UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

THESE

présentée à

L'UNIVERSITE DES SCIENCES ET TECHNIQUES DE LILLE

pour obtenir

le titre de Docteur Troisième Cycle

par

Marc GILLANT

LANGAGES DE DESCRIPTION DE PROCESSUS

ET D'AUTOMATISMES SEQUENTIELS

Membres du Jury : MM. BACCHUS, Président
CARREZ, Examinateur
VIDAL, Examinateur
DRIEUX, Rapporteur

Soutenue le 5 Juin 1972

UNIVERSITE DES SCIENCES
ET TECHNIQUES DE LILLE.

DOYENS HONORAIRES

MM. H.LEFEBVRE, PARREAU.

PROFESSEURS HONORAIRES

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

PROFESSEURS TITULAIRES

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

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

PROFESSEURS A TITRE PERSONNEL

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

PROFESSEURS SANS CHAIRE

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

MAITRES DE CONFERENCES (et chargés de fonctions)

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

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

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

Je voudrais exprimer ma gratitude à Monsieur le Professeur BACCHUS, *Directeur de l'U.E.R. Informatique-Electronique-Electrotechnique et Automatique de l'Université de LILLE pour l'honneur qu'il me fait de présider le jury.*

Je remercie vivement Monsieur le Professeur CARREZ *et Monsieur le Professeur* VIDAL *qui se sont intéressés à ce travail et qui ont accepté de le juger.*

J'adresse aussi ma vive reconnaissance et mes remerciements à Monsieur le Professeur DRIEUX *qui m'a donné l'idée de ce travail et a suivi de près sa réalisation.*

Enfin, je n'aurais garde d'oublier les personnes du laboratoire de Calcul de LILLE qui ont aidé à la réalisation de cette thèse, Madame et Monsieur DEBOCK *et particulièrement Mademoiselle* DRIESSENS *qui a fait preuve de sa compétence habituelle dans la réalisation matérielle de cette thèse.*

TABLE des MATIERES

INTRODUCTION

Un automatisme est un système qui, à partir d'un certain nombre d'informations, commande et contrôle l'exécution de tâches réalisées sur un ensemble de machines que nous appellerons équipement.

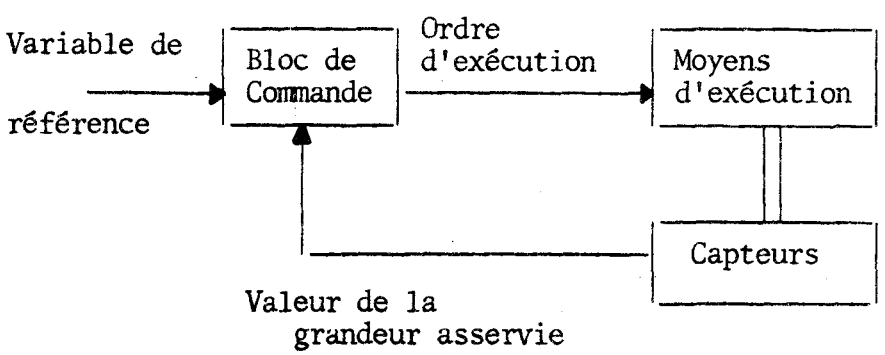
Un tel système se décompose en deux parties :

- le bloc de commande qui émet les ordres d'exécution des tâches,
- l'ensemble des moyens d'exécution des tâches.

En outre, des capteurs associés à l'équipement fournissent au bloc de commande des informations qui le renseignent sur l'exécution des tâches en cours.

Cependant la notion d'automatisme recouvre en fait des domaines d'application fort différents à la fois du point de vue utilisation et du point de vue réalisation ; nous distinguerons trois types d'automatisme :

i) *les asservissements* : ils sont chargés d'asservir une position, une vitesse ou plus généralement une grandeur quelconque à la valeur d'une variable de référence considérée comme une variable d'entrée par le bloc de commande. Ces automatismes sont structurés suivant le schéma suivant :



ii) *les automatismes séquentiels* : les moyens d'exécution des tâches contrôlées par un automatisme séquentiel se caractérisent par deux états : l'état "travail" et l'état "repos". Les ordres d'exécution émis par le bloc de commande sont donc des signaux "tout ou rien" auxquels correspondent des variables booléennes dites de sortie.

Ainsi l'état d'un automatisme séquentiel est caractérisé par les valeurs des variables d'entrée, de sortie et des variables émises par les capteurs.

Un système séquentiel en fonctionnement passe successivement, à partir d'un état de repos défini, par une suite déterminée d'états à laquelle correspond une seule séquence de commandes émises par le bloc de commande. En règle générale un système séquentiel est en mesure d'exécuter plusieurs séquences de commandes différentes.

Un système séquentiel est dit

- asynchrone si, durant son fonctionnement, l'émission d'un ordre d'exécution suit immédiatement celle du ou des ordres d'exécution précédents;

- synchrone si, durant son fonctionnement, il existe un écart de temps fixe τ entre deux émissions successives d'ordres d'exécution.

De plus la notion de système séquentiel peut être étendue et appliquée à des équipements complexes où interviennent des calculateurs et des asservissements.

La propriété essentielle de ces systèmes est qu'ils ne peuvent présenter qu'un nombre limité de fonctionnements et donc ne peuvent remplir qu'un nombre limité de fonctions.

iii) *les systèmes programmés* ; ils se caractérisent essentiellement par le fait que leur fonctionnement peut être modifié d'une façon arbitraire par leur utilisateur. Le nombre de séquences de commandes possibles est donc potentiellement infini.

Les spécialistes distinguent deux types de systèmes programmés :

- les systèmes de positionnement ou systèmes point à point : ils exécutent deux sortes de tâche : les déplacements et les usinages. Les opérations d'usinage ne sont effectuées que lorsque les positionnements d'organes ont été

correctement exécutés sur des points fixes. Ces machines fonctionnent généralement à l'aide de commandes tout ou rien.

- les systèmes à positionnement continu ou systèmes de contournage : les déplacements s'effectuent alors suivant des trajectoires déterminées en même temps que s'exécutent des tâches d'usinage. Ces systèmes sont habituellement commandés en "continu".

Les automatismes peuvent donc être regroupés suivant deux critères :

- un critère "software" qui permet de distinguer les automatismes séquentiels et programmés

- un critère "hardware" qui différencie les automatismes à commande tout ou rien des systèmes à commande continue.

La combinaison de ces critères permet donc, en théorie du moins, de classer les automatismes en quatre classes :

séquentiel continu,
séquentiel "tout ou rien"
programmé "continu"
programmé " tout ou rien".

En première approximation on peut considérer que : les asservissements constituent la classe des automatismes séquentiels "continus", les systèmes séquentiels définis ci-dessus constituent celle des automatismes séquentiels "tout ou rien" et enfin que les systèmes de positionnement point à point constituent la classe des automatismes programmés "tout ou rien" tandis que les systèmes de contournages constituent la classe des systèmes programmés continus.

Cette classification reste grossière ; en effet :

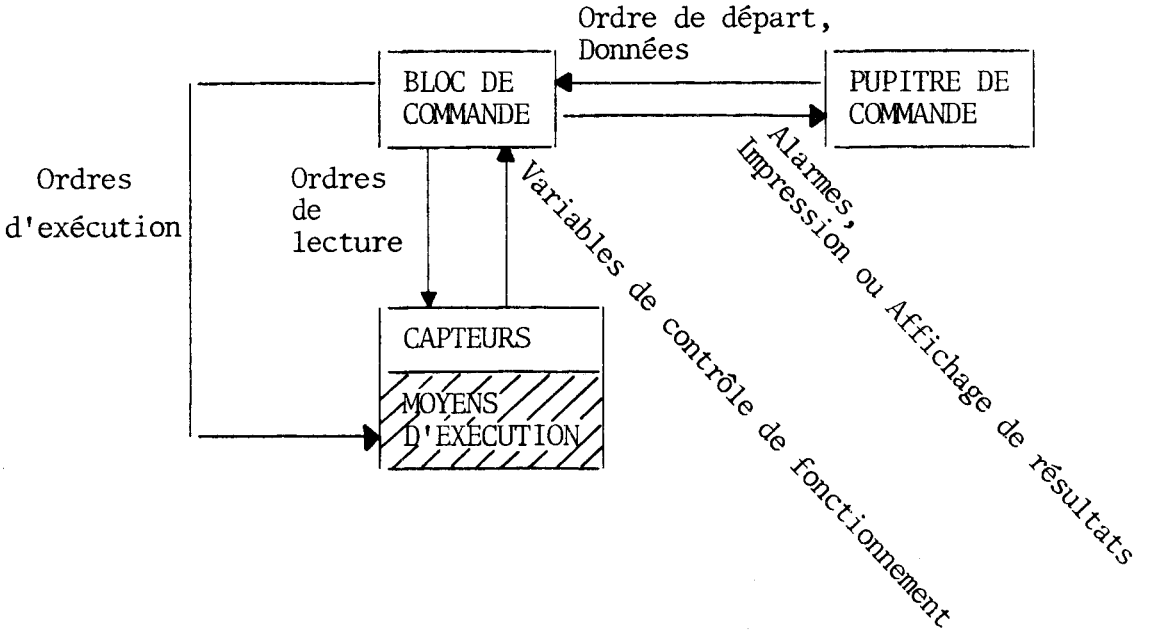
- du point de vue "hardware", d'une part certains systèmes peuvent être réalisés d'une façon équivalente en continu ou en "tout ou rien", d'autre part dans un même équipement peuvent cohabiter plusieurs technologies ;

- du point de vue "software", la différence entre systèmes séquentiels et systèmes programmés n'est pas toujours évidente.

Pour lever ces ambiguïtés nous appellerons systèmes programmés les systèmes dont le "programme" de fonctionnement contient des ordres d'exécution.

Dans le cadre de l'étude présentée, nous ne nous sommes intéressés qu'aux systèmes à commande "tout ou rien" séquentiels (asynchrones ou non) ou programmés.

La structure de tels automatismes peut être illustrée par le schéma ci-dessous



A partir du pupitre de commande l'opérateur déclenche le fonctionnement du système et fixe différents paramètres rationnels ou booléens : les données.

Dès qu'il a reçu l'ordre de départ, le bloc de commande déclenche en fonction des données les ordres d'exécutions des tâches à réaliser. Si besoin est, il "lit" et prend en compte les variables émises par les capteurs ; ces variables le renseignent sur le déroulement des tâches en cours d'exécution.

Le rôle des automaticiens est de réaliser :

- le pupitre de commande,
- le bloc de commande,
- les systèmes de capteurs,
- les connexions de ces éléments entre eux et avec les moyens d'exécution.

Ces réalisations se font à partir du cahier des charges, qui, dans le cas d'un système séquentiel, doit contenir :

- les tâches à exécuter,
- les données,
- tout ou partie de l'ensemble des variables de contrôle de fonctionnement,
- les enchaînements des tâches entre elles en fonction des données et des variables fournies par les capteurs,
- tout renseignement supplémentaire concernant la nature des moyens d'exécutions, les contraintes du milieu et les mesures de sécurité.

Les étapes successives du travail de réalisation d'un automatisme séquentiel sont les suivantes :

1) Prise en compte et vérification du cahier des charges,
2) Elaboration de l'organigramme de commande qui décrit les différentes séquences d'ordres d'exécution,
3) Elaboration de l'organigramme de processus qui permet de connaître à tout instant l'état du système,
4) Détermination et réduction des équations booléennes (dans le cas des systèmes asynchrones),
5) Câblage et implantation.

Les systèmes programmés ne font appel qu'aux points 1 et 5. Actuellement il existe déjà des programmes de détermination et de réduction des équations booléennes ; par contre le problème reste entier en ce qui concerne les points 1, 2 et 3.

Quelques essais de mise au point automatique des organigrammes de processus ont déjà eu lieu mais d'une part ils supposent la connaissance de toutes les variables d'entrée et de sortie du système étudié ce qui implique que le travail de conception est fort avancé et d'autre part leur utilisation pour des systèmes relativement complexes paraît difficile.

Nous nous sommes attachés à étudier les moyens d'automatiser au maximum les 3 premières étapes.

La prise en compte et la vérification du cahier des charges suppose l'existence d'une méthode systématique de description de ce cahier des charges. Au chapitre IV, nous développerons le langage DESAUCYK qui est une solution à ce premier problème.

A partir de la description du cahier des charges en DESAUCYK l'ordinateur devra élaborer les organigrammes de commande et de processus. Nous présentons au chapitre III le langage DESPROG qui permet de décrire facilement ces organigrammes. Chacun de ces langages répond à des impératifs différents.

DESPROG est un langage destiné aux automaticiens : la méthode employée pour décrire le fonctionnement d'un système se rapproche au maximum des méthodes utilisées habituellement par les automaticiens.

Par contre, les principes de conception du langage DESAUCYK sont à l'opposé de ceux du langage DESPROG. Ce sont les éléments de la réponse à la question suivante : comment procèderait un non-automaticien pour décrire un automatisme ? Le langage DESAUCYK doit donc permettre de décrire le fonctionnement de l'automatisme considéré aussi précisément que possible sans empiéter sur la conception ; il doit également se rapprocher au maximum du langage courant et permettre la définition de toutes les contraintes intervenant dans le choix des types de fonctionnement et de leur réalisation technologique.

Enfin, au chapitre IX nous proposerons des modifications aux langages DESPROG et DESAUCYK de manière à rendre possible la description de systèmes programmés.

Nous avons fait précéder l'étude de ces langages de deux chapitres qui ont pour but de définir plus précisément les éléments que ces langages ont à décrire.

chapitre I

DEFINITIONS ET PROPRIETES DES PROCESSUS

Le fonctionnement d'un automatisme se caractèrise par :

. un ensemble d'ordres d'exécution ou instructions,

. un ensemble de variables d'entrée (données ou variables de contrôle de fonctionnement),

. un ensemble de règles d'enchaînement des instructions. Ces enchaînements sont conditionnés par les valeurs des variables d'entrée.

Ce type de fonctionnement n'est pas particulier aux automatismes. On le retrouve notamment en microprogrammation.

Nous appellerons donc plus généralement PROCESSUS tout système qui présente un fonctionnement de ce type.

Les chapitres I et II seront consacrés à la définition des caractéristiques des processus. C'est à partir de ces caractéristiques que seront justifiées la forme et les possibilités d'exploitation du langage DESPROG (cf. chapitre III).

Dans les paragraphes 1 et 2 du présent chapitre, nous étudierons les éléments nécessaires à la détermination de la forme de DESPROG alors que le paragraphe 3 recense l'ensemble des critères de validité devant être vérifiés par un processus correctement défini.

I.1 - DEFINITIONS DE BASE

Un *processus* P est un 8-uple

$$P = \{VE, VS, RE, M, F, R, DE, S\}$$

où

- VE, VS, RE et M sont des ensembles finis dont les éléments sont appelés respectivement : *variables d'entrée, variables de sortie, registres internes* (au bloc de commande), *ordre d'exécutions*.

Un ordre d'exécution [1] peut être :

. une *commande externe*, c'est-à-dire une affectation de valeur à une variable de sortie

. une *commande interne*, c'est-à-dire un changement de registre précède éventuellement d'autres opérations (addition...)

. un *test* sur une variable d'entrée ou le contenu d'un registre

- F est un ensemble fini de relation booléennes associé à M tel que $\forall\, m_i \in M$;

  i) si m_i correspond à l'exécution d'une commande interne ou externe, une seule fonction $f_i \in F$ est associée à m_i,

  ii) si m_i correspond à l'exécution d'un test, deux fonctions $f_{i_1}, f_{i_2} \in F$ sont associées à m_i, chacune d'elles correspondant à un résultat possible du test.

- R est une partie finie de M×F×M, dont les éléments sont appelés *enchaînements* [1], telle que :

$$\forall\, m_i, m_j \in M,\ \forall \text{ la fonction } f_h \text{ de F associée à } m_i$$
$$(m_i, f_h, m_j) \in R \text{ si et seulement si}$$

  i) m_i étant exécuté, si la relation f_h=1 est vraie, alors m_j est exécutable,

  ii) $\nexists\, m_\ell \in M$, $f_k \in F$ tels que

$$(m_i, f_h, m_\ell) \in R \text{ et } (m_\ell, f_k, m_j) \in R$$

---

(1) Ces éléments sont définis plus précisément au paragraphe I.2

- DE est une partie de M, vérifiant :

$$\forall\, m_i \in DE, \nexists\, m_j \in M,\ f_k \in F \text{ et } (m_j, f_k, m_i) \in R$$

Les éléments de DE sont appelés *ordres de départ*.

- T est une partie finie de M vérifiant :

$$\forall\, m_i \in T, \nexists\, m_j \in M,\ f_k \in F \text{ t.q. } (m_i, f_k, m_j) \in R$$

Les éléments de T sont appelés *ordres terminaux*.

Remarque 1 : Il peut arriver que $T=\emptyset$, par exemple, s'il existe un ordre m_i correspondant à l'exécution d'un test, tel que les fonctions f_{i_1} et f_{i_2} de F sont associées à m_i vérifient :

- $\exists\, m_j$ tel que $(m_i, f_{i_1}, m_j) \in R$

- $\nexists\, m_k \in M$ tel que $(m_i, f_{i_2}, m_k) \in R$.

Pour éviter cet inconvénient, nous ajouterons alors à M un ordre spécial noté fin, et nous placerons dans R le triplet :

$$(m_i, f_{i_2}, \text{fin})$$

Remarque 2 : Deux ordres m_i et m_j différents peuvent correspondre à l'exécution d'une même commande ou d'un même test mais à des instants différents du déroulement du processus étudié.

Remarque 3 : Lorsqu'un processus est commandé manuellement, il existe une variable d'entrée particulière correspondant à l'impulsion de départ. La valeur de cette variable d'entrée est testée systématiquement, nous l'appellerons *variable de commande*. Nous noterons m_0 l'élément de M correspondant à ce test, et nous associerons à m_0, les fonctions f_{0_1} et f_{0_2} de F de telle sorte que :

$$- (m_0, f_{0_1}, m_0) \in R$$

$$- \forall\, m_i \in D,\ (m_0, f_{0_2}, m_i) \in R.$$

Remarque 4 : Au début du fonctionnement du processus, il est nécessaire de connaître la valeur des variables d'entrée et de sortie du processus ainsi que les contenus des registres internes.

Nous appellerons *conditions initiales* ces valeurs.

En fait, il est possible que la valeur d'une variable d'entrée booléenne [1] au début du fonctionnement du processus soit indifférente. Par contre les variables de sortie, les variables d'entrée rationnelles [1] ainsi que les contenus de registres doivent avoir une valeur déterminée.

En général, les registres sont remis à zéro, cependant il faut excepter le cas des registres contenant des paramètres fixés avant le lancement du processus par l'opérateur.

Pour préciser cette définition générale des processus, nous définissons les éléments suivants :

l'ensemble VE des variables d'entrée peut être partitionné en deux sous-ensembles:

VEB ensemble des *variables d'entrée booléennes*,
VER ensemble des *variables d'entrée rationnelles*.

$\forall p_i \in$ VER, p_i prend sa valeur dans l'ensemble des nombres rationnels $\mathbb{Q}$. Ces variables sont prises en compte par le bloc de commande et interviennent dans les calculs ou dans les tests.

l'ensemble VS des variables de sortie est un ensemble de variables booléennes. C'est le bloc de commande qui impose aux éléments de VS leur valeur.

Pour l'instant, nous supposerons qu'il n'existe pas de variables de sortie rationnelles

Les variables de VS peuvent aussi correspondre à des impulsions. La nécessité d'une distinction entre variable de sortie booléenne et impulsion interviendra dans la réalisation de l'organigramme de processus (cf. chapitre II).

l'ensemble RE des registres internes au bloc de commande peut être partitionné en deux sous-ensembles :

REB ensemble des *registres à contenu booléen*
RER ensemble des *registres à contenu rationnel*.

Pour tout $r_i \in$ RE nous noterons $C(r_i)$ l'ensemble des *valeurs chargeables* dans r_i.

---

(1) Ces notions sont définies ci-dessous

Ainsi, si $r_i \in$ REB, alors $C(r_i) = \{0,1\}$

si $r_i \in$ RER, alors le cardinal de $C(r_i)$ est égal à la capacité de r_i (un registre à n bits a une capacité de 2^n).

Avec ces notations

$$C(RE) = \bigcup_{r_i \in RE} C(r_i)$$ constitue l'*ensemble des états internes* du processus.

Nous noterons C l'ensemble des *commandes* émises par un processus. C peut être partitionné en trois sous-ensembles C'_0, C''_0 et TB.

i) C'_0 est l'ensemble des *affectations de valeurs* aux variables de sortie. A tout $s_i \in$ VS, peuvent correspondre deux affectations de valeurs

$$(s_i=0) \text{ et } (s_i=1)$$

Pour simplifier, nous noterons s_i l'affectation $(s_i=1)$ et $\overline{s_i}$ l'affectation $(s_i=0)$.

Remarquons que card C'_0=2 × card (VS) et que chaque élément de C'_0 correspond à l'exécution d'une opération extérieure au bloc de commande.

ii) C''_0 est l'ensemble des commandes internes. Quel que soit l'objet de ces opérations chacune d'elle sera accompagnée du chargement d'une valeur dans un registre. Ainsi si $c''_j \subset C''_0$ correspond à l'exécution d'une addition,il existera un registre r_k où sera stocké le résultat de l'opération.

Nous poserons $C_0 = C'_0 \cup C''_0$.

iii) TB est l'ensemble des tests exécutables sur la valeur des éléments de VE ou C(RE).

Nous appellerons *test* la vérification d'une relation booléenne de la forme : $g_i=1$ où g_i est une fonction booléenne du type suivant

((1° terme) (comparateur) (2° terme))

les deux termes doivent être tous deux du même type, soit booléen, soit rationnel.

Dans le premier cas, le comparateur sera obligatoirement un comparateur d'égalité.

Dans le second cas, ce pourra être le comparateur :

d'égalité	=
de supériorité	>
d'infériorité	<
d'inégalité	≠

Nous noterons TB(VEB), TB(VER) et TB(C(RE)) les ensembles de tests sur les valeurs des éléments respectivement de VEB, VER et C(RE).

Précisons maintenant la notion d'ordre d'exécution à l'aide de la notion de commande.

Une même commande est susceptible d'être exécutée plusieurs fois lors du déroulement d'un processus ; il est donc nécessaire d'affecter à chacune de ses occurences un indice qui permette de la différencier des autres occurences.

D'après la définition d'un processus et compte tenu de la remarque 3, nous pouvons considérer que M est une partie finie de $C \times \mathbb{N}$.

Si $c_j \in C_0$ nous noterons g_i une variable booléenne associée à c_j telle que $g_i=1$ dès que c_j est exécutée.

Si $c_j \in TB$ nous noterons g_i une variable booléenne associée à c_i telle que $g_i=1$ si le résultat du test c_j est positif et $g_i=0$ si le résultat est négatif.

La définition de F donnée plus haut peut alors être précisée de la manière suivante :

$$\forall m_i \in M : m_i = (c_j, n_i) \in C \times \mathbb{N}.$$

i) Si $c_j \in C_0$: la relation $f_i \in F$ associée à m_i sera de la forme $(g_i=1)$ et elle sera vraie dès que m_i sera exécutée.

ii) Si $c_j \in TB$; les relations f_{i_1} et f_{i_2} associées à m_i seront respectivement de la forme $(g_i=1)$ et $(g_i=0)$ chacune de ces fonctions correspondant à un résultat différent du test c_j.

La nature des éléments de M et de F étant précisée, il est plus facile de comprendre la signification de l'ensemble R. Chaque élément de R correspond à un enchaînement d'ordres. Les enchaînements d'ordres définis par R déterminent le déroulement du processus considéré.

$\forall\, m_i, m_j \in M$, $f_k \in F$ tels que $(m_i, f_k, m_j) \in R$

m_i sera appelé *prédécesseur direct de* m_j et

m_i sera appelé *successeur direct de* m_i.

Pour simplifier la notation, nous remplacerons

(m_i, f_k, m_j) avec $m_i = (c_h, n_i)$, par :

i) (m_i, m_ℓ) sans préciser f_k si $c_h \in C_0$

ii) (m_i, P, m_ℓ) si $c_h \in TR$ et $f_k = (g_i = 1)$

iii) (m_i, N, m_ℓ) si $c_h \in TR$ et $f_k = (g_i = 0)$.

Nous pouvons maintenant définir à partir de R un certain nombre de fonctions, qui étendent la notion d'enchaînement.

Nous appellerons "fonction successeur direct" l'application δ *de* F×M *dans* P(M) *définie par :*

$\forall\, f_j \in F$, $\forall\, m_i \in M$

$$\boxed{\delta(f_j, m_i) = \{m_k \in M \mid (m_i, f_j, m_k) \in R\}}$$

Cette application peut être étendue dans P(M).

Pour cela associons à tout Z de P(M) l'ensemble :

$$\Pi F(Z) = \{k \mid k = \prod_{m_i \in Z} f_i\}$$

tel que si $Z = \{m_{i_1}, m_{i_2} \ldots m_{i_n}\}$

avec $\forall\, j \in [1,n]$ $f_{i_j} \in F$ est associé à m_{i_j}

$$k = f_{i_1} \times f_{i_2} \times \ldots \times f_{i_n}$$

"×" indiquait ici l'opérateur booléen.

$\forall k \in \Pi F(Z)$, k sera donc un produit booléen d'éléments de F.

Si $\forall m_{i_j} \in Z$, m_{i_j} correspond à un élément de C_0

$$\text{card } (\Pi F(Z)) = 1.$$

Dans le cas contraire, c'est-à-dire lorsqu'il existe au moins un élément de Z correspondant à l'exécution d'un test : card $(\Pi F(Z)) = 2^p$, p étant le nombre d'ordre correspondant à l'exécution d'un test.

Nous noterons :

$$\Pi F = \bigcup_{Z \in P(M)} \Pi F(Z).$$

La fonction "successeurs multiples directs" se définit alors comme une application de $\Pi F \times P(M)$ *dans* $P(M)$:

$\forall Z \in P(M)$, $k \in \Pi F(Z)$:

$$\Delta(k,Z) = \{m_i \mid \exists\, m_j \in Z, \exists\, f_h \in F, \textit{ t.q. } k = f_h \times k' \textit{ et } (m_j, f_h, m_i) \in R\}$$

Si card (Z) = 1 donc si $Z = \{m_i\}$ alors

$$\Delta(k,Z) = \delta(f_j, m_i) \text{ avec } k = f_j.$$

D'une façon analogue, nous définirons la fonction "prédécesseur direct" application δ^{-1} *de* M *dans* P(M) *par* :

$\forall m_i \in M$:

$$\delta^{-1}(m_i) = \{m_j \in M \mid \exists\, f_h \in F : (m_j, f_h, m_i) \in R\}$$

et la fonction "prédécesseur multiple direct" application Δ^{-1} *de* P(M) *dans* P(M) *par* :

$\forall Z \in P(M)$:

$$\Delta^{-1}(Z) = \{w \in P(M) \mid \exists\, k \in \Pi F(w) : Z = \Delta(k,w)\}$$

Nous utiliserons également dans la suite la fonction *"successeur possible direct"*, application de M dans $P(M)$ définie par :

$$\forall\, m_i \in M : \quad \boxed{d(m_i) = \{m_j \in M \mid \exists\, f_h \in F \text{ tel que } (m_i, f_h, m_j) \in R\}}$$

et la fonction *"successeur multiple possible direct"*, application de $P(M)$ dans lui-même définie par :

$$\forall\, Z \in P(M) : \quad \boxed{D(Z) = \bigcup_{m_i \in R} d(m_i) = \bigcup_{k \in \Pi F(Z)} \Delta(k,Z)}$$

Si $\forall\, m_i \in Z$, $m_i = (c_j, n_j)$ avec $c_j \in C_0$,

$$D(Z) = \Delta(k,Z) \text{ car card } \Pi F(Z) = 1.$$

de même $\forall\, m_i \in M$, $m_i = (c_j, n_j)$, $c_j \in C_0$ entraîne que $d(m_i) = \delta(f_i, m_i)$, f_i étant unique.

Jusqu'à présent, nous nous sommes restreints aux successions directes, or l'étude complète d'un processus nécessite la connaissance de relations de succession beaucoup plus "lointaines" notamment pour la construction du graphe d'état (cf. chapitre II).

Pour cette raison, nous introduirons un certain nombre de notions supplémentaires :

Nous appellerons *suite* tout élément de ΠF^* (ΠF^* étant le monoïde libre construit sur ΠF).

Nous définissons alors la fonction "successeur indirect".

Δ^+, *application de* $\Pi F^* \times P(M)$ *dans* $P(M)$ *définie par*

$\forall\, Z \in P(M)$, $\alpha \in \Pi F^*$ *tel que, si nous notons "." l'opérateur de concaténation sur* ΠF^*

$$\alpha = k_1.k_2. \text{ — } .k_p \text{ avec } k_1 \in \Pi F(Z_j)$$

$$\boxed{\Delta^+(\alpha,Z) = \bigcup_{j=1}^{p} \Delta(k_p, (\Delta(k_{p-1}, \text{ — } (\Delta(k_1,Z)) \text{ —})}$$

C'est-à-dire qu'il existe $Z_1, Z_2, \ldots, Z_p \in P(M)$ tels que :

$$- Z_1 = Z,$$

$$- Z_{j+1} = \Delta(k_j, Z_j) \text{ pour } j=1,2,\ldots,p-1.$$

En étendant la notion de successeur direct nous noterons

$$D^+(Z) = \{m_j \mid m_j \in M, \exists\, \alpha \in \Pi F^* \text{ tel que } m_j \in \Delta^+(\alpha, Z)\}$$

Pour un élément Z de P(M) quelconque, $D^+(Z)$ représente l'ensemble des ordres dont l'exécution suit celle des éléments de Z.

I.2 - REDUCTION D'UN PROCESSUS

Nous abordons dans ce paragraphe la partie essentielle du chapitre I. Pour simplifier notre exposé, nous allons d'abord donner une représentation graphique d'un processus.

A chaque processus P nous pouvons associer :

un graphe orienté $G(P) = \{X, \Gamma\}$.

L'ensemble des sommets X est en relation biunivoque avec l'ensemble des commandes M et l'ensemble des arcs. Γ partie de X^2 est en relation biunivoque avec R, c'est-à-dire que $\forall\, x_i, x_j \in X$ en relation biunivoque avec m_i, $m_j \in M$ tels que

$$(m_i, f_h, m_j) \in R$$

$$\gamma = (x_i, x_j) \in \Gamma$$

et à γ sera associée la fonction f_h. x_i sera l'origine et x_j l'extrémité de l'arc γ.

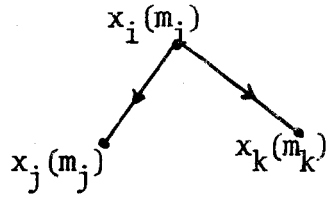
D'un point de vue descriptif, en reprenant les simplifications de notation définies plus haut

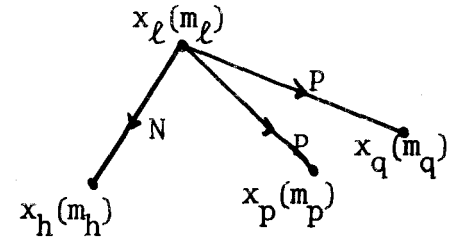
- si x_i correspond à $m_i = (c_h, n_i)$ avec $c_h \in C_0$, il est alors inutile d'affecter une fonction booléenne à tout arc issu du sommet x_i puisque cette

fonction est unique et égale à $(g_i=1)$

- si x_i correspond à $m_i = (c_h, n_i)$ avec $c_h \in TB$, alors nous affecterons la lettre P à tout arc correspondant à la fonction $(g_i=1)$ et la lettre N à tout arc correspondant à la fonction $(g_i=0)$.

Exemples :





Dès que le processus étudié atteint une certaine complexité, le graphe associé devient inexploitable. Il peut être alors intéressant de scinder l'étude du processus en plusieurs études plus restreintes.

C'est ce que nous entendons par "réduction d'un processus" c'est-à-dire la simplification de son graphe représentatif, ce qui revient à réduire les ensembles R et M.

Une méthode classique pour simplifier un graphe est la recherche dans ce graphe de sous-graphes connexes que l'on élimine du graphe initial et que l'on étudie ensuite séparément ; cependant, pour des raisons d'ordre technologique il n'est pas possible de décomposer le graphe associé à un processus d'une façon quelconque. Nous avons donc été amené à introduire les notions de séquence linéaire, processus secondaire, processus irréductible et processus autonome et à envisager ainsi une analyse des processus en 3 étapes :

- décomposition du processus en processus autonome

- décomposition des processus autonomes en processus irréductibles

- étude de chaque processus irréductible eux-mêmes simplifiables par élimination des séquences linéaires et des processus secondaires.

La plupart du temps les processus à étudier sont irréductibles et indécomposables en processus autonome, aussi définirons-nous d'abord les notions de séquences linéaires et de mode secondaire, puis après avoir introduit les notions de processus

irréductible et de processus autonome, nous donnerons une généralisation de la notion de processus, permettant de décrire le fonctionnement d'un processus complexe en considérant les processus irréductibles qui le composent comme des éléments commandés.

I.21 - 1ère notion : Séquences et séquences linéaires

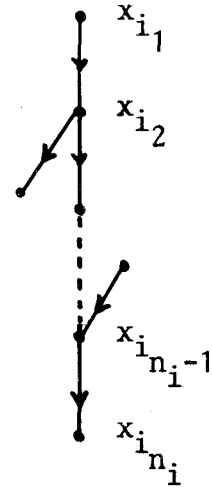
Nous appellerons *"séquence"* un ensemble

$$s_i = \{m_{i_1}, m_{i_2}, \ldots, m_{i_{n_i}}\}$$

d'éléments de M tel que

$$\forall\, j \in [1, n_{i-1}] : \exists\, f_h \in F \text{ tel que } (m_{i_j}, f_h, m_{i_{j+1}}) \in R$$

Graphiquement, une séquence correspond à un chemin



x_{i_j} est en relation biunivoque avec m_{i_j}

m_{i_1} sera dit *origine* et $m_{i_{n_i}}$ *extrémité* de la séquence.

Nous appellerons *"séquence linéaire"* une séquence $\sigma_i = \{m_{i_1}, .., m_{i_{n_i}}\}$ qui vérifie les quatre conditions suivantes :

i) $\forall\, j \in [1, n_i - 1]$, $\nexists\, f_h \in F$, $m_s \in M$, $m_s \neq m_{i_{j+1}}$ tels que

$$(m_{i_j}, f_h, m_s) \in R$$

ii) $\forall\, j \in [1, n_i - 1]$, $\forall\, f_h \in F$ tels que $(m_{i_j}, f_h, m_{i_{j+1}}) \in R$

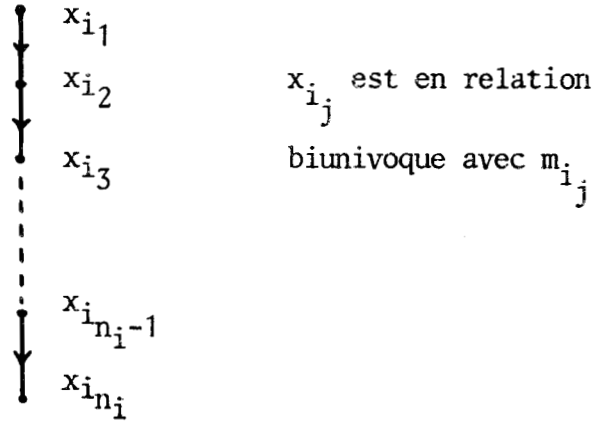
$$\nexists\, m_s \in M,\ m_s \neq m_{i_{j+1}},\ (m_{i_j}, f_h, m_s) \in R$$

iii) $\forall j \in [Z,n_i]$, $\nexists m_s \in M \mid \sigma_i, f_h \in F$ tels que

$$m_s \neq m_{i_{j-1}} \text{ et } (m_s,f_h,m_{i_j}) \in R$$

iv) $m_{i_{n_i}} = (c_\mu,n_\mu)$ avec $c_\mu \in C_0$.

Une séquence linéaire correspond graphiquement à un chemin dont chaque sommet est l'extrémité d'un arc unique et l'origine d'un autre arc unique. De plus un seul arc aura le sommet origine du chemin comme origine et un seul arc aura le sommet extrémité du chemin comme extrémité



Nous pouvons étendre la notion de séquence linéaire de la manière suivante :

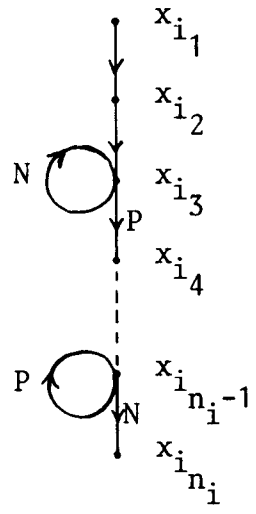
i) Si une séquence $s_i = \{m_{i_1} \ldots m_{i_j} \ldots m_{i_{n_i}}\}$ répond aux quatre conditions définissant une séquence linéaire sauf pour un certain nombre d'ordres $m_{i_{j_1}} \ldots m_{i_{j_k}}$ tels que $\forall u \in [1,k], m_{i_{j_u}} = (c_h,n_{i_{j_u}})$ avec $c_h \in TB$

$$\text{et } (m_{i_{j_u}},x,m_{i_{j_u}}) \in R \qquad x,y \in \{P,N\}$$

$$(m_{i_{j_u}},y,m_{i_{j_u+1}}) \in R \qquad x \neq y$$

alors s_i sera considérée comme une séquence linéaire.

Ce que nous pouvons illustrer par le graphe ci-dessous



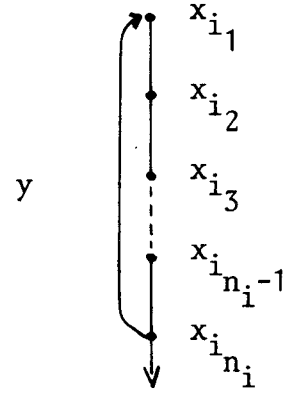
ii) Si une séquence $s_i = \{m_{i_1} \ldots m_{i_{n_i}}\}$ répond aux quatre conditions linéaires ou à la propriété définie ci-dessus excepté pour $m_{i_{n_i}}$ qui vérifie :

$$m_{i_{n_i}} = (c_h, n_{i_{n_i}}),\ c_h \in TB \text{ et}$$

$$(m_{i_{n_i}}, y, m_{i_1}) \in P,\ y \in \{P,N\}$$

alors s_i sera considéré comme une séquence linéaire.

Ce que nous pouvons illustrer par le graphe ci-dessous



x_{i_j} est en correspondance biunivoque avec m_{i_j}

Afin de simplifier la description du déroulement du processus et pour chaque séquence linéaire

$$\sigma_i = \{m_{i_1}, m_{i_2} \ldots m_{i_{n_i}}\}$$

nous pouvons construire les nouveaux ensembles d'enchaînements suivants :

$$\text{i)}\quad R' = (R \setminus (\bigcup_{m_k \in \delta^{-1}(m_{i_1})} (m_k, f_{h_k}, m_{i_1}))) \bigcup_{m_k \in \delta^{-1}(m_{i_1})} (m_k, f_{h_k}, \sigma_i)$$

c'est-à-dire que chaque enchaînement de la forme

(m_k, f_{h_k}, m_{i_1}) est remplacé par un nouvel enchaînement (m_k, f_{h_k}, σ_i)

$$\text{ii)}\quad R'' = R' \setminus \bigcup_{j \in [2, n_i]} (m_{i_j}, f_{h_j}, m_{i_{j+1}})$$

c'est-à-dire que l'on retire de R' les enchaînements constituant la séquence linéaire σ_i

$$\text{iii)}\quad R''' = (R'' \setminus \bigcup_{m_k \in d(m_{i_{n_i}})} (m_{i_{n_i}}, f_{i_{n_i}}, m_k)) \bigcup_{m_k \in d(m_{i_{n_i}})} (\sigma_i, f_i, m_k)$$

c'est-à-dire que chaque enchaînement de la forme

$(m_{i_{n_i}}, f_{i_{n_i}}, m_k)$ est remplacé par (σ_i, f_i, m_k)

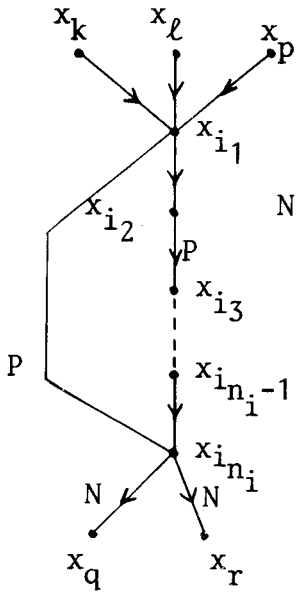
il faut remarquer que f_i est une nouvelle fonction booléenne associée à σ_i ; f_i est unique, σ_i étant assimilée à un élément de $C_0 \times \mathbb{N}$.

Dans le cas où $m_{i_{n_i}} = (c_h, n_{i_{n_i}})$, $c_h \in TB$ et

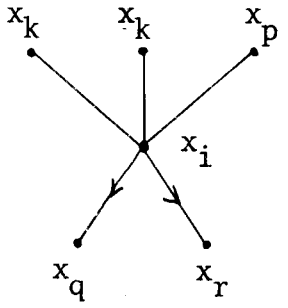
$(m_{i_{n_i}}, x, m_{i_1}) \in R$

cet enchaînement sera éliminé de R'' et une fonction f_i unique correspondra à σ_i.

Nous pouvons graphiquement illustrer cette simplification du processus étudié comme suit



est remplacé par



Remarquons que dans un même processus, il peut exister deux séquences linéaires σ_i et σ_j telles que

- $\sigma_i = \{m_1, \ldots, m_p\}$ et $\sigma_j = \{m'_1, \ldots, m'_p\}$
- $\forall k \in [1,p] \; \exists \; c_h \in C, \exists \; p_k, q_k \in \mathbb{N} : m_k = (c_h, p_k)$

$$m'_k = (c_h q_k)$$

Nous pouvons alors représenter σ_i et σ_j par des doublets de la forme

$$\sigma_i = (\tau_r, n_i), \; \sigma_j = (\tau_r, n_j)$$

avec $\tau_r = \{c_{j_1}, c_{j_2}, \ldots, c_{j_n}\}$ ensemble des commandes associées à σ_i et σ_j.

Par analogie avec la définition de M, nous pouvons construire un ensemble M'_{σ_i} pour une séquence linéaire, tel que

$$M'_{\sigma_i} = (M \setminus \bigcup_{j=1}^{n_i} m_{i_j}) \cup \sigma_i.$$

Nous noterons $M'' = \bigcup_{\sigma_i \in \Sigma} M'_{\sigma_i}$ avec Σ ensemble des séquences linéaires. Ainsi auront été éliminés les ordres internes aux séquences linéaires et ajoutés des ordres correspondant à chacune de ces séquences linéaires.

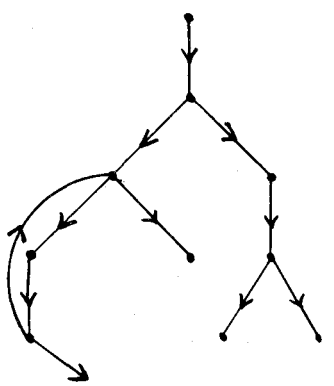
Le processus P pourra alors être représenté graphiquement par un graphe $G''(X'',\Gamma'')$ tel que X" soit en relation biunivoque avec M" et Γ'' en relation biunivoque avec R^n.

Dans ce graphe il n'y aura pas de chemins tels que

$$(x_{i_1},x_{i_2}) \in \Gamma'', \ (x_{i_2},x_{i_3}) \in \Gamma'' \quad \text{et tel que}$$

$\nexists\ (x_{i_2},x_{i_4}) \in \Gamma''$ avec $x_{i_4} \neq x_{i_3}$,

C'est-à-dire que l'on aura un graphe de la forme



Nous verrons au chapitre III que le regroupement en séquences linéaires simplifie considérablement la description d'un processus et le traitement machine de cette description. Mais le processus n'est en aucun cas modifié et reste défini par les mêmes éléments.

I.22 - 2ème Notion : Processus secondaire

Nous appellerons *processus secondaire* d'un processus quelconque

$P = \{VE,VS,RE,M,F,R,DE,T\}$ un processus

$Q_{ij} = \{VE_{ij},VS_{ij},RE_{ij},M_{ij},F_{ij},R_{ij},DE_{ij},T_{ij}\}$

tel que
- $VE_{ij} \subset VE$
- $VS_{ij} \subset VS$
- $RE_{ij} \subset RE$
- $DE_{ij} = \Delta(k,Z_i)$ avec $Z_i = \{m_{i_1} \ \ldots \ m_{i_{n_i}}\} \in P(M)$ et $k \in \Pi F(Z_i)$
- $T_{ij} = Z_j = \{m_{j_1} \ \ldots \ m_{j_{n_j}}\} \in P(M)$

Z_i et Z_j sont liés par les propriétés suivantes :

i) $\forall\, m_{i_k} \in Z_i$, $\exists\, f_k \in F$ tel que

$$\delta(f_1,m_{i_1}) = \delta(f_2,m_{i_2}) = \ldots = \delta(f_{n_i},m_{i_{n_i}}) = DE_{ij}$$

c'est-à-dire que $k = f_1 \times f_2 \times \ldots \times f_{i_{n_i}}$

ii) Il n'existe pas de séquences dont l'origine appartient à DE_{ij} et dont l'un des sommets n'appartient pas à Z_j

iii) $\exists\, !\ k \in \Pi F(Z_j)$ tel que $\Delta(k,Z_j) \neq \emptyset$ et

$k = f_1 \times f_2 \times \ldots \times f_{n_j}$ telles que

$$\delta(f_1,m_{j_1}) = \delta(f_2,m_{j_2}) = \ldots = \delta(f_{n_j},m_{j_{n_j}}) \neq \emptyset$$

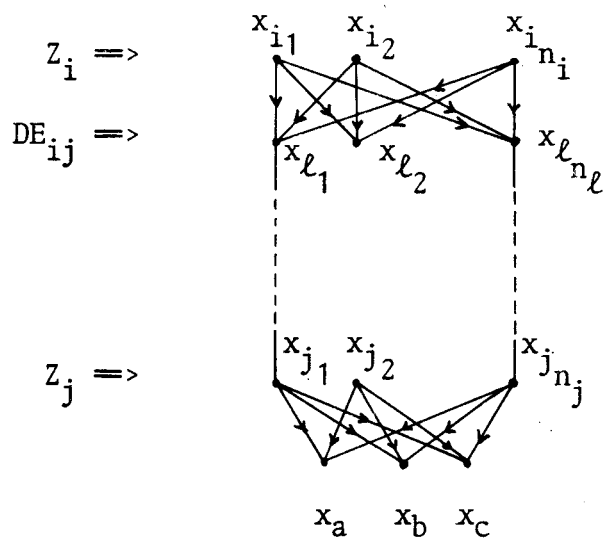
iv) $\forall\, s_j$ une séquence dont l'origine appartient à DE et dont l'extrémité appartient à Z_j, $\exists\, m_{i_k} \in Z_i$, $\exists\, m_s \in DE_{ij}$ tels que

$s_j = \{m_{j_1} \ldots m_{i_k}, m_s \ldots m_{j_\ell}\}$.

Il en résulte que

- $M_{ij} = M'_{ij} \cup \text{fin}$ avec $M'_{ij} \subset M$
- $F_{ij} \subset F$
- $R_{ij} = R'_y \bigcup_{k=1}^{n_j} (m_{j_k}, f_k, \text{fin})$ avec $R'_y \subset R$.

Le graphe G_{ij} associé à Q_{ij} est un sous-graphe connexe de G(P), ce qui peut s'illustrer par :



Remarque 1 : Pour définir complètement un processus secondaire, il est nécessaire de lui adjoindre un ordre de départ m_0. Il y a deux cas possibles :

i) $Z_i = \{m_i\}$, $m_i = (c_k, n_i)$ avec $c_k \in TB$; posons alors : $m_0 = (c_k, n'_0)$ et adjoignons à R_{ij} : (m_0, f_1, m_0) et $\bigcup_{m_k \in DE_{ij}} (m_0, f_2, m_k)$.

ii) $\exists\, m_i \in Z_i$, $m_i = (c_k, n_i)$ avec $c_k \in C_0$: dans ce cas, il faudra introduire un ordre m_0 supplémentaire correspondant à une variable de départ fictive. On adjoindra à R_{ij} : (m_0, f_1, m_0) et $\bigcup_{m_k \in DE_{ij}} (m_0, f_2, m_k)$.

Cette remarque restera valable dans la suite de ce paragraphe.

Remarque 2 : Si nous appelons processus cyclique un processus dont le déroulement est conçu de manière à ce que lorsqu'après exécution de la dernière commande "fin" le bloc de commande se retrouve dans les conditions initiales, nous pouvons remarquer qu'un processus secondaire n'est pas nécessairement un processus cyclique. Cette propriété sera utilisée dans les chapitres III et IV.

Pour simplifier la description du fonctionnement du processus initial, nous pouvons construire pour chaque processus secondaire, un nouvel ensemble d'enchaînements :

$$R' = (R \setminus (R_{ij} \bigcup_{m_{j_\ell} \in Z_j} \bigcup_{m_j \in \Delta(k, Z_j)} (m_{j_\ell}, f_\ell, m_s))) \cup R(Z_i) \cup R(Z_j)$$

avec : $R(Z_i) = \bigcup_{m_{i_k} \in Z_i} (m_{i_k}, f_k, Q_{ij})$

et $R(Z_j) = \bigcup_{m_s \in \Delta(k, Z_j)} (Q_{ij}, f_{ij}, m_s)$.

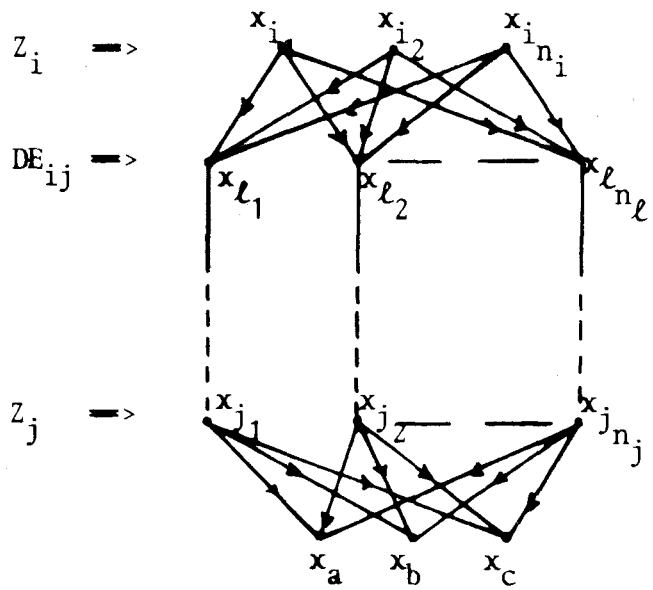
R' ne contient pas les enchaînements apparaissant dans Q_{ij} ni les enchaînements relatifs à $\Delta(k, Z_j)$ mais contient des enchaînements supplémentaires relatifs à Q_{ij}, qui est alors considéré comme un ordre.

De la même façon nous pouvons construire M'

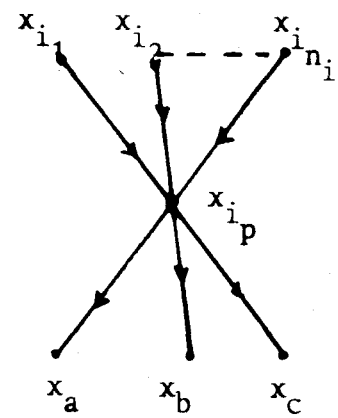
$$M' = (M \setminus M_{ij}) \cup Q_{ij}.$$

Le fonctionnement simplifié du processus P étudié pourra être représenté graphiquement par un graphe G'(X',Γ') tel que X' soit en relation biunivoque avec M' et Γ' en relation biunivoque avec R'.

Ce que nous pouvons illustrer par les graphes ci-dessous



se réduit à



La décomposition d'un processus en processus secondaires permet de simplifier considérablement sa description. Nous utiliserons cette propriété pour définir DESPROG et DESAUCYK.

I.23 - 3ème Notion : Processus irréductible

Remarque préalable : Soit S l'ensemble des séquences correspondant au fonctionnement d'un processus P.

Nous noterons * l'opération concaténation de séquences sur l'ensemble S.

Cette opération a les propriétés suivantes :

i) elle est associative :

$$(s_i * s_j)* \ s_k = s_i * (s_j * s_k)$$

ii) la séquence $s_\emptyset = \emptyset$ est l'élément neutre de la concaténation.

Nous dirons alors qu'un processus P est *réductible* si S étant l'ensemble des séquences définies par le déroulement de P, S est partitionnable en sous-ensembles S_1, S_2 ... S_r tels que

i) $\forall$ i,j $\in$ [1,r] card $(S_i \cap S_j) < \varepsilon$, $\varepsilon \in \mathbb{N}$ donné, c'est-à-dire que deux ensembles de séquences ne peuvent avoir en commun qu'un nombre donné de séquences.

ii) $\forall$ i $\in$ [1,r], S_i est tel que $\forall$ $s_j \in S_i$ on peut construire au moins une concaténation de séquences

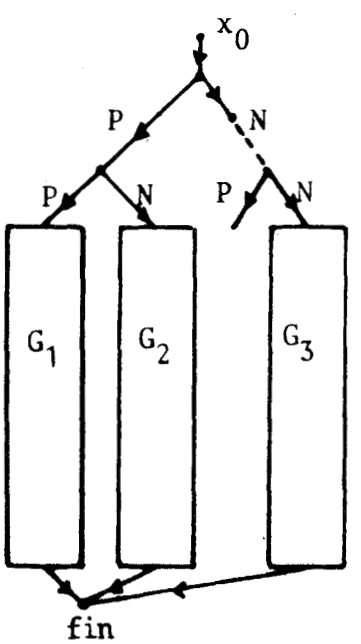
$w_j^* s_j^* w_j'$, avec w_j, $w_j' \in S_i^*$ telle que

- le premier ordre de w_j est m_0
- le dernier ordre de w_j' est fin
- si w_j n'existe pas cela implique que s_j commence par m_0
- - w_j' - - - - - - se termine par fin

iii) $\forall$ i $\in$ [1,r], S_i n'est pas réductible c'est-à-dire ne vérifie pas les conditions i et ii.

Dire qu'un processus P est réductible signifie que le déroulement de P est décomposable en plusieurs déroulements nettement différents c'est-à-dire que l'équipement qui exécute P est susceptible,en fait, d'exécuter plusieurs fonctions différentes, chaque fonction correspondant à un processus irréductible de P.

Un processus P réductible est représenté par un graphe du type suivant :



G_1, G_2, G_3 correspondent à des processus *irréductibles*

Remarque : En fait, un processus irréductible ne correspond pas toujours à un sous-graphe connexe du graphe associé au processus réductible correspondant.

I.24 - 4ème Notion : Processus Autonome

Nous dirons qu'un processus P est décomposable en *processus autonomes* si l'ensemble S des séquences correspondant à P est partitionnable en sous-ensembles $S_1, S_2, \ldots, S_r$ tels que

i) $\forall\, i, j \in [1,r], i \neq j, \nexists\, m_k \in S_i, \nexists\, m_\ell \in S_j$ tels que

$$m_k = m_\ell \in C_0 \times \mathbb{N}$$

c'est-à-dire qu'à l'exception des tests, une même commande ne peut apparaître dans deux séquences différentes

ii) $\forall\, k \in [1,r]$ les ordres apparaissant dans S_k s'enchaînent entre eux de manière à pouvoir être représenté par un graphe connexe, c'est-à-dire que l'on peut faire correspondre à S_k un processus P_k

iii) Toute concaténation de séquence du processus P d'origine m_0 et d'extrémité fin
s'écrira : $w_1 * w_2 * \ldots * w_s$

où $\forall\, j \in [1,s]$, w_j est une concaténation de séquences appartenant à un seul ensemble S_k c'est-à-dire $\nexists\, s_i, s_\ell \in S$ telles que $w_j = w_j' * s_i * w_j'' * s_\ell * w_j'''$ avec $s_i \in S_r$, $s_\ell \in S_q$ et $S_r \neq S_q$.

De plus, si w_j est une concaténation de séquences appartenant à S_r et w_{j+1} une concaténation de séquence appartenant à S_q, w_{j+1} aura pour origine un micro-ordre de départ du processus P_q associé à S_q.

Les processus $P_1, P_2, \ldots, P_r$ correspondant aux ensembles de séquences $S_1, S_2, \ldots, S_r$ seront appelés *processus autonomes* du processus P.

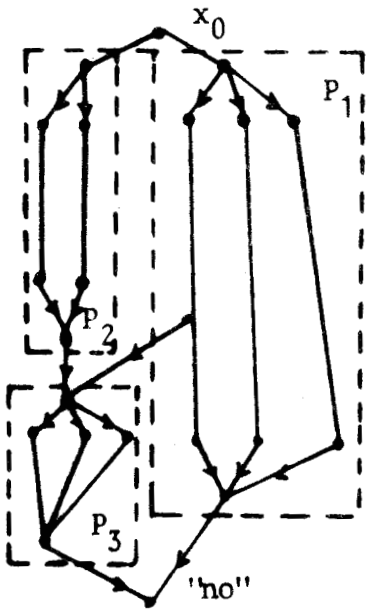
Remarque 1 : Un processus autonome ne peut constituer un processus secondaire d'un autre processus autonome (par la propriété iii).

Un processus autonome peut provoquer le départ d'un autre processus autonome (par la propriété iii).

Un processus autonome n'est pas obligatoirement un processus secondaire du processus initial.

Un processus autonome peut être réductible.

Illustration graphique



P_1, P_2 et P_3 constituent des processus autonomes

P_1 et P_2 ne peuvent constituer deux processus autonomes (cf. propriétés iii des processus autonomes)

Remarque 2 : Un processus autonome P_i correspond au fonctionnement d'une partie de l'équipement pour lequel a été défini le processus P. Cette partie de l'équipement ne possède aucun organe en commun avec le reste du système (cf. point i) ; il peut être intéressant, notamment du point de vue maintenance, de construire un bloc de commande autonome pour chacun de ces sous-systèmes.

I.25 - Généralisation

Soit un ensemble $G = \{P_1, P_2, \ldots, P_N\}$ de processus $\forall\, i \in [1,n]$ posons :

$$P_i = \{VE_i, VS_i, RE_i, M_i, F_i, R_i, DE_i, T_i\}.$$

Définissons le *processus général* P(G) *associé à* G comme un 9 uple de la forme

$$P(G) = \{VE, \Gamma, G, MG, FG, RG, DG, TG, \psi\}$$

où

- VE est l'ensemble des variables de commande des éléments de G tel que

$\forall\, e_i \in VE, \exists\, P_j \in G$, e_i variable de commande de P_j

$\forall\, k \in [1,n]$, $k \neq j$

i) $e_i \notin VS_k$

ii) $e_i \notin f[T(VER_k) \cup T(C(RE_k))]$ où $f\,[T(VER_k) \cup T(C(RE_k))]$ est l'ensemble des fonctions booléennes correspondant à l'ensemble des tests sur les variables d'entrée rationnelles et les contenus de registres. Ainsi e_i ne peut être de la forme (M=W)

iii) $(e_i \notin VER_k) \cup (e_i$ variable de commande de $P_k)$.

- Γ est l'ensemble des *variables d'interconnexion* tel que

i) $\Gamma \subset \bigcup_{i=1}^{N} (VE_i \cup VS_i \cup C(RE_i))$

ii) $\forall\, e_i \in \Gamma, \exists\, P_j \in G$ tel que

e_i booléenne ==> e_i variable de commande de P_j

e_i rationnelle => il existe un test sur la valeur de e_i qui correspond à m_0 ordre de départ de P_j

iii) $\Gamma \cap VE = \emptyset$

- MG, ensemble des ordres, est une partie finie de CG × ℕ, où :

  . CG est l'ensemble des commandes générales c'est-à-dire que
  $CG = CG_0 \cup TG$

  . CG_0 est l'ensemble des commandes de processus tel que :
  en notant P_j la commande d'exécution de $P_j \in G$
  et $\bar{P}_j$ la commande d'arrêt en cours d'exécution de $P_j \in G$;
  nous écrirons alors $\bar{G} = \{\bar{P}_1, \bar{P}_2, \ldots, \bar{P}_N\}$
  et $CG_0 = G \cup \bar{G}$

  . TG est l'ensemble des tests à effectuer sur les valeurs des éléments de VE et de Γ tel que

  $$TG = TG(VE) \cup TG(\Gamma)$$

- FG est l'ensemble des fonctions booléennes associées aux éléments de MG

- RG est l'ensemble des enchaînements d'éléments de MG tel que

  $$RG \subset MG \times FG \times MG$$

- DG est l'ensemble des ordres de départ, constitué par des tests répétitifs sur la valeur de variables de commande d'éléments de G

- TG est l'ensemble des ordres terminaux, constitué par un ordre "fg" de non opération générale indiquent la fin d'exécution de tous les processus en cours

- ψ est l'ensemble des connexions tel que $\psi \subset V \times G \times G$
  avec $V = \bigcup_{i=1}^{N} (VE_i \cup VS_i \cup C(RE_i))$
  et $\forall \mu \in \psi : \mu = (v_i, P_j, P_k)$
  v_i répondant aux propriétés suivantes :

  i) $v_i \notin VE \cup \Gamma$

  ii) $\exists\, !\, P_j \in G,\ v_i \in VS_j \cup C(RE_i)$

  iii) $\exists\, P_k \in G,\ v_i \in VE_k$

Remarque 1 : Si pour $m_i = (c_h, n_h)$ avec $c_h \in G$ et $m_j = (c_k, n_k)$ avec $c_k \in TG$. On a $(m_i, f_i, m_j) \in RG$ alors

soit $c_k \in TG(VE)$ ce qui signifie que m_i indique une exécution complète de processus

soit $c_k \in TG(\Gamma)$ ce qui signifie que m_i indique seulement le lancement de l'exécution d'un processus.

Dans le même ordre d'idée $m_i = (c_k, n_k)$ avec $c_k \in \bar{G}$ indique un arrêt en cours d'exécution d'un processus.

Remarque 2 : Il est inutile de définir un ordre général de départ ; les ordres de départ relatif à chaque élément de G suffisent.

Les conditions initiales sont évidentes : au départ tous les éléments de G sont au repos.

Nous dirons qu'un processus général P(G) est cyclique si $\forall P_i \in G$, P_i est un processus cyclique.

I.3 - PROPRIETES

L'ensemble des propriétés définies ci-après devra être vérifié pour chaque description ; ces propriétés pourront être contrôlées de diverses façons, notamment à l'aide des graphes d'état (cf. chapitre II) et des règles de logique formelle (cf. chapitre V). Enfin, au chapitre IX, sera exposée une méthode de contrôle particulière aux processus asynchrones.

Nous distinguerons trois types de propriétés :

- des propriétés sémantiques relatives au fonctionnement du processus étudié
- des propriétés relatives au déroulement séquentiel
- des propriétés vérifiées par des déroulements simultanés.

Pour ce qui est des propriétés sémantiques :

Nous dirons que *deux commandes sont incompatibles* si elles ne peuvent être exécutées simultanément.

Nous dirons que *deux variables d'entrée booléennes sont incompatibles* si elles ne peuvent prendre simultanément la valeur 1. Ainsi, $e_1, e_2, \ldots, e_k$ constituent un ensemble de variables d'entrée incompatibles si

$$\nexists\ i,j \in [1,k] \ : e_i, e_j \in VEB \text{ tels que } e_i = e_j = 1 \text{ alors } i=j.$$

Nous dirons qu'une commande c_i modifie la valeur d'une variable d'entrée booléenne e_j si

- l'exécution de c_i entraîne le passage de e_j de 1 à 0
- l'exécution de c_i est susceptible de faire passer c_j de 0 à 1 au bout d'un temps τ variable.

c_j est dite alors *commande modifiante*.

Généralement une commande modifiante est un élément de C_0' ; à chaque élément de C_0' est associée une variable de sortie booléenne. Nous dirons qu'une variable de sortie est modifiante si elle est associée à une commande modifiante.

Pour une variable d'entrée booléenne $e_j \in VEB$, il peut exister plusieurs commandes modifiantes et donc plusieurs variables de sortie modifiantes.

Nous noterons $A(e_j)$ l'ensemble des commandes modifiant la valeur de e_j c'est-à-dire

$$A(e_j) = \{c_j \in C_0 \mid c_j \text{ commande modifiant la valeur de } e_j\}.$$

Ces définitions relatives à la sémantique du processus étudié restent valables pour les processus généraux. De plus, elles permettent de définir d'autres *propriétés liées au déroulement séquentiel* du fonctionnement d'un processus et, notamment, de déterminer la valeur des éléments de VEB à un instant quelconque du déroulement.

En effet, pour ce faire, trois types de données sont nécessaires

1) les tests sur la valeur de ces éléments
2) les ensembles de variables d'entrée incompatibles
3) les commandes modifiant la valeur de ces éléments.

Ainsi, par exemple, si $m_i = (c_j, n_i)$, $e_j \in T(VEB)$

si $(m_i, N, m_i) \in R$ et $(m_i, P, m_h) \in R$

et si e_j possède des commandes modifiantes c'est-à-dire que la variable testée par c_j n'est pas impulsionnelle alors, lorsque m_h sera exécutable, la valeur de la variable testée par c_j sera connue.

Nous verrons l'utilisation de cette propriété au chapitre V dans les possibilités de contrôle d'une description.

Nous pouvons en outre remarquer que :

i) Un ordre ne peut attribuer à une variable de sortie ou à un contenu de registre une valeur qu'il possède déjà.

ii) Entre deux tests sur la valeur d'une variable d'entrée booléenne modifiable e_b : e_b c VEB et $A(e_b) \neq \emptyset$, ou sur la valeur du contenu d'un registre il doit exister au moins une séquence contenant un ordre de modification de la valeur de l'élément testé.

iii) Dans une séquence linéaire $\sigma_i = \{m_{i_1} \ldots m_{i_{n_i}}\}$ contenant deux ordres m_{i_j} et m_{i_k} qui correspondent tous deux à une modification du contenu d'un registre $r_\ell \in RE$, il doit exister entre m_{i_j} et m_{i_k} au moins un ordre m_{i_h} correspondant soit à un test sur le contenu de r_ℓ, soit à un transfert du contenu de r_ℓ.

Si ce n'était pas le cas, l'exécution de m_{i_j} serait sans objet. Si la séquence σ_i n'est pas linéaire, cela implique qu'il existe entre m_{i_j} et m_{i_k} au moins une séquence contenant un ordre m_{i_k} ayant les mêmes propriétés que ci-dessus.

Il reste à définir *les conditions* dans lesquelles peuvent être exécutés *des déroulements séquentiels simultanés*.

Soit donc un processus P quelconque tel que $G(P) = \{X, \Gamma\}$. Pour deux suites quelconques $\alpha, \beta \in \Pi F^+$ et telles que

$$\alpha = k_1 . k_2 \ldots\ldots k_p$$

$$\beta = k'_1 . k'_2 \ldots\ldots k'_q \quad \text{avec } k_i, k'_i \in \Pi F \text{ et } p \geq q.$$

Nous dirons que β *est incluse dans* α *que nous noterons*

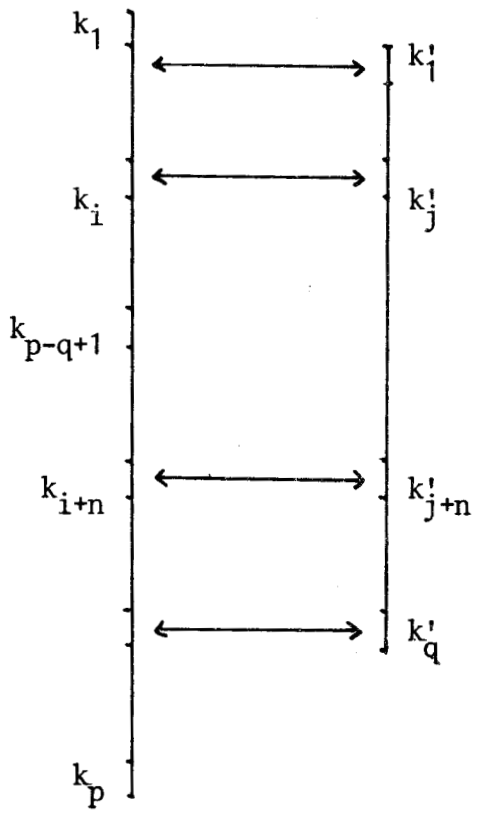
$$\beta \tilde{\subseteq} \alpha$$

si $\exists\ i \in [1,p-q+1]$, $\exists\ j \in [1,q]$ *tels que*

$\forall\ n \in [0,q-j]$, $\exists\ k''_n \in \Pi F$ *avec*

$$k_{i+n} = k'_{j+1} \times k''_n \ .$$

En figurant une suite par une ligne nous pouvons illustrer cette définition comme suit :



On vérifie aisément que la relation d'inclusion entre deux suites est une relation d'ordre ; c'est une relation d'ordre partielle car tous les éléments de ΠF^+ ne sont pas liés deux à deux par cette relation.

A P(M) est associé un ensemble de produits de fonctions booléennes ΠF ; *pour toute suite* $\alpha \in \Pi F^*$ *nous définirons un ensemble* $\bar{\alpha}$ *de suites tel que*

si nous posons $F_0 = \{f_j \in F \mid f_j$ *en correspondance biunivoque avec* $m_j = (c_h, n_j)$, $c_h \in C_0\}$ *et* $\Pi F_0 = \{\Pi f_j \mid f_j \in F_0\}$

alors

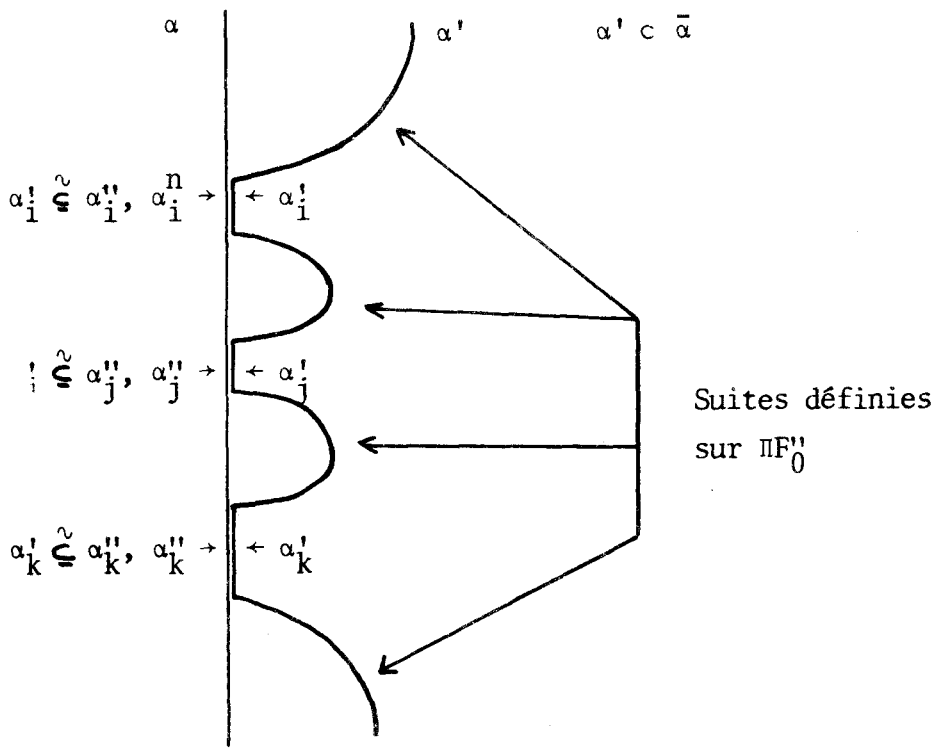
$$\bar{\alpha} = \{\mu \in \Pi F^* \mid \exists\, \mu'_1, \mu'_2, \ldots, \mu'_{n+1} \in \Pi F_0^* \quad \exists\, \alpha''_1, \alpha'_1, \alpha_1, \ldots, \alpha''_n, \alpha'_n, \alpha_n, \alpha_{n+1} \in \Pi F^*$$

tels que $\mu = \mu'_1\, \alpha'_1 \ldots \mu'_n\, \alpha'_n\, \mu_{n+1}$

$\alpha = \alpha_1\, \alpha''_1 \ldots \alpha_n\, \alpha''_n\, \alpha_{n+1}$

et $\alpha'_i \tilde{\subseteq} \alpha''_i \quad \forall\, i \in [1,n]\}$.

$\bar{\alpha}$ représente *l'ensemble des suites compatibles* avec α c'est-à-dire que si $\alpha' \in \bar{\alpha}$ le processus pourra reconnaître simultanément les commandes correspondant à α et α'. En figurant une suite par une ligne, il est possible d'illustrer la relation entre α et α' de la façon suivante :



Nous pouvons alors définir la propriété de compatibilité comme suit

$\forall \alpha' \in \bar{\alpha}$, $\forall k' \in \Pi F$ *tel que* $k' = k'_1 \times f_1 \times k'_2 \times f_2 \times \ldots \times f_r \times k'_r$ *où* $f_1, f_2, \ldots, f_r$ *sont en relation biunivoque avec* $m_1, m_2, \ldots, m_r$ *et* $\forall j \in [1, r]$: $m_j = (c_k, m_j)$, $c_k \in TB$.

Si $\alpha' = \alpha'_1.k'.\alpha'_2$

alors $\exists k \in \Pi F$, $k'' \in \Pi F$ *tels que* $\alpha = \alpha_1.k.\alpha_2$ *et* $k = k' \times k''$.

La notion d'ensemble compatible de suite ainsi que la propriété de compatibilité qui en découle seront utilisées pour définir et construire le graphe d'état (cf. chapitre II).

Par ailleurs des déroulement séquentiels simultanés ne peuvent effectivement se concevoir que si un certain nombre de propriétés découlant des propriétés sémantiques du processus étudié est vérifié.

Avant de définir ces propriétés il importe de préciser le sens des termes qui seront employés :

- Nous appellerons *séquences simultanées* des séquences susceptibles de se dérouler simultanément et telles qu'il n'existe pas d'enchaînement entre les ordres appartenant à ces deux séquences. Afin de faciliter la définition des propriétés étudiées ci-après, nous nous restreindrons au cas de deux séquences simultanées, les propriétés énoncées restent valables dans le cas de n séquences simultanées (n>2).

- Nous dirons que m_i et m_j appartenant à M sont liés par une relation de précédence si $m_j \in D^+(m_i)$. Dans ces conditions, si deux éléments m_i et m_j de M apparaissent respectivement dans deux séquences simultanées s_h et s_k, alors

i) m_i *et* m_j *ne peuvent correspondre à deux commandes incompatibles*. En effet, si ce n'était le cas, l'absence de relation de précédence entre s_h et s_k permettrait l'exécution simultanée de m_i et m_j c'est-à-dire un fonctionnement aberrant du processus. Il faut noter cependant que le problème est légèrement différent suivant que l'on considère un processus synchrone ou asynchrone (cf. chapitre II).

ii) m_i *et* m_j *ne peuvent correspondre à une même commande*, c'est-à-dire qu'il est impossible d'avoir $m_i = (c_\ell, n_i)$ et $m_j = (c_\ell, n_j)$, $c_\ell \in C$. En effet, cela supposerait que c_ℓ puisse être commandé de deux façons différentes et simultanées. Il y a néanmoins deux possibilités à envisager :

. $c_\ell \in C_0$, c_ℓ est une commande de variable de sortie ou une opération interne, le processus est donc mal conçu

. $c_\ell \in TB$, m_i et m_j peuvent alors être confondus. s_h et s_k sont liés par une relation de précédence au niveau du test c_ℓ qui leur est commun.

iii) *Si* $m_i = (c_\ell, n_i)$ *avec* $c_\ell \in C'_0$ *et*
$m_j = (c_r, n_j)$ *avec* $c_r \in TB$

c_r *définissant un test sur une variable d'entrée booléenne* e_b *dont la valeur est modifiée par l'exécution* c_ℓ, il y a deux possibilités :

. $\exists\, m_f$ dans s_k, $m_f = (c_s, n_f)$, c_s modifiant la valeur de e_b ; c_ℓ et c_s appartenant à $A(e_b)$, cela implique "physiquement" que ces ordres sont relatifs à un même organe commandé, c_ℓ et c_s constituant alors vraisemblablement un ensemble incompatible ; il y a donc eu omission d'une relation de précédence entre s_j et s_k pour interdire l'exécution simultanée de m_i et m_f.

. $\nexists\, m_f$ dans s_k (m_f gardant la même définition que ci-dessus) : le changement de valeur de e_b est donc causé par l'exécution de c_h. En effet, si ce n'était pas le cas, cela supposerait qu'il existe un ordre $m_g = (c_r, n_g)$ dont l'exécution a précédé l'exécution des séquences s_k et s_h et tel que $c_r \in A(e_b)$; cependant comme $c_\ell \in A(e_b)$, il en résulte qu'entre l'exécution de m_g et celle de m_i doit exister un ordre annulant l'effet de m_g ; donc m_g ne peut être la cause du changement de valeur de e_b.

iv) *Si* m_i *et* m_j *correspondent tous deux au changement d'une valeur dans un registre unique*, il doit exister une relation de précédence entre m_i et m_j, car il est inconcevable de charger simultanément deux valeurs dans un même registre.

v) *Si* m_i *correspond à un changement de valeur dans un registre et* m_j *au transfert du contenu de ce registre dans un autre registre*, il doit exister une relation de précédence entre m_i et m_j car il est inconcevable de charger et de lire simultanément un même registre.

vi) *Si* m_i *correspond à un chargement de valeur dans un registre et* m_j *à un test sur le contenu de ce registre,* il y a deux éventualités à envisager :

. $\exists\, m_f \in s_k$, m_f correspondant à un chargement de valeur dans le registre considéré : on est alors ramené au point iv, m_i,m_j et m_f étant liés deux par deux par des enchaînements.

. $\nexists\, m_f \in s_k$, m_f défini comme ci-dessus, alors m_i et m_j sont liés par des enchaînements.

vii) *Si* m_i *correspond à un transfert du contenu d'un registre et* m_j *à un test sur le contenu du même registre,* m_i et m_j doivent être liés par des enchaînements.

Une dernière propriété évidente n'a pas été mentionnée : les enchaînements doivent constituer un ensemble cohérent c'est-à-dire que le graphe correspondant au processus doit être un graphe connexe.

DIFFERENTS TYPES DE PROCESSUS

2.0 - INTRODUCTION

Dans le présent chapitre, nous allons rechercher les possibilités d'exploitation des informations fournies par une description de processus conforme aux définitions et propriétés développées au chapitre I.

Nous envisagerons successivement :

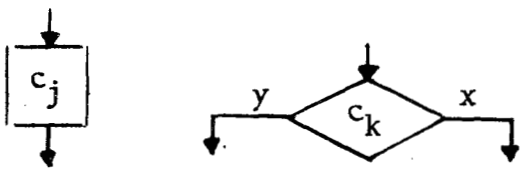
- la construction et la réduction des équations booléennes des systèmes asynchrones
- la définition de la structure du bloc de commande des systèmes synchrones.

Nous définirons ensuite les systèmes programmés.

Pour cela, nous utiliserons les notions suivantes :

- Etant donné un processus P, nous appellerons *organigramme de commande* le graphe qui décrit les enchaînements de commandes et donc le fonctionnement du processus. Il est facile de reconstituer l'ensemble M à partir de cet organigramme en indiçant les diverses occurences des commandes qui y figurent.

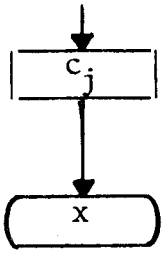
Pour la représentation de cet organigramme, nous utiliserons les conventions suivantes : pour $c_j \in C_0$ et $c_k \in TB$ avec $x,y \in [P,N]$ nous représenterons



Nous appellerons *organigramme de processus* le graphe décrivant en outre les variations de valeur d'éléments de VEB, ces variations étant provoquées par l'exécution de

commandes modifiantes et n'ayant d'autre rôle que de repositionner correctement les variables d'entrée booléennes concernées pour une séquence ultérieure.

Pour la représentation de cet organigramme, nous utiliserons la convention suivante : si $c_j \in C_0$, $x \in VEB \cup \overline{VEB}$, l'exécution de c_j faisant passer x de 0 à 1,nous représenterons



- D'autre part, soit G[X,Γ] le graphe correspondant à P (cf. chapitre I), nous appellerons *matrice de précédence* B de P, la matrice booléenne carrée de dimension n×n avec n = card [X] telle que :

$$B = \begin{bmatrix} b_n \ldots b_{1_j} \ldots b_{1_n} \\ b_{i_1} \ldots b_{i_j} \\ b_{n_1} \ldots\ldots\ldots b_{n_n} \end{bmatrix}$$

avec $b_{i_j} = 1$ si et seulement si $[x_i \rightarrow x_j] \in \Gamma$

$b_{i_j} = 0$ dans les autres cas.

Une telle matrice n'est significative que pour des processus P tels que

$$\forall\, m_i \in M,\ m_i \in (c_j, n_j) \text{ avec } c_j \in C_0.$$

Lorsqu'un processus comporte des tests, la matrice de précédence doit être modifiée.

En effet, dans ce cas, si $x_i \in X$ correspond à un test, tout arc γ d'origine x_i est accompagné de l'indication N ou P (cf. chapitre I).

Plus généralement, nous définirons donc la matrice de précédence B d'un processus P quelconque auquel est associé le graphe G = (X,Γ), comme une matrice

carrée de dimension W = card (X), avec $\forall x_i, x_j \in X$

b_{ij} = 1 si et seulement si $(x_i,x_j) \in \Gamma$

b_{ij} = P - - - - (x_i,x_j), $P \in \Gamma$

b_{ij} = N - - - - (x_i,x_j), $N \in \Gamma$

b_{ij} = 0 dans tous les autres cas.

Dans ce qui suit, nous considérerons toujours qu'un processus est représenté par un organigramme de commande ou de processus à partir duquel ont été successivement construits l'ensemble des ordres M et la matrice de précédence B correspondante.

2.1 - PROCESSUS ASYNCHRONES

Nous appellerons *processus asynchrone* un processus tel que l'exécution d'un ordre est immédiatement consécutive à la fin d'exécution de la totalité de ses prédécesseurs directs et ce pour un ensemble de conditions de fonctionnement donné ; le moment d'exécution d'un ordre n'est donc pas fixé dans le temps.

Comme les temps d'exécution sont variables, cette définition implique que le processus peut évoluer différemment dans le temps suivant que l'exécution de tel ou tel ordre dure plus longtemps que celle de tel ou tel autre. Il peut même apparaître au cours du déroulement, notamment durant les états transitoires des exécutions simultanées de commandes incompatibles. Il est donc nécessaire de pouvoir définir, à partir des enchaînements décrivant le déroulement du processus, l'ensemble des différentes évolutions possibles de ce processus.

C'est ce que nous allons voir ci-dessous en introduisant la notion de graphe d'état d'un processus.

Nous dirons qu'un élément Z de P(M) et une suite α de ΠF^* sont *associés* si $Z = \Delta(\alpha,DE)$, ce que nous noterons $Z(\alpha)$.

En outre, nous dirons que $Z(\alpha)$ est un *ensemble compatible* si aucune raison d'ordre sémantique ni aucune relation de précédence n'interdit l'exécution

simultanée de tous les éléments de $Z(\alpha)$. Ainsi DE ensemble de départ et c_d condition de départ sont tels que $DE(c_d=1)$ constitue un ensemble compatible, en effet si $c_d=1$, $\forall\, m_i \in DE$, m_i est exécutable.

Plus précisément, nous définirons *un ensemble compatible* $Z(\alpha)$ comme suit

$$Z \in P(M),\ \exists\ \alpha \in \Pi F^*$$

i) $\forall\, m_i \in Z(\alpha),\ \exists\ \alpha_i \in \bar{\alpha}$ tel que $m_i \in \Delta(\alpha_i, DE)$

ii) $\forall\, m_i \in Z(\alpha),\ m_i \notin D^+[Z(\alpha)]$

iii) $\forall\, m_i, m_j \in Z(\alpha)$ tels que $m_i = (c_k, n_i)$ et $m_j = (c_h, n_j)$

c_k et c_h ne sont pas incompatibles

iv) $\forall\, m_i, m_j \in Z(\alpha)$ tels que $\alpha_i, \alpha_j \in \bar{\alpha}$ vérifient

$m_i \in \Delta(\alpha_i, DE)$ et $m_j \in \Delta(\alpha_j, DE)$

alors $\forall\, m_k = (c_g, n_k)$ avec $c_g \in TB$, m_k ayant f_{ℓ_1} et f_{ℓ_2} comme éléments associés dans F

si $\exists\ k_i' \in \Pi F,\ \exists\ \mu_i, \upsilon_i \in \Pi F^*$ tels que

$$k_i = f_{\ell_1} \times k_i'$$

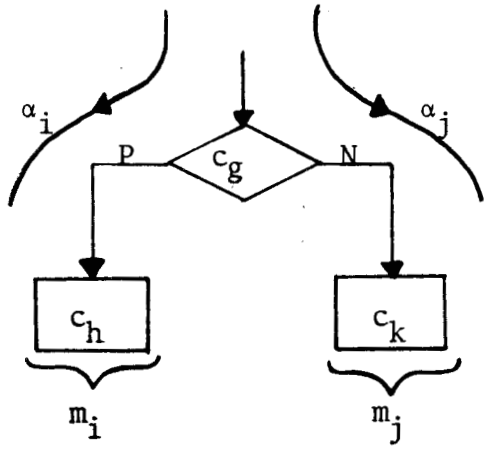
$$\alpha_i = \mu_i . k_i . \upsilon_i$$

alors $\not\exists\ k_j' \in \Pi F,\ \exists\ \mu_j, \upsilon_j \in \Pi F^*$ tels que

$$k_j = f_{\ell_2} \times k_j'$$

$$\alpha_j = \mu_j . k_j . \upsilon_j$$

Nous pouvons illustrer graphiquement le contraire de cette propriété comme suit



Dans ce cas, m_i et m_j ne peuvent pas appartenir à un ensemble compatible.

Remarque : Nous avons considéré un processus ne présentant pas de rebouclage ; dans le cas contraire la définition d'un ensemble compatible est plus lourde.

Nous appellerons *graphe d'état d'un processus asynchrone* P un graphe orienté noté GE = {E,Θ} où

- chaque sommet, élément de E, correspond à un ensemble compatible exécutable

- $\Theta \in E^2$, ensemble des arcs orientés est tel que

- à chaque élément $\eta \in \Theta$ tel que $\eta = \{e_i, e_j\}$ est associé un élément du produit cartésien $P(M) \times \Pi F$ noté (e_h, k) tel que $e_h \subseteq e_i$ et que e_i étant exécutable, e_h exécuté et k vrai alors e_j est exécutable.

Pour construire le graphe d'état, nous aurons besoin du théorème suivant :

Théorème : Pour un processus P tel que G(P) ne contienne pas de circuit, si $Z(\alpha)$ est un ensemble compatible exécutable et $Z_1(\alpha)$ un sous-ensemble de $Z(\alpha)$, alors en notant :

$$Z_2(\alpha) = Z(\alpha) \setminus Z_1(\alpha),$$

le nouvel ensemble compatible exécutable après exécution de $Z_1(\alpha)$ sous la condition $k_s \in \Pi F(Z_1(\alpha))$ sera défini par

$$\boxed{Z(\alpha.k_s) = Z_2(\alpha) \cup (\Delta(k_s, Z_1(\alpha)) \setminus (D^+(Z_2(\alpha)) \cup D^+(\Delta(k_s, Z_1(\alpha)))))}$$

Démonstration : Nous devons vérifier pour $Z(\alpha.k_s)$ les quatre propriétés des ensembles compatibles.

Nous noterons :

$$Z_3(\alpha) = \Delta(k_s, Z, (\alpha)) \setminus (D^+(Z_2(\alpha)) \cup D^+(\Delta(k_s, Z_1(\alpha))))$$

$Z_3(\alpha) \neq \Delta(k_s, Z_1(\alpha))$ car

i) si $\exists\, m_i \in \Delta(k_s, Z_1(\alpha))$ et $m_i \in D^+(Z_2(\alpha))$, il n'est pas possible de construire un ensemble compatible contenant à la fois m_i et $Z_2(\alpha)$.

ii) si $\exists\, m_i \in \Delta(k_s, Z_1(\alpha))$ et $m_i \in D^+(\Delta(k_s, Z_1(\alpha)))$, $\Delta(k_s, Z_1(\alpha))$ ne constitue pas un ensemble compatible.

La composition de $Z_3(\alpha)$ étant ainsi justifiée, les propriétés I, III et IV sont vérifiées par construction.

Il reste à démontrer que $Z_2(\alpha) \cup Z_3(\alpha)$ vérifient la deuxième propriété des ensembles compatibles.

$\forall\, m_i \in Z_2(\alpha)$, $m_i \notin D^+(Z_2(\alpha))$ puisque $Z_2(\alpha)$ est un ensemble compatible.

$\forall\, m_i \in Z_2(\alpha)$, $m_i \notin D(Z_1(\alpha))$ car $Z_1(\alpha) \cup Z_2(\alpha) = Z(\alpha)$ et $Z(\alpha)$ est un ensemble compatible

d'où $m_i \notin D^+(Z_3(\alpha))$

$\forall\, m_j \in Z_3(\alpha)$, $m_i \notin D^+(Z_2(\alpha))$ par construction. Comme $Z_3(\alpha) \subset \Delta(k_s, Z_1(\alpha))$ si $m_j \in D^+(Z_3)$ cela entraînerait $m_j \in D^+(\Delta(k_s, Z_1(\alpha))$ or par construction $Z_3(\alpha) \cap D^+(\Delta(k_s, Z_1(\alpha)) = \emptyset$ d'où $m_3 \notin D^+(Z_3(\alpha))$.

En résumé :

$$\forall\, m_i \in Z_2(\alpha) : m_i \notin D^+(Z_2(\alpha)) \cup D^+(Z_3(\alpha))$$

$$\forall\, m_j \in Z_3(\alpha) : m_j \notin D^+(Z_2(\alpha)) \cup D^+(Z_3(\alpha))$$

Le théorème est démontré. Il permet de construire les enchaînements des ensembles compatibles d'un processus P, et fournit donc un algorithme de construction du graphe d'état associé à P.

Algorithme de construction du graphe d'état

Notons B la matrice de précédence associée au graphe de définition G d'un processus P.

B étant connue, l'algorithme consiste dans les étapes suivantes :

a) Prendre DE comme premier ensemble compatible, c'est-à-dire celui correspondant aux colonnes de B ne comportant que des zéros, stocker DE dans une pile S.

b) Pour chaque ensemble compatible $Z(\alpha)$ précédemment construit, déterminer par lecture de B l'ensemble $\Pi F(Z_1(\alpha))$, pour toute partie $Z_1(\alpha)$ de $Z(\alpha)$.

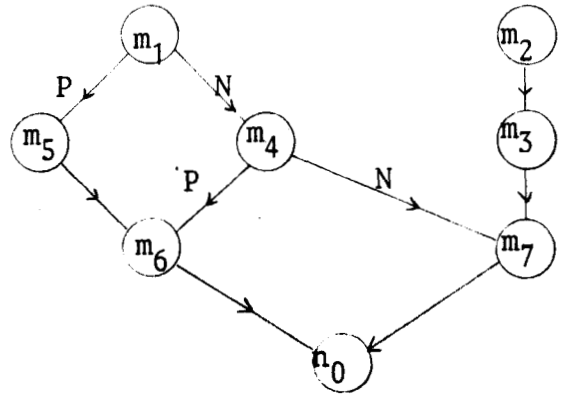
c) Pour chaque $Z_1(\alpha)$, pour chaque élément k de $\Pi F(Z_1(\alpha))$, appliquer le théorème précédent, ce qui détermine ainsi $Z(\alpha.k_s)$ à stocker dans la pile S.

d) Les trois ensembles $Z_1(\alpha)$, $Z(\alpha)$, $Z(\alpha.k_s)$ correspondent à un arc du graphe d'état du processus étudié, et aux deux sommets extrémités de cet arc.

e) Itérer b,c,d, tant que l'on obtient de nouveaux ensembles compatibles, c'est-à-dire des ensembles non encore stockés dans S.

Remarque : Chaque fois que l'algorithme engendre un nouvel ensemble compatible, il faut contrôler qu'il n'existe pas un ensemble de commandes incompatibles (cf. paragraphe I.3) qui corresponde à tout ou partie de l'ensemble compatible engendré par l'algorithme.

Exemple : Soit un processus P décrit par le graphe G ci-dessous où X ≡ M



Auquel correspond la matrice de précédence

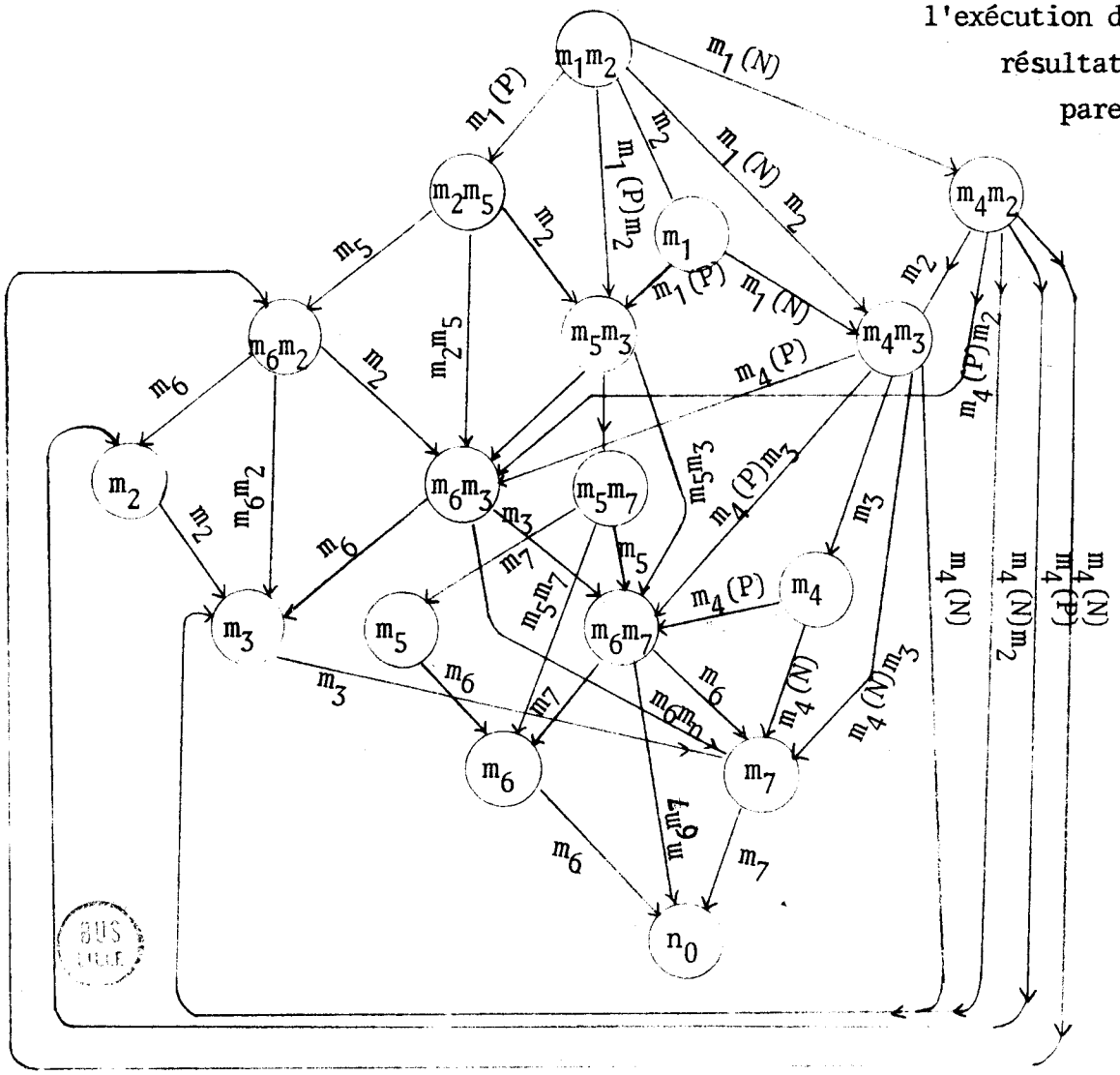
B ⟶

D = (m_1, m_2)

B	m_1	m_2	m_3	m_4	m_5	m_6	m_7	n_0
m_1				N	P			
m_2			1					
m_3							1	
m_4						P	N	
m_5						1		
m_6								1
m_7								1

Graphe d'état

Si $m_i = (c_h, n_h)$ avec $c_h \in T$
on notera $m_i(X)$ avec $X \in [N,P]$
l'exécution du test c_h avec
résultat du test
parenthésé



Propriété du graphe d'état :

L'algorithme de construction défini précédemment s'applique également au cas où il existe des boucles dans le déroulement du processus étudié.

Il existe une boucle dans un processus si $\exists\ m_i, m_j \in M$ tels que $(m_i, f_i, m_j) \in R$ et $m_i \in d(m_j)$.

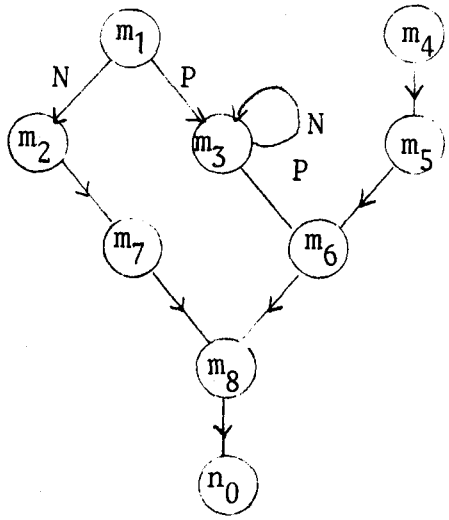
L'existence d'une boucle dans un processus P se traduit par l'existence d'un circuit dans le graphe $G(X,\Gamma)$ associé à P.

Au niveau du graphe d'état, si m_i est l'ordre sur lequel s'effectue le rebouclage, il y aura autant de circuits que d'ensembles compatibles contenant m_i. Il faut cependant remarquer qu'un cycle doit comprendre un test.

Il existe de nombreux algorithmes permettant de détecter les circuits existant dans un graphe G, nous ne développerons pas ces algorithmes et nous nous contenterons de donner un exemple :

Exemple

Soit $G(X,\Gamma)$ associé à un processus P avec $X \equiv M$

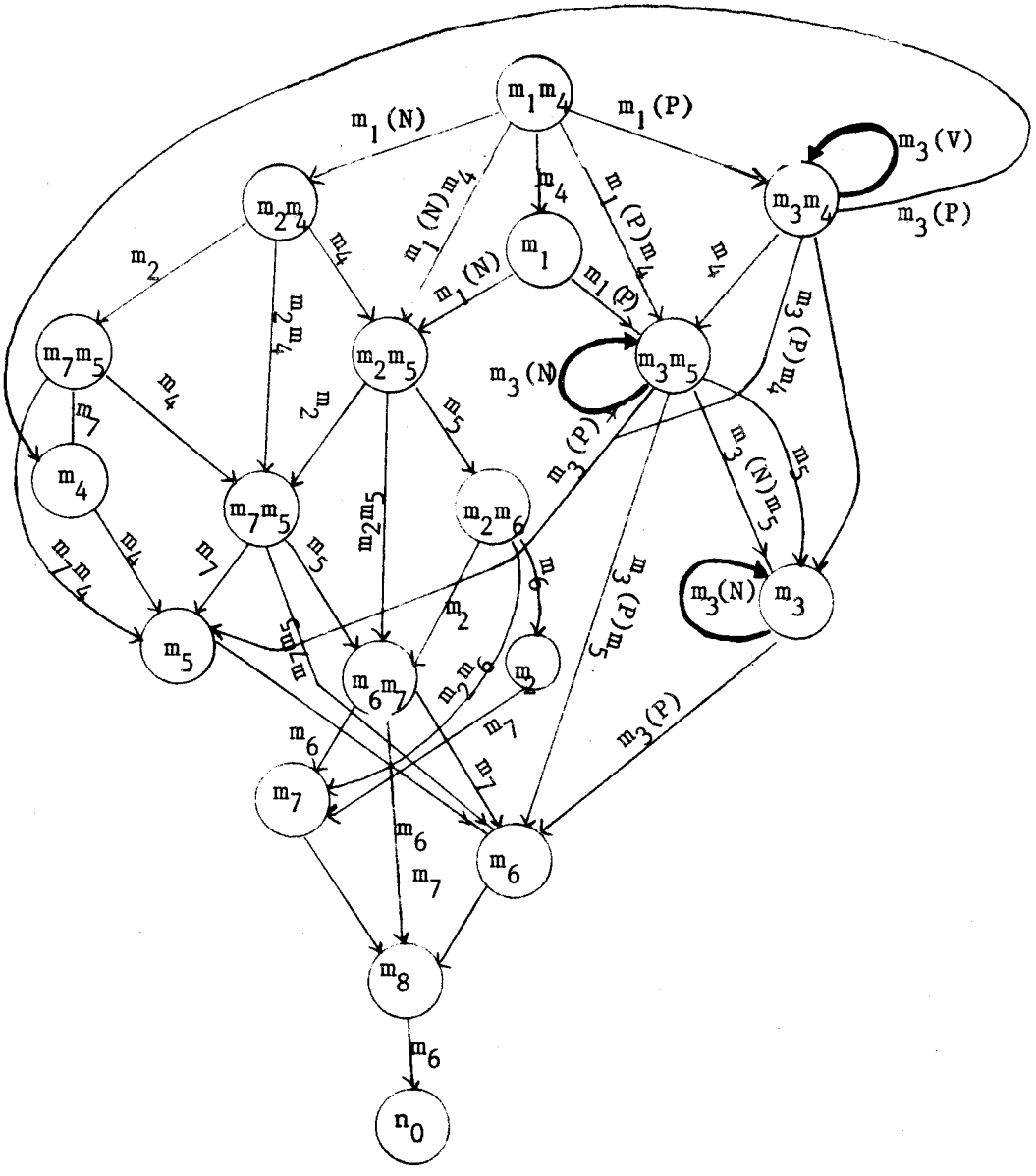


A G correspond la matrice de précédence B

B	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	n_0
m_1		N	P						
m_2							1		
m_3			N			P			
m_4					1				
m_5						1			
m_6								1	
m_7								1	
m_8									1

$D = \{m_1, m_4\}$

Graphe d'état

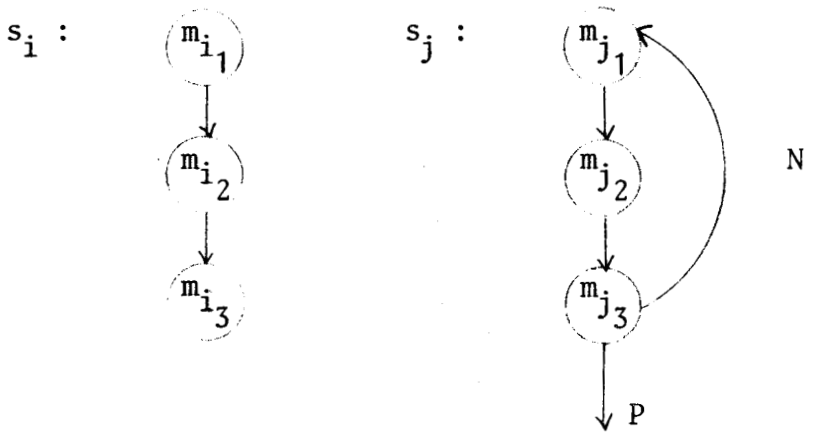


La boucle sur m_3 se retrouve ici sur les ensembles compatibles (m_3,m_4), (m_3,m_5) et (m_3).

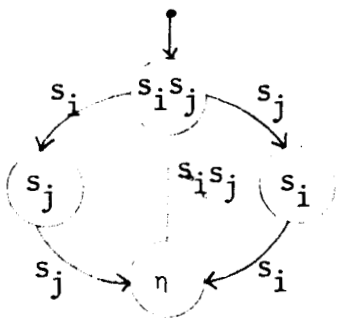
Le graphe d'état correspondant à un processus P *présente un nombre de sommets considérablement plus élevé que le graphe associé à* P *qui décrit les enchaînements.*

Cependant, *lorsque le processus* P *est une séquence linéaire, les graphes d'état et le graphe associé à* P *se confondent,* car les ensembles compatibles correspondent à des ordres simples. On voit donc l'intérêt que présente l'élimination des séquences linéaires dans la représentation graphique G d'un processus P. Le graphe d'état correspondant au graphe G' sans séquence linéaire sera beaucoup plus simple que celui correspondant au graphe G de départ. De plus, pour retrouver les ensembles compatibles réels, il suffira de décomposer les ensembles compatibles du graphe d'état correspondant à G'.

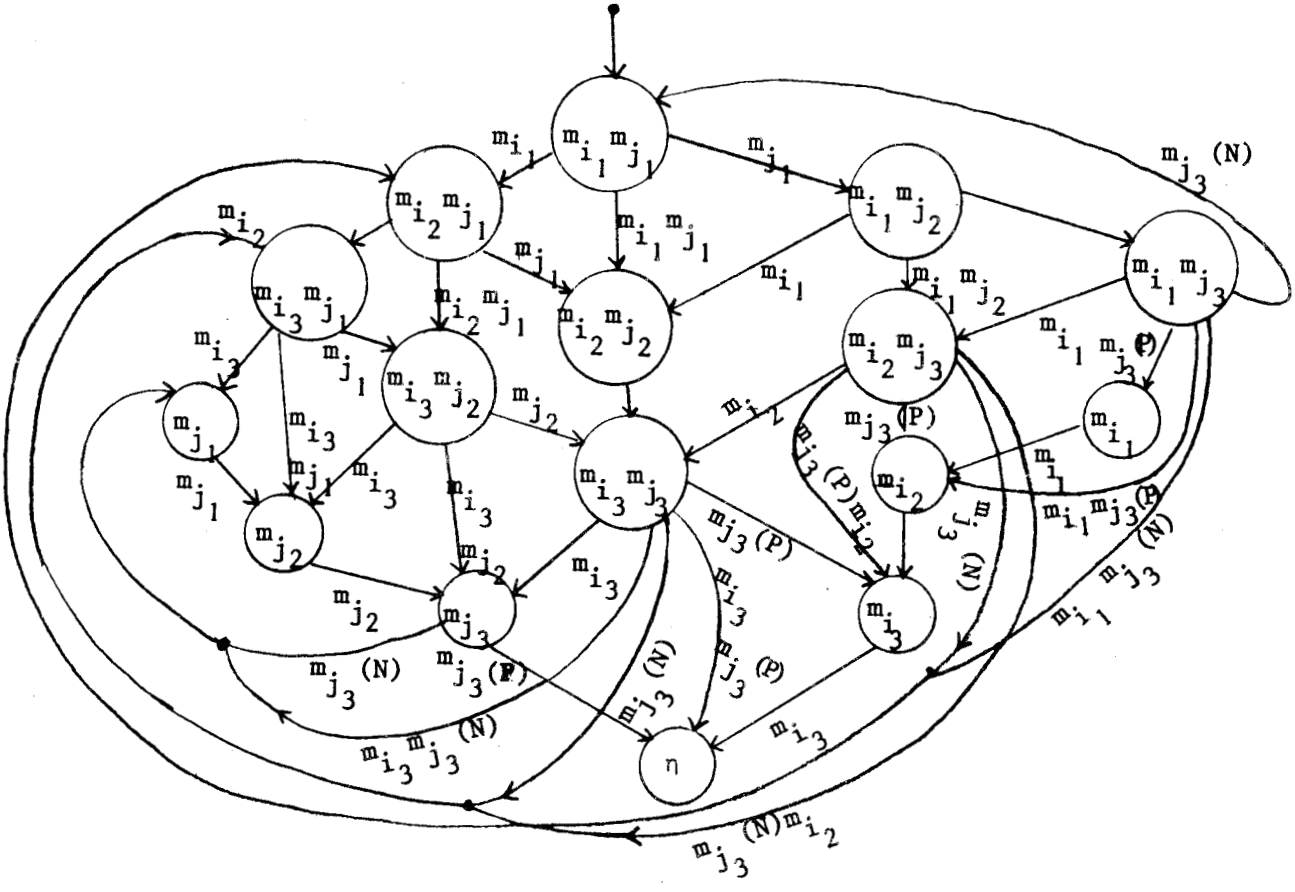
Exemple : Soient deux séquences s_i et s_j d'un processus P



Si dans le graphe d'état correspondant à G' il existe un sous-graphe du type ci-après :



Nous pouvons construire le sous-graphe correspondant dans le graphe d'état associé à G(P) de la façon suivante



Création de variables internes

A tout instant un processus asynchrone contrôlé par un bloc de commande est défini par la valeur de ses entrées, de ses sorties et de ses états internes. Supposons pour simplifier l'exposé du problème que le processus étudié ne dispose que de variables d'entrée et de sortie booléennes et soit dépourvu d'état interne.

A tout instant t, notons

- E le vecteur d'entrée c'est-à-dire l'ensemble des valeurs des variables d'entrée à l'instant t, (card (E) = card (VEB)),

- S0 le vecteur de sortie c'est-à-dire l'ensemble des valeurs des variables de sortie à l'instant t, (card (S0) = card (VS)). Les éléments de S0 dépendent de la succession de commandes appartenant à C'_0 exécutées depuis le début du déroulement du processus.

Considérons l'enchaînement suivant :

$$m_i, m_j \in M,\ m_i = (c_h, n_i),\ c_h \in TB,\ m_j = (c_k, n_j),\ c_k \in C'_0$$
$$\text{et} \qquad (m_i, P, m_j) \in R$$

c_h correspond à un test sur une variable d'entrée booléenne $e_b \in VEB$ donc, si $e_b=1$, m_j est immédiatement exécuté.

Soit t_1 l'instant où est exécuté m_i ; à cet instant un vecteur E_1 et un vecteur SO_1 définissent l'état du processus.

Soit t_2 l'instant où est connu le résultat du test sur m_i ; à cet instant E_1 devient E_2 ; si la commutation était instantanée, SO_1 deviendrait immédiatement SO_2, en fait le tableau ci-dessous indique le déroulement réel de l'enchaînement

t_1	t_2	t_3
VE_1	VE_2	VE_2
VS_1	VS_1	VS_2

S'il existe dans le déroulement du processus étudié une séquence d'ordres qui corresponde au tableau ci-dessous

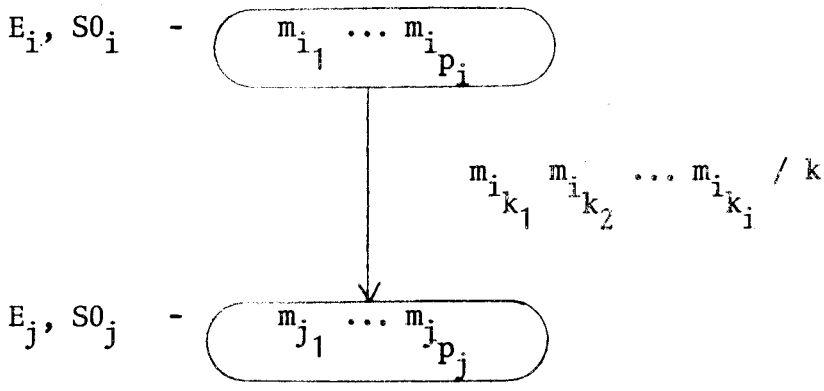
t_1	t_2	t_3	t_4	t_5	t_6	t_7
E_1	E_2	E_2	E_2	E_3	E_2	E_2
SO_1	SO_1	SO_2	SO_2	SO_1	SO_1	SO_4
m_i, P, m_j		m_ℓ, N, m_h		m_r, P, m_s		
	m_j, m_ℓ		m_h, m_r			

Rien ne différencie l'état du processus à l'instant t_2 de son état à l'instant t_6. Le système étant asynchrone, il y a ambiguïté de fonctionnement.

Il convient alors d'introduire un registre booléen supplémentaire dans le bloc de commande. Les deux contenus possibles de ce registre constitueront deux états internes du processus qui permettront de différencier les états du processus aux instants t_2 et t_6.

Le graphe d'état d'un processus permet de détecter de telles ambiguïtés de fonctionnement. En effet, connaissant l'organigramme de processus, il est possible de construire le graphe d'état et d'attribuer à chaque sommet de ce graphe un vecteur E_i et un vecteur $S0_i$ déterminant l'état du processus.

Ainsi pour le sous-graphe d'état ci-dessous



nous pouvons écrire que les vecteurs E_i, $S0_i$ déterminent l'état du processus au moment où l'ensemble compatible $(m_{i_1} \ldots m_{i_{p_i}})$ est exécutable et que les vecteurs E_j, $S0_j$ déterminent l'état du processus au moment où l'ensemble compatible $(m_{j_1} \ldots m_{j_{p_j}})$ est exécutable.

L'exécution de l'ensemble compatible $\{m_{i_{h_1}} m_{i_{h_2}} \ldots m_{i_{h_i}}\}$ a fait passer le processus de l'état caractérisé par E_i, $S0_i$ à l'état caractérisé par E_j, $S0_j$.

Les vecteurs E_j et $S0_j$ se déduisent respectivement des vecteurs E_i et $S0_i$ grâce à la connaissance de $m_{i_{k_1}} \ldots m_{i_{k_i}}$ et de k.

Si à deux sommets du graphe d'état, c'est-à-dire à deux ensembles compatibles exécutables différents correspondent un même état du processus, donc un même couple de vecteur E_i, $S0_i$, il y a ambiguïté dans le déroulement du processus.

Le graphe d'état d'un processus asynchrone permet d'autre part de détecter des erreurs de conception :

En effet, si un sommet du graphe d'état $\{m_{i_1} \ldots m_{i_{p_i}}\}$ correspond à un ensemble de commande $c_{i_1} \ldots c_{i_{p_i}}$ tel que tout ou partie de cet ensemble constitue un ensemble incompatible, il en résulte que le processus étudié ne peut fonctionner correctement.

Il est d'ailleurs possible de distinguer deux niveaux de gravité dans les erreurs de ce type.

i) Il existe un ou plusieurs chemins du graphe d'états passant par un sommet correspondant à un ensemble incompatible, auquel cas ce mauvais fonctionnement peut être supprimé par une modification relativement peu importante du processus.

ii) Tous les chemins issus du sommet correspondant à l'ensemble de départ DE aboutissent à un sommet correspondant à un ensemble incompatible, c'est alors l'ensemble du processus qui est à revoir.

Le graphe d'état d'un processus asynchrone correctement défini donc ne présentant ni ambiguïté de fonctionnement ni ensemble de commandes incompatibles dans le graphe d'état *permet de construire les équations booléennes* qui sont à la base de la réalisation du bloc de commande. Des programmes créant et réduisant les fonctions booléennes, à partir du graphe d'état existent déjà actuellement.

2.2 - PROCESSUS SYNCHRONES

Nous appellerons *processus synchrone* un processus dont le fonctionnement est cadencé par une base de temps c'est-à-dire que si nous considérons un enchaînement quelconque

$$(m_i, f_h, m_j) \in R \; ; \; m_i, \; m_j \in M, \; f_h \in F,$$

il existe une horloge H émettant un signal h suivant une période τ telle que si au temps t, h=1 entraîne l'exécution de m_i, au temps $t+\tau$, h=1 entraînera, sous la condition f_h, l'exécution de m_j.

Le temps d'exécution d'un ordre doit donc être inférieur à la durée du signal h.

Cette définition implique que le processus peut évoluer différemment suivant qu'à chaque top d'horloge s'exécute tout ou partie de l'ensemble des ordres successeurs immédiats des ordres exécutés au top précédent. Parmi toutes les possibilités envisageables, il peut même apparaître des ensembles d'ordres incompatibles, d'où la nécessité de pouvoir construire à partir des enchaînements décrivant le déroulement du processus l'ensemble des différentes évolutions possibles de ce processus.

La définition du graphe d'état d'un tel processus diffère quelque peu de celle introduite pour un processus asynchrone.

Nous appellerons *graphe d'état d'un processus synchrone*, un graphe orienté noté GE = {E,Θ} où

- chaque sommet, élément de E, correspond à un ensemble compatible exécutable

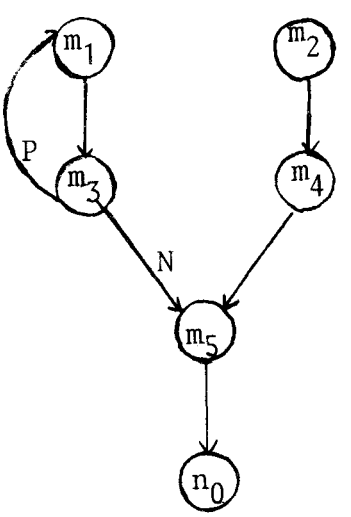
- $\Theta \subset E^2$ est l'ensemble des arcs

- à chaque élément $\eta \in \Theta$, $\eta = \{e_i, e_j\}$, est associé biunivoquement un élément du produit cartésien R(M) × ΠF du type $\{e_h, k\}$ tel que $e_h \subseteq e_i$ et que e_i étant exécutable au temps t, e_h exécuté, e_j est exécutable au temps $t+\tau$ pour k vrai.

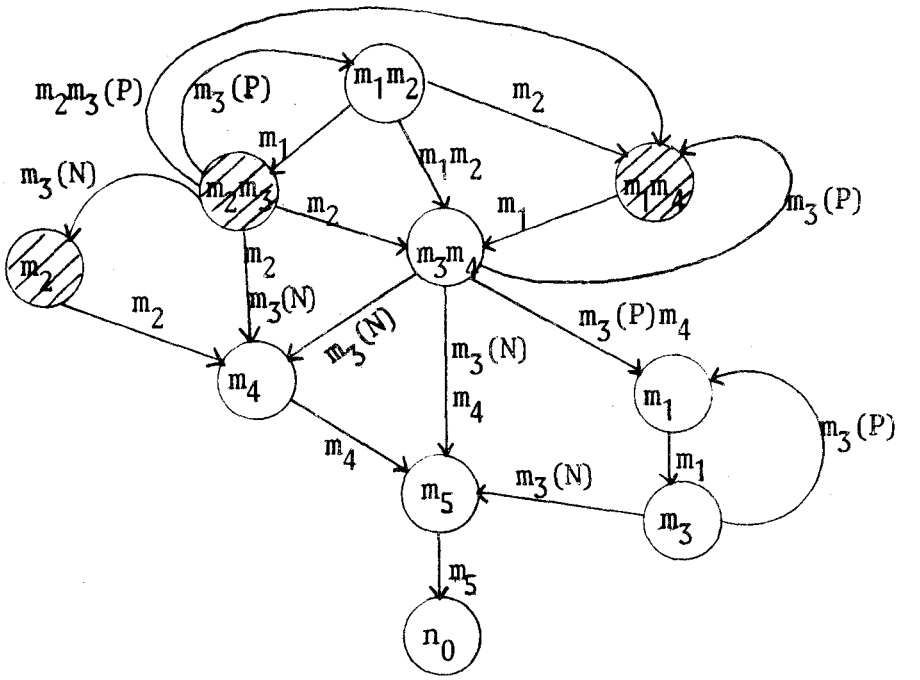
Construction du graphe d'état

La méthode de construction est proche de celle utilisée dans le cas des processus asynchrones. Cependant, dans le cas où il existe des boucles dans le déroulement du processus le problème se pose différemment.

Exemple



En appliquant brutalement l'algorithme de construction du graphe d'état relatif aux processus asynchrones on obtiendrait :



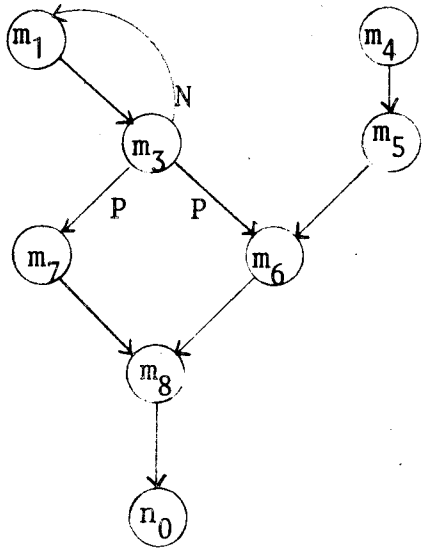
Or le graphe d'état d'un processus asynchrone donne toutes les possibilités envisageables d'exécution. Parmi ces possibilités nous pouvons éliminer d'emblée les cas du type : m_3 m_2 ou m_1 m_4 ou m_2 à cause de l'existence de la boucle. En effet, si on exécutait simultanément m_3 et m_2, au top d'horloge suivant on devrait exécuter, suivant le résultat de m_3, soit m_1m_4 soit m_4 alors que si on avait exécuté simultanément m_3 et m_4,, au top d'horloge suivant on exécutait m_1, m_2 ou m_5, c'est-à-dire que l'on gagnerait une période d'horloge sur la durée d'exécution du processus.

Nous pouvons donc éliminer du graphe d'état les sommets hachurés et les arcs correspondants car ils compliquent le graphe et définissent des possibilités d'exécution plus lentes donc inintéressantes. Il est évident que, de la même façon que dans les cas des processus asynchrones, il faut éliminer du graphe tous les chemins aboutissants ou issus d'un sommet correspondant à un ensemble de commandes incompatibles.

On pourrait proposer également, par une remarque analogue à celle faite sur les boucles de supprimer le sommet noté m_4, nous verrons plus loin pourquoi cette hypothèse de fonctionnement n'a pas été éliminée.

Propriété du graphe d'état :

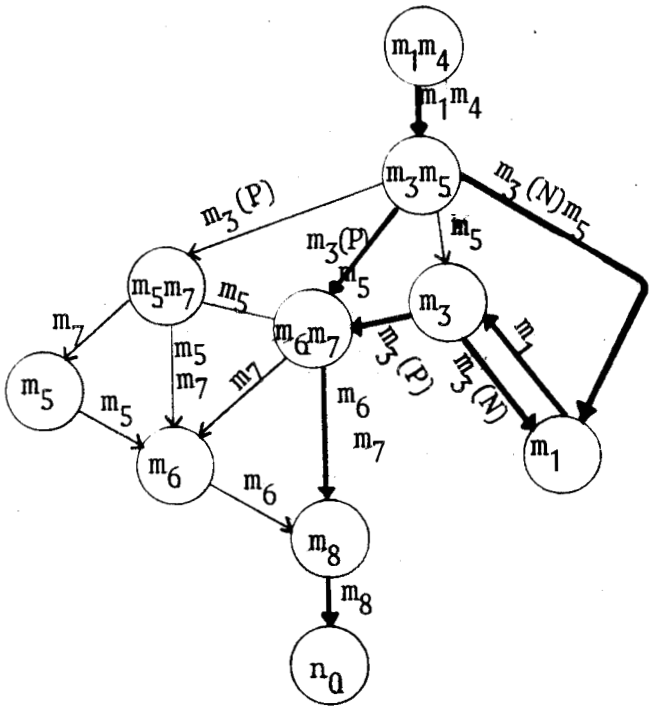
Soit le processus suivant défini par son graphe :



A ce prossus est associée la matrice de précédence suivante :

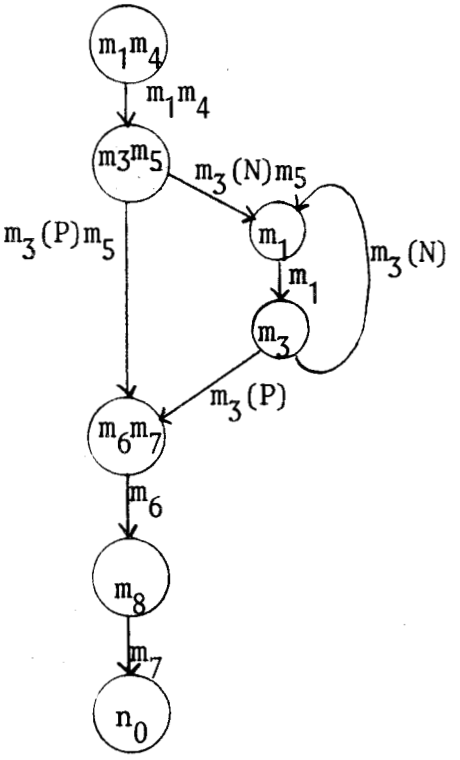
	m_1	m_3	m_4	m_5	m_6	m_7	m_8	n_0
m_1		1						
m_3	N				P	P		
m_4				1				
m_5					1			
m_6							1	
m_7							1	
m_8								1

Le graphe d'état correspondant est le suivant



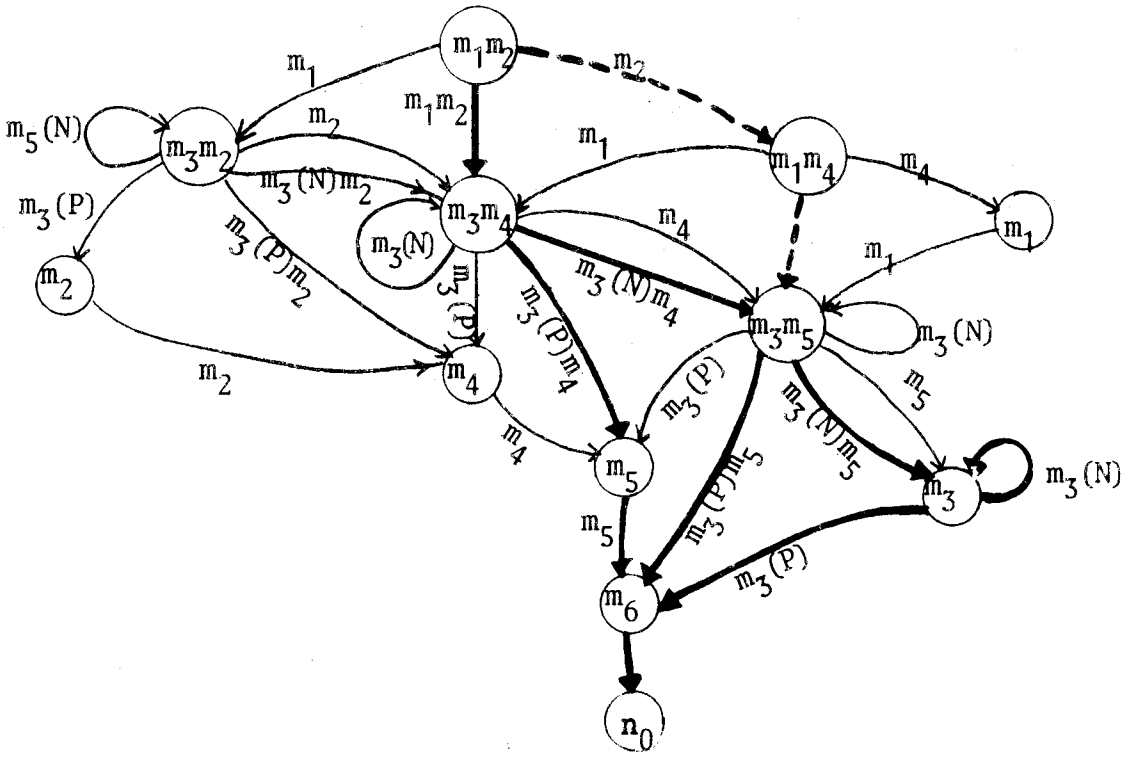
les simplifications relatives aux ensembles compatibles construits à partir des sommets de la boucle ont été réalisées

A chaque arc correspond un temps d'horloge τ. Pour obtenir un fonctionnement du processus le plus rapide possible il suffit de rechercher le sous-graphe minimal du graphe (ici en traits forts) qui permet l'exécution de tous les ordres

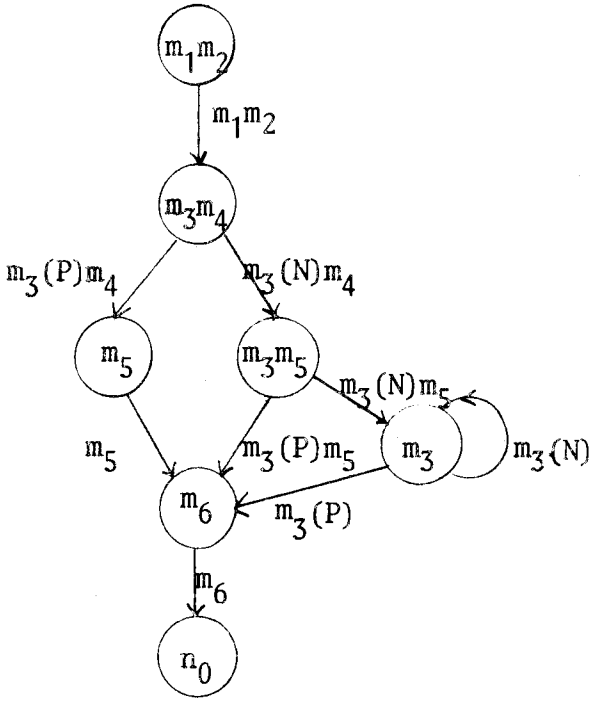


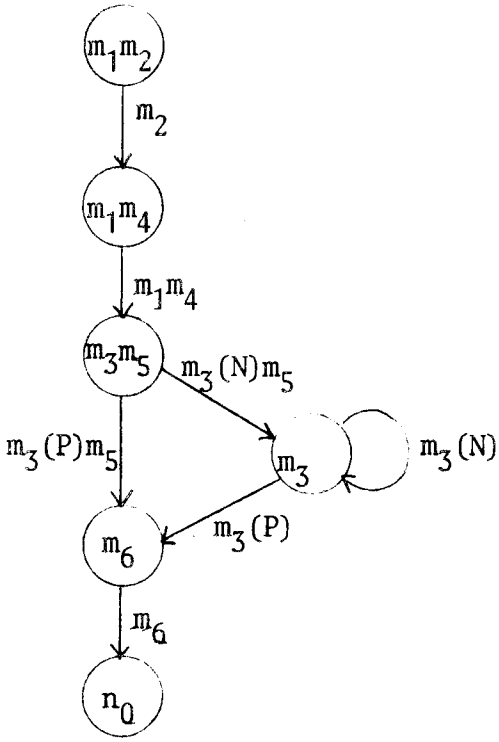
Il peut y avoir plusieurs sous-graphes minimaux c'est-à-dire plusieurs possibilités de fonctionnement dans un temps minimal donné.

C'est le cas pour le graphe d'état suivant :



Dans ce cas il y a deux sous-graphes minimaux





Il est possible d'associer à chaçun de ces graphes minimaux un automate déterministe

$$A = \{\Lambda, \Sigma, s_0, f, \{n_0\}\}$$

- Λ est l'ensemble des états de l'automate, Λ est en relation biunivoque avec l'ensemble Z des ensembles compatibles exécutés associés aux arcs du graphe minimal. Nous noterons $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_p\}$

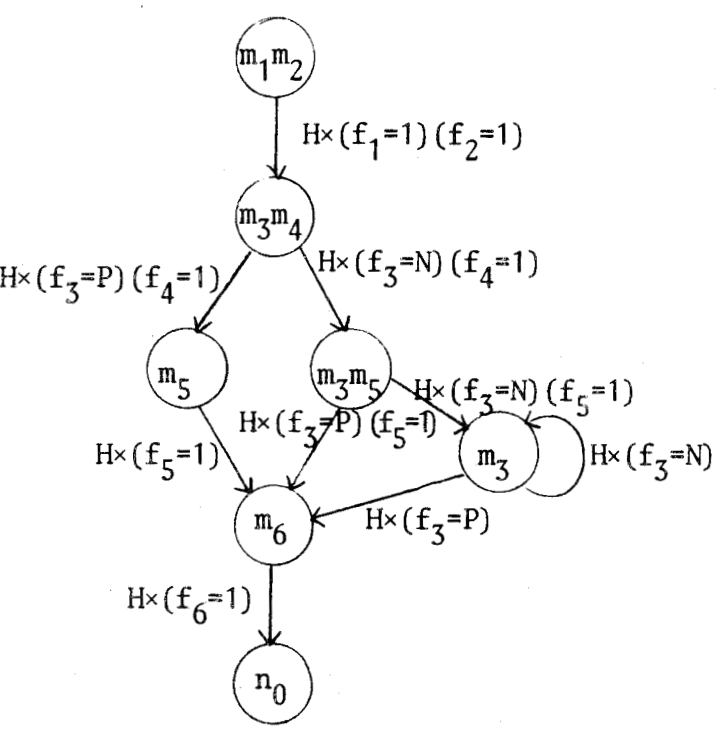
- Σ est l'alphabet d'entrée, $\Sigma = \bigcup_{Z_i \in Z} \Pi F(Z_i) \times H$, H étant le signal d'horloge de synchronisation

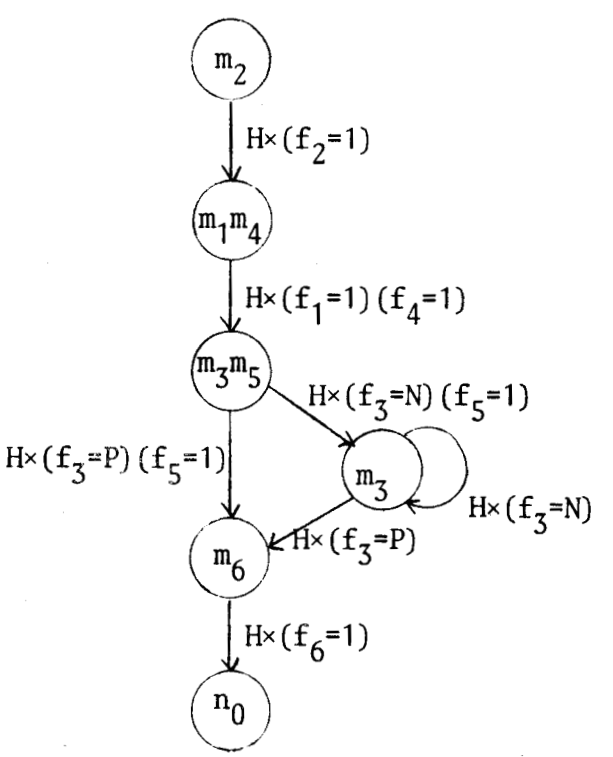
- s_0 correspond à l'ensemble compatible de départ

- n_0 correspond à l'ordre "non opération" : c'est l'état final unique de l'automate

- $\forall\, k \times h \in \Sigma$, $\forall\, \lambda_i \in S$, $f(\lambda_i, k\times h) = \lambda_j \Longleftrightarrow Z_j = \Delta(k, Z_i)$
Z_i étant en relation biunivoque avec λ_i et Z_j avec λ_j.

Ainsi pour l'exemple précédent on obtient les automates ci-après





On peut montrer que l'automate obtenu est minimum. A partir de ces automates minimaux, il est possible de réaliser le bloc de commande.

Il faut néanmoins remarquer qu'il n'est pas toujours intéressant de rechercher l'automate minimum, notamment s'il s'agit de construire un bloc de commande susceptible de contrôler différents processus. Il peut alors être préférable de choisir des automates non minimaux pour représenter les différents processus à contrôler, de façon à avoir un maximum de similitude entre les différents automates et par là même à envisager une structure plus compacte du bloc de commande.

2.3 - SYSTEMES PROGRAMMES

Dans l'introduction, les systèmes programmés ont été définis comme présentant un nombre potentiellement infini de fonctionnement, chaque fonctionnement étant caractérisé par un "programme", il reste à définir comment est constitué un programme.

Nous appellerons *système programmé* un système qui permet d'engendrer un nombre infini de processus. Donc, comme pour un processus, il sera possible de définir :

- des entrées
- des sorties
- des états internes
- des commandes.

Par contre, il n'est pas possible de définir d'enchaînements pré-établis entre ces commandes comme pour les processus.

Au contraire, l'utilisateur d'un système programmé doit pouvoir disposer d'un ensemble d "instructions", chacune d'elles correspondant à l'exécution d'une tâche, et il créera ses programmes en enchaînant ces instructions.

Les instructions peuvent être de deux types :

- des instructions se réduisant à une commande simple

- des instructions d'exécution de processus cycliques ou non cycliques (pour des raisons d'ordre technologique, ces processus sont pratiquement toujours des processus synchrones).

L'utilisateur définira son programme à l'aide de relations d'enchaînement entre instructions. Ces relations sont du même type que les relations d'enchaînement définissant le fonctionnement d'un processus classique, à la seule différence qu'il sera impossible d'exécuter deux instructions simultanément.

La définition complète d'un processus implique la détermination de conditions initiales et de variables d'entrée fixées par l'opérateur ; les programmes comporteront des données qui joueront soit le rôle de conditions initiales soit le rôle de données proprement dites.

En outre un système programmé devra avoir une commande de départ et chaque programme devra se terminer sur un ordre "non opération" final.

Pour un système programmé, il est également possible de définir :

- des ensembles incompatibles de commande
- des ensembles incompatibles de variables d'entrée booléennes
- des variables de sortie modifiant la valeur des variables d'entrée.

Les entrées, sorties, états internes et commandes étant définis de la même façon que pour un processus, il s'agit de formaliser correctement les instructions.

En ce qui concerne les instructions se réduisant à une commande, il n'y a pas de problème, par contre la formalisation des instructions d'exécution de processus est plus complexe.

De telles instructions ont pour but de lancer un processus. Ce processus est défini d'une façon classique par

- un ensemble de conditions initiales
- un ensemble d'enchaînements.

Deux instructions distinctes peuvent utiliser une même commande . A la limite, une commande peut à la fois définir une instruction et appartenir à un processus définissant une instruction, cependant nous n'envisagerons pas le cas d'instructions d'exécution de processus récursives.

Par contre on peut être amené à définir pour un processus des variables spécifiques. Suivant les conditions d'utilisation du processus dans un programme, ces variables d'entrée spécifiques prendront la valeur de variables du système programmé choisies par l'utilisateur.

Par exemple, si pour le processus correspondant à une instruction, on a défini dans l'ordre les variables spécifiques

$$e_{p_1}, e_{p_2}, \ldots, p_{p_j}$$

et si on appelle I_k l'instruction d'exécution du processus considéré dans un programme, I_k apparaîtra accompagnée des variables du système programmé utilisées comme valeur pour $e_{p_1}, e_{p_2}, \ldots, e_{p_j}$:

ainsi si le programmeur utilise l'instruction

$$I_k\ (e_{k_1} \ \ldots\ e_{k_p})$$

e_{p_1} prendra la valeur de e_{k_1} et ainsi de suite.

Il est possible d'envisager également d'imposer des relations d'ordres entre les instructions du type $\neg\ (I_j \in D(I_k))$. Généralement ces contraintes seront en relation avec les propriétés sémantiques du système programmé (ensembles de commandes incompatibles...)

Structuration du bloc de commande

Nous avons montré au paragraphe 2.2 qu'un processus synchrone pouvait être représenté par un automate déterministe

$$A = \{\Lambda, \Sigma, f, s_0, \{n_0\}\}$$

Si pour un système programmé nous disposont d'un ensemble

$$I = \{I_1, I_2, \ldots, I_N\}$$

d'instructions, nous pourrons lui faire correspondre un ensemble

$$A = \{A_1, A_2, \ldots, A_N\}$$

d'automates déterministes. Dans le cas d'une instruction se réduisant à une commande nous aurons un automate à deux états : un état correspondant à l'exécution de la commande et n_0.

Pour i=1,...,N, notons : $A_i = \{\Lambda_i, \Sigma_i, f_i, s_{0_i}, \{n_{0_i}\}\}$.

Le bloc de commande du système programmé devra être en mesure d'exécuter tous ces processus. Plus les différents automates correspondant aux instructions seront semblables plus le bloc de commande sera réduit. Lorsque pour un processus donné il existe plusieurs automates minimaux possibles il faudra choisir celui qui présente le plus de similitudes avec les automates correspondant aux autres instructions.

Considérons alors l'ensemble du bloc de commande comme un automate déterministe présentant N fonctionnements particuliers, correspondant chacun au fonctionnement d'un automate élémentaire A_i.

Pour cela nous définirons un ensemble de conditions booléennes supplémentaires

$$C = \{c_1\ c_2\ \ldots\ c_\ell\}$$

permettant de coder les N fonctionnements différents que nous noterons $d_1, d_2, \ldots, d_n$ de l'automate global.

$\forall\ i \in [1,N]$, pour reconnaître le fonctionnement d_i à exécuter, il est nécessaire de tester les éléments de C. Soit T(C) l'ensemble des tests à effectuer sur les éléments de C.

Nous noterons M(C) une partie finie de $T(C) \times \mathbb{N}$.

Dans ces conditions, A automate global représentant le bloc de commande sera défini comme suit

$$A = \{\Lambda, \Sigma, f, s_0, N_0\}$$

où

- $\Lambda = ((\bigcup_{i=1}^{N} \Lambda_i \times (\{d_1, d_2, \ldots, d_N\} \cup \emptyset)) \cup \{s_0\})$ *avec*

$$s_0 \notin \bigcup_{i=1}^{N} \Lambda_i \times (\{d_1, d_2, \ldots, d_N\} \cup \emptyset)$$

- $\Sigma = \mathbb{F}((\bigcup_{i=1}^{N} Z_i) \times (M(C) \cup \emptyset)) \times H$

- s_0 *état de départ correspond à l'exécution des éléments de* T(C)

- $N_0 = \{n_{0_1}, n_{0_2}, \ldots, n_{0_N}\}$ *ensemble des états finaux*

- $\forall A_i \in A$, λ_{ij}, $\lambda_{i_n} \in \Lambda_i$, $k_j^h \in \Pi F(Z_{ij})$, $Z_{ij}, Z_{i_h} \in Z_i$, Z_{ij} *en relation biunivoque avec* λ_{ij}, Z_{i_h} *avec* λ_{i_h} *et tels que :*

$$f_i(\lambda_{ij}, k_j^h) = \lambda_{i_h}$$

il y a deux possibilités :

i) si $\nexists A_s \in A$ *tel que* $\exists Z_{ij}, Z_{s_t} \in Z_s$, $Z_{s_t} \neq Z_{i_h}$, $k_j^h \in \Pi F(Z_{i_j})$ *avec* λ_{i_j} *en relation biunivoque avec* Z_{i_j} *et* λ_{s_t} *avec* Z_{s_t}, *et* $f_s(\lambda_{ij}, k_j^h) = \lambda_{s_t}$ *alors* $f(\lambda_{ij}, k_j^h) = \lambda_{i_h}$

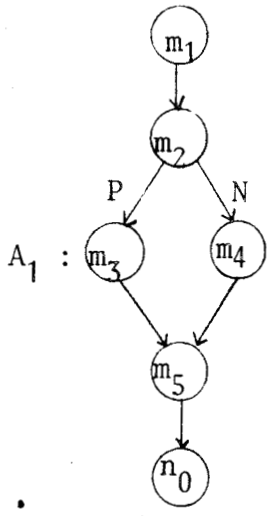
ii) si $\exists A_s \in A$ *tel que* $\exists Z_{ij}, Z_{s_t} \in Z_s$, $Z_{s_t} \neq Z_{i_h}$, $k_j^h \in \Pi F(Z_{ij})$ *avec* λ_{ij} *en relation biunivoque avec* Z_{ij} *et* λ_{s_t} *avec* Z_{s_t} *et* $f_s(\lambda_{ij}, k_j^h) = \lambda_{s_t}$

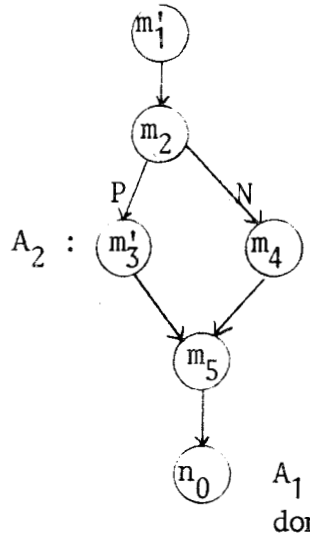
alors $f(\lambda_{ij} \times d_i, k_j^h \times \alpha) = \lambda_{i_h}$

$f(\lambda_{ij} \times d_s, k_j^h \times \beta) = \lambda_{s_t}$ *avec*

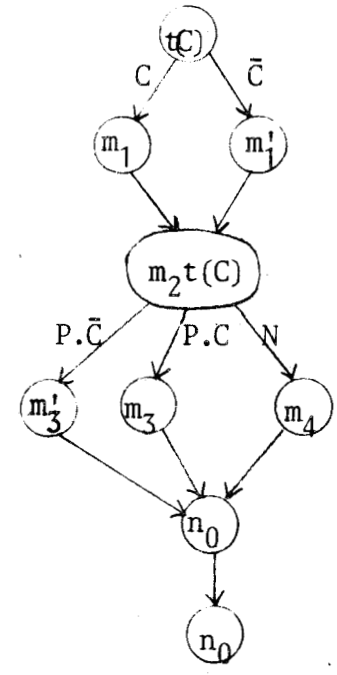
$$\alpha, \beta \in \Pi F(M(C))$$

Exemple





A_1 et A_2 donne A :



Une autre méthode de construction de A est envisageable. En effet, à partir du moment où on essaie de rassembler au maximum l'ensemble des automates correspondant aux diverses instructions, il n'est pas évident que le choix de l'automate minimum pour chaque instruction permette d'obtenir le bloc de commande le plus compact ; au contraire, il peut être interessant de redécomposer certains ensembles compatibles de manière à effectuer des regroupements d'états plus importants. Dans une telle hypothèse, une fois le ou les automates minimaux déterminés pour chaque instruction, on peut se fixer un taux d'allongement maximum x et considérer pour chaque instruction des différents sous-graphes dont le temps d'exécution est inférieur à $T(1+x)$, T étant le temps minimal d'exécution. C'est ensuite, à partir de ces sous-graphes que sera déterminée la structure du bloc de commande.

Chapitre III

LANGAGE DE DESCRIPTION DES SYSTEMES CYCLIQUES DESPROG

- Présentation -

Le langage DESPROG, que nous présentons dans ce chapitre, doit permettre la mise au point et la description des organigrammes de commande et de processus qui définissent le fonctionnement d'un processus ou d'un système cyclique. Les éléments de la définition d'un processus cyclique ont été donnés au chapitre I.

Il est cependant nécessaire de tenir compte des méthodes de travail actuelles des utilisateurs, qui sont centrées sur des notions d' "entrée" et de "sortie" acceptées dans un sens plus large qu'au chapitre I. Nous avons donc cherché un compromis entre l'exigence d'une adaptation aux méthodes de travail des utilisateurs potentiels et l'intérêt évident d'une maximalisation des possibilités de traitement sur ordinateur.

Si nous nous plaçons dans le cas d'un processus relativement simple qui ne puisse être décomposé en processus autonomes ou encore scinder en plusieurs processus irréductibles différents, il s'agit de définir un ensemble minimal d'éléments qui permette néanmoins de connaître ou de retrouver toutes les données nécessaires à la définition du processus considéré (cf. chapitre I) ; il importe également de définir les éléments qui présentent un intérêt du point de vue technologique.

Compte tenu des données du chapitre I et des contraintes d'utilisation, nous pouvons envisager la description en DESPROG des éléments suivants :

- les variables d'entrée, de sortie, les processus secondaires
- les conditions initiales et les enchaînements
- des informations supplémentaires d'ordre sémantique ou technologique.

3.1 - PRISE EN COMPTE DES VARIABLES ET DES PROCESSUS SECONDAIRES

La *prise en compte d'une variable de sortie* $s_i \in VS$ nécessite la connaissance des données suivantes :

- définition de la variable
- nom de la variable
- nom du moyen d'exécution contrôlé par l'intermédiaire de la variable
- type de la variable.

La définition de la variable est un commentaire destiné à faciliter la lecture d'une description. La notion de type sera définie par la suite.

A chaque moyen d'exécution contrôlé par le processus correspond une ou plusieurs variables de sortie. La connaissance du moyen d'exécution contrôlé par chaque variable de sortie permet de déterminer des ensembles incompatibles de commandes.

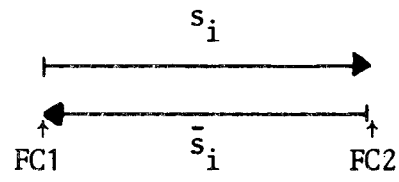
Dans ce qui suit, nous assimilons les registres booléens ou rationnels à des variables de sortie.

Une variable de sortie $s_i \in VS$ peut correspondre physiquement à 3 types de fonctionnement :

- en impulsion
- "simple effet"
- "double effet".

Dans le cas d'un fonctionnement "simple effet", s_i correspond à l'état "travail" d'un moyen d'exécution et $\bar{s}_i$ à l'état "repos" ; dans le cas d'un fonctionnement "double effet" $\bar{s}_i$ peut correspondre comme s_i à l'état "travail", l'état "repos" étant caractérisé par $\bar{s}_i$ et la valeur d'une variable de contrôle de fonctionnement.

Exemple : Un verin peut être commandé par la variable s_i dans un sens, par $\bar{s}_i$ dans l'autre sens, l'état repos du verin étant caractérisé par $\bar{s}_i$ et un signal de fin de course noté ici FC1.



"état repos" $\equiv \bar{s}_i \wedge FC1$

Dans ce qui suit, les variables de sortie comme les éléments qui leurs sont assimilés seront appelées "éléments commandés". On distingue donc 5 types d'élément commandé :

- variable de sortie à simple effet
- " " double effet
- " " impulsion
- mémoires à contenu booléen
- " " rationnel.

Certains processus possèdent un ou plusieurs ensembles de moyens d'exécution de même type tels que, lors du fonctionnement du processus, un seul élément d'un ensemble donné soit utilisé. Nous considèrerons qu'il existe des éléments commandés fictifs qui prennent, au départ du fonctionnement du processus, la valeur et le type d'un élément commandé sélectionné. Il suffira alors d'indiquer, dans la définition de l'élément, l'ensemble des éléments sélectionnables.

Dans le cas d'une variable de sortie nous distinguerons deux types de renseignement complémentaire possibles

- le nom du moyen d'exécution contrôlé
- un ensemble d'éléments sélectionnables.

Dans le cas d'un registre non booléen nous distinguerons deux types de renseignement complémentaire possibles

- la capacité du registre dans le cas d'un registre rationnel
- un ensemble d'éléments sélectionnables.

Enfin, il n'y aura pas de renseignement complémentaire pour les registres booléens, l'indication de la capacité est inutile.

La prise en compte des variables d'entrée nécessite la connaissance d'un nombre plus important d'informations ; en effet ces variables seront classées suivant leur rôle et leur origine. Nous distinguerons quatre rôles

- les contrôles de fonctionnement. C'est le cas de toutes les variables émises par les capteurs. Ce sont des variables booléennes,

- les conditions de fonctionnement. C'est le cas des variables booléennes affichées par l'opérateur,

- les paramètres. Ce sont des variables rationnelles affichées par l'opérateur ou émises par des appareils de mesure,

- les variables de cadencement. Ce sont des variables correspondant à des impulsions émises par un système de cadencement (horloge, temporisation, minuterie), des variables de départ qui commandent le départ du déroulement du processus, ou encore des variables de sortie émises par un autre bloc de commande.

Il y a quatre origines possibles, selon qu'un signal est émis :

- à partir du pupitre par l'opérateur,
- par un capteur ou un appareil de mesure,
- par un système de cadencement,
- par un autre bloc de commande.

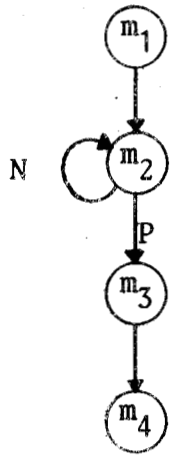
Rôle et origine sont liés. Ainsi un paramètre ne peut avoir pour origine un signal émis par un système de cadencement, mais uniquement un signal émis par un capteur ou à partir du pupitre ; les contrôles de fonctionnement sont obligatoirement issus de capteurs etc...

Pour chaque variable d'entrée il peut être défini une plage de valeurs dans laquelle la variable d'entrée considérée prend sa valeur. Cette plage sera définie soit par ses bornes soit par un ensemble de valeurs ou de variables. Seuls les paramètres nécessitent obligatoirement la définition d'une plage de valeur.

Lorsqu'une variable de contrôle de fonctionnement est définie par une plage de valeurs, cette dernière est nécessairement un ensemble de variables et la variable de contrôle prise en compte correspond à un moyen d'exécution sélectionné.

D'autre part, afin de faciliter la description d'un processus, il peut être intéressant de définir un ensemble de processus secondaires commandant l'exécution d'opérations bien déterminées et d'usage courant. Ces processus secondaires correspondraient alors à des solutions technologiques préétablies et leur utilisation dans la description du processus principal ne nécessiterait que l'indication de leurs noms.

Exemple : Le processus secondaire correspondant à un compteur peut se définir comme ci-après



m_1 = commande de déverrouillage du compteur

m_2 = test d'attente de l'impulsion de comptage

m_3 = incrémentation du registre de comptage

m_1 = commande de verrouillage du compteur

Ce processus secondaire se réduit à une séquence linéaire que l'on peut définir par un nom. Seul ce nom apparaîtra dans la description du processus principal.

De tels processus secondaires seront appelés *processus types*. Suivant les besoins du secteur concerné, les utilisateurs pourront déterminer une liste de processus secondaires-types. Ces processus types seront définis par

- une définition
- un nom
- un type
- des variables de définition qui déterminent le fonctionnement du processus type
- des variables émises durant l'exécution du processus type.

Eventuellement les variables de définition et les variables émises pourront se voir attribuer une plage de valeur.

Les variables émises par un processus type seront considérées par le processus principal comme des variables d'entrée.

Les processus types seront utilisés dans la description du déroulement du processus principal comme des processus secondaires.

Exemple : Considérons le cas du positionnement à partir d'une position quelconque d'un élément mobile sur un point déterminé d'un axe, la suite des opérations exécutées est la suivante

- détermination, par comparaison des positions de départ et d'arrivée, du sens du mouvement
- commande du mouvement
- détection de la proximité du point de positionnement
- positionnement précis sur ce point.

Pour un processus type de positionnement, les variables de définition seront : la ou les deux variables de sortie commandant le mouvement
la variable d'entrée correspondant au point de positionnement.
Il n'y a pas de variable émise par un tel processus type.

Au chapitre I nous avons vu l'intérêt que présentait pour la simplification de la description d'un processus la recherche et l'étude séparée des processus secondaires.

Donc, si un certain nombre de processus secondaires apparaissent plusieurs fois dans le processus étudié, ou si la définition de sous-processus secondaires facilite le travail de description ou de compréhension du processus étudié, il doit être possible de définir ces processus secondaires cycliques ou non cycliques avant de décrire le déroulement du processus étudié.

Pour chaque processus secondaire utilisé il est nécessaire de définir

- les éléments commandés
- les variables d'entrée
- les processus types utilisés
- les processus secondaires
- les relations d'enchaînement qui caractérisent le déroulement du processus secondaire étudié.

En bref, il s'agira de décrire le processus secondaire de la même façon qu'un processus principal, hormis ce qui concerne les variables de commande de départ qui n'existent pas toujours pour un processus secondaire (cf. chapitre I).

En outre, un processus secondaire cyclique peut être sélectionné parmi plusieurs processus secondaires cycliques définis auparavant.

La connaissance du type du processus secondaire intervient dans les procédures de contrôle du processus irréductible étudié, car dans le cas d'un processus secondaire non cyclique, l'état du processus étudié après exécution du processus secondaire est différent de celui avant cette exécution.

Il n'est pas possible de définir un processus secondaire non cyclique comme étant sélectionné parmi un ensemble de processus secondaires non cycliques ; en effet, dans le cas contraire, il serait impossible de contrôler l'exécution du processus étudié puisque l'état de ce processus après exécution du processus secondaire ne pourrait être déterminé.

Il est possible également d'envisager des variables spécifiques à ces processus secondaires, ce qui signifierait qu'au cours du déroulement du processus principal, le processus secondaire considéré pourrait fonctionner sur différents jeux de variables.

Au cours de la description du mode secondaire, les variables spécifiques auront alors été définies comme étant sélectionnables parmi un ensemble d'éléments, ces éléments étant eux mêmes définis dans la description du processus principal.

L'ensemble des éléments nécessaires à la reconstitution d'un ensemble d'ordre M le plus réduit possible étant défini, nous allons maintenant étudier la description du fonctionnement du processus c'est-à-dire la définition des enchaînements.

2 - DEFINITION DES ENCHAINEMENTS

Définir les enchaînements revient à donner les indications nécessaires à la reconstitution des éléments de M×F×M.

Les éléments de C_0 seront définis à partir des éléments commandés, ceux de TB à partir des variables d'entrée et des registres.

Cependant, d'un point de vue descriptif, il n'est pas intéressant de traiter de la même façon les définitions des éléments de C_0 et de TB, car cela conduit à des descriptions très lourdes. D'une façon générale, nous considèrerons, du point de vue

descriptif, les enchaînements entre éléments de C_0, c'est-à-dire que les éléments de base de la définition des enchaînements seront ceux de

$$R_1^+ = (C_0 \times \mathbb{N})^2 \times F^+ \quad \text{en notant :}$$

$$F^+ = \{ f_{i_1} \times \ldots \times f_{i_n} \in \Pi F \mid \exists\, m_{i_1},\ldots,m_{i_{n+1}} \in F : (m_{i_j},f_{i_j},m_{i_{j+1}}) \in R \text{ pour } j=1,\ldots,n+1$$

Ainsi, si

$$(m_i,m_j,f_k) \in R_1^+ \qquad \text{avec} \qquad m_i,m_j \in C_0 \times \mathbb{N} \qquad \text{et} \qquad f_k \in F$$

cela implique qu'il existe $m_h \in M$ tel que l'on ait

$$m_h = (c_r,n_h),\ c_r \in TB \quad \text{et}$$

$$(m_i,m_h) \in R \qquad \text{et} \qquad (m_h,f_k,m_j) \in R.$$

Un tel mode de description nécessite deux types de phrase :

- *les phrases commandes* décrivant des exécutions de commande c'est-à-dire d'éléments de C_0,

- *les phrases-tests* indiquant les résultats de tests.

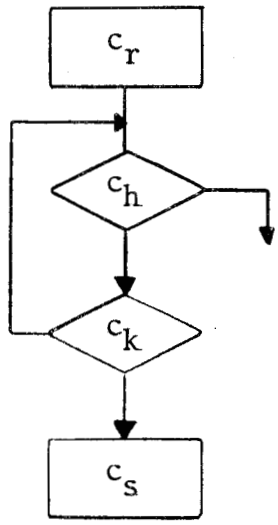
Cependant dans le cas d'un cycle ne comprenant que des tests, la méthode exposée ci-dessus ne permet plus une description correcte des enchaînements, d'où la nécessité d'ajouter aux phrases commandes une phrase décrivant l'exécution d'un test.

Exemple : Si $\exists\, m_g, m_\ell \in M$, $m_g = (c_r,n_g)$, $m_\ell = (c_s,u_\ell)$ et $c_r,c_s \in C_0$

$$\exists\, m_i, m_j \in M,\ m_i = (c_h,n_i),\ m_j = (c_k,n_j) \text{ et}$$

$$c_h,\ c_k \in TB$$

tels que $(m_j,f_{j_1},m_i) \in R$, $(m_i,f_{i_1},m_j) \in R$, $(m_g,m_i) \in R$ et $(m_j,f_{j_2},m_\ell) \in R$



nous écrirons

$$(m_g, m_\ell, f_{i_1} \times f_{j_2}) \in R_1^+$$

$$(m_g, m_i, f_{i_1} \times f_{j_1}) \in R^+ = (C_0 \times N) \times (C \times N) \times F^+.$$

Donc en fait, les éléments de base de la définition des enchaînements seront les éléments de $(C_0 \times N) \times (C \times N) \times F^+ = R^+$.

Remarque : $\forall\, \alpha \in F^+$, α n'est pas uniquement une fonction booléenne, mais traduit également l'ordre chronologique dans lequel sont effectués les tests qui reconnaissent la fonction booléenne décrite par α.

Il y aura cinq types de phrase commande : toute commande c_i de C peut correspondre à :

- l'exécution d'un test,
- une attribution de valeur,
- une opération interne,
- l'exécution d'un processus secondaire ou d'un processus type,
- l'émission d'une impulsion.

Comme il s'agit de décrire les éléments de M et non pas ceux de C, il faudra adjoindre à chaque phrase commande un indice indiquant l'occurrence de la commande décrite. Cet indice sera inutile dans le cas où la commande considérée n'est utilisée qu'une seule fois dans le déroulement du processus.

Les phrases tests décrivent le résultat des tests ; nous en distinguerons deux types, selon qu'elles indiquent le résultat d'un test normal ou d'un test d'attente

A un test d'attente correspondent les enchaînements suivants :

$$m_i \in M,\ m_i = (c_j, n_j),\ c_j \in TB$$
$$(m_i, x, m_i) \in R \quad x \in [P,N]$$
$$(m_i, y, m_k) \in R \quad y \neq x,\ y \in [P,N],\ m_k \neq m_i,\ m_k \in M.$$

Il reste à définir comment s'enchaîneront ces phrases. D'une façon générale, deux phrases commandes et une phrase test seront associées de façon à décrire un enchaînement. Par contre, il peut également arriver que l'exécution successive de deux commandes soit subordonnée à l'exécution de plusieurs tests en cascade ; dans ce cas il sera nécessaire d'associer plusieurs phrases tests en série.

Une description simplifiée des enchaînements sera prévue lorsqu'à l'exécution d'une commande succèdera l'exécution simultanée de plusieurs commandes ou que, plus généralement l'exécution simultanée de plusieurs commandes succèdera à l'exécution simultanée d'un autre ensemble de commande.

Enfin les séquences linéaires pourront être décrites par une suite alternée de phrases commandes et de phrases tests et il devra exister obligatoirement des phrases commandes indiquant le début et la fin du déroulement.

Il reste à définir un moyen de description des variations inactives de variables d'entrée permettant la description des organigrammes de processus. A cet effet, nous introduirons la notion de phrase de repositionnement.

Une phrase de ce type décrira le repositionnement d'une variable d'entrée sur une valeur donnée après exécution d'une commande modifiante ; elle sera donc placée immédiatement après la phrase décrivant la commande modifiante. Lorsque plusieurs variables d'entrée se repositionnent après une même commande modifiante, il faudra utiliser plusieurs phrases de repositionnement.

Avant de décrire les enchaînements, il faudra évidemment définir les conditions initiales, en indiquant pour chaque variable d'entrée, chaque variable de sortie et chaque registre, sa valeur ou son contenu à l'instant initial.

Il faut aussi fixer les valeurs initiales des variables d'entrée de contrôle de fonctionnement. Par contre, il n'y a pas à initialiser les paramètres

et les conditions de fonctionnement ; de même, les variables rationnelles émises par des processus types ou les variables de cadencement n'ont pas à être initialisées puisque l'on considère qu'au temps initial, elles sont nulles.

En ce qui concerne les éléments commandés, sont à initialiser :

- les variables de sortie
- les mémoires

il n'est évidemment pas question d'initialiser les processus secondaires ou les processus types ; les variables initialisées dans la description des processus secondaires non cycliques seront considérées comme faisant partie des conditions initiales du processus étudié, ce qui sera inutile dans le cas des processus secondaires cycliques.

Enfin, l'initialisation d'une mémoire consiste à en fixer son contenu.

3 - INFORMATIONS SUPPLEMENTAIRES

Outre la description du fonctionnement du processus, il importe de définir d'une part les propriétés sémantiques (cf. paragraphe I.3), d'autre part des conditions de fonctionnement inhabituelles qu'il serait inopportun de décrire simultanément au déroulement normal du processus.

En ce qui concerne les propriétés sémantiques, nous distinguons trois sortes d'information à prendre en compte.

- *définition des ensembles incompatibles de commandes*. Il s'agit d'indiquer à partir des éléments commandés, les commandes qui ne peuvent être exécutées simultanément. Les ensembles incompatibles peuvent être décrits à l'aide de phrases de commande. Suivant des problèmes d'ordre technologique, nous distinguerons deux types d'ensembles incompatibles :

- ceux entraînant un arrêt du déroulement du processus dès leur apparition,

- ceux dont l'existence doit être rendue impossible par modification du système.

Il est inutile de définir les ensembles incompatibles relatifs aux registres.

Pour décrire les ensembles incompatibles de commande, nous utiliserons des phrases commandes auxquelles sera adjointe une mention indiquant le type de l'ensemble incompatible décrit.

- *définition des ensembles incompatibles de variables d'entrée* [1]. Il s'agit d'indiquer les ensembles incompatibles de variables booléennes de contrôle de fonctionnement, ces données étant nécessaires à l'élaboration de l'organigramme de processus.

- *définition des ensembles de commandes modifiant la valeur* [1] *des variables d'entrée booléennes*. Il s'agit ici des variables de contrôle de fonctionnement. A chacune de ces variables est adjoint un ensemble de commandes modifiant la valeur de ces variables.

Sur tout système il existe généralement des mesures de sécurité qui permettent d'éviter certains types de fonctionnement aberrants susceptibles d'entraîner une détérioration de l'équipement.

Ces mesures de sécurité consistent à tester des variables d'entrée booléennes dites *variables de sécurité*. Le passage à 1 d'une de ces variables indique un danger et entraîne l'arrêt total du déroulement du processus.

Une variable de sécurité se définit à l'aide d'un nom si elle est d'origine extérieure (capteur, autre bloc de commande) ou comme une suite de phrases tests si la nature du danger à détecter nécessite la connaissance de la valeur de plusieurs variables d'entrée ; un commentaire accompagnera la définition de manière à en faciliter la lecture. Dans la plupart des cas, la variable de sécurité sera testée à chaque enchaînement, néanmoins il arrive qu'un test systématique soit inutile et qu'il suffise de le faire en quelques points précis du déroulement. Dans ce cas, il importe de préciser ces points de test. Ils seront repérés par la phrase commande qui précède immédiatement le test considéré, cette phrase commande étant éventuellement suivie d'une ou plusieurs phrases tests.

D'une façon similaire aux variables de sécurité, il existe des variables d'entrée booléennes dites *variables d'intervention*. Le passage à 1 d'une variable d'intervention indique un fonctionnement anormal ce qui entraîne l'exécution d'un *processus d'urgence* dont le rôle est de supprimer les conditions anormales de fonc-

---

(1) Les informations sémantiques relatives aux variables d'entrée seront insérées dans les déclarations de variable d'entrée.

tionnement. Ces processus d'urgence sont nécessairement des processus cycliques puisqu'après leur exécution on doit retrouver les conditions de fonctionnement normales existant avant le passage à 1 de la variable d'intervention. La valeur de chaque variable d'intervention sera testée à chaque enchaînement ; en effet, si ce n'était pas le cas la variable d'intervention pourrait être considérée comme la variable de départ d'un processus secondaire cyclique.

En outre,des commentaires seront associés aux noms de la variable d'intervention et du processus d'urgence afin de faciliter la compréhension de la description.

Il est ensuite nécessaire de décrire le processus d'urgence, de la même façon qu'un processus normal.

Un certain nombre d'éléments commandés est généralement doublé d'une commande manuelle. Il s'agit alors de donner la liste des éléments commandés de ce type. Dans le cas des mémoires, deux possibilités différentes de commande manuelle peuvent se présenter :

- une simple remise à zéro
- la possibilité de mémoriser manuellement une valeur quelconque.

Il s'agit également de préciser s'il existe une commande manuelle d'arrêt immédiat.

Les différentes variables étant déclarées, les enchaînements définis ainsi que diverses autres indications connues, la mise au point de l'organigramme de processus est possible, il ne reste plus qu'à indiquer si le processus décrit est un synchrone ou asynchrone. Dans le cas du processus synchrone, il est nécessaire de préciser la période d'horloge de base

.4 - ANALYSE D'UN PROCESSUS COMPLEXE

Dans le cas d'un processus complexe, le type de description proposée dans ce chapitre risque d'être mal adapté ; il est alors intéressant de se donner la possibilité de "découper" le processus initial en processus plus simples et donc plus faciles à étudier.

D'après ce que nous avons dit au chapitre I, l'analyse d'un processus suppose que l'on réponde aux deux questions suivantes :

i) le processus peut-il être décomposé en processus autonomes ne possédant en commun aucun élément commandé ?

ii) pour chaque processus autonome ou pour le processus global, si celui-ci n'a pu être décomposé, peut-on définir plusieurs processus irréductibles ?

Il s'agit alors de décrire tous les processus irréductibles (qu'il y en ait un ou plusieurs) de chaque processus autonome (ou du processus initial s'il n'est pas décomposable en processus autonome) suivant la méthode présentée dans ce chapitre.

Ensuite le fonctionnement global du processus initial doit être décrit à partir des relations entre les divers processus irréductibles (cf. § I.3). Cette description sera complétée par la liste des différents verrouillages [1], elle comprendra :

- la description des enchaînements (à l'aide de mêmes phrases que celles utilisées pour décrire les enchaînements d'ordre)

- la définition des ensembles incompatibles de processus et de variables d'entrée

- la liste des processus modifiant la valeur de variables d'entrée

- la liste des verrouillages à effectuer.

La définition des connexions (ψ dans le § I.2) sera intégrée à la description des différents processus irréductibles, chaque variable appartenant à plusieurs descriptions étant repérée comme telle au moment de sa définition.

Il est possible que dans la description du fonctionnement d'un processus global, un processus P soit enclenché et jamais déclenché parce que, par exemple, il émettra au cours de son déroulement des variables d'interconnexion. Cependant, il sera considéré comme étant exécuté en totalité avant la fin du fonctionnement global.

---

(1) Nous dirons qu'un processus en cours de fonctionnement verrouille une variable de sortie utilisée par un autre processus à l'arrêt lorsqu'il force la valeur de cette variable à zéro.

DEFINITION DU LANGAGE DESPROG

Une fois définies la liste des éléments à décrire et la consistance des descriptions élémentaires correspondantes, il importe d'une part de choisir un plan de description qui se rapproche le plus possible de la démarche effectuée par un automaticien pour mettre au point un organigramme de processus ou de commande, d'autre part de préciser le contenu de chaque description élémentaire. Pour ce faire nous allons définir la structure du langage DESPROG compte tenu des indications du paragraphe 3.4.

Nous utiliserons la forme normale de BACKUS [1] pour décrire les règles de production du langage DESPROG.

Dans ce chapitre, nous utiliserons en outre les métarègles suivantes sui permettent de simplifier les descriptions ultérieures.

< *Liste de* (α)s > ::= α < *Liste de* (α)s > | < *vide* >
< *Suite de* (α)s > ::= α < *Liste de* (α)s >
< *Série de* (α)s > ::= α < *Suite de* (α)s >
< *Ensemble de* (α)s > ::= (α < *Succession de* (α)s >)
< *Succession de* (α)s > ::= < *Liste de* (α)s > | < *Suite de* (α)s >

α pouvant être remplacée par un élément *décomposable* quelconque.

4.1 - STRUCTURE GENERALE

La structure générale est définie par les règles suivantes :

< *Description d'un processus* > ::= < *Définition d'un processus* >
< *Définition et description des processus autonomes* >
< *Description du fonctionnement global* >

Quatre notions de base apparaissent dans la structure du langage

- les descriptions (en particulier celles d'enchaînements),
- les définitions (en particulier celles des processus),
- les déclarations (permettant de définir des variables d'entrée, de sortie, processus-types etc...)
- les indications (permettant de définir un certain nombre de propriété supplémentaires notamment les propriétés sémantique

La définition d'un processus comporte quatre ou cinq éléments suivant les cas

< *Définition d'un processus* > ::= DG < *Commentaires* > < *Nom de processus* > < *Code* > < *Type* P >

< *nom de processus* > ::= < *Nom* >

< *Type* P > ::= < *vide* > | AS | SY < *durée de la période* > (1)

- les commentaires facilitent la compréhension de la description par son utilisateur

- le code est un ensemble de chiffres, de lettres et de "/". Il est destiné à servir de support d'information à différents domaines : date, code, client, code marché, code fabrication etc...

- le type P permet de savoir si le processus est synchrone (SY) ou asynchrone (AS), il faut cependant remarquer que dans le cas où un processus est décomposé en sous-processus de types différents, on ne peut lui attribuer de type. Si le processus est synchrone, la définition est complétée par la durée de la période de l'horloge de synchronisation, exprimée en seconde .

---

(1) Les notions de commentaire, nom, code, nombre seront définies ultérieurement.

"Définition et description des processus autonomes" est définie par les règles :

< *Définition et description des processus autonomes* > ::=
< *Définition et description de processus irréductibles* > |
DPA < *Série de définition d'un processus* > < *Série de description d'un processus autonome* >

La définition de processus autonomes correspond au découpage du processus étudié en processus autonomes, cet élément de la description n'apparaît donc pas systématiquement puisque tous les processus ne sont pas décomposables. La définition des processus autonomes contiendra au moins deux "définitions d'un processus", un processus autonome se définissant comme un processus, toutefois si un type a été attribué a été attribué au processus initial, ce type est celui de tous les processus autonomes, il est donc inutile de le rappeler à chaque définition. Par contre, dans le cas où il n'a pas été attribué de type au processus principal, il convient de définir celui de chaque processus autonome.

Chaque processus autonome défini doit ensuite être décrit. L'ordre de description des processus autonomes est sans importance :

Les règles de cette description sont les suivantes :

< *Description d'un processus autonome* > ::= DES < *Nom de processus* >
< *Définition et description de processus irréductibles* >

La description d'un processus autonome commencera par le nom du processus décrit.

Dans le cas où le processus initialement étudié est indécomposable, il suffit de passer directement à la définition des processus irréductibles.

La définition et la description des processus irréductibles répondent aux règles suivantes :

< *Définition et description de processus irréductibles* > ::=
< *Description de base* > |
< *Série de définitions d'un processus irréductible* >

< *Série de description d'un processus irréductible* >
< *Définition d'un processus irréductible* > ::= < *Commentaires* > < *Nom de processus* >
< *Description d'un processus irréductible*> ::= DEP < *Nom de processus* > < *Description de base* > (1)

Chaque description de processus irréductible est précédée du nom de l'élément étudié.

La liste de définitions de processus irréductible doit comporter, si elle existe, au moins deux définitions.

Un processus irréductible se définit simplement par un nom puisqu'il conserve le type et le code du processus autonome ou du processus initial dont il est issu.

Lorsque le processus initial est indécomposable et irréductible la description de base s'enchaîne directement sur la définition du processus, c'est-à-dire une structure simple de la forme :

< *Définition d'un processus* > < *Description de base* >

Lorsque le processus autonome considéré est irréductible, apparaît alors une structure simplifiée de la forme :

DEP < *Nom de processus* > < *Description de base* >

Les descriptions partielles relatives aux processus autonomes déclarés étant terminées, il convient de décrire le fonctionnement global du processus

< *Description du fonctionnement global* > ::= FDG |
DEG < *Description des relations entre processus* >
< *Liste d'indications supplémentaires* > FDG

< *Description des relations entre processus* > ::= < *vide* > |
< *Description des enchaînements de processus* >

---

(1) La notion < *description de base* > sera définie au paragraphe suivant.

< *Indications Supplémentaires* > ::=
< *Indication des processus incompatibles* > |
< *Indication des variables d'entrée incompatibles* > |
< *Indication des variables d'entrée modifiées* > |
< *Indication des verrouillages* > (1)

La description du fonctionnement global se termine par un caractère de fin de description FDG. Dans le cas de l'étude d'un processus indécomposable et irréductible la description du fonctionnement global se réduit à FDG. Dans la plupart des cas, seule une partie des indications supplémentaires existera et même lorsqu'un processus est décomposable en processus autonomes reliés entre eux uniquement par des relations d'interconnexion, le fonctionnement global se réduit à FDG . L'ordre d'écriture des indications supplémentaires est sans importance.

4.2 - STRUCTURE D'UNE DESCRIPTION DE BASE

La description de base est l'élément essentiel d'une description de processus. Suivant les indications données au chapitre III, la description de base d'un processus sera structurée comme suit :

< *Description de base* > ::= < *Suite de déclarations de variables* >
< *indication des conditions initiales* > < *description des enchaînements* > < *liste d'indications complémentaires* >
FDB

En effet, il semble logique de définir d'abord les variables apparaissant dans les enchaînements, ensuite de les initialiser puis de décrire les enchaînements et enfin de donner des indications complémentaires sur le fonctionnement du processus.

Nous appellerons déclaration la définition d'un élément nécessaire à la description d'un processus. D'après le chapitre III, il y a quatre sortes d'éléments à déclarer. Nous avons donc introduit la règle suivante :

---

(1) Ces notions sont définies ultérieurement.

< *Déclarations de variables* > := < *Déclarations d'éléments commandés* > |
< *Déclarations de variables d'entrée* > |
< *Déclarations de processus secondaires* > |
< *Déclarations de processus-types* >

Quel que soit le type de l'élément déclaré, la structure de la déclaration reste la suivante :

<*Déclaration de* (α)s> ::= <*Indicateur d'*(α)> <*Déclaration générale d'un* (α)>
< *Liste de déclaration d'un* (α) > < *Liste d'indications sémantiques sur les* (α)s >

< *Déclaration générale d'un* (α) > ::= < *Déclaration d'un* (α) > |
< *Déclaration d'un* (α) *déjà défini* >

< *Déclaration d'un* (α) *déjà défini* > ::= R < *Nom de variable* > DANS <*nom de processus*>
< *nom de variable* > ::= < *nom* >

Lorsqu'une variable a déjà été déclarée dans la description d'un autre processus irréductible, il suffit d'indiquer son nom ainsi que le nom du processus dans la description duquel la variable considérée a été déclarée pour la première fois.

Dans le cas des *éléments commandes*, nous aurons une structure de la forme suivante :

< *Indicateur d'élément commandé* > ::= DCC
< *Déclaration d'un élément commandé* > ::= < *commentaires* >
< *Nom* > < *Type* C >
< *Type* C > ::= < *Type* S > < *Moyen d'exécution commandé* > |
< *Ensemble d'éléments sélectionnables* > | RB | RR < *nombre* >
< *Elément sélectionnable* > ::= < *Nom* >
< *Type* S > ::= SE | DE | IP

Il y a cinq types "C" :

simple effet noté	SE
double effet "	DE
impulsion "	IP
mémoire booléenne notée	RB
- rationnelle "	RR

Le moyen d'exécution commandé sera défini par un nom éventuellement accompagné d'un commentaire et d'un type, soit :

< *Moyen d'exécution commandé* > ::= OC < *commentaires* > < *Nom* > < *Type* M >

Si un élément commandé est sélectionné, il est évident que l'ensemble des éléments sélectionnables devra avoir une cardinalité supérieure ou égale à deux.

Les règles de dérivation de "Type M" seront définies par l'utilisateur. Pour les *variables d'entrée*,les déclarations auront la structure suivante

< *Indicateur de variable d'entrée* > ::= DCE
< *Déclaration d'une variable d'entrée* > ::=
    < *Commentaires* > < *Nom* > < *Rôle et origine* > < *Plagide* > < *Comovide* >
< *Rôle et origine* > ::= CF / CD / VC < *Cadenceur* > / PA < *Origine* >
< *Cadenceur* > ::= PU / < *Emetteur* > < *Nom* >
< *Emetteur* > ::= SC / AP
< *Origine* > ::= PU / EX / AP < *Nom* >
< *Plagide* > ::= < *Plage de Valeurs* > / < *vide* >
< *Comovide* > ::= < *Groupe de commandes modifiantes* > / < *vide* >

A la déclaration d'une variable d'entrée proprement dite définie d'après les indications du chapitre III, il faut ajouter des indications sémantiques relatives aux ensembles de variables d'entrée booléennes incompatibles et définir également pour chaque variable la liste des commandes modifiant la valeur de cette variable.

Dans la syntaxe précédente, les quatre rôles possibles d'une variable d'entrée sont notés par :

- contrôle du fonctionnement : CF
- condition de fonctionnement : CD
- paramètre rationnel : PA
- variable de cadencement : VC.

Suivant son rôle et en fonction du processus étudié les quatre origines possibles pour une variable d'entrée notées pour :

- le pupitre : PU (correspondant aux rôles CD, PA, VC)
- capteurs ou appareils de mesure : EX (correspondant aux rôles CF et PA)
- système de cadencement : SC (Nom)
  le nom est celui du système émetteur de la variable
  (correspondant au rôle VC)
- un autre bloc de commande : AP (Nom)
  le nom est celui du processus générateur de la variable.
  (correspondant aux rôles PA et VC).

Une plage de valeurs peut être définie

i) par ses bornes
ii) comme un ensemble de valeurs ou de variables

d'où la structure suivante :

< *Plage de valeurs* > ::= (< *Nombre* > *A* < *Nombre* >) | < *Ensemble de valeurs* >
< *Valeur* > ::= < *Nom* > | < *Nombre* >

A priori, rien n'empêche qu'un ensemble de valeurs comporte à la fois des noms et des nombres. Les noms seront néanmoins ceux de variables d'entrée précédemment déclarées, ayant toutes le même rôle et la même origine. Dans certains cas, la plage de valeur peut ne comporter qu'un seul élément. Quand une plage est définie par ses bornes, il est nécessaire d'indiquer la cardinalité de l'ensemble de valeurs en imposant une précision sur les bornes.

Exemple : (0 A 12) suppose l'existence de 13 valeurs possibles
(10,0 A 13,0) suppose l'existence de 121 valeurs possibles.

Pour les variables de rôle CF il faut indiquer l'ensemble des commandes modifiant la valeur de la variable déclarée.

Nous aurons :

< *Groupe de commandes modifiantes* > ::= CM < *Ensemble des commandes modifiantes* >
< *Commande modifiante* > ::= ¬< *Nom* > | < *Nom* >

Les noms apparaissant dans un ensemble de commande modifiante correspondront à des variables de sortie du type SE, DE ou IM ou éventuellement à des processus

secondaires ou des processus types. La liste des ensembles de variables d'entrée incompatibles n'apparaît qu'à la fin d'un ensemble de déclarations de variables d'entrée et seules des variables booléennes peuvent apparaître dans ces ensembles. Ceux-ci constituent une indication sémantique structurée comme suit :

< *Indication sémantique sur les variables d'entrée* > ::=
VI < *Suite d'ensembles de variables incompatibles* >
< *Variable incompatible* > ::= < *Nom* >

La cardinalité d'un ensemble de variables incompatibles est évidemment supérieure ou égale à 2. Le genre d'ensemble n'existe pas nécessairement.

En ce qui concerne les *processus secondaires*, il y en a deux sortes :

- les processus secondaires cycliques que nous noterons PC
- " " " non cycliques que nous noterons PA.

Ces processus seront décrits de la même façon que des processus irréductibles suivant les règles indiquées ci-dessous :

< *Indicateur de processus secondaire* > ::= DSP
< *Déclaration d'un processus secondaire* > ::=
< *Commentaires* > < *Nom* > < *Type* F > < *Groupe de variables spécifiques* >
< *Type* F > ::= < *Type cyclique* > < *Type acyclique* >
< *Type cyclique* > ::= PC < *Conception* >
< *Type acyclique* > ::= PA < *Description de base* >
< *Conception* > ::= < *Ensemble d'éléments sélectionnables* > | < *Description de base* >
< *Groupe de variables spécifiques* > ::= < *Ensemble de variable spécifique* > | < *vide* >
< *Variable spécifique* > ::= < *Nom* >

Il ne reste plus qu'à indiquer la structure des déclarations de *processus types :*

< *Indicateur de processus type* > ::= DCT

L'ordre de déclaration des variables émises, comme celui des variables de définition, est imposé par le type de processus. Une variable de définition comme une variable émise est déclarée par un nom auquel est éventuellement associée une plage de

valeurs ; c'est notamment le cas pour les processus types de mesure. Dans ces conditions les règles de production utilisables seront les suivantes :

< *Déclaration d'un processus type* > ::=
< *Commentaire* > < *Nom* > < *Type* T > < *Vavide* > < *Vavide* >
< *Vavide* > ::= < *Ensemble de variable* > | < *vide* >
< *Variable* > ::= < *Nom* > < *Plagide* >

les règles de dérivation de "Type T" seront définies par l'utilisateur.

Après les déclarations des différents éléments de base, il importe de préciser les conditions initiales du fonctionnement en appliquant les règles suivantes :

< *Indication des conditions initiales* > ::= ICE < *Conditions initiales sur les variables d'entrée* >
< *Conditions initiales sur les éléments commandes* >
< *Conditions initiales sur les variables d'entrée* > ::= < *Initialisation* >
< *Liste d'Initialisations* >
< *Initialisation* > ::= < *Indicateur* > < *Nom* > | < *Nom* >
< *Indicateur* > ::= * | ¬
< *Conditions initiales sur les éléments commandés* > ::=
ICC < *Terme* > < *Liste de termes* >
< *Terme* > ::= < *Commande modifiante* > | < *Nom* > = < *Nombre* >-

Passons maintenant à la description des enchaînements.

Nous avons vu au chapitre III que les enchaînements seraient décrits à l'aide de phrases commandes et de phrases tests et que ces phrases seraient généralement assemblées de façon à décrire un enchaînement ou une séquence linéaire. Il s'agit donc de donner ici les règles d'assemblage des phrases élémentaires ainsi que la forme de ces dernières.

Nous appellerons phrases simples les phrases décrivant les enchaînements et phrases linéaires celles relatives à la description des séquences linéaires.

Nous aurons les règles suivantes :

< *Description des enchaînements* > ::= DSE < *Phrase* > < *Liste de phrases* >
< *Phrase* > ::= < *Phrase linéaire* > | < *Phrase simple* > |
< *Phrase de repositionnement* >

Il doit être possible de décrire d'une façon simplifiée l'exécution de commandes simultanées ; d'autre part un enchaînement de deux commandes ne nécessite pas obligatoirement l'exécution d'un test d'où la structure de "phrase simple" suivante:

< *Phrase simple* > ::= < *Ordre multiple* > PUIS < *Ordre multiple* > < *Liste de phrases tests* >
< *Ordre multiple* > ::= < *Phrase commande* > < *Indice* > < *Liste d'ordres* >
< *Indice* > ::= < *Nombre entier* > | < *vide* >
< *Ordre* > ::= ET < *Phrase commande* > < *Indice* >

Nous avons cinq types de phrases commandes que nous noterons comme suit :

$\forall s_i \in S$, ENCL s_i signifiera forcer s_i à 1
DECL s_i " " s_i à 0
EMET s_i " que s_i correspond à une impulsion

$\forall p_i$ processus secondaire
EXEC p_i signifiera exécuter le processus p_i

$\forall m_i \in R$, $\forall 0_j$ opération interne (1)
$m_i = 0_j$ signifiera que le contenu de m_i est le résultat de 0_j

$\forall w_j$ expression booléenne (1)
TEST w_j signifiera exécuter le test sur w_j.

La syntaxe des phrases commandes est donc :

< *Phrase commande* > ::= < *Commande* > < *Nom* > |
< *Nom* > = < *Opération interne* >
< *Commande* > ::= ENCL | DECL | EMET | EXEC | DEBUT | FIN

En ce qui concerne les phrases tests,elles sont de deux types :
- les phrases "test simple" qui commenceront par SI
- les phrases "test d'attente" qui commenceront par DES QUE.

(1) Nous définirons ces termes ultérieurement.

Nous aurons les règles suivantes :

< *Phrase test* > ::= SI < *Expression booléenne* > | < *Phrase test d'attente* >
< *Phrase test d'attente* > ::= DES QUE < *Expression booléenne* >

Les *phrases linéaires* seront structurées comme suit :

< *Phrase linéaire* > ::= < *Ordre multiple* > < *Série d'ordres linéaires* >
< *Ordre linéaire* > ::= PUIS < *Ordre multiple* > < *Liste de phrases test d'attente* >

Les *phrases de repositionnement* seront structurées suivant les règles ci-dessous :

< *Phrase de repositionnement* > ::= < *Ordre multiple* > PUIS
< *Repositionnement multiple* >
< *Repositionnement multiple* > ::= < *Repositionnement* >
< *Liste d'indications de repositionnement* >
< *Repositionnement* > ::= REP < *Entrée* >
< *Entrée* > ::= ¬ < *Nom* > | < *Nom* >
< *Indication de repositionnement* > ::= ET <*Repositionnement* >

Une opération interne est soit une opération booléenne soit une opération arithmétique ou encore un simple rangement de valeur dans un registre. Sa structure sera définie par les règles suivantes :

< *Opération interne* > ::= < *Changement de registre* > | < *Opération booléenne* > |
< *Opération arithmétique* >
< *Chargement de registre* > ::= < *Nombre* > | < *Variable d'entrée* > | < *Registre* >
< *Variable d'entrée* > ::= < *Nom* >
< *Opération booléenne* > ::= < *terme booléen* > < *opérateur booléen* > < *terme booléen* >
< *Opérateur booléen* > ::= ∧ | ∨
< *Terme booléen* > ::= ¬ < *Primaire booléen* > | < *Primaire booléen* >
< *Primaire booléen* > ::= < *Variable d'entrée booléenne* > | < *Registre booléen* >
< *Variable d'entrée booléenne* > ::= < *Nom* >
< *Registre booléen* > ::= < *Nom* >
< *Opération arithmétique* > ::=
< *terme arithmétique* > < *opération arithmétique* > < *terme arithmétique* > |
< *opérateur additif* > < *terme arithmétique* >
< *Registre* > ::= < *Nom* >

< *Opérateur arithmétique* > ::= < *Opérateur additif* > | < *Opérateur multiplicatif* >
< *Opérateur additif* > ::= + | -
< *Opérateur multiplicatif* > ::= × | /
< *Terme arithmétique* > ::= < *Nombre* > | < *Variable d'entrée rationnelle* > |
< *Mémoire rationnelle* >
< *Variable d'entrée rationnelle* > ::= < *Nom* >
< *Mémoire rationnelle* > ::= < *Nom* >

Une expression booléenne a la structure suivante :

< *Expression booléenne* > ::= < *Primaire booléen* > |
< *Primaire booléen* > = < *Primaire booléen* > |
< *Expression arithmétique* >
< *Expression arithmétique* > ::=
< *Terme arithmétique* > < *comparateur* > < *terme arithmétique* >
< *Comparateur* > ::= = | > | < | ≠

Il ne reste plus qu'à définir les règles de production relatives aux *indications complémentaires*. Nous avons vu au chapitre III qu'il y avait quatre types d'indication complémentaire. L'ordre dans lequel elles seront données est sans importance ce qui équivaut à écrire :

< *Indication complémentaire* > ::=
< *Indication des groupes de commandes incompatibles* > |
< *Déclarations des variables de sécurité* > |
< *Indication de processus d'urgence* > |
< *Indication de commandes manuelles* >

Les *groupes de commandes incompatibles* sont classés en deux catégories que nous repèrerons par

STOP pour les arrêts de déroulement de processus et par
MOD pour les modifications de conception.

< *Indication des groupes de commandes incompatibles* > ::=
ICI < *Groupe de commandes incompatibles* > < *Liste de groupes de commandes incompatibles* >

< *Groupe de commandes incompatibles* > ::=
< *Série de phrases commandes* > < *Type* I >
< *Type* I > ::= STOP | MOD

Les déclarations de *variables de sécurité* ont la même structure que celle d'une variable d'entrée ou de sortie, il suffit donc de définir :
< *Indicateur de variable de sécurité* > ::= DVS
< *Déclaration d'une variable de sécurité* > ::=
< *Commentaires* > < *Nature* > < *Type* P >
< *Nature* > ::= < *Nom* > | < *Suite de phrases tests* >
< *Type* P > ::= < *Test ponctuel* > | < *vide* >
< *Test ponctuel* > ::= < *Point testé* > < *Type* P >
< *Point testé* > ::= < *Phrase commande* > < *Indice* > < *Suite de phrases tests* >

Dans le cas où une variable de sécurité a déjà été définie pour un autre processus, il importe de déclarer cette variable comme suit :

< *Déclaration d'une variable de sécurité déjà définie* > ::=
R < *Nom* > < *Nature* > < *Type* P >

La description des *processus d'urgence* sera précédée des déclarations de la variable d'intervention et du processus d'urgence. D'où les règles :

< *Indication de processus d'urgence* > ::=
IPU < *Indication d'un processus d'urgence* > < *Liste d'indications d'un processus d'urgence* >
< *Indication d'un processus d'urgence* > ::=
< *déclaration générale d'une variable d'intervention* >
< *déclaration générale d'un processus d'urgence* >
< *Déclaration générale d'une variable d'intervention* > ::=
< *Déclaration d'une variable d'intervention déjà définie* > |
< *Déclaration d'une variable d'intervention* >
< *Déclaration d'une variable d'intervention déjà définie* > ::=
R < *Nom* > < *Nature* >
< *Déclaration d'une variable d'intervention* > ::=
< *Commentaires* > < *Nature* >
< *Déclaration d'un processus d'urgence* > ::=
< *Commentaires* > < *Nom* > < *Description de base* >

La variable d'intervention peut avoir déjà été définie pour un processus décrit précédemment et déclencher dans ce processus un processus d'urgence donné, alors que dans le processus étudié la même variable d'intervention déclenche un autre processus d'urgence. La valeur de la variable d'urgence est testée systématiquement au long du déroulement du processus étudié.

Pour ce qui est de la définition des *commandes manuelles* nous aurons la structure suivante :

< *Indication des commandes manuelles* > ::=
ICM < *Indication d'une commande manuelle* >
< *Liste d'indications d'une commande manuelle* >
< *Indication d'une commande manuelle* > ::= VAM | < *Nom* > < *Mention* >
< *Mention* > ::= RAZ | MCM

VAM : indique la présence d'une commande d'arrêt manuelle.

La mention concerne uniquement les mémoires

RAZ : pour les remises à zéro de mémoire

MCM : pour les mémoires dont le contenu est susceptible d'être modifié manuellement.

.3 - DESCRIPTION DU FONCTIONNEMENT GLOBAL

La description des enchaînements de processus est structurée de la même façon que la description des enchaînements d'un processus irréductible hormis le fait que l'on ne peut avoir des phrases du type

EMET X et M = Wj.

Les seuls éléments commandés seront les processus irréductibles et éventuellement les processus d'urgence communs à plusieurs processus irréductibles.

ENCL P- indiquera le lancement d'un processus, alors que

EXEC P- indiquera l'exécution complète d'un processus (cf. chapitre I).

Les processus incompatibles se définiront de la même façon que les commandes incompatibles d'un simple processus :

< *Indication des processus incompatibles* > ::=
IPI < *Groupe de processus incompatibles* >
< *Liste de groupes de processus incompatibles* >
< *Groupe de processus incompatibles* > ::=
< *Série de phrases commandes* > < *Type* I >

Il n'est pas possible cependant d'utiliser ici des phrases commençant par EMET.

Les variables d'entrée dont la valeur est modifiée par l'exécution d'un processus seront indiquées de la façon suivante :

< *Indication des variables d'entrée modifiées* > ::=
IVM < *Groupe de commandes modifiant une variable d'entrée* >
< *Liste de groupes de commandes modifiant une variable d'entrée* >
< *Groupe de commandes modifiant une variable d'entrée* > ::=
< *Nom* > (< *Suite de commandes modifiantes* >)

Enfin la description des verrouillages utilisera les règles suivantes :

< *Indication des verrouillages* > ::=
IVR < *Verrouillage* > < *Liste de verrouillages* >
< *Verrouillage* > ::= < *Nom* > < *Groupe de Commandes verrouillées* > < *liste de groupes de commandes verrouillées* >
< *Groupe de commandes verrouillées* > ::=
< *Nom* > (< *Suite de Commandes modifiantes* >)

Un verrouillage commence par le nom du processus en cours de fonctionnement.

Un groupe de commandes verrouillées est précédé par le nom du processus concerné par le verrouillage.

.4 - COMPLEMENTS ET EXEMPLES

Nous complèterons la syntaxe précédente par les règles :

< *Nom* > ::= < *Lettre* > < *Liste de caractères* > ;
< *Lettre* > ::= A | B | ... | W
< *Caractère* > ::= < *Lettre* > | < *Chiffre* > | *espace*
< *Commentaires* > ::= / < *Nom* > | < *vide* >

Un nom commence toujours par une lettre. On peut utiliser toutes les lettres de l'alphabet sauf X, Y et Z réservés à la génération de variables par l'ordinateur lors du traitement de la description. Un nom se termine toujours par un tiret de façon à avoir une séparation nette entre les termes d'une description.

Il y aura trois sortes de nombres : les nombres entiers, décimaux et les puissances de dix.

< *Nombre* > ::= < *Nombre entier* > | < *Nombre décimal* > | < *Puissance de dix* >
< *Nombre entier* > ::= < *Suite de chiffres* >
< *Chiffre* > ::= 0 | 1 | ... | 9
< *Nombre décimal* > ::= < *Nombre entier* >, < *Nombre entier* >
< *Puissance de dix* > ::= < *Nombre entier* >• 10 < *Opérateur additif* > < *Nombre entier* >
< *Code* > ::= < *Signe* > < *Liste de signe* > ;
< *Signe* > ::= < *Lettre* > | < *Chiffre* > | /

Exemples :

Définition d'un processus

DG /Contrôle du fonctionnement d'un bac à électrolyse ; CEB ; 12/3 ; AS

Définition et description des sous-processus autonomes

DPA DG /Convoyeur amont de plaques ; CAP ; 22/01/5M ;
DG /Convoyeur aval de plaques ; CVP ; 22/02/5M ;
DG /Plateau de relevage ; PR ; 64/17/K8 ;

DES PR /Fonctionnement en stockeur ; ST ;
/Fonctionnement en pourvoyeur ; PR ;

Description d'un processus irréductible

```
DEP   ST ;
```

Description de base

Déclaration d'un élément commandé

```
DCC  /Relevage table stockage ; RTS ; SE OC Table ; TA ; 12 ;
      R  ABS ; PR ;
     /Registre de comptage plaque ; RCP ; RR 10 ;
     /Butée d'arrêt plaque ; BAP ; DE [B1;B2;]
```

Déclaration d'une variable d'entrée

```
DCE  /Plaque en butée ; PB ; CF CM [V1 ; V2 ;]
     /Sélection de longueur de plaque ; SL ; CD PU
     /Mesure exacte de la longueur des plaques ; MP ; PA EX [0 A 100]
     /Indicateur de remplissage table stockage ; IR ; CF     [IR1; IR2;]
     /Signal de temporisation de ralentissement plaque ; TP ; VC SC TEMP;
```

Déclaration d'un processus secondaire

```
DEP  /Sous-programme de piquage ; SPP ; PC  ;

                    description de base

                    (VA ; VB ; C1 ; )
```

Déclaration d'un processus type

```
DET  /Mesure de longueur billette ; MLB ; M1 (L;)
     /Minuterie de commande
          remplissage cuve          ; MIN ; T2* (n;(5 A 10) ;(50 A GO)) (M1 )
```

Indication des conditions initiales

```
IC E   ¬A;  B; *C;
IC C   ¬W;  R;  M;= 0
```

Description des enchaînements

DES

DEBUT PUIS ENCL V; DES QUE C;
ENCL V; PUIS EXEC W;1 PUIS EMET IV; DES QUE H;
DECL R; 2 PUIS TEST A;2 SI A; DES QUE B; SI A; SI B;
DECL H1; ET DECL H2; PUIS EXEC C; DES QUE RW;
ENCL V; PUIS REP A;

Indications complémentaires

Indications de commandes incompatibles

ICI DECL V; EXCE R; STOP;

Déclaration d'une variable de sécurité

DVS /Absence de chariot ; AC ; DECL W;1
R AG ; SP3 ; DECL R; SI A;

Indication d'un processus d'urgence

IPV /Cuve d'électrolyse enflammée ; CE ;/Programme d'intervention ; PI ;

description de base
du programme d'intervention

Indication des commandes manuelles

ICM VAM V1; V2; M; RAZ

Chapitre V

INTERPRETATION D'UNE DESCRIPTION EN DESPROG

L'intérêt d'une description systématique d'un processus est d'automatiser les contrôles et la conception de ce processus, c'est-à-dire qu'il s'agit de retrouver rapidement et simplement, à partir d'une description en DESPROG les éléments définissant un processus tels qu'ils ont été présentés au chapitre I. Il sera alors possible de construire le graphe d'état, donnée de base du travail de réalisation.

5.1 - RELATION ENTRE DESCRIPTION ET ORGANIGRAMMES

L'organigramme de commande est la formulation graphique des enchaînements qui décrivent le déroulement du processus. Il s'agit donc de retrouver ces enchaînements à partir des phrases décrivant en DESPROG le fonctionnement du processus.

Dans le cas où il n'existe pas de test ni de rebouclage il n'y a pas de problème, la phrase de description en DESPROG sera du type

< *Phrase commande* > < *indice* > PUIS < *Phrase commande* > < *indice* >

A chaque couple (*Phrase commande*) (*indice*) correspond un élément de **M** : ici $m_i = (c_h, n_i), c'_h \in C_0$, $m_j = (c_k, n_j)$, $c_k \in C_0$ tels que

$$(m_i, m_j) \in R$$

Lorsque l'enchaînement entre deux phrases commandes est fonction du résultat d'un certain nombre de test, deux cas peuvent se présenter :

a) tous les tests sont des tests d'attente. La phrase de description en DESPROG sera du type

< *Phrase commande* > < *indice* > PUIS < *Phrase commande* > < *indice* > < *Suite de phrases test d'attente* >

A chaque phrase "test d'attente" correspond une commande $m_j \in M$, $m_j = (c_k, n_k)$, c_k e TB telle que

$$\exists\, x \in [P,N] \text{ avec } (m_j, x, m_j) \in R.$$

Si on appelle m_i et m_k les commandes correspondant aux phrases commandes et $m_{j_1}, m_{j_2}, \ldots, m_{j_p}$ les commandes correspondant aux phrases "test d'attente" on pourra reconstituer les enchaînements suivants :

$$\exists\, x_1, x_2, \ldots, x_p \in [P,N]$$

$$(m_i, m_{j_1})$$

$$(m_{j_1}, x_1, m_{j_1})$$

$$(m_{j_1}, \bar{x}_1, m_{j_2})$$

$$- - - - - -$$

$$(m_{j_p}, x_p, m_{j_p})$$

$$(m_{j_p}, \bar{x}_p, m_k)$$

en notant $\bar{x}_i = C_{[P,N]}\, x_i$

b) il existe des tests simples et des tests d'attente.

Pour un même ordre multiple il existera plusieurs phrases du type

< *Phrase commande* > < *indice* > PUIS < *Phrase commande* > < *indice* > < *Suite de phrases tests*

S'il n'existe pas de tests d'attente, dans chacune de ces phrases, la suite de phrases tests doit avoir une structure analogue c'est-à-dire si $c_1, c_2, \ldots, c_n$ sont les expressions booléennes testées successivement pour une phrase, pour les autres phrases les expressions booléennes testées seront du type

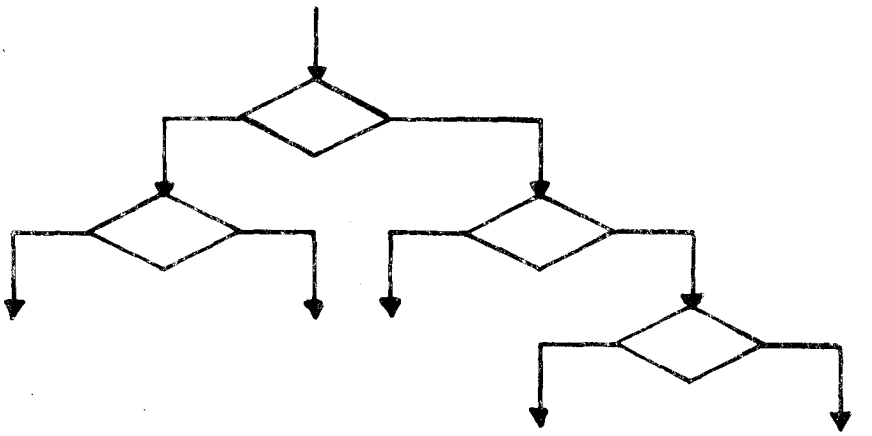
$$c'_1, c'_2 \ldots c'_p$$

tel que $\quad c'_1\, c'_2 \ldots c'_p = \alpha_1, \alpha_2$

et $\quad c_1\, c_2 \ldots c_n = \alpha_1, \beta_2$

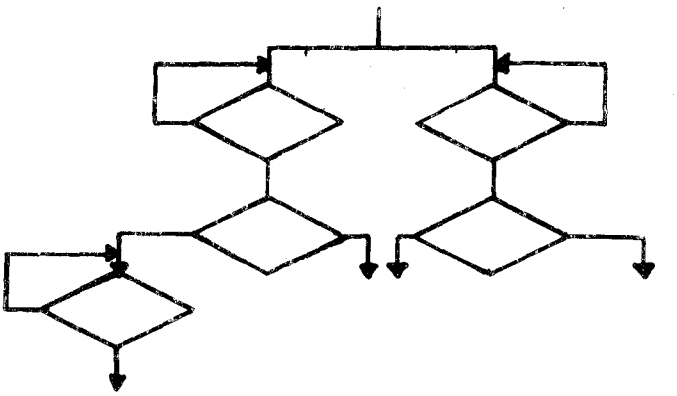
α_1, α_2, β_2 étant des sous-suites de phrases tests.

Ce qui correspond à un schéma du type suivant



S'il existe des phrases tests d'attente intercalées entre des phrases tests simples, les possibilités de structuration des phrases tests sont nettement plus variées puisque l'on peut avoir des exécutions simultanées de tests d'attente et de tests simples.

Exemple



Dans tous les cas, il est possible de rétablir à partir de la description en DESPROG la suite des enchaînements élémentaires ; mais si, dans le cas de suites de phrases "test simple" il est de plus possible de détecter des erreurs dans l'ordre de description des tests, cette possibilité n'existe plus dans le cas où des tests simples et d'attente sont utilisés.

S'il existe *plusieurs commandes simultanées* c'est-à-dire une phrase du type ci-après :

< *phrase commande* > < *indice* > PUIS < *phrase commande* > < *indice* >
ET < *phrase commande* > < *indice* >
.....
ET < *phrase commande* > < *indice* > < *liste de phrases tests* >

Alors, si m_i correspond à la phrase commande initiale
$m_{k_1} \ldots m_{k_r}$ correspondent aux phrases commandes finales
et $m_{j_1} \ldots m_{j_p}$ correspondent aux phrases tests,
nous aurons les enchaînements suivants :

$$(m_i, m_{j_1}),\ (m_{j_1}, x_1, m_{j_2}),\ \ldots\ (m_{j_p}, x_p, m_{k_1})$$
$$\text{et}\quad (m_{j_p}, x_p, m_{k_2})$$
$$\vdots$$
$$(m_{j_p}, x_p, m_{k_r}) \in R$$

avec $x_1, x_2 \ldots x_p \in [P,N]$.

D'une façon analogue dans le cas d'une phrase du type

< *phrase commande* > < *indice* >
ET < *phrase commande* > < *indice* >
.....
ET < *phrase commande* > < *indice* > PUIS < *phrase commande* > < *indice* > < *liste de phrases tests* >

nous aurons les enchaînements suivants :

$$(m_{i_1}, m_{j_1}),\ (m_{j_1}, x_1, m_{j_2}),\ \ldots\ (m_{j_p}, x_p, m_k)$$
$$(m_{i_2}, m_{j_1})$$
$$\vdots \quad \vdots$$
$$(m_{i_r}, m_{j_1})$$

Il est possible d'avoir simultanément les deux cas étudiés ci-dessus.

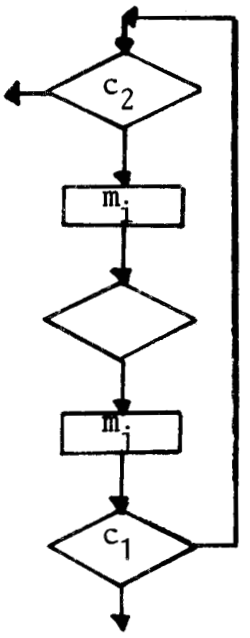
Il est donc aisé de retrouver à partir des phrases linéaires les enchaînements qui constituent la séquence linéaire décrite.

En ce qui concerne les "rebouclages" : trois cas peuvent se présenter

a) le rebouclage se fait sur une phrase commande, auquel cas une phrase simple décrit sans problème le rebouclage ;

b) le rebouclage se fait sur une phrase test mais il existe au moins une phrase commande dans le circuit ainsi constitué

m_j PUIS m_i SIC_1 SIC_2



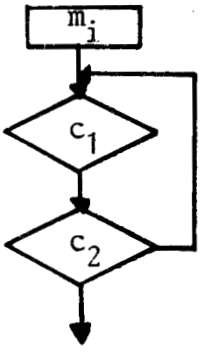
le rebouclage sera décrit par une phrase simple

<phrase commande> *<indice>* PUIS
<phrase commande> *<indice>*
<suite de phrases test >

c) le rebouclage se fait sur une phrase test mais il n'existe pas de phrase commande dans le circuit ainsi constitué. Le rebouclage sera alors décrit par une phrase simple du type

<phrase commande> *<indice>* PUIS TEST *<expression booléenne>* *<suite de phrases test>*

Exemple :



m_i PUIS TEST C1 SI C1 SI C2

L'existence de variables de sécurité entraîne la création de trois enchaînements supplémentaires à chaque point de test : si nous appellons $m_s = (c_s,n_s)$ l'ordre de test de la variable sécurité, nous aurons, au point de fonctionnement caractérisé par $(m_i,f_i,m_j) \in R$

$$(m_i,f_i,m_s),\ (m_s,y,m_j),\ (m_s,x,\text{STOP}) \text{ avec } x,y \in [N,P],\ x \neq y$$

STOP implique la mise à zéro de toutes les variables de sortie qui sont à ce moment à l'état 1.

En ce qui concerne le déclenchement des processus d'urgence, si nous appellons $m_s = (c_s,n_s)$ l'ordre de test de la variable de sécurité, nous aurons, après exécution de chaque ordre m_i d'exécution d'éléments de C_0, quatre enchaînements supplémentaires :

$(m_i,m_s) \in R$

(m_s,x,m_j) pour (m_i,m_j) dans le processus initial

(m_s,y,pu) pu : processus d'urgence

(pu,m_j) $x,y \in (N,P),\ x \neq y.$

Ainsi il est aisé de reconstituer les enchaînements correspondants au déclenchement d'un processus d'urgence à partir d'une description en DESPROG.

De plus, il sera parfois nécessaire d'*introduire de nouvelles variables d'entrée* dans le déroulement d'un processus asynchrone notamment lorsque deux ordres d'exécution d'éléments de C_0 s'enchaînent c'est-à-dire pour $m_i,m_j \in M$: $m_i = (c_k,n_i)$, $m_j = (c_h,n_j)$, c_k, $c_h \in C_0$, $(m_i,m_j) \in R$; d'un point de vue technologique il est nécessaire d'introduire une variable d'entrée de contrôle de fin d'exécution de la commande c_k. Soit m_ℓ l'ordre correspondant au test sur cette variable, il faudra modifier les enchaînements de la façon suivante :

$$(m_i,m_\ell),\ (m_\ell,N,m_\ell),(m_\ell,P,m_j)$$

Tous les enchaînements étant construits, les ensembles M et R sont totalement définis ; il est alors possible de construire la matrice de précédence B correspondant au graphe $G(P) = \{X,\Gamma\}$, X étant en relation biunivoque avec M et Γ avec R.

Après avoir pris en compte la description de l'organigramme de commande en DESPROG, il est possible de construire automatiquement *l'organigramme de processus* et même de le décrire en DESPROG, c'est-à-dire que chaque fois que l'exécution d'un élément de C_0 entraîne la variation de valeur de une ou plusieurs variables d'entrée, une phrase du type

<Phrase commande> *<indice>* PUIS REP *<entrée>* ET ... ET REP *<entrée>*

sera engendrée.

Pour la même phrase commande suivie du même indice, il existera dans la description du processus une phrase du type

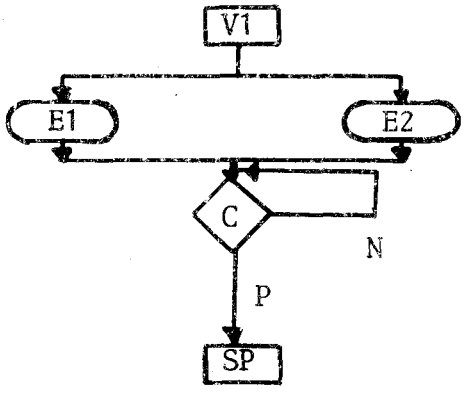
<Phrase commande> *<indice>* PUIS *<ordre multiple>* *<liste de phrases tests>*

Il est alors facile de reconstituer l'organigramme de processus.

Exemple : Si nous avons les phrases suivantes

ENCL V1; PUIS REP E1; ET REP E2;
ENCL V1; PUIS EXEC SP; 3; DES QUE C;

Cela correspond à l'organigramme de commande suivant :



Mais il ne suffit pas de construire les organigrammes de commande et de processus, il faut également contrôler la validité de la description étudiée.

5.2 - POSSIBILITES DE CONTROLE

Nous avons détaillé au chapitre I un certain nombre de propriétés relatives aux processus.

La première des propriétés à vérifier est la connexité du graphe d'état associé au déroulement du processus ; en effet, si ce graphe n'était pas un graphe connexe cela signifierait que le moment d'exécution de certains ordres serait inconnu.

Une autre propriété essentielle, caractéristique des processus cycliques est à vérifier : le fait qu'en fin d'exécution du processus, le bloc de commande doit se retrouver dans les conditions initiales.

Enfin, la troisième propriété importante à vérifier est l'absence de possibilité d'apparition d'ensembles incompatibles durant le déroulement du processus et l'absence d'ambiguïté de fonctionnement.

Nous n'avons cité ci-dessus que les contrôles importants ; il faudrait évidemment reprendre ce qui a été exposé au chapitre I.

Nous allons maintenant étudier plus en détail les moyens de contrôle susceptibles d'être employés dans la vérification des propriétés des processus.

Quatre moyens viennent à l'esprit, ayant chacun des possibilités différentes

- le contrôle de la matrice de précédence B
- le contrôle de l'évolution des vecteurs d'entrée et de sortie au long du déroulement
- la construction du graphe d'état
- l'utilisation de la logique formelle.

De plus, nous verrons au chapitre IX une méthode de contrôle spécifique à un type particulier de réalisation de processus asynchrone.

Les différents contrôles à exécuter sur l'organigramme de commande pourront être réalisés en grande partie grâce au *Contrôle de la matrice de précédence* :

i) *Existence d'un ordre de départ* : Dans la matrice de précédence B associée à un processus correctement conçu doit exister au moins une colonne dont tous les termes sont nuls. L'ordre correspondant à cette colonne sera un ordre de départ c'est-à-dire que si $\exists\ j \in [1,N]$, N×N dimension de B tel que $\forall\ i \in [1,N]$: $b_{ij} = 0$, alors m_j est un ordre de départ.

ii) *Existence d'un ordre "fin"* : la matrice B de précédence associée à un processus correctement conçu doit contenir au moins une ligne dont tous les termes sont nuls. L'ordre correspondant à cette colonne doit être l'ordre "fin" c'est-à-dire si $\exists\ i \in [1,N]$ tel que $\forall\ j \in [1,N]$, $b_{ij} = 0$, alors m_j = fin.

iii) *Contrôle de la connexité du graphe* : Pour que le graphe $G(P) = \{X,\Gamma\}$ associé à un processus P et représenté par la matrice B soit connexe il faut et il suffit qu'il n'existe pas d'ordre sans prédécesseur ni successeur direct, c'est-à-dire que $\nexists\ \alpha \in [1,N]$ tel que $\forall\ i \in [1,N]\ b_{i\alpha} = 0$ et $\forall\ j \in [1,N]\ b_{\alpha j} = 0$.

iv) *Contrôle sur la définition complète du graphe* : Si le processus P a été correctement décrit, cela implique que quel que soit le sommet du graphe G(P) considéré, il existe un chemin dont l'origine est x_0 correspondant à m_0 ordre de départ et l'extrémité x_f correspondant à fin qui passe par ce sommet, c'est-à-dire que $\forall\ m_i \in M\ \exists\ i_1\ i_2 \ldots i_p, i_{p+1}, \ldots, i_q \in \mathbb{N}$ tels que

$$b_{0i_1}, b_{i_1i_2}, \ldots, b_{i_{p-1}i_p}, b_{i_pi}, b_{ii_{p+1}}, b_{i_{p+1}i_{p+2}}, \ldots, b_{i_{q-1}i_q}, b_{i_qi_f}$$

soient différents de zéro (i_f étant l'indice associée à l'ordre fin).

Cependant l'étude de la matrice de précédence ne permet pas de connaître la valeur des différentes variables d'entrée et de sortie à tout moment du déroulement du processus ; or nous avons vu au chapitre I que le contrôle de l'évolution des vecteurs d'entrée et de sortie permet de vérifier qu'un processus est cyclique.

Contrôler l'évolution des vecteurs d'entrée et de sortie suppose que ces vecteurs puissent être construits.

Pour cela, il suffit d'utiliser les indications fournies par les déclarations de variables d'entrée :

- type
- origine
- commandes modifiantes
- ensembles incompatibles de variable d'entrée.

L'étude du type et de l'origine des variables permet de distinguer variables rationnelles et variables booléennes, et dans ces variables booléennes entre les impulsions et les variables non impulsionnelles.

La connaissance des ensembles de variables incompatibles et des commandes modifiantes donne des renseignements sur l'évolution des valeurs des variables booléennes, non impulsionnelles.

Exemple : Soit un processus P tel que :

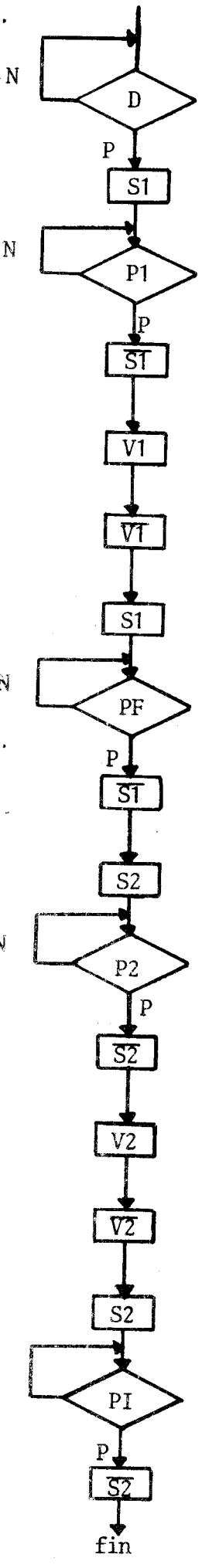
D : variable de départ
PI, P1, P2, PF : contrôles de fonctionnement
S1,S2,V1,V2 : variables de sortie
A(PI) = A(P1) = A(P2) = A(PF) = $\{S_1, S_2\}$
(P1,P2,PI,PF) : ensemble de variables d'entrée incompatibles.

Nous avons ici un déroulement de processus constitué d'une seule séquence linéaire.

A chaque instant du déroulement, nous associerons les valeurs des différentes variables d'entrée ou de sortie que nous pouvons déduire des propriétés sémantiques du processus en fonction de l'ordre exécuté ou en cours d'exécution.

Nous illustrerons cet exemple à l'aide du tableau de la page suivante.

Dans ce tableau à trois colonnes, nous mettrons,en regard de chaque arc ou de chaque sommet de la séquence linéaire qui constitue le processus, les valeurs des variables déduites des propriétés sémantiques du processus ou de l'ordre en cours d'exécution.



$\bar{D}$	PI	$\overline{PI}$	$\overline{P2}$	$\overline{PF}$	$\overline{S1}$	$\overline{S2}$	$\overline{V1}$	$\overline{V2}$	
									Conditions initiales
D									
✱ $\bar{D}$					S1				
	$\overline{PI}$								S1 ∈ A(PI)
	$\overline{PI}$	P1	$\overline{P2}$	$\overline{PF}$					Variables d'entrée incompatibles
					$\overline{S1}$				
							V1		
							$\overline{V1}$		La valeur de PI,P1,P2,PF n'a pu varier car aucune commande modifiante n'a été exécutée
	$\overline{PI}$	P1	$\overline{P2}$	$\overline{PF}$	S1				
		$\overline{P1}$							S1 ∈ A(P1)
	$\overline{PI}$	$\overline{P1}$	$\overline{P2}$	PF					
					$\overline{S1}$				Variables d'entrée incompatible
	$\overline{PI}$	$\overline{P1}$	$\overline{P2}$	PF		S2			La valeur de PI,P1,P2,PF n'a pas pu varier
				$\overline{PF}$					S2 ∈ A(PF)
	$\overline{PI}$	$\overline{P1}$	P2	$\overline{PF}$					Variables d'entrée incompatibles
						$\overline{S2}$			
							V2		
							$\overline{V2}$		La valeur de PI,P1,P2,PF n'a pu varier car aucune commande modifiante n'a été exécutée
	$\overline{PI}$	$\overline{P1}$	P2	$\overline{PF}$		S2			
			$\overline{P2}$						S2 ∈ A(P2)
	PI	$\overline{P1}$	$\overline{P2}$	$\overline{PF}$					Variables d'entrée incompatibles
						$\overline{S2}$			
$\bar{D}$	PI	$\overline{PI}$	$\overline{P2}$	$\overline{PF}$	$\overline{S1}$	$\overline{S2}$	$\overline{V1}$	$\overline{V2}$	Conditions initiales

Remarque 1 : Si une séquence $s_i = \{m_{i_1} \ldots m_{i_k} \ldots \text{fin}\}$ est telle que l'exécution de m_{i_k} fixe la valeur d'un élément de VEB et qu'entre m_{i_k} et fin il n'existe plus de test sur cet élément, quels que soient les ordres exécutés entre m_{i_k} et fin, nous considèrerons qu'en "fin" la variable d'entrée dont la valeur a varié avec l'exécution de m_{i_k} a conservé depuis la même valeur. Si cette valeur ne coïncide pas avec la valeur initiale de la variable considéree, il y a deux cas possibles :

i) aucune commande modifiant la valeur de la variable n'apparaît entre m_{i_k} et fin, il y a une erreur de conception;

ii) il existe au moins une commande modifiant la valeur de la variable considérée entre m_{i_k} et fin, il importe alors d'étudier le rôle de cette commande dans l'évolution de la valeur de la variable étudiée.

Remarque 2 : S'il existe des commandes à exécuter simultanément, il est nécessaire de scinder les vecteurs d'entrée et de sortie en autant de sous-vecteurs qu'il y a de commandes à exécuter.

Le graphe d'état peut être construit à partir de l'organigramme de processus :
Les sommets du graphe d'état sont constitués d'ensembles compatibles qui peuvent être décrits à l'aide des phrases commandes.

Les arcs du graphe d'état sont repérés par des ensembles d'ordres exécutés qui peuvent être décrits à l'aide de phrases commandes pour les exécutions d'éléments de C_0 et de phrases tests pour les résultats de tests.

Néanmoins, à ce niveau, on pourrait concevoir l'utilisation systématique d'expressions booléennes tant pour les exécutions d'éléments de C_0 que pour les tests, il suffirait d'utiliser les équivalences décrites ci-après :

ENCL V	$\equiv$	V
DECL V	$\equiv$	$\bar{V}$
M = X	$\equiv$	M = X
EMET I	$\equiv$	I
EXEC P	$\equiv$	P
TEST W	$\equiv$	W
SI W	$\equiv$	W
SI $\neg$ W	$\equiv$	$\bar{W}$

Les propriétés du graphe d'état ont été étudiées au chapitre I. Sa construction permet à la fois de détecter les ambiguïtés de fonctionnement et l'apparition d'ensembles de commandes incompatibles.

Enfin, il est possible d'utiliser *les propriétés de la logique formelle* pour vérifier la cohérence d'un processus.

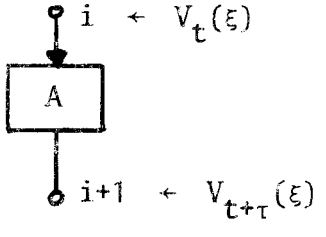
Soit P un processus décrit par un organigramme de commande. Notons ξ l'ensemble des variables d'entrée, de sortie et des contenus de registre et $V_t(\xi)$ l'ensemble des valeurs des éléments de ξ à un instant t ;si l'on exécute un processus secondaire utilisant tout ou partie des éléments de ξ, le processus se retrouvera au temps $t+\tau$, après exécution du processus secondaire, dans un nouvel état caractérisé par $V_{t+\tau}(\xi)$.

En fait, le processus secondaire exécuté aurait pu se réduire à une simple commande. Dans ce qui suit nous utiliserons le terme "tâche" pour indiquer soit un processus secondaire soit une commande exécutée par le processus étudié.

Nous appellerons "point" un instant du déroulement du processus. A tous les arcs de l'organigramme associé correspond un point correspondant lui-même au début ou à la fin d'exécution d'une tâche.

A chaque point est associé un ensemble de valeur des éléments de ξ c'est-à-dire un état du processus noté $V_t(\xi)$. Un changement d'état sera caractérisé par le passage d'un point à un autre point immédiatement consécutif après exécution d'une tâche. Deux points caractérisant un changement d'état sont appelés points adjacents.

Exemple



i et i+1 sont deux points adjacents et A une tâche

L'exécution d'une tâche modifie la valeur d'un ou de plusieurs éléments de ξ. N'importe quelle tâche ne peut être exécutée en n'importe quel état du processus :

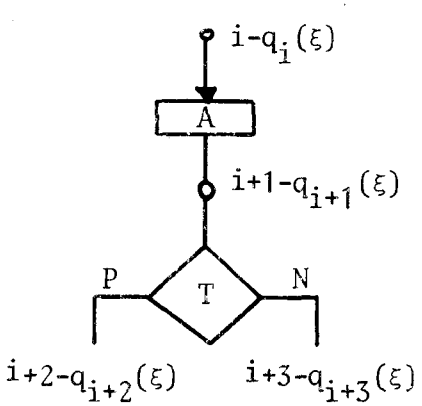
c'est l'état du processus qui détermine la tâche à exécuter ; or nous avons vu que l'état d'un processus était caractérisé par un ensemble de valeurs des éléments de ξ, ce qui peut s'exprimer à l'aide d'un prédicat ; nous noterons $q_i(\xi)$ le prédicat caractérisant l'état d'un processus à un point i de son déroulement.

Une tâche exécutée entre deux points adjacents i et i+1 étant déterminée par $q_i(\xi)$, le passage du point i au point i+1 peut être formalisé par une implication du type $q_i(\xi) \supset q_{i+1}(\xi)$.

Si un test est exécuté sur tout ou partie de l'ensemble des valeurs $V_t(\xi)$ qui caractérise l'état de la machine à l'instant t, un prédicat $p(\xi)$ sera associé à ce test tel que $p(\xi)$ soit vrai si le résultat du test est positif et que $p(\xi)$ soit faux en cas contraire.

Il est alors possible d'établir un ensemble de formules bien formées du calcul des prédicats donnant les expressions générales de chacune de ces relations et essayer d'en déduire formellement les propriétés du processus.

Exemples :



soit $p(\xi)$ le prédicat associé au test T, nous aurons :

$$q_i(\xi) \supset q_{i+1}(\xi)$$

$$q_{i+1}(\xi) \wedge p(\xi) \supset q_{i+2}(\xi)$$

$$q_{i+1}(\xi) \wedge \neg p(\xi) \supset q_{i+3}(\xi)$$

Dans ce qui suit, nous noterons, pour simplifier $V_i(\xi)$ par ζ et $V_{i+1}(\zeta)$ par ζ'.

Nous pouvons étudier différents déroulements du processus étudié en fonction d'un prédicat initial ϕ_1 qui détermine un ensemble d'état initiaux pour lesquels ϕ_1 est vrai. La plupart du temps il y aura un état initial unique.

Chaque état initial détermine une possibilité d'exécution du processus. A tout point i du déroulement est alors associé un prédicat ϕ_i qui indique les valeurs que peuvent prendre les éléments de ξ.

Suivant la cardinalité de l'ensemble des états initiaux, ϕ_i correspondra à un champ de valeurs acceptables plus ou moins restreint.

Suivant la précision des informations apportées par les prédicats du type p_i nous poserons les définitions suivantes :

L'ensemble $\phi_2 \ldots \phi_n$ de prédicats est appelé ensemble de prédicats valides et convergents ou prédicats VC pour les conditions initiales ϕ_1 s'il satisfait à la propriété suivante : Pour l'ensemble des exécutions du processus avec les conditions initiales ϕ_1, ϕ_i est vraie pour tous les états correspondants au point i.

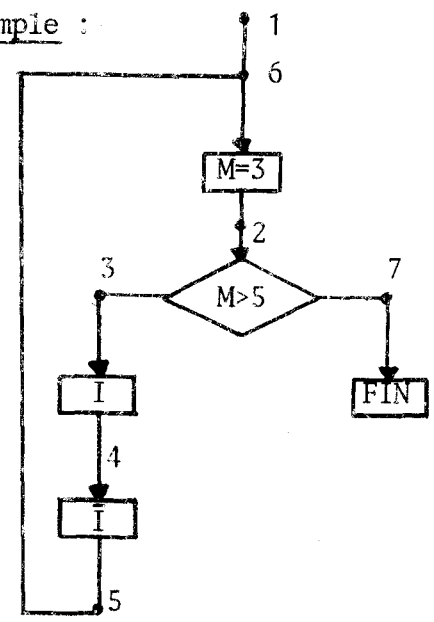
L'ensemble $\phi_2 \ldots \phi_n$ de prédicats est appelé l'ensemble des prédicats minimaux valides et convergents ou prédicats MVC pour les conditions initiales ϕ_1 s'il satisfait à la propriété suivante : Pour l'ensemble des exécutions du processus avec les conditions initiales ϕ_1, $\phi_i(\zeta)$ est vrai si et seulement si le point associé est atteint avec l'état ζ.

Exemple : Si $\phi_i(X) = (0 < x \leq 5)$ est un élément de l'ensemble des prédicats MVC.
$\phi_i'(x) = (0 < x)$ est un élément de l'ensemble des prédicats VC.

On peut alors démontrer le théorème suivant :

Si $q_0(\xi)$ est le prédicat associé au point final du déroulement du processus, et si quand ϕ_1 remplace q_1, q_0 est remplacé par le prédicat FAUX, alors le processus ne peut pas fonctionner.

Exemple :



W1 : $q_1(\zeta) \supset q_6(\xi)$

W2 : $q_6(\zeta) \supset q_2(\zeta)$

W3 : $q_2(\zeta) \wedge p(\zeta) \supset q_0(\xi)$

W4 : $q_2(\zeta) \wedge \neg p(\zeta) \supset q_4(\xi)$

W5 : $q_4(\xi) \supset q_6(\xi)$

Si nous remplaçons $q_1(\xi)$ par $\phi_1 = (M = 0) \wedge (I = 0)$.

Nous avons alors en appliquant les définitions précédentes :

ϕ_1 : $(M = 0) \wedge (I = 0)$

ϕ_2 : $(M = 3) \wedge (I = 0)$

ϕ_3 : FAUX

ϕ_4 : $(M = 3) \wedge (I = 1)$

ϕ_5 : $(M = 3) \wedge (I = 0)$

D'après le théorème énoncé précédemment, le processus ne peut pas fonctionn

Ainsi, l'utilisation des propriétés de la logique formelle permet essentiellement de contrôler la validité des conditions initiales.

5.3 - TRAITEMENT MACHINE

Après avoir précisé les relations entre description et organigrammes de commande et de processus et étudié les divers moyens de contrôler la validité de la description, donnons maintenant les *grandes lignes du traitement machine*.

Chaque description en DESPROG pourra éventuellement être précédée d'un éditeur qui permet à l'utilisateur de modifier la liste de termes employés dans une description classique.

Le traitement machine se décompose en trois étapes distinctes :

1) Prise en compte et vérification de la validité de la description

2) Construction de l'organigramme de processus (ce qui suppose que les processus types soient développés)

3) Elaboration du graphe d'état.

En ce qui concerne la prise en compte et la vérification de la validité de la description, tous les éléments déclarés, leur rôle, leur type, leur origine et d'une façon générale toute indication relative à ces éléments seront mémorisés après

que des vérifications aient établi la validité de ces déclarations ; une même procédure sera appliquée aux indications complémentaires. En ce qui concerne la description du déroulement, tous les termes apparaissant dans celle-ci devront être soit des termes du langage, soit des éléments déclarés ; une fois ce point établi, la machine exécutera les différents contrôles du point de vue logique et sémantique.

La construction de l'organigramme de processus débutera ensuite. Elle nécessitera l'établissement d'un dialogue entre l'opérateur et la machine. Deux possibilités de dialogue sont envisageables :

- une aide à la mise au point de la description, c'est-à-dire un système qui permette à l'opérateur de réparer les omissions dans les déclarations et les descriptions d'enchaînement, et de corriger les erreurs ;

- une aide au développement de l'organigramme de commande en permettant :

  . d'introduire des précisions technologiques sur les variables d'entrée et de sortie (changement du type de variable de sortie, introduction de nouvelles variables d'entrée)

  . d'introduire, dans l'organigramme, des processus-types supplémentaires ; cette opération peut être effectuée systématiquement par l'ordinateur, ou bien l'opérateur peut déterminer lui-même le type et l'implantation de ces nouveaux processus types

  . de développer des processus types. L'ordinateur pourra choisir dans une bibliothèque créée à cet effet les organigrammes correspondants. Le choix sera fait en fonction des indications de code ou sur ordre de l'opérateur. Enfin, l'opérateur pourra lui-même introduire des descriptions particulières de processus types.

De plus, l'opérateur pourra procéder à un certain nombre de modifications dans les enchaînements (suppression ou insertion de nouveaux enchaînements).

Ensuite l'ordinateur introduira, après les avoir détecter, les variations inactives de variables d'entrée dans l'organigramme de commande, le transformant ainsi en *organigramme de processus*.

Pour *élaborer le graphe d'état*, il faudra alors programmer l'algorithme défini au chapitre II.

EN CONCLUSION

Le langage DESPROG présente deux inconvénients majeurs :

i) son utilisation suppose une analyse détaillée du processus, qui ne peut être l'oeuvre d'un automaticien et qui implique un certain travail de conception ;

ii) DESPROG ne permet pas de définir les contraintes d'environnement (chaleur, humidité...) ni les contraintes cinétiques (fin de course, vitesse) qui interviennent néanmoins dans la conception de l'équipement au niveau de l'utilisation et du développement des processus types, des diverses mesures de sécurité, de la conception des capteurs ainsi que, dans certains cas, des enchaînements d'ordre d'exécution.

Il apparaît donc intéressant de définir un nouveau langage plus particulier que DESPROG et strictement réservé à la description des automatismes. Ce langage sera d'une utilisation plus facile que DESPROG et permettra de décrire les différentes contraintes citées ci-dessus ; par contre son usage sera plus restreint et son utilisation moins intéressante au niveau de l'organigramme de processus.

Remarquons enfin qu'il est possible d'envisager deux types d'emploi de DESPROG :

- un emploi classique avec l'aide d'un manuel de référence,
- un questionnaire rigide à remplir. Dans ce cas, le travail mécanographique sera plus important car il sera nécessaire de réécrire les questionnaires dans une forme acceptable par les ordinateurs. Cette solution offre cependant l'avantage de simplifier l'emploi du langage.

Chapitre VI

PRESENTATION DU LANGAGE DESAUCYK

La démarche qui amène à la définition d'un langage de description des automatismes cycliques est inverse de celle utilisée dans la conception du langage DESPROG ; elle est la réponse à la question suivante :

Qu'est-ce qu'un automatisme séquentiel pour un non-automaticien ?

C'est un équipement qui comporte essentiellement :

- des moyens d'exécution
  - . organes en mouvement (ex : chariot...)
  - . systèmes bistables (ex : électrode, vanne, ...)
  - . systèmes de mesures
- des éléments testables
  - . position d'organes en mouvement
  - . état d'un système bistable
  - . grandeur mesurée
- éventuellement des organes de calcul et d'impression de données.

Les moyens d'exécution sont commandés automatiquement par le bloc de commande et les différentes commandes s'enchaînent suivant la valeur des éléments testables.

En outre, il est nécessaire de définir diverses mesures de sécurité, d'urgence, etc...

Enfin, à ce stade élémentaire de la conception, il faut connaître l'environnement (température, vibration...) et la configuration géographique de l'équipement ; dans ces conditions, les moyens d'exécution seront définis non seulement par leur type mais également :

- par leur masse, leur inertie etc...,

- par les temps d'exécution,

- la précision de l'exécution (sur la vitesse, sur les positionnements, sur la mesure...).

Ainsi, alors que DESPROG permettait uniquement la description du bloc de commande, il s'agit maintenant de construire un langage autorisant des études plus importantes puisqu'il sera possible de définir la configuration géographique du système et les contraintes externes ; néanmoins cette extension du domaine décrit n'a d'autre but que de pousser plus loin l'automatisation de la conception du bloc de commande.

La suite du chapitre sera consacré à la présentation de ce langage que nous appellerons DESAUCYK. Etant donné le type d'utilisation prévu pour ce langage, DESAUCYK devra présenter les trois caractéristiques suivantes :

- simplicité d'utilisation

- forme d'expression se rapprochant le plus possible du langage courant

- nombre restreint de déclarations préalables.

Néanmoins, les descriptions obtenues devront être suffisamment détaillées.

6.1 - ELEMENTS A DECRIRE

Avant toute description, il importe de situer l'équipement

- dans le *secteur d'activité* où il sera utilisé (métallurgie, chimie, textile)

- par le *genre de fonction* qu'il remplit (mesure, manutention, levage...).

Le secteur d'activité donne une première indication sur l'environnement probable et les types de contraintes de l'équipement. Ainsi, par exemple en métallurg des problèmes de chaleur et de poussière vont certainement se poser, alors que dans u centrale à béton les difficultés essentielles seront dues au fait qu'il faudra prévoi une robustesse particulière des organes de commande confiés à des utilisateurs non qualifiés.

Le genre de fonction permet de déterminer un ensemble de fonctions-types qui apparaissent généralement dans le fonctionnement des équipements du genre considéré ; à ces fonctions-types pourront correspondre en machine des processus-types stockés dans une bibliothèque et utilisables au moment de la construction de l'organigramme de processus ou même de commande (cf. chapitre V).

Une fois l'équipement situé dans son contexte industriel, il faut passer à la définition des *paramètres d'environnement*. Celle-ci est très arbitraire ; il en résulte que, dans la pratique, les paramètres choisis varieront suivant les utilisateurs; néanmoins, il est possible de dresser une liste non exhaustive des principaux facteurs susceptibles d'influer sur la conception d'un équipement :

- poussière
- chaleur
- rayonnement
- acidité
- vibration
- frottement
- isolement électrique...

Le travail se poursuivra ensuite par la description *des moyens d'exécution, organes de calcul et d'impression de données*. La description d'un moyen d'exécution se décompose en deux parties :

- la description brute de la nature et du rôle de l'élément considéré
- la définition des contraintes imposées au fonctionnement de l'élément. Ces contraintes se classent en trois catégories :

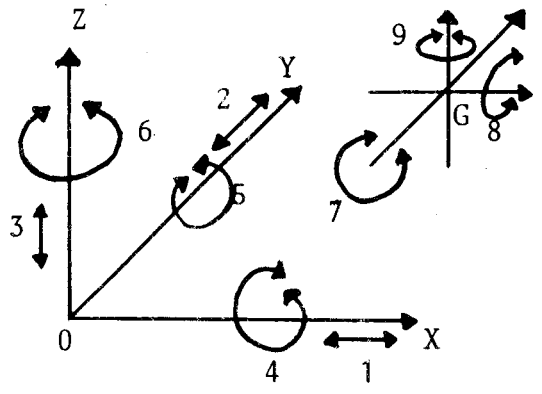
  . les ordres de grandeur (masse, tension, pression)
  . la précision (sur les vitesses, les mesures...)
  . les valeurs maximales (des accélérations, des temps d'exécution...).

En ce qui concerne les *organes en mouvement*, on repère habituellement les mouvements par rapport à un trièdre de référence :

- mouvements de translation suivant les axes du trièdre
- mouvements de rotation autour du centre de gravité du mobile considéré.

Néanmoins pour des raisons d'ordre technologique, nous décomposerons les mouvements en neuf mouvements élémentaires :

- trois mouvements de translations suivant les axes du trièdre de référence
- trois mouvements de rotation par rapport aux axes du trièdre de référence
- trois mouvements de rotation par rapport aux parallèles aux axes du trièdre de référence passant par le centre de gravité du mobile considéré



Nous attribuerons un trièdre de référence à l'ensemble de l'équipement étudié ; un mouvement suivant un axe fixe se définira alors par :

- l'axe du déplacement,
- le sens du déplacement,
- l'origine et la fin de course du déplacement
- éventuellement des positions intermédiaires particulières entre l'origine et la fin de course.

Un mouvement de rotation autour d'un axe quelconque se définira par :

- l'axe rotation,
- le sens de rotation,
- l'origine et la fin de course angulaire de la rotation si elles existent,
- la distance axe-centre de gravité du mobile,
- éventuellement des positions angulaires particulières entre l'origine et la fin de course de la rotation.

Dans le cas où l'axe de déplacement ou de rotation du mouvement considéré est mobile, cela suppose un élément support de l'axe dont le mouvement se détermine

par rapport à un repère fixe ; si ce n'est pas le cas l'élément support se meut lui-même par rapport à un autre élément support, et ainsi de suite... Mais il existera finalement un élément support d'axe se déplaçant lui-même dans un repère fixe.

Dans certains cas, plusieurs organes peuvent se mouvoir par rapport à un même axe de translation ou de rotation (sauf si cet axe de rotation passe par le centre de gravité de l'organe en mouvement).

Un *système bistable* est un organe comportant deux états stables.

Ex : ouverture et fermeture de vanne, mise sous tension d'électrode.

Dans le cas où un organe mobile de l'équipement présente 2^n état stables, il est nécessaire de décomposer cet organe en n éléments bistables et un organe en mouvement support des éléments bistables. Un certain nombre d'organes en mouvement peuvent être considérés comme des éléments bistables (rouleaux de convoyeurs, malaxeurs...) à un état "repos" (arrêt du mouvement) et un état "travail" (organe en mouvement).

Enfin, sur un équipement, existent la plupart du temps des capteurs et autres systèmes qui mesurent des tensions des courants, des longueurs... La quantité mesurée est stockée dans un registre pour être soit testée directement afin de contrôler la suite du fonctionnement, soit utilisée dans des calculs, soit enfin imprimée. Le temps est également mesuré par l'intermédiaire des horloges, minuteries et temporisations ; en général une durée est immédiatement testée.

Nous assimilerons aux moyens d'exécution des *organes de calcul et d'impression de données* : en effet à partir des résultats des mesures, on peut être amené à faire un certain nombre de calculs simples (arithmétique), dont les résultats sont destinés soit à être imprimés, soit à être testés afin de contrôler la suite du fonctionnement.

Un certain nombre de *contraintes* s'exerce sur ces moyens d'exécution ; ces contraintes s'expriment à l'aide de nombres rationnels. Ce sont, par exemple des :

- inerties
- masses
- vitesses de déplacement
- durées de temporisation
- ordres de grandeur, de mesure
- temps de réponse maximums à une commande en vitesse ou en accélération
- temps de commutation d'un élément bistable.

Dans certains cas, peuvent en outre intervenir des précisions sur les vitesses de déplacement.

Enfin, l'évolution du fonctionnement d'un équipement dépend de la valeur des *éléments testables*. Il est possible de tester :

- la position d'un organe en mouvement
- l'état d'un système bistable
- des signaux de cadencement (ordre de départ, signaux de sécurité, temporisation etc...)
- des paramètres mesurés, calculés ou affichés par l'opérateur.

Les *contraintes* qui s'exercent sur les éléments testables sont des contrainte de précision de positionnement ou précision de mesure.

L'étape suivante est la description du fonctionnement. Elle doit permettre la réalisation automatique de l'organigramme de commande du processus correspondant au fonctionnement de l'équipement.

L'enchaînement des commandes doit être décrit d'une façon précise et néanmoins souple c'est-à-dire que les contraintes syntaxiques doivent être réduites au minimum. Le texte de la description se rapprochera le plus possible du langage habituel. Les seuls éléments qui pourront apparaître dans le langage seront soit des moyens d'exécution avec leurs différentes caractéristiques, soit des éléments testables.

Comme dans le cas de la description d'un processus en DESPROG, il est nécessaire de préciser des *indications supplémentaires* qui porteront sur :

- les ensembles de commandes incompatibles,
- les variables de sécurité,
- les processus d'urgence,
- les commandes manuelles.

Remarquons que, à la différence de DESPROG, DESAUCYK permet de décrire l'implantation géographique des divers organes en mouvement ; à partir de celle-ci, il est possible de déterminer les ensembles de variables d'entrée incompatibles, les ensembles de commandes modifiantes ainsi qu'un certain nombre d'ensembles de commandes incompatibles.

Connaissant les éléments à décrire nous pouvons maintenant définir les possibilités du langage DESAUCYK.

.2 - POSSIBILITES DU LANGAGE

Pour tous les *paramètres d'environnement* considérés, l'utilisateur donnera une note indiquant l'acuité du problème pour l'équipement à décrire et chaque paramètre sera désigné par un numéro de code.

En ce qui concerne les *moyens d'exécution*, deux catégories de définition sont à distinguer :

- la déclaration de l'élément proprement dite
- la description de son contexte géographique (sauf pour les registres évidemment).

Pour déclarer un *élément commandé* correspondant à un moyen d'exécution, il faudra indiquer :

- la définition de l'élément,
- le nom de l'élément,
- le type.

Il y a cinq types d'élément commandé :

- les éléments mobiles,
- les éléments bistables,
- les éléments bistables mobiles,
- les registres à contenu rationnel,
- les registres à contenu booléen.

Sur certains équipements, on peut avoir un ensemble d'éléments commandés de même type tel que seul l'un de ces éléments puisse être utilisé à la fois après avoir été sélectionné manuellement par l'opérateur.

Aux éléments commandés classiques seront donc ajoutés les éléments commandés sélectionnés pour lesquels le type sera remplacé par la liste des éléments sélectionnables.

Enfin un sixième type d'élément commandé sera admis : celui de *mode fonctionnement secondaire* qui correspondra à la notion de *processus secondaire cyclique*; il ne sera pas possible de déclarer des processus secondaires non cycliques car dans ce cas les problèmes d'initialisation et d'utilisation sont trop compliqués ; par contre, les déclarations de fonctionnements secondaires sélectionnés seront admises. Les déroulements de ces processus secondaires seront décrits après le déroulement du processus principal.

Pour déclarer un *élément testable* il est nécessaire d'indiquer

- la définition de l'élément,
- son nom,
- son type,
- éventuellement la plage de valeurs attribuée à l'élément.

Les registres qui sont des éléments testables n'ont pas à être déclarés comme tels puisqu'ils ont déjà été introduits comme éléments commandés.

Il est inutile de déclarer les positions repérées sur des axes de translation ou autour des axes de rotation puisque ces éléments seront définis dans la description géographique de l'équipement.

Il reste donc à définir trois types d'éléments testables :

- les paramètres rationnels affichés par l'opérateur,
- " " booléens " " "
- les conditions externes de fonctionnement c'est-à-dire celles correspondant à des signaux de cadencement d'origine extérieure au bloc de commande de l'équipement.

En ce qui concerne les *systèmes de mesure* nous distinguerons deux sortes de fonctions :

- les fonctions temps,
- les fonctions mesures.

Il y a trois types de fonctions temps :

- les horloges c'est-à-dire des systèmes qui émettent un signal périodiquement suivant une période τ, généralement fixe, et ce jusqu'à réception d'un signal d'arrêt

- les minuteries c'est-à-dire des systèmes qui, recevant un ordre à un instant t_o, émettent n signaux successifs régulièrement espacés d'un temps τ, le premier signal apparaissant au temps $t_o+\tau$, n et τ étant le plus souvent réglables

- les temporisations c'est-à-dire des systèmes qui recevant un ordre à un instant t_o émettent un signal au temps $t_o+\tau$, τ étant le plus souvent réglable.

Une fonction-temps sera définie par

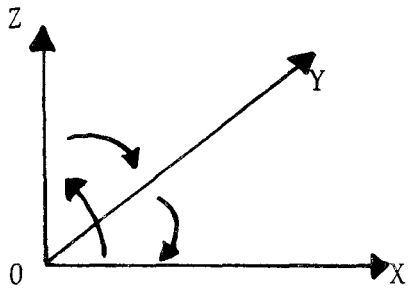
- son nom,
- son type,
- les paramètres intervenant dans son fonctionnement,
- la variable qu'elle émet.

Pour une fonction mesure, il suffit de déclarer le résultat de la mesure comme un élément commandé dont le type est la grandeur mesurée avec sa valeur maximum c'est-à-dire que l'on considère les fonctions mesure comme des registres.

Il reste à donner *une description géométrique* de l'équipement : il faut noter que cette description géométrique est utile même dans le cas d'un équipement

sans élément mobile, ne serait-ce que pour avoir un ordre de grandeur du coût du cablage.

Il est tout d'abord nécessaire de définir un trièdre de référence pour l'ensemble de l'équipement



Dans ce qui suit, les axes d'un trièdre quelconque seront toujours notés respectivement OX, OY, OZ, de manière à pouvoir reconnaître leur position respective.

Par rapport au trièdre de référence de l'équipement nous définirons pour chaque axe fixe de translation :

- le type d'axe (OX, OY, OZ),
- les éléments mobiles se déplaçant le long de cet axe,
- les coordonnées des extrémités de la portion utile de l'axe,
- les abscisses des positions intermédiaires repérées sur cet axe,
- les noms des vitesses sur cet axe dans chaque sens.

Le type d'axe est inutile dans la mesure où,pour le ou les éléments mobiles concernés, l'axe considéré est l'unique axe de déplacement.

Pour chaque axe fixe de rotation, on définira

- le type d'axe (OX, OY, OZ),
- les éléments mobiles tournant autour de cet axe,
- les coordonnées de la portion utile de l'axe,
- la distance axe-centre de gravité du ou des mobiles,
- les positions angulaires repérées autour de cet axe,
- les noms des vitesses angulaires dans chaque sens de rotation.

Si pour un axe de rotation il y a plusieurs éléments mobiles tournant autour de cet axe, mais à des distances différentes, il est nécessaire de redéfinir l'axe pour chaque distance de rotation.

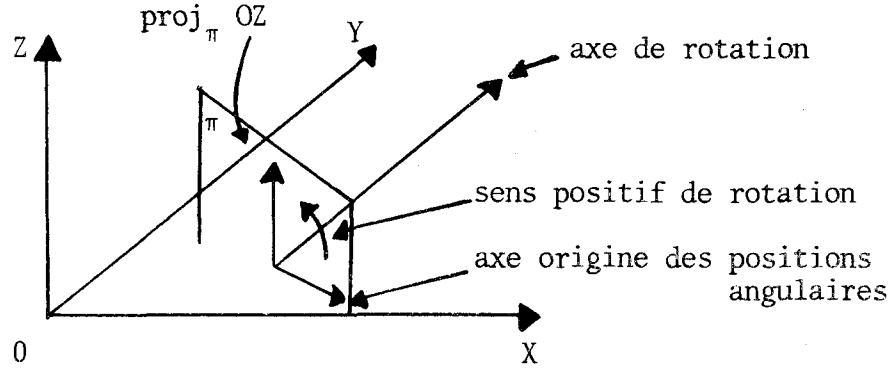
La remarque sur le type d'axe faite dans le cas des axes de translation restent valables pour les axes de rotation. Il reste à déterminer une règle permettant de fixer le sens positif de rotation et l'origine des positions angulaires, nous envisagerons la méthode suivante :

- considérer l'axe de rotation orienté positivement suivant les coordonnées de la position utile de l'axe,

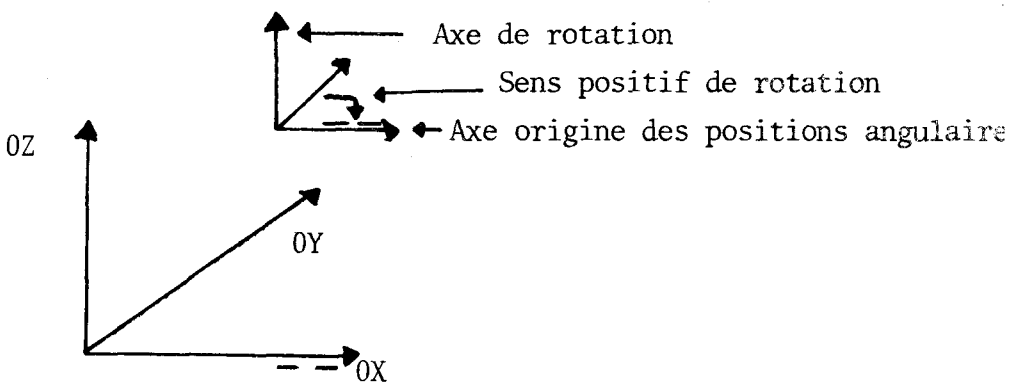
- construire le plan π perpendiculaire à l'axe de rotation,

- projeter dans ce plan π l'axe OZ du trièdre de référence de l'équipement,

- si on prend cette projection comme axe OZ d'un nouveau trièdre dont l'axe OY sera constitué par l'axe de rotation, il est facile de déterminer l'axe OX, origine des positions angulaires, le sens positif de rotation étant alors le sens trigonométrique.



Si l'axe de rotation est parallèle à l'axe OZ de référence, le sens positif de rotation sera celui du trièdre de référence par rapport à OZ ou le sens opposé, suivant l'orientation de l'axe de rotation. L'axe origine des positions angulaires sera parallèle à l'axe OX du trièdre de référence



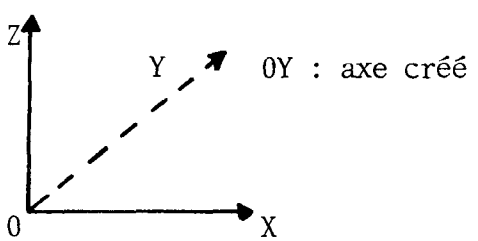
Quand un axe de translation ou de rotation est matérialisé sur un élément mobile ou support, il sera repéré dans le trièdre de référence des mouvements de son support ; celui-ci étant considéré immobilisé à l'origine du trièdre de référence de ses mouvements ou sur sa position angulaire d'origine.

Si ce support présente :

- trois degrés de liberté, il peut se déplacer suivant trois axes qui constituent le trièdre de référence de ses déplacements

- deux degrés de liberté, il ne peut se déplacer que suivant deux axes, il faut alors compléter le trièdre de référence par un troisième axe en fonction des positions respectives des deux axes de déplacement utilisé.

Exemple



- un degré de liberté, il se déplace suivant un axe. Le trièdre de référence doit être défini à l'aide de la méthode présentée pour la détermination de l'axe origine des positionnements angulaires pour un mouvement de rotation.

La déclaration d'un axe de rotation ou de translation sera alors précédée du nom de l'élément support.

Dans le cas des éléments bistables fixes, il faut déterminer la position de leur centre de gravité dans le trièdre de référence de l'équipement. Si un élément bistable est solidaire d'un élément mobile, la position de son centre de gravité sera repérée dans le trièdre de référence des mouvements de son support, celui-ci étant immobilisé à l'origine du trièdre ou sur sa position angulaire origine.

Enfin, lorsqu'on a déclaré un élément mobile sélectionné, il suffit de définir les correspondances entre les positions de l'élément sélectionné et celles des éléments sélectionnables.

Les éléments constitutifs de l'automatisme étant définis, il s'agit de décrire le fonctionnement du système, c'est-à-dire les enchaînements de mouvement et d'ordres divers. Cela nécessite trois sortes de phrases :

- celles décrivant une exécution de commande ou phrase "commande",
- celles décrivant les éléments testables ou phrases états,
- celles décrivant les tests ou phrases tests.

Ces phrases devront pouvoir s'articuler entre elles. Trois types d'articulations sont envisageables :

- les articulations de succession immédiate,
- les indicateurs de simultanéité,
- les articulations de renvoi à d'autres phrases du texte.

Les phrases de commande devront pouvoir décrire :

- des mouvements,
- des changements d'état,
- des calculs et des chargements de registres,
- des exécutions de fonctionnements secondaires ou de fonctions-types c'est-à-dire de processus secondaires ou de processus types.

Les phrases états devront pouvoir décrire :

- des positionnements d'organes mobiles,
- des états de systèmes bistables,
- des résultats de test sur des expressions booléennes.

Les phrases tests seront constituées à l'aide de phrases états. On pourra distinguer quatre variantes de phrases décrivant :

- des tests simples,
- des tests d'attente,
- des tests répétitifs,
- des tests pseudo-répétitifs.

Nous appelons test répétitif une succession de tests qui est systématiquement exécutée tant que le dernier test ne donne pas un résultat déterminé.

Nous appelons test pseudo-répétitif une succession de tests qui est ou non réexécutée suivant le résultat du dernier test.

Chaque phrase commande ou phrase test pourra être précédée d'un titre qui sera utilisé pour définir les enchaînements en cas de rupture du déroulement séquentiel

Chaque description devra commencer et se terminer par un terme spécifique utilisable comme titre.

D'autre part des phrases tests supplémentaires permettront d'indiquer l'existence de plusieurs possibilités pour un même enchaînement de phrases commandes.

Dans le cas où une séquence de commandes est réexécutée systématiquement tant qu'un test n'a pas changé de résultat, il sera possible de décrire cette séquence en allégeant la description des enchaînements.

Lorsqu'une horloge où une minuterie est déclenchée, les signaux émis commandent des processus secondaires qui ne sont pas directement insérés dans le processus global et qu'il convient donc de décrire séparément. La description du fonctionnement de ces processus secondaires cycliques ou acycliques sera du même type que celle du fonctionnement d'un processus irréductible.

Lorsqu'un processus secondaire acyclique apparaît plusieurs fois dans le déroulement d'un processus, il est inutile de le redécrire entièrement à chaque occurence d'utilisation. Si la description du processus secondaire, à sa première occurence d'utilisation, est repérée par un titre en début et un titre en fin de description, dans la suite de la description du processus global étudié, chaque

occurence du processus secondaire sera définie par ces étiquettes. Eventuellement, le même processus secondaire pourra être réutilisé avec d'autres jeux de variables ; il importe alors de définir ces modifications par adjonction d'un complément à la phrase commande.

Enfin chaque processus secondaire cyclique déclaré comme élément commandé doit être décrit comme un processus irréductible.

Il ne reste plus alors qu'à donner des *indications complémentaires*.

Il n'est pas nécessaire de définir les ensembles de commandes incompatibles puisque la description géographique de l'équipement permet de trouver ces ensembles.

Les déclarations des *variables de sécurité* se feront d'une façon similaire à celles du langage DESPROG en indiquant la nature de la variable et la position des tests sur la valeur de cette variable. Lorsque la valeur d'une variable de sécurité n'est pas testée systématiquement, il suffit d'indiquer les points de tests par un titre.

Les déclarations de *processus d'urgence* comportent deux parties :

- la déclaration de la variable aléatoire
- la déclaration du processus d'urgence proprement dit

la valeur de la variable aléatoire est testée systématiquement et la variable aléatoire peut être constituée par une suite de tests.

Enfin il importe d'indiquer quels sont les éléments commandés qui possèdent un jeu de *commandes manuelles*.

3 - METHODE DE DESCRIPTION GLOBALE :

Lorsqu'il s'agit de décrire le fonctionnement d'un automatisme complexe, la méthode de description est la suivante :

- Décomposition de l'équipement en sous-machines autonomes. Une sous-machine autonome est un sous-ensemble de l'équipement qui exécute un sous-processus du processus correspondant à l'automatisme étudié c'est-à-dire que deux sous-machines

distinctes ne peuvent disposer en commun d'un même moyen d'exécution ou d'un même registre.

- Pour chaque sous-machine ou pour l'équipement dans sa totalité s'il n'est pas décomposable, définition des modes de fonctionnement principaux : un mode de fonctionnement principal est l'ensemble des fonctions exécutées par un équipement au cours du déroulement d'un processus irréductible [1].

- Description, pour chaque sous-machine, des modes de fonctionnement principaux.

- Description du fonctionnement global. Chaque mode principal est considéré comme un élément commandé du mode de fonctionnement global. La description du fonctionnement global utilise les mêmes phrases que les descriptions de modes de fonctionnement. De plus il importe de définir :

. les ensembles de modes de fonctionnement incompatibles

. les verrouillages de commandes entraînés par l'exécution de chaque mode de fonctionnement principal. Ces verrouillages seront décrits à l'aide de phrases états.

En outre pour l'équipement dans son ensemble et/ou éventuellement pour chaque sous-machine, la description sera précédée d'une définition de l'environnement.

---

(1) Par analogie, nous appellerons dans la suite mode de fonctionnement secondaire les processus secondaires (cf. § 6.2) et mode de fonctionnement aléatoire les processus d'urgence.

Chapitre VII

DEFINITION DU LANGAGE DESAUCYK

La structure du langage découle de la méthode de description donnée au paragraphe 6.3.

Les notions de description, définition, déclaration et indication utilisées au chapitre IV restent valables dans ce qui suit ainsi que les métarègles définies en tête de ce chapitre IV.

Nous définirons d'abord la structure générale d'une description d'automatisme puis nous étudierons chaque élément de cette description en détail, enfin nous terminerons le chapitre par quelques exemples.

7.1 - STRUCTURE GENERALE

La description d'un automatisme se décompose en trois grandes étapes suivant la règle ci- après :

<Description d'un automatisme> ::=
<Définition d'une machine>
<Définition et description des sous-machines>
<Description du fonctionnement global de la machine>

La définition d'une machine comporte trois ou quatre éléments suivant les cas :

<Définition d'une machine> ::=
DM *<Commentaires>* *<Nom d'une machine>* *<Numéro de secteur d'activité>*
<Indications d'environnement>

<Nom d'une machine> ::= *<Nom>*

Les commentaires facilitent la lecture de la description. Les indications d'environnement permettent de définir dans quelles conditions fonctionne l'équipement.

<Indications d'environnement> ::= *<vide>* |
IEV *<Suite de paramètres d'environnement>*
<Paramètre d'environnement> ::=
<Numéro du paramètre> ; *<Nombre entier>* ;
<Numéro de paramètre> ::= *<Nombre entier>*
<Numéro de secteur d'activité> ::= *<Nombre entier>*

La définition des sous-machines correspond au découpage de la machine ou de l'équipement étudié en sous-machines. Cet élément de description n'apparaît donc pas systématiquement puisque toutes les machines ne sont pas décomposables en sous-machines, d'où les règles suivantes :

<Définition et description des sous-machines> ::=
<Numéro de genre de fonctionnement>
<Définition et description de modes de fonctionnement principaux> |
DSM *<Série de définitions d'une sous-machine>*
<Série de descriptions d'une sous-machine>
<Numéro de genre de fonctionnement> ::= *<Nombre entier>*
<Définition d'une sous-machine> ::=
<Nom> *<Numéro de genre de fonctionnement>*
<Indications d'environnement>

Une sous-machine se définit comme une machine hormis le fait que le numéro de secteur d'activité est remplacé par le numéro de genre de fonctionnement. S'il n'y a pas eu d'indications d'environnement d'associées à la définition de la machine globale, il est alors nécessaire de joindre de telles indications à la définition de chaque sous-machine ; mais inversement, des indications d'environnement supplémentaires peuvent être associées à la définition d'une sous-machine de manière à compléter les indications apparaissant avec la définition de la machine globale.

Chaque sous-machine définie doit être ensuite décrite ; l'ordre de description des sous-machines est sans importance.

La description d'une sous-machine commence normalement par le nom de la sous-machine décrite. Cependant dans le cas où la machine initialement étudiée est indécomposable en sous-machine, il est possible de passer directement à la définition des modes de fonctionnement principaux après avoir défini le numéro du genre de fonctionnement.

Chaque mode de fonctionnement défini doit être décrit ; l'ordre de cette description n'importe pas. Dans le cas où il existe plusieurs modes de fonctionnement pour une machine ou une sous-machine, chaque description d'un mode de fonctionnement doit être précédée du nom du mode décrit.

<Description d'une sous-machine> ::= DES *<Nom d'une machine>*
<Définitions et descriptions des modes de fonctionnement principaux>
<Definitions et descriptions des modes de fonctionnement principaux> ::=
<Description élémentaire> |
<Série de définitions d'un mode de fonctionnement>
<Série de descriptions d'un mode de fonctionnement>
<Définition d'un mode de fonctionnement> ::= *<Commentaires>* *<Nom>*
<Description d'un mode de fonctionnement> ::= DMF*<Nom>* *<Description élémentaire>*

Lorsque la machine initiale est indécomposable en sous-machines et ne présente qu'un seul mode de fonctionnement, la description élémentaire s'enchaîne directement sur la définition de la machine accompagnée du numéro de genre de fonctionnement.

DM *<Commentaires>* *<Nom>* *<Numéro de secteur d'activité>* *<Indications d'environnement>*
<Numéro de genre de fonctionnement> *<Description élémentaire>*

Lorsque la sous-machine étudiée ne présente qu'un seul mode de fonctionnement on a alors une structure simplifiée de la forme

DES *<Nom>* *<Description élémentaire>*

La structure de la description du fonctionnement global de la machine est similaire à celle de la description du fonctionnement global d'un processus en DESPROG et répond aux règles ci-dessous :

<Description du fonctionnement global de la machine> ::=
DEG *<Description des commandes de mode de fonctionnement>*
<Liste d'indications supplémentaires> FDG |
<Liste d'indications supplémentaires> FDG
<Indication supplémentaire> ::=
<Indication des modes de fonctionnement incompatibles> |
<Indication des verrouillages>

L'élément de base d'une description est donc ce que nous avons appelé "description élémentaire".

7.2 - STRUCTURE D'UNE DESCRIPTION ELEMENTAIRE

Nous pouvons déduire du chapitre VI la règle de production suivante :

<Description élémentaire> ::=
<Liste de déclarations d'éléments constitutifs>
<Description du fonctionnement>
<Liste d'indications complémentaires> FDE

Il n'existe pas obligatoirement de déclarations d'éléments constitutifs car pour une machine donnée présentant plusieurs modes de fonctionnement il est possible que tous ces modes utilisent les mêmes éléments ;il est alors inutile de les déclarer systématiquement pour chaque nouveau mode décrit.

D'après le chapitre VI il existe quatre sortes d'éléments constitutifs à déclarer :

<Déclarations d'éléments constitutifs> ::= *<Déclarations d'éléments commandés>* |
<Déclarations d'éléments testables> |
<Déclarations de fonctions temps> |
<Déclarations de positionnements>

Quelle que soit la sorte d'élément déclaré, la structure d'une déclaration est la suivante :

<Déclaration de (a)s> ::=*<Indicateur de (a)>* *<Suite de déclaration d'un (a)>*

Il est inutile de redéclarer un élément déjà déclaré pour un autre mode de fonctionnement. Compte tenu du nombre réduit d'éléments le problème de la recherche des éléments déclarés pour d'autres fonctionnements est moins important que dans le cas de DESPROG.

En ce qui concerne les *éléments commandés* nous aurons :

<Indicateur d'élément commandé> ::= DCC
<Déclaration d'un élément commandé>
<Commentaires> *<Nom>* *<Type C>*

Il y a sept types d'éléments commandés que nous noterons de la manière suivante :

- élément mobile : EM
- élément bistable : EB
- élément bistable mobile : EBM
- mode de fonctionnement secondaire : MFS
- registre booléen : RB
- registre rationnel : RR
- fonction mesure [1].

Lorsqu'un élément est sélectionné parmi un ensemble d'éléments, le type est remplacé par l'ensemble des éléments sélectionnables.

<Type C> ::= EM | EB | EBM | MFS | RB | *<Type M>* *<Nombre>*
| *<Ensemble d'éléments sélectionnables>*
<Type M> ::= RR | *<Mesure>*
<Elément sélectionnable> ::= *<Nom>*

Dans le cas d'un registre rationnel ou d'une fonction mesure, il est nécessaire de préciser la capacité du registre ou la grandeur maximale mesurée.

La déclaration d'un *élément testable* sera structurée suivant les règles suivantes :

<Indicateur d'élément testable> ::= DCT
<Déclaration d'un élément testable> ::=
<Commentaires> *<Nom>* *<Type T>* *<Plagide>*

Il y a trois types d'éléments testables à déclarer :

PR : paramètre rationnel
PB : - booléen
CE : condition de fonctionnement.

---

(1) La liste des types de fonctions mesures doit être définie par l'utilisateur.

Ce que nous exprimerons par la règle suivante :

<Type T> ::= PR | PB | CE

La plage de valeur n'apparaît systématiquement dans le cas des paramètres rationnels.

La structure des plages de valeurs reste celle définie pour le langage DESPROG.

La déclaration des *fonctions-temps* est analogue à celle des processus-types dans DESPROG.

<Indicateur de fonctions-temps> ::= DCF
<Déclaration d'une fonction temps> ::=
<Commentaires> *<Nom>* *<Type F>* *<Vadevide>* *<Vadevide>*

Il y a trois sortes de fonctions-temps que nous noterons :

- T pour temporisation
- M pour minuterie
- H pour horloge

d'où la règle de décomposition de "Type F"

<Type F> ::= T | M | H

"Vadevide" a la même structure que dans DESPROG et permet la définition des paramètres et des variables émises.

Il ne reste plus qu'à définir la façon de décrire les *positionnements*

<Indicateur de positionnement> ::= DCG.

Au chapitre VI, nous avons défini trois sortes de positionnement :

<Déclaration d'un positionnement> ::=
<Déclaration d'un positionnement axial> |
<Déclaration d'un positionnement ponctuel> |
<Déclaration d'un positionnement sélectionné> (1)

(1) Cf. Chapitre VI.

Il y a deux **variantes** de déclaration d'un positionnement axial suivant que l'axe considéré est fixe ou mobile.

<Déclaration d'un positionnement axial>::=
SUR *<Nom>* AXE *<Type A>* DE *<Suite de Noms>* *<Coordonnées>*
<Coordonnées> *<Positions>* |
AXE *<Type A>* DE *<Suite de Noms>* *<Coordonnées>*
<Coordonnées> *<Positions>*

"Type A" permet de repérer l'axe considéré suivant la règle donnée au chapitre VI c'est-à-dire que nous aurons

<Type A> ::= OX | OY | OZ|*<vide>*

L'indication de type est suivie des noms des éléments mobiles sur l'axe.

Les coordonnées des extrêmités de la portion utile de l'axe seront définies par trois nombres :

<Coordonnées> ::= (*<Nombre>* ; *<Nombre>* ; *<Nombre>*)

Le terme "Position" recouvre les définitions de positions repérées sur l'axe et de vitesses sur cet axe.

<Position> ::= *<Rovide>* *<Ensemble des positions repérées>*
<Ensemble des vitesses dans le sens positif>
<Ensemble des vitesses dans le sens négatif>
<Rovide> ::= R *<Nombre>* ; | *<vide>*
<Vitesse dans le sens positif> ::= *<Nombre>* ;
<Vitesse dans le sens négatif> ::= *<Nombre>* ;
<Position repérée> ::= *<Nom>* *<Nombre>* ;

"Rovide" est prévue pour indiquer, dans le cas d'un mouvement de rotation, la distance axe-centre de gravité de l'élément mobile.

Lorsqu'un positionnement est sélectionné, il importe d'indiquer les correspondances entre les éléments déclarés pour le positionnement sélectionné et ceux déclarés pour les positionnements sélectionnables :

<Déclaration d'un positionnement sélectionné> ::=
POUR *<Nom>* *<Ensemble de Noms>* *<Suite de sélections>*

Après avoir indiqué le nom de l'élément sélectionné, il faudra préciser les noms des éléments sélectionnables.

<Sélection> ::= *<Nom>* *<Ensemble de noms>*

A chaque position repérée pour l'élément sélectionné, il s'agit de faire correspondre les positions repérées correspondantes suivant l'ordre indiqué par la suite des noms des éléments sélectionnables. Par exemple, si deux chariots de manutention sont sélectionnables, chacun ayant son axe de déplacement propre, il faudra déclarer un élément sélectionné et les positions d'arrêt de cet élément sélectionné, lesquelles doivent être en relation biunivoque avec les positions d'arrêt des deux chariots sélectionnables.

Enfin, en ce qui concerne les déclarations de positionnement ponctuel, la définition est simplifiée. Il y a cependant deux variantes de déclaration suivant que l'on a un système fixe ou mobile :

<Déclaration d'un positionnement ponctuel> ::=
SUR *<Nom>* *<Nom>* *<Coordonnées>* |
<Nom> *<Coordonnées>*

La description du fonctionnement s'enchaîne ensuite sur ce dernier groupe de déclarations. Elle sera cependant précédée de la définition des conditions initiales

<Description de fonctionnement> ::=
ICI *<Conditions Initiales>* *<Texte>*
<Liste de description d'un mode de fonctionnement> FDB

Les conditions initiales seront décrites à l'aide de phrases-états définies ultérieurement.

<Conditions initiales> ::= *<Série de phrases-états>*

En ce qui concerne le "texte" de la description, nous avons vu au chapitre précédent que trois sortes de phrases seraient utilisées : les phrases-commandes, les phrases-états et les phrases-tests.

Un enchaînement de deux commandes sera décrit à l'aide de deux phrases "commandes" sans liaison ni ponctuation ; de même un enchaînement de deux tests sera décrit à l'aide de deux phrases tests sans liaison ni ponctuation, un enchaînement d'une commande et d'un test sera décrit à l'aide d'une phrase commande immédiatement suivie d'une phrase-état sans liaison ni ponctuation et vice-versa. Par contre, deux commandes exécutées simultanément seront décrites à l'aide de deux phrases commandes reliées par ET et deux phrases-tests pourront être liées par un OU pour décrire une alternative. Chaque description commencera par DEBUT et comportera une seule fois le mot FIN :

<Texte> ::= DEBUT *<Phrase>* *<Liste de phrases étiquetées>*
<Dernière phrase> *<Liste de fonctionnements cadencés>*
<Phrase étiquettée> ::= *<Titre>* *<Phrase>* |*<Phrase>*
<Titre> ::= *<Suite de caractères>* ;
<Phrase> ::= *<Phrase commande multiple>* | *<Phrase test multiple>*
<Dernière phrase> ::= ALLER A *<Titre général>* | FIN
<Titre général> ::= *<Titre>* | DEBUT | FIN

ALLER A *<titre>* indique que la commande ou le test suivant est décrit par la phrase qui suit le titre indiqué.

Nous appellons "fonctionnement cadencé" les processus secondaires déclenchés par un signal émis par une fonction-temps

<Fonctionnement cadencé> ::= FC *<Nom>* *<Phrase>*
<Liste de phrases étiquetées> *<phrase finale>*
<Phrase finale> ::= FIN | ALLER A *<Tifin>*
<Tifin> ::= *<Titre>* | FIN

D'après les remarques ci-dessus nous avons :

<Phrase commande multiple> ::= *<Phrase commande>*
<Liste de phrases commandes liées>
<Phrase commande liée> ::= ET *<Tivide>* *<Phrase Commande>*
<Tivide> ::= *<Titre>* | *<vide>*
<Phrase test multiple> ::= *<Phrase test>* *<Liste de phrases tests liées>*
<Phrase test liée> ::= OU *<Tivide>* *<Phrase test>*

Nous pouvons définir huit types de phrases commande (cf. Chapitre VI) :

<*Phrase commande*> ::= <*Phrase commande de mouvement*> |
<*Phrase commande d'élément bistable ou assimilé*> |
<*Phrase commande de mode secondaire*> |
<*Phrase commande de registre*> | <*Boucle*>
DE <*Titre*> | ALLER A <*Titre*> | FIN

DE <*Titre*> indique que la commande ou le test précédent est celui dont la description précède le titre indiqué.

Les phrases *commandes de mouvement* auront la structure suivante :

<*Phrase commande de mouvement*> ::=
DEMARRER <*Type du mouvement*> <*Précisions*> |
ARRETER <*Type du mouvement*> SUR <*Position d'arrêt*> |
AMENER <*Mouvement*> <*Nom de l'élément mobile*> SUR <*Point fixe*>
<*Type du mouvement*> ::=
<*Indication de mouvement*> <*Nom de l'élément mobile*>
<*Nom de l'élément mobile*> ::= <*Nom*>
<*Position d'arrêt*> ::= <*Nom*>
<*Indication de mouvement*> ::=
DEPLACEMENT AVANT <*Axe*> |
DEPLACEMENT ARRIERE <*Axe*> |
ROTATION <*Axe de rotation*>
<*Axe*> ::= SUIVANT <*Type A*> | AUTOUR DE <*Type A*> | <*Vide*>
<*Axe de rotation*> ::= AUTOUR DE <*Type A*> |
INVERSE AUTOUR DE <*Type A*> | <*Vide*>
<*Précisions*> ::= <*Vivide*> TANT QUE <*expression booléenne*> |
<*Vivide*> JUSQU'A <*Arrêt*> |
<*Vivide*> DE <*déplacement*> | <*vide*>
<*Vivide*> ::= A <*vitesse*> | <*vide*>
<*Arrêt*> ::= <*Phrase état*>
<*déplacement*> ::= <*Nom*> | <*Nombre*>
<*Vitesse*> ::= <*Nombre*>
<*Mouvement*> ::= <*vide*> | SUIVANT <*Type A*> |
AUTOUR DE <*Type A*>
<*Point fixe*> ::= <*Nom*>

Nous aurons ainsi des phrases du type :

DEMARRER DEPLACEMENT AVANT SUIVANT OX DE T ; ou
A 12 TANT QUE A ; > 120
ARRETER ROTATION DE RLX ; SUR B1 ;

La définition de l'axe par "type A" est inutile quand l'élément mobile considéré ne possède qu'un axe de mouvement. De même, il est inutile de préciser la vitesse quand celle-ci est unique.

En outre, le terme "précision" peut être vide.

Pour les modes secondaires, nous aurons les règles suivantes :

<Phrase commande de mode secondaire> ::=
EXECUTER *<Indication de type>* *<Dénomination>*
<Indication de type> ::= *<vide>* | REPRISE EN SEQUENCE DE
<Dénomination> ::= *<Nom>* |DE*<Titre>* A *<Titre>* *<Affectations>*
<Affectations> ::= *(<Suite de correspondance>)* | *<vide>*
<Correspondance> ::= *<Nom de la variable utilisée la première fois>*
<Nom de la variable remplaçante>
<Nom de la variable utilisée la première fois> ::= *<Nom>*
<Nom de la variable remplaçante> ::= *<Nom>*

Un mode secondaire est soit cyclique, et alors il a été déclaré avec les autres éléments commandés, soit non cyclique et il est alors défini par deux titres. Les "affectations" permettent de remplacer dans une occurence d'utilisation d'un mode secondaire non cyclique des variables par d'autres variables de même type. Dans ces conditions les "correspondances" permettent d'indiquer les relations biunivoques entre variables remplacées et variables de remplacement. Un mode secondaire non cyclique ne pourra être repris en séquence car cela nécessiterait une interprétation sur ordinateur trop compliquée.

Pour les *éléments bistables ou assimilés* nous procéderons comme suit :

<Phrase commande d'élément bistable ou assimilé> ::= COMMANDER *<Type de commande>*
<Nom>

Suivant le genre d'élément commandé, nous aurons les "types de commande" suivants :

MISE SOUS TENSION DE MISE A LA MASSE DE	Commande électrique de puissance
OUVERTURE DE FERMETURE DE	Commande de systèmes mécaniques
MISE SOUS PRESSION DE MISE AU REPOS DE	Commande hydraulique
MONTEE DE DESCENTE DE ROTATION DE ARRET DE AVANCE DE RECUL DE	Commande de mouvement
ENCLENCHEMENT DECLENCHEMENT	Commande électrique de faible puissance
MESURE DE	Commande de mesure limitée dans le temps
IMPRESSION DE	
AFFICHAGE DE	
DEBUT MESURE DE FIN MESURE	Commande de mesure continue dans le temps

Il n'existe qu'un seul type de *phrase commande de registre*

<Phrase commande de registre> ::=
RANGER *<Expression>* DANS *<registre>*
<Registre> ::= *<Nom>*

Si le registre est booléen, l'expression sera booléenne ; si le registre est rationnel, l'expression sera arithmétique.

<Expression> ::= *<Expression booléenne>* | *<Expression arithmétique>* (1)

Il reste à donner la structure de *boucle*

<Boucle> ::= ENSUITE *<Suite de phrases étiquetées>* TANT QUE *<Phrase état>*

Nous avons défini au chapitre VI quatre types de *phrases-tests :*

<Phrase test> ::= *<Phrase test unique>* |
<Phrase test répétitif> |
<Phrase test pseudo répétitif> |
<Phrase test unique> ::= *<Indicateur de test>* *<Phrase état>* *<Liste de phrases états liées>*
<Indicateur de test> ::= SI | DES QUE
<Phrase test répétitif> ::= *<Suite phrases tests uniques>* TANT QUE *<Phrase état>*
<Phrase test pseudo répétitif> ::=
<Suite de phrases tests uniques> ET SI ALORS *<Phrase état>*
<Phrase état liée> ::= ET *<Phrase état>*

Nous distinguons trois sortes de *phrases états :*

<Phrase état>::=*<Nom>* *<Indication de position>* | *<Nom>* *<Etat>* | *<Expression booléenne>*
<Indication de position> ::=SUR *<Nom>* | NON SUR *<Nom>*

Un "état" est un participe passé en rapport avec un "type de commande", nous aurons ainsi :

MIS A LA MASSE, MIS SOUS TENSION, MONTE, DESCENDU,
EN ROTATION, A L'ARRET, AVANCE, RECULE ...

Les *expressions booléennes* seront structurées comme suit :

<expression booléenne>::=*<facteur booléen>* |
<facteur booléen> ∨ *<expression booléenne>*
<facteur booléen> ::= *<secondaire>* | *<secondaire>* ∧ *<facteur booléen>*
<secondaire> ::= ¬*<primaire>* | *<primaire>* |* *<primaire>* (2)
<primaire> ::= *<variable booléenne>* | *<relation>* | (*<terme booléen>*)

---

(1) Ces termes seront définis ultérieurement.

(2) * *<Primaire>* indique une valeur indifférente. Ce terme ne peut apparaître que dans les conditions initiales.

<Variable booléenne> ::= <Registre booléen> | <Elément testable booléen>

<Registre booléen> ::=<Nom>
<Element testable booléen> ::=<Nom>
<Terme booléen> ::= <Fonction booléenne> = <Fonction booléenne>
<Fonction booléenne> ::=(<Expression booléenne>)| 0 | 1
<Relation> ::= (<expression arithmétique> <comparateur> <expression arithmétique>)
<Comparateur> ::= ≤ | ≥ | > | < = | ≠
<Expression arithmétique> ::=
    <Signe> <Terme arithmétique> |
    <Signe> <Terme arithmétique> <Opérateur additif> <Expression arithmétique>
<Signe> ::= - | <vide>
<Opérateur additif> ::= + | -
<Terme arithmétique> ::=<facteur arithmétique> |
     <facteur arithmétique> <Opérateur multiplicatif> <facteur arithmétique>
<Opérateur multiplicatif> ::= × | /
<Facteur arithmétique> ::= <variable arithmétique> | <registre rationnel> |
          <Nombre> | (<expression arithmétique>)
<Variable arithmétique> ::= <Nom>
<Registre rationnel> ::= <Nom>

D'après le chapitre VI, il y a trois sortes *d'indications complémentaires* :

<Indication complémentaire> ::=
          <Déclarations de variables de sécurité> |
          <Définitions de processus d'urgence> |
          <Indications de commandes manuelles>

En ce qui concerne *les variables de sécurité*, il est nécessaire de considére deux types de déclaration :

- déclaration d'une nouvelle variable de sécurité,
- rappel d'une variable déjà déclarée pour un autre mode de fonctionnement.

D'où la structure suivante :

<Indicateur de variable de sécurité> ::= DVS
<Déclaration d'une variable de sécurité> ::=
    R <Nom de variable de sécurité> <Nom de mode de fonctionnement>
                                        <Positionnement des tests> |

<Commentaire> <Description> <Positionnement des tests>
<Nom de variable de sécurité> ::= *<Nom>*
<Nom de mode de fonctionnement> ::= *<Nom>*
<Description> ::= *<Suite de phrases tests>* | *<Nom>*
<Positionnement des tests> ::= *<Suite de titres>* | *<vide>*

Nous avons ici quelque chose de très voisin de ce qui a été proposé pour DESPROG.

En ce qui concerne les *définitions de processus* d'urgence, il est évident que la définition comprendra deux parties :

- définition de la variable aléatoire,
- définition du mode de fonctionnement aléatoire.

La description des modes aléatoires se fera ensuite avec celle des modes secondaires.

<Définition de processus d'urgence> ::=
DPU *<Définition d'un processus d'urgence> <Liste de définitions d'un processus d'urgence>*
<Définition d'un processus d'urgence> ::=
<Déclaration de la variable aléatoire> <Déclaration du mode de fonctionnement aléatoire>
<Déclaration de la variable aléatoire> ::=
R *<Nom de la variable aléatoire> <Nom de mode de fonctionnement>* |
<Commentaires> <Description>
<Nom de la variable aléatoire> ::= *<Nom>*
<Déclaration du mode de fonctionnement aléatoire> ::=
<Nom du mode de fonctionnement aléatoire>
<Nom de mode de fonctionnement> |
<Commentaires> <Nom>
<Nom du mode de fonctionnement aléatoire> ::= *<Nom>*

Nous avons ici aussi une structure analogue à celle présentée en DESPROG pour les processus d'urgence. Les remarques faites pour les déclarations de processus d'urgence restent valables pour les modes de fonctionnement aléatoires.

La définition des *éléments commandés manuellement* se fait simplement en en donnant la liste. Dans le cas des registres il importe d'indiquer si l'on peut

modifier arbitrairement le contenu ou simplement procéder à une remise à zéro.

<Indication de commandes manuelles> ::=
ICM *<Indication d'un élément commandé manuellement>*
<Liste d'indications d'un élément commandé manuellement>
<Indication d'un élément commandé manuellement> ::=
VAM | *<Nom>* *<Mention>*
<Mention> ::= *<vide>* | RAZ | MCM

VAM indique la présence d'une commande d'arrêt manuel
RAZ - - - remise à zéro manuelle de registre
MCM - - - commande manuelle de registre.

7.3 - STRUCTURE DE LA DESCRIPTION DU FONCTIONNEMENT GLOBAL DE LA MACHINE

Chaque mode principal est considéré soit comme un élément bistable, soit comme un mode de fonctionnement secondaire du fonctionnement global. Lorsqu'un mode de fonctionnement aléatoire est commun à plusieurs modes principaux, ce mode doit également apparaître dans la description du fonctionnement.

La description du fonctionnement utilisera ici un nombre restreint de phrases commandes

celles concernant les modes secondaires
et COMMANDER ENCLENCHEMENT
- DECLENCHEMENT

Les *indications de mode de fonctionnement incompatibles* seront structurées comme suit :

<Indication des modes de fonctionnement incompatibles> ::=
IMI *<Indication d'un ensemble de modes incompatibles>*
<Liste d'indications d'un ensemble de modes incompatibles>
<Indication d'un ensemble de modes incompatibles> ::=
(*<Série de Noms>*)

Pour les *verrouillages* nous aurons

<Indication des verrouillages> ::= IVR *<Verrouillage>* *<Liste de verrouillages>*

<Verrouillage> ::=
<Nom du mode de fonctionnement actif>
<Nom du mode de fonctionnement bloqué>
<Phrase état> <Liste de phrases états liées>
<Nom du mode de fonctionnement actif> ::= <Nom>
<Nom du mode de fonctionnement bloqué> ::=<Nom>

Remarque :

"Nom" "Commentaires" et "Nombre" seront structurés suivant les règles définies pour DESPROG.

4 - PRESENTATIONS

Définition d'une machine :

DM / Système de piquage de cuves à électrolyse ; SPCE ; 4

Indication d'environnement :

IEV 1;4; 2;3; 5;7;

Définition et description des sous-machines

DSM / Portique de piquage ; PP ; 3
/ Chargeur d'alumine ; CA ; 2

DES PP; / Fonctionnement en piqueur ; FP ;
/ Fonctionnement en chargeur ; FC ;

Description d'un mode de fonctionnement

DMF FP ;

Description de base

Déclaration d'élément commandé

DCC / Portique ; PO ; EM
/ Piquage ; SPP; MFS

Déclaration d'un élément testable

DCT / Commande de départ ; CD; CE
/ Commande de répétition ; M; CE

Déclaration d'un positionnement axial

AXE DE P0 ; (0;0;0) (10;0;0) (A;0;B;10;) (1;) (1;)

Description d'un fonctionnement :

ICI P0 SUR A ; ET * M ; ET D ;
DEBUT DES QUE CD ;
T1 ; DEMARRER DEPLACEMENT AVANT DE P0 ;
T2 ; SI ¬ M ; TANT QUE P0 ; NON SUR B ;
ARRETER DEPLACEMENT AVANT DE P0 ;
DEMARRER DEPLACEMENT ARRIERE DE P0 ;
JUSQUE P0 ; SUR A ;
FIN
DE T2 SI M ; DES QUE B ; ARRETER DEPLACEMENT
AVANT DE P0 ; DEMARRER DEPLACEMENT ARRIERE
DE P0 ; DES QUE P0 ; SUR A ; ARRETER DEPLACEMENT ARRIERE
DE P0 ; ALLER A T1 ;

Indications complémentaires

Déclarations de variables de sécurité

DVS / Absence de chariot ; AC ; P ; T1 ;

Déclaration de processus d'urgence :

IPU / Cuve enflammée ; CE ; / Piquage aléatoire ; PA ;

Commandes manuelles

ICM VAM P0 ; R1 ; RAZ

Fonctionnement global

Verrouillages :

IVR MF1 ; MF2 ; P ; OUVERT ET E ; MIS A LA MASSE
FDE

Chapitre VIII

INTERPRETATION D'UNE DESCRIPTION EN DESAUCYK

Le problème est de déterminer la façon dont sera exploitée une description en DESAUCYK. Pour cela, il importe de connaître la relation entre une description en DESAUCYK et les organigrammes de commande et de processus correspondants.

1 - TRAITEMENT D'UNE DESCRIPTION EN DESAUCYK

Le premier travail à effectuer est la détermination des différentes variables. Pour les variables de sortie, correspondant à des éléments mobiles, il faut considérer chaque axe de déplacement: à chacun d'eux correspondra autant de variables de sortie qu'il y aura de vitesses différentes dans chaque sens de déplacement. Ainsi la commande d'un élément se déplaçant en translation le long d'un seul axe avec deux vitesses possibles par sens nécessite quatre variables de sortie. Pour les mouvements de rotation les conclusions sont identiques.

Pour les *éléments bistables*, une variable de sortie sera associée à chaque sens de commutation ; les *éléments mobiles bistables* relèvent, eux, à la fois des deux catégories précédentes.

Les *modes secondaires*, *fonctions mesures* et *fonctions temps* constituent des processus secondaires ou des processus types tels qu'ils ont été définis aux chapitres I et III. Les *registres* déclarés restent inchangés au niveau des organigrammes.

Les éléments testables ainsi que les positions définies sur ou autour des axes de déplacement constituent l'ensemble des *variables d'entrée*.

On peut remarquer que toute expression booléenne complexe (Ex : $R = B^2+4AC$) sera décomposée suivant les règles définies au chapitre précédent. Elle ne constituera donc pas une variable d'entrée, mais entraînera par contre la création de plusieurs registres annexes.

Ayant déterminé les variables d'entrée et de sortie, il est possible de reconstituer les *enchaînements*.

Une phrase décrivant un déplacement sera le plus souvent structurée comme suit :

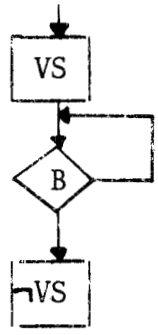
<verbe> <type du mouvement> <élément> <vitesse> <précisions>

les quatre premiers termes permettent de déterminer la variable de sortie considérée et la valeur attribuée à cette variable.

Les "précisions", quand elles existent, indiquent les conditions du changement de valeur de la variable de sortie considérée.

Ex : ENCLENCHER DEPLACEMENT SUIVANT AXE OX DE P0; A 12 JUSQUE P0; SUR B;

VS



les enchaînements se retrouvent facilement.

Une phrase commandant un élément bistable est structurée comme suit :

<verbe> <type de commande> <élément>

cette phrase implique la mise à 1 de la variable de sortie correspondante considérée comme une variable à double effet.

Enfin, dans certains cas particuliers, les phrases commençant par

EXECUTER...
POSITIONNER...
AMENER...
COMMANDER MESURE...
- ENCLENCHEMENT (fonction temps)
- DECLENCHEMENT (fonction temps)

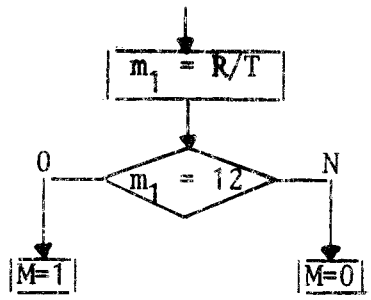
correspondent à l'exécution de processus secondaire ou de processus types.

Par contre,le rangement d'une valeur dans une mesure correspond à une commande simple.

Dans le cas où l'on range le résultat d'une expression complexe, il faut redécomposer l'expression de manière à faire apparaître les ordres élémentaires.

De même lorsque l'on teste la valeur d'une expression complexe, il est nécessaire de redécomposer ce test :

Ex : M = (12 = R/T) se décompose

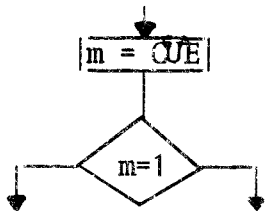


Il reste à définir la *signification des termes* ET et OU.

- Un ET entre deux phrases commande indique l'exécution simultanée de plusieurs commandes.

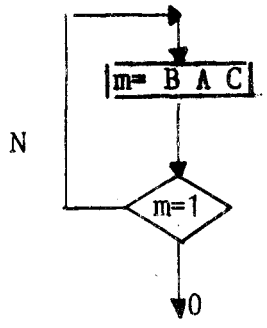
- Un OU entre deux phrases test indique une structure du type de celle de l'exemple suivant :

SI B- SUR C- OU SI D- SUR E-



- Un ET entre deux phrases états dans un test implique la structure du type de celle de l'exemple suivant :

DES QUE B- ET A- SUR C-



Quand il n'y a pas de liaison, les enchaînements correspondent à ceux des phrases dans le texte.

Il est donc facile de reconstituer l'organigramme de commande à partir de la description en DESAUCYK. Pour réaliser l'organigramme de processus il importe de procéder à la *détermination des ensembles incompatibles de variables d'entrée et de commandes et à celle des ensembles de commandes modifiant la valeur des variables d'entrée :*

Les ensembles incompatibles de variables d'entrée sont, la plupart du temps, consécutifs à l'implantation géographique de la machine. Ex : n positions sur un axe OZ : A_1, A_2,...,A_n constituent un ensemble de variables d'entrée incompatibles.

De même deux variables d'entrée E1 et E2 correspondant aux deux états d'un système bistable forment un ensemble de variables d'entrée incompatibles.

Les ensembles de commandes incompatibles ont deux origines :

- les registres,
- les commandes de moyens d'exécution.

Dans le premier cas, la détermination des ensembles de commandes incompatibles est facile.

Dans le second cas, il y a deux moyens de la faire :

- l'ensemble des commandes d'un élément bistable, celui des commandes relatives à un même mouvement d'un élément mobile constituent des ensembles de commandes incompatibles,

- l'étude de l'implantation géographique permet de déterminer des ensembles de commandes incompatibles qui entraînent soit la création de variables de sécurité soit

une modification de conception.

Les ensembles de commandes modifiant la valeur de variables d'entrée se déduisent facilement de la description des éléments mobiles ou bistables. Une description de fonctionnement en DESAUCYK permet donc de créer l'organigramme de processus et le graphe d'état [1].

2 - POSSIBILITES DE CONTROLE ET DE MISE AU POINT

Etant donné qu'une description de fonctionnement en DESAUCYK permet de reconstituer les organigrammes de commande et de processus, nous envisagerons un traitement machine des descriptions en DESAUCYK en quatre étapes :

1. Prise en compte et vérification de la validité de la description et détermination des contraintes,

2. Elaboration de l'organigramme de commande,

3. Elaboration de l'organigramme de processus,

4. Création du graphe d'état.

Les étapes 1, 2 et 3 nécessitent un dialogue opérateur-machine.

La *détermination des contraintes* se fera sur un mode conversationnel au fur et à mesure de la prise en compte de la description par l'ordinateur.

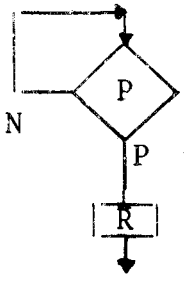
La *mise au point de l'organigramme de commande* nécessitera les mêmes contrôles que celle effectuée à partir d'une description en DESPROG. De plus des vérifications d'ordre sémantique seront exécutées. Par exemple, dans le cas d'une translation de long d'un axe orienté dont FC représente la position de fin de course dans le sens positif, les variables de sortie contrôlant le mouvement dans le sens positif doivent être remises à zéro dès que le mobile en mouvement atteint FC. On peut déterminer des règles analogues pour les commandes d'éléments bistables. Enfin, une étude systématique de la description et notamment du positionnement des différents organes permet l'introduc-

---

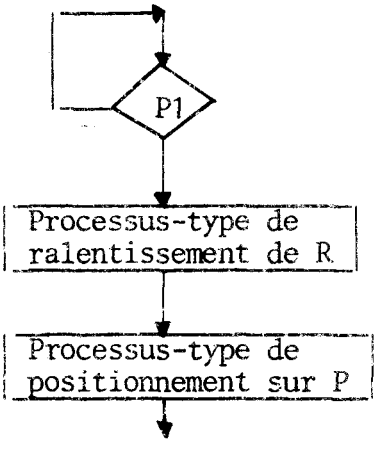
(1) Nous traiterons le problème du développement des fonctions-temps, fonctions-mesures et fonctions-types ultérieurement.

tion systématique d'un certain nombre de mesures de sécurité. La connaissance des contraintes s'exerçant sur les différents moyens d'exécution permet soit d'envisager des mesures de sécurité supplémentaires (isolation électrique, protection d'appareils de mesure) soit d'introduire des processus-types supplémentaires.

Par exemple, soit un système en rotation R à arrêter sur une position P. Le système de commande le plus simple peut être le suivant



Si l'inertie du système, sa vitesse de rotation et la précision du positionnement dépassent certaines normes, on peut être amené à envisager le système suivant :



P1 position repérée supplémentaire précédant P

Ce genre de travail peut être réalisé soit automatiquement par l'ordinateur si celui-ci dispose de tableaux de contraintes avec les solutions qui leurs sont associées, soit par un dialogue opérateur-machine.

Au niveau de l'organigramme de processus, les fonctions-types, temps ou mesure seront remplacées par un organigramme. Selon la sévérité des contraintes, on pourra être amené à choisir entre différentes solutions pour une même fonction. Ainsi suivant la masse en mouvement, un ralentissement pourra se faire progressivement ou par paliers. De même que pour l'introduction de fonctions-types supplémentaires ces choix seront effectués soit automatiquement soit sur un mode conversationnel.

L'ordinateur ajoutera automatiquement une classe d'éléments commandés supplémentaires correspondants aux impulsions qui sont utilisées dans la commande de diverses fonctions (temporisation, mesure...) et apparaissent donc dans le développement de ces fonctions.

Des variables de contrôle d'état d'élément bistable pourront être introduites systématiquement par la machine et constitueront avec l'ensemble des positions repérées une classe supplémentaire d'éléments testables appelés variables de contrôle de fonctionnement. Les variables émises par les fonctions-temps seront rassemblées de manière à former une classe d'éléments testables nommés variables de cadencement. De cette façon l'organigramme de processus pourra être décrit sans faire appel à des phrases états autres que des expressions booléennes.

L'étude des positionnements permettra à l'ordinateur de repérer les variations inactives de variables d'entrée ici appelées élément testable. Une phrase commande supplémentaire uniquement utilisée par la machine est alors nécessaire, elle sera structurée comme suit :

ATTENDRE *<repositionnement>*

avec *<repositionnement>* ::= *<élément testable>* | ¬*<élément testable>*

L'organigramme de processus étant établi, le graphe d'état sera construit grâce à l'algorithme présenté au chapitre II.

Nous sommes maintenant en mesure d'effectuer des comparaisons entre DESAUCYK et DESPROG tant du point de vue utilisation que du point de vue traitement sur ordinateur. Par ailleurs, étant donnée la complémentarité des deux langages dans leur utilisation, (DESAUCYK nécessite peu de connaissances en automatique ou en informatique alors que DESPROG est réservé à des spécialistes des processus), il est intéressant d'étudier les possibilités de transcription de DESAUCYK en DESPROG.

- ETUDE COMPAREE DE DESPROG ET DESAUCYK

Les possibilités des deux langages sont nettement distinctes :

Du point de vue utilisation, DESPROG permet de décrire n'importe quel processus avec suffisamment de précision pour permettre la réalisation quasiment automatique des organigrammes de commande et de processus ainsi que le graphe d'état correspondant.

Son utilisation suppose néanmoins :

- que tous les éléments constitutifs du système ont été définis,
- qu'un choix de réalisations technologiques a été fait. C'est donc un langage destiné à des techniciens spécialistes de telle ou telle catégorie de processus.

Par contre DESAUCYK est d'une utilisation plus restreinte : il permet seulement la description des automatismes séquentiels, mais est utilisable par des non automaticiens c'est-à-dire qu'il est possible d'envisager son utilisation dans la description d'un cahier des charges par un utilisateur d'automatisme. A partir d'un cahier des charges décrit en DESAUCYK, un ordinateur est en mesure, par un travail conversationnel d'établir les organigrammes de commande et de processus ainsi que le graphe d'état. DESAUCYK se caractèrise particulièrement

- par sa ressemblance avec le langage courant,
- par le fait qu'il ne nécessite aucune connaissance en automatisme et permet néanmoins de décrire avec précision le fonctionnement d'un automatisme séquentiel sans empiéter sur le domaine de la conception.

Du point de vue description, DESAUCYK présente, dans le domaine restreint des automatismes séquentiels, les mêmes possibilités de description que DESPROG ; la possibilité de décrire en DESAUCYK non seulement le fonctionnement de l'automatisme mais également son environnement, son implantation géographique et les contraintes de fonctionnement permet une automatisation beaucoup plus poussée de la conception puisque l'on peut envisager de faire procéder par l'ordinateur, en partie grâce à une bibliothèque de processus-type, en partie par un fonctionnement conversationnel, à des choix d'ordre technologique et à la détermination des différentes variables d'entrée et de sortie.

Si nous comparons le plan d'une description en DESAUCYK avec celui d'une description en DESPROG, nous obtenons le tableau ci-dessous :

DESAUCYK	DESPROG
Découpage en sous-machines	Découpage en sous-processus
Etude de chaque sous-machine	Etude de chaque sous-processus
Etude de chaque mode de fonctionnement principal	Etude de chaque processus irréductible
Etude du fonctionnement global	Etude du fonctionnement global

L'identité de structure apparaît immédiatement ; de plus une sous-machine correspond à un sous-processus, et un mode de fonctionnement à un processus irréductible.

Si maintenant nous comparons le plan d'une description de mode principal en DESAUCYK à celui d'une description de processus irréductible en DESPROG, nous obtenons :

DESAUCYK	DESPROG
Déclaration des éléments commandés	Déclaration des éléments commandés
Déclaration des éléments testables	Déclaration des variables d'entrée
Déclaration des fonctions-temps	Déclaration des processus secondaires
Déclaration des positionnements	
Conditions initiales	Conditions initiales
Description du fonctionnement	Description du fonctionnement
Indications complémentaires	Indications complémentaires

Compte tenu des données du paragraphe 8.1, le passage de DESAUCYK à DESPROG ne pose donc pas de problème important :

Un mode secondaire correspond à un processus secondaire cyclique. Une fonction-mesure correspond à un processus-type ; il en est de même des fonctions-temps. Les variables de sortie, les ensembles incompatibles de commande, de variables d'entrée, les ensembles de commandes modifiantes se déduisent facilement des éléments testables et de la description géographique.

La seule question litigieuse est la détermination du type des variables de sortie : simple effet, double effet, impulsion. L'utilisation d'impulsion n'intervient guère que dans certains processus-type de mesure ou de déclenchement de minuterie ou temporisation ; la détection de variable de ce type est réalisée à partir de la connaissance de la nature de l'élément commandé ; par contre le choix entre une variable à

simple ou à double effet est beaucoup plus arbitraire.

Ainsi, à condition de choisir au préalable le type simple ou double effet, il est permis d'envisager une traduction automatique d'une description de DESAUCYK en DESPROG. Cette description n'est possible qu'après exécution des deux premières étapes du traitement machine d'une description en DESAUCYK de manière à connaître toutes les variables intervenant dans le fonctionnement de l'équipement.

Chapitre IX

EXTENSION DES POSSIBILITES DES LANGAGES

Jusqu'à présent les systèmes décrits correspondaient à des processus séquentiels cycliques. Dans ce chapitre, nous étudions une généralisation des méthodes de descriptions étudiées précédemment suivant deux voies nettement distinctes : tout d'abord nous envisagerons la possibilité de décrire certains *asservissements* : en effet, l'introduction des *contraintes* [1] dans une description en DESAUCYK amène à considérer deux problèmes :

- la précision d'une vitesse, d'un positionnement,
- le temps de réponse maximal à une commande en vitesse ou en position.

A partir d'un certain niveau, des contraintes de précision en position ou en vitesse peuvent entraîner l'introduction d'asservissements, linéaires ou non dans l'équipement étudié. En outre, dans des équipements complexes, les systèmes séquentiels et les asservissements peuvent être étroitement imbriqués, d'où la nécessité, du moins au niveau du langage DESAUCYK, de pouvoir décrire des asservissements.

D'autre part, nous avons vu au chapitre II qu'un système programmé se définissait à l'aide des mêmes éléments (hormis les enchaînements) qu'un processus séquentiel ; l'utilisation de tels systèmes est courante dans les équipements d'automatisme (commande numérique de machines-outils, machines textiles...). Il semble donc intéressant de pouvoir les décrire à l'aide de DESPROG ou DESAUCYK.

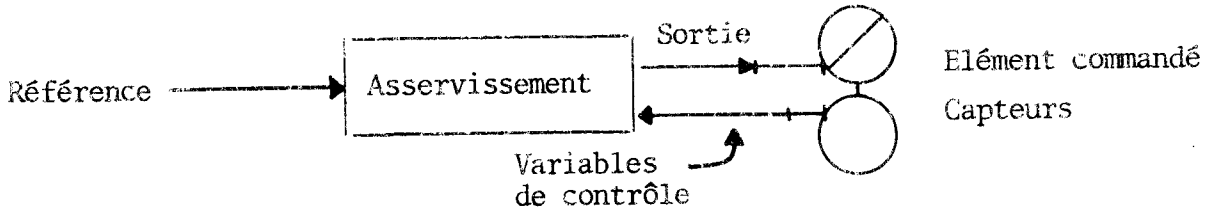
Enfin nous passerons rapidement en revue les différentes possibilités de réalisations.

---

(1) Cf. Chapitre VI.

9.1 - METHODE DE DESCRIPTION DES ASSERVISSEMENTS ET DES SYSTEMES PROGRAMMES EN DESAUCYK

Un asservissement permet de mettre en relation les valeurs d'une variable de sortie avec celles d'une variable d'entrée ; cette relation est le plus souvent un coefficient multiplicatif mais les contraintes physiques impliquent parfois une relation plus complexe (fonction de transfert). Cependant, par rapport aux systèmes séquentiels, un asservissement peut être représenté par le schéma suivant :



c'est-à-dire que le système séquentiel "voit" :

- un registre contenant la référence,
- une commande de verrouillage (éventuelle)

et que du point de vue séquentiel, seul le temps de réponse du système sera pris en compte et utilisé pour la réalisation du graphe d'état.

Pour définir un asservissement, il suffira donc de donner la grandeur commandé et le registre contenant la référence ; du point de vue contrainte il faudra fournir :

- la précision sur la référence,
- le temps maximal de réponse,
- les contraintes physiques (masse, inertie, etc...).

Ces dernières ne sont pas utilisées pour la description du fonctionnement séquentiel mais peuvent servir ultérieurement.

Les systèmes programmés (cf. chapitre II) se définissent par :

- des éléments commandés (variables de sortie, registres),
- des éléments testables (variables d'entrée),
- des instructions (processus-types, processus secondaires).

Comme pour les processus classiques, il sera nécessaire de préciser des indications complémentaires ; il faudra en outre définir :

- les conditions d'initialisations (éventuellement),
- les relations de précédence imposées entre les instructions.

Nous considérerons uniquement le cas d'instructions ne pouvant être exécutées simultanément. Les systèmes programmés seront synchrones : il y aura donc obligatoirement une déclaration d'une fonction-temps horloge.

Nous allons étudier successivement des méthodes de description de commandes d'asservissement puis des systèmes programmés.

Les éléments commandés par asservissement sont soit des éléments mobiles, soit certains éléments que nous classerons parmi les éléments bistables, comme des systèmes de chauffage, soufflerie etc... Pour les éléments mobiles, la notion d'asservissement sera introduite dans les "précisions" en indiquant :

- A (élément) pour les asservissements en vitesse,
- DE (élément) pour les asservissements en position.

La position sera toujours définie par rapport à l'origine des abscisses ou des positions angulaires. Ces deux indications pourront apparaître simultanément.

Le terme "élément" correspond soit à un registre, soit à une mesure.

De plus l'existence de contraintes sévères sur les positionnements et la vitesse peut amener l'ordinateur à proposer d'introduire des asservissements supplémentaires.

Lorsqu'il s'agit de commandes asservies d'éléments assimilés aux éléments bistables, il suffit d'utiliser les phrases commandes d'éléments bistables auxquelles seront adjointes des précisions du type :

SUR (élément) pour les asservissements en position,
A (élément) pour les asservissements en vitesse.

L'élément de référence est alors soit un registre soit une mesure. Les deux indications complémentaires peuvent être utilisées simultanément.

Dans le cas d'asservissement en vitesse d'un mouvement, la description géométrique ne comportera pas de déclaration de vitesse ; par contre, même dans le

cas d'un asservissement en position, il existera au moins deux positions repérées sur l'axe qui détermineront des signaux de fin de course de sécurité.

La description des systèmes programmés nécessite celle

- des instructions,
- des relations d'ordre imposées entre les instructions.

La description du fonctionnement se réduira à celle des conditions initiales en outre il faudra indiquer l'élément testable de commande du système.

Les instructions sont de deux types :

- des commandes simples se réduisant à une phrase,
- des modes secondaires cycliques ou non cycliques.

Les commandes de fonctions-mesures seront assimilées à des commandes simples. Les systèmes programmés étant synchrones, il n'existera qu'une seule fonction-temps : l'horloge de base. Un système programmé n'est pas décomposable en sous-machines mais une sous-machine d'un système non totalement programmé peut être un système programmé.

La description élémentaire d'un système programmé commencera donc par un ensemble de déclarations classiques ; puis la description du fonctionnement sera remplacée par :

- un ensemble de conditions initiales [1],
- la déclaration des instructions,
- les indications concernant les relations de précédence imposées entre les instructions.

Enfin la description se terminera par des indications complémentaires.

---

(1) Il peut exister des cas où aucune initialisation n'est nécessaire ; il est alors inutile d'indiquer des conditions initiales.

Une instruction se définira :

- par les variables sur lesquelles peut s'exécuter l'instruction ; ce sont des variables sélectionnées propres à l'instruction ; elles seront remplacées dans les programmes par des variables sélectionnables,

- par la ou les phrases décrivant le déroulement de l'instruction.

Pour indiquer les relations de précédence imposées entre les instructions, il suffit de ranger leurs noms par couple.

Ces remarques nous conduisent à modifier la syntaxe du langage DESAUCYK :

Le terme "description élémentaire" sera redéfini de la manière suivante :

<Description élémentaire> ::=
PROG *<Suite de déclarations d'éléments constitutifs>*
<Indication des conditions initiales>
<Déclarations d'instructions>
<Liste d'indications complémentaires> FDE

Les conditions initiales n'existant pas nécessairement nous aurons :

<Indication des conditions initiales> ::= CI *<Conditions initiales>* DCI | DCI

Pour les déclarations d'instruction nous aurons les règles suivantes :

<Déclarations d'instructions> ::= *<Suite de déclarations d'une instruction>* *<Indications des relations de précédence>* *<Liste de descriptions d'un mode de fonctionnement>*

Une instruction se déclarera comme suit :

<Déclaration d'une instruction> ::= *<Nom de l'instruction>*
<Liste d'affectations> *<Suite de phrases>* FIN
<Nom de l'instruction> ::= *<Nom>*
<Affectation> ::= (*<Suite de Noms>*)
<Indications des relations de précédence> ::=

<vide> | IRP *<Suite d'indications d'une relation de précédence>*
<Indication d'une relation de précédence> ::=
(*<Nom de l'instruction>* *<Nom de l'instruction>*)

Remarque

Un système programmé possède obligatoirement une commande de départ qui permet de lancer l'exécution d'un programme.

Exemple

Soit par exemple un ensemble de chariots C_1, C_2...C_N se déplaçant chacun suivant un seul axe. On aura alors une instruction du type suivant

I1 ; (C ; L ;) DEMARRER MOUVEMENT AVANT DE C ; DE L ;

C ; a été déclaré comme élément commandé sélectionnable parmi C_1, C_2 ... C_N

L ; a été déclaré comme un élément testable (paramètre)

I1 ; est donc une instruction de déplacement d'un chariot dans un sens d'une longueur donnée L ;

Dans un programme on écrira I1 ; (C_j ; L;) pour indiquer que l'instruction s'applique au chariot C_j ; qui doit se déplacer d'une longueur L ;

Compte tenu des correspondances entre les langages DESPROG et DESAUCYK, il est possible d'envisager une modification de structure analogue de DESPROG de manière à permettre la description de systèmes programmés.

9.2 - MODIFICATION DE DESPROG

Un système programmé ne pouvant être décomposé en sous-processus ou en processus irréductibles, les systèmes programmés constitueront un type nouveau de "processus".

Les conditions initiales seront facultatives comme en DESAUCYK. La description des enchaînements sera remplacée par la déclaration des instructions suivie des

indications de relations de précédence entre instructions [1].

Un rôle supplémentaire sera défini pour les variables d'entrée : CD commande de départ. La commande de départ n'aura pas besoin d'être décrite comme une variable de cadencement.

Dans le cas d'un système programmé, "Type P" se décomposera comme suit :

<Type P> ::= SP *<Période d'horloge>*

Le terme "Définition et description des processus autonomes" présentera une troisième possibilité de décomposition :

<Définition et description des processus autonomes> ::=
<Indication possible des conditions initiales>
<Déclarations d'instructions>
<Indications des relations de précédence>
<Indication possibles des conditions initiales> ::=
<Indication des conditions initiales> | *<vide>*

Les déclarations d'instruction se structurent comme suit :

<Déclarations d'instruction> ::= DCC *<Suite de déclarations d'une instruction>*
<Déclaration d'une instruction> ::=
<Commentaires> *<Nom>* *<Ensemble de variables de définition>* *<Développement>*
<Développement> ::= *<Commande>* FIN | *<Suite de phrases>* FIN

Enfin les relations de précédence entre instructions seront définies par :

<Indication des relations de précédence> ::=
<vide> | IRP *<Suite d'indication d'une relation de précédence>*
<Indication d'une relation de précédence> ::=
(*<Nom d'une instruction>* *<Nom d'une instruction>*)
<Nom d'une instruction> ::= *<Nom>*

---

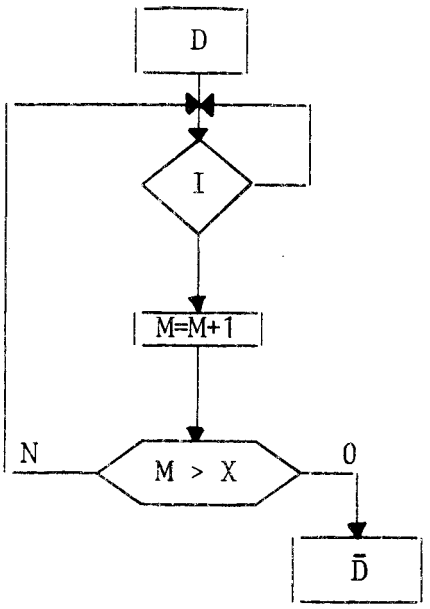
(1) Pour les éléments du langage non décomposés, il suffit de se reporter au paragraphe précédent.

Exemple :

```
/Instruction déplacement chariot ; DC ; (X;)
     ENCL D ; PUIS M ; = M; +1 DES QUE I ;
     M; = M; +1 PUIS DECL D; SI M; > X;
     M; = M; +1 PUIS M; = M; +1  SI M ; > X ; DES QUE I ;
     DECL D ; PUIS FIN
```

D; est une variable de sortie commandant le déplacement de DC; , X; est un paramètre affiché, M; est un registre et I; une variable d'entrée émise par un capteur

Cette instruction se représente par l'organigramme de commande suivant



Remarque concernant l'interprétation sur ordinateur

Que ce soit dans le cas d'une description en DESAUCYK ou en DESPROG, la première étape du traitement machine (prise en compte et vérification de la validité de la description) reste nécessaire. Par contre au lieu de réaliser ensuite les organigrammes de commande et de processus généraux, il faut réaliser pour chaque instructic correspondant à un processus secondaire :

- l'organigramme de commande,
- l'organigramme de processus
- le graphe d'état.

Ce qui sera généralement simple.

Les instructions étant correctement et complètement définies, une mise sous forme d'automate de l'ensemble des instructions conformément aux données du chapitre II est alors envisageable, c'est-à-dire que l'on aborde le problème de la conception. La conception d'un système implique que l'on détermine, à partir de la description détaillée, les éléments technologiques qui interviendront au niveau de la réalisation.

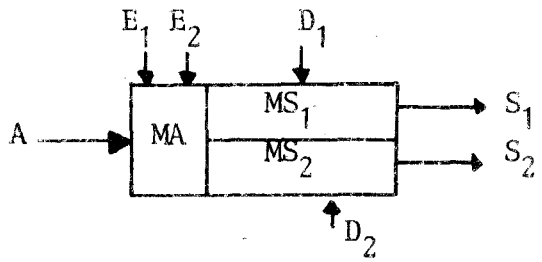
9.3 - UTILISATION DES DESCRIPTIONS DANS LE TRAVAIL DE CONCEPTION

Il y a trois directions de recherche : le cas des systèmes asynchrones, celui des systèmes synchrones et enfin celui des systèmes programmés.

Dans le cas d'un *système asynchrone*, il existe déjà actuellement des programmes, qui à partir du graphe d'état permettent de construire puis de réduire les équations booléennes correspondant au fonctionnement.

Il est cependant possible d'envisager d'autres moyens de réalisation notamment *la réalisation du bloc de commande à l'aide de modules de contrôle.*

Un module de contrôle est un système logique structuré suivant le schéma :



Ce module comporte trois mémoires booléennes :

- une mémoire d'activation : MA
- deux mémoires de commande sortie : MS_1 et MS_2.

La mémoire MA est mise à 1 par le passage à 1 du signal d'activation A.

La mémoire MS_1 est mise à 1 par le passage à 1 du signal MA $\wedge$ E_1, E_1 étant un signal d'entrée supplémentaire.

La mémoire MS_2 est mise à 1 par le passage à 1 du signal $MA \wedge E_2$, E_2 étant un signal d'entrée supplémentaire.

D_1 et D_2 sont les signaux de mise à zéro de MS_1 et de MS_2 (désactivations de MS_1 et MS_2), MA est mise à zéro par le signal $\overline{MS}_1 \wedge \overline{MS}_2$. $S_1=MS_1$, $S_2=MS_2$ sont les deux sorties du module.

Ce module permet de réaliser *cinq fonctions de base* qui apparaissent dans un déroulement de processus ; ces fonctions sont représentées par un graphe orienté appelé graphe de définition et qui ne présente qu'un seul sommet, appelé sommet central du graphe. Les arcs seront repérés par des indices (cf. la notion de réseau définie ultérieurement). A chaque arc "ℓ" nous associerons une variable $v(\ell)$ à valeur dans [0,1], lorsque $v(\ell) = 1$ l'arc correspondant sera dit activé, dans le cas contraire il sera dit désactivé.

Nous appellerons arc entrant un arc dont l'extrêmité est le sommet central et arc sortant un arc dont l'origine est le sommet central.

Nous considèrerons dans ce qui suit les règles suivantes :

- une condition externe au module peut forcer la variable associée à un arc entrant à 1 mais pas à 0,

- une condition externe au module peut forcer la variable associée à un arc sortant à 0 mais pas à 1.

Les fonctions de base sont les suivantes :

- la fonction SD (successeur double) permet de contrôler l'exécution de deux commandes simultanées,

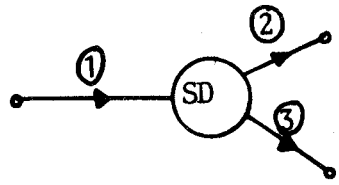
- la fonction PD (prédecesseur double) permet de contrôler une commande ayant deux prédécesseurs directs,

- la fonction MT (module test) permet de contrôler l'exécution d'un test,

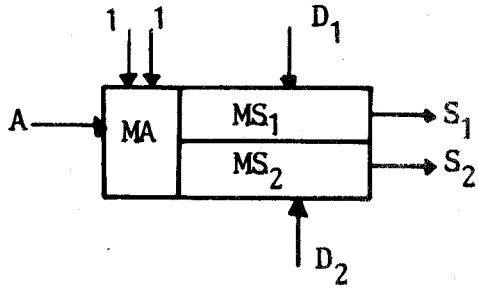
- la fonction MS (module successeur) permet de contrôler une séquence linéaire,

- la fonction MR (module rebouclage) permet de contrôler des rebouclages sur des tests.

Exemple : Soit la fonction de base SD (successeur double) dont le graphe de définition est le suivant :



Dès que v(1) passe à 1, v(2) et v(3) passent à 1. Le module S0 peut se réaliser à l'aide du module de contrôle comme suit



v(1) = MA
v(2) = S1
v(3) = M2

E1 = E2 = 1 entraîne que MS_1 et MS_2 passent à 1 quand MA passe à 1 ; MS_1 et MS_2 sont remises à zéro par D_1 et D_2 respectivement.

Nous associerons à chaque fonction de base un *diagramme de fonctionnement*, défini par un graphe orienté appelé graphe de fonctionnement. Les sommets de ce graphe sont divisés en trois classes :

- les sommets "commande",
- les sommets "attente",
- les sommets "tests".

A chaque sommet est associé le nom d'un arc du graphe de définition.

Un sommet "commande" relatif à un arc entrant ℓ implique que $v(\ell)$ passe de 1 à 0.

Un sommet "commande" relatif à un arc sortant ℓ implique que $v(\ell)$ passe de 0 à 1.

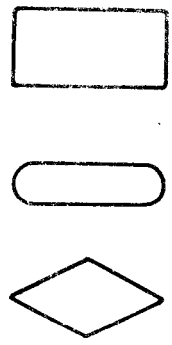
Un sommet "attente" relatif à un arc entrant ℓ implique l'attente du passage de $v(\ell)$ à 1.

Un sommet "attente" relatif à un arc sortant ℓ implique l'attente du passage de $v(\ell)$ à 0.

Un sommet "test" correspond à l'exécution d'un test sur une condition externe, le résultat du test détermine l'arc sortant du graphe de définition dont la valeur associée est forcée à 1.

Notons que chaque sommet du graphe de fonctionnement est l'extrêmité d'un arc.

Nous noterons :

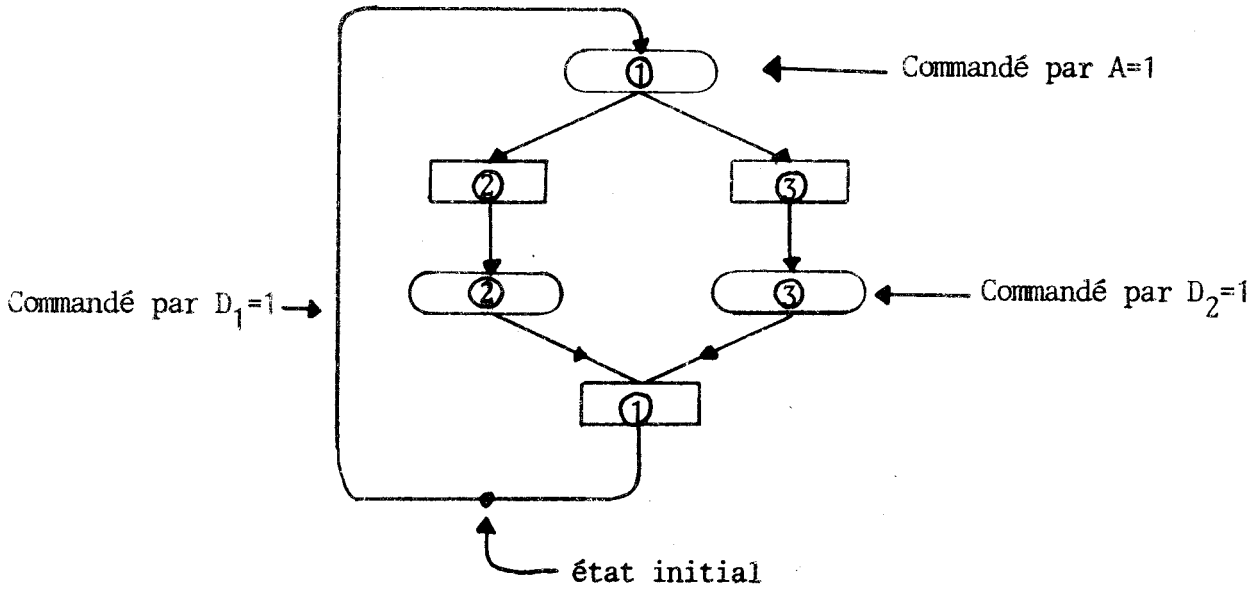


un sommet commande

un sommet attente

un sommet test

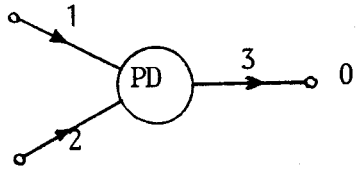
Le graphe de fonctionnement associé à la fonction SD est le suivant :



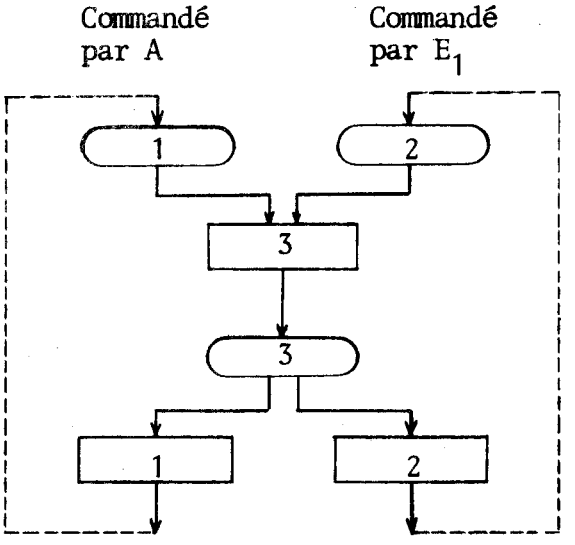
Un module de contrôle M est dit dans son état initial si tous les arcs d'entrée et de sortie du graphe de définition sont à zéro c'est-à-dire que toutes les variables $v(\ell)$ associées aux sommets du graphe de fonctionnement sont à zéro.

Les *autres fonctions de base* sont définies comme suit :
Nous schématiserons le montage réalisant la négation d'une variable par 

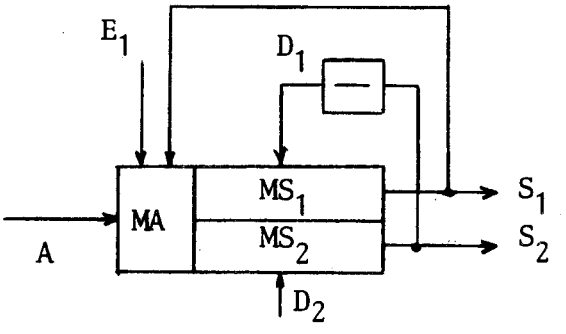
Module PD (prédécesseur double)



Graphe de définition



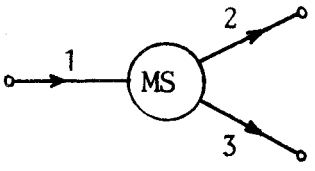
Graphe de fonctionnement

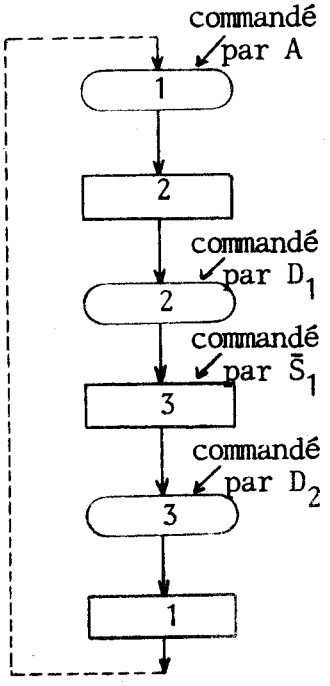


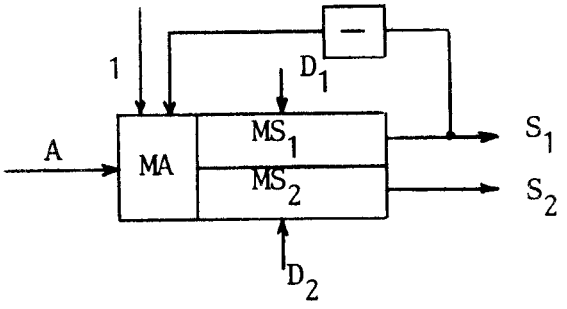
$MA \equiv v(1)$
$MS_1 \equiv v(2)$
$MS_2 \equiv v(3)$

Utilisation du module

Module Séquence : MS

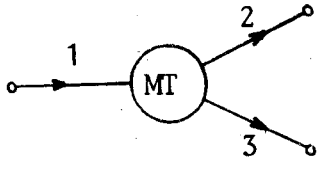


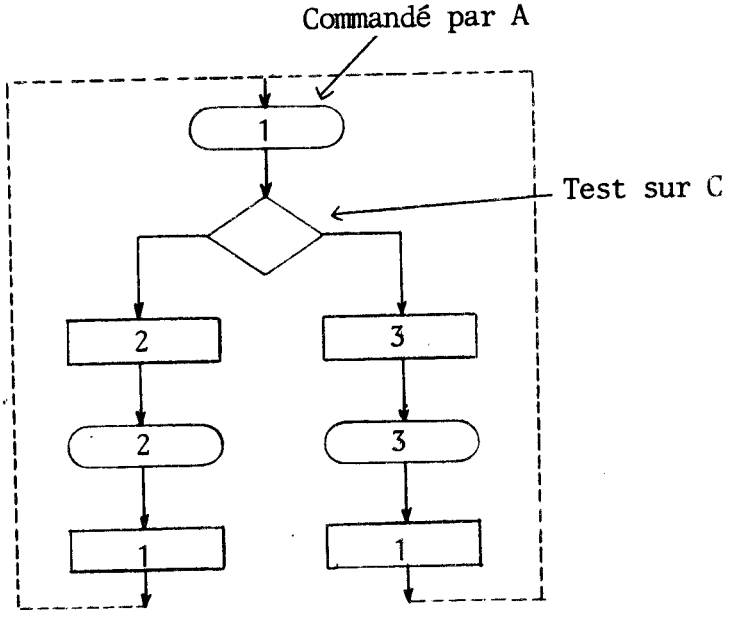


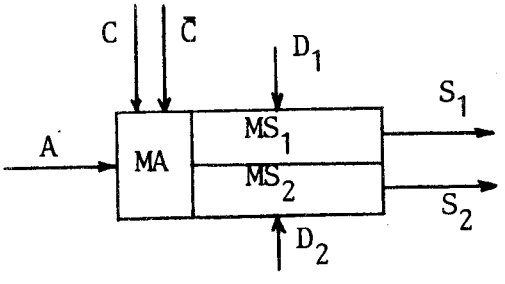


MA ≡ v(1)
MS_1 ≡ v(2)
MS_2 ≡ v(3)

Module test : MT

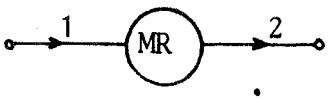


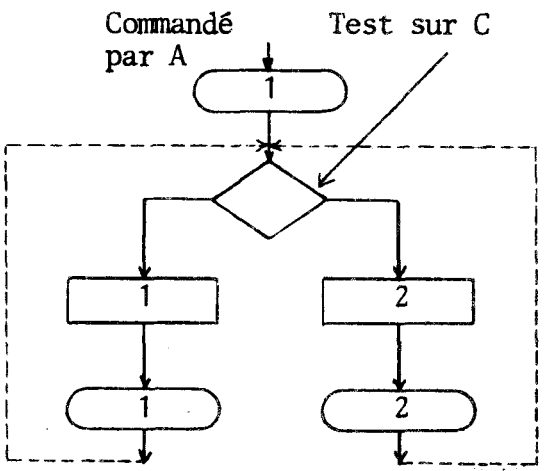


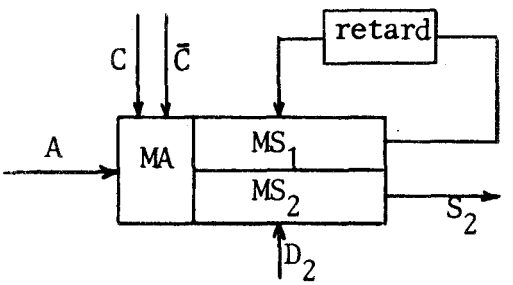


MA ≡ v(1)
MS_1 ≡ v(2)
MS_2 ≡ v(3)

Module Répétitif : MR







MA ≡ v(1)

MS_2 ≡ v(2)

Un *réseau de modules de contrôle* est constitué par un ensemble de modules connectés entre eux. Les fonctions réalisées par ce réseau sont définies par le réalisateur.

Un tel réseau se définit par un graphe orienté dit graphe du réseau.

Le graphe G(R) d'un réseau est construit au moyen des règles suivantes :

1) Si $S_M(G)$ est l'ensemble des sommets de G qui correspondent à un module de contrôle du réseau R.

$$S_M(G) \neq \emptyset \quad \text{et} \quad S_M(G) \subseteq S(G)$$

2) Chaque arc du graphe a au moins une extrêmité qui appartient à $S_M(G)$.

3) Les liaisons entre sommets de $S_M(G)$ concordent avec les fonctions des modules de contrôle correspondants. Les sommets appartenant à $S(G) \setminus S_M(G)$ sont extrêmité d'un seul arc.

Un arc dont une seule extrêmité correspond à un module de contrôle est appelé :

- arc entrant si l'arc est orienté vers cette extrêmité,
- arc sortant si l'arc est issu de cette extrêmité.

De même, les éléments de $S(G) \setminus S_M(G)$ sont appelés soit sommets d'entrée soit sommets de sortie de G(R).

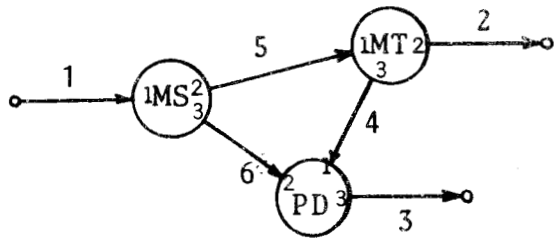
Le fonctionnement du réseau R est déterminé par le fonctionnement de chaque module de contrôle du réseau.

Le réseau sera dit "inutilisé" si chaque module de contrôle est à l'état initial ; le réseau sera dit "en attente" si chaque module de contrôle est dans un état d'attente.

Nous dirons qu'un réseau R est bien construit si, à partir de l'état initial, il est impossible que le réseau réalise simultanément les conditions suivantes :

- tous les arcs entrant de R sont *activés*,
- tous les arcs sortant de R sont *désactivés*,
- tous les modules de R sont en attente.

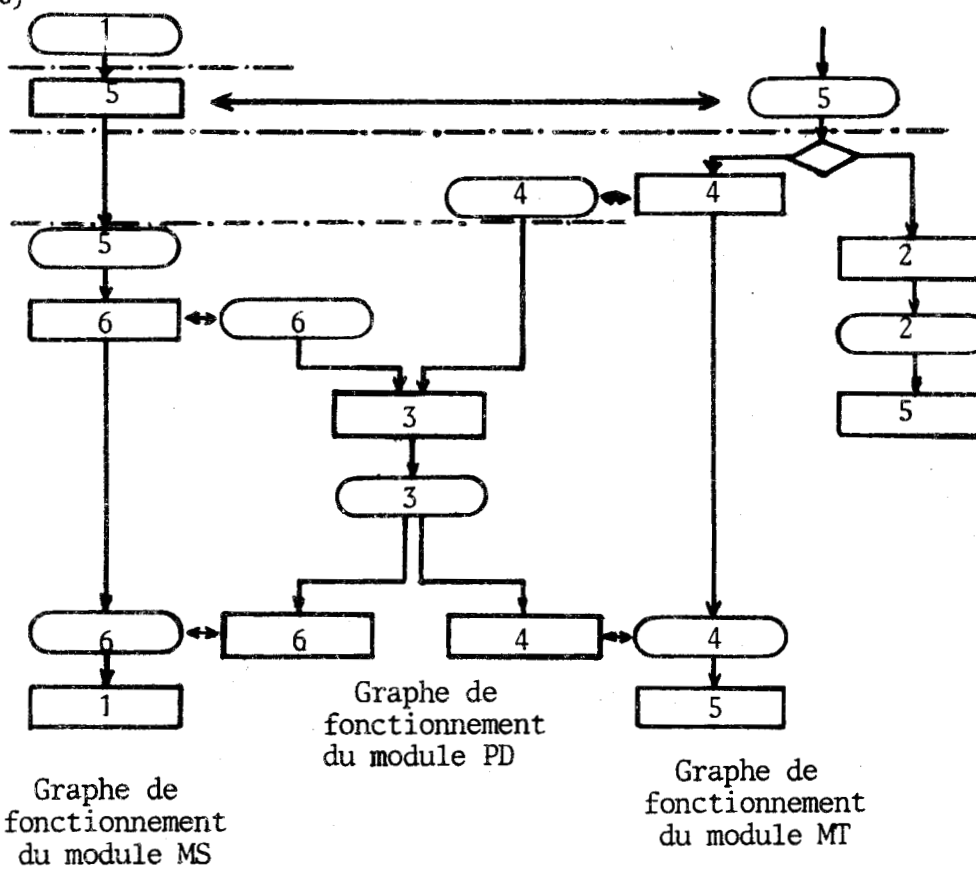
Exemple :



Les chiffres dans les sommets permettent de se référer aux graphes de définition des modules utilisés.

Comparons les graphes de fonctionnement de chaque module

v(1)	v(2)	v(3)	v(4)	v(5)	v(6)
1	0	0	0	0	0
1	0	0	0	1	0
1	0	0	1	1	0



Graphe de fonctionnement du module MS

Graphe de fonctionnement du module PD

Graphe de fonctionnement du module MT

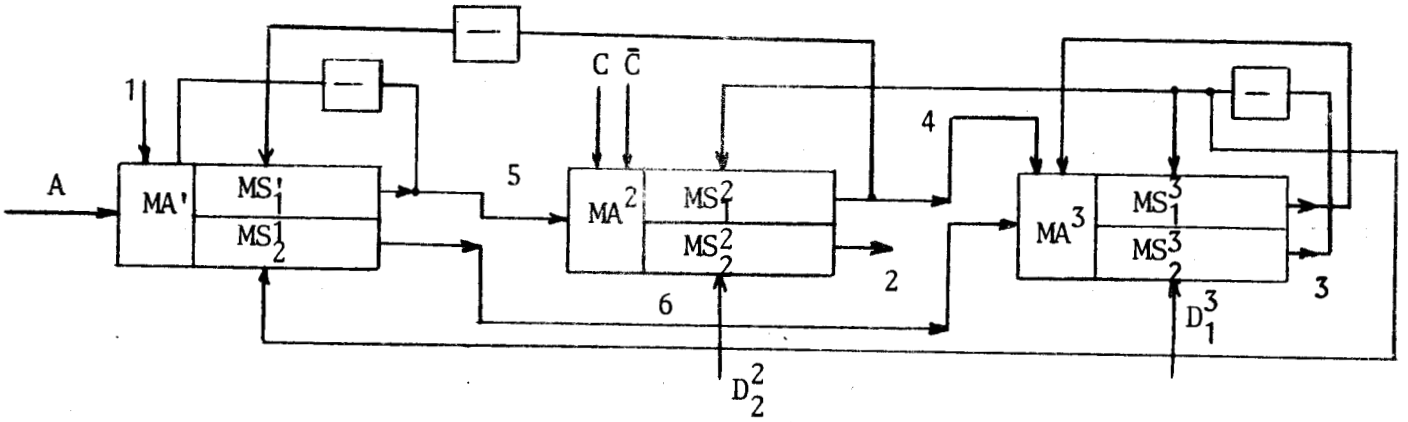
Le système ne peut plus fonctionner à partir du moment où 100110 est atteint, parce que v(5) ne passe à zéro qu'après que v(4) soit retombé à zéro.

Le réseau est mal formé. Nous pouvons vérifier que :

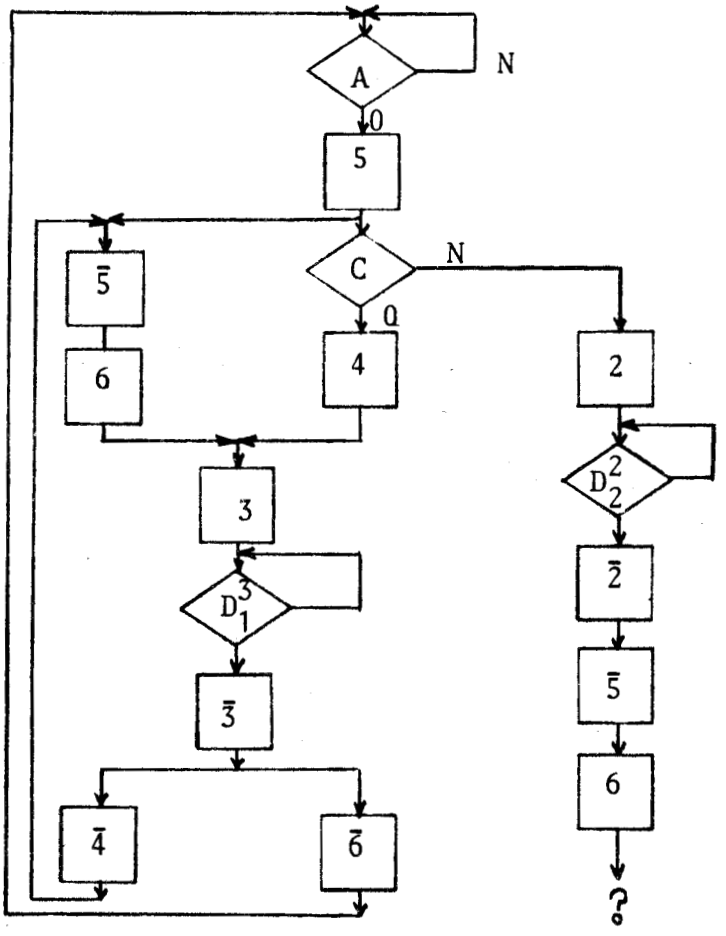
- tous les arcs entrant de R sont activés (v(1)=1)
- tous les arcs sortant de R sont désactivés (v(2)=v(3)=0)
- tous les modules de R sont en attente

MS attente sur 5
PD attente sur 6
MT attente sur 4.

Ce réseau de modules de contrôle sera câblé comme suit :



Nous pouvons faire correspondre à ce réseau l'organigramme de commande suivant :

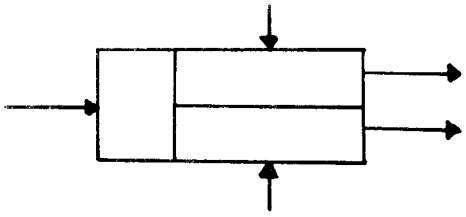


En appliquant à cet organigramme les méthodes de contrôle définies aux chapitres précédents, nous pouvons vérifier qu'un tel fonctionnement est inacceptable. Par la méthode de construction du graphe d'état, nous vérifierions de la même façon que quel que soit le fonctionnement nous aboutissons systématiquement à un ensemble incompatible.

On pourrait aisément démontrer qu'un réseau de contrôle R connexe est bien construit si et seulement si G(R) ne contient pas de cycles.

Si nous pouvons construire directement le réseau de contrôle d'un processus asynchrone à partir de l'organigramme de commande, il sera alors possible de contrôler et de réaliser immédiatement le bloc de commande sans avoir à construire l'organigramme de processus ni le graphe d'état.

L'exemple ci-dessus nous a permis de montrer qu'il existe des *relations entre enchaînements et module de contrôle*. D'une façon systématique nous pouvons définir les enchaînements correspondant à chaque fonction de base. Par exemple pour la fonction SD nous avons



Pour cette fonction :

S_1 et S_2 correspondent à deux variables de sortie booléenne et
A, D_1, D_2 - trois variables d'entrée booléenne,

et il est possible de construire les enchaînements suivants :

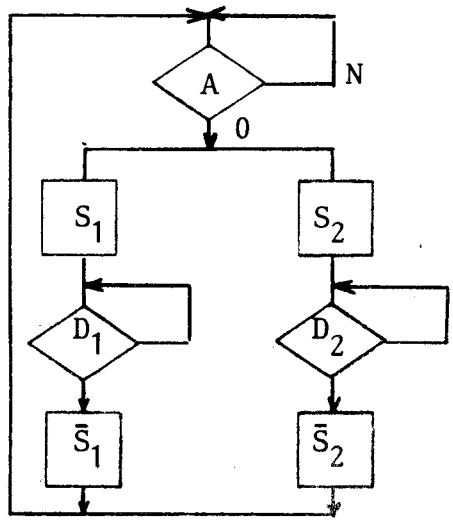
$((A=1), P, (S_1=1))$; $((A=1), P, (S_2=1))$

$((A=1), N, (A=1))$

$((S_1=1), (D_1=1))$; $((D_1=1), P, (S_1=0))$
$((D_1=1), N, (D_1=1))$

$((S_2=1), (D_2=1))$; $((D_2=1), P, (S_2=0))$
$((D_2=1), N, (D_1=1))$

D'où l'organigramme de commande suivant :



Ces modules constituent donc un moyen de réaliser le bloc de commande d'un processus asynchrone sans avoir besoin de construire ni de réduire les équations booléennes correspondant au fonctionnement de ce processus.

Dans le cas des *systèmes synchrones* il est inutile de calculer puis de réduire les équations booléennes correspondant au fonctionnement du processus ; si nous nous rapportons à la définition du graphe d'état d'un processus synchrone, il apparaît qu'il s'agit de déterminer le chemin minimal de ce graphe auquel doit correspondre une implantation du bloc de commande du type bloc microprogrammé. Des essais de réalisation d'une implantation microprogrammée à partir d'une description du processus à contrôler ont déjà été réalisés.

Enfin, en ce qui concerne la réalisation des systèmes programmés qui sont synchrones, l'objectif est la réalisation d'un bloc de commande microprogrammé. Cependant avant de définir son implantation, il faut résoudre le problème de l'optimisation du bloc de commande.

ooOoo

C O N C L U S I O N

Si la notion de processus a été introduite au début de cette thèse, c'était essentiellement en vue de son utilisation pour définir des automatismes séquentiels ; cependant cette notion est applicable à d'autres domaines, notamment à la microprogrammation ; d'autre part comme cela a été indiqué au chapitre II, il est possible de généraliser la notion de processus et de définir des processus de processus.

Cette généralisation conduit naturellement à étendre des langages du type DESPROG et DESAUCYK de manière à pouvoir décrire des systèmes de plus en plus complexes. Cette extension "horizontale" du domaine descriptible à partir des processus implique une extension "verticale" vers une automatisation plus complète et plus étendue des problèmes de contrôle et de réalisation.

Notamment, dans ce domaine deux voies de développement sont envisageables :

- développement des possibilités de contrôle à l'aide des graphes d'état, de la logique formelle et des propriétés des modules de contrôle,

- développement des possibilités d'optimisation et de réalisation de blocs de commande microprogrammés.

BIBLIOGRAPHIE

1 Le nouveau langage scientifique Algol . HERMANN - PARIS.

2 *Control Units for Sequencing complex Asynchronous Operations.* I.E.E.E.
A. GRASSELLI - Août 1962. p.483-493.

3 *On Simulating Networks of Parallel Processes in which Simultaneous Events May Occur.* D.L.PARNAL - Communication of the ACM - Sept. 1969. p. 519-531.

4 *Graphes contrôlés et instructions.* G. BOULAYE - RIRO. B-2 1971. p.43-60.

5 *A New Method of Checking the Concistency of Precedence Matrices.*
R.B. MARIMONT - JACM - Janv. 1959. p.164-171.

6 *On the Concistency of Precedence Matrices.*
F. HARARY - JACM - 1960. p.255-259.

7 *Optimal Sequential Partitions of Graphs.*
B.W. KERNIGHAN - JACM - 1971. p.34-40.

8 *Mille et un algorithmes pour les problèmes de cheminement dans les graphes.*
C. PAIR - RIRO - B-3 1970. p.125-143.

9 *Introduction à la notion de réceptivité dans l'étude des automatismes séquentiels.*
M.P. GIRARD - Document Le Carbonne - Lorraine.

10 *Méthode pratique d'analyse d'un cahier des charges et d'organisation des programmes concernant un automatisme industriel programmé.*
D. DURRANDE - Document Alsthom

11 *Modélisation d'un automate central.*
Document Télémécanique Electrique.

12 *Programme d'aide à la conception des microprogrammes.*
Document Inter-Technique.

13 *The Application to formal logic to programs and programming.*
C.D. ALLEN - I.B.M. Syst. Jour. N°1 - 1971 - p.2-38.

ANNEXE DU CHAPITRE IV

Liste des éléments du langage

1. Liste des éléments terminaux :

Caractères utilisables

A, B, ..., W 0, 1, 2, ..., 9 ; / , . ( ) + - × / ¬ ∨ ∧ = > <	Termes utilisables

Total : 49

2. Termes imposés

DG : indicateur de début de description d'un processus
AS, SY : types de processus
DPA : indicateur de début de définition et description de processus autonomes
DES : - - - - description d'un processus autonome
DEP : - - - - - - - irréductible
DEG : - - - - - du fonctionnement global
FDG : - - fin de description du fonctionnement global
FDB : - - - - - de base
R : - - rappel de déclaration
DCC : - - déclaration d'éléments commandés
SE, DE, IP, RB, RR : types d'éléments commandés
OC : indicateur de moyen d'exécution commandé
DCE : indicateur de déclaration de variables d'entrée
CF, CD, VC, PA : types de variables d'entrée
PU, SC, AP, EX : origines de variables d'entrée
A : utilisé pour décrire les plages de valeur
CM : indicateur de commandes modifiantes
VI : indicateur de variables d'entrée incompatibles

DCP : indicateur de déclaration de processus secondaire
PC, PA : types de processus secondaires
DCT : indicateur de déclaration de processus type
ICE, ICC : indicateurs de conditions initiales
DSE : indicateur de description des enchaînements

PUIS, ET, SI, DES QUE, DEBUT, FIN, ENCL, DECL, EMET, TEST, EXEC, REP apparaissent dans les phrases :

ICI : indicateur de commandes incompatibles
STOP, MOD : types d'incompatibilités
DVS : indicateur de déclaration de variables de sécurité
IPU : - définition de processus d'urgence
ICM : - - commandes manuelles
VAM, RAZ, MCM : types de commandes manuelles
IVM : indicateur de variables modifiées par des commandes de processus
IVR : - verrouillages

Total : 59

3. Liste des éléments décomposables

N°	Définitions
Eléments apparaissant dans la structure générale	
1	Description d'un processus
2	Définition d'un processus
3	Définition et description des processus autonomes
4	Description du fonctionnement global
5	Type P
6	Nom de processus
7	Durée de la période
8	Description d'un processus autonome
9	Définition et description de processus irréductibles
10	Description de base
11	Définition d'un processus irréductible
12	Description d'un processus irréductible
13	Description des relations entre processus
14	Description des enchaînements de processus
15	Indications supplémentaires

16 Indication des processus incompatibles
17 - des variables d'entrée incompatibles
18 - - - modifiées
19 - des verrouillages

Eléments apparaissant dans une description de base

20 Déclarations de variables
21 Indication des conditions initiales
22 Description des enchaînements
23 Indication complémentaire

Eléments apparaissant dans les déclarations de variables

24 Déclarations d'éléments commandés
25 Déclarations de variables d'entrée
26 Déclarations de processus secondaires
27 Déclarations de processus types
28 Indicateur d'élément commandé
29 Déclaration d'un élément commandé
30 Type C
31 Type S
32 Moyen d'exécution commandé
33 Elément sélectionnable
34 Type M
35 Indicateur de variable d'entrée
36 Déclaration d'une variable d'entrée
37 Rôle et Origine
38 Plagide
39 Comovide
40 Cadenceur
41 Origine
42 Emetteur
43 Plage de valeurs
44 Groupe de commandes modifiantes
45 Valeur
46 Commande modifiante
47 Indication sémantique sur les variables d'entrée
48 Variable incompatible

49 Indicateur de processus secondaire
50 Déclaration d'un processus secondaire
51 Type F
52 Type cyclique
53 Type acyclique
54 Conception
55 Description de base
56 Groupe de variables spécifiques
57 Variable spécifique
58 Indicateur de processus type
59 Déclaration d'un processus type
60 Type T
61 Vavide
62 Variable

Elements apparaissant dans les indications des conditions initiales

63 Conditions initiales sur les variables d'entrée
64 Conditions initiales sur les éléments commandés
65 Initialisation
66 Indicateur
67 Terme

Elements apparaissant dans les descriptions d'enchaînements

68 Phrase
69 Phrase simple
70 Phrase linéaire
71 Phrase de repositionnement
72 Ordre multiple
73 Phrase commande
74 Indice
75 Ordre
76 Commande
77 Opération interne
78 Phrase test
79 Phrase test d'attente
80 Expression booléenne
81 Repositionnement multiple
82 Repositionnement
83 Indication de repositionnement

84 Entrée
85 Chargement de registre
86 Opération booléenne
87 Opération arithmétique
88 Registre
89 Terme booléen
90 Opérateur booléen
91 Primaire booléen
92 Variable d'entrée booléenne
93 Registre booléen
94 Terme arithmétique
95 Opérateur arithmétique
96 Opérateur additif
97 Opérateur multiplicatif
98 Variable d'entrée rationnelle
99 Mémoire rationnelle
100 Expression arithmétique
101 Comparateur

Eléments intervenant dans les descriptions des indications complémentaires

102 Indication des groupes de commandes incompatibles
103 Déclarations de variables de sécurité
104 Indication de processus d'urgence
105 Indication de commandes manuelles
106 Groupe de commandes incompatibles
107 Type I
108 Indicateur de variable de sécurité
109 Déclaration d'une variable de sécurité
110 Nature
111 Type P
112 Test ponctuel
113 Point testé
114 Déclaration d'une variable de sécurité déjà définie
115 Indication d'un processus d'urgence
116 Déclaration générale d'une variable d'intervention
117 Déclaration d'une variable d'intervention
118 Déclaration d'une variable d'intervention déjà définie
119 Déclaration générale d'un processus d'urgence
120 Déclaration d'un processus d'urgence

121 Indication d'une commande manuelle

122 Mention

Eléments intervenant dans la description du fonctionnement global

123 Groupe de processus incompatibles

124 Verrouillage

125 Groupe de commandes verrouillées

ELéments de description divers

126 Nom

127 Lettre

128 Caractère

129 Commentaires

130 Nombre

131 Nombre entier

132 Nombre décimal

133 Puissance de dix

134 Chiffre

135 Code

136 Signe

ANNEXE DU CHAPITRE VII

Eléments du langage DESAUCYK

1. Eléments terminaux :

Caractères utilisables

0, 1, 2, ..., 9
A, B, C, ..., Z
; L , ( ) :
+ - × / ¬ ∧ ∨
= ≠ > < ≥ ≤

Termes imposés :

DM : indicateur de définition d'une machine
IEV : indicateur d'indication d'environnement
DSM : - de définition et description de sous-machines
DES : - description d'une sous-machine
DMF : - - d'un mode de fonctionnement
DEG : - - du fonctionnement global
FDG : - fin de description du fonctionnement global
FDE : - - - élémentaire
DCC : - de déclaration d'élément commandé
EM, EB, EBM, MFS, RB, RR : types d'éléments commandés
DCT : indicateur de déclaration d'éléments testables
PR, PB, CE : types d'éléments testables
DCF : indicateur de déclaration des fonctions temps
T, M, H : types de fonctions temps
DCG : indicateur de déclaration de positionnement géométrique
OX, OY, OZ : types d'axes
SUR, R, AXE, DE, POUR : indicateurs divers de positionnement axial
ICI : indicateur de conditions initiales

DEBUT, FIN, DE, ALLER A, ET SI ALORS, ENSUITE, DANS, ET, OU, SUR,
NON SUR, A, TANT QUE, JUSQU'A, SI, DES QUE, FC :
apparaissent dans un texte
DEMARRER, ARRETER, AMENER, DEPLACEMENT AVANT,
RANGER, EXECUTER etc... : caractérisent les phrases commande

DVS : indicateur de déclaration de variable de sécurité
R : - - - - rappelé
P : - positionnement de test
DPU : - définition de processus d'urgence
ICM : indication de commande manuelle
VAM, RAZ, MCM : types de commandes manuelles
IMI : indication des modes de fonctionnement incompatibles
IVR : - de verrouillages.

2. Liste des termes décomposables :

N°	Dénominations
1	Description d'un automatisme
2	Définition d'une machine
3	Définition et description des sous-machines
4	Description du fonctionnement global de la machine
5	Commentaires
6	Nom d'une machine
7	Numéro de secteur d'activité
8	Indications d'environnement
9	Nom
10	Paramètre d'environnement
11	Numéro du paramètre
12	Nombre entier
13	Numéro de genre de fonctionnement
14	Définition et description de modes de fonctionnement principaux
15	Définition d'une sous-machine
16	Description d'une sous-machine
17	Description élémentaire
18	Définition d'un mode de fonctionnement
19	Description d'un mode de fonctionnement
20	Description des commandes de mode de fonctionnement

21 Indication supplémentaire
22 Indication des modes de fonctionnement incompatibles
23 Indication de verrouillages
24 Déclarations d'éléments constitutifs
25 Description du fonctionnement
26 Indications complémentaires
27 Déclarations d'éléments commandés
28 Déclarations d'éléments testables
29 - de fonctions temps
30 - de positionnements
31 Indicateur d'élément commandé
32 Déclaration d'un élément commandé
33 Type C
34 Elément sélectionnable
35 Nombre
36 Type M
37 Mesure
38 Indicateur d'élément testable
39 Déclaration d'un élément testable
40 Type T
41 Plagide
42 Indicateur de fonction-temps
43 Déclaration d'une fonction-temps
44 Type F
45 Vadevide
46 Indicateur de positionnement
47 Déclaration d'un positionnement
48 - - axial
49 - - ponctuel
50 - - sélectionné
51 Type A
52 Coordonnées
53 Positions
54 Position repérée
55 Vitesse dans le sens positif
56 - - - négatif
57 Sélection
58 Conditions initiales
59 Texte

60 Phrase
61 Dernière phrase
62 Fonctionnement cadencé
63 Phrase étiquetée
64 Titre
65 Phrase commande multiple
66 - test multiple
67 Titre général
68 Phrase commande
69 Phrase commande liée
70 Tivide
71 Phrase test
72 Phrase test liée
73 Phrase finale
74 Tifin
75 Boucle
76 Phrase commande de mouvement
77 - - d'élément bistable ou assimilé
78 - - de modes secondaires
79 - - de registres
80 Mouvement
81 Type du mouvement
82 Précisions
83 Position d'arrêt
83b Arrêt
84 Vitesse
85 Déplacement
86 Indication de Mouvement
87 Nom de l'élément mobile
88 Axe
89 Axe de rotation
90 Point fixe
91 Dénomination
92 Indication de type
93 Affectations
94 Correspondance
95 Nom de la variable utilisée la première fois
96 Nom de la variable remplaçante
97 Type de commande
98 Expression

99 Registre
100 Expression booléenne
101 Expression arithmétique
102 Phrase-test unique
103 - - répétitif
104 - - pseudo-répétitif
105 Indicateur de test
106 Phrase état
107 Phrase état liée
108 Etat
109 Indication de position
110 Facteur booléen
111 Terme booléen
112 secondaire
113 Primaire
114 Variable booléenne
115 Relation
116 Registre booléen
117 Elément testable booléen
118 Fonction booléenne
119 Comparateur
120 Terme arithmétique
121 Signe
122 Opérateur additif
123 Facteur arithmétique
124 Opérateur multiplicatif
125 Variable arithmétique
126 Registre rationnel
127 Déclaration de variable de sécurité
128 Définition de processus d'urgence
129 Indication de commandes manuelles
130 Indicateur de variable de sécurité
131 Déclaration d'une variable de sécurité
132 Nom de la variable de sécurité
133 Nom de mode de fonctionnement
134 Description
135 Positionnement des tests
136 Définition d'un processus d'urgence

137 Déclaration de la variable aléatoire
138 Nom de la variable aléatoire
139 Déclaration du mode de fonctionnement aléatoire
140 Nom du mode de fonctionnement aléatoire
141 Indication d'un élément commandé manuellement
142 Mention
143 Indication d'un ensemble de modes incompatibles
144 Verrouillage
145 Nom du mode de fonctionnement actif
146 - - - bloqué