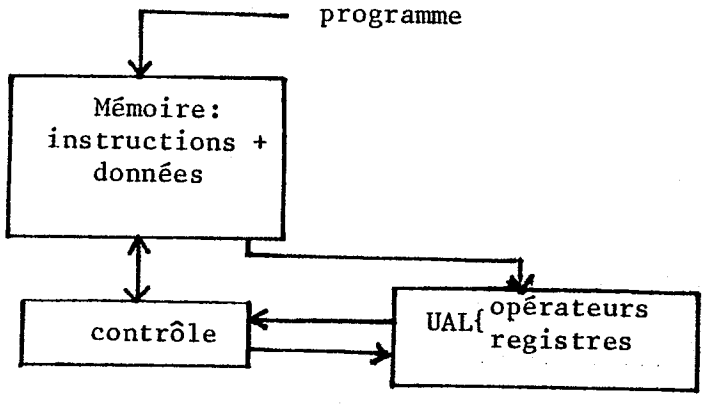
- des signaux de réponse vers la partie contrôle : compte-rendu, code condition (par exemple, à la suite d'un test)
- un ensemble de résultats : Z' , qui représente l'information calculée qui est envoyée au monde extérieur.

I -2. IMPLEMENTATION MATERIELLE

Deux types de réalisation sont principalement utilisés :

α) *Réalisation câblée*

Le contrôle câblé est un automate généralement synchrone du type automate de Moore ; il analyse les différentes instructions stockées en mémoire.



Tous les codages possibles d'un ensemble de bascules sont associés bijectivement aux états Q_i ; les fonctions de transition et de sortie sont implémentées en logique discrète ou PLA.

Un cas intéressant est celui du codage 1 parmi n : une bascule est associée à chaque état ; la réalisation des fonctions de transition et de sortie est alors directement déduite des schémas algorithmiques [ROB,75b].

Il existe deux types de problèmes pour ce type d'unités de contrôle :

- si le nombre d'instructions est très important, la quantité de matériel nécessaire pour implémenter l'unité de contrôle devient exhorbitante
- le jeu d'instructions est figé : l'unité de contrôle ne peut être modifiée sans changer le matériel et la conception.

β) *Réalisation microprogrammée*

Pour éviter ces problèmes, Wilkes proposa en 1951 une unité de contrôle dite microprogrammée. L'unité de contrôle a son propre jeu d'instructions : les microinstructions. Chaque instruction est interprétée par un microprogramme qui est une séquence de microinstructions. Ces microprogrammes sont stockés dans une mémoire morte ROM. L'exécution de ces microprogrammes est contrôlée par une machine séquentielle simple : le microséquenceur.

Pour éviter les problèmes de diaphonie on utilise un registre (MIR) qui mémorise les sorties de la mémoire ROM.

On a le schéma général suivant [HIL,73 - HUS,70].

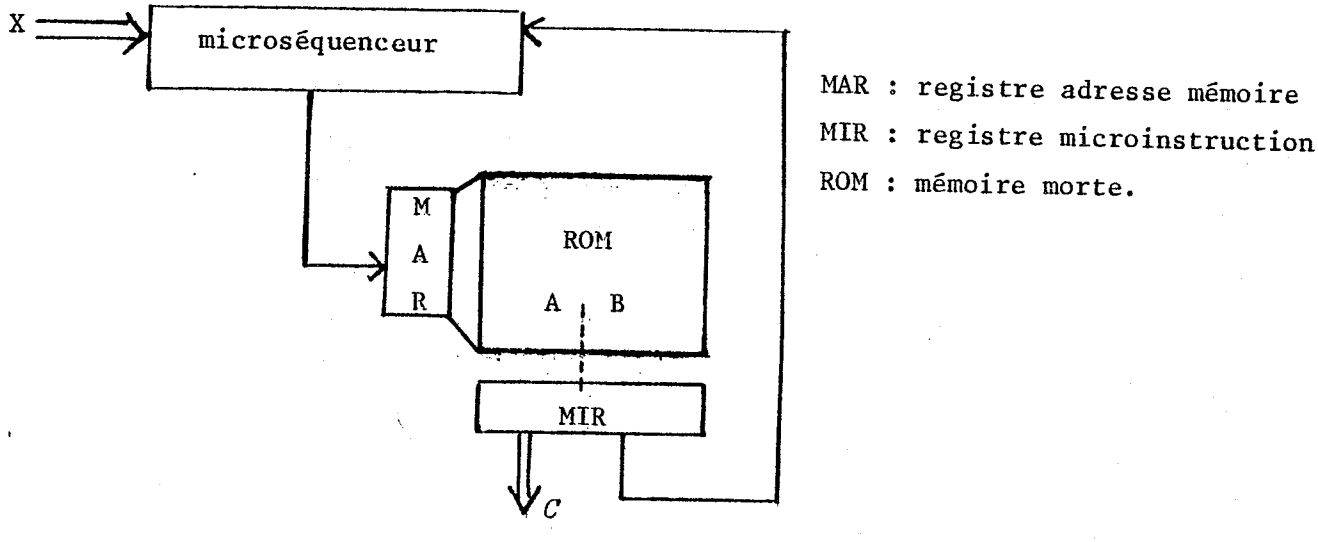


Figure 2 : Réalisation microprogrammée

Les états du contrôle sont alors les différentes valeurs du registre MAR ; comme il y a bijection entre les valeurs de MAR et le contenu de la ROM, on considère dans la suite qu'un état, à un moment donné, est entièrement caractérisé par le contenu du registre microinstruction MIR.

II - IDENTIFICATION D'UN ETAT ET DISTINGUABILITE [ROB,78]

II - 1. CONTEXTE GENERAL - DISTINGUABILITE

Un état Q de la partie contrôle est caractérisé par l'ensemble des commandes $\lambda(Q)$ et par la fonction de transition $\delta(Q,X)$: fonction de sélection de l'adresse suivante.

Deux états non-équivalents du contrôle Q_i et Q_j peuvent être distingués

- soit par leurs commandes si $\lambda(Q_i) \neq \lambda(Q_j)$ c'est-à-dire $C_i \neq C_j$
- soit par leur fonction de transition :

  si $\lambda(Q_i) = \lambda(Q_j)$ il existe une séquence d'entrée I telle que $\delta(Q_i, I) \neq \delta(Q_j, I)$.

Dans ce cas, il est intéressant de chercher une séquence d'entrée minimale I_m telle que

$$\lambda[\delta(Q_i, I_m)] \neq \lambda[\delta(Q_j, I_m)]$$

c'est-à-dire la plus courte séquence conduisant à deux états directement distinguables par leurs commandes.

Remarque - pour un contrôle câblé on a généralement $|I_m| = 1$. Il s'ensuit que l'identification d'un état Q_i se fera par l'ensemble des sorties C_i c'est-à-dire à travers la partie opérative.
L'ensemble des sorties directes Z est constitué de signaux de fin d'algorithme (débordement,...) et est supposé ne pas être significatif pour identifier un état autre que l'état final de l'algorithme.

II - 2. OBSERVABILITE A TRAVERS LA PARTIE OPERATIVE

La partie opérative peut être considérée comme un ensemble d'unités fonctionnelles $\{U_k\}$.

On notera c^k l'ensemble des commandes associées à l'unité U_k : c^k est l'ensemble des commandes permettant d'assurer les fonctions de l'unité U_k et des commandes validant les entrées sur cette unité.

Remarque - il peut y avoir parallélisme d'exécution de micro-opérations dans la partie opérative.

On obtient le schéma général suivant :

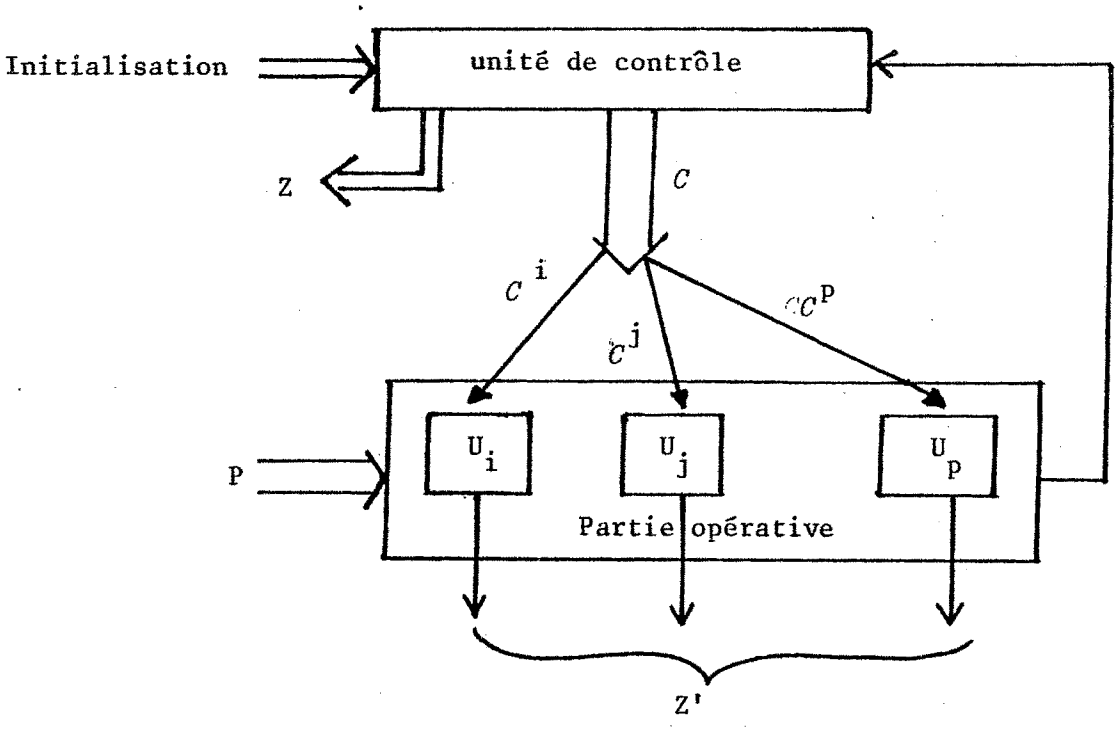


Figure 3 : Modèle général

Exemple 1 - [ROB,75a]

Soit une partie opérative comportant trois unités fonctionnelles (figure 4) :

- un registre H
- un registre N
- un additionneur ADD

dont les sorties sont observables sur deux bus ALPHA et BETA.

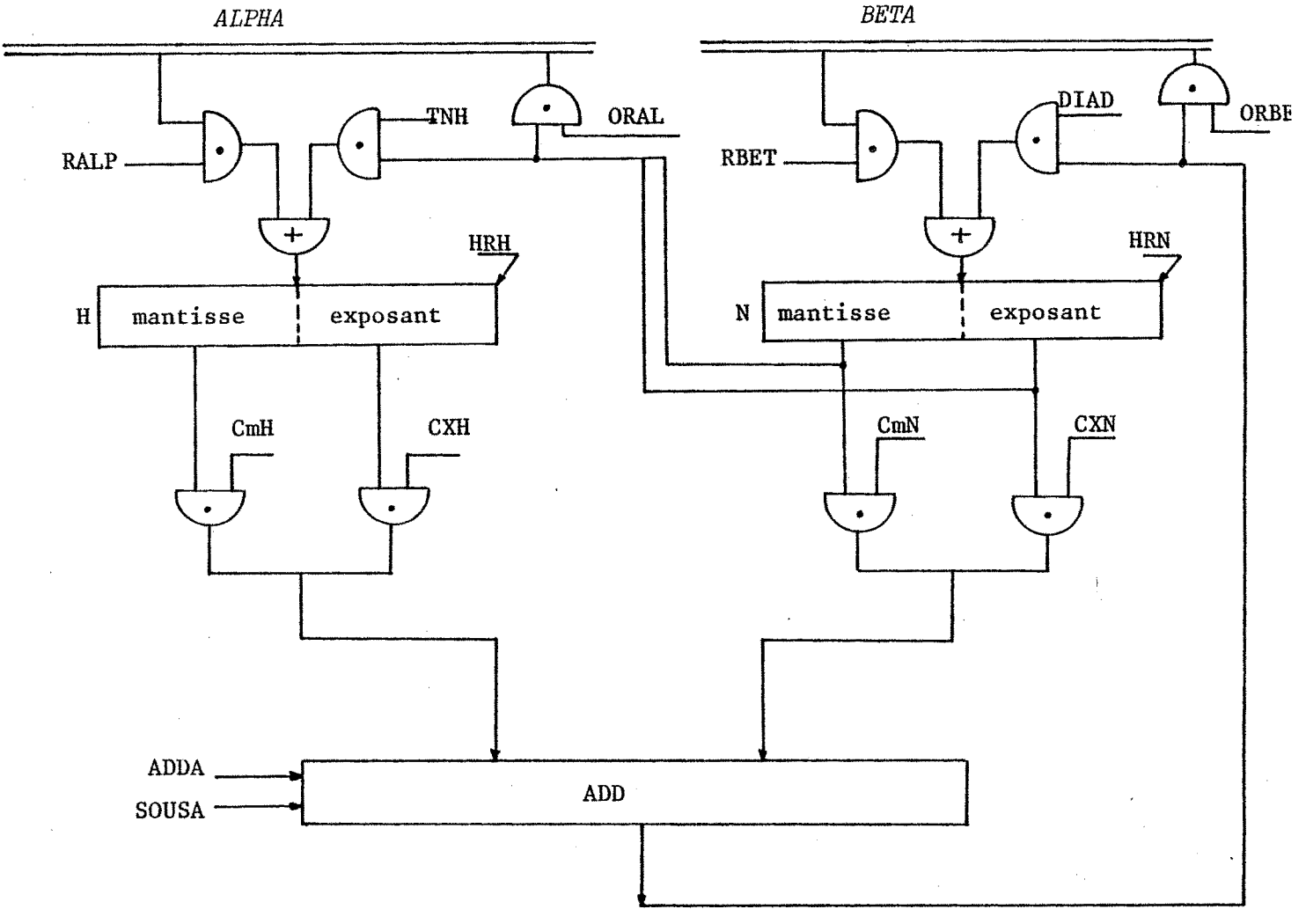


Figure .4. Exemple 1

C^H = {HRH, RALP, TNH}

C^N = {HRN, DIAD, RBET}

C^{ADD} = {ADDA, SOUSA, CmH, CXH, CmN, CXN}

fonctions additionneur — validation des entrées

On définit l'ensemble C_i^k comme la restriction de C_i (ensemble des commandes activées par l'état Q_i) à l'unité U_k c'est-à-dire l'ensemble des commandes activées par C_i et envoyées à U_k

$$C_i^k = C_i \cap C^k$$

Remarque - C_i^k peut être vide.

Soit un état Q_1 qui charge les registres H et N avec deux valeurs en provenance respectivement des bus ALPHA et BETA.

Soit un état Q_2 qui additionne les mantisses des deux registres et charge le résultat dans N.

On obtient :

$C_1^H = \{RALP , HRH\}$ $\qquad$ $C_2^H = \phi$

$C_1^N = \{RBET , HRN\}$ $\qquad$ $C_2^N = \{DIAD , HRN\}$

$C_1^{ADD} = \phi$ $\qquad$ $C_2^{ADD} = \{CmH , CmN , ADDA\}$

α) *Moyens d'observation*

Les moyens d'observation sont définis comme les voies de communication entre la partie opérative et l'extérieur (généralement des bus), sur lesquelles on peut lire un résultat.

L'unité observée est alors l'unité source sur cette voie de communication. On peut alors distinguer deux types d'unités de contrôle :

- les unités de contrôle locales, généralement câblées, qui sont gérées par un contrôle central (généralement microprogrammé) ; dans le cas du test d'un contrôle local, on peut choisir l'unité que l'on désire observer en spécifiant dans la microinstruction du contrôle central le champ source de la voie de communication.
- les unités de contrôle centrales: dans ce cas, la sortie des unités de la partie opérative sur les voies de communications

est imposée.On peut alors introduire des points de test pour observer une unité donnée (matériel de test).

β) *Observabilité des unités fonctionnelles*

L'observation d'un état se fait à travers les unités fonctionnelles qu'il active et dépend donc de l'observabilité de chaque unité fonctionnelle.

L'observabilité d'une unité est définie comme la facilité d'observer les sorties de cette unité à l'extérieur du système (par les voies de communication).

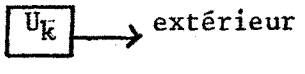
On définit trois types d'observabilité :

- l'observabilité spatiale
- l'observabilité temporelle
- l'observabilité parallèle.

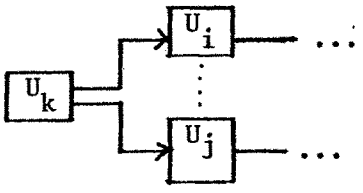
. *Observabilité spatiale*

Une unité peut avoir une observabilité

- soit directe, lorsque ses sorties peuvent être observées directement à l'extérieur



- soit indirecte lorsque ses sorties sont uniquement l'entrée d'autres unités



Une fonction I (U_k) est associée à toute unité fonctionnelle U_k : si la sortie de U_k est directement observable, on note $I(U_k) = \phi$; si l'unité U_k est observable à travers les unités U_i ou ...ou U_j on note $I(U_k) = \{U_i,..,U_j\}$.

Exemple 1- L'additionneur ADD et le registre N sont directement observables respectivement sur les bus BETA et ALPHA (commandes ORBE et ORAL).

$I(N) = I(ADD) = \phi$

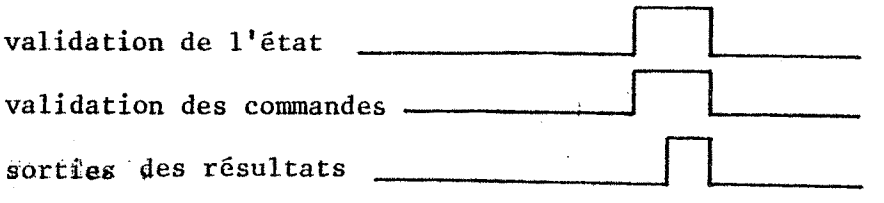
Le registre H ne peut être observé qu'à travers l'additionneur ADD

$I(H) = \{ADD\}$

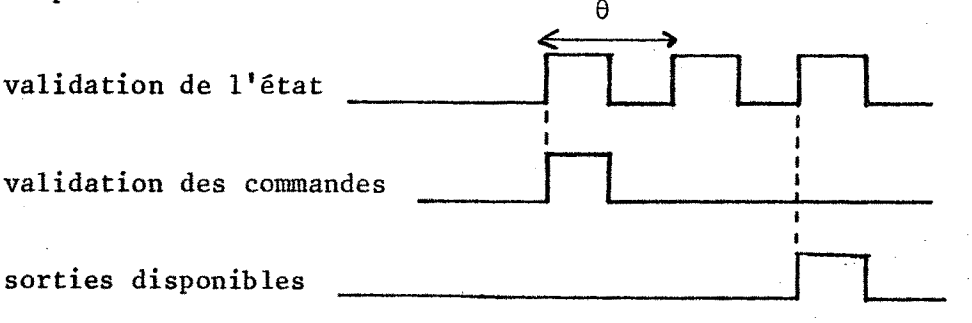
. *Observabilité temporelle*

L'observabilité d'une unité peut être :

- soit immédiate lorsque ses sorties sont disponibles pendant la période élémentaire θ de validation des commandes (même niveau d'horloge).



- soit différée lorsque ses sorties sont disponibles après un temps $k\theta$; c'est le cas, par exemple, pour une unité séquentielle dont le résultat n'est validé qu'au bout d'un certain temps.



Un délai $\Delta(U_k)$ est associé à toute unité fonctionnelle U_k ; si l'observabilité est immédiate, on note $\Delta(U_k) = 0$; si l'observabilité de l'unité U_k se fait après un temps $k\theta$ on note $\Delta(U_k) = k$.

Exemple 1 - l'additionneur ADD a une observabilité immédiate (comme tout circuit combinatoire)

$\Delta(ADD) = 0$

Les registres H et N ont une observabilité différée : les résultats ne sont disponibles qu'au top d'horloge suivant

$$\Delta(N) = \Delta(H) = 1$$

. *Observabilité parallèle*

Il y a observabilité parallèle dès qu'au moins deux unités fonctionnelles peuvent être observées simultanément à un moment donné.

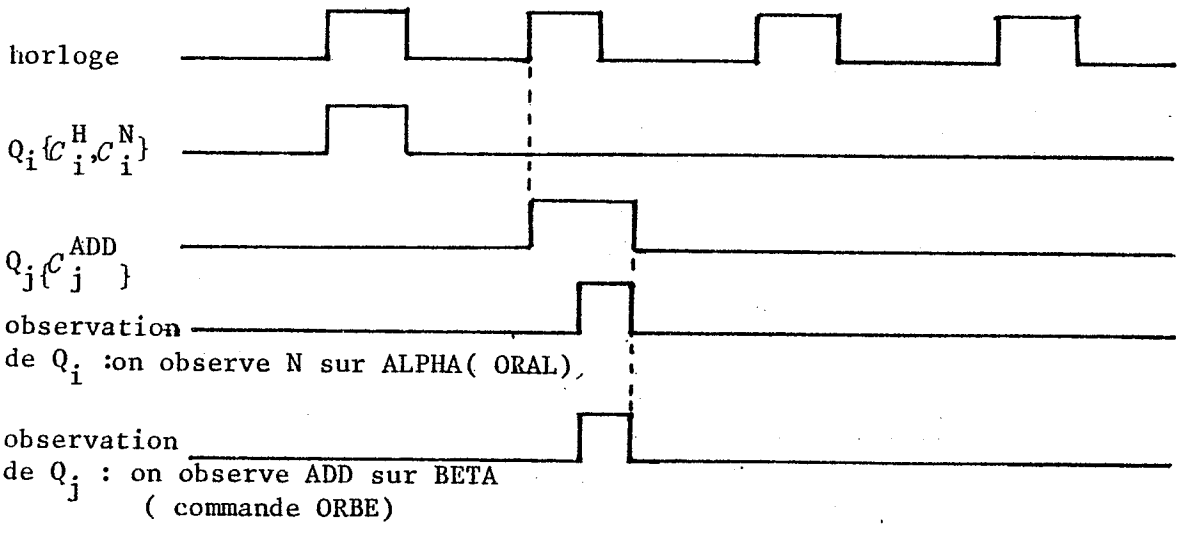
Le degré de parallélisme est donc fonction des moyens d'observation.

On peut considérer plusieurs situations :

- soit un état Q_1 active pendant la période θ deux unités U_1 et U_2 qui sont simultanément observables pendant cette période
- soit deux états Q_i et Q_j sont observables à travers deux unités fonctionnelles lorsqu'il y a observabilité indirecte ou différée de l'une des unités.

Exemple 1 - Soit l'état Q_i qui charge des données dans les registres H et N :
C_i^H = {RALP , HRH} , C_i^N = {RBET , HRN}

Soit l'état Q_j qui fait l'addition des deux registres et sort le contenu de l'additionneur sur le bus : C_j^{ADD} = {ADD , ORBE , CmH , CXH , CmN , CXN}



γ) *Matrice de représentation*

On définit la matrice de représentation M du contrôle, à n lignes $c_1 \ldots c_n$ qui sont les commandes élémentaires, à m colonnes $Q_1 \ldots Q_m$ qui sont les états du contrôle, telle que $m_{ij} = 1$ si l'état Q_j génère la commande c_i

$m_{ij} = 0$ sinon

La matrice est décomposée en sous-matrices M_k relatives à chaque unité fonctionnelle U_k : la partition se fait en regroupant dans M_k les lignes (commandes) qui activent l'unité U_k.

Exemple 2 -

Soit une unité arithmétique et logique opérant sur des nombres entiers et comportant une gestion automatique de la multiplication. La structure de cette unité est donnée en Figure 5 :

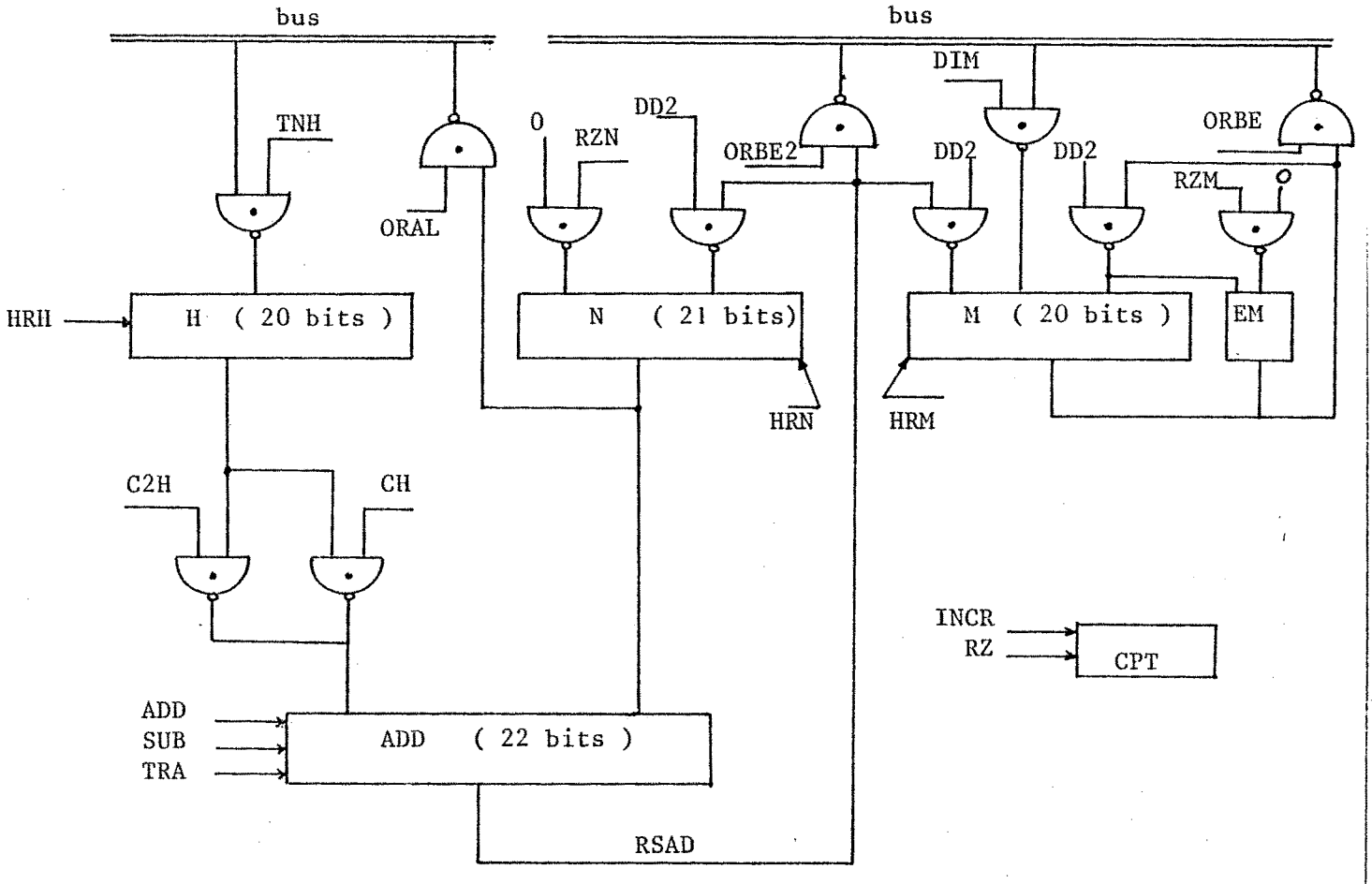


Figure 5 - Exemple 2 : Structure de l'UAL

La multiplication est automatique et s'exécute en une séquence de dix cycles. Son déroulement est le suivant :

a) *Initialisation*

Le registre N et le compteur de cycles CPT sont initialisés à zéro, les opérandes sont chargés dans le registre H (multiplicande) et le registre M (multiplicateur), respectivement à partir des bus α et β .

b) *Cycle de multiplication*

Ce cycle comporte les étapes suivantes :

- le test des trois bits de poids faible du multiplicateur : M(1) , M(0) , EM
- suivant le résultat du test, la sortie du registre H est décalée ou non de deux positions sur la gauche
- suivant la valeur du bit M(1), l'additionneur effectue une addition, une soustraction ou un simple transfert du registre N

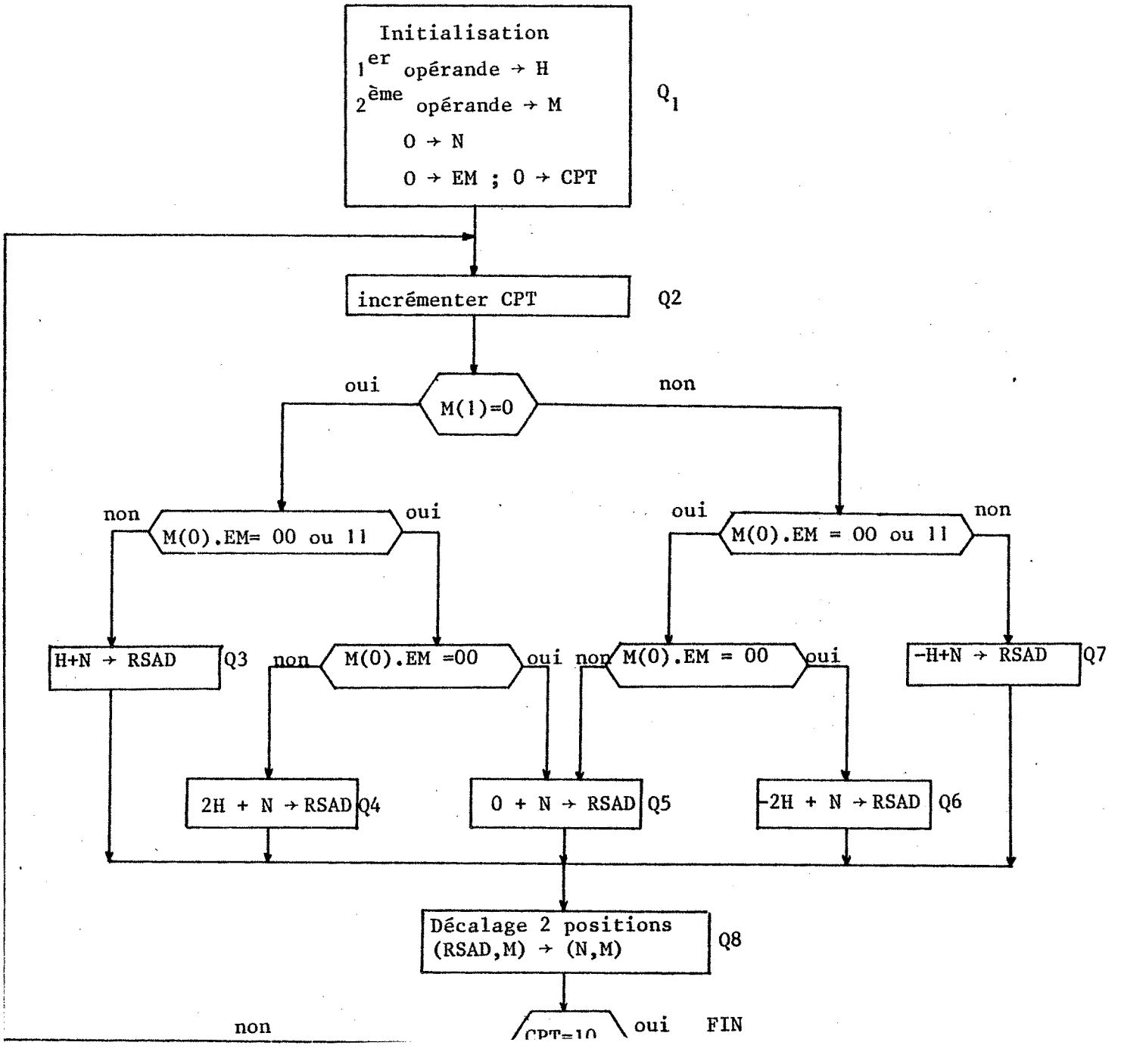
M(1)	M(0)	EM	Opération
0	0	0	0
0	0	1	+H
0	1	0	+H
0	1	1	+2H
1	0	0	-2H
1	0	1	-H
1	1	0	-H
1	1	1	0

- le résultat RSAD obtenu à la sortie de l'additionneur est rangé dans N et M et décalé sur la droite de deux positions : c'est le résultat partiel de la multiplication. Par ce même décalage, le multiplicateur est également décalé : M(1) → EM.

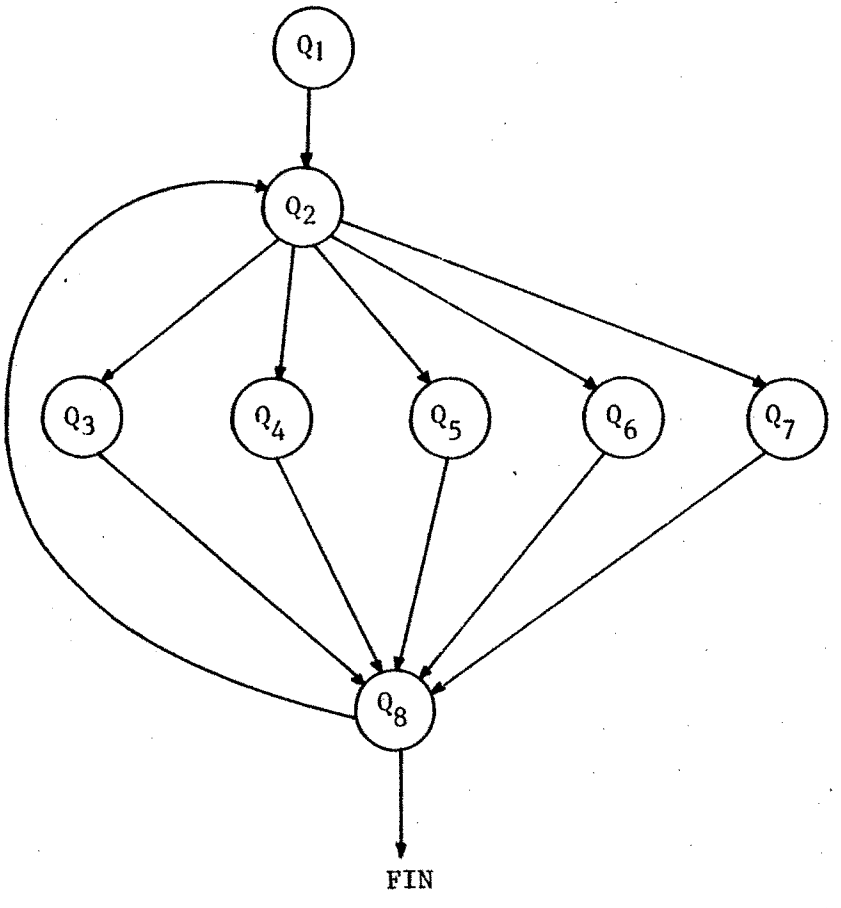
- le compteur de cycles avance d'une position et un nouveau cycle commence ; en fin de multiplication (CPT = 10) les poids forts du résultat de la multiplication sont disponibles dans N et les poids faibles dans M. Ils peuvent être émis sur le bus α (poids forts) et le bus β (poids faibles).

On peut observer simultanément les registres N et M ou le registre N et l'additionneur ADD, respectivement sur les bus α et β , en positionnant les commandes ORAL, ORBE1 ou ORBE2.

L'organigramme de multiplication est le suivant :



L'organigramme de contrôle est le suivant



Les unités fonctionnelles activées sont les registres H, N et M, la bascule EM, l'additionneur ADD et le compteur de cycles CPT.

$$\mathcal{C}^{H} = \{HRH , TNH\}$$

$$\mathcal{C}^{N} = \{HRN , RZN , DD2\}$$

$$\mathcal{C}^{M} = \{HRM , DIM , DD2\}$$

$$\mathcal{C}^{EM} = \{HRM , RZM , DD2\}$$

$$\mathcal{C}^{ADD} = \{ADD , SUB , TRA , C2H , CH\}$$

$$\mathcal{C}^{CPT} = \{INCR , RZ\}$$

La matrice de représentation est la suivante :

Etats / Commandes	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Unités
HRH	1	0	0	0	0	0	0	0	H
TNH	1	0	0	0	0	0	0	0	
HRN	1	0	0	0	0	0	0	1	
RZN	1	0	0	0	0	0	0	0	N
DD2	0	0	0	0	0	0	0	1	
HRM	1	0	0	0	0	0	0	1	
DIM	1	0	0	0	0	0	0	0	M
DD2	0	0	0	0	0	0	0	1	
HRM	1	0	0	0	0	0	0	1	
RZM	1	0	0	0	0	0	0	0	EM
DD2	0	0	0	0	0	0	0	1	
ADD	0	0	1	1	0	0	0	0	
SUB	0	0	0	0	0	1	1	0	
TRA	0	0	0	0	1	0	0	0	ADD
C2H	0	0	0	1	0	1	0	0	
CH	0	0	1	0	0	0	1	0	
INCR	0	1	0	0	0	0	0	0	CPT
RZ	1	0	0	0	0	0	0	0	

II - 3. DISTINGUABILITE A TRAVERS LA PARTIE OPERATIVE

α) *Etats distinguables par observation des unités fonctionnelles*

L'identification complète des états [KOH,70] peut être réalisée par l'observation des sorties Z et C ; comme indiqué au paragraphe II.1, nous nous limitons ici à l'observation des commandes C.

Cependant, l'observation de la valeur de toutes les commandes pour chaque état est difficile sinon impossible. Aussi cherche-t'on un sous-ensemble de commandes qui caractérisent cet état en le distinguant de tout autre état.

- On définit le vecteur de commande C_i^k comme le vecteur logique des commandes $C_j \in \mathcal{C}^k$ pour l'état Q_i.

Pour l'exemple 2 :

$C_1^M = (110)$; $C_8^M = (101)$

$C_2^M = C_3^M = C_4^M = C_5^M = C_6^M = C_7^m = (000)$

- Définition : distinguabilité de deux états :

Soient deux états Q_i et Q_j ; on dit que Q_i et Q_j sont distinguables par l'unité U_k si $C_i^k \neq C_j^k$.

Cette relation est notée $Q_i \; D_{u_k} \; Q_j$

Note : $C_i^k \neq C_j^k \Leftrightarrow \mathcal{C}_i^k \neq \mathcal{C}_j^k$

- Interprétation pour le test :

Si l'unité de contrôle est normalement dans l'état Q_i, l'observation des vecteurs de commande C_i^k ou C_j^k permet de vérifier qu'elle n'est pas anormalement dans l'état Q_j.

- Généralisation

Soient p états $Q_1 \ldots Q_p$; on dit que $Q_1 \; D_{u_k} \; (Q_2 \ldots Q_p)$ si

$\forall j \in [2,p]$, $Q_1 \; D_{u_k} \; Q_j$

soit $\forall j \in [2,p] \quad C_1^k \neq C_j^k$

Remarque : la relation $Q_1 \; D_{u_k} \; (Q_2 \ldots Q_p)$ n'implique pas la relation de distinguabilité entre les couples d'états (Q_j, Q_ℓ) pour j et $\ell \neq 1$.

Un ensemble minimal d'unités fonctionnelles U_k peut être associé à chaque état Q_i de telle sorte que Q_i soit distinguable de tous les autres états ; cet ensemble minimal est obtenu par la méthode classique de couverture.

• Exemple 2 : On obtient

$Q_2 \; D_N \; (Q_1 , Q_8)$

$Q_2 \; D_M \; (Q_1 , Q_8)$

$Q_2 \; D_{ADD} \; (Q_3 , Q_4 , Q_5 , Q_6 , Q_7)$

Par conséquent, pour distinguer Q_1 de tous les autres états, on peut observer simultanément N et ADD.

β) *Observation d'une unité fonctionnelle: propriété de perturbation*

Soit un état Q_i distinguable des états $Q_j \ldots . Q_p$ par son vecteur de commande C_i^k ; il s'agit maintenant d'assurer la distinguabilité de cet état à travers cette unité fonctionnelle U_k : cela revient à assurer que l'opération réalisée par U_k donne des résultats différents suivant qu'on a eu C_i^k ou $\{C_j^k, \ldots C_p^k\}$;

S'il existe un opérande d'entrée P de U_k tel que le résultat de l'opération réalisée dans U_k soit différent dans chacun des trois cas suivants :

a) $C_i^k = 1 \quad$ et $\forall j \neq i \quad C_j^k = 0$

cas de fonctionnement normal : C_i^k normalement actif et C_j^k normalement inactif.

b) $C_i^k = 1$ et $\exists j \neq i$ tel que $C_j^k = 1$

un état Q_j est anormalement actif

c) $C_i^k = 0$ et $\forall j \neq i \quad C_j^k = 0$

l'état Q_i est anormalement inactif

alors on dit que les commandes C_i^k , $C_j^k \ldots C_p^k$ se perturbent.

- Interprétation par rapport au test

Supposons que l'unité de contrôle est dans l'état Q_i

- l'état de bon fonctionnement induit l'opération correcte Opa
- dans le premier cas de fonctionnement incorrect (b) il existe au moins une commande C_j^k , $j \neq i$ qui est anormalement activée alors que C_i^k est normalement active. Ceci induit une opération Opb
- dans le deuxième cas de mauvais fonctionnement (c), la commande C_i^k est anormalement inactive, les commandes C_j^k étant normalement inactives. Ceci induit une opération Opc

S'il existe un opérande P tel que

$$Opa\ (P) \neq Opb\ (P) \quad \text{et} \quad Opa\ (P) \neq Opc\ (P)$$

alors Q_i peut être distingué des états $Q_j \ldots Q_p$ par une seule exécution de l'opération dans U_k et ceci assure le test du vecteur de commande C_i^k.

S'il n'existe pas un tel opérande, il est alors nécessaire de faire exécuter m fois l'opération dans U_k (perturbation partielle).

III - LE TEST DE LA PARTIE CONTROLE

III - 1. DEFAUTS MATERIELS ET ERREURS FONCTIONNELLES

Un défaut peut affecter:

- les éléments de mémorisation
  - registre MAR dans un contrôle microprogrammé
  - bascules d'état dans un contrôle câblé
- le matériel réalisant la fonction de sortie λ :
  - partie A de la mémoire morte (figure 2), logique de décodage dans une réalisation microprogrammée
  - circuit de sortie dans une réalisation câblée

- le matériel réalisant la fonction de transition δ :

  - partie B de la ROM et microséquenceur dans un contrôle micro-programmé
  - circuit implémentant la fonction "état suivant" dans un contrôle câblé.

III - 2. METHODOLOGIE DE TEST

α) *Principe général*

Le test d'une unité de contrôle consiste à assurer

- l'identification complète de tous les états
- la vérification du séquencement.

- L'identification complète d'un état consiste à

  - distinguer cet état de tous les autres états par ses vecteurs de commande
  - vérifier les vecteurs de commande de cet état à travers chaque unité fonctionnelle.

Pour mener à bien cette identification, on utilise la relation de distinguabilité qui permet de définir un ensemble minimal d'unités à observer et la propriété de perturbation qui permet de définir les opérandes d'entrée des unités choisies pour assurer la distinguabilité à la sortie de ces unités.

Cette phase du test doit tenir compte

- des contraintes dûes à l'observabilité différée et/ou indirecte
- des possibilités d'observation parallèle.

- La vérification du séquencement est réalisée en passant par chaque transition pour tous les prédicats élémentaires autorisant cette transition.

Par exemple : si P = P1 . P2 + P3 on vérifie deux fois la transition : une fois pour le prédicat P1 . P2 et une autre fois pour le prédicat P3. [GOO,75].

β) *Outils de test*

. Les états distingués sont marqués dans une <u>matrice de distinguabilité</u> $\mathcal{D}$: $|d_{ij}|$

C'est une matrice carrée à m lignes et m colonnes $Q_1 \ldots Q_m$

$d_{ij} = n$ si $Q_i \; D \; Q_j$ au pas de test n.

<u>Note</u> : Les éléments d_{ii} ne sont pas à remplir (il y a donc $m^2 - m$ éléments à marquer).

. Les commandes testées sont marquées dans une <u>matrice de commande</u> $C : |c_{ij}|$

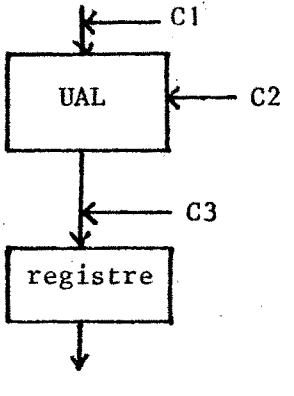
C'est une matrice à p lignes : $C^1 \ldots C^p$ (p unités fonctionnelles) et m colonnes $Q_1 \ldots Q_m$.

Lorsqu'un état Q_i est observé à travers l'unité fonctionnelle U_k, son vecteur de commande C_i^k est testé (test des commandes actives et des commandes inactives).

On note $c_{ik} = n$ (numéro du pas de test).

La matrice de commande peut comporter des éléments qui ne sont pas à marquer :

- soit parce qu'il y a masquage d'une valeur incorrecte



Si C1 ou C2 est activée à tort, la commande C3 bloquant l'entrée du registre, il n'y a pas de conséquence (dans l'hypothèse d'une panne simple).

Il y a un phénomène de masquage : ce mauvais fonctionnement est inobservable

- soit parce que la commande incorrecte transforme la valeur du registre mais ce registre n'est plus lu avant d'être ré-écrit ou n'est plus lu avant la fin de l'algorithme.

. Le séquencement testé est marqué dans une matrice de séquencement $S : |s_{ij}|$

C'est une matrice carrée à m lignes et m colonnes : $Q_1 \ldots Q_m$ et remplie comme suit :

$s_{ij} = 1,n$ si la transition $Q_i \rightarrow Q_j$ est réalisée sans condition au pas de test n

$s_{ij} = (P, n')$ si la transition $Q_i \rightarrow Q_j$ est réalisée si le prédicat P est vrai, au pas de test n'

$s_{ij} = 0$ s'il n'existe pas de transition possible entre Q_i et Q_j.

Lorsqu'un prédicat P est fonction de plusieurs prédicats élémentaires : P = P1 . P2 + P3 on aura

$s_{ij} = (P1.P2,n1) + (P3,n2)$.

. Le test

L'identification complète des états est assurée lorsque les matrices de distinguabilité et de commande sont entièrement marquées.

La vérification du séquencement est achevée lorsque tous les éléments non nuls s_{ij} sont marqués pour tous les prédicats élémentaires.

γ) *Méthode de test*

1) - Choix d'un ensemble minimal de chemins qui couvre toutes les transitions

- Pour chaque état, détermination du sous-ensemble d'unités assurant la distinguabilité de cet état d'avec tous les autres états.

2) . Sur l'ensemble des chemins on construit un tableau de test comme suit :

Pas du test	Etat Q_i activé	Unités observées directement en immédiat	Sorties indisponibles	Unités observées directement en différé
(1)	(2)	(3)	(4)	(5)

Unités activées indirectement observées	Autres unités utilisées	Commandes testées et observées	Etats dont Q_i est distingué :	Séquence-ments vérifiés
(6)	(7)	(8)	(9)	(10)

a) commentaires

. La colonne (1) indique le numéro du pas du test ; ce numéro, lorsqu'il y a détection d'erreur, permet de connaître ce qui est cause de l'erreur en lisant les matrices de distinguabilité , de commande ou de séquencement. Il permet de retrouver dans quel chemin de quel algorithme cette erreur est apparue et dans quel état elle s'est manifestée.

. La colonne (2) représente la séquence des états activés.

. Les colonnes (3) et (5) indiquent un choix des unités que l'on veut observer sur les voies de communication.

La colonne (3) liste les unités qui sont observables immédiatement et la colonne (5) liste les unités qui sont observables en différé, parmi les unités directement observables que l'on a choisi.

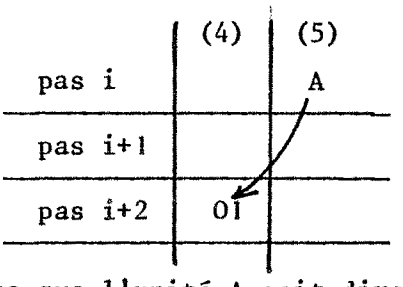
Comment se fait le choix des unités observées en (3) et (5) ?

. C'est d'abord l'ensemble des unités assurant la distinguabilité de l'état activé au pas considéré

. si la distinguabilité est déjà faite, ces unités sont choisies en fonction des vecteurs de commande qui ne sont pas encore testés.

. La colonne (4) est induite par la colonne (5) puisqu'elle traduit le retard d'observation.

Exemple



Supposons que l'unité A soit directement observable sur la voie de communication 01 et que $\Delta(A) = 2$; on note qu'au pas i+2, la voie 01 est réservée pour l'observation de A.

. La colonne (6) liste les unités activées par l'état considéré et qui sont observables indirectement.

. La colonne (7) liste les unités qui sont utilisées : ce sont, par exemple, des registres qui sont lus mais non écrits pendant cet état.

Note : les colonnes (6) et (7) sont induites par l'algorithme mais on ne les note que si elles apportent un élément nouveau pour le test.

. La colonne (8) liste les vecteurs de commande qui sont testés et observés au pas de test considéré.

. La colonne (9) liste les états dont l'état en cours est distingué par les unités observées - directement ou indirectement - (utilisation de la relation de distinguabilité et de la propriété de perturbation).

. La colonne (10) indique la transition qui est vérifiée au pas de test considéré.

b) dépendance des colonnes (5) et (6) et le test des commandes

Si la valeur d'une unité B dépend de la valeur générée dans l'unité A, cette dépendance étant induite par l'algorithme, ceci induit que :

. matériellement A est observable à travers B : on note cette dépendance $B \leftarrow A$

. le vecteur de commande C_i^A est testé lors du passage dans l'état i et observé à travers B lors du passage par l'état j (i activant A et j activant B).

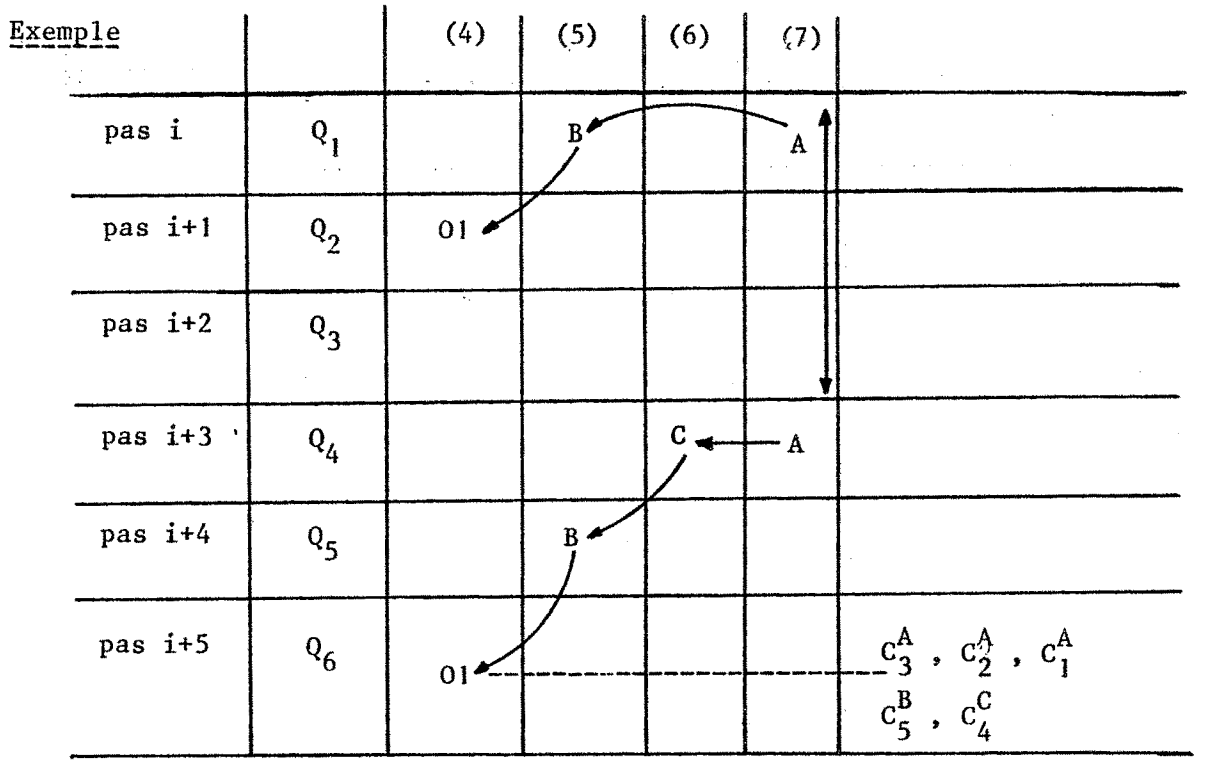
Exemple

	Etat	(4)	(5)	(6)	(8)
pas i	Q_α		A	B , C	
pas i+1	Q_β	01	A	D	C_α^A
pas i+2	Q_γ	01	A		C_β^A
pas i+3	Q_δ	01			C_γ^A , C_β^D , C_α^B , C_α^C

c) cas particulier des unités utilisées: colonne (7)

Le principe de dépendance est le même : dépendance entre une unité observable directement : (4) ou (5), ou une unité observable indirectement (6) et une unité utilisée (7).

Mais l'implication sur le test des commandes est un peu différent : cette unité étant seulement lue (par exemple au pas i), lorsqu'on l'observera à travers une autre unité, on pourra assurer qu'il n'y a pas eu de chargement à tort de cette unité du dernier point d'observation jusqu'au pas i-1 (chargement au pas i inobservable - bascule maître esclave).



Algorithme de test

(1) Choix d'un ensemble minimal de chemins couvrant toutes les transitions : c'est une suite d'états Q_o ... Q_n

---

(2) Etude du pas i de l'algorithme ↔ Q(i)

a.- *marquer dans (10) le séquencement*
Q(i-1) → Q(i) s'il n'est pas déjà testé
- mise à jour de la matrice S.

---

b.- *Lister les unités activées par Q(i) indirectement observables soit UA*
Lister les unités utilisées par Q(i) : soit UU
remarque : UU ∩ UA peut être non vide

---

c.- noter les sorties indisponibles (colonne 4)
- fléchage $5(j) \rightarrow 4(i)$ $j < i$
- noter dans la colonne 8 : $C^{Q(j)}$ $j < i$
- Mise à jour de la matrice C

---

d.- Choix des unités pour la distinguabilité

1. Choisir des unités directement observables
   s'il y a observabilité immédiate, prendre en compte
   les sorties disponibles au pas i

2. si nécessaire, choisir des unités activées indirectement
   observables ($\in$ UA)

1.2 $\Longrightarrow$ on remplit partiellement les colonnes 3,5 et 6.

Remarque : les unités choisies dans 3 et 5 ne sont pas nécessairement des unités activées par l'état Q(i).

3. Mise à jour de la matrice D
   fléchage $6(i) \rightarrow 3(i)$ et $6(i) \rightarrow 5(i)$
   noter dans la colonne 8 : $C^{Q(i)}$
   Mise à jour de la matrice C.

---

e. Choix des unités pour le test des commandes

1. en fonction des commandes déjà testées et des sorties
   encore disponibles en i, choisir des unités observables
   directement et immédiatement
   $\Rightarrow$ compléter colonne 3 et colonne 8
   Mise à jour de C.

2. en fonction des commandes déjà testées on peut choisir
   - des unités observables directement en différé
     (compatibilité avec les retards)
   - des unités activées indirectement
     $\Rightarrow$ compléter colonnes 5 et 6

3. fléchage 6(i) → 5(i)
   6(i) → 5(i)
   Mise à jour de C.

---

6. *Choix des unités utilisées, pour le test des commandes*

1. parmi les unités utilisées par Q(i), lister dans la colonne 7 celles qui apparaissent en 6(j), j < i et qui n'ont pas encore de fléchage
   flécher 6(j) → 7(i)

2. en fonction des commandes déjà testées, choisir des unités $\in UU$ et n'appartenant pas à 3(i) ou 5(i)
   fléchage 7(i) → 3(i)
   7(i) → 5(i)
   7(i) → 6(i)
   Mise à jour de C.

---

(3) Est-ce que i = n?

oui → . Le séquencement est testé
. Est-ce que les matrices D et C sont complètement marquées ?

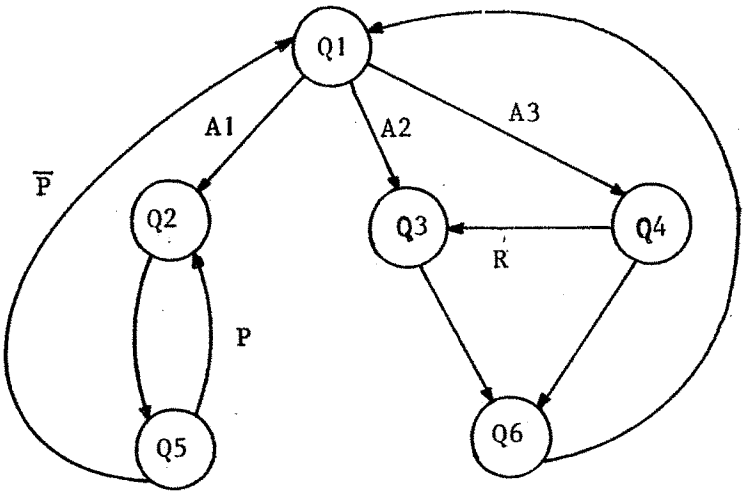
non → i = i+1
aller en (2)

Oui → Fin

en fonction de ce qui reste à tester
choisir un ou plusieurs chemins : $Q'_o \ldots Q'_m$ de i = 0 à m
appliquer le pas (2).

III - 4. EXEMPLE THEORIQUE

Considérons une partie contrôle modélisée comme suit :



et dont la matrice de représentation est la suivante :

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6	
C_1	0	1	0	1	0	0	
C_2	0	0	0	0	1	0	unité u1
C_3	1	0	0	0	0	1	
C_4	1	0	0	1	0	1	unité u2
C_5	0	1	1	0	1	0	
C_6	1	1	1	0	0	0	unité u3
C_7	0	0	0	1	0	1	

On suppose que :

- les unités u1 et u3 sont observables sur un bus B1 et l'unité u2 est observable sur un bus B2.

- les bus B1 et B2 peuvent être lus simultanément
- l'unité u3 est observable en différé : Δ (u3) = 1.

Cet exemple théorique, très simple, ne fait pas intervenir d'unités à observabilité indirecte. Le tableau de test associé ne comporte donc qu'une partie des colonnes définies précédemment.

Les différents chemins possibles sont :

Q1 (Q2 Q5)* Q1
Q1 Q3 Q6 Q1
Q1 Q4 Q3 Q6 Q1
Q1 Q4 Q6 Q1

α) *Choix initial d'un ensemble de chemins*

On choisit l'ensemble { Q1 Q2 Q5 Q1 , Q1 Q4 Q3 Q6 Q1} qui permet de passer par tous les états.

β) *Exécution de l'algorithme de test*

Pas du test	Etat Q_i	Unités observées directement en immédiat	Sortie indisponibles	Unités observées directement en différé	Etats dont Q_i est distingué	Commandes testées	Transitions activées
1	Q1	u2		u3	2,3,4,5,6	C_1^2	
2	Q2	u2	B1		1,4,6	C_2^2, C_1^3	$Q1 \xrightarrow{A1} Q2$
3	Q5	u1,u2	-	-	1,2,3,4,6	C_5^1, C_5^2	$Q2 \xrightarrow{1} Q5$
4	Q1	u1				C_1^1	$Q5 \xrightarrow{P} Q1$
5	Q2	u1			1,3,5,6	C_2^1	
6	Q5			u3		-	
7	Q1		B1			C_5^3	
8	Q4	u1,u2			1,2,3,5,6	C_4^1, C_4^2	$Q1 \xrightarrow{A3} Q4$
9	Q3	u1,u2			1,2,4,5,6	C_3^1, C_3^2	$Q4 \xrightarrow{R} Q3$
10	Q6	u1		u3	1,2,3,4,5	C_6^1	$Q3 \xrightarrow{1} Q6$
11	Q1		B1			C_6^3	$Q6 \xrightarrow{1} Q1$

Fin du test de distinguabilité : $\mathcal{D}$ remplie, C marquée à 80%, S marquée à 70%.

Test complémentaire : marquage de M

A	Q2			u3			
B	Q5		B1			C_2^3	
C	Q1						
D	Q3			u3			$Q^{1} \overset{A2}{\rightarrow} Q3$
E	Q6	u2	B1			C_6^2, C_3^3	
F	Q1						
G	Q4			u3			$Q4 \overset{B}{\rightarrow} Q6$
H	Q6		B1			C_4^3	
I	Q1						

Fin du test complet d'identification : M marquée, S marquée à 90%

Test de séquencement : marquage de S

α	Q2	u1					
β	Q5	u1					
γ	Q2	u1					$Q5 \overset{P}{\rightarrow} Q2$

Fin du test de séquencement : S est marquée

La matrice de distinguabilité $\mathcal{D}$ est remplie aux pas 0 à 11

	Q1	Q2	Q3	Q4	Q5	Q6
Q1		1	1	1	1	1
Q2	2		5	2	5	2
Q3	9	9		9	9	9
Q4	8	8	8		8	8
Q5	3	3	3	3		3
Q6	10	10	10	10	10	

La matrice de commande C est marquée aux pas 0 à 11 et A à I.

	Q1	Q2	Q3	Q4	Q5	Q6	
C^1	4	5	9	8	3	10	U1
C^2	1	2	9	8	3	E	U2
C^3	2	B	E	H	7	11	U3

La matrice de séquencement est marquée au cours des pas 0 à 11, A à I et α à γ du test

	Q1	Q2	Q3	Q4	Q5	Q6
Q1	0	A1 pas 2	A2 pas E	A3 pas 8	0	0
	0	0	0	0	1 pas 3	0
Q3	0	0	0	0	0	1 pas 10
Q4	0	0	R pas 9	0	0	$\bar{R}$ pas H
Q5	$\bar{P}$ pas 4	P pas γ	0	0	0	0
Q6	1 pas 11	0	0	0	0	0

IV - EXEMPLE PRATIQUE

L'exemple pratique étudié est l'unité de contrôle locale d'une unité arithmétique et logique. Cette unité câblée fait partie d'un calculateur numérique microprogrammé, orienté vers des traitements en temps réel (géophysique) [ROB, 75a].

Cette unité arithmétique et logique (Figure 6) est spécialisée dans les opérations sur nombres flottants. Elle effectue également les opérations d'addition et de multiplication sur nombres entiers.

Les opérations sur les nombres flottants, la multiplication et les décalages sont initialisés par microprogramme et s'exécutent automatiquement sur plusieurs cycles.

L'UAL est principalement composée de :

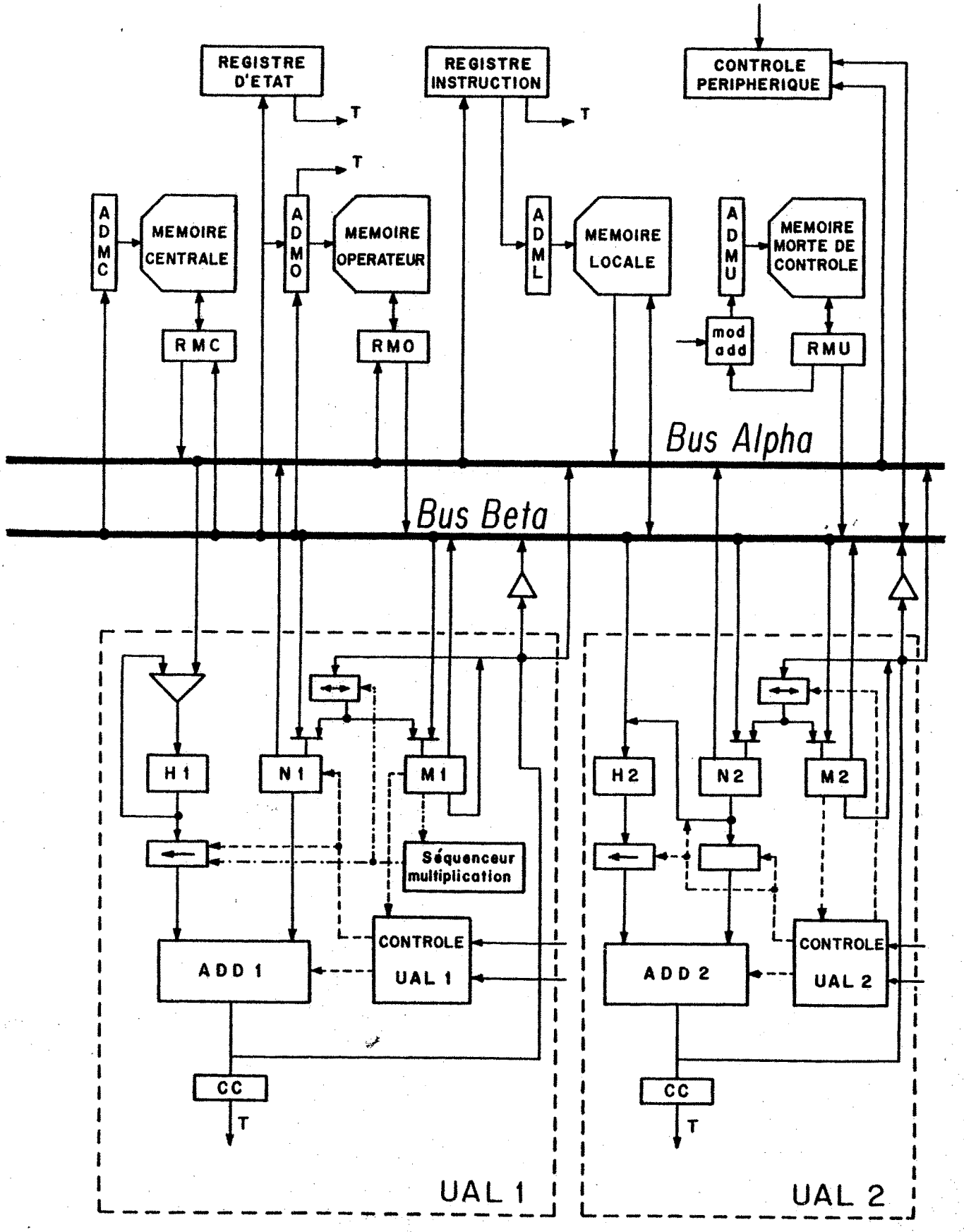
- quatre registres H, N, M, RT
- deux additionneurs ADD1 et ADD2
- la partie contrôle assurant la gestion automatique des opérations.

Les opérations automatiques sur nombres flottants sont au nombre de 10 : addition / soustraction, multiplication, normalisation, décalages arithmétiques ou logiques, ...

Chaque opération automatique exécutée est représentée par un algorithme (Figure 7) : à chaque phase de l'algorithme (ou noeud) est associée une bascule, active pendant un cycle de base ; cette bascule valide un ensemble de commandes exécutées simultanément, qui activent les différentes unités fonctionnelles de l'UAL.

La réalisation physique, relative à l'ensemble de ces algorithmes, est une imbrication de tous les algorithmes (Figure 8) ; le schéma de câblage est directement déduit de cette structure (codage 1 parmi n).

Nous allons exposer la méthode de test sur un algorithme particulier, celui d'addition - soustraction sur nombres flottants, donné en Figure 7.


REGISTRE D'ETAT
REGISTRE INSTRUCTION
CONTROLE PERIPHERIQUE
T
T
T
ADMC
MEMOIRE CENTRALE
RMC
ADMO
MEMOIRE OPERATEUR
RMO
ADML
MEMOIRE LOCALE
ADMU
MEMOIRE MORTE DE CONTROLE
mod add
RMU
Bus Alpha
Bus Beta
H1
N1
M1
Séquenceur multiplication
CONTROLE UAL 1
ADD 1
CC
T
UAL 1
H2
N2
M2
CONTROLE UAL 2
ADD 2
CC
T
UAL 2

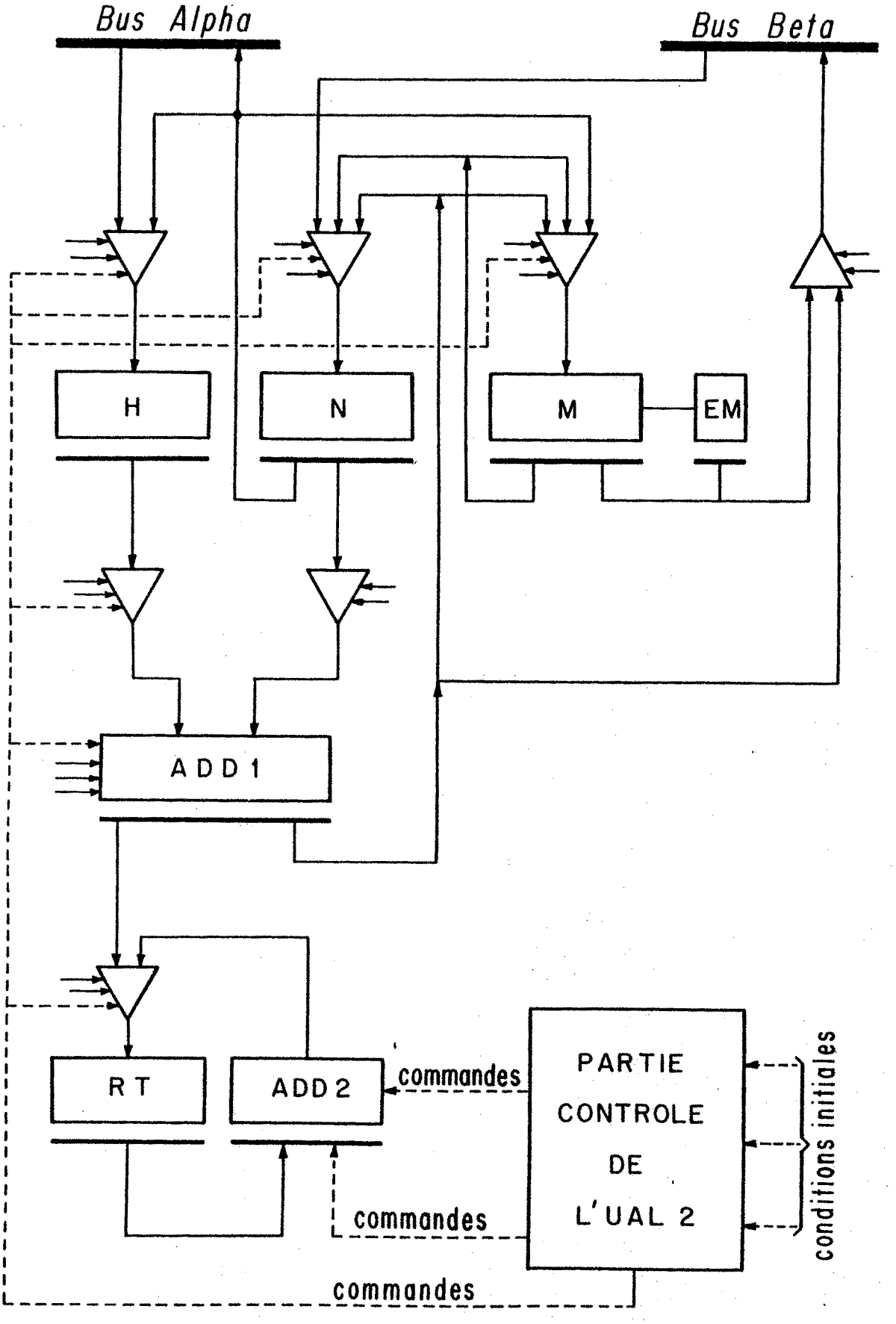


Figure 6. Structure de l'unité arithmétique et logique

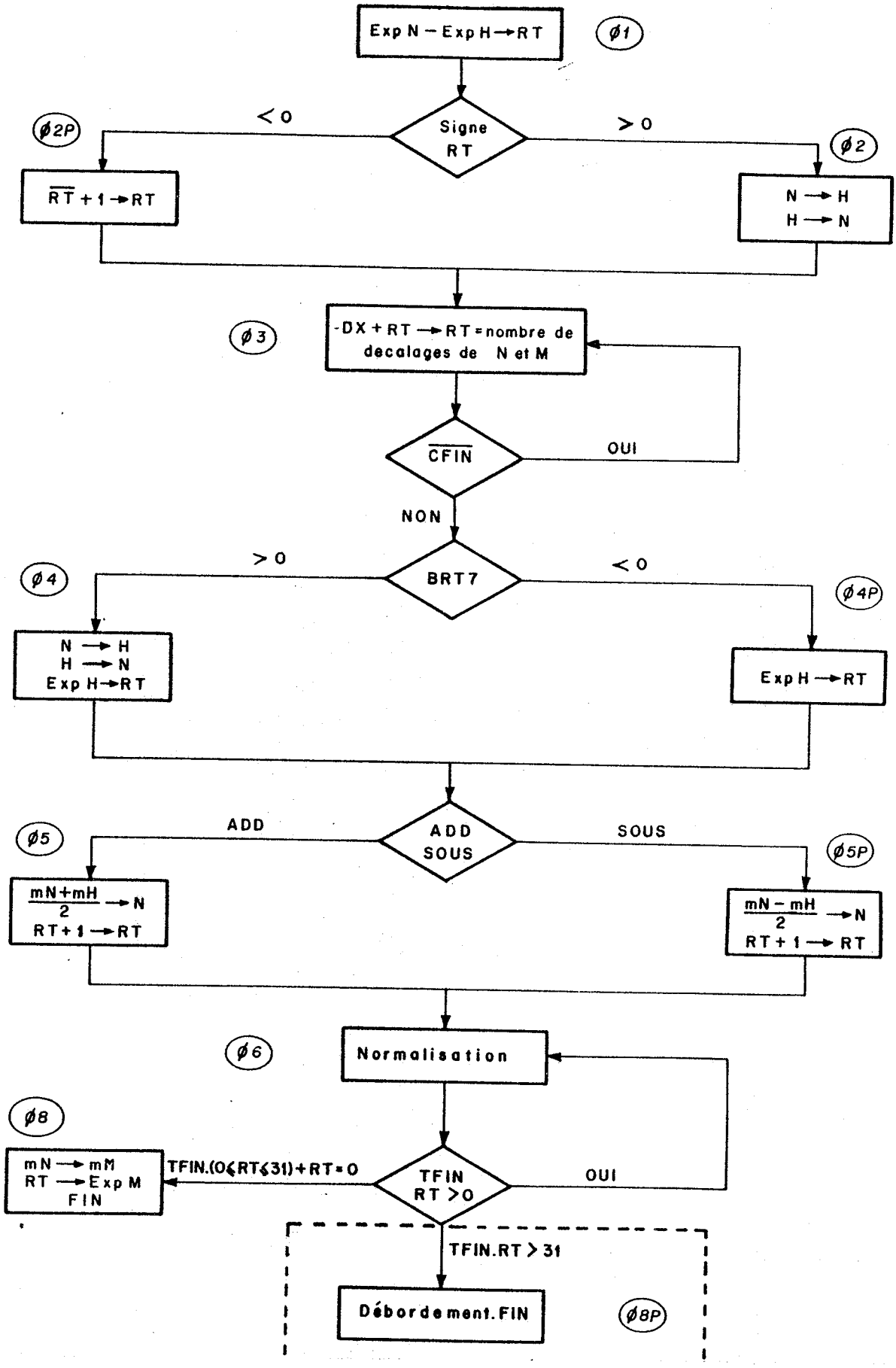


Figure 7. Algorithme d'addition- soustraction

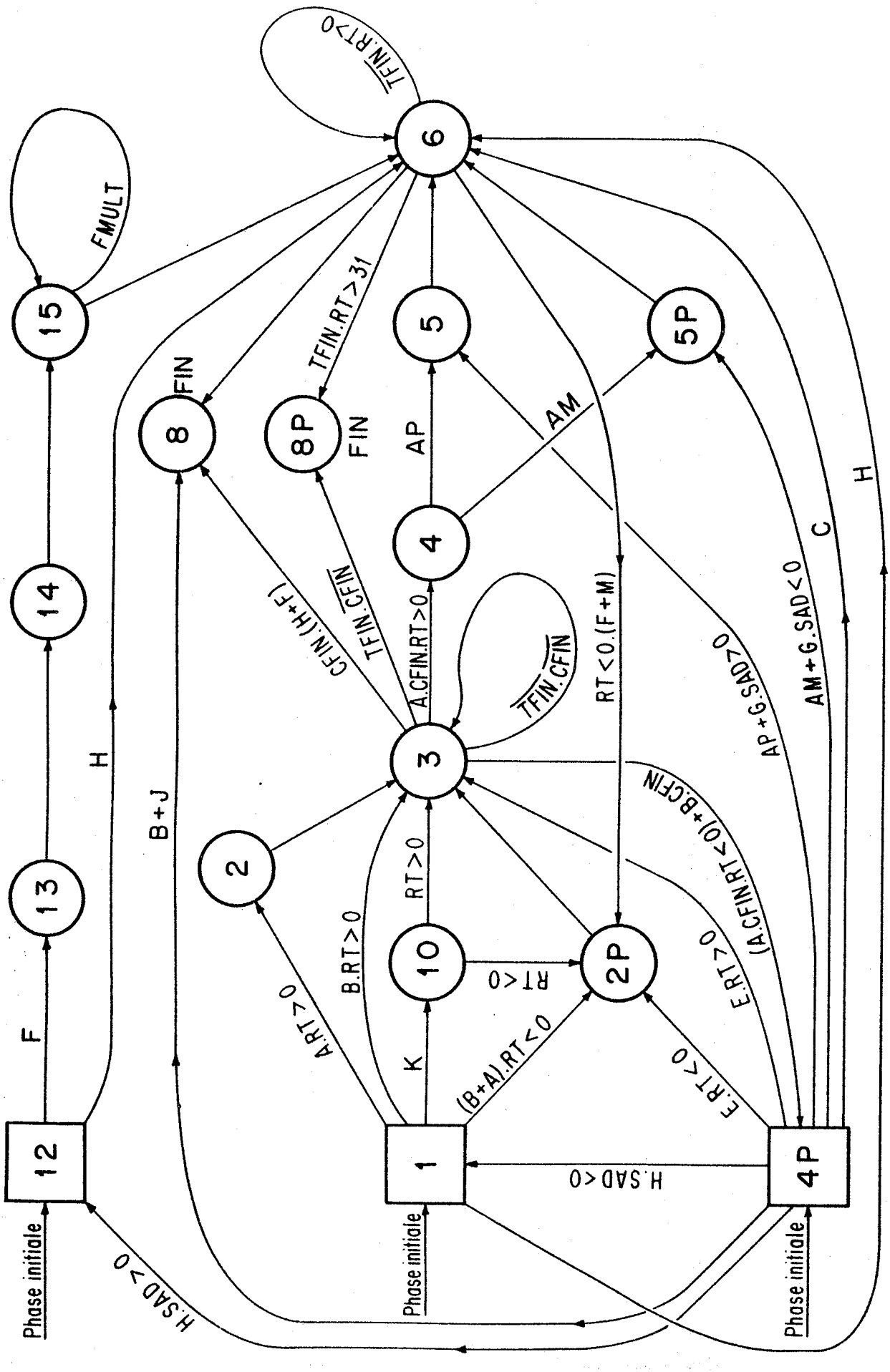


Figure 8. Représentation de la partie contrôle

IV - 1. ANALYSE DE L'ALGORITHME

Cet algorithme réalise l'addition / soustraction de deux nombres flottants : H ± N.

La première étape consiste à égaliser les exposants. On prendra pour exposant du résultat non normalisé le plus grand des exposants des opérandes (Ø1, Ø2, Ø2 P). Il suffit alors de décaler sur la droite la mantisse correspondant à l'exposant le plus faible, d'un nombre de positions égal à la différence des exposants des opérandes (Ø3).

Sur les opérandes modifiés (Ø4, Ø4 P) on fait alors l'addition (Ø5) ou la soustraction (Ø5 P) des mantisses. Après normalisation (Ø6) le résultat est chargé dans le registre M (Ø8).

Nous allons analyser les différentes phases et leurs actions sur les unités fonctionnelles (Figure 9).

. Phase 1 : Opérations { Exposant N - Exposant H → RT ; Signe de l'opération → BRT7

Commandes
- CXH : transfert exposant H
- CXN : transfert exposant N
- SOUSA : exp. N - exp. H
- REA : retenue additionneur
- DIRT (0:4) : transfert F (0:4) → RT (0:4)
- DIRT (5) : transfert F (5) → RT (5)
- DIRT (6:7) : transfert F (5) → RT (6:7)
- HRT : horloge forçage bascule signe BRT7
- HRT (0:6) : horloge registre RT.

. Phase 2 :

Opérations N → H ; H → N

Commandes
- HRH : horloge registre H
- TNH : transfert N → H
- CXH : transfert exposant H
- CMH : transfert mantisse H
- ADDA : addition H + O
- DIAD : sortie additionneur dans N
- HRN : horloge registre N.

. Phase 2P

Opération : $\overline{RT} + 1 \rightarrow RT$

Commandes : - ADDR : commande soustraction dans ADD2
- RER : retenue d'entrée ADD2
- SRT : transfert sortie ADD2 → RT
- HRT (0:6) : horloge RT (0:6).

. Phase 3

Opération : décalage des mantisses de N et M du contenu de RT
- DX + RT → RT

Commandes : - HRN : horloge registre N
- CMN : transfert mantisse de N
- TDNA : N + 0
- HRM : horloge registre M
- DX : nombre de décalages à soustraire du contenu de RT
- SRT : transfert sortie ADD2 → RT
- HRT (0:6) : horloge RT (0:6).

. Phase 4

Opération N → H ; H → N ; exposant H → RT

phase 2

Commandes : mêmes commandes qu'en phase 2 et
- DIRT (0:4) : transfert F (0:4) → RT (0:4)
- HRT (0:6) : horloge registre RT (0:6)

. Phase 4P

Opération : Exposant H → RT

Commandes : - CXH : transfert exposant H
- ADDA : addition exp H + 0
- DIRT (0:4) : transfert F (0:4) → RT (0:4)
- HRT (0:6) : horloge RT (0:6).

. Phase 5

Opération : $\frac{\text{mantisse N + mantisse H}}{2} \rightarrow N$

RT + 1 → RT

Pour éviter tout débordement résultant de l'addition des mantisses, on décale le résultat de l'opération d'une position à droite et on incrémente RT.

Commandes :
- HRN : horloge registre N
- CMN : transfert mantisse de N
- CMH : transfert mantisse de H
- ADDA : addition
- DD1 : décalage une position sur N
- HRT (0:6): horloge RT (0:6)
- IRT : incrémentation registre RT.

. Phase 5P

Opération : $\frac{\text{mantisse N - mantisse H}}{2} \rightarrow N$

RT + 1 → RT

Commandes : mêmes commandes qu'en phase 5, à l'exception de ADDA qui est remplacé par

- SOUSA : soustraction
- REA : retenue d'entrée de l'additionneur.

. Phase 6

Opération : Normalisation sur tests : décalage gauche de N et M du contenu de RT.

Commandes :
- HRN : horloge registre N
- CMN : transfert mantisse de N
- TDNA : addition N + 0
- HRM : horloge registre M
- HRT (0:6) : horloge registre RT
- DX : nombre de décalages à soustraire de RT
- SRT : transfert sortie ADD2 → RT

. Phase 8

Opération : mantisse N → mantisse M

RT → exposant M

Commandes : - CTRM : commande de transfert

mant N → mant M

RT → exp. M

- HRM : horloge registre M

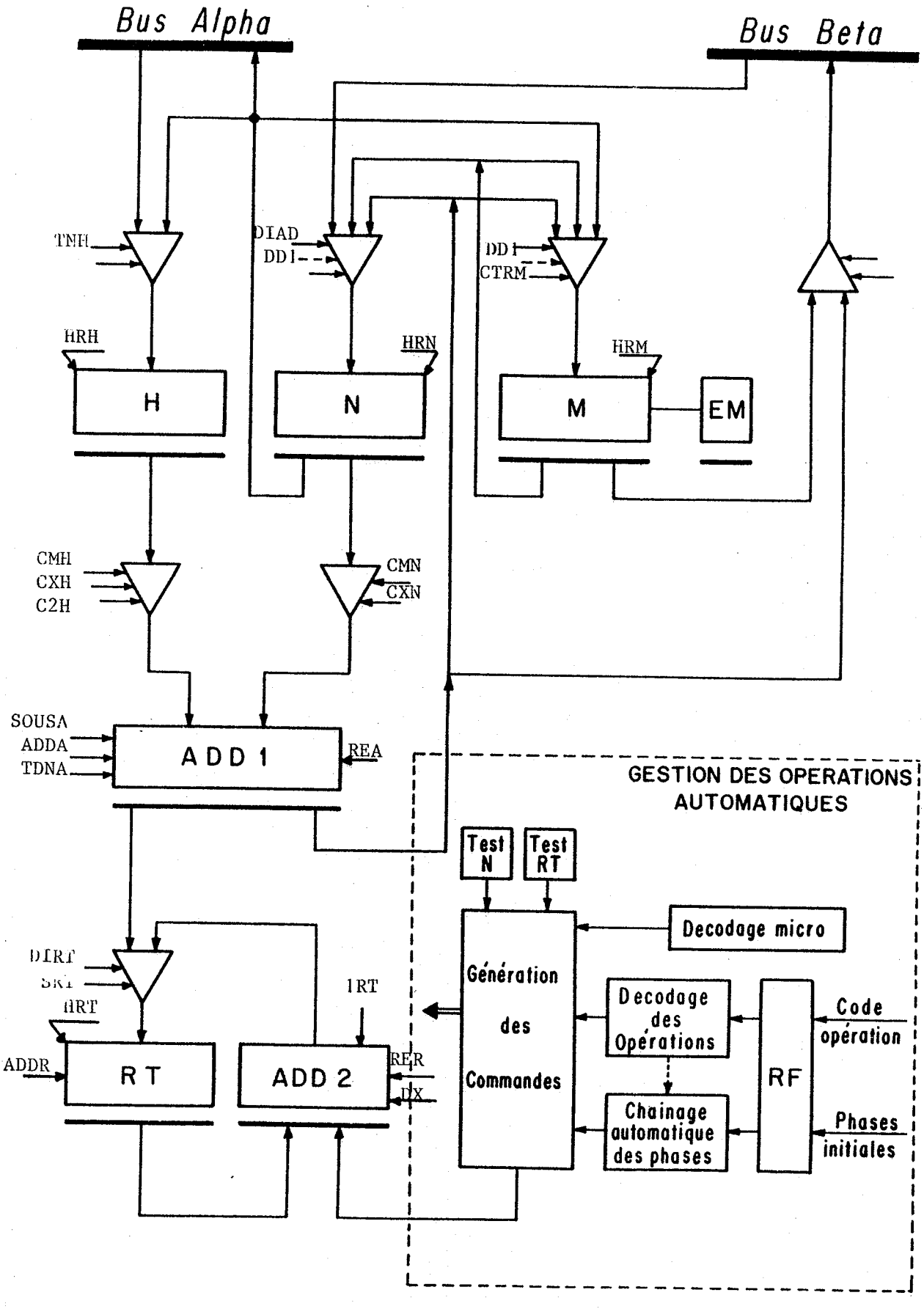


Figure 9. Stucture de l'UAL et ses commandes

Matrice de représentation

	Q1	Q2	Q2P	Q3	Q4	Q4P	Q5	Q5P	Q6	Q8	
REA	1							1			additionneur ADD1
SOUSA	1							1			
ADDA		1			1	1	1				
TDNA				1					1		
CMH		1			1		1	1			
CXH	1	1			1	1					
CMN				1			1	1	1		
CXN	1										
HRH		1			1						registre H
TNH		1			1						
HRN		1		1*	1		1	1	1**		registre N
DD1							1	1			
DIAD		1			1						
CTRM										1	registre M
HRM				1*					1**	1	
ADDR			1								additionneur ADD2
RER			1								
DX				1					1		
DIRT 0-4	1				1	1					registre RT
DIRT 5	1										
DIRT 6-7	1										
HRT 0-6	1		1	1*	1	1	1	1	1*		
HRT 7	1										
1 RT							1	1			
SRT			1	1*					1**		

Commentaires :

1^* signifie que la commande est active tant qu'on est en cours de décalage (sur test des bits de l'opérande)

1^{**} signifie que la commande est active tant qu'on est en normalisation (sur test).

IV -2 . PARCOURS DE L'ALGORITHME

Il n'existe que quatre chemins possibles dans cet algorithme :

1 - 2P - 3^* - 4P - 5 - 6^* - 8
1 - 2 - 3^* - 4 - 5 - 6^* - 8
1 - 2P - 3^* - 4P - 5P - 6^* - 8
1 - 2 - 3^* - 4 - 5P - 6^* - 8

En effet, l'état 2P ne peut être suivi que de l'état 4P : ces deux états sont activés lorsque la différence des exposants de N et de H est négative ; de même l'état 2 ne peut être suivi que de l'état 4 (différence positive).

Le parcours de chacun de ces chemins induit une détermination partielle des opérandes d'entrée : par exemple, le passage par le premier chemin impose d'avoir exposant H > exposant N.

IV - 3 . OBSERVABILITE DES UNITES FONCTIONNELLES

. Observabilité spatiale

L'additionneur ADD1 est directement observable sur le bus BETA (commande ORBES3), de même que le registre N sur le bus ALPHA (commande ORAL2) et le registre M sur le bus BETA (commande ORBE2).

Les autres unités sont observables indirectement.

I(H) = {ADD}
I(RT) = {N,M,ADD2}
I(ADD2) = {RT}

. Observabilité temporelle

Les additionneurs ont une observabilité immédiate

Δ (ADD1) = Δ (ADD2) = 0

Les registres ont une observabilité différée

Δ (H) = Δ (N) = Δ (M) = Δ (RT) = 1

. Observabilité parallèle

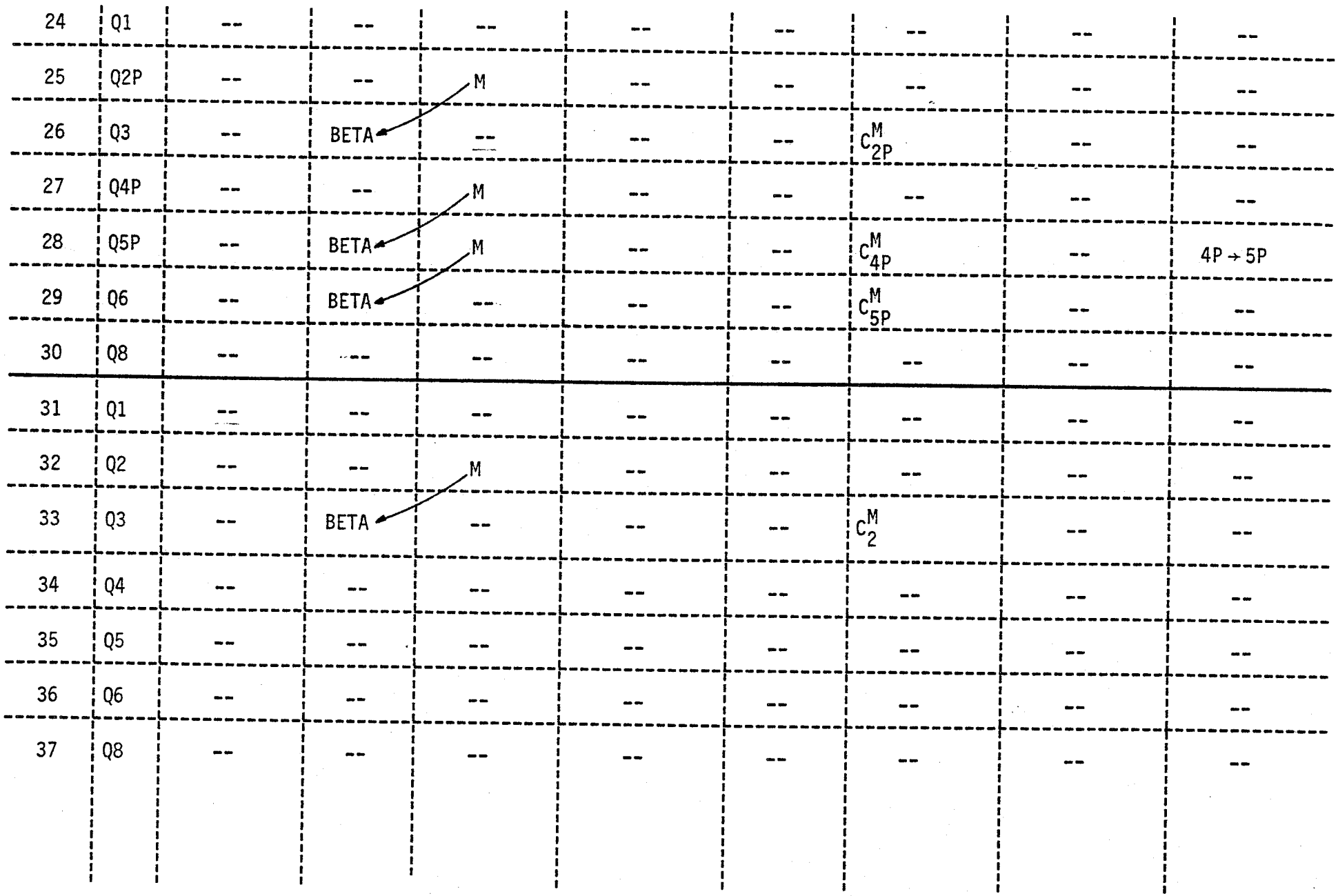
Il y a deux voies de communication de la partie opérative vers l'extérieur : les bus ALPHA et BETA. Ces deux bus permettent donc une observabilité simultanée de deux unités.

soit ADD1 et N respectivement par ORBE3 et ORAL
soit M et N respectivement par ORBE2 et ORAL.

IV - 4 . ALGORITHME DE TEST

Pas de test	Etat Qi	Unités observées directement en immédiat	Sorties indispo-nibles	Unités observées directement en différé	Unités activées indirectement observées	Autres unités utilisées	Commandes testées et observées	Etats dont Qi est distingué	Séquencement vérifié
1	Q1	ADD1	--	N	RT	H	C_1^{ADD1}	tout état par ADD1	--
2	Q2P	ADD1	ALPHA	N	RT , ADD2	--	C_{2P}^{ADD1} , C_1^N	Q8 par RT, les autres par ADD1	1→ 2P
3	Q3	ADD1	ALPHA	N	RT , ADD2	--	C_3^{ADD1} , C_{2P}^N	Q6 par N, les autres par ADD1	2P→ 3 3→ 3
4	Q4P	ADD1	ALPHA	N	RT	H	C_{4P}^{ADD1}, C_3^N, C_{2P}^{RT}, C_1^{RT}, C_{2P}^{ADD2}, C_3^{RT}, C_3^{ADD2}	tout état par ADD1	3→ 4P
5	Q5	ADD1	ALPHA	N	RT , ADD2	H	C_5^{ADD1} , C_{4P}^N	tout état par ADD1	4P→ 5
6	Q6	ADD1	ALPHA	N M	RT , ADD2	--	C_6^{ADD1}, C_5^N, C_1^H, C_{2P}^H, C_3^H, C_{4P}^H	tout état par N ou M	5→ 6 6→ 6
7	Q8	--	ALPHA BETA	M	--	N , RT	C_6^M, C_6^N, C_5^{RT}, C_{4P}^{RT}, C_5^{ADD2}, C_6^{RT}, C_6^{ADD2}	tout état par M	6→ 8
8	lecture résultat	--	BETA	--	--	--	C_8^M	--	--

9	Q1	--	--	M	--	--	--	--	--	
10	Q2	--	BETA	N	RT	H	C_1^M	Q4 par RT, autres par N	1→2	
11	Q3	--	ALPHA	N	RT , ADD2	--	C_2^N, C_2^H	--	2→3	
12	Q4	ADD1	ALPHA	N	RT	H	C_4^{ADD1}, C_2^{RT}	Q2,Q3 par RT, autres par ADD1	3→4	
13	Q5P	ADD1	ALPHA	N	RT , ADD2	H	C_4^N, C_{5P}^{ADD1}	tout état par ADD1	4→5P	
14	Q6	--	ALPHA	N	RT , ADD2	--	C_{5P}^N, C_4^H	--	5P→6	
15	Q8	--	ALPHA BETA	M N	--	--	$C_{5P}^{RT}, C_{5P}^{ADD2}, C_4^{RT}$	--	--	
16	lecture résultat	--	ALPHA	--	--	--	C_8^N	--	--	
17	Q1	--	--	--	--	--	--	--	--	
18	Q2	ADD1	--	--	--	--	C_2^{ADD1}	--	--	
19	Q3	--	--	M	--	--	--	--	--	
20	Q4	--	BETA	M	--	--	C_3^M	--	--	
21	Q5	--	BETA	M	--	--	C_4^M	--	4→5	
22	Q6	--	BETA	--	--	--	C_5^M	--	--	
23	Q8	--	--	--	--	--	--	--	--	


24 Q1 -- -- -- -- -- -- -- --
25 Q2P -- -- M -- -- -- -- --
26 Q3 -- BETA -- -- -- C_{2P}^{M} -- --
27 Q4P -- -- M -- -- -- -- --
28 Q5P -- BETA M -- -- C_{4P}^{M} -- 4P → 5P
29 Q6 -- BETA -- -- -- C_{5P}^{M} -- --
30 Q8 -- -- -- -- -- -- -- --
31 Q1 -- -- -- -- -- -- -- --
32 Q2 -- -- M -- -- -- -- --
33 Q3 -- BETA -- -- -- C_{2}^{M} -- --
34 Q4 -- -- -- -- -- -- -- --
35 Q5 -- -- -- -- -- -- -- --
36 Q6 -- -- -- -- -- -- -- --
37 Q8 -- -- -- -- -- -- -- --

Marquage de la matrice de distinguabilité

	Q1	Q2	Q2P	Q3	Q4	Q4P	Q5	Q5P	Q6	Q8
Q1		1	1	1	1	1	1	1	1	1
Q2	10		10	10	10	10	10	10	10	10
Q2P	2	2		2	2	2	2	2	2	2
Q3	3	3	3		3	3	3	3	3	3
Q4	12	12	12	12		12	12	12	12	12
Q4P	4	4	4	4	4		4	4	4	4
Q5	5	5	5	5	5	5		5	5	5
Q5P	13	13	13	13	13	13	13		13	13
Q6	6	6	6	6	6	6	6	6		6
Q8	7	7	7	7	7	7	7	7	7	

Marquage de la matrice de commande :

	Q1	Q2	Q2P	Q3	Q4	Q4P	Q5	Q5P	Q6	Q8
C^{ADD1}	1	18	2	3	12	4	5	13	6	
C^{H}	6	11	6	6	14	6				
C^{N}	2	11	3	4	13	5	6	14	7	16
C^{M}	10	33	26	20	21	28	22	29	7	8
C^{ADD2}			4	4			7	15	7	
$C^{RT.}$	4	12	4	4	15	7	7	15	7	

Marquage de la matrice de séquencement (on n'a pas noté le prédicat associé)

Qi → Qj	Q1	Q2	Q2P	Q3	Q4	Q4P	Q5	Q5P	Q6	Q8
Q1		10	2							
Q2				11						
Q2P				3						
Q3				3	12	4				
Q4							21	13		
Q4P							5	28		
Q5									6	
Q5P									14	
Q6									6	7
Q8										

IV - 5 . CONCLUSION

Sur l'ensemble des dix algorithmes, on a obtenu le résultat suivant :

- 10 chemins ont été nécessaires pour tester les états (éléments de mémorisation) : identification complète
- les pannes de séquencement ont été testées en parcourant 30 chemins.

Pour l'ensemble du contrôle (215 portes, 21 bascules et 1 compteur) le microprogramme de test a nécessité :

- 100 microinstructions, dont le cycle de base est 70 ns
- 340 vecteurs de test : opérandes de test X et P et résultats à comparer aux résultats réels, stockés en mémoire centrale.

CHAPITRE III

LE TEST D'UN SYSTÈME À MICROPROCESSEUR

INTRODUCTION

Le concepteur de systèmes à microprocesseurs se trouve confronté au problème du test des microprocesseurs, d'une part en cours et en fin de fabrication du système, d'autre part lors de la vie du système (maintenance préventive et curative). Ce problème est d'autant plus crucial pour le concepteur de systèmes à haute sécurité.

Les deux préoccupations du concepteur sont :

* la qualité du test qu'il désire obtenir; cette qualité est généralement dépendante de la méthode de test utilisée et doit être évaluée par son taux de couverture par rapport à l'ensemble des pannes possibles. Notons que l'importance de ce facteur de couverture a été largement démontré [BOU,71];
* la mise en oeuvre du test : coût, investissement d'un équipement de test, temps d'exécution du test.

La Section 1 fait le point des méthodes de test utilisées (§I), étudie le problème de la mise en oeuvre du test (§II) et analyse l'impact du test sur la conception (§III).

Une approche particulière, exposée en Section 2, concerne les microprocesseurs intégrés dans un système à application spécifique; cette approche consiste à réaliser le test à travers le programme d'application. Elle est particulièrement attrayante pour l'utilisateur (coût, mise en oeuvre) et s'applique surtout au contexte de la maintenance.

SECTION 1 : LE TEST DES SYSTEMES A MICROPROCESSEURS : L'ETAT DE L'ART ET LES PERSPECTIVES

INTRODUCTION [ROB,79]

Les microprocesseurs sont parmi les plus complexes des circuits LSI fabriqués et posent en tant que tels des problèmes nouveaux pour le test. Le problème se pose aussi bien au constructeur qui doit vérifier la qualité de son produit en sortie de chaîne de fabrication, qu'à l'utilisateur qui doit prévoir la maintenance des systèmes utilisant ces circuits.

Or plusieurs raisons font que ce problème se pose en termes différents de celui du test des circuits logiques classiques.
Il y a trois points de vue bien différents dans le test des microprocesseurs qui induisent des méthodes de test bien spécifiques :

- celui du constructeur
- celui du fabricant de systèmes à base de microprocesseurs
- celui de l'utilisateur.

1. Le point de vue du constructeur:

Pour le constructeur, le microprocesseur est un circuit intégré nu, avec son jeu d'instructions; il a accès aux diagrammes logiques détaillés de son circuit, son volume de test est élevé et il doit garantir le bon fonctionnement de son circuit à l'intérieur de ses spécifications.
La plupart du temps, le programme de test défini est appliqué dans des conditions de pire cas (test dynamique); ces conditions de pire cas sont fournies par un retour client ou par l'expérience : par exemple, des instructions, séquences d'instructions, données, séquences d'entrées/sorties,... connues pour être difficiles à exécuter par le microprocesseur (Motorola) [HOL,77]. L'expérience a également montré qu'il existe pour les microprocesseurs un phénomène semblable à la "sensibilité aux patterns" dans les mémoires, à savoir : certaines instructions qui sont exécutées correctement lors d'un seul passage, sont cause d'erreurs lorsqu'elles sont répétées dans une boucle (Macrodata)[HOL,77].

2. *Les points de vue du fabricant de systèmes à microprocesseurs et de l'utilisateur:*

Dans ce cas, on peut noter les caractéristiques suivantes :

a) On ne connaît pas de schéma équivalent du circuit : ce schéma logique, pour être représentatif, doit en effet être déduit de la structure des masques eux-mêmes qui sont du domaine du secret industriel.

b) De toutes façons, la connaissance des schémas logiques équivalents est d'une utilisation délicate. En effet, les hypothèses de pannes classiques (collages, court-circuits) qui ont permis des progrès considérables dans le domaine du test, sont remises en cause par les nouvelles technologies utilisées pour la réalisation de tels circuits.

c) Enfin, en ce qui concerne l'utilisateur ou le fabricant (dans ses étapes de fin de fabrication ou d'exploitation), le microprocesseur est généralement implanté dans un système programmé pour une application particulière. De ce fait, à moins de pouvoir désenficher chaque boîtier et le tester à part, on ne peut à ce niveau utiliser aisément des séquences de test préétablies que pourrait fournir le constructeur.
Le microprocesseur n'est alors qu'un des éléments du système dans lequel il est intégré au même titre que la mémoire vive, les circuits d'interface, la mémoire morte... et son test devient alors indissociable du test de ces autres éléments.

Le principal critère du test est donc le niveau d'information sur le microprocesseur. On peut distinguer trois niveaux :

a) *Le niveau architectural* : on ne connaît du microprocesseur que son jeu d'instructions, ce qui induit un test de comportement.

b) *Le niveau fonctionnel* : outre le jeu d'instructions et/ou le programme d'application, on connaît les modules fonctionnels du microprocesseur et, en particulier, ses éléments de mémorisation; on fera alors un test fonctionnel.

c) *Le niveau structurel* : on entre dans le domaine des renseignements matériels fins du microprocesseur (renseignements topologiques).

- connaissance du découpage réel : le microprocesseur peut être décomposé en blocs fonctionnels relativement indépendants, correspondant à l'implémentation réelle du circuit intégré : UAL, compteur programme, logique de décodage, registres et bus de communications avec l'extérieur (données, adresses, contrôle), liaisons entre ces blocs (lignes de données), signaux de contrôle issus de la logique de décodage;
- renseignements supplémentaires fournis par la connaissance du masque :
  . densité des blocs topologiques,
  . schémas logiques cohérents avec la structure matérielle : renseignements tels que la proximité des connexions (métallisation).

Les situations *a)* et *b)* sont les deux situations dans lesquelles se trouve le plus souvent l'utilisateur; la situation c) est généralement l'apanage du fabricant.

I - MÉTHODES DE GÉNÉRATION DE L'INFORMATION DE TEST

On peut soit considérer un microprocesseur comme un composant de grande complexité et chercher à étendre les méthodes de test utilisées pour les composants [ROB,78e], soit le considérer sous son aspect fonctionnel et le traiter par des méthodes de test classiques de processeurs [ROB,75].

Nous nous proposons de rappeler d'abord les méthodes de test classiques aux niveaux composant et sous-système et leur extension aux microprocesseurs.

I - 1. RAPPEL SUR LES METHODES CLASSIQUES DE TEST

A) Définitions [ROB,78e]

. Deux types de test sont utilisés pour détecter les défaillances dans la vie d'un composant :

- des tests paramétriques (statiques ou dynamiques) qui sont appliqués en fin de fabrication du composant ou en contrôle d'entrée; ces tests comparent les limites réelles de certains paramètres (tension, courant, charge,..) à celles spécifiées par le cachier des charges et vérifient les temps de propagation, largeur d'impulsion,...
- des tests logiques qui sont appliqués en fin de fabrication ou en maintenance; ces tests permettent de garantir que le circuit assure sa fonction logique dans des conditions d'environnement semblables à celles de son utilisation (alimentation, température,...).

Ce sont les méthodes de test logique qui font l'objet de la suite de ce chapitre.

. Un défaut ou défaillance est une imperfection physique qui peut entraîner un mauvais fonctionnement du composant (hors spécifications).

. Une panne est la manifestation logique de ce défaut (collage à 0 ou à 1 d'une connexion,...).

. Une erreur est un mauvais fonctionnement du composant dû à une panne.

. Hypothèses de pannes : le modèle de pannes généralement adopté est le modèle de collage, c'est-à-dire : un défaut résulte en un collage permanent d'une ligne à la valeur logique 0 ou 1.
Dans certaines technologies, les court-circuits entre lignes adjacentes peuvent être modélisés par une fonction (ET, OU,..) entre ces lignes.

B) Les méthodes de test classiques au niveau composant

On peut considérer deux classes de méthodes :

- Les méthodes de test avec vecteurs prédéterminés : les vecteurs sont calculés par une étude préalable du circuit et sont ensuite appliqués lors du test effectif du circuit; l'ensemble des vecteurs de test proposés est en général quasi-complet par rapport à un ensemble de pannes. Le taux de couverture des vecteurs évalue l'efficacité du test, par rapport aux hypothèses de pannes considérées.

- Les méthodes de test aléatoire : les données de test sont générées aléatoirement lors du test effectif du circuit.

α) *Les méthodes de test avec vecteurs prédéterminés*

La méthode de génération de ces vecteurs peut être de l'un des trois types suivants :

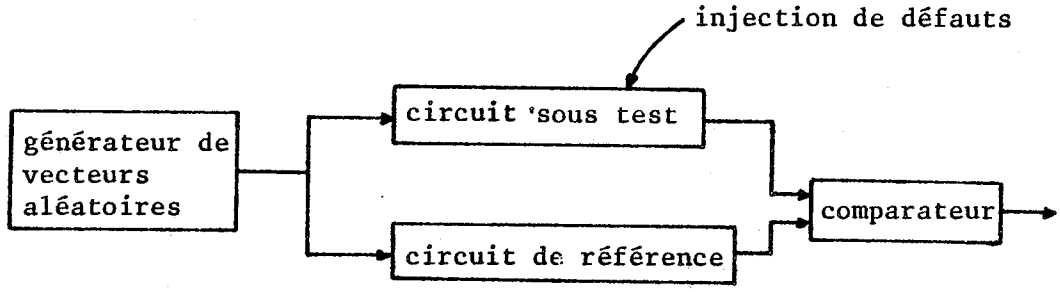
. génération déterministe : ces méthodes prennent en compte la structure ou la fonction du circuit.

Les approches structurelles s'appuient sur une analyse du schéma logique équivalent (approche topologique). Les deux principales méthodes de ce type sont :

- la méthode de propagation des pannes par un chemin de sensibilisation (D-algorithme) [ROT,68]
- une méthode algébrique (différence booléenne).

Les approches fonctionnelles sont indépendantes de la réalisation physique du circuit et prennent uniquement en compte son aspect fonctionnel. On vérifie par exemple qu'un compteur incrémente (ou décrémente) correctement...

. <u>génération aléatoire</u> : les combinaisons d'entrée sont fournies par un générateur de séquences aléatoires; la simulation des circuits en présence de pannes et du circuit juste permet d'établir les séquences de test détectant les pannes appartenant à un dictionnaire de pannes; chaque configuration d'entrée qui détecte au moins une nouvelle panne est enregistrée ainsi que la sortie juste correspondante. Le schéma suivant illustre cette méthode :



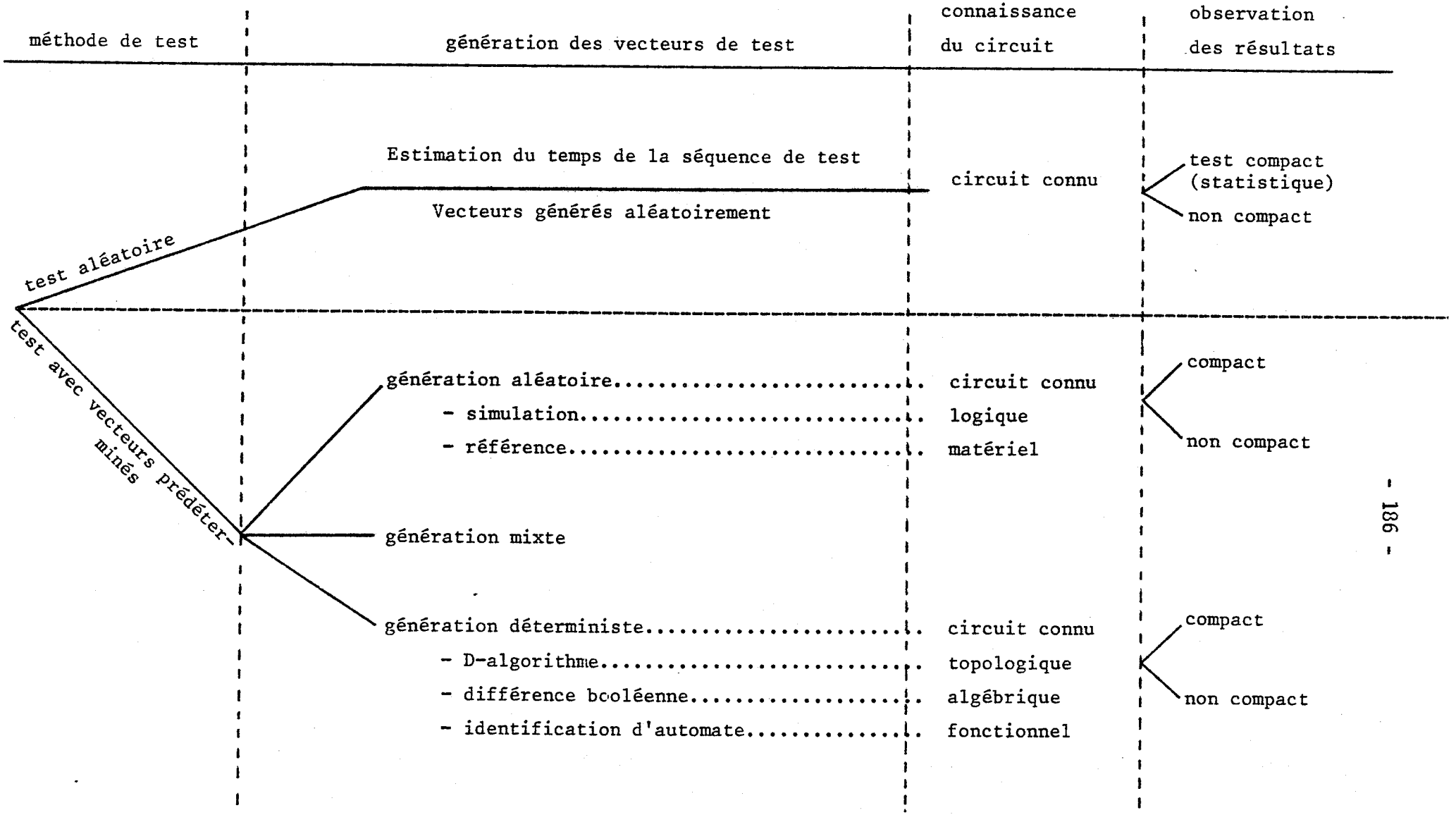
. <u>génération mixte</u> : la méthode utilisée consiste en une combinaison de génération aléatoire et de génération déterministe. La méthode de génération est initialisée par une génération aléatoire. Si, après application d'un nombre fixé de vecteurs, on n'a détecté aucune nouvelle panne, la génération se continue de manière déterministe.

β) *Les méthodes de test aléatoire* [THE,78]

Les vecteurs de test sont générés aléatoirement au moment du test effectif du circuit considéré et envoyés simultanément au circuit sous test et à un circuit réputé bon. Le problème est alors d'estimer la longueur de la séquence de test nécessaire pour assurer une qualité de détection donnée.

On peut considérer deux cas :

- soit la structure du circuit est connue : une analyse probabiliste du circuit permet de déterminer la longueur de la séquence de test; le test est dit <u>aléatoire</u>,
- soit le circuit est connu par des propriétés statistiques : on évalue alors une fonction de la réponse du circuit; le test est dit <u>statistique</u>.


méthode de test
génération des vecteurs de test
connaissance du circuit
observation des résultats
Estimation du temps de la séquence de test
Vecteurs générés aléatoirement
circuit connu
test compact (statistique)
non compact
test aléatoire
test avec vecteurs prédéterminés
génération aléatoire
- simulation
- référence
circuit connu
logique
matériel
compact
non compact
génération mixte
génération déterministe
- D-algorithme
- différence booléenne
- identification d'automate
circuit connu
topologique
algébrique
fonctionnel
compact
non compact

Pour la plupart des méthodes de test définies précédemment, deux options sont possibles :

- soit on observe toute la séquence de sortie,
- soit on observe une fonction de la sortie : comptage des transitions, comptage de 1 ou 0,...; le test est alors dit compact.

C) Les méthodes classiques au niveau sous-système

Les méthodes de test au niveau d'ensembles plus complexes consistent à [ROB,76]

- partitionner le système en blocs fonctionnels (ROM, RAM, ALU, chemin de données),
- établir un ordonnancement dans le test de ces blocs ("start-small", "start-big", "multiple-clue" [DEN ,68];la méthode de test la plus courante est la politique de boule de neige ou "start-small"; l'ordonnancement suppose l'étude des moyens d'accès à chacun de ces blocs (commandabilité, observabilité),
- définir les opérandes de test supposés être un ensemble de test complet pour chacun de ces blocs; ces opérandes sont déterminés par l'une quelconque des méthodes précédentes (du type à vecteurs prédéterminés); on rappelle dans le tableau suivant les blocs fonctionnels les plus classiques et les méthodes utilisées :

bloc	ROM	RAM	Chemin de données	ALU	Séquenceur
Méthode	-lecture -check-sum	-galloping -walking -ping-pong [ROB,78a]	- méthodes de propagation	méthodes de propagation	- identification d'automates - méthode algorithmique[ROB,78c]

I - 2. EXTENSIONS DE CES METHODES AUX MICROPROCESSEURS

A) Les hypothèses de défaillances dans les microprocesseurs

Les nouvelles technologies font apparaître de nouveaux types de pannes; ces pannes sont difficiles à définir parce qu'elles sont spécifiques de chaque technologie et demandent une analyse approfondie de la réalisation physique du circuit; elles doivent être ensuite modélisées pour permettre l'élaboration de nouvelles méthodes de test structurelles permettant leur détection.

Les hypothèses suivantes semblent crédibles [GAL,78 - LAN,78 - WAD,78] :

- collages étendus aux collages multiples
- collages transitoires
- court -circuits entre connexions voisines
- pannes modifiant les temps de propagation
- pannes d'interdépendance entre points de mémorisation
- transformation d'un circuit logique en circuit séquentiel.

B) Expériences de test de microprocesseurs

La plupart des auteurs proposent un découpage fonctionnel classique [FEE,77] et un test de type "start-small", la grande difficulté restant la détermination des vecteurs de test relatifs à un bloc.
On peut citer deux tentatives illustrant les efforts dans ce domaine : la première est fondée sur la détection de pannes avec hypothèses de défaillances, la deuxième est fondée sur la détection d'erreurs avec hypothèses d'erreurs fonctionnelles.

α) *Expérience 1* : sur hypothèses de pannes [CAI,76] : c'est une expérience industrielle (Thomson-Sescosem), au niveau fin de fabrication, pour laquelle le modèle de collage a été utilisé avec un certain succès; la méthode de test utilisée pour tester le microprocesseur -dérivé de l'AMD 2901- est une méthode déterministe.

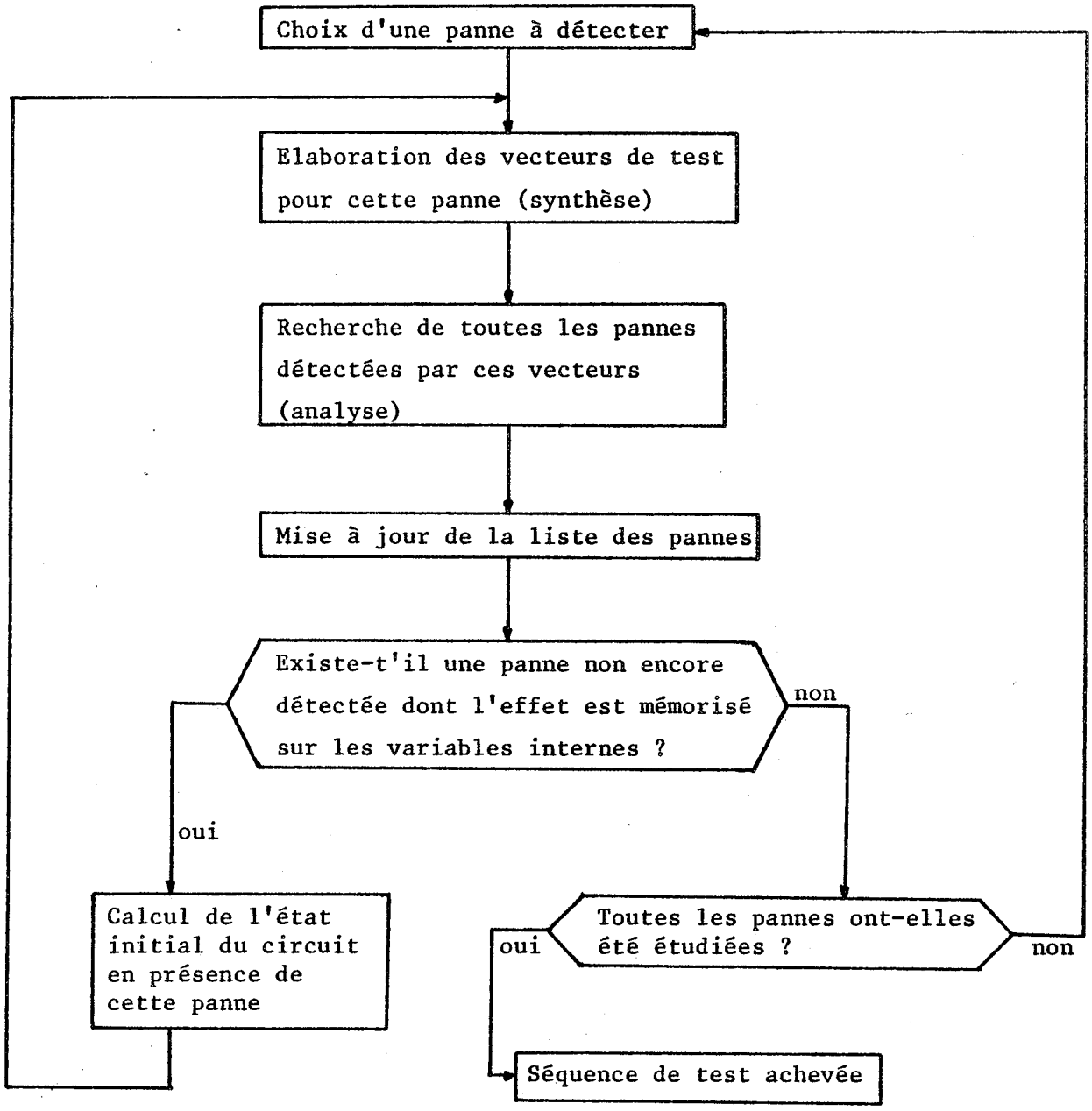
La connaissance du masque a permis de définir un schéma logique équivalent, conforme à la réalisation matérielle et d'établir une liste de pannes, à savoir :

- les collages sur les connexions
- les court-circuits entre connexions physiquement voisines.

Le modèle utilisé pour représenter le microprocesseur est la représentation combinatoire itérative de Roth [ROT,68]. La stratégie de test repose sur deux étapes :

- une étape de synthèse : élaboration des vecteurs de test pour un sous-ensemble de pannes,
- une étape d'analyse : recherche de toutes les pannes détectées par les vecteurs trouvés lors de la phase de synthèse.

L'algorithme de principe est le suivant [CAI,76] :



. Le programme implanté sur IRIS 80 et écrit en FORTRAN occupe 120 K-octets d'instructions en mémoire centrale. Sa programmation est telle que sa taille dépend du circuit à tester : pour le microprocesseur 80 K-octets supplémentaires sont nécessaires. Le programme laisse à l'utilisateur la possibilité d'intervenir lors du déroulement des opérations : il y a interaction programme-utilisateur (résultats partiels accessibles).

. Le microprocesseur comporte 528 connexions et 770 pannes sont considérées. La séquence de test est de 78 vecteurs détectant 94% des pannes et la durée de génération de cette séquence est de 18 mn.
L'algorithme de test est appliqué sur chaque bloc défini dans le microprocesseur :

Pannes sur le bloc	Temps de génération(s)	% pannes détectées dans ce bloc
entrées/sorties	108	91
partie contrôle	46	86
registres et étages de décalage	154	79
Interconnexions	163	87
UAL et étages de sélection	325	91

Une phase d'analyse sur l'ensemble des blocs permet de voir si les vecteurs trouvés ne détectent pas d'autres pannes : cette phase dure 190 s et donne les résultats suivants :

pannes d'E/S..... 95 %
pannes sur la partie contrôle..... 96 %
pannes sur les registres et étages de décalage..... 88 %
pannes de l'UAL et étages de sélection..... 91 %
pannes sur les interconnexions..... 99 %

Les causes d'échec sont dûes d'une part à l'initialisation et aux fonctionnements interdits, d'autre part à l'incapacité du programme de générer certaines séquences de test dans un temps raisonnable.

β) *Expérience 2* : sur hypothèses d'erreurs

La contribution d'Abraham [THA,78] consiste à définir un ensemble d'erreurs fonctionnelles et à chercher un ensemble de test complet par rapport à ces erreurs. L'intérêt de cette approche réside dans le souci de recherche d'une couverture par rapport à un ensemble d'erreurs. Cependant, elle met en évidence la difficulté de déterminer un ensemble "d'erreurs fonctionnelles" qui soit représentatif de la réalité physique.

Rappelons que l'auteur classe les instructions en deux grandes classes :

Classe A
- transfert d'information entre mémoire centrale et registres directement adressables
- instructions d'E/S, branchements, sauts

Classe B
- B1 . instruction UAL
- B2 . transfert d'information entre registres directement adressables
- B3 . transfert d'information entre registres directement adressables et registres indirectement adressables
- B4 . transfert d'information entre registres indirectement adressables

A chaque bloc fonctionnel est associé un ensemble d'erreurs fonctionnelles. Par exemple, trois types d'erreurs fonctionnelles peuvent survenir dans un multiplexeur :

E1 : aucune source n'est sélectionnée

E2 : une mauvaise source est sélectionnée

E3 : plusieurs sources sont sélectionnées simultanément

Un algorithme de test complet par rapport à chaque famille est alors donné.

C) Test d'un système à microprocesseur [GOB,78]

Cette étude est illustrée sur une carte à microprocesseur classique [ROB,78d], conçue pour le contrôle de systèmes de télécommunications.

α) *Structure générale (Figure 3.1)*

Cette carte à microprocesseur est constituée de :

- un bloc de commande comportant le microprocesseur (8080), le circuit de génération d'horloge (8224) et le bus driver bidirectionnel (8228);
- une mémoire programme : c'est une mémoire morte de 8 K octets, contenant le programme d'application;
- une mémoire de données : c'est une mémoire vive de 1 K octets;
- les bus : les signaux sur le bus d'adresses sont amplifiés par des circuits 8212 (registres d'interface sur le bus, bidirectionnels); les signaux sur le bus de données et le bus de contrôle sont amplifiés par des buffers bidirectionnels (8216).

Les trois bus peuvent être isolés du microprocesseur et recevoir des valeurs du monde extérieur, pendant le test.

- le bloc des entrées/sorties : les entrées/sorties sont gérées par le microprocesseur;
- un compteur programmable (8253) qui génère trois horloges : une horloge h_0 envoyée vers des cartes terminales, une horloge temps réel h_1 et une horloge h_2 pour les transmissions séries;
- un circuit de gestion des interruptions (8259) qui enregistre jusqu'à huit interruptions et gère la priorité et le masquage des interruptions sous contrôle du microprocesseur;
- des connecteurs de test : cette carte CPU comporte un connecteur supplémentaire où sont disponibles certains signaux particuliers (pour le test)

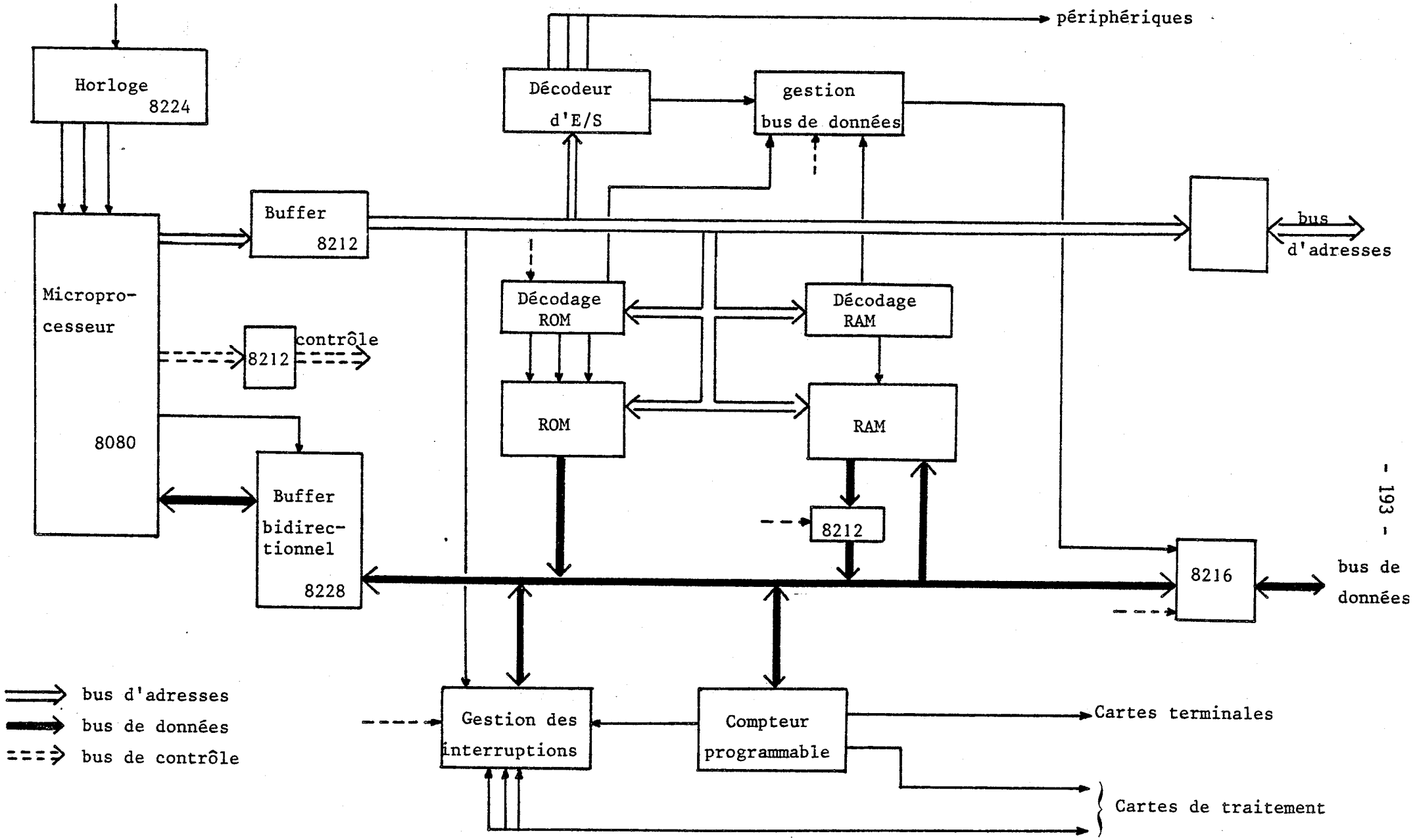


Figure 3.1. Structure de la carte à microprocesseur

β) *Stratégie de test*

. Structure générale de la procédure de test

La procédure comporte trois phases :

- dans la première phase, le microprocesseur et ses circuits esclaves (8224, 8228) sont isolés (signaux HOLD et BUSEN). Le test des autres circuits de la carte est géré par un outil de test externe;

- dans une seconde phase, le microprocesseur et ses circuits esclaves sont remis en service (suppression de HOLD et BUSEN); le signal de sélection (CS) de la mémoire programme est inactif (inhibition de la mémoire); le microprocesseur et ses circuits esclaves sont testés à travers les autres circuits de la carte, le test étant géré par l'extérieur;

- dans la troisième et dernière phase, la mémoire programme est remise en service; on fait alors exécuter le programme d'application qui permet de vérifier le programme d'application et la ROM sur laquelle il est implanté et de faire un test général de comportement de la carte.

La stratégie de test adoptée est la politique classique de "start-small".

Le test se déroule comme suit :

* test des fonctions vitales
  - alimentation : on teste les court-circuits entre les trois sources d'alimentation +12V, +5V, -5V,
  - initialisation et horloge
  - certains signaux de contrôle
  - bus d'adresses et de données : vérification des court-circuits entre lignes et des collages sur ces lignes.

* test des circuits de la carte à l'exception du bloc de commande

  Les méthodes de test utilisées pour ces circuits sont les méthodes de test classiques (§ I.1),
  - mémoire vive et son circuit de décodage,
  - mémoire programme et son circuit de décodage éventuellement; ce test peut être soit une lecture de la ROM (ROM de petite taille), soit une procédure de check-sum,

- test du compteur programmable : ce circuit génère 3 horloges qui sont les sorties de 3 compteurs; le test consiste à vérifier chacun de ces compteurs,
- test du circuit de gestion des interruptions; on vérifiera chacun des trois registres qui composent ce circuit : le registre de demande d'interruptions, le registre de masquage des interruptions et le circuit de file d'attente des interruptions,
- test du décodage des entrées/sorties.

* test du microprocesseur et de ses circuits esclaves
La procédure de test peut être l'une de celles exposées en I.1.

I - 3. TEST D'UN MICROPROCESSEUR A TRAVERS UNE APPLICATION

Une telle approche est fondée sur l'analyse du programme d'application [ROB,78b].
Le programme est représenté par un organigramme compacté où les noeuds correspondent à une ou plusieurs instructions, les arcs étant renseignés par des conditions de branchements.

Des points d'observation sont définis comme des noeuds particuliers du programme correspondant à une instruction exécutant une fonction observable : une instruction dont le résultat peut être saisi par le monde extérieur, par un moyen matériel ou qui est transmis au monde extérieur par des instructions de test rajoutées au programme d'application.

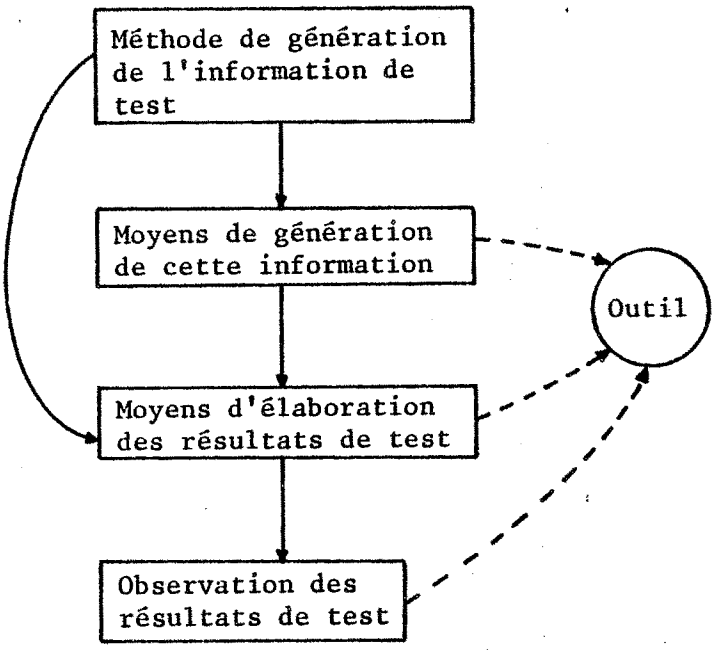
L'étude consiste à :

- déterminer des données permettant un passage à travers toutes les branches du programme (la simulation de cet environnement doit être possible),
- définir les fonctions du microprocesseur observées entre deux points d'observation successifs; il s'agit de réaliser une couverture complète des fonctions au cours du test,
- compléter ce test face à des hypothèses d'erreurs fonctionnelles ou structurelles; si les hypothèses d'erreurs ne peuvent être connues, le nombre d'activations exigées pour un circuit correspondra à la longueur d'un test aléatoire assurant une qualité de détection donnée [THE,78].

Cette approche permet d'avoir une activation "équilibrée" des différents blocs. Elle est présentée en détail en Section 2.
Une approche très restrictive de cette méthode a été proposée dans [KER,78] où la seule hypothèse d'erreur fonctionnelle considérée a été la distorsion du temps d'exécution des instructions; les erreurs affectant uniquement les données (non utilisées pour un branchement ultérieur) ne sont pas détectées.

II - MISE EN OEUVRE DU TEST ET OUTILS DE TEST

La mise en oeuvre du test comporte quatre étapes qui déterminent le type d'outil de test à utiliser.



II - 1. ANALYSE DES ETAPES DE LA MISE EN OEUVRE

A) Méthode de génération de l'information de test

Comme on l'a vu en I (§I.1), on a deux types de génération :

- l'information est générée aléatoirement au moment du test effectif,
- l'information de test est prédéterminée et appliquée chaque fois que l'on fait le test d'un circuit.

B) Moyens de génération et outils

α) *La génération aléatoire* (GI) au cours du test effectif implique un générateur pseudo-aléatoire, qui envoie des informations au microprocesseur sous test et à une référence.

β) *La génération prédéterminée* utilise l'une des méthodes (structurelle, fonctionnelle, par programme d'application) données en I.
Les moyens de génération peuvent être de l'un des types suivants :
- (G2) simulation structurelle du microprocesseur
- (G3) simulation fonctionnelle du microprocesseur
- (G4) référence "papier".

* La simulation structurelle : compte tenu de la difficulté de connaître la structure interne d'un microprocesseur, la simulation structurelle est difficile à mettre en oeuvre et induit généralement un outil de test sophistiqué.
Par contre, l'analyse fine de la structure qui est nécessaire (avec hypothèses de défaillances), permet souvent d'effectuer des tests paramétriques statiques et dynamiques.
L'outil de test comporte alors une mémoire de masse contenant l'information de test et les résultats justes fournis par la simulation.

* La simulation fonctionnelle : l'outil de test (générateur algorithmique) travaille sur une représentation fonctionnelle du microprocesseur: chacun des blocs du microprocesseur est décrit par un un algorithme qui est l'image de son fonctionnement.
Par exemple, le compteur programme est décrit par un algorithme d'incrémentation.

* La référence "papier" : l'utilisateur prédétermine à la fois l'information de test et les résultats justes correspondants, par l'une des méthodes citées en I; ces données sont stockées en mémoire de masse.

C) Les moyens d'élaboration des résultats de test

Les résultats justes sont :
- θ1 - soit issus d'une référence matérielle
- θ2 - soit fournis par une simulation, structurelle ou fonctionnelle
- θ3 - soit fournis par une analyse du microprocesseur et élaborés en même temps que l'information d'entrée.

Les moyens d'élaboration sont dépendants des moyens de génération de l'information de test.

* Résultats issus d'une référence matérielle :

  - l'information de test étant envoyée simultanément au microprocesseur sous test et à la référence physique, les résultats justes sont issus de cette référence matérielle.
    Ce moyen d'observation sera appliqué :
    - soit lorsqu'il y a génération aléatoire de l'information de test (G1),
    - soit éventuellement lorsque la génération est prédéterminée sur la machine papier (G4).

* Résultats fournis par une simulation : c'est bien entendu le cas où les données de test sont elles-mêmes générées par simulation; ces résultats sont :
  - soit stockés dans une mémoire de masse dans le cas d'une simulation structurelle (G2),
  - soit directement fournis par le générateur algorithmique dans le cas d'une simulation fonctionnelle (G3).

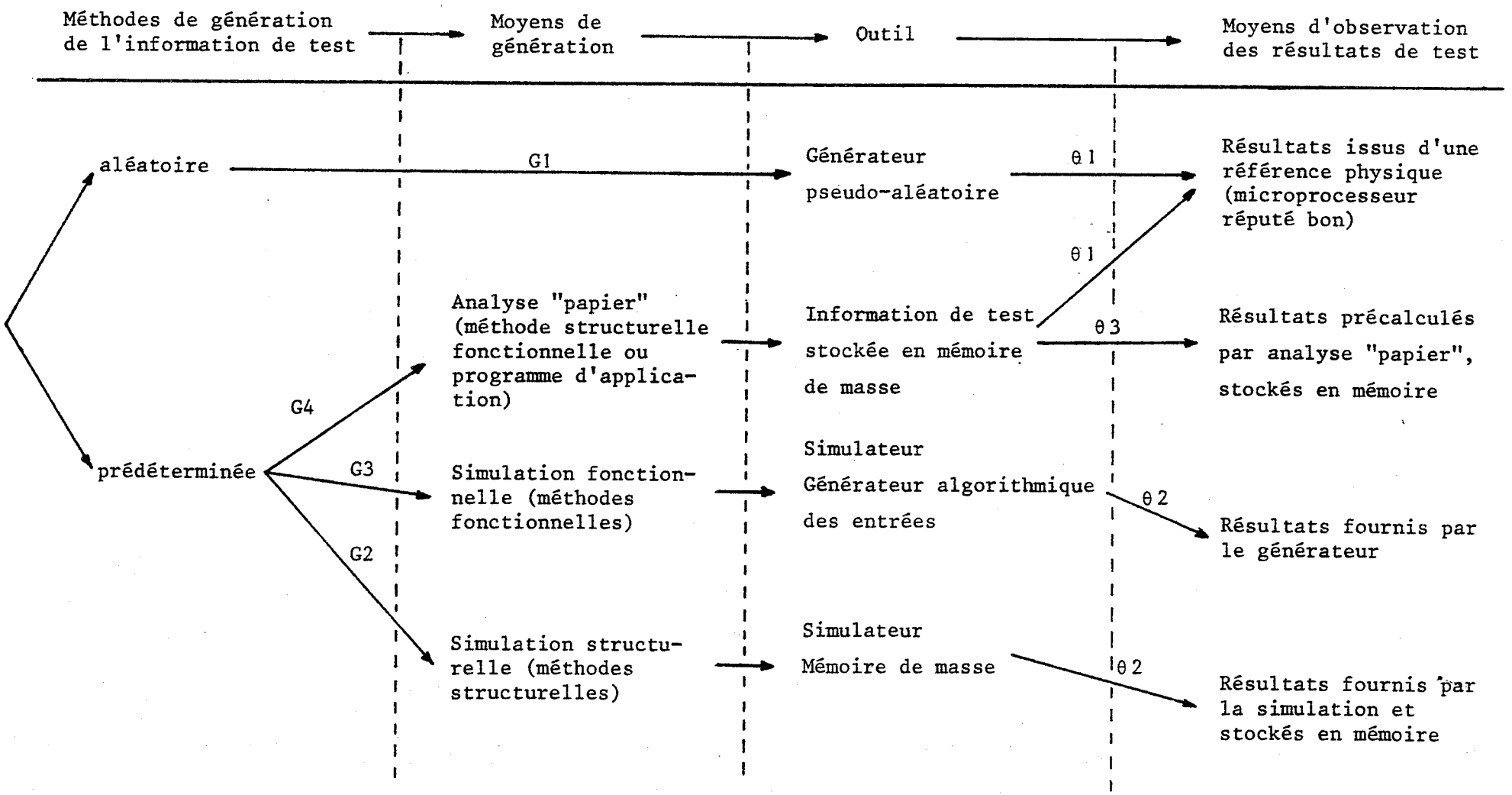
* Résultats fournis par une analyse "papier" : l'analyse du comportement du microprocesseur (structurelle ou fonctionnelle) fournit, en même temps que les données de test, les résultats attendus. Ces résultats sont alors enregistrés en mémoire.

D) Observation des résultats de test

Dans tous les cas on effectue une comparaison entre les résultats attendus et les résultats réels issus du microprocesseur sous test; ces résultats réels sont la transformation de l'information de test par le microprocesseur sous test.

E) Tableau récapitulatif

On peut noter que les combinaisons (G2 - O2, G3 - O2, G4 - O1) sont utilisées par des systèmes de test commerciaux, alors que les autres situations font plutôt l'objet d'outils de test in-situ (conçu par l'utilisateur luimême) ou à la demande.


Méthodes de génération de l'information de test
Moyens de génération
Outil
Moyens d'observation des résultats de test
aléatoire
G1
Générateur pseudo-aléatoire
θ1
Résultats issus d'une référence physique (microprocesseur réputé bon)
θ1
Analyse "papier" (méthode structurelle fonctionnelle ou programme d'application)
Information de test stockée en mémoire de masse
θ3
Résultats précalculés par analyse "papier", stockés en mémoire
G4
prédéterminée
G3
Simulation fonctionnelle (méthodes fonctionnelles)
Simulateur Générateur algorithmique des entrées
θ2
Résultats fournis par le générateur
G2
Simulation structurelle (méthodes structurelles)
Simulateur Mémoire de masse
θ2
Résultats fournis par la simulation et stockés en mémoire

II - 2. LES OUTILS DE TEST [MAR,77]

A) Mise en oeuvre dite "par comparaison" : chemin G1-01

Avantages : - observation des défauts en temps réel
- implantation facile (pas de stockage des données de test)
- test relativement complet (générateur très rapide)
- système de test peu coûteux

Inconvénients : - pas de renseignement sur les pannes détectées
- difficulté d'évaluer l'efficacité du test
- dépendance sur un microprocesseur "réputé bon"

Les outils de test : il n'existe pratiquement pas de testeur commercial utilisant une référence physique (on peut citer Micro-control M-10A); ce type d'outil est plutôt réalisé par l'utilisateur lui même.

B) Mise en oeuvre dite "en environnement réel" : chemin G4-03

Cette stratégie, employée par de petits utilisateurs, revient à tester le microprocesseur dans son environnement réel.

Avantages : - test dans des conditions de fonctionnement normal (test dynamique, prise en compte d'interruptions par exemple),
- équipement facile à implémenter et peu coûteux.

Inconvénients : - le chip n'est pas vérifié pour ses spécifications (il peut ne pas fonctionner correctement lorsqu'il est implémenté dans un autre système).

Outils de test : il n'existe pas réellement de systèmes de test commerciaux utilisant cette mise en oeuvre; cependant certains fabricants (Motorola, Intel, National Semiconductors) fournissent des analyseurs de chips conçus pour aider l'utilisateur à tester son chip dans des conditions de fonctionnement normal (fin de fabrication).

C) Mise en oeuvre dite "avec référence stockée" : chemin G2-02

La mémoire de masse où sont stockées les informations de test et les résultats attendus, est généralement complètée par une mémoire rapide : les données de la mémoire de masse sont transférées par paquets dans la mémoire rapide avant émission, à une fréquence rapide, vers le microprocesseur sous test.

Avantages : - mesures paramétriques possibles
- test relativement complet

Inconvénients : - peu d'indications sur les pannes
- mémoire de masse importante et temps de transfert longs vers la mémoire rapide
- programme non flexible
- mémoire rapide chère

Outils de test : un certain nombre de systèmes de test commerciaux utilisent cette stratégie; ce sont des systèmes très sophistiqués, qui réalisent des tests paramétriques statiques et dynamiques et des tests logiques.
On peut citer entre autres :

- Fairchild Systems Technology : Sentry 610 (un des plus populaires sur le marché).
- Tektronix S-3260 (bonnes mesures paramétriques dynamiques et caractérisation de circuits).
- E-H Research Laboratories 4500 (mesures paramétriques statiques excellentes et E/S bidirectionnelles temps réel).

D) Mise en oeuvre dite "par génération algorithmique" : chemin G3-02

Avantages : - test relativement flexible
- systèmes de test moins coûteux que les précédents

Inconvénients : - tests logiques uniquement.

Outils de test : il existe de nombreux systèmes de test commerciaux utilisant cette stratégie; ils sont moins complexes que les précédents et par conséquent moins chers. Ces outils sont en général caractérisés par des "cartes de personnalité", associées à chaque modèle de microprocesseur.

On peut citer entre autres :

- Macrodata MD-104, MD-154, MD-501
- Data-Test DT-400
- Fairchild Systems Tech. Sentry II

III - AIDE AU TEST AU COURS DE LA CONCEPTION

On peut considérer deux types d'aide à la conception :

- une aide minimale qui consiste en l'insertion de points de test : points d'accès et d'observation, dont le coût est évalué par rapport au nombre de points de sortie,
- une aide plus importante qui consiste en la réalisation de parties auto-testées, dont le coût s'évalue par rapport à la surface en silicium.

Nous allons passer en revue ces deux aspects.

III - 1. TESTABILITE ET POINTS DE TEST

Ce problème a été intensivement étudié (chapitre I) et repose sur les notions suivantes :

α) Analyse des flots de données entre les points d'accès du système.

β) Mesure de commandabilité, soit le maximum de la quantité d'information qu'un module peut recevoir dans un chemin de données.

γ) Mesure d'observabilité, soit le maximum de la quantité d'information que l'on peut avoir sur la sortie supposée inconnue du module, connaissant les valeurs d'entrée/sortie du chemin de données.

δ) Mesure de testabilité, c'est-à-dire la facilité d'assurer la maintenance (détection et localisation des pannes matérielles survenant dans le système).

L'étude de deux cartes à microprocesseurs, prévues respectivement pour

- un système à haute sécurité (avionique - CROUZET),
- un système à haute disponibilité (télécommunications - TELIC)

a permis de dégager quelques points critiques pour le test et la maintenance de ces cartes. Ces points devraient dont être pris en considération lors de la conception "future" de cartes à microprocesseurs intégrées. Le concepteur de cartes à microprocesseur devrait fournir les possibilités suivantes :

a) Bus d'adresses bi-directionnel : ce bus est généralement unidirectionnel; la bidirectionnalité du bus d'adresses permettrait, dans le mode test,

. de tester ce bus préalablement au test de la carte proprement dite,

. d'adresser directement la mémoire vive (RAM) et la mémoire programme (ROM) pour les tester indépendamment du microprocesseur.

b) Mise en haute impédance des boitiers connectés au bus de données, ceci afin :

. d'isoler le microprocesseur et plus particulièrement le bus driver bidirectionnel,

. d'inhiber le signal de sélection de la mémoire programme,

. d'inhiber le signal de sélection de la mémoire vive.

Cette mise en haute impédance permet :

. de tester le bus de données préalablement au test proprement dit de la carte,

. de tester "isolément" les différents boîtiers de la carte, indépendamment du microprocesseur : test plus rapide et plus complet,

. de pouvoir envoyer un programme de test depuis l'extérieur de la carte, spécifique aux modules testés.

c) Sortie de certains signaux de contrôle, permettant la vérification des "fonctions essentielles" du microprocesseur

- signal HOLD qui déconnecte le microprocesseur du reste du système,
- signal READY qui met le microprocesseur en état d'attente : ce signal permet le pas à pas dans le test,
- signal de sélection mémoire de la (P)ROM, qui permet d'observer la prise en compte de l'instruction,
- les signaux de lecture/écriture en mémoire vive, pour tester cette mémoire indépendamment du microprocesseur,
- un signal d'horloge du microprocesseur (signal de synchronisation par exemple) qui permet de vérifier l'horloge préalablement au test effectif.

III - 2. VERS UN MICROPROCESSEUR AUTO-TESTABLE

Deux principales stratégies peuvent être considérées :

A) Dans la première stratégie, le système à microprocesseur comporte un programme de diagnostic efficace, chargé partiellement en mémoire morte du système, partiellement en mémoire extérieure. Le programme de diagnostic résident est élaboré selon les méthodes discutées en I et exécuté sous un mode spécial du contrôle. Le problème concerne alors l'espace mémoire disponible pour ce programme, la qualité de ce programme de test et principalement le hardcore (comparateur).

B) La seconde stratégie consiste à inclure des facilités de test, tels que les codes détecteurs ou correcteurs d'erreurs, pour certaines pannes critiques ou pour l'ensemble du microprocesseur.
Cette stratégie repose sur les notions suivantes :

α) *Circuit auto-test simple* : soit un circuit protégé par un code détecteur d'erreurs; ce circuit est dit "auto-test" pour un ensemble donné de pannes si et seulement si, pour toute panne de cet ensemble, il existe au moins une entrée telle que la sortie correspondante n'appartienne pas au code; autrement dit, toute panne cause au moins une erreur qui sera détectée (mais elle peut en causer d'autres qui ne seront pas détectées).

β) *Circuit sûr* : un circuit protégé par un code détecteur d'erreur est dit "sûr" pour un ensemble donné de pannes si et seulement si, pour toute entrée, soit la sortie est correcte
soit elle n'appartient pas au code.

γ) *Circuit totalement auto-testable* : c'est un circuit qui est à la fois sûr et auto-test simple.

Si l'implémentation de solutions "sûres" n'est peut être pas très réaliste, la protection permise par une solution "auto-test" n'est par contre pas suffisante.

Notons deux propositions de solutions sûres qui ont été récemment proposées :

. La première, proposée par RENNELS, AVIZIENIS et ERCEGOVAC [REN,78], définit un ensemble de quatre circuits VLSI qui peuvent être rajoutés à un microprocesseur 16 bits et à des mémoires pour en faire un système auto vérifiable. Ces circuits sont :

- une interface mémoire détecteur et correcteur d'erreurs, utilisant les techniques suivantes : code de Hamming corrigeant les données mémoire, remplacement d'un bit défectueux par une réserve, code de parité sur le bus interne et détection de pannes internes,
- une interface bus programmable, qui peut être microprogrammée en vue de jouer le rôle d'un adaptateur de bus ou d'un contrôleur de bus. Les techniques utilisées sont de deux types : utilisation d'un code de parité pour protéger l'information mémoire et duplication avec comparaison pour le reste de la logique,
- un module d'entrées/sorties qui fournit un jeu de fonctions standard utilisable pour la majorité des applications. Les techniques de tolérance aux pannes utilisées sont le code de parité et la duplication avec comparaison,
- un module de base qui est le noyau du système, ce module :
  * détecte les pannes du microprocesseur en synchronisant et en comparant deux microprocesseurs en duplex,
  * collecte les indications d'erreurs fournies par lui-même et les autres modules,
  * déconnecte le système sur détection de panne permanente.

. La seconde, proposée par SEDMAK et LIEBERGOT [SED,78] définit un calculateur général à base de VLSI réalisable dans un avenir moins proche.

Les caractéristiques de ce calculateur vis à vis de la tolérance aux pannes sont :

* une détection de pannes
* une reprise après pannes
* et une localisation des défauts

} très élevées

Les principales techniques utilisées sont :

* une redondance interne utilisant de la logique complémentaire
* des codes correcteurs et détecteurs d'erreurs et de la redondance pour la logique inter-chips,

* des contrôleurs d'erreurs totalement auto-testables,
* une vérification pour les parties externes au chip : alimentation, horloge,
* un chip de traitement d'erreurs : collecte, tri et isolation des pannes sont assurés dans ce chip.

Ces circuits étant coûteux, ne peuvent être retenus que pour des applications où la sécurité prime.

C) Une approche intéressante est celle de RAINARD [RAI,79] qui propose un microprocesseur "périodiquement auto-testable", ce qui est un compromis entre les circuits auto- test et le test hors ligne périodique.
Cette approche consiste à :

α) partitionner le microprocesseur en blocs auto-test,

β) insérer un programme de diagnostic, chargé en mémoire morte, ou en mémoire vive externe, ayant les propriétés suivantes :
- il est complet par rapport à l'ensemble des pannes considérées,
- toute panne détectée par le programme de test est stoppée par les blocs auto-test : (par exemple, par des codes détecteurs d'erreurs), c'est-à-dire que, pour toute panne dans la classe de pannes considérée, le programme comporte au moins un vecteur (ou une séquence) de test tel que la sortie n'appartienne pas au code.

Cette approche est attrayante pour diverses raisons :
- le matériel supplémentaire requis est comparable à celui nécessité par une solution auto-test simple,
- la propriété de "sûreté" est fournie par le programme de diagnostic,
- les problèmes de hardcore et de stockage des résultats de test sont évités.

Cette solution est en particulier intéressante pour les systèmes à haute disponibilité (systèmes de télécommunications), puisqu'il faut prendre en compte le temps nécessaire au passage du programme de diagnostic.
Par contre, on peut noter que les pannes transitoires ne sont pas détectées.

SECTION 2 : LE TEST DE SYSTEMES A MICROPROCESSEUR PAR LEUR PROGRAMME D'APPLICATION

INTRODUCTION

On étudie le test d'un système microprocesseur à application spécifique : le programme exécuté par le système est déterminé par l'application et enregistré en mémoire morte (ROM). On cherche à tester ce système à travers son programme d'application sans développer de programme de test spécifique. C'est un test hors ligne dont les objectifs sont les suivants :

- il détecte les pannes du microprocesseur et de ses circuits esclaves, susceptibles de se manifester pendant l'exécution du programme d'application, pour une classe de pannes donnée,
- il vérifie la mémoire programme
- il permet de localiser le sous-ensemble d'instructions du programme manifestant un mauvais fonctionnement.

Les avantages d'une telle approche sont nombreux :

- c'est une méthode de test simple et globale (test de bon fonctionnement de l'ensemble du système), particulièrement bien adaptée au contexte de la maintenance préventive (vérification périodique du système) et de la maintenance curative;
- on ne teste que la partie du système -et en particulier, du microprocesseur- qui est réellement utilisée par l'application : un programme d'application utilise rarement la totalité du jeu d'instructions et l'ensemble des possibilités du système;
- cette méthode est utilisable en fin de fabrication par des fabricants de petites séries, qui ne peuvent développer ou acheter des systèmes de test coûteux tant en matériel qu'en logiciel.

De plus, il n'est pas nécessaire de prévoir des boitiers microprocesseur ou ROM désenfichables, ce qui est souvent le cas avec les méthodes de test classiques où la mémoire programme est remplacée par une mémoire de test et/ou le microprocesseur sous test remplacé par un microprocesseur "sain"; des boitiers désenfichables sont plus sensibles à un environnement perturbé.

Le type de test utilisé est un test fonctionnel; mais une bonne expérience en matière de test permet d'éviter l'impasse d'un test fonctionnel exhaustif; en effet on se propose de prendre en compte :

- des hypothèses de pannes matérielles lorsqu'on connaît la structure interne du microprocesseur ou de certains de ses constituants (registres, UAL),
- des hypothèses d'erreurs fonctionnelles réalistes lorsque la structure n'est pas connue de façon assez précise (partie contrôle en particulier).

On ne trouve pratiquement pas dans la littérature de propositions de méthodes de test de systèmes à microprocesseurs à travers leur application. On peut citer une étude de Kerntopf [KER,78] qui cherche à détecter à travers le programme d'application, la distorsion du temps d'exécution d'une séquence d'instructions.

I - MISE EN OEUVRE D'UN TEST PAR PROGRAMME D'APPLICATION

La mise en oeuvre d'un tel test implique certaines contraintes sur les accès au système, aussi bien en ce qui concerne les moyens de commande que les moyens d'observation.

Les moyens de test requis pourront être fournis par le système lui-même (à la conception) ou par un testeur simple, extérieur au système et connecté aux voies d'accès normales. Ces moyens sont les suivants :

- on doit pouvoir commander les entrées du système, c'est-à-dire envoyer des valeurs prédéterminées. Dans le cas d'un testeur, il y a donc simulation de l'environnement : les valeurs sont envoyées par le testeur à la place des périphériques et/ou des capteurs. Le testeur doit donc réagir aux commandes du système vers l'extérieur;
- on doit pouvoir observer les sorties du système et en déduire l'état de bon ou mauvais fonctionnement. Dans le cas où le système lui-même est le testeur, on aura un mode spécial de test, permettant de récupérer ces valeurs.

I - 1. COMMANDE ET OBSERVATION DU SYSTEME

Suivant la structure du système testé, la commande et l'observation seront plus ou moins faciles et/ou complètes.

. Soit un système simple, par exemple une carte comportant un microprocesseur et ses circuits annexes (horloge, buffer d'E/S, gestion d'interruptions,...), une mémoire morte ROM, une mémoire vive RAM et éventuellement munie d'un ADM (Figure 1);

- observation : en l'absence du module PREDECODAGE, toute adresse circulant sur le bus adresse A serait observable; dans le cas de la figure 1, seul l'adressage des périphériques est observable sur A. Toute donnée circulant sur le bus de données D est observable à l'extérieur (qu'elle soit émise de la ROM, de/vers la RAM ou vers les périphériques);

- commande : seuls le microprocesseur ou le module ADM peuvent émettre des valeurs sur le bus A, qui n'est pas commandable de l'extérieur (buffer unidirectionnel). Le bus D est commandable pour les données venant des périphériques vers le microprocesseur ou l'ADM; on pourrait inhiber la mémoire RAM (matériel de test supplémentaire) pour fournir directement au microprocesseur des valeurs données à chaque lecture RAM (nous verrons par la suite que ce n'est pas nécessaire).

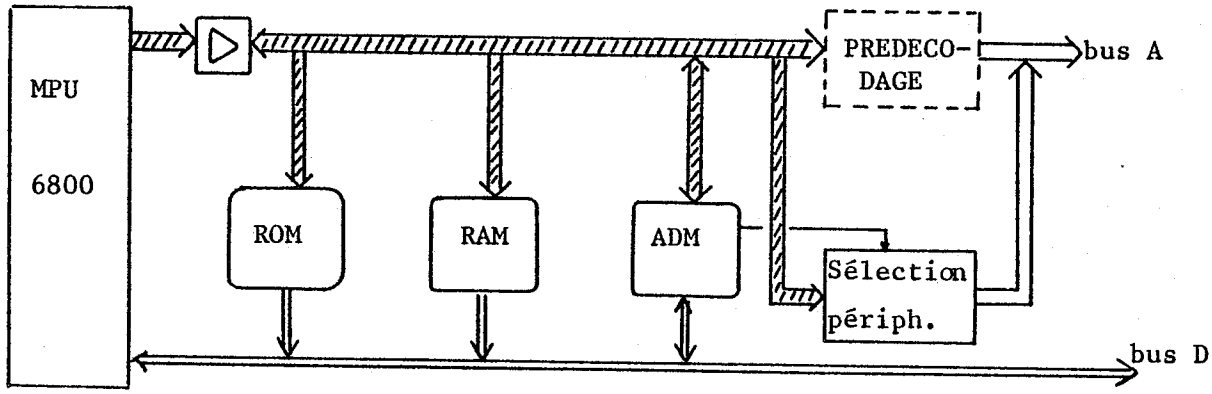


Figure 1. Système microprocesseur 6800

- Soit un système plus complexe, comportant des modules d'E/S (non transparents fonctionnellement, comme l'est l'ADM) : ACIA ou PIA pour un 6800, PPI (8255) ou PCI (8251) pour la série 80 d'Intel, comme en Figure 2;
  - observation : seuls l'adressage de la périphérie et les valeurs (commande et données) qui lui sont destinées sont observables;
  - commande : comme dans le cas précédent, les valeurs émises par la périphérie sont commandables.

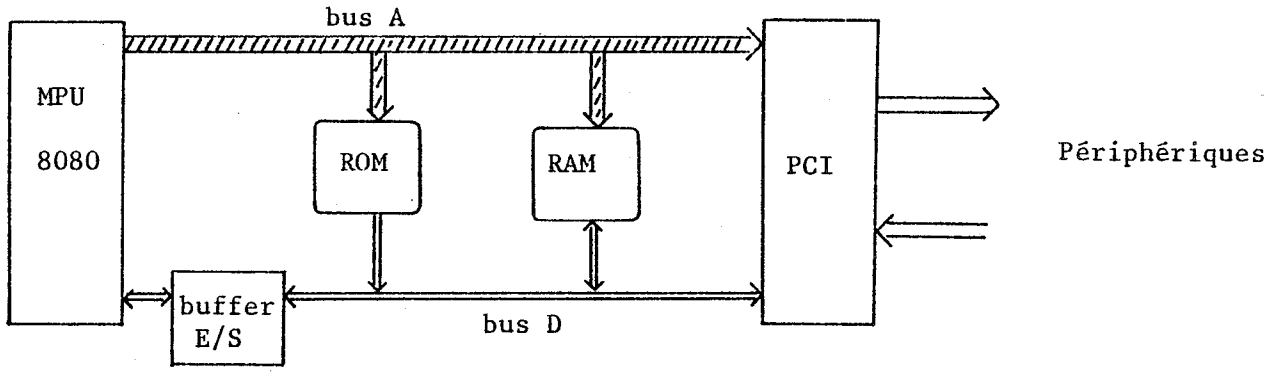


Figure 2. Système 8080 à modules d'E/S

. Selon la structure du système on peut donc se trouver dans l'une des situations suivantes :

α) *Pour le bus de données*

- le bus de données est observable en totalité; on peut alors observer toute instruction lue en ROM
  toute donnée lue ou écrite en RAM
  toute donnée émise vers l'extérieur
- le bus de données est partiellement observable, à savoir : seules les valeurs émises vers l'extérieur sont observables et uniquement après traitement par un module d'E/S;
- le bus de données est totalement commandable : ceci revient au test d'un microprocesseur "nu", ce qui n'est pas notre propos;
- le bus de données est partiellement commandable, par simulation de l'environnement du système par exemple.

β) *Pour le bus d'adresses*

- le bus d'adresse est totalement observable : on peut en particulier vérifier directement le séquencement du programme d'application, par observation de l'adresse de l'instruction lue en ROM;

- le bus d'adresse est partiellement observable : on peut en général vérifier uniquement l'adressage de l'environnement;
- le bus d'adresse est dans la majorité des cas non commandable.

γ) *Exemple pratique* : l'exemple pratique sur lequel est illustrée la suite de cette étude est un cas "moyen" dans lequel
- le bus de données est totalement observable et partiellement commandable,
- le bus d'adresses est partiellement observable et non commandable.

L'étude proposée est néanmoins applicable quels que soient les moyens de commande et d'observation.

I - 2. SIMULATION DE L'ENVIRONNEMENT ET ANALYSE DES RESULTATS

. La simulation de l'environnement peut être assurée par un testeur ou le système lui-même; l'étude proposée nous amène à définir des valeurs d'entrée permettant d'exécuter un certain nombre de fois les différents chemins du programme d'application, et ceci de façon déterministe.

On peut donc définir une "pile" de valeurs que le testeur devra émettre vers le système sous test;la synchronisation de l'émission de ces valeurs sera assurée à l'aide de signaux que le système émet normalement vers sa périphérie.

. De même, les valeurs de sortie seront observées de façon déterministe (en cas de bon fonctionnement); il suffit alors que le testeur soit prévenu de la présence d'une valeur significative sur le bus de données (par V_{MA} et R/W par exemple, dans un système 6800) et qu'il stocke cette valeur dans une pile.

. Pour simplifier l'analyse des résultats et réduire la place mémoire (taille de la pile des résultats), il n'est pas souhaitable de prélever les instructions lues en ROM, bien qu'elles soient observables sur le bus de données; par contre, il est intéressant d'implanter sur le système testé, un module de reconnaissance d'adresses RAM : les données lues et écrites en RAM seront observées, pour s'assurer que le programme d'application travaille avec des entrées justes et fournit des résultats justes.

La méthode de test proposée est donc simple à mettre en oeuvre et un même testeur peut être utilisé pour tout système microprocesseur de la même gamme. Il est même facile de concevoir un testeur général, utilisable avec plusieurs gammes de microprocesseurs.

I - 3. PRINCIPE GENERAL DU TEST

Contrairement au cas général, on ne cherche pas à définir un programme de test : le programme d'application est le programme de test et le concepteur du test doit :

- choisir les chemins de son programme d'application qu'il désire exécuter,
- déterminer les valeurs d'entrée du programme
- calculer le nombre d'activations de chaque chemin considéré

de manière à garantir une certaine efficacité de test.

L'ensemble du système peut être décomposé en

partie contrôle : ROM, partie contrôle du microprocesseur
+
partie opérative : partie opérative du microprocesseur, RAM et circuits annexes

En fait, on ne cherchera pas à tester la mémoire vive RAM à travers le programme d'application : ce serait extrêmement long.

α) *Test de la partie contrôle* : la partie contrôle du microprocesseur est non explicite, c'est-à-dire qu'on n'en connaît pas la structure. Elle n'est donc pas directement observable sur les voies d'accès du système : elle est testée par ses actions à travers la partie opérative [ROB,78c]. Sa structure n'étant pas connue, on travaille sur des hypothèses d'erreurs fonctionnelles
- erreurs de séquencement implicite ou explicite,
- mauvaise exécution d'une ou plusieurs instructions,
- mauvais positionnement des indicateurs.

La mémoire ROM peut être testée par lecture complète de ses instructions ; ceci n'étant pas envisagé pour des raisons de temps et de place mémoire, elle sera également observée à travers la partie opérative.

Le test de la partie contrôle consistera donc à :
- exécuter chaque instruction du programme d'application : test de la ROM et de l'exécution des instructions,
- effectuer tous les branchements possibles : test du séquencement et des indicateurs.

β) *Le test de la partie opérative*

La partie opérative est constituée de :
- éléments de mémorisation : registres,...
- opérateurs : UAL, circuit de décalage,..
- bus de données
- bus d'adresses et circuit d'adressage mémoire.

On peut choisir entre deux types de méthodes de test pour vérifier les constituants de la partie opérative :
- une méthode de test aléatoire ou pseudo aléatoire
- une méthode avec génération déterministe des vecteurs de test.

. Le principe du test aléatoire est le suivant [THE,78] : des entrées aléatoires sont envoyées au circuit à tester et à un circuit sain (matériel ou simulé); les résultats issus des deux circuits sont comparés.
Il a été montré que le nombre d'entrée L à envoyer au circuit pour garantir une qualité de détection donnée est fourni par la formule :

$$L = 2^n \frac{\text{Log } Q_D}{\sigma}$$

où n est le nombre d'entrées du circuit considéré,
Q_D est la qualité de détection ($Q_D = 10^{-3}$ indique qu'une panne sur mille n'est pas détectée),
σ est la surface de détection : $\sigma = \frac{P_D}{2^n}$ où P_D est la probabilité de détection de la panne la plus difficile à détecter; pour des circuits combinatoires $\sigma \geq 1$, pour des circuits séquentiels σ peut être très inférieur à 1.

. le principe du test déterministe est de calculer des vecteurs de test sur analyse du circuit avec hypothèses de pannes. On pourra considérer :

- soit des hypothèses d'erreurs matérielles, lorsque la structure du circuit est bien connue. Par exemple, pour un registre on fait les hymothèses suivantes :
  - * un ou plusieurs bits sont collés à la valeur 0 ou 1
  - * un ou plusieurs bits n'effectuent pas les transitions $0 \rightarrow 1$ ou $1 \rightarrow 0$,
  - * il y a couplage de deux bits i et j : la transition de la valeur x à la valeur y du bit i change l'état du bit j (de x à y ou de y à x) où $x = \overline{y}$,
- soit des hypothèses d'erreurs fonctionnelles lorsque la structure fine est non connue; par exemple, pour un décodeur on fait les hypothèses suivantes :

  Pour une adresse de destination :

  E1 : aucune destination n'est sélectionnée

  E2 : une mauvaise destination est sélectionnée

  E3 : plusieurs destinations sont sélectionnées simultanément.

. L'objectif du test de la partie opérative est de garantir le bon fonctionnement de chacun de ses éléments et d'évaluer l'efficacité du test par rapport aux hypothèses de pannes considérées. Cette efficacité sera mesurée par le taux de couverture, c'est-à-dire le pourcentage de pannes qui seront détectées par ce test, dans la classe de pannes considérées.

II - MODÉLISATION DU PROGRAMME D'APPLICATION

L'objectif est de donner au programme d'application une représentation qui soit directement utilisable pour le test.
Ceci nous amène à définir trois notions importantes :

. La notion de points de commande et d'observation qui seront les points du programme d'application où l'on pourra respectivement entrer des valeurs choisies et sortir des résultats significatifs;

. La notion de séquence qui est définie intuitivement comme le champ de test à chaque point d'observation;

. La notion de phase qui est définie intuitivement comme la plus "petite quantité" d'instructions significative pour le test.

II - 1. POINTS D'OBSERVATION - POINTS DE COMMANDE - PHASES

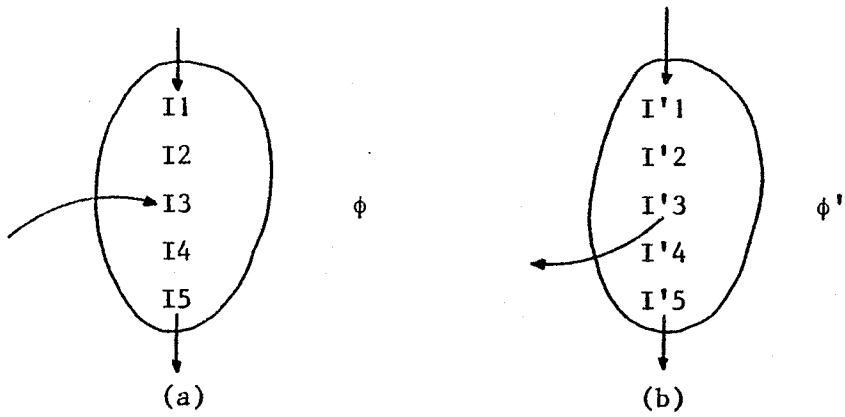
On définit :

. Un point d'observation comme une instruction telle que le résultat de son exécution est observable à la sortie du système.
Les points d'observation pourront être :
- les instructions de chargement en mémoire,
- les instructions de sortie
et seront fonction des moyens d'observation propres au système considéré (bus de données, bus d'adresse,...).

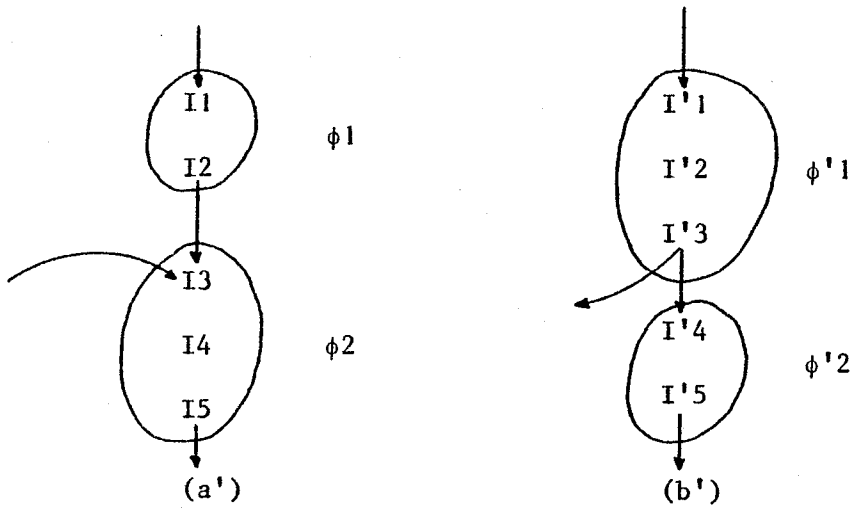
. Un point de commande comme une instruction qui acquiert des données commandables, c'est-à-dire des données dont le testeur est maître; ces données constituent donc les opérandes de test.
Les points de commande pourront être :
- les instructions d'entrée explicites
- les instructions de lecture en mémoire vive à une adresse où le programme n'a pas écrit (valeurs d'initialisation du programme, entrées implicites faites par ADM ou par un autre microprocesseur); le testeur pourra donc écrire des valeurs déterminées à ces emplacements mémoire.

. Une phase comme une suite ordonnée d'instructions telle que :
- un point d'observation est toujours la dernière instruction d'une phase,
- un point de commande est toujours la première instruction d'une phase,
- on ne peut entrer dans (ou sortir de) une phase que par sa première (ou sa dernière) instruction.

Par exemple :



Les situations (a) et (b) sont interdites et doivent être modélisées respectivement par (a') et (b').



Ces situations sont généralement dûes aux instructions de branchement :

- l'instruction de branchement est la dernière instruction d'une phase,
- l'instruction adressée par le branchement est la première instruction d'une phase.

. Une phase est dite observable lorsque sa dernière instruction est un point d'observation.
Une phase est dite commandable lorsque sa première instruction est un point de commande.

II - 2. MODELISATION DU PROGRAMME D'APPLICATION : GRAPHE DE CONTROLE

Le programme d'application est représenté par un graphe orienté où

- un noeud est une phase
- les arcs représentent les séquencements possibles entre phases.

C'est un graphe étiqueté : les arcs sont renseignés par les prédicats de séquencement.
Un graphe modélisant un programme d'application possède une phase initiale unique et une ou plusieurs phases finales; un chemin de la phase initiale à une phase finale représente une exécution élémentaire du programme; le retour d'une phase finale à la phase initiale permet de réexécuter le programme d'application (par le même chemin ou un autre chemin).
Ce graphe peut être directement obtenu à partir du programme d'application : on peut noter que ce passage est facilité lorsqu'une méthodologie de conception descendante du système à microprocesseur est utilisée [ZAC,77].

II - 3. EXEMPLE

L'exemple étudié est une système à microprocesseurs qui traite les impulsions de numérotation dans un centre de commutation téléphonique (CNET) [ROU,77].

a) *Définition générale et structure* (Figure 3)
Le système reçoit un message : numéro de voie, information de changements d'états; il effectue la transformation des changements d'états en chiffres et émet un message de sortie : numéro de voie, chiffre enregistré. Les messages sont échangés avec l'extérieur par l'intermédiaire de registres tampons.

La structure comporte trois unités centrales à microprocesseurs : chaque UC possède ses propres bus (données, adresses, contrôle) donnant accès à la mémoire programme MP. Les organes communs (ou resscurces communes) sont les E/S et les mémoires de travail MT, gérées par un allocateur.

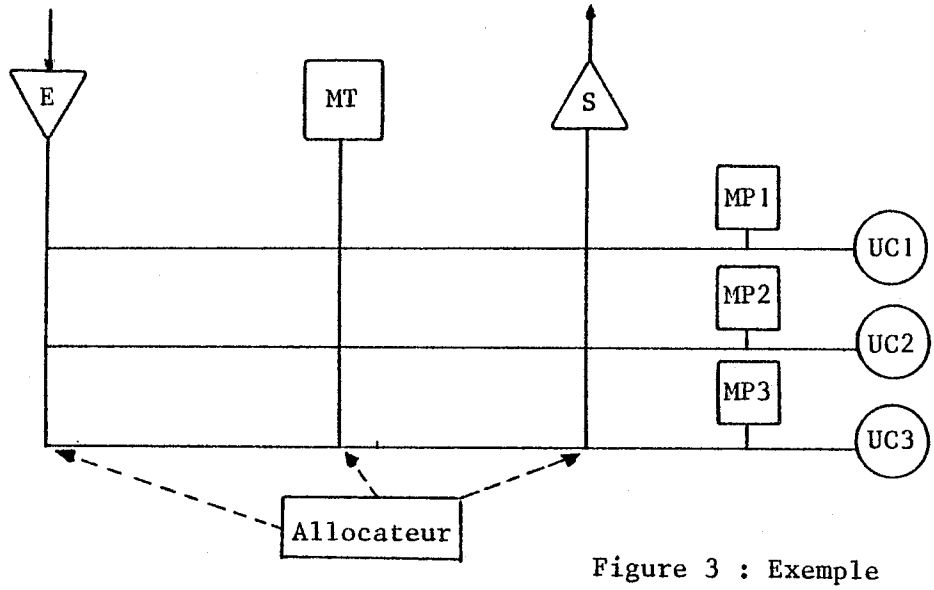


Figure 3 : Exemple

α) *L'unité centrale* (Figure 4)

On s'intéresse plus particulièrement à une unité centrale du système. Une unité centrale constitue une carte logique et comprend :

- un microprocesseur (Zilog Z80),
- les circuits nécessaires à son fonctionnement : horloge, amplificateur de bus, circuits de gestion des interruptions,...
- des circuits spécifiques de l'application : lecture de l'heure,..

β) *Le programme d'application*

Les programmes des trois processeurs sont identiques; chaque processeur doit :

- lire un message, libérer le coupleur
- analyser le numéro de voie et initialiser ou complèter la zone mémoire correspondante
- contrôler les temporisations pour déceler l'interchiffre
- présenter en sortie un message contenant le numéro de voie et le chiffre traduit.

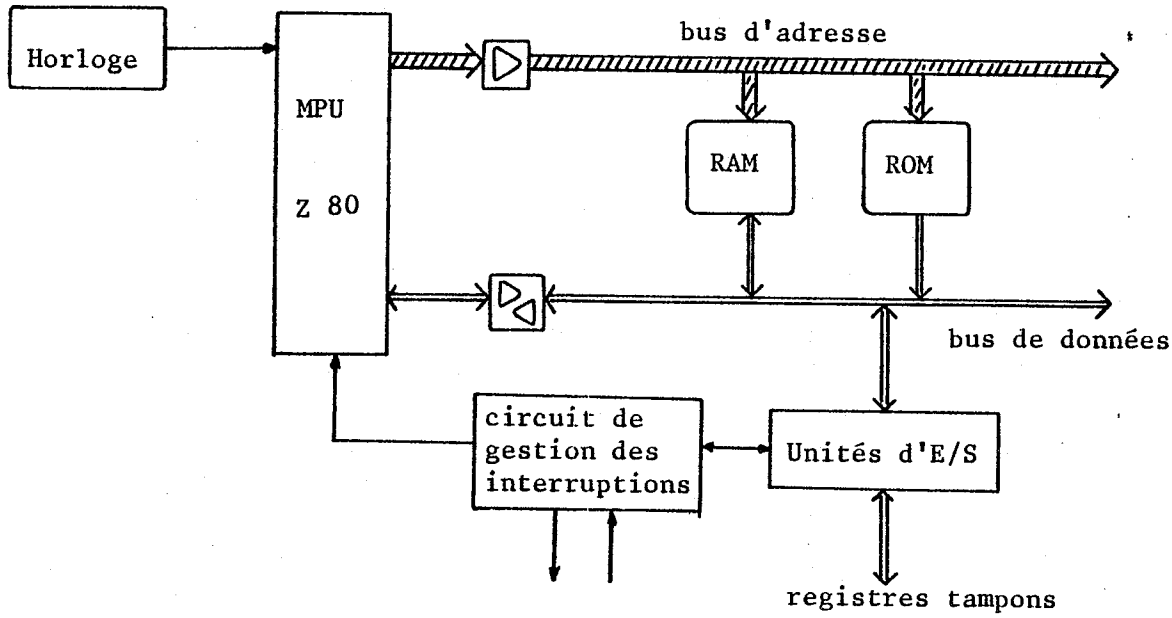
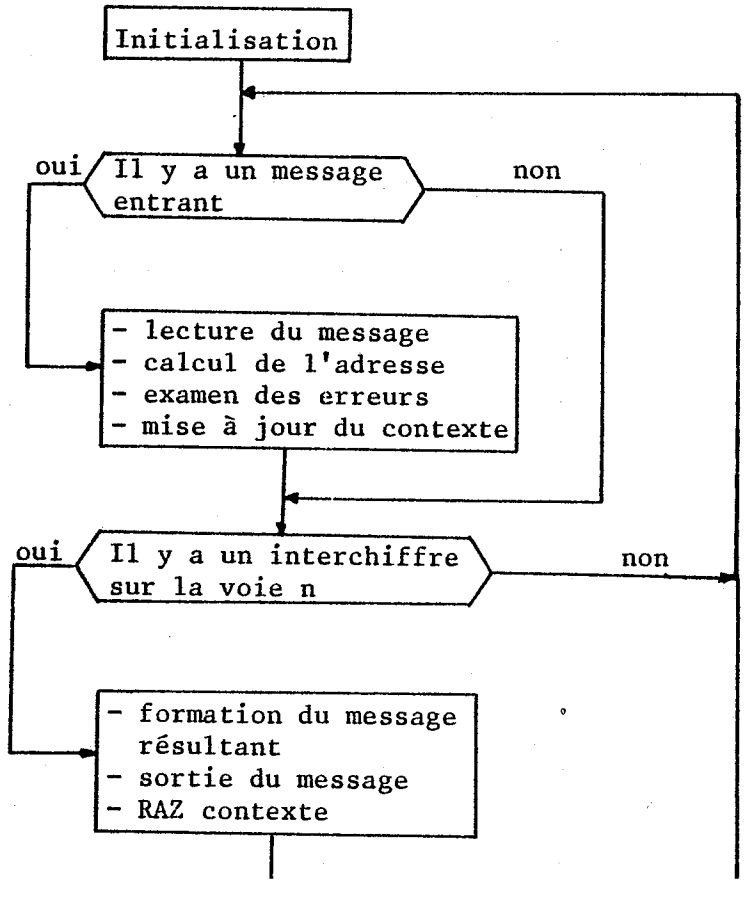


Figure 4 : l'unité centrale

Le programme répond à l'organigramme général suivant :



On considère une partie de ce programme d'application qui est la préparation de l'appel mémoire et la demande d'accès au bus : le programme calcule l'adresse mémoire tampon où sera stocké le message de sortie, en fonction du message d'entrée mémorisé en RAM locale à (ADOC1,ADOC2).

calcul de l'adresse	I1	LD A, (ADOC2).....	commandable
	I2	RRCA	
	I3	AND n1	
	I4	OR n2	
	I5	LD H,A	
	I6	LD A,(ADOC2).....	commandable
	I7	AND n3	
	I8	LD B,A	
	I9	LD A,(ADOC1).....	commandable
	I10	AND n4	
	I11	OR B	
	I12	RRCA	
	I13	LD L,A	
	I14	LD (ADAT), HL.....	observable
	I15	LD IX, (ADAT)	
détermination du bloc mémoire	I16	LD A, (ADOC2).....	commandable
	I17	AND n5	
	I18	JP NZ, PMTB.....	branchement 1
	I19	LD IY, DABMA	
	I20	JP PRANT.....	branchement 2
	I21	PMTB LB IY, DABMB.....	adresse de branchement 1
	I22	LD A, n6	
	I23	LD (AMMT), A.....	observable
demande d'accès bus	I24	PRANT LD A, (IY+0).....	adresse de branchement 2
	I25	AND n7	
	I26	JP NZ, PRANT.....	branchement 2

(ADOC1), (ADOC2), sont des variables d'entrée commandables (message).

Le graphe de contrôle modélisant cette partie du programme est donc le suvant :

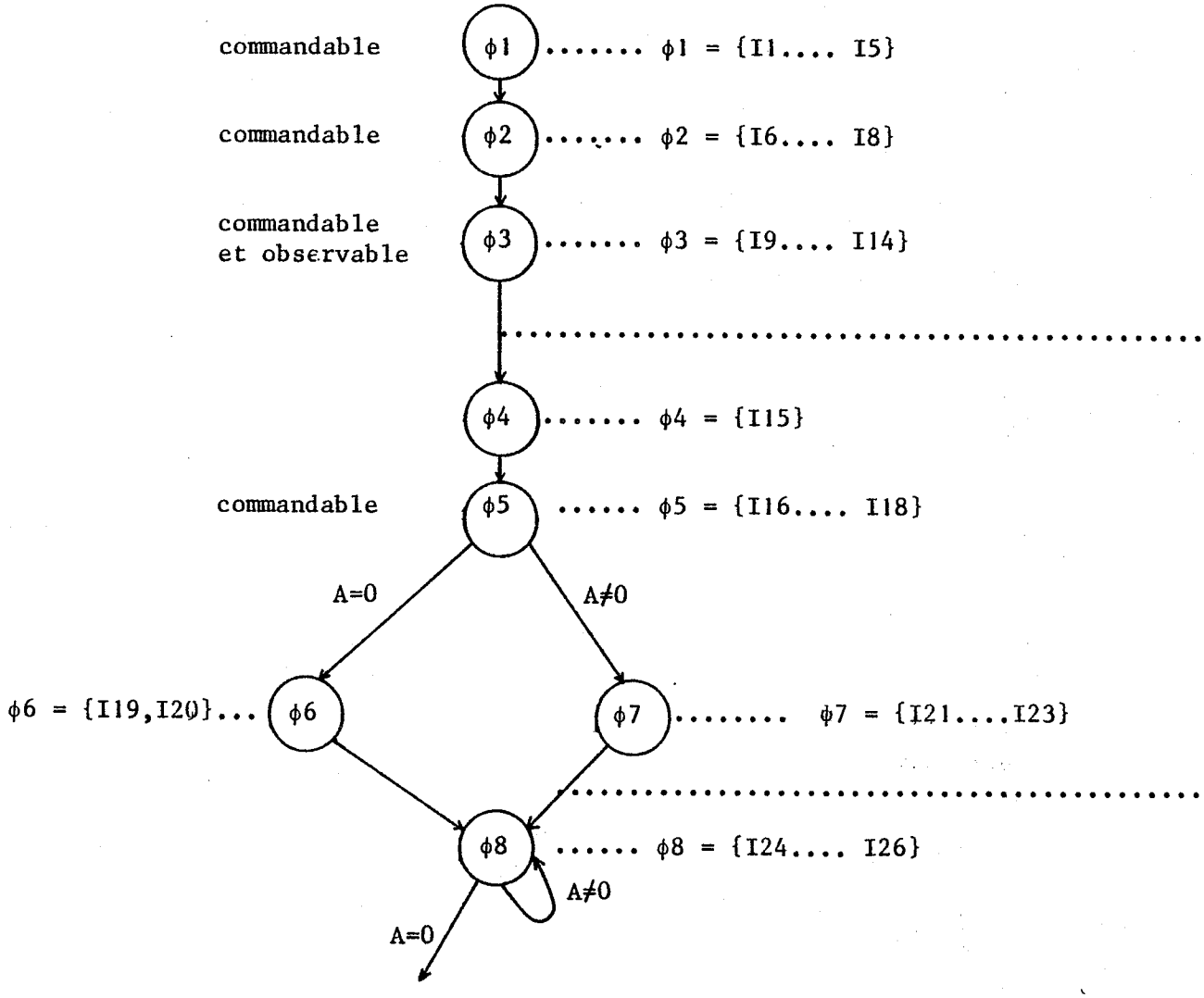


Figure 5 : Modélisation du programme d'application

II - 4. GRAPHE DE DEPENDANCE

Cette notion a été définie en [BEL,77] dans le but d'étudier la propagation dans une phase d'une ou plusieurs variables erronées et d'évaluer la contamination dûe à ces variables.
Ce graphe est utilisé ici pour étudier l'observabilité et la commandabilité d'un élément matériel à travers d'autres éléments imposés par le programme d'application.

α) *Graphe de dépendance d'une instruction*

Le graphe indique la dépendance entre les variables d'entrée utilisées par l'instruction et la variable finale qu'elle définit, ainsi que les fonctions et opérateurs activés implicitement ou explicitement lors de l'exécution de cette instruction.

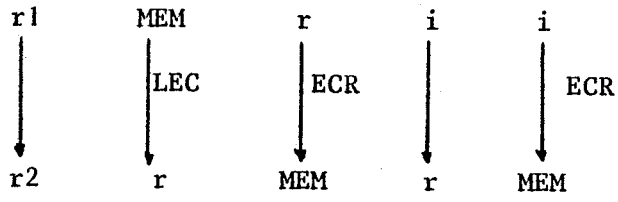
- Les variables sont de manière générale des organes de mémorisation (registres, mémoire) ou l'extérieur (E/S);

- Les opérateurs sont les organes de traitement (UAL, circuits de décalage, circuit de calcul d'adresse,...); ils ne seront pas en fait représentés en tant que tels sur le graphe de dépendance puisqu'ils sont implicitement nommés par leurs fonctions;

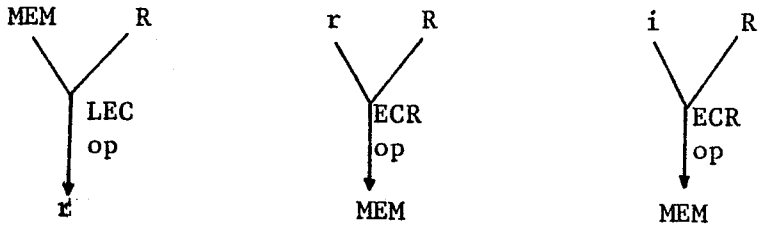
- Les fonctions sont toutes les fonctions implicites et explicites exécutées au cours de l'instruction. Cependant, certaines fonctions simples (transfert de registre à registre, lecture de l'instruction) ou répétitives (incrémentation du compteur ordinal) pourront ne pas être représentées sur le graphe pour des raisons de simplification.
  En particulier, le chargement des indicateurs(de signe, de valeur nulle,...) ne sera représenté que si l'indicateur est effectivement utilisé (branchement suivant la valeur de cet indicateur).
  Vis à vis du graphe de dépendance, on peut classer les instructions en 3 groupes :
  - le groupe de transfert de données
  - le groupe arithmétique et logique
  - le groupe des branchements.

* Groupe de transfert de données : ce groupe comprend toutes les instructions de transfert de données entre registres, mémoire et registres, extérieur et mémoires, les instructions d'E/S ainsi que les instructions de manipulation de pile (sauvegarde et restauration de contexte).

Le graphe de dépendance est de l'un des type suivants :



instructions simples de transfert



instructions avec calcul d'adresse ou manipulation de pile

où MEM est la mémoire vive ou un mot venant des périphériques

r est un registre interne (8 bits) ou un couple de registres (16 bits)

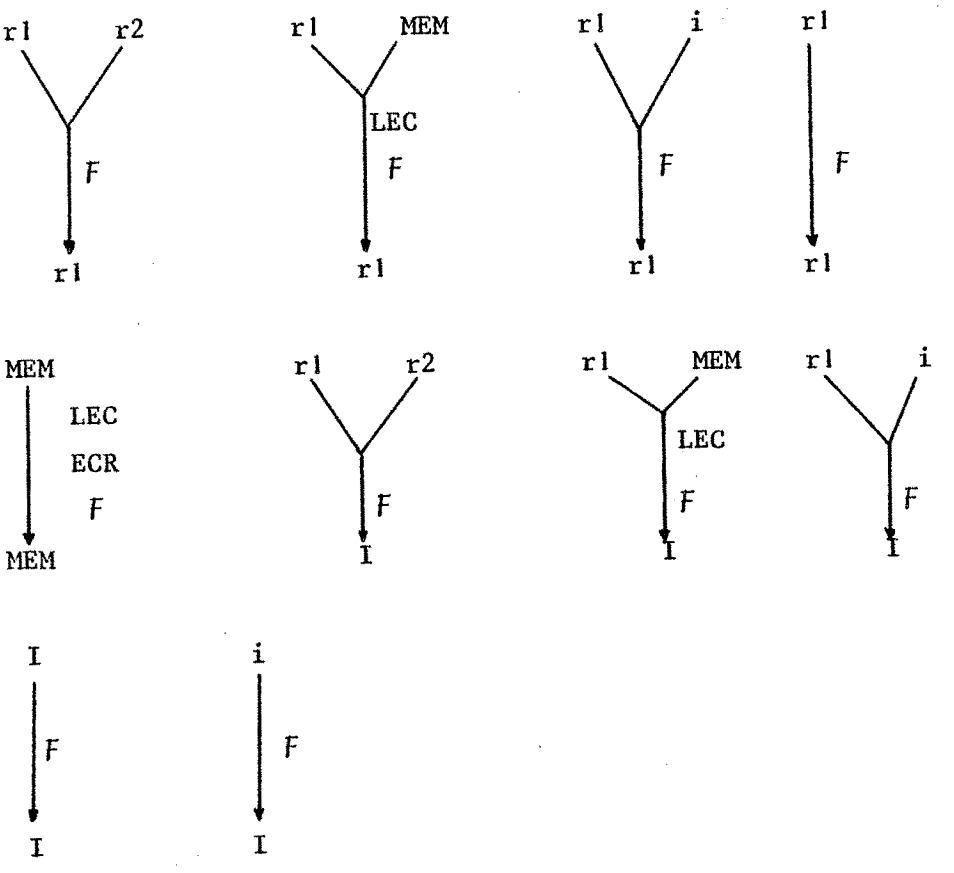
i est une valeur immédiate

R est un registre (ou une paire) utilisé implicitement par le mode d'adressage; dans le 8080, R sera généralement le compteur programme SP ou la paire de registre HL servant au calcul d'adresse.

op représente les opérateurs utilisés implicitement par l'instruction; par exemple unité arithmétique spécialisée dans le calcul d'adresses

LEC, ECR sont respectivement les fonctions de lecture écriture mémoire.

* Groupe arithmétique et logique : ce groupe comprend toutes les instructions qui effectuent une opération arithmétique ou logique sur des données venant de la mémoire ou des registres.
Le graphe de dépendance est de l'un des types suivants :

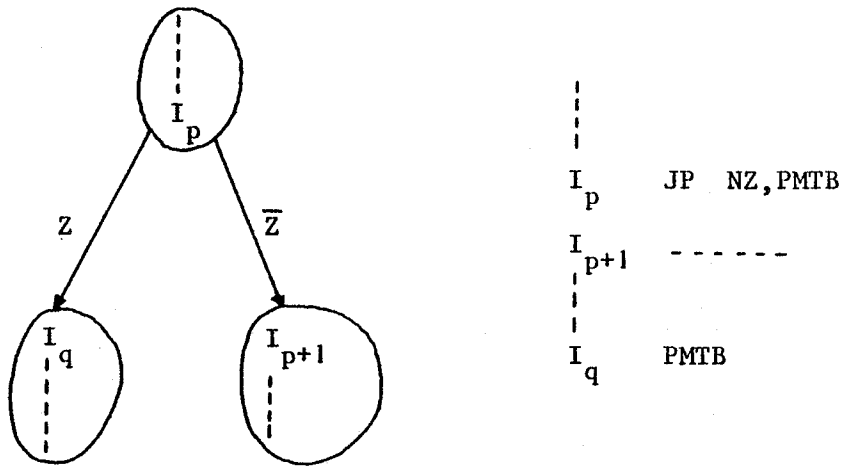


où F est l'opération arithmétique ou logique exécutée par l'instruction
I est un indicateur

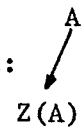
* Groupe de branchements : ce groupe comporte toutes les instructions de saut conditionnel et inconditionnel, d'appel sous-routines et de retour.
Les instructions de ce groupe sont considérées comme des instuctions vides vis à vis du graphe de dépendance, puisqu'elles ne concernent que le contrôle (séquencement).
Les instructions de branchement inconditionnel, d'appel sous-routine et de retour sont donc complètement transparentes.

Les instructions de branchement conditionnel -branchement sur indicateur- explicitent le calcul de l'indicateur conditionnant le branchement; cet indicateur est porté sur les transitions qui suivent cette instruction de branchement.

Exemple :



Graphe de dépendance:



β) *Graphe de dépendance d'une phase*

Le graphe de dépendance est déduit par concaténation des graphes de dépendance des instructions constituant la phase.

Chaque phase est alors caractérisée par :

- des variables
- les fonctions utilisées par la phase.

Les variables d'entrée sont les variables nécessaires à l'exécution de la phase, avant modification par les opérateurs : ce sont les racines du graphe (aucun arc n'arrive sur ces variables).

Les variables de sortie sont les variables modifiées par la phase qui contiennent les résultats de la procédure associée à cette phase : ce sont les feuilles du graphe (aucun arc ne sort de ces variables).

Les variables de travail sont toutes les variables utilisées par la phase, autres que les variables d'entrée et de sortie.

γ) *Exemple* : La figure 6 donne les graphes de dépendance des phases 1, 2, 3, 4, 5, 7 de l'exemple donné au §2.3.

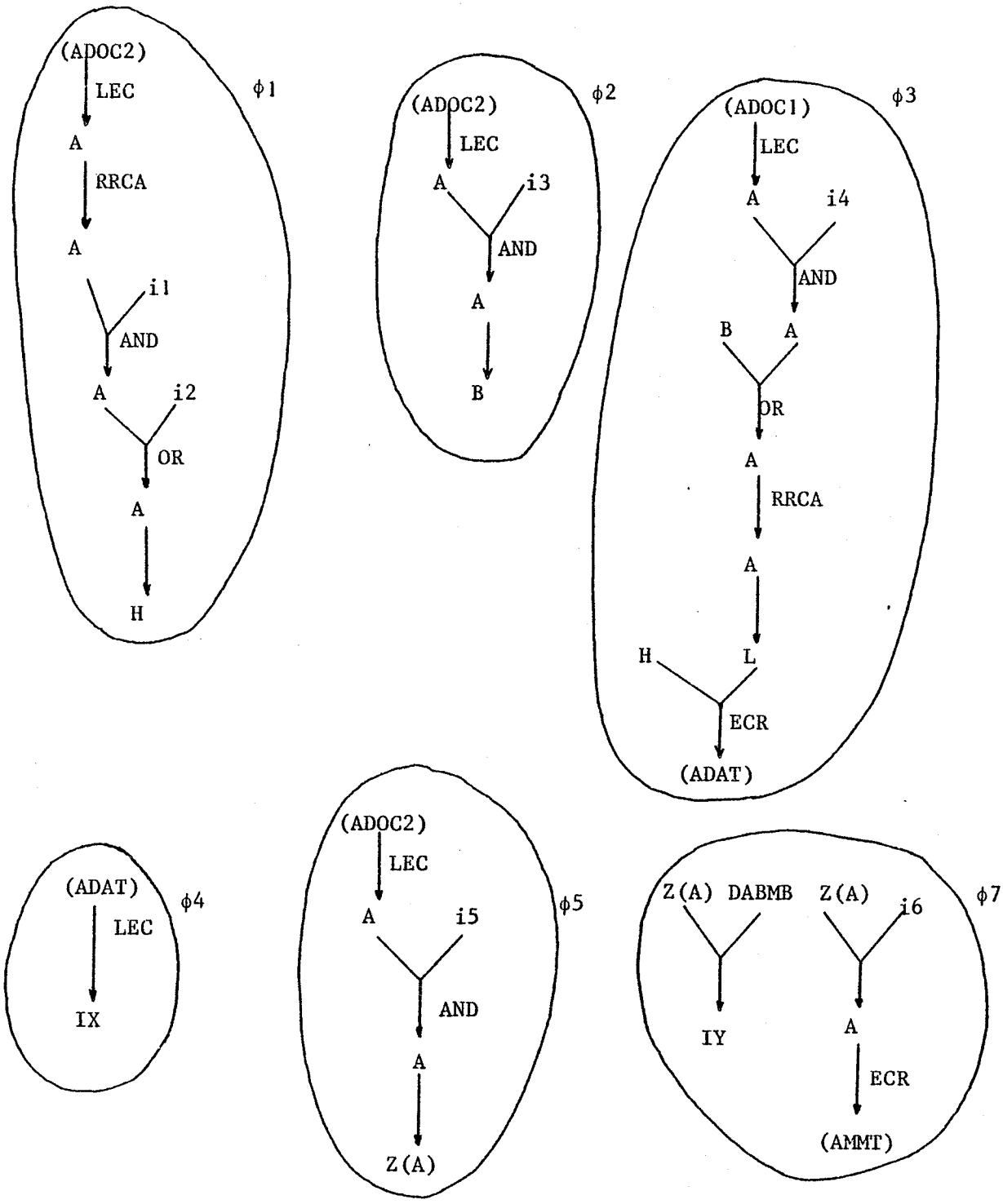


Figure 6. Graphes de dépendance

	Variables d'entrée	variables de travail	variables de sortie
$\phi 1$	(ADOC2)	A	H
$\phi 2$	(ADOC2)	A	B
$\phi 3$	(ADOC1), B, H	A, L	(ADAT)
$\phi 4$	(ADAT)	—	IX
$\phi 5$	(ADOC2)	A	Z
$\phi 7$	DABMB, Z	A	IY, (AMMT)

II - 5. SEQUENCE

Sur un chemin donné du graphe de contrôle on définit une séquence comme un ensemble ordonné de phases entre deux points d'observation consécutifs : la dernière phase d'une séquence et elle seule est une phase observable.

Exemple : $\{\phi 1, \phi 2, \phi 3\}$ et $\{\phi 4, \phi 5, \phi 7\}$ sont des séquences.

III - TEST DU SYSTÈME À MICROPROCESSEUR

III - 1. ORGANIGRAMME DE TEST

L'organigramme de conception du test est le suivant : soit l'ensemble des chemins possibles du programme d'application; un chemin est une suite ordonnée de phases, dont la première phase est la phase d'initialisation et dont la dernière phase est une phase finale du programme.

. a) pour chaque chemin du programme
- à chaque point d'observation du chemin (dernière instruction d'une séquence), définir le matériel, les fonctions et les instructions testées observables à ce point,
- déterminer les zones des valeurs d'entrée commandables permettant de passer par ce chemin, c'est-à-dire assurant les séquencements voulus.

. b) pour l'ensemble du programme
- déterminer un ensemble minimal de chemins tel que toutes les transitions soient couvertes : vérification du séquencement et de la ROM,
- évaluer l'efficacité du test lors d'un passage par cet ensemble de chemins, c'est-à-dire le degré de couverture par rapport aux hypothèses de pannes choisies.

. c) complétude du test
Si le test n'est pas complet par rapport aux hypothèses de pannes considérées, on le rend complet,
- soit en activant plus d'une fois un chemin donné de l'ensemble défini en b),
- soit en choisissant un (ou plusieurs) chemin(s) du programme d'application n'appartenant pas à l'ensemble défini en b).

. d) détermination complète des valeurs d'entrée, à l'intérieur des zones de valeur imposées par le séquencement, en tenant compte des hypothèses d'erreurs choisies pour les différents constituants du système.

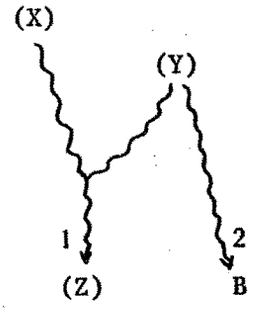
III - 2. ETAPE a) : LE TEST PAR UN CHEMIN DONNE

Le chemin est décomposé en séquences et l'objectif est de déterminer pour chaque séquence, c'est-à-dire à chaque point d'observation du chemin, ce qui est testé, c'est-à-dire le matériel, les fonctions et les instructions qui peuvent être observées à ce point.

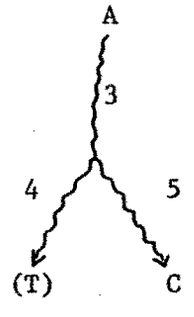
Plusieurs situations peuvent se présenter :

- les variables d'entrée de la séquence sont :
  soit commandables : ce seront les vecteurs de test,
  soit immédiates : entrées prédéterminées,
  soit non commandables : entrées calculées par le programme d'application antérieurement dans le chemin,
- les variables de sortie sont ou non observables, mais il en existe une -et une seule- qui est observable (par définition de la séquence).

On peut représenter les diverses situations comme suit :



séquence S



séquence S'

où . (X) et (Y) sont des variables d'entrée commandables ou immédiates
. A est une variable d'entrée non commandable
. (Z) et (T) sont des variables de sortie observables
. B et C sont des variables de sortie non observables.

α) *Qu'est-ce qui est réellement testé dans ces séquences ?*

. Séquence S : - le chemin (1) est commandable et observable : toutes les instructions de ce chemin, les fonctions et le matériel associés sont testés au point d'observation de S;

- le chemin (2) est commandable mais non observable : toutes les instructions, les fonctions et le matériel associés à ce chemin seront testés à un point d'observation O" d'une séquence S" telle que B soit sur un chemin observable de S" : propagation aval.

. Séquence S' : - le chemin (3,4) est observable mais non commandable : au point d'observation de la séquence S' on teste l'ensemble des instructions, des fonctions et du matériel de ce chemin ainsi que l'ensemble associé dans une séquence antérieure au chemin commandable mais non observable dont la sortie était A : consistence,

- la branche (5) n'est ni commandable ni observable : il faut faire à la fois la propagation aval de la variable C vers un point d'observation et la consistence de la variable A vers une entrée commandable ou immédiate.

. Si on détecte une erreur : - au point d'observation de S, on accusera les instructions, les fonctions et le matériel associé au chemin (1),

- au point d'observation de S', on accusera l'ensemble associé aux branches (3,4) ainsi qu'aux antécédents de A (à partir d'une entrée commandable ou immédiate),

- on ne pourra pas donner de verdict de bon fonctionnement des branches (2,5); il faut attendre que les variables B et C soient respectivement observées dans une séquence ultérieure.

En fait, l'algorithme de test permet d'éviter le problème de la consistence : le chemin étant parcouru dans l'ordre d'exécution des séquences on peut se ramener à une étude qui ne concerne que les variables de propagation.

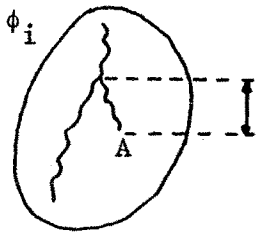
β) *Algorithme de diagnostic dans un chemin*

. Définitions

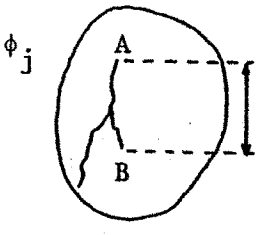
- E est l'ensemble des entrées qui ne sont ni commandables ni immédiates dans une séquence,
- T_j est l'ensemble de test de la phase ϕ_j, c'est-à-dire l'ensemble des instructions, du matériel et des fonctions activées par un chemin colorié,
- T^i est l'ensemble de test de la séquence S_i, c'est-à-dire l'ensemble des instructions, du matériel et des fonctions qui sont observables au point d'observation de S_i et qui sont accusées en cas d'erreur: T^i est donc l'ensemble de localisation.

- P est l'ensemble de propagation, c'est-à-dire l'ensemble des feuilles qui, à un moment donné du test, n'ont pas encore été observées à un point d'observation
- {V(p)} est l'ensemble de test associé à l'ensemble de propagation c'est-à-dire l'ensemble des instructions, du matériel et des fonctions qui sont observables à travers p $\in P$.

Soit une séquence ... ϕ_i... ϕ_j telle que



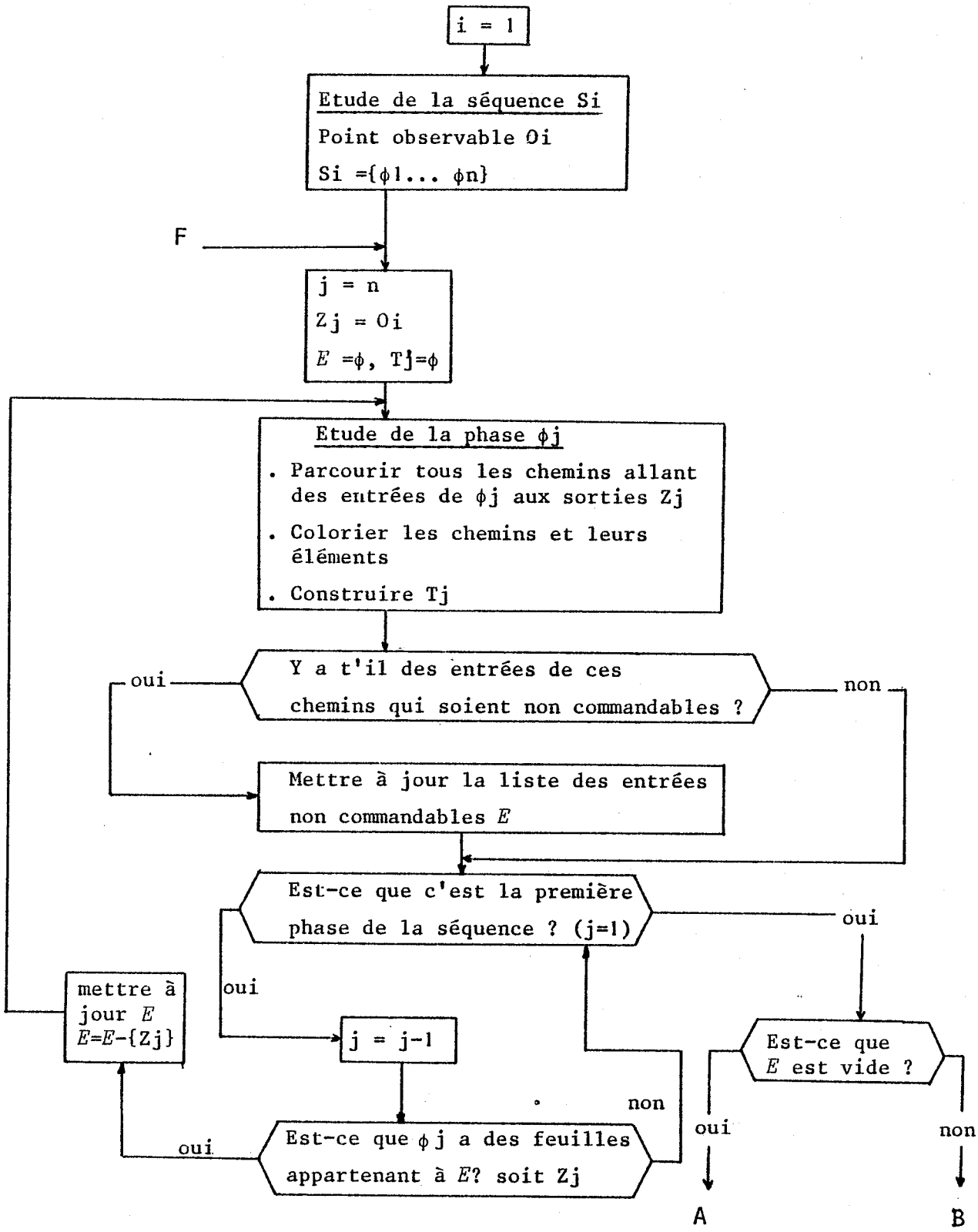
A non observable
A $\in P$
A $\leftrightarrow$ V(A)

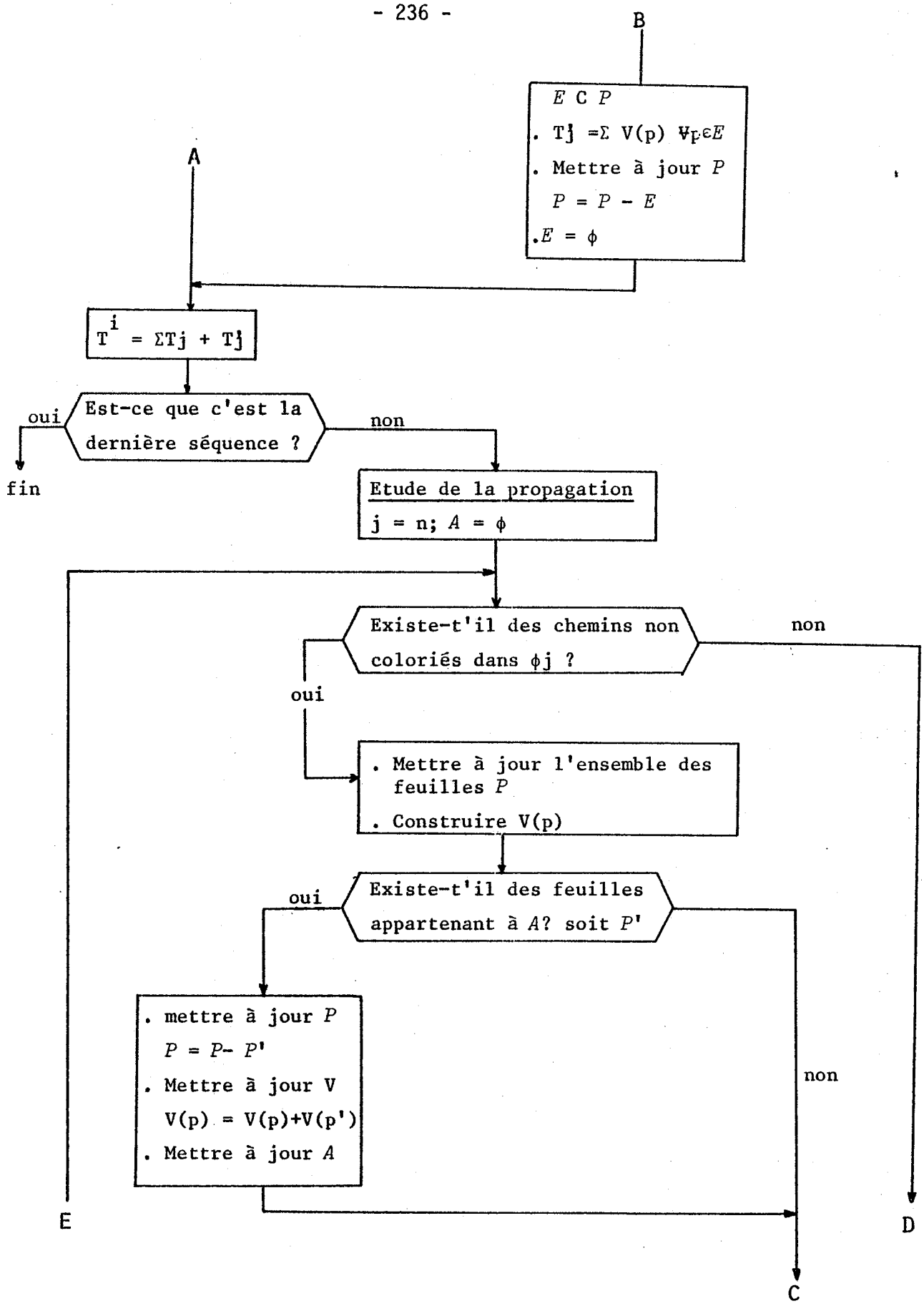


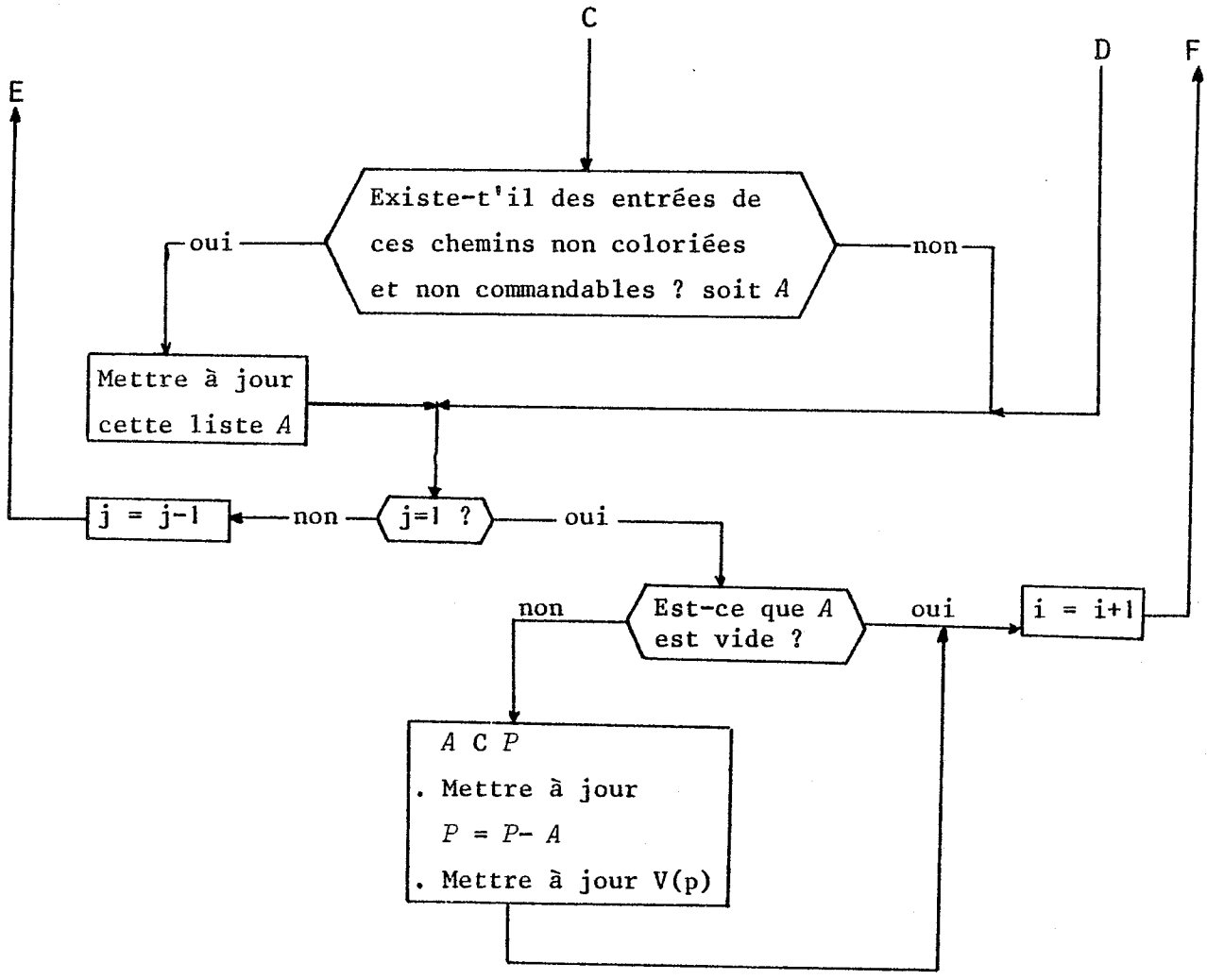
B non observable
B $\in P$
B $\leftrightarrow$ V(B)

Sur l'ensemble $\{\phi_i, \phi_j\}$ on définit V(B) = V(B)+V(A)

- A est l'ensemble des entrées non coloriées et non commandables pendant l'étape de la propagation.


i = 1
Etude de la séquence Si
Point observable Oi
Si ={ϕ1... ϕn}
F
j = n
Zj = Oi
E =ϕ, Tj=ϕ
Etude de la phase ϕj
. Parcourir tous les chemins allant des entrées de ϕj aux sorties Zj
. Colorier les chemins et leurs éléments
. Construire Tj
Y a t'il des entrées de ces chemins qui soient non commandables ?
oui
non
Mettre à jour la liste des entrées non commandables E
Est-ce que c'est la première phase de la séquence ? (j=1)
oui
oui
mettre à jour E E=E-{Zj}
j = j-1
Est-ce que E est vide ?
non
oui
non
Est-ce que ϕj a des feuilles appartenant à E? soit Zj
oui
A
B


B
E C P
. Tj =Σ V(p) ∀p∈E
. Mettre à jour P
P = P - E
.E = ϕ
A
T^i = ΣTj + Tj
Est-ce que c'est la dernière séquence ?
oui
non
fin
Etude de la propagation
j = n; A = ϕ
Existe-t'il des chemins non coloriés dans ϕj ?
non
oui
. Mettre à jour l'ensemble des feuilles P
. Construire V(p)
Existe-t'il des feuilles appartenant à A? soit P'
oui
non
. mettre à jour P
P = P- P'
. Mettre à jour V
V(p) = V(p)+V(p')
. Mettre à jour A
E
D
C


C
D
F
E
Existe-t'il des entrées de ces chemins non coloriées et non commandables ? soit A
oui
non
Mettre à jour cette liste A
j = j-1
non
j=1 ?
oui
non
Est-ce que A est vide ?
oui
i = i+1
A C P
. Mettre à jour P = P- A
. Mettre à jour V(p)

γ) *Exemple*

L'exemple d'application proposé pour illustrer cet algorithme est un exemple théorique faisant intervenir toutes les situations possibles.

On considère un programme comportant trois séquences S1, S2 et S3, chacune de ces séquences étant formée des phases suivantes :

S1 = {ϕ1, ϕ2, ϕ3

S2 = {ϕ4, ϕ5}

S3 = {ϕ6, ϕ7}

La figure suivante représente les graphes de dépendance de chacune des phases; pour la commodité de l'écriture, les arcs de ces graphes de dépendance sont indicés par le numéro de l'instruction exécutée qui caractérisera le matériel et les fonctions activées par cette instruction :

$\downarrow n$ $\qquad n \equiv (I_n$; {opérateurs, registres...}; {fonctions}

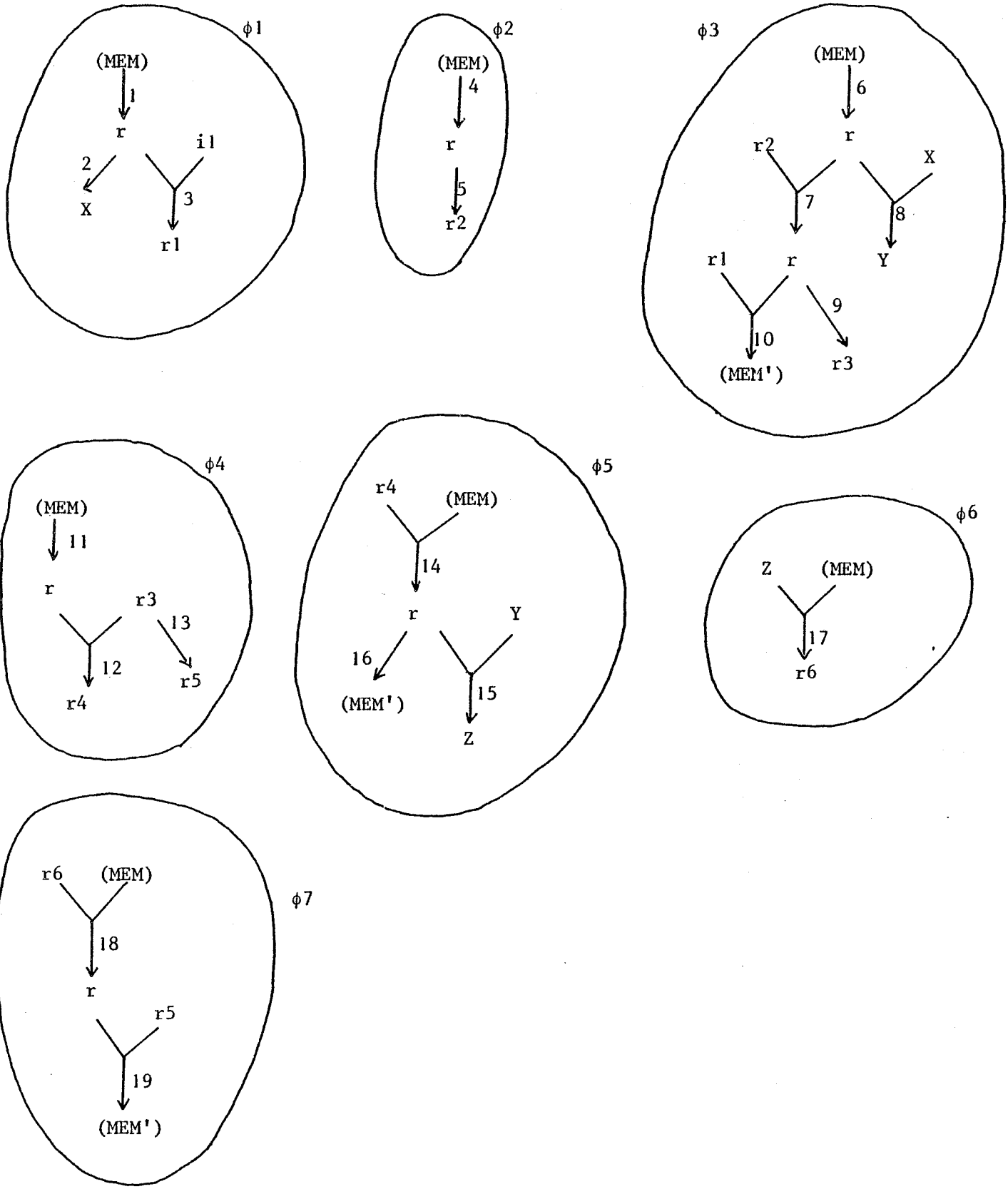
Remarque

(MEM) représente une entrée commandable

i_n représente une entrée immédiate

(MEM') représente une sortie observable

X, Y, Z, r, r_n représentent des variables (registres)



L'algorithme de test est le suivant:

. i = 1 : Etude de S1 S1 ={ϕ1 ϕ2 ϕ3}

. J = 3 : étude de ϕ3

$E = \phi$; $T'_3 = \phi$

Colorier 6,7,10

$T_3 = \{6,7,10\}$

r1 et r2 non commandables $E = \{r1,r2\}$

. J = 2 : étude de ϕ2

$Z_2 = \{r2\}$; $E = \{r1\}$

$T_2 = \{4,5\}$

. J = 1 : étude de ϕ1

Z1 = {r1} ; $E = \phi$

$T_1 = \{1,3\}$

. $T^1 = \{T1,T2,T3\}$

$\boxed{T^1 = \{1,3,4,5,6,7,10\}}$

Etude de la propagation

. J = 3 : étude de ϕ3

$A = \phi$

$P = \{r3,Y\}$

V(r3)= 9; V(Y) = 8

$A = \{X\}$

. J = 2 : étude de ϕ2

Il n'existe pas de chemins non coloriés dans ϕ2

. J = 1 : étude de ϕ1

$P = \{r3,Y,X\}$

V(X) = 2

$P' = \{X\}$

$P = \{r3,Y\}$

V(r3) = 9; V(Y) = {8,2}

$A = \phi$

. i = 2 : étude de S2 S2 = {ϕ4, ϕ5}

. J = 2 : étude de ϕ5

$E = \phi$; $T'_2 = \phi$

Colorier 14,16

$T_2 = \{14,16\}$

$E = \{r4\}$

. J = 1 : étude de ϕ4

$Z1 = \{r4\}$; $E = \phi$

$T1 = \{11,12\}$

$E = \{r3\}$

$T'_1 = V(r3) = 9$

$P = \{Y\}$; $E = \phi$

. $T^2 = \{T1, T2, T'1\}$

$\boxed{T^2 = \{9,11,12,14,16\}}$

Etude de la propagation

. J = 2 : étude de ϕ5

$A = \phi$

$P = \{Z,Y\}$

$V(Z) = 15$; $V(Y) = \{8,2\}$

$A = \{Y\}$

. J = 1 : étude de 4

$P = P \cup \{r5\} = \{Y,Z,r5\}$

$V(r5) = 13$

A non vide : $P' = \{Y\}$; $P = \{Z,r5\}$

$V(Z) = V(Z) + V(Y) = \{15,8,2\}$

$V(r5) = 13$

. i = 3 : Etude de S3 S3 ={ϕ6,ϕ7}

. J = 2 : étude de ϕ7

$E = \phi$; T'2 = ϕ

Colorier 18,19

T2 = {18,19}

E = {r5,r6}

. J = 1 : étude de ϕ6

Z1 = {r6}

E = {r5}

T1 = {17}

E = {r5,Z}

E non vide : T'1 = V(r5) + V(Z)

T'1 = {13,15,8,2}

T3 = {T1,T2,T'1}

$\boxed{T^3 = \{19,18,17,13,15,8,2\}}$

δ) *Prédétermination des entrées*

Le parcours d'un chemin de données induit une détermination partielle des variables d'entrée, permettant de vérifier les prédicats associés aux séquencements du graphe de contrôle. Cette prédétermination conduit à définir une "zone de valeur d'entrée" X_i qui traduit la réduction du domaine des entrées X par une certaine fonction pour le chemin i.

III - 3. EVALUATION DE L'EFFICACITE DU TEST

α) *Détermination de l'ensemble de chemins C*

On choisit un ensemble minimal de chemins couvrant toutes les transitions possibles du graphe de contrôle : tous les séquencements du programme d'application sont exécutés et la mémoire programme est entièrement parcourue.

La détermination de cet ensemble se fait par une méthode classique de couverture des transitions.

β) *Evaluation de l'efficacité du test*

Soit l'ensemble minimal de chemins

C = {C1... Cn}

Le problème consiste à évaluer l'efficacité du test lors d'un (ou de plusieurs) passage(s) par cet ensemble C.

On peut considérer deux cas :

- soit on a connaissance de la structure physique des constituants du microprocesseur : registres et unités de traitement; dans ce cas, on peut définir l'ensemble des pannes considérées pour chacun de ces constituants et un ensemble d'opérandes de test générés de façon déterministe peut être associé à chacun de ces modules, pour la classe de pannes envisagée [ROB,78e].

  Une solution consiste alors à chercher une activation des chemins qui enverrait à chaque opérateur cet ensemble d'opérandes : ceci revient à déterminer pour chaque chemin Ci $\in C$ les variables d'entrée commandables à l'intérieur de leur zone de valeur X_i,

- soit on ne connaît pas la structure interne de ces constituants ou cette structure est difficilement exploitable à des fins de test (formulation difficile des hypohtèses de défaillance, par exemple); une solution consiste alors à faire un test aléatoire de ces constituants : le problème revient à estimer la longueur N_{Li} d'une séquence de test aléatoire qu'il faut émettre à l'entrée d'un chemin C_i afin de simuler à l'entrée de chacun des modules utilisés par ce chemin des conditions de test analogues à celles du test aléatoire de ces modules s'ils étaient isolés.

III - 4. PREMIER CAS : TEST DETERMINISTE

On se place dans l'hypothèse où l'on connaît la structure physique des différents constituants du microprocesseur (partie opérative) et où l'on peut donc définir des hypothèses de pannes; de nombreuses méthodes de test [ROB,78a] permettent alors de déterminer pour ces constituants un ensemble minimal ou pseudo minimal de vecteurs de test, lorsque ce module est considéré comme isolé.

Lorsque ce module est considéré dans le contexte d'un programme d'application il se pose le problème de la composition de ces vecteurs, c'est-à-dire :

- comment générer des valeurs déterminées à l'entrée d'un module donné à partir des entrées du programme ? (c'est le problème de la consistence),
- comment s'assurer que les valeurs de test issues d'un module donné sont toujours significatives aux points d'observation du programme ? (c'est le problème de la propagation).

La consistence et la propagation doivent donc comporter une étude de la transformation de données par d'autres modules,

- entre les entrées du programme et l'entrée du module testé
- entre la sortie du module et les points d'observation du programme.

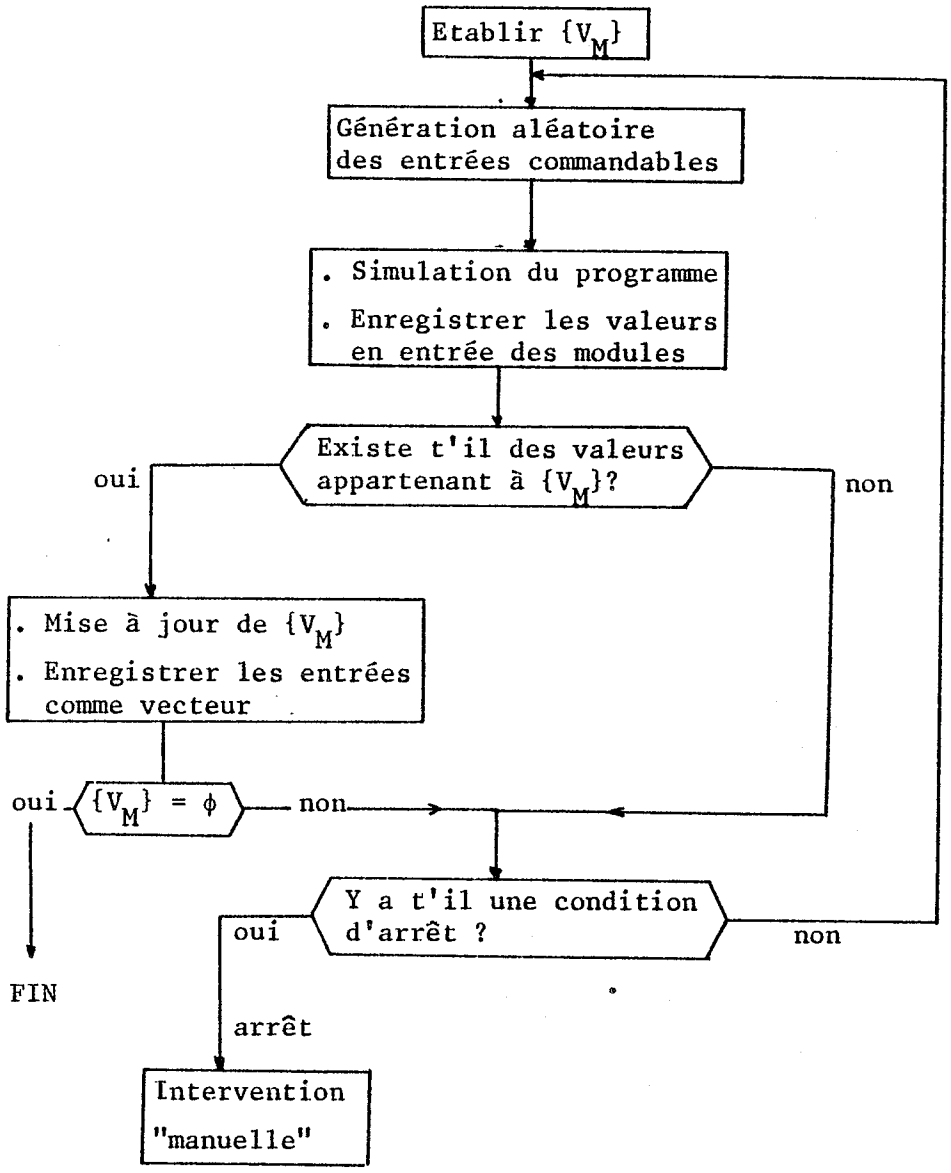
Ces problèmes peuvent difficilement être pris en compte de façon manuelle à l'exception de programmes d'application très simples et très courts. Aussi la solution adoptée utilise t'elle un outil de description fonctionnelle qui est exposé en [ZAC,77]; les graphes de dépendances peuvent être utilisés directement par cet outil (MAS).

. Génération des opérandes de test

On définit pour chaque module du microprocesseur qui est effectivement utilisé par le programme d'application l'ensemble minimal ou pseudo minimal de vecteurs de test assurant un test complet : à chaque module M on associe donc une liste V_M.

On fait une génération aléatoire des entrées commandables du programme : E, et, pour chaque chemin de l'ensemble C, la simulation fournit les valeurs obtenues en entrée des modules activés par le chemin.
Toute valeur obtenue en entrée de M qui appartient à V_M est retirée de cette liste et l'ensemble E des entrées commandables est enregistré.

L'organigramme de test est le suivant :



La simulation s'arrête dans les cas suivants :

- soit on a dépassé un temps limite prédéterminé
- soit on a exécuté n simulations et $\{V_M\}$ n'a pas été modifié, c'est-à-dire qu'on n'a généré aucun nouveau vecteur, n étant préalablement fixé,
- soit $\{V_M\} = \phi$ c'est-à-dire tous les vecteurs de test requis ont été générés pour tous les modules.
  L'ensemble des vecteurs de test est alors trouvé.

Les deux premières causes d'arrêt induisent une intervention "manuelle"; cette intervention comporte deux phases :

α) Faire exécuter l'algorithme de test sur les chemins du programme d'application n'appartenant pas à C; les valeurs limites de n et θ peuvent être différentes.

β) Si la phase α n'a pas conduit à $\{V_M\} = \phi$ et si l'on estime que le test n'est pas suffisant :

1) On vérifie si les valeurs de V_M sont réellement commandables à partir des entrées du programme.

2) Si elles sont commandables on fait la consistence pour déterminer les entrées commandables du programme.

III - 5. DEUXIEME CAS : TEST ALEATOIRE

α) *Matrice d'activabilité*

Pour chaque chemin $C_i \in C$ on définit une matrice d'activabilité :

	instructions	variables	mémoire	opérateurs
⋮				
S_i	$\{I\}$	p	F(m)	f(n)
⋮				

Cette matrice comporte quatre zones qui sont remplies pour chaque séquence du chemin considéré comme suit :

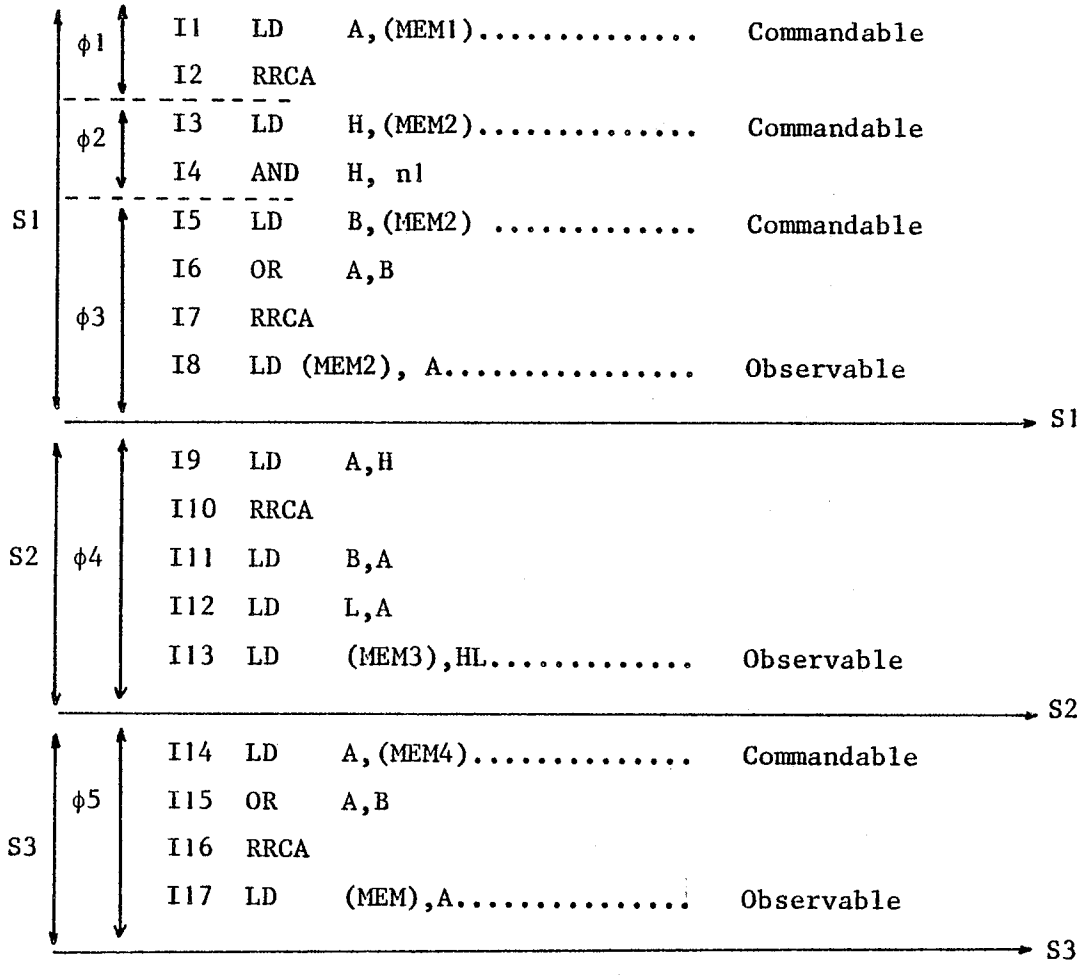
. une zone "instructions" contient la liste des instructions I qui sont accusées lors de la détection de mauvais fonctionnement au point d'observation de la séquence,

. une <u>zone "variable"</u> qui définit pour chaque variable (registre, point de mémorisation) utilisée par l'ensemble I le nombre de fois p où cette variable est transformée. Le nombre p peut être obtenu directement sur les graphes de dépendance comme le nombre d'arcs entrant sur la variable concernée,

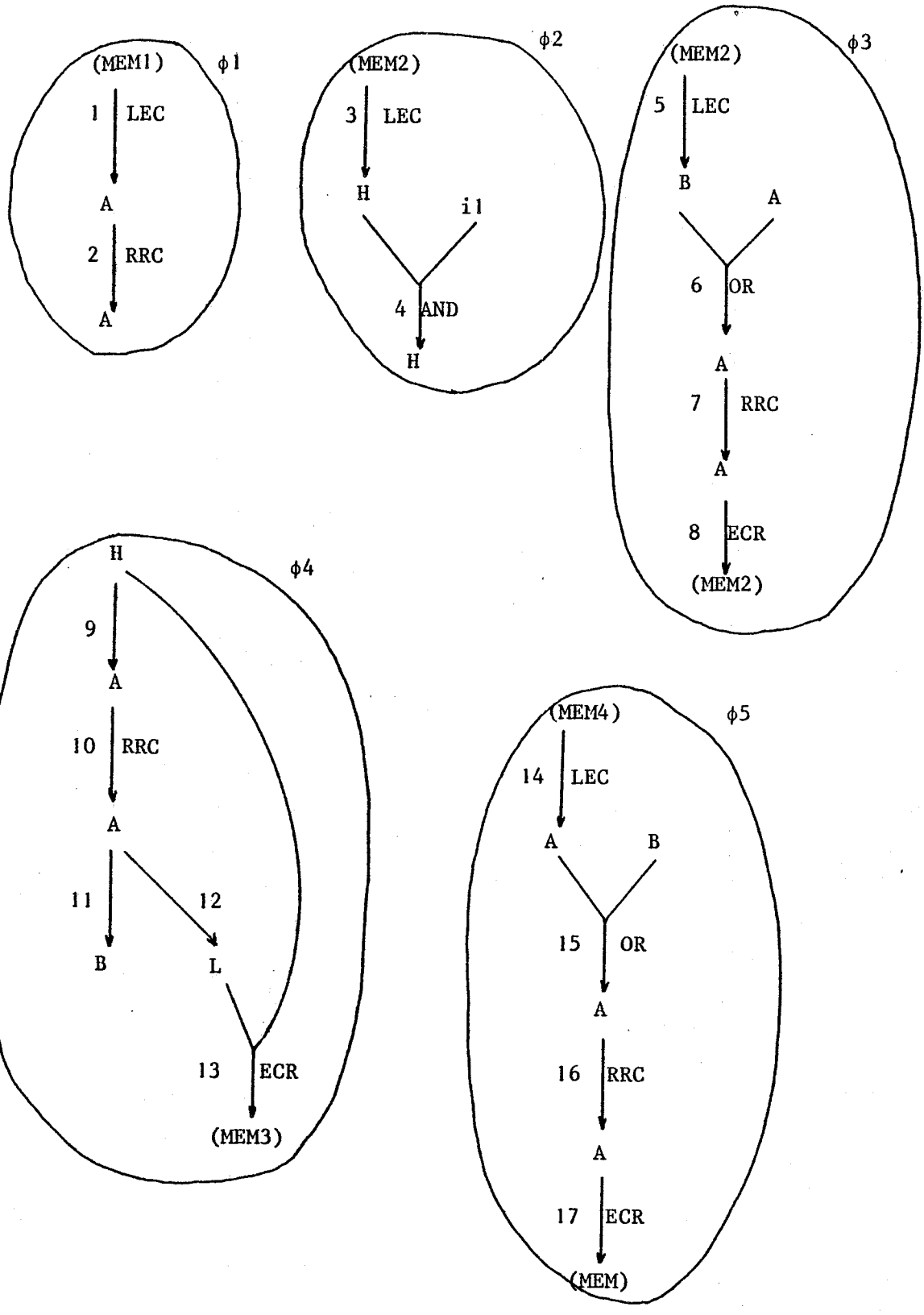
. une <u>zone "mémoire</u> qui définit la fonction mémoire F : lecture ou écriture et le nombre de fois m où cette fonction est activée par I

. une <u>zone "opérateurs"</u> qui définit pour chaque opérateur (UAL,...) les fonctions f exécutées dans I et le nombre de fois n où ces fonctions sont exécutées.

<u>Exemple</u> : Considérons l'exemple suivant :



Les graphes de dépendance sont les suivants :


(MEM1)
φ1
1 LEC
A
2 RRC
A
(MEM2)
φ2
3 LEC
H
i1
4 AND
H
(MEM2)
φ3
5 LEC
B
A
6 OR
A
7 RRC
A
8 ECR
(MEM2)
H
φ4
9
A
10 RRC
A
11
12
B
L
13 ECR
(MEM3)
(MEM4)
φ5
14 LEC
A
B
15 OR
A
16 RRC
A
17 ECR
(MEM)

La matrice d'activabilité est la suivante :

	I	A	H	B	L	MEM	UAL
S1	1,2,5 6,7,8	4	-	1	-	LEC(2) ECR(1)	RRC(2) OR(1)
S2	3,4,9,10 12,13	2	2	-	1	LEC(1) ECR(1)	RRC(1) AND(1)
S3	11,14,15 16,17	3	-	1	-	LEC(1) ECR(1)	OR(1) RRC(1)

L'exemple pratique donné aux §2.4 et 2.5. donne la partie de la matrice suivante :

	I	A	H	B	L	MEM	UAL
S1	I1..I14	10	1	1	1	LEC(3) ECR(1) OR(2)	RRC(2) AND(3)
S2	I16..I18 I21..I23	3				LEC(2) ECR(1)	AND(1) Z(A)

β) *Etude de la perte d'information*

Les résultats acquis dans le domaine du test aléatoire [THE,78] permettent de déterminer la longueur d'une séquence de test aléatoire N_{Lj} pour chacun des modules j (registre, unité de traitement,...) utilisé par le programme d'application, lorsque ce module est isolé.

Lorsque ce module fonctionne dans un système, on est amené à étudier le problème de la perte d'information: le fait d'exécuter une certaine opération sur un module à un point donné du programme d'application, peut faire perdre une partie des renseignements sur son bon fonctionnement (réduction du domaine de fonction).

On appelle quantité d'information le rapport du nombre de variables d'entrée (de sortie) possibles sur le nombre de variables d'entrée (de sortie) totales.

Exemple :

Soit un registre R de 8 bits sur lequel on effectue un décalage d'une position; si l'on suppose que l'on peut charger ce registre avec une valeur quelconque, après décalage on a seulement 7 bits significatifs (le 8e bit étant systématiquement à 0).
La quantité d'information en entrée vaut 1; la quantité d'information en sortie vaut $\frac{8-1}{8} = \frac{7}{8}$.

On peut donc pour chaque instruction utilisée évaluer la quantité d'information de la variable de sortie par rapport à la quantité d'information de la variable d'entrée. Cette quantité d'information est portée sur les graphes de dépendance des instructions, comme coefficient sur les variables.

La propagation de ces quantités d'information, dans chaque séquence, permet d'évaluer la quantité d'information aux points d'observation du programme.
On définit donc, en chaque point d'observation, un coefficient de réduction Ri égal à la quantité d'information disponible au point observable de la séquence Si.

γ) *Détermination du nombre d'activations aléatoires des modules*

Soit un module M du microprocesseur qui est activé p fois par une séquence Si; si ce module était isolé cela signifierait que, pour un test aléatoire on a eu p activations indépendantes; le coefficient de réduction traduit la dépendance entre les activations et on écrira que le passage par la séquence Si a le même effet que si l'on avait activé p x Ri fois le module isolé.
Le coefficient de réduction traduit donc la proportion d'augmentation du nombre d'opérandes aléatoires à envoyer sur un module donné.

On appelle N_i^M = p.Ri indice d'activation du module dans la séquence Si.

Sur l'ensemble des séquences d'un chemin Cj on calcule l'indice d'activation de chaque module dans le chemin

$$N_{Cj}^M = \sum_i p.Ri$$

Sur l'ensemble des chemins de C on calcule l'indice l'activation global

$$N_C^M = \sum_j N_{Cj}^M$$

Ce nombre N_C^M doit être comparé à une valeur limite N_{LM}, où N_{LM} est la longueur de la séquence de test aléatoire du module M isolé.

Si, pour tout module $N_C^M \geq N_{LM}$ alors l'ensemble de chemins C est suffisant pour tester le système avec une qualité de détection donnée (cette qualité de détection intervient dans le calcul de N_{LM}).

Si ce n'est pas le cas, on choisit un certain nombre de chemins supplémentaires -soit parmi l'ensemble C ,soit parmi les chemins non encore utilisés- permettant d'obtenir ∀M, $N_{C'}^M \geq N_{LM}$

Le choix se fait en tenant compte de l'indice d'activation des modules non encore testé et en utilisant des méhtodes de couverture classiques.

δ) *Détermination du coefficient de réduction*

La détermination du coefficient de réduction en chaque point d'observation peut se faire de façon "fastidieuse" en calculant la quantité d'information disponible après chaque transformation de cette information.

L'automatisation de l'étude proposée n'est possible que si l'on trouve un moyen de calcul de ce coefficient, à partir de règles précises sur la perte d'information après transfert et opérations sur les données.

Les problèmes rencontrés sont dûs :

- au phénomène de masquage entraîné par les opérations successives,
- à la présence d'entrées immédiates dont la commandabilité est nulle.

Des règles simples simples peuvent être trouvées facilement et fournir une approximation très grossière du nombre d'activations d'un chemin donné.
La détermination d'un coefficient de réduction non empirique pourrait se faire à l'aide des outils de théorie de l'information proposés pour l'étude de la testabilité d'un système et constitue un point de recherche à développer.

III - 6. CONCLUSION

Nous avons donc proposé une méthode de test devant spécialement satisfaire le concepteur d'un système à microprocesseur pour une application spécialisée , ce concepteur recherchant un test rapide, peu coûteux en matériel et "cernant son application". La méthode proposée consiste en l'étude du programme d'application "à travers" le matériel (décomposition de ce programme par rapport aux points de commande et d'observation du matériel). Cette étude fait donc une synthèse entre des notions de logiciel (test de programme) et des analyses du matériel support. Les améliorations doivent porter sur une meilleure prise en compte des défaillances matérielles au niveau des informations de test. Les prolongements ultérieurs pourront concerner l'aide à la programmation d'applications facilement testables sur microprocesseur.

CONCLUSION GENERALE

Il nous semble que le problème du test des systèmes informatiques sera toujours un problème difficile. En effet:

- c'est un problème à maîtriser avant et au cours de la conception: niveau système, niveau architecture,...

Il exige alors de modéliser le système de façon très globale et d'avoir des méthodes d'estimation à priori de testabilité. Cette approche, nous l'avons vu, est basée sur l'analyse de l'écoulement de l'information dans le système.

- c'est un problème pratique crucial; il s'agit de proposer des méthodes efficaces, très proches de la technologie utilisée. En particulier, l'évolution rapide vers la technologie très intégrée a nécéssité la recherche de méthodes de test à la fois fonctionnelles et structurelles.

L'importance de l'approche structurelle oblige à une enquête difficile sur les défaillances physiques dans les nouvelles technologies.

- c'est un problème qui intéresse à la fois les constructeurs et les utilisateurs, du composant au système; les méthodes de test proposées doivent être compatibles et cohérentes pour l'ensemble du problème du test.

En particulier, une attention particulière va à l'utilisateur de systèmes à microprocesseurs à application spécifique, qui désire une maintenance appropriée et efficace.

Il semble que le sujet soit "inépuisable", qu'une recherche incessante s'impose, dûe à l'évolution technologique. Il semble également qu'un jeu d'outils standards (bibliothèque de programmes de test pour les types de microprocesseurs les plus courants) serait d'une utilité nationale.

BIBLIOGRAPHIE

REFERENCES CHAPITRE 1:

" MODÈLE DE TESTABILITÉ D'UN SYSTÈME LOGIQUE "

[AGN,65] AGNEW.P.W, RUTHERFORD.D.H, SUHOCKI.R.J, YEN.C.M and MULLER.D.E: *"An architectural study for a self-repairing computer", U.S. Air Force Space Division, Final Tech. Doc. Rep. SSD-TR-65-159, AD 474976, Novembre 1965.*

[AKE,76] AKERS.S.B: *"Partitioning for testability", International Symposium on Fault-tolerant Computing, Pittsburgh,USA, Juin 1976, pp 121-126.*

[ANS,77] ANSELMO.G: *"Système E 10. Echange entre les sous-ensembles", Commut. et Electron., Janvier 1977, pp25-38.*

[BAR,76a] BARBACCI.M.R: *"A comparison of register transfer languages for describing computers and digital systems", IEEE Trans. on Computers, Février 1975, pp137-150.*

[BAR,75] BARSI.F, GRANDONI.F, MAESTRINI.P: *"Diagnosability of systems partitioned into complex units", International Symposium on Fault-Tolerant Computing, Paris, Juin 1975, pp171-175.*

[BAR,76b] BARSI.F, GRANDONI.F, MAESTRINI.P: *"A theory of diagnosability of digital systems", IEEE Trans. on Computers, Juin 1976, pp 585-593.*

[BEL,78] BELLON.C, ROBACH.C, KUBIAK.C: *"Modélisation des systèmes distribués en vue de la détection des pannes", Annales des Télécommunications, Novembre- Décembre 1978,pp 383-395.*

[BER,63] BERGE.C:
"Théorie des graphes et ses applications", Editions DUNOD, Collection Universitaire de Mathématiques, Paris, 1963.

[CAS,77a] CASPI.P, ROBACH.C:
"Application de la théorie de l'information à l'évaluation de la testabilité d'un système", Rapport de recherche IMAG n° 69, Université de Grenoble, Mars 1977.

[CAS,77b] CASPI.P, MILI.A, ROBACH.C:
"An information measure on nets- Application to the testability of digital systems", IFAC Workshop: Information and Systems, Compiègne, Octobre 1977, pp37-41.

[CHA,74] CHANG.H.Y, HEIMBIGNER.G.W:
"LAMP: Controllability, Observability and Maintenance Engineering Technique (COMET)", The Bell System Technical Journal, Octobre 1974, pp1505-1534.

[COU,77] COUSOT.P:
"Asynchronous iterative methods for solving a fixed point system of monotonic equations in a complete lattice", Rapport de recherche IMAG n° 88, Université de Grenoble, Septembre 1977.

[DEN,68] DENT.J.J:
"Diagnostic engineering requirements", Proc. AFIPS, SJCC, 1968, pp503-507.

[DUB,61] DUBREIL.P, DUBREIL-JACOTIN.M.L:
"Leçon d'Algèbre moderne", Dunod, Paris, 1961.

[FOR,65] FORBES.R.E, RUTHERFORD.D.H, STIEGLITZ.C.B and TUNG.L.H:
" A self-diagnosable computer", 1965 Fault-Joint Computer Conference, AFIPS Conf. Proc., Vol 27, Washington D.C: Spartan, 1965, pp 1073-1086.

[FRI,75] FRIEDMAN.A.D:
"A new measure of digital system diagnosis", International Symposium on Fault-Tolerant Computing, Paris, Juin 1975, pp167-17(

[GLU,69] GLUSHKOV.V.M, LETICHEVSKII.A.A:
"Advances in information systems science", Vol.1, Plenum Press, New-York, Ed. Julius T.TOU, 1969.

[HAY,76] HAYES.J.P:
"A graph model for fault-tolerant computing systems", IEEE Trans. on Computers, Septembre 1976, pp 875-854.

[KIM,70] KIME.C.R:
"An analysis model for digital system diagnosis", IEEE Trans. on Computers, Novembre 1970, pp1063-1073.

[KUB,76] KUBIAK.C, PRIGENT.A, ROBACH.C, SAUCIER.G:
"Modélisation et maitenance- Application au système E 10", Colloque AFCET-ADEPA: Automatismes logiques - Recherche et Applications industrielles, Paris, Décembre 1976, pp179-200.

[MIL,78] MILI.A:
"Outils d'aide à la décision dans le test des systèmes logiques", Thèse de 3ème cycle, Université de Grenoble, Juin 1978.

[OGU,74] OGUS.R.C:
"Probability of a correct output from a combinational circuit", International Symposium on Fault-Tolerant Computing, Champaign, USA, Juin 1974, pp13-19.

[PET,56] PETRICK.S.R:
"A direct determination of the irredundant forms of a boolean function from the set of prime implicants", U.S Air Force Cambridge Res. Center, Bedford Mass., Tech. Rep. 56-110, Avril 1956.

[PRE,67] PREPARATA.F.P, METZE.G, CHIEN.R.T:
"On the connection assignment problem of diagnosable systems", IEEE Trans. on Computers, Vol. EC-16, Décembre 1967, pp848-854.

[PRE,68] PREPARATA.F.P:
"Some results on sequentially diagnosable systems", Proc. Hawaii Int. Conf. Syst. Sci., University of Hawaii, Janvier 1968, pp 623-626.

[RAM,67] RAMAMOORTHY.C.V:
"A structural theory of machine diagnosis", 1967 Spring Joint Comp. Conference, AFIPS Conf. Proc. Vol. 30, Washington D.C: Thompson, 1967, pp743-756.

[RAM,72] RAMAMOORTHY.C.V, CHANG.L.C:
"System modeling and testing procedures for microdiagnostics", IEEE Trans. on Computers, Novembre 1972, pp1169-1183.

[ROB,75] ROBACH.C:
"Méthodologie de test de processeurs- Impacts sur la conception", Thèse de Docteur-ingénieur, Université de Grenoble, Mai 1975.

[ROB,76] ROBACH.C, SAUCIER.G, LEBRUN.J:
"Processor testability and design consequences", IEEE Trans. on Computers, Juin 1976, pp645-652.

[ROB,77a] ROBACH.C, SAUCIER.G:
"System modeling and diagnosability", COMPCON Spring 77, Février-Mars 1977, San Francisco, USA, pp294-299.

[ROB,77b] ROBACH.C, CASPI.P:
"Modèles pour l'étude de la testabilité des systèmes informatiques Congrès AFCET: Modélisation et maîtrise des systèmes techniques, économiques, sociaux, Versailles, Novembre 1977, pp654-665.

[ROB,78] ROBACH.C:
"Le test en production et en exploitation", Rapports de recherche n° 112 et 132, Université de Grenoble, Février-Août 1978.

[RUS,73a] RUSSEL.J.D, KIME.C.R:
"On the diagnosability of digital systems", Proc. Fault-Tolerant Computing Conference, Juin 1973, pp139-144.

[RUS,73b] RUSSEL.J.D:
"On the diagnosability of digital systems", Ph.D. Dissertation, University of Wisconsin, Madison, Juillet 1973.

[RUS,75a] RUSSEL.J.D, KIME.C.R:
"System fault diagnosis: Closure and diagnosability with repair", IEEE Trans. on Computers, Novembre 1975, pp1078-1089.

[RUS,75b] RUSSEL.J.D, KIME.C.R:
"System fault diagnosis: Masking, exposure and diagnosability without repair", IEEE Trans. on computers, Décembre 1975, pp1155-1161.

[SES,71] SESHAGIRI.N:
"Completely self-diagnosable digital systems", Int. J. Syst. Sc. Vol.1, Janvier 1971, pp235-246.

[STE,76] STEPHENSON.J.E, GRASON.J:
"A testability measure for register transfer level digital circuits", International symposium on Fault-Tolerant Computing, Pittsburgh, Juin 1976, pp101-107.

[TAR,55] TARSKI.A:
"A lattice theoretical fixpoint theorem and its applications", Pacific J. of Math.5, 1965, pp285-309.

[VAN,72] VANCURA.R.P, KIME.C.R:
"On numerical bounds in diagnosable systems", International Symposium on Fault-Tolerant Computing, Juin 1972, pp148-153.

REFERENCES CHAPITRE 2:

" LE TEST DES UNITÉS DE CONTRÔLE "

[BEL,76] BELLON.C,SAUCIER.G:
"Graceful degradation of performance in a fault-tolerant multiprocessor system", Euromicro Symposium, Venise, Octobre 1976, pp 75-81.

[BEL,78] BELLON.C, ROBACH.C, KUBIAK.C:
"Modélisation des systèmes distribués en vue de la détection des pannes", Annales des Télécommunications, Novembre-Décembre 1978, pp383-395.

[COO,73] COOK.R.W, SISSON.W.H, STOREY.T.F and TOY.W.N:
"Design of a self-checking microprogram control", IEEE Trans. on Computers, Mars 1973, pp 255-262.

[GLU,69] GLUSHKOV.V.M, LETICHEVSKII.A.A:
"Advances in information systems science", Vol.1, Plenum Press, New York, edited by Julius T. Tou, 1969.

[GOO,75] GOODENOUGH.J.B, GERHART.S.L:
"Toward a theory of test data selection", International Conference on reliable software, Los Angeles, USA, 1975, pp 493-510.

[HIL,73] HILL.F.J, PETERSON.G.R:
"Digital systems: hardware organization and design", John Wiley and Sons, Inc., New York, 1973.

[HUS,70] HUSSON.S.S:
"Microprogramming: principles and practices", Prentice Hall, 1970.

[KOH,70] KOHAVI.Z:
"Switching and finite automata theory", Chap. 13, Mc Graw Hill, Computer Science Series, 1970.

[NOR,77] NORDMANN.B.J, Mc CORMICK.B.H:
"Modular asynchronous control design", IEEE Trans. on Computers, Mars 1977, pp196-207.

[ROB,75a] ROBACH.C:
"Méthodologie de test de processeurs- Impacts sur la conception", Thèse de Docteur-Ingénieur, Université de Grenoble, Mai 1975.

[ROB,75b] ROBACH.C, SAUCIER.G:
"Diversified test methods for local control units", IEEE Trans. on Computers, Mai 1975, pp562-567.

[ROB,76a] ROBACH.C, SAUCIER.G:
"Processor testability and design consequences", IEEE Trans. on Computers, Juin 1976, pp 645-652.

[ROB,76b] ROBACH.C, SAUCIER.G:
"Easily diagnosable control in a processor", International Symposium on Fault-Tolerant Computing, Pittsburgh, USA, Juin 1976.

[ROB,78] ROBACH.C, SAUCIER.G:
"Dynamic testing of control units", IEEE Trans. on Computers, Juillet 1978, pp617-623.

[ROT,68] ROTH.J.P:
"Diagnosis of automata failure: a calculus and a method", IBM Journal R&D, Vol. 10, 1968, pp 278-291.

[WAK,74] WAKERLY.J:
"Low-cost error detection techniques for small computers", Chap. 7, Ph. D. Dissertation, Stanford University, 1974.

REFERENCES CHAPITRE 3:

" LE TEST D'UN SYSTÈME À MICROPROCESSEUR "

[BEL,77] BELLON.C, SAUCIER.G:
"On-line test modeling in non redundant distributed systems", Fault-Tolerant Computing Symposium, Los Angeles, USA, Juin 1977, pp94-102.

[BEL,78] BELLON.C, ROBACH.C, KUBIAK.C:
"Modélisation des systèmes distribués en vue de la détection des pannes, Annales des Télécommunications, Novembre-Décembre 1978, pp 383-395.

[BOU,71] BOURICIUS.W.G, CARTER.W.C and al.:
"Reliability modeling for fault-tolerant computers", IEEE Trans. on Computers, Vol C-20, Novembre 1971, pp1306-1311.

[CAI,76] CAILLAT.J:
"Contribution au test des circuits intégrés logiques", Thèse de 3ème cycle, Université de Grenoble, Octobre 1976.

[CRO,78] CROUZET.Y:
"Conception de circuits à large échelle d'intégration totalement autotestables", Thèse de Docteur-ingénieur, Institut National Polytechnique de Toulouse, Novembre 1978.

[DEN,68] DENT.J.J:
"Diagnostic engineering requirements", Proc. AFIPS, SJCC, 1968, pp 503-507.

[FEE,77] FEE.W.G:
"Tutorial on LSI testing", COMPCON Spring 77, San Francisco, USA, Février 1977.

[GAL,78] GALIAY.J:
"Conception de circuits à large échelle d'intégration facilement testables", Thèse de Spécialité, Université Paul Sabatier, Toulouse, Novembre 1978.

[GOB,78] GOBBI.J.M:
"Outil pour le test de cartes à microprocesseurs", Rapport DEA, Université de Grenoble, Juin 1978.

[HOL,77] HOLYFIELD.S:
"The microprocessor test quandary", Tutorial on LSI testing, COMPCON Spring 77, San Francisco, USA, Février 77,pp 46-50.

[KER,78] KERNIOPF.P, MARCZYNSKI.R.W, MICHALSKI.A:
"Microcomputer fault-detection using the time-based method", Euromicro Symposium, Munich, Octobre 1978, pp 74-79.

[LAN,78] LANDRAULT.C, ROUSSEAU.P:
"Vers des microprocesseurs facilement testables et autotestables", L' Onde Electrique, Vol 58, n° 12, Décembre 1978, pp 836-841.

[MAR,77] MARSHALL.M:
"Microprocessors in testingland: the jury receives its instructions", Tutorial on LSI testing, COMPCON Spring 77, San Francisco, USA, Février 1977, pp36-45.

[RAI,79] RAINARD.J.L:
"Vers des microprocesseurs auto-testables", Journée d'études CNET, Grenoble, 26 Avril 1979.

[REN,78] RENNELS.D.A, AVIZIENIS.A, ERCEGOVAC.M:
"A study of standard building blocks for the design of fault-tolerant distributed computer systems", Fault-Tolerant Computing Symposium, Toulouse, Juin 1978, pp 144-149.

[ROB,75] ROBACH.C:
"Méthodologie de test de processeurs- Impacts sur la conception", Thèse de Docteur-ingénieur, Université de Grenoble, Mai 1975.

[ROB,76] ROBACH.C, SAUCIER.G, LEBRUN.J:
"Processor testability and design consequences", IEEE Trans. on Computers, Juin 1976, pp 645-652.

[ROB,78a] ROBACH.C:
"Le test en production et en exploitation", Rapport de recherche IMAG n° 112- 132, Université de Grenoble, Février- Août 1978.

[ROB,78b] ROBACH.C, SAUCIER.G:
"Constructeurs et utilisateurs face au test des microprocesseurs", Colloque international sur la fiabilité et la maintenabilité, Paris, Juin 1978, pp 273-282.

[ROB,78c] ROBACH.C, SAUCIER.G:
"Dynamic testing of control units", IEEE Trans. on Computers, Juillet 1978, pp 617-623.

[ROB,78d] ROBACH.C, GOBBI.J.M:
"Microprocessor systems testing", Euromicro Symposium, Munich, Octobre 1978, pp 66-73.

[ROB,78e] ROBACH.C, SAUCIER.G:
"Le test logique des circuits intégrés", L'Onde Electrique, Vol 58, n° 12, Décembre 1978, pp842-849.

[ROB,79] ROBACH.C, SAUCIER.G, ALEONARD.C:
"Microprocessor systems testing: a rewiew and future prospects", EUROMICRO Journal, Janvier 1979, pp 31-37.

[ROT,68] ROTH.J.P:
"Diagnosis of automata failure: a calculus and a method", IBM Journal R&D, Vol. 10, 1968, pp 278-291.

[ROU,77] ROUAUD.Y:
"Maquette à microprocesseurs et allocateur de ressources pour traitement d'impulsions de numérotation", Rapport technique CNET, FT/ RCI/ PLC/ 1, Juin 1977.

[SED,78] SEDMAK.R.M, LIEBERGOT.H.L:
"Fault-tolerance of a general purpose computer implemented by very large scale integration", Fault-tolerant Computing Symposium, Toulouse, Juin 1978, pp 137-143.

[SMI,77] SMITH.D.H:
"Microprocessor testing- Method or madness", Tutorial on LSI testing, COMPCON Spring 77, San Francisco, USA, Février 1977, pp 21-23.

[THA,78] THATTE.S.M, ABRAHAM.J.A:
"A methodology for functional level testing of microprocessors", Fault-tolerant Computing Symposium, Toulouse, Juin 1978, pp90-95.

[THE,78] THEVENOD-FOSSE.P:
"Contribution à l'étude du test aléatoire des circuits séquentiels et des mémoires", Thèse de Docteur-ingénieur, Université de Grenoble, Février 1978.

[WAD,78] WADSACK.R.L:
"Fault modeling and logic simulation of CMOS and MOS integrated circuits", The Bell System Technical Journal, Mai-Juin 1978, pp 1449-1473.

[ZAC,77] ZACHARIADES.M:
"MAS: Réalisation d'un langage d'aide à la description et à la conception des systèmes logiques", Thèse de 3ème cycle INPG, Université de Grenoble, Septembre 1977.

TABLE DES MATIÈRES

III - ANALYSE DU MODÈLE STRUCTUREL

III - 1. Notations - Définitions
III - 2. Mesure de commandabilité
III - 3. Mesure d'observabilité
III - 4. Interprétation
III - 5. Etude de la testabilité

IV - CONCLUSION

CHAPITRE II : LE TEST DES UNITES DE CONTRÔLE 119

III - TEST DU SYSTÈME À MICROPROCESSEUR

III - 1. Organigramme de test

III - 2. Etape a) : le test par un chemin donné

III - 3. Evaluation de l'efficacité du test

III - 4. Test déterministe

III - 5. Test aléatoire

CONCLUSION

CONCLUSION GÉNÉRALE