

85 | 858

Université de Nancy 1

UER de Mathématiques

Sc N 85 /
279 A

**LA COMMUNICATION : UN OUTIL
POUR LA SPECIFICATION, LA CONSTRUCTION
ET LA VERIFICATION DE SYSTEMES PARALLELES**

=====

Thèse présentée à l'Université de Nancy 1
pour l'obtention du grade de

**DOCTEUR ES SCIENCES MATHÉMATIQUES
(Mention : Informatique)**

par

Guy-René Perrin

=====

Thèse soutenue le 3 octobre 1985 devant la Commission
d'Examen composée de Messieurs:

Jean-Pierre Finance, Université de Nancy 1 (Président)

Michel Sintzoff, Université Catholique de Louvain

Jean-Pierre Verjus, Université de Rennes (Rapporteurs)

Jean-Claude Boussard, Université de Nice

Jean-Claude Derniame, Université de Nancy 1

Gérard Roucairol, Société Bull

Michel Tréhel, Université de Besançon (Examineurs)



**LA COMMUNICATION : UN OUTIL
POUR LA SPECIFICATION, LA CONSTRUCTION
ET LA VERIFICATION DE SYSTEMES PARALLELES**

=====

Thèse présentée à l'Université de Nancy 1
pour l'obtention du grade de

**DOCTEUR ES SCIENCES MATHÉMATIQUES
(Mention : Informatique)**

par

Guy-René Perrin

=====

Thèse soutenue le 3 octobre 1985 devant la Commission
d'Examen composée de Messieurs:

Jean-Pierre Finance, Université de Nancy 1 (Président)

Michel Sintzoff, Université Catholique de Louvain

Jean-Pierre Verjus, Université de Rennes (Rapporteurs)

Jean-Claude Boussard, Université de Nice

Jean-Claude Derniame, Université de Nancy 1

Gérard Roucairol, Société Bull

Michel Tréhel, Université de Besançon (Examineurs)

Il m'est très agréable de remercier amicalement Jean-Pierre Finance, pour le temps qu'il m'a consacré et l'orientation qu'il a donnée à ce travail, m'amenant à l'utilisation d'outils formels pour une réflexion sur l'aspect méthode de la programmation. En présidant ce jury, il témoigne du travail d'équipe que nous avons entrepris, et qui doit se poursuivre.

Michel Sintzoff a accepté d'être rapporteur de cette thèse: je l'en remercie. Il a été un lecteur critique, qui a, bien entendu, su mettre l'accent, avec humour, sur les points clés qui nous préoccupent à propos de la construction des programmes. Je suis sûr que notre discussion n'est pas close.

J'ai souhaité que Jean-Pierre Verjus soit rapporteur de ce mémoire: la connaissance qu'il a des thèmes du parallélisme et de la répartition le désignait à l'évidence pour évaluer l'apport et l'originalité de mon travail. Ses questions m'ont obligé à clarifier mes idées.

Jean-Claude Boussard est dans ce jury, et j'en suis très heureux: il m'a suivi, aidé et encouragé depuis ma thèse de troisième cycle. Lui aussi a contribué à éveiller mon intérêt pour tenter de comprendre et formuler les raisonnements de la programmation. Sa présence ici est pour moi l'occasion de lui dire mon amitié.

Merci à Jean-Claude Derniame, avec qui il m'a été très agréable de collaborer, dès mon arrivée au CRIN: mon travail sur l'expression des communications et son intérêt pour la conception d'applications réparties ont tout naturellement conduit à notre rencontre. Elle a été pour moi très enrichissante.

J'ai souhaité la présence de Gérard Roucairol dans ce jury. Je le remercie d'avoir accepté, ainsi que de l'intérêt qu'il a toujours témoigné pour mon travail dans ce domaine du parallélisme, dont il est un spécialiste.

Je remercie Michel Tréhel d'avoir bien voulu être, en quelque sorte, le "représentant de Besançon" dans ce jury. Il témoigne ainsi des travaux passés, présents et à venir, développés chez nous, sur les algorithmes parallèles et distribués.

Quelques mots encore pour remercier très amicalement

- les membres de l'équipe Comète, et plus particulièrement Jacques Julliard et Catherine Mongenet, dont j'apprécie le travail et l'amitié,
- les membres du groupe Annagram: quelques uns sont dans ce jury. Tous verront combien les réflexions développées dans ce mémoire doivent à nos discussions sur la construction des programmes.

Enfin, je veux dire à Claude Pair combien je lui suis reconnaissant pour les encouragements et les conseils qu'il m'a toujours prodigués. C'est à lui que je dois ma présence au CRIN, et par conséquent les résultats qui suivent.

TABLE DES MATIERES

INTRODUCTION

CHAPITRE 1- SYNCHRONISATION ET COMMUNICATION	100
1-1- OBSERVATION DU PARALLELISME	102
1-2- INTERPRETATION DU PARALLELISME	103
1-2-1- Modèles du non-déterminisme	104
1-2-2- Modèles du parallélisme asynchrone	105
1-2-3- Modèles du parallélisme synchrone	109
1-3- EXPRESSIONS DU PARALLELISME	111
1-3-1- Expression implicite des synchronisations	111
1-3-2- Expression explicite des synchronisations	112
1-3-3- Les langages à "rendez-vous"	115
1-3-4- Expression de la concurrence	117
1-4- NOTION DE COMMUNICATION	118
1-4-1- Relation de communication	118
1-4-2- Exemples	120
1-4-3- Caractéristique d'une communication	122
CHAPITRE 2- DEFINITION D'UNE COMMUNICATION ET EXPRESSION DE SYSTEME PARALLELES	200
2-1- TYPE DE COMMUNICATION	202
2-1-1- Définition	203
2-1-2- Le type "ensemble-indexé"	206
2-1-3- Exemples de types de communication	211
2-2- EXPRESSION D'UN SYSTEME PARALLELE	219
2-2-1- Définition sommaire du langage	219
2-2-2- Hiérarchisation de processus	222
2-2-3- Sémantique intuitive d'un système parallèle	228

CHAPITRE 3- EXEMPLES DE SYSTEMES PARALLELES	300
3-1- EXEMPLE 1 : CALCUL DE FACTORIELLE	300
3-2- EXEMPLE 2 : CALCUL DU P.G.C.D. DE N NOMBRES	304
3-2-1- Décomposition en facteurs premiers	304
3-2-2- Algorithme d'Euclide	307
3-3- EXEMPLE 3 : CALCUL DES COEFFICIENTS DU BINOME	310
3-4- EXEMPLE 4 : ALGORITHMES ITERATIFS EN CALCULS SCIENTIFIQUES	313
3-4-1- Calcul d'un point fixe	314
3-4-2- Recherche d'un zéro d'une fonction	317
3-5- EXEMPLE 5 : PLUS COURT CHEMIN DANS UN GRAPHE	321
CHAPITRE 4- SEMANTIQUE D'UN SYSTEME PARALLELE	400
4-1- SEMANTIQUE OPERATIONNELLE D'UN SYSTEME PARALLELE	401
4-1-1- Sémantique d'un processus séquentiel	401
4-1-2- Produit de processus	403
4-1-3- Opérateur de communication	407
4-2- INTERPRETATION D'UN TYPE DE COMMUNICATION	410
4-2-1- Notion de suite temporelle	410
4-2-2- Domaine d'interprétation d'un type de communication	412
4-2-3- Interprétation d'un type de communication	416
CHAPITRE 5- PROPRIETES D'UNE COMMUNICATION	500
5-1- AUTOMATE ASSOCIE	500
5-1-1- Définition	500
5-1-2- Exemples	503
5-1-3- Ordres de communications	508
5-1-4- Application aux modifications de programmes	511

5-2- GRAPHE D'IMPLANTATION	513
5-2-1- Définition	514
5-2-2- Protocole et priorité	515
5-2-3- Equité et non blocage	517
5-3- IMPLANTATION D'UNE COMMUNICATION	518
5-3-1- Noeuds d'observation	518
5-3-2- Théorème	521
CHAPITRE 6- VERIFICATION D'UN SYSTEME PARALLELE	600
6-1- RAPPELS BIBLIOGRAPHIQUES	601
6-1-1- Introduction	601
6-1-2- Extension de la "logique de Hoare" à des systèmes avec données partagées	603
6-1-3- Coopération de preuves	604
6-1-4- Logiques de type "temporelles"	609
6-2- SERIALISATION D'UN SYSTEME PARALLELE	614
6-2-1- Trace d'exécution	614
6-2-2- Assertions d'une unité d'exécution	618
6-2-3- Sérialisation d'un ensemble d'unités d'exécution	620
6-3- SYSTEME DE PREUVE	623
6-3-1- Axiomes et règles d'inférence	623
6-3-2- Propriétés d'un système parallèle	625
6-3-3- Consistence et complétude du système de preuve	627
6-4- EXEMPLES ET PROPRIETES	630
6-4-1- Calcul de factorielle	630
6-4-2- Calcul des coefficients du binôme	636
CHAPITRE 7- LA COMMUNICATION DANS LA CONCEPTION DE PROGRAMMES	700
7-1- SPECIFICATION ET CONSTRUCTION DE PROGRAMMES	701
7-1-1- Spécification des propriétés d'un système	701
7-1-2- Synthèse de processus communicants	704
7-1-3- Construction de programmes	708

7-2- EXEMPLE 1 : RESOLUTION D'UN SYSTEME D'EQUATIONS AUX DERIVEES PARTIELLES	711
7-2-1- Définition du problème	711
7-2-2- Différentes solutions	713
7-3- EXEMPLE 2 : LOUEUR DE BATEAUX	717
7-3-1- Spécification du problème	717
7-3-2- Définition de communications	720
7-3-3- Représentations du système obtenu	722
7-4- EXEMPLE 3 : CONTROLEUR D'UNITE DE DISQUE	727
7-4-1- Présentation informelle	727
7-4-2- Spécification fonctionnelle	727
7-4-3- Construction d'une solution	734
CONCLUSION	800
BIBLIOGRAPHIE	900

INTRODUCTION

Le développement de systèmes distribués ou de multiprocesseurs justifie que l'on s'intéresse à la programmation de tels systèmes.

Les systèmes distribués mettent en évidence un parallélisme dit "de situation", c'est-à-dire dans lequel des événements indépendants sont susceptibles de se produire "simultanément".

Au contraire, la programmation de systèmes multiprocesseurs intègre un parallélisme dit "de résolution", pour lequel la définition de "processus parallèles" et leurs interactions peut être, en partie, laissée à l'appréciation du concepteur selon les objectifs visés par la solution qu'il met en œuvre.

En fait, il n'y a pas de véritable opposition entre ces deux situations, qui peuvent en outre être complémentaires: dans le premier cas le concepteur peut avoir une certaine latitude quant aux choix d'implantation ou aux contraintes finalement induites par la solution retenue. Inversement, dans le second cas le concepteur peut être contraint par des caractéristiques d'architecture ou de système.

L'objectif de cette thèse est de proposer un cadre d'expression formelle induisant un mode de raisonnement pour la conception de systèmes parallèles "grandeur nature", c'est-à-dire dépassant le cadre d'exercices jouets.

Pour atteindre cet objectif, nous développons une seule idée simple: comme la conception d'algorithmes séquentiels nous l'a appris, une démarche méthodique suppose qu'on passe progressivement du problème posé à un programme solution en étant

- . le plus "statique" possible,
- . le plus "modulaire" possible,
- . le plus "descendant" possible,
- . indépendant, le plus longtemps possible, du système informatique "cible".

Tous ces "bémols" mis à ces caractéristiques méthodologiques le sont par humilité face à la conception d'algorithmes "réels": nous constaterons, dans "l'univers du parallélisme" aussi, que l'humilité doit être de rigueur pour proposer, non point une méthode de conception de systèmes parallèles "réels", mais seulement une contribution à l'étude d'un outil dont on montrera qu'il est certainement adapté à un raisonnement méthodique.

Ce point pourra paraître bien peu formel: pourtant cette thèse définit et utilise des concepts formels, dans la mesure où ces formalismes sont nécessaires à la compréhension, l'utilisation et l'implémentation des notions utilisées.

Dans "l'univers parallèle" que nous étudions ici, les critères énoncés plus haut signifient: concevoir un système parallèle comme un ensemble d'entités le plus indépendantes possibles, puis progressivement, préciser ces entités et leurs relations jusqu'à leur expression dans le système cible visé.

Nous mettons en particulier l'accent sur l'expression

des relations entre ces entités, qui sont définies comme suit: si deux entités "P" et "C" d'un système parallèle sont en relation c'est qu'au moins l'une -disons "C"- utilise au moins une donnée -disons "a"- manipulée par l'autre -donc "P".

Si on note

- . $(p_1, p_2, \dots, p_i, \dots)$ la suite des valeurs de "a" déterminée par "P",
- . $(c_1, c_2, \dots, c_j, \dots)$ la suite des valeurs de "a" utilisées par "C",

la relation entre ces entités est exprimée par la relation entre ces deux suites, c'est-à-dire la définition de chaque terme "c_j" en fonction des termes de la suite $(p_1, p_2, \dots, p_i, \dots)$: nous appelons communication cette relation.

L'expression des "communications" est donc l'outil que nous proposons pour induire un mode de raisonnement pour la conception des systèmes parallèles. Cette thèse est consacrée

- à l'étude de cet outil:

- . expression formelle et sémantique des systèmes,
- . définition d'un modèle et interprétation,
- . propriétés des relations de communication,
- . système formel de preuves,

- et à sa validation comme outil méthodologique:

- . exemples d'applications,
- . transformations de solutions,
- . développement de solutions pour des problèmes réels.

Le chapitre 1 est une introduction plus précise de cette notion de communication qu'on oppose aux approches classiques dans la littérature. Celles-ci reposent essentiellement sur le concept de synchronisation, soit par la définition explicite de relations entre des occurrences d'opérations, soit par le respect de certaines contraintes

d'accès à des ressources communes.

Ces approches ne traitent pas explicitement le problème des valeurs des données échangées ou partagées: tout au plus les langages à "rendez-vous", directement issus de la philosophie des processus concurrents, lient-ils échange de valeur et synchronisation.

Ce point de vue a les mêmes caractéristiques désastreuses, sur la conception, la lisibilité et la vérification des programmes, que l'utilisation de "GOTOs" dans un programme séquentiel. Au contraire, la notion de "communication", telle que nous la présentons, augure d'une approche statique permettant une démarche modulaire et descendante.

Dans le chapitre 2 nous proposons un cadre formel pour l'expression des relations de communication: à chaque "donnée communiquée" est associé un type de communication, structure abstraite définie algébriquement, caractérisant les opérations de communication autorisées sur cette donnée. Cette notion est illustrée par un ensemble d'exemples réutilisés dans la suite.

Cette expression formelle des communications est intégrée dans un langage d'expression des systèmes parallèles en termes de processus communicants.

Le chapitre 3 présente une série d'exemples de systèmes parallèles pour illustrer les notions précédentes: ces exemples, de taille modeste, vont de problèmes "jouets" à des algorithmes dont l'intérêt pratique réel est discuté dans la littérature référencée. Ils sont volontairement simples, mais représentatifs de classes de problèmes différentes et de solutions algorithmiques pertinentes.

Au chapitre 4, une sémantique du langage d'expression des systèmes est donnée sous forme opérationnelle: le co-domaine de cette fonction sémantique est l'univers des systèmes non déterministes, décrits comme des systèmes de transitions. Les processus sont composés en algèbre par deux opérateurs:

- l'opérateur "produit" définit l'entrelacement ou la

simultanéité des opérations de chaque processus,

- l'opérateur "communication" définit les transitions associées aux opérations de communication.

Puis nous précisons, par une sémantique fonctionnelle déterministe, la notion de type de communication dans l'univers des "suites temporelles". Cette interprétation permet d'exprimer clairement les éventuelles situations "d'attente" des processus pour réaliser leurs opérations de communication. En outre le domaine d'interprétation choisi permettra d'entrevoir, au chapitre 7, une méthodologie de construction d'une solution en termes de processus communicants à partir d'une spécification de problème en termes de suites temporelles.

Le chapitre 5 définit quelques propriétés induites par un type de communication. Ces propriétés concernent

- d'une part l'implantation des types de communication,
- d'autre part la définition de relations d'ordre entre ces types de communication.

Ces relations, définissant un treillis, permettent de rendre compte de propriétés "d'asynchronisme" des systèmes: elles sont applicables à des modifications et des preuves de programmes parallèles.

Le chapitre 6 traite de la vérification des systèmes parallèles. Nous rappelons d'abord les différentes approches présentées dans la littérature: extension de la "logique de Hoare" ou utilisation de la logique temporelle.

Nous proposons ensuite un système formel de preuves pour le langage défini au chapitre 2 et dont la sémantique est définie au chapitre 4. Ce système est illustré par quelques exemples déjà traités au chapitre 3.

Enfin nous abordons dans le chapitre 7 le problème de la conception de systèmes parallèles et nous observons que l'introduction de la notion de communication, telle qu'elle est définie dans cette thèse, permet une démarche méthodique et modulaire. Dans un premier temps, nous rappelons quelques propositions en matière de spécification et construction de

programmes parallèles. Ensuite, nous présentons trois exemples illustrant les points suivants:

- différents choix de types de communication, de plus en plus "asynchrones", comme outils de modification de programmes,
- implantations de types de communication, par des types plus "synchrones", comme outils de construction de programmes -y compris de programmes séquentiels,
- définition d'un type de communication -à partir de propriétés temporelles- comme outil de spécification d'un problème.

CHAPITRE 1- SYNCHRONISATION ET COMMUNICATION

Avant une première présentation de la notion de communication, nous rappelons dans ce chapitre les trois "vues" du parallélisme, telles qu'elles sont traitées dans la littérature:

- un aspect qualitatif, intuitif, qui définit différents niveaux d'"observation" souhaitables du parallélisme,
- un aspect sémantique définissant des modèles d'interprétation pour ces niveaux d'observation,
- un aspect syntaxique qui précise, dans chaque cas, différents modes d'expression.

Nous commençons par préciser quelques points de vocabulaire.

Intuitivement, on parle de parallélisme dès que plusieurs "procédés" agissent simultanément avec, éventuellement des interactions des uns sur les autres. Pour préciser quelque peu ce point de vue nous proposons les définitions suivantes.

De manière très générale, on peut considérer un dispositif physique, un algorithme (système de commande, calcul de la solution d'un problème, etc.), ou un programme

-nous dirons par la suite un système- comme un ensemble de données et un ensemble de processus organisés selon un certain schéma de contrôle.

Une donnée est un objet d'un certain type. Un type définit l'ensemble des opérations admises sur ses objets.

Un processus est un certain agencement de ces opérations sur des données pour produire un ensemble de résultats.

Un schéma de contrôle définit l'organisation de ces processus pour constituer un système. Un tel schéma peut prendre plusieurs formes:

- . il peut être complètement ou partiellement implicite dans des implantations de type "data-flow" ((Den_75), (SyA_76), (AGP_77), etc.),

- . il peut utiliser diverses primitives classiques: séquence, alternative (déterministe ou non), itération, etc.,

- . il peut spécifier des activations simultanées d'opérations.

Nous appelons système parallèle un système dans lequel le schéma de contrôle autorise des activations simultanées de certaines occurrences d'opérations. Nous appelons processus parallèles des processus comportant de telles opérations.

Des processus parallèles peuvent être indépendants ou non. Lorsqu'ils sont dépendants leurs interactions peuvent s'exprimer de diverses manières:

- une première façon de faire, qui est très courante, est de préciser un ensemble de contraintes de synchronisations entre différentes occurrences d'opérations de processus parallèles. Ce point de vue est justifié dans certains problèmes de parallélisme "de situation", où on cherche à exprimer des relations de causalité d'événements,

- une deuxième voie est de préciser un ensemble de ressources "partagées" par des processus parallèles, et les conditions de ce partage. Ce point de vue, abusivement utilisé, devrait être limité -au niveau d'une certaine abstraction- à des exemples où le partage fait partie de la spécification du problème (problème du ramasse-miettes, par exemple (cf. (DLM_78), (Fin_83), ...)),

- un troisième mode d'expression, bien adapté à un certain niveau d'abstraction, est de considérer que les interactions de processus sont des "échanges" de données, ou communications. C'est ce point de vue que nous adoptons.

En fait ces différents points de vue reviennent à différentes expressions et interprétations du "parallélisme" qui sont l'objet du présent chapitre et qui conduisent à des discussions qui sont un des thèmes principaux de cette thèse. Sans enqoquer pour l'instant le lecteur dans cette discussion nous parlerons dans la suite, et de manière très générale, de processus communicants pour définir des processus parallèles dont les interactions sont exprimées par des communications de données.

1-1- OBSERVATIONS DU PARALLÉLISME-

Les différents modèles du parallélisme dans lesquels on peut interpréter l'expression de systèmes reflètent différents niveaux "d'observation" de leur parallélisme. Le niveau pertinent d'observation du parallélisme dépend du problème considéré. On répertorie assez classiquement les niveaux suivants:

-Observation du non-déterminisme:

A ce niveau d'observation on assimile le parallélisme de deux opérations atomiques au fait qu'elles peuvent être activées dans un ordre quelconque de manière indifférentiable sur l'état résultant du système.

Si on note "." la mise en séquence de deux opérations et "+" le choix non-déterministe entre deux opérations, on peut exprimer le parallélisme de deux opérations atomiques A et B ("approximé" par le non-déterminisme de leur ordonnancement) par l'équation:

$$A // B = A.B + B.A$$

Cette "approximation" du parallélisme exclut en particulier l'interprétation d'actions bloquantes.

-Observation du parallélisme asynchrone:

Dans les modèles de ce type on observe que les processus parallèles d'un système évoluent de manière autonome et indépendante (d'où le terme "asynchrone") hormis quelques contraintes ponctuelles de "rendez-vous" pour l'activation d'opérations complémentaires ou le respect de propriétés d'intégrité du système. On peut résumer ce type d'expression du parallélisme -dit parallélisme à synchronisation explicite dans (DuB_82)- par la formule suivante:

PARALLELISME = NON-DETERMINISME et SYNCHRONISATIONS
--

-Observation du parallélisme synchrone:

Dans les modèles de ce type on observe l'évolution de chaque processus parallèle relativement à une représentation abstraite du temps selon une échelle discrète. Chaque opération est alors considérée comme instantanée dans cette échelle. Cette description est particulièrement adaptée pour des problèmes dits "temps-réel" ou des "algorithmes synchrones" (terminologie de (LiA_81)) où toutes les opérations sont activées relativement à une horloge globale: algorithmes SIMD, vectoriels ou systoliques ((Kun_79), (Mon_85)).

1-2- INTERPRETATIONS DU PARALLELISME-

Aux différents niveaux d'observation du parallélisme correspondent différents modèles dans lesquels sont interprétées diverses expressions du parallélisme. Nous présentons très brièvement les modèles du non-déterminisme, du parallélisme asynchrone puis du parallélisme synchrone. Cette classification en recoupe une autre liée à la nature des objets définis dans le modèle: mots, vecteurs ou arbres.

1-2-1-Modèles du non-déterminisme-

Dans ces modèles on définit un système comme un ensemble de "comportements". Un comportement est un mot éventuellement infini sur un alphabet d'opérations et de symboles de contrôle (séquence, choix non déterministe, etc.).

Exemples: $(A + B)^\omega$ définit un système activant indéfiniment les opérations A ou B.

$(A.B + B.A)^\omega$ définit un système activant indéfiniment la paire d'opérations A et B dans un ordre quelconque.

En fait les études portant sur cette approche "langage formel" considèrent les propriétés de ces langages par la définition des automates qui les reconnaissent (voir par exemple (Arn_79), (Niv_80) ou (NiA_82)) et qui expriment le schéma de contrôle du système (Rou_78'). Un tel système est alors vu comme une suite, éventuellement infinie, d'états pour former un système de transitions ((Kel_76), (Sif_79)).

Un système de transitions est un couple $(Q, \rightarrow)$ où Q est un ensemble, éventuellement infini, d'états -ou "configurations" dans (Arn_79), (NiA_82) et (Plo_81)- et où $\rightarrow$ est une relation binaire sur Q. Cette relation définit un ensemble de "transitions déclenchables". Au déclenchement, supposé indivisible, de chaque transition est associée l'activation d'une opération du système ainsi décrit.

La caractéristique d'un système parallèle (décrit comme un système non-déterministe) est que cette relation n'est pas une fonction, c'est-à-dire qu'étant donné q dans Q il peut exister plusieurs q' dans Q tels que $q \rightarrow q'$. Ces états q' sont ceux qui résultent de l'état q par le déclenchement de toute transition autorisée dans cet état. Pour interpréter un système parallèle on peut étendre ce modèle en étiquetant chaque transition par le nom de l'opération associée.

On obtient ainsi un système de transitions étiquetés qui est un triplet $(E, Q, \rightarrow)$ où E est l'ensemble des étiquettes, $(Q, \rightarrow)$ est un système de transitions et

$\forall t \in E$, $--t->$ est une fonction telle que:
 $\forall q \in Q$, il existe au plus un $q' \in Q$ image de q par $--t->$ avec $q-->q'$.

Si q' existe on dit que la transition t est autorisée dans l'état q .

Plus précisément, $\forall t \in E$, la fonction $--t->$ est définie par:

Pour tout $q \in Q$,
 . Un prédicat d'autorisation $Pt(q)$, avec $Pt: Q \rightarrow \text{booléen}$
 . Une application $Ft(q)$, avec $Ft: Q \rightarrow Q$

tels que $q--t->q'$ ssi $Pt(q)$ et $Ft(q)=q'$.

1-2-2-Modèles du parallélisme asynchrone-

Dans ces modèles on exprime d'une part les processus, comme des systèmes non-déterministes sous la forme de suites d'opérations ou de systèmes de transitions, d'autre part une relation de synchronisation qui précise les opérations qui doivent être activées simultanément.

Lorsque les processus sont décrits sous forme de systèmes de transitions les synchronisations sont exprimées implicitement par une propriété invariante sur l'état global du système. On peut alors approximer le parallélisme de ce système en le réalisant à son tour sous la forme d'un système de transitions, dont on doit garantir qu'il vérifie la dite propriété.

Lorsque les processus sont décrits par des mots sur l'alphabet des opérations les synchronisations sont définies explicitement

- soit par la liste des symboles correspondant à des opérations "synchronisées". C'est le cas dans (NiA_82) avec les "vecteurs instantanés de synchronisation". Un simple exemple permet d'illustrer ce point de vue:

Sur l'ensemble d'opérations $\{A, B, C, D\}$ on définit les processus P1 et P2 suivants:

$$P1 = (A.B)^{\omega}, \quad P2 = (C.D)^{\omega}$$

La condition pour que B et D soient en exclusion mutuelle, et que A et C soient simultanées est exprimée par l'ensemble des vecteurs de synchronisation suivant:

$$S = \{ \langle A,C \rangle, \langle B,A \rangle, \langle A,D \rangle \}$$

où $\wedge$ est l'opération "vide".

- soit par la définition d'opérations complémentaires activées simultanément: par exemple, l'émission et la réception d'un signal sur un canal de communication. Ce point de vue est présenté, entre autres, dans (Mil_80) où on définit une interprétation des "agents communicants" dans un modèle d'algèbre d'arbres infinis, appelés arbres de comportements. Nous illustrons brièvement ce modèle:

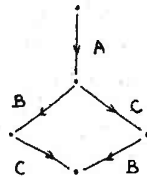
Chaque processus -ou agent- est caractérisé par un ensemble de comportements non-déterministes interprété par un arbre éventuellement infini, représentant l'automate du langage de ces comportements, dont les branches sont étiquetées par un nom d'opération et dont les noeuds représentent l'état de l'agent résultant de l'activation de cette opération.

Par exemple les processus P1 et P2 ci-dessus peuvent s'interpréter par des arbres infinis figurés par les graphes suivants:

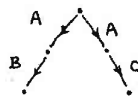
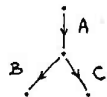


Un système non-déterministe peut être interprété de la manière suivante, illustrée par l'exemple:

$$A.(B // C) = A.(B.C + C.B)$$

Remarque-

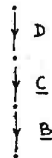
Dans cette interprétation $A.(B+C) \neq A.B + A.C$



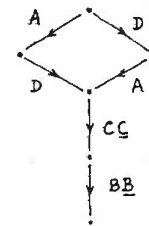
puisque dans le premier cas le choix non déterministe est fait après l'action "A".

A partir de la définition d'opérations complémentaires (pour toute opération "X", nous noterons " $\bar{X}$ " l'opération complémentaire de "X"), on définit une algèbre d'arbres de comportements par l'opération de connexion présentée sur l'exemple suivant:

Soit $D.\bar{C}.\bar{B}$ le comportement d'un agent, représenté par l'arbre



La connexion des deux arbres ci-dessus définit l'arbre

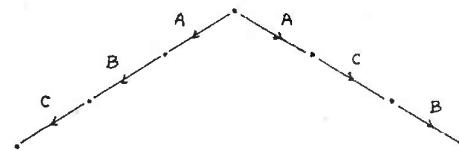


qui interprète le système dont l'ensemble des comportements est

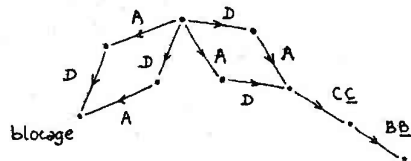
$$(A.(B // C)) // (D.\bar{C}.\bar{B})$$

Remarquons que ce modèle du parallélisme est différent d'un modèle du non-déterminisme puisqu'il permet d'exhiber des situations de blocage.

Exemple: l'ensemble de comportements $A.B.C + A.C.B$ interprété par



est différent de l'ensemble $A.(B // C)$. En effet, la connexion de son arbre avec l'arbre précédent associé à $D.\bar{C}.\bar{B}$ fait apparaître un état du système correspondant à un blocage:



1-2-3-Modèles du parallélisme synchrone-

Dans ces modèles on considère que l'observation d'un changement de l'état global d'un système est conditionnée par une évolution de tous les processus parallèles qui le composent, par l'activation d'une opération. Dans ces conditions, ces opérations peuvent être considérées comme instantanées par rapport à l'échelle du temps discrétisé choisie.

Un premier type d'expression présente un système comme un vecteur dont les composantes sont les suites d'opérations des processus parallèles.

Exemple: le vecteur $\begin{pmatrix} A^\omega \\ B^\omega \end{pmatrix}$ définit un système composé de deux processus parallèles. Le premier active indéfiniment l'opération A, le second active indéfiniment l'opération B.

L'évolution de ce système est alors interprétée par un ensemble de vecteurs dont les composantes sont des préfixes des mots décrivant les processus.

Une interprétation synchrone considère les préfixes suivants, pour l'exemple:

$$\begin{pmatrix} A \\ B \end{pmatrix}, \begin{pmatrix} A.A \\ B.B \end{pmatrix}, \dots, \begin{pmatrix} A^n \\ B^n \end{pmatrix}, \dots$$

On peut admettre également une interprétation asynchrone de ce type d'expression en enrichissant l'alphabet de l'opération vide Λ . Pour l'exemple considéré les vecteurs suivants présentent une évolution possible du système:

$$\begin{pmatrix} \Lambda \\ B \end{pmatrix}, \begin{pmatrix} \Lambda \\ B.B \end{pmatrix}, \begin{pmatrix} A \\ B.B \end{pmatrix}, \begin{pmatrix} A.A \\ B.B \end{pmatrix}, \dots$$

Un autre modèle de ce type consiste à interpréter une expression de système parallèle dans une algèbre de processus. C'est le cas par exemple dans (Mil_83) où on présente un système comme un "produit" d'agents dont l'ensemble des "calculs" est défini par la règle de déduction suivante:

- . si l'agent p' résulte de l'agent p par l'opération "a",
- . si l'agent q' résulte de l'agent q par l'opération "b",

la règle:

$$\frac{p \xrightarrow{a} p' \quad , \quad q \xrightarrow{b} q'}{p * q \xrightarrow{ab} p' * q'}$$

définit l'évolution du système produit par l'activation simultanée de a et de b.

Il faut noter que ces modèles permettent également une interprétation asynchrone des systèmes parallèles par l'introduction de l'opération vide.

Celle-ci est définie, en particulier dans (Mil_80'), (Mil_83) ou (NiA_82), à partir d'une opération neutre de "délai" qui intuitivement exprime une inactivité d'un processus, par exemple en attente d'un événement. Cette opération neutre, notée "l", est telle que

$$\frac{p \xrightarrow{l} p' \quad , \quad q \xrightarrow{b} q'}{p * q \xrightarrow{lb} p' * q'}$$

Dans le cas particulier où on définit $p \xrightarrow{l} p$, pour tout agent p , on peut donc interpréter un système parallèle de manière asynchrone.

Certains auteurs (voir, par exemple (Jor_82) ou (AuB_83)) adoptent une solution "mixte" en définissant l'évolution d'un système ou bien par l'activation simultanée d'opérations par des processus parallèles, ou bien par un entrelacement d'opérations, par les règles de déduction suivantes (cf. le calcul MEIJE (Bou_85)):

```

      p --a-> p'
-----
    p*q --a-> p'*q

      q --b-> q'
-----
    p*q --b-> p*q'

    p --a-> p'      ,      q --b-> q'
-----
    p*q --ab-> p'*q'

```

1-3- EXPRESSIONS DU PARALLELISME-

Différentes expressions du parallélisme sont proposées dans la littérature, qui s'interprètent dans les modèles présentés ci-dessus. Elles diffèrent essentiellement par la façon d'exprimer les synchronisations.

1-3-1-Expression implicite des synchronisations-

Ce type d'expression s'interprète dans un modèle "états-transitions", par la définition de propriétés invariantes sur l'état global du système. Ces propriétés sont écrites en termes de compteurs associés à chaque opération (voir, par exemple (RoV_77) ou (Mar_81)). Une transition est déclenchable lorsque la valeur des compteurs de l'opération associée satisfait cet invariant.

Exemple: les opérations "déposer" et "retirer" définies sur un tampon de communication à n places vérifient la propriété suivante:

aut (déposer) - # term (retirer) $\leq n$
 # term (déposer) - # aut (retirer) ≥ 0

où # désigne le compteur d'événements aut (pour

autorisation) ou term (pour terminaison).

De la définition d'un tel invariant on déduit des prédicats d'autorisation des transitions, exprimés par exemple sous la forme de "modules de contrôle" (RoV_77).

1-3-2-Expression explicite des synchronisations-

Parmi les moyens d'expression explicite des synchronisations, les mots de synchronisation (Rou_78) sont une généralisation de la notion de compteur d'événements, qui gardent la trace de l'ordre et du type de ces événements. Ces "mots" sont définis sur un alphabet d'opérations: à tout instant, un mot de synchronisation définit la suite des requêtes d'opérations nécessitant une synchronisation.

Exemple: Une solution à l'exemple précédent du tampon à n places nécessite la manipulation de trois mots de synchronisation:

- un mot, noté P, de productions dans le tampon,
- un mot, noté C, de consommations dans le tampon,
- un mot, noté L, de l'état d'occupation des places du tampon.

Initialement, les mots P et C sont vides; le mot L égale 1^n pour indiquer que les n places sont libres.

Les opérations "déposer" et "retirer" définissent comme suit les mots de synchronisation:

<u>ajout</u> (P:p)	<u>test</u> (C:c)
<u>test</u> (L:l)	<u>retirer</u>
<u>déposer</u>	<u>efface</u> (C:c)
<u>ajout</u> (C:c)	<u>ajout</u> (L:l)
<u>efface</u> (L:l , P:p)	

où ajout, test, efface sont des primitives de manipulation des mots dont la signification est bien suggérée par le nom.

Une autre expression classique des synchronisations sont les expressions de chemin (voir un ensemble d'articles ou de thèses: entre autres, (CaH_74), (LaS_78) et (LTS_79)).

Ces expressions définissent les relations de séquençement, d'exclusion mutuelle, etc. entre les opérations nécessitant des synchronisations, par des schémas du type:

```
path p ; q end
path p , q end
```

Certaines constructions en COSY, ou certaines extensions (FlH_76) permettent une expression simple des synchronisations des opérations sur un tampon à n places, sous la forme suivante:

```
path ( déposer - retirer )n end
```

Enfin une expression très répandue des systèmes parallèles sont les réseaux de Petri, qui sont l'objet d'une abondante littérature. Pour une étude générale le lecteur peut se reporter à (Pet_81). Pour des études plus particulières -propriétés structurelles, réductions, extensions, systèmes de preuves associés, etc.- on se reportera à (Ber_79), (Ber_78), (Sif_79), (Que_82), (ISI_85), etc.

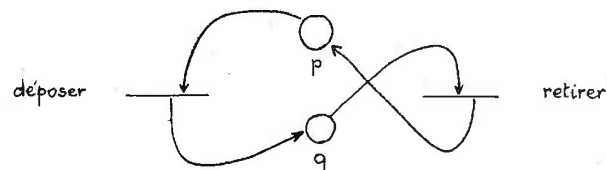
Nous nous contenterons ici des quelques rappels suivants:

Un réseau de Petri est une représentation d'un système de transitions sous la forme d'un graphe biparti. Les noeuds sont

- soit des places schématisées graphiquement par des cercles, qui représentent l'état du système, par une fonction de "marquage",

- soit des transitions schématisées graphiquement par des barres, qui figurent les opérations du système.

Les arcs relient les transitions via des places, comme le montre l'exemple suivant:



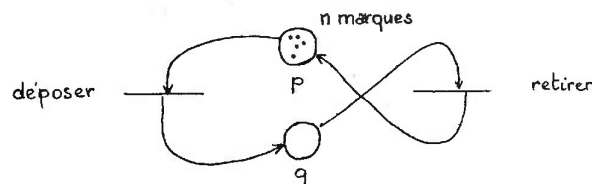
Sur ce réseau, la place "p" est dite en "entrée" de la transition "déposer" et en "sortie" de la transition "retirer" (et inversement pour la place "q").

Le marquage d'un réseau de Petri est une application de l'ensemble de ses places dans $\mathbb{N}$. Si M est un marquage, $M(p)$ est le marquage de la place "p" et est représenté graphiquement par $M(p)$ points, ou marques, dans le cercle figurant "p".

Une transition "t" est déclenchable, pour un marquage M , si et seulement si pour toute place "p" en entrée de "t", $M(p) > 0$.

Le déclenchement d'une transition définit un nouveau marquage dans lequel une marque est retirée de toutes les places en entrée de cette transition, et une marque ajoutée à toutes ses places en sortie.

Exemple: le marquage "initial" suivant



définit les synchronisations des opérations "déposer" et "retirer" sur un tampon à n places.

On trouve dans la littérature des extensions de ces réseaux

- qui n'augmentent pas la "puissance" du modèle sous-jacent: par exemple, la valuation des arcs (réseaux dits

"généralisés"),
 - ou qui augmentent cette puissance, par ajout du "test du zéro", permettant la simulation d'une machine de Turing: par exemple, les réseaux avec arcs "inhibiteurs", les réseaux avec prédicats ou les réseaux temporisés.

1-3-3-Les langages à "rendez-vous"-

Nous évoquons rapidement d'autres moyens d'expression du parallélisme asynchrone

- soit sous la forme d'agents communicants dans CCS (Mil_80), ou LCS (Ber_85), considérés comme des potentialités de communication et dont le comportement est interprété dans une algèbre d'arbres. Des noms de ports complémentaires attachés à chaque agent définissent ces potentialités.

- soit sous la forme de langages impératifs du type CSP (Hoa_78) ou ADA (IcA_79), où les processus, ou tâches, sont synchronisés par "rendez-vous" de la manière suivante:

Les processus (séquentiels, non-déterministes) de CSP communiquent des valeurs de données locales via un canal défini de façon symétrique par le nom du processus interlocuteur et de la donnée communiquée. Un symbole réservé indique la direction d'utilisation du canal: "!" pour l'émission, "?" pour la réception, selon un schéma du type suivant:

```

P :: .      Q :: .
   .      .
   [      [
   .      .
   .      .
   Q!x    P?y
   .      .
   .      .
   ]      ]

```

Ces deux notations définissent des opérations complémentaires, au sens de 1-2-2- ci-dessus, qui sont donc activées simultanément.

Les tâches ADA sont synchronisées par l'appel de "points d'entrée", selon un schéma du type suivant:

```

task P is
  entry retirer(y:out...)
end ;
task body P is
  .
  .
  .
  begin
  .
  .
  accept retirer(y:out...) do
  .
  .
  end ;
  .
  .
end P ;

task Q ;
task body Q is
  .
  .
  .
  begin
  .
  .
  P.retirer(y)
  .
  .
end Q ;

```

Dans ce schéma l'appel du point d'entrée "retirer" dans la tâche Q est satisfait par l'opération "accept" de la tâche P.

On observe en fait sur ces expressions dites "communications par rendez-vous" une confusion entre la notion de synchronisation qui établit une relation de dépendance entre certaines occurrences d'opérations, et la notion de communication proprement dite, qui ne devrait établir que des relations entre des valeurs de certaines données des processus communicants.

Cette confusion existe également dans d'autres langages où sont définies d'autres opérations primitives de communication, du type "communications par messages", qui induisent plus ou moins de contraintes de synchronisations. Citons par exemple GYPSY (AmA_77), PLITS (Fel_79), SYGARE (Tho_80) ou SR (And_81).

Cette confusion peut conduire à des sur-spécifications prématurées des relations entre processus, dans une étape de description d'un système parallèle ou de conception d'une

solution algorithmique. Nous reviendrons sur ce point dans les paragraphes suivants: il est l'un des sujets de discussion essentiels de cette thèse.

1-3-4-Expression de la concurrence-

On n'a pas encore parlé d'expressions pourtant classiques de systèmes parallèles, formés de processus concurrents au vrai sens du terme, puisqu'ils doivent concrètement accéder à des ressources communes en respectant des contraintes d'intégrité des dites ressources.

Certains modèles en sont proposés, qui reposent sur la notion de processus et de "variables partagées" ((LyF_81), (Pet_83)).

Ce type d'expression est courant dans beaucoup d'études, notamment sur les preuves de systèmes parallèles. Nous y reviendrons au chapitre 6, mais on peut d'ores et déjà citer (Dij_78), (Sin_79), (OwG_76), (Apt_81), (Lam_80), (Pnu_81), (OwL_82), etc.

Dans toutes ces présentations, on décrit un système parallèle de la manière suivante:

$S = \text{cobegin } S1 // S2 // \dots // Sn \text{ coend}$

où les processus $S1, S2, \dots, Sn$ partagent des variables globales, ce qui nécessite l'introduction de nouvelles primitives de contrôle du type condition-action (await b then S, etc.) ou définissant des sections critiques.

Ces primitives se réfèrent en fait aux premiers outils d'expression pour la synchronisation des processus: sémaphores (Dij_72), régions critiques conditionnelles (BrH_72) puis moniteurs (Hoa_74). Essentiellement conçus pour résoudre des problèmes d'exclusion mutuelle, ils deviennent dans les expressions citées ci-dessus le seul moyen de description des interactions entre processus parallèles, par l'introduction de ressources communes. On montre dans (Ray_82) comment on exprime ces contraintes d'accès dans d'autres langages (modules de contrôle et expressions de chemin).

1-4- NOTION DE COMMUNICATION-

On remarque que les modèles présentés ci-dessus, parce qu'ils caractérisent tout système par son comportement, c'est-à-dire une succession d'opérations, permettent d'interpréter la synchronisation des processus, soit par la définition explicite de relations entre des occurrences d'opérations, soit par le respect de contraintes d'accès à des ressources communes. Ces modèles permettent également de caractériser des situations particulières des systèmes parallèles: par exemple, un état de blocage, un risque "d'attente infinie", etc.

Mais ces modèles ne traitent pas des valeurs des données partagées ou échangées: tout au plus les langages à "rendez-vous", directement issus de la philosophie inhérente aux processus concurrents, lient-ils échange de valeur et synchronisation, ce qui ne laisse pas d'entretenir la confusion sur ces deux notions.

Au contraire, nous proposons un modèle, qui caractérise tout système par les manipulations autorisées sur des ensembles de données, et permet d'interpréter réellement la communication entre processus, au sens de la définition de relations entre des valeurs de données. De ces relations sont déduites les synchronisations des processus, nécessaires à leur réalisation.

De la sorte, et en regard de la formule présentée au paragraphe 1-1- ci-dessus, le modèle de parallélisme proposé est défini par:

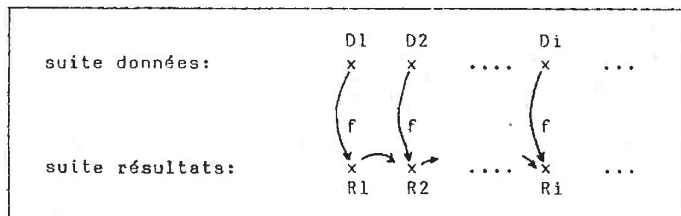
PARALLELISME = NON-DETERMINISME et COMMUNICATIONS
--

1-4-1-Relation de communication-

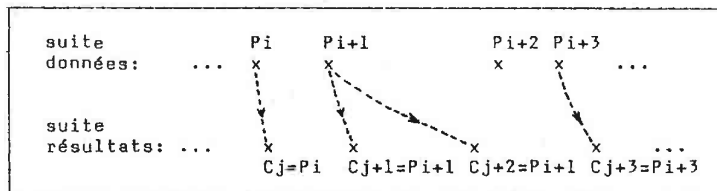
Il est classique d'interpréter "l'univers séquentiel" dans un système formel de type ensembliste, c'est-à-dire permettant la définition d'ensembles, éventuellement

caractérisés par un prédicat, l'utilisation d'opérations ensemblistes (union, intersection, ...), de relations, etc. (voir par exemple (KeS_75), (BPW_79), (Abr_78)).

On peut ainsi considérer un modèle dont les objets sont des suites d'éléments dont le type est, par exemple, défini par un type abstrait algébrique (Fin_79). Alors on peut spécifier fonctionnellement un problème -séquentiel- par la définition d'une relation statique entre des suites de données et des suites de résultats. Nous appelons relation de calcul une telle relation, qu'on peut figurer comme suit:



Notamment dans l'"univers parallèle", il est intéressant de particulariser une autre classe de relations, que nous appelons relations de communication (FJP_83). Elles transforment généralement le nombre d'occurrences et l'ordre des termes sans en modifier les valeurs, comme le montre l'exemple de la figure suivante:



Nous présentons maintenant un ensemble d'exemples, décrits dans un langage de type CSP, illustrant quelques situations "intéressantes" du point de vue de l'expression de ce que nous venons de présenter comme des "relations de communication".

1-4-2-Exemples-

Considérons le schéma itératif producteur-consommateur suivant:

```

P :: var x:...;      C :: var y:...;
  * [
    .
    .
    C!x;              P?y;
    .
    .
  ]

```

Ce schéma exprime deux propriétés:

- la première, déjà mentionnée, est que les opérations complémentaires de communication sont activées simultanément,
- la seconde est la définition d'une relation entre la suite des valeurs émises par le processus P et la suite des valeurs reçues par le processus C: si aucune autre opération de communication n'apparaît dans le corps des processus P ou C, d'après les contraintes de synchronisation ci-dessus, cette relation est ici l'égalité.

On comprend bien maintenant que ce type d'expression n'est pas adapté au niveau de la conception d'un système, niveau pour lequel on souhaiterait pouvoir raisonner "à l'envers", c'est-à-dire déduire de la relation voulue entre les suites de valeurs échangées, les contraintes de synchronisation nécessaires.

Ceci d'autant plus que cette relation n'est pas toujours aussi triviale que celle présentée ici: on trouvera dans cette thèse (et également pour certains d'entre eux dans (Per_82), (Jul_83), (Per_83) ou (Per_83')) de nombreux exemples de systèmes producteur-consommateur plus "complexes", qui peuvent être décrits en introduisant un processus intermédiaire entre les processus P et C précédents.

Exemple: la relation entre la suite des valeurs émises et la suite des valeurs reçues est telle que chaque valeur traitée par le processus C est la "dernière" valeur émise par P, au "moment" de ce traitement (sous des hypothèses convenables d'équité du choix non-déterministe, non discutées ici):

```

P :: var x:...;
  * [ .
    .
    !x;
    .
    .
  ]
  T :: var d:...;
    p,b:bool;
    [ p:=false;
      * [
        [ P?d --> p:=true
          [ C?b ^ p --> C!d
        ]
      ]
    ]

C :: var y:...;
  s:bool;
  * [ .
    .
    !s;
    !?y;
    .
    .
  ]

```

Cet exemple met en évidence deux problèmes:

- le premier est que, pour décrire la relation voulue, on doit introduire une communication "parasite" du processus C vers le processus T. Ceci nuit à la modularité souhaitable, en particulier, au niveau d'un langage de spécification.

- le second est celui que pose l'univers du parallélisme, exprimé naïvement par les termes "dernier" et "moment". Cela signifie que la relation entre les suites de valeurs émises et reçues ne peut certainement pas être une relation statique dans les systèmes formels évoqués ci-dessus.

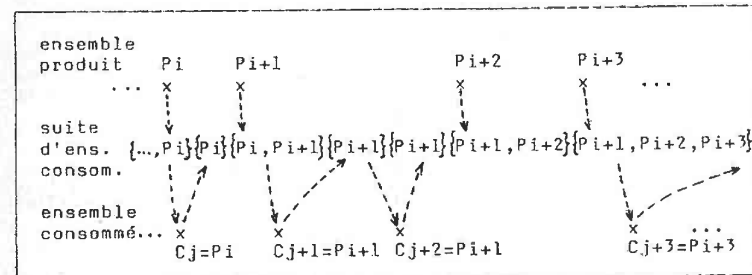
Cet aspect "dynamique" peut être exprimé dans le cadre formel présenté au chapitre 2, à partir des quelques considérations intuitives suivantes.

1-4-3-Caractéristiques d'une communication-

Lorsqu'un processus consommateur "souhaite" recevoir la valeur d'une donnée il dispose d'un ensemble de valeurs émises par son interlocuteur producteur: une relation de communication doit indiquer quelle valeur est choisie dans cet ensemble. Mais cette relation doit également indiquer comment est modifié cet ensemble par une telle opération: par exemple, si chaque valeur est consommée une fois au plus, elle doit disparaître de l'ensemble. A l'inverse, si une valeur peut être consultée à loisir, elle doit figurer à nouveau dans l'ensemble.

On voit ainsi que l'élément essentiel de la définition d'une relation de communication est cet ensemble de valeurs émises par le processus producteur et dont dispose le processus consommateur. Dans la suite, nous appellerons cet ensemble l'ensemble consommable. Du point de vue des processus eux-mêmes, l'ensemble des valeurs émises, appelé l'ensemble produit, et l'ensemble des valeurs reçues, appelé l'ensemble consommé, sont respectivement caractéristiques du producteur et du consommateur.

Le modèle présenté dans les chapitres suivants reprend ce point de vue: chaque relation de communication y est caractérisée par la définition de ces trois ensembles. Pour introduire ce modèle nous montrons sur la figure suivante comment évoluent ces ensembles sur l'exemple de relation de communication présenté ci-dessus en 1-4-1-:



La relation exprimée sur cette figure, par les flèches en pointillé, inclut donc cet aspect "temporel" inévitable dans la spécification d'un système parallèle: chaque ensemble produit ou consommé étant indicé selon la chronologie de définition de ses termes, la définition d'une relation de communication consiste à établir une correspondance entre ces deux chronologies.

Cette notion de chronologie sera exprimée au chapitre 4 par la notion de suite temporelle:

- On peut, pour l'instant comprendre intuitivement, pour cet exemple, la définition suivante de la suite temporelle S_2 d'ensembles consommables (cf. paragraphe 4-3-2-). Son terme général est défini par:

```

S2l = si B0 <> {}
-- B0 est l'ensemble des valeurs produites
   jusqu'à l'instant, noté instant(SC1), de la
   première requête de consommation:

B0 = {Pi tels que instant(SPi) < instant(SC1)}
    où instant(SPi) est l'instant de la ième
    production Pi
--
alors (B0, instant(SC1))
sinon ({Pi}, instant(SPl))
fsi

```

et pour tout $j > 0$:

```

ensemble(S2j+1) = Bj
instant(S2j+1) = instant(SCj+1)
-- Bj est l'union de l'ensemble consommable
   résultant de la consommation précédente et
   des productions survenues jusqu'à
   l'instant, noté instant(SCj+1) de la j+1ème
   requête de consommation. La seconde
   équation signifie l'absence d'attente du
   processus consommateur.

Bj = {Pi ∈ ensemble(S2j) tel que
      instant(SPi) = max{instant(SPk),
                        Pk ∈ ensemble(S2j)}}
    U {Pk tels que instant(SCj) < instant(SPk)
        = instant(SCj+1)}

sur la figure Bj = {Pi, Pi+1}
--

```

- la suite temporelle S_1 exprime la suite des ensembles produits,

- la suite temporelle S_3 exprime la suite des ensembles consommés. Elle est définie par:

```

ensemble(S3j+1) = ensemble(S3j) U {Pi}
tel que Pi ∈ ensemble(S2j)
et instant(SPi) = max{instant(SPk),
                      Pk ∈ ensemble(S2j)}

```

Le chapitre 2 présente un cadre formel dans lequel une relation de communication est exprimée par la manipulation d'un objet - triplet d'ensembles, noté $\langle SP, A, SC \rangle$ - par trois opérations significatives de cette relation: pré-cons, val-cons et post-cons.

Sur l'exemple,

1- de la définition conditionnelle de S_2 : $B_0 \neq \{\}$
 et de la post-condition de consommation (dont on déduit $B_j \neq \{\}$)

on déduit que la pré-condition de consommation est que l'ensemble consommable soit non vide. Ce qui se traduira par la définition:

pré-cons($\langle SP, A, SC \rangle$) = non vide?(A)

2- de la définition de S_3 : $j+1$ on déduit que la donnée consommée est la "dernière" ajoutée à l'ensemble consommable. Ce qui se traduira par la définition:

val-cons($\langle SP, A, SC \rangle$) = dernier(A)

3- de la définition de B_j , dont la première composante est l'état résultant de l'ensemble consommable:

```

{Pi ∈ ensemble(S2j) tel que
 instant(SPi) = max{instant(SPk),
                    Pk ∈ ensemble(S2j)}
}

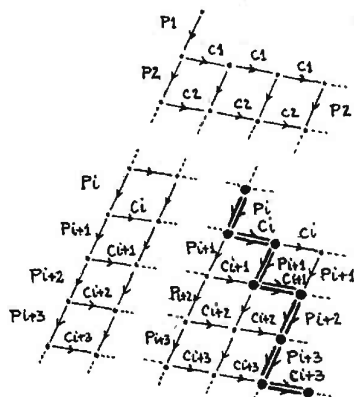
```

on déduit que cet état résultant est l'ensemble réduit à son dernier élément. Ce qui se traduira par la définition:

post-cons($\langle SP, A, SC \rangle$) = finir(A)

Cette structure et ces opérations définissent un type abstrait, ou type de communication. Pour l'exemple considéré ici ce type sera appelé "aléatoire" (cf. paragraphe 2-1-3-).

Cette définition peut être illustrée par le graphe suivant, qui définit l'ensemble des calculs possibles d'un système limité à des opérations de production et de consommation. Sur les arcs de consommation on indique l'indice de production de la valeur consommée.



-- on a indiqué en double flèche un chemin correspondant à la situation décrite sur la figure précédente

Au chapitre 5, l'étude de tels graphes permet de montrer quelques propriétés des types de communication.

CHAPITRE 2- DEFINITION D'UNE COMMUNICATION ET EXPRESSION DE SYSTEMES PARALLELES

Nous considérons un système parallèle comme un ensemble de processus qui communiquent par des "ports". Chaque port exprime les propriétés de la relation de communication des données qui l'utilise.

Nous adoptons l'expression suivante, pour un processus séquentiel communicant:

processus P ::

<u>entrée</u> ...	-- définition de l'utilisation des
<u>sortie</u> ...	ports de communication, déclarés
	dans le système parallèle

-- déclarations de données ou processus locaux --

corps

-- définition des actions du processus --

fin P_i

Les clauses "entrée" et "sortie" constituent l'"interface" du processus. Les autres éléments de sa

définition constituent son "texte", et sont internes à ce processus.

Expression d'un système parallèle:

Le corps d'un processus peut faire référence à des processus internes: -un tel processus exprime alors un système parallèle-, chacun d'eux étant déclaré dans un paragraphe "processus":

. lorsque ces processus internes communiquent entre eux une clause "port" définit les ports utilisés,

. chaque processus interne comprend alors, à son tour, l'indication de l'utilisation des ports de communication, par la définition des "données communiquées" associées,

. chaque processus interne est "fermé", c'est-à-dire ne définit pas de port de communication visible depuis le processus externe,

. les processus internes sont activés simultanément dans le corps du processus externe, noté alors "composition".

Ainsi, un système producteur-consommateur du type de celui présenté au paragraphe 1-4-2- est défini par:

```
processus PROD_CONS ::
```

```
  port a: ... i
    -- définition du port de communication
    et de ses propriétés
  --
```

```
  processus PROD ::
    sortie xi ... vers a:
    -- utilisation du port de communication
  --
```

```
  corps
  .
  .
  produire(x) i
  .
  .
```

```
fin PROD;
```

```
processus CONS ::
  entrée y: ... via a:
  -- utilisation du port de communication
  --
```

```
  corps
  .
  .
  consommer(y) i
  .
  .
```

```
fin CONS;
```

```
composition
```

```
  PROD // CONS
```

```
fin PROD_CONS;
```

Remarques-

- Dans la clause "sortie", on indique la liste des noms de ports de communication sur lesquels est produite chaque donnée communiquée,

- Les clauses "ports", "entrée" et "sortie" définissent le "réseau de communications" d'un système parallèle.

L'objet de ce chapitre est de préciser

- la définition de l'aspect "communication",

- succinctement, l'expression d'un système parallèle incluant ce concept

et de présenter quelques exemples simples illustrant ces propos. Un ensemble plus complet d'exemples est proposé au chapitre 3.

2-1- TYPE DE COMMUNICATION-

Nous avons choisi, pour sa simplicité et sa puissance d'expression, d'exprimer les dépendances entre processus communicants par la notion de port de communication: un tel port est défini par trois éléments:

- un nom,

- le nom et le type des données (ou "messages" dans la littérature) utilisant le port, définis dans les clauses entrée et sortie -un port est unidirectionnel-: ces données, que nous appelons données communiquées, peuvent être traitées par deux opérations:

- . une opération de production, notée produire. Tout processus activant cette opération est appelé "processus producteur" de cette donnée,
- . une opération de consommation, notée consommer. Tout processus activant cette opération est appelé "processus consommateur" de cette donnée.

L'association de ces deux éléments: nom de port et noms de données communiquées l'utilisant définit un canal de communication,

- un type de communication, définissant les propriétés des relations de communication des données communiquées utilisant ce port, c'est-à-dire la sémantique des opérations autorisées sur ces données.

Lorsqu'aucune confusion ne sera possible, nous confondrons abusivement le nom d'une donnée communiquée et celui du port: on parlera alors du type de communication associé à cette donnée.

Nous notons "TC" l'application qui associe un type de communication à un port: si "a" est le nom d'un tel port (ou d'une donnée communiquée ayant le même nom), "TC.a" désigne le type de communication associé à "a". Lorsqu'aucune confusion ne sera possible, et pour ne pas parler d'une donnée communiquée particulière, nous noterons par abus de langage "TC" un type de communication en confondant l'application et son image.

2-1-1-Définition-

Comme nous l'avons vu au précédent chapitre, le type de communication associé à une donnée communiquée doit permettre l'expression d'une relation entre la suite des valeurs de cette donnée, émises par un processus "producteur"

(par activations d'une opération de production), et la suite des valeurs de cette donnée, reçues par un processus "consommateur" (par activations d'une opération de consommation). La définition de cette relation fait intervenir trois ensembles: l'ensemble produit, l'ensemble consommable et l'ensemble consommé.

Un objet "tc" de type "TC" est donc constitué d'un triplet d'objets, chacun d'eux étant du type "ensemble-indexé", défini algébriquement au paragraphe 2-1-2-ci-dessous.

Chaque élément de ce triplet est accessible par une fonction de projection, de "TC" dans "ensemble-indexé", notées respectivement "SP", "A" et "SC". Nous confondrons dans la suite, également, ces fonctions et les trois éléments d'un objet de type "TC", que nous noterons:

tc = < SP , A , SC >

Permettons nous une parenthèse pour justifier le choix de ces identificateurs. Nous présentons les trois objets composant un type de communication comme des objets d'un même type, permettant un accès aléatoire à leurs éléments (cf. paragraphe 2-1-2-); cependant, s'il est bien clair que, pour certaines situations, un tel accès aléatoire aux éléments de l'ensemble consommable est nécessaire, le traitement des ensembles produit et consommé est dans une large part identifiable à un traitement sur une suite. D'où les identificateurs "SP" (pour suite produite) et "SC" (pour suite consommée); ces deux termes pouvant être employés dans le développement qui suit, s'ils simplifient l'expression.

Sur cette structure on définit, outre les projections précédentes, les opérations élémentaires suivantes:

- init : ---> TC créé un objet de type TC, défini par:

init () = < créer(), créer(), créer() >

où "créer" est une opération du type "ensemble-indexé" défini ci-dessous.

- prod : TC * elt ---> TC où "elt" est le type de la donnée

communiquée.
 Cette opération prend en compte
 la production d'une valeur et
 définit l'objet suivant:

prod (<SP,A,SC>,p) =
 < ajouter(SP,p),apparaître(A,maximum(SP)+1,p),SC >

où "ajouter" et "apparaître" sont
 des opérations du type
 "ensemble-indexé" défini
 ci-dessous.

- pré-cons : TC ---> booléen restreint le domaine de
 définition de l'opération de
 consommation du processus
 consommateur.
- val-cons : TC ---> elt définit la valeur de la donnée
 communiquée prise en compte par
 l'opération de consommation.
- post-cons : TC ---> TC définit l'état de l'ensemble
 consommable, résultant d'une
 opération de consommation, et
 ajoute le terme
val-cons(<SP,A,SC>) à l'ensemble
 consommé, en définissant l'objet
 suivant:

post-cons (<SP,A,SC>) =
 <SP,A',ajouter(SC,val-cons(<SP,A,SC>))>

Ces trois dernières opérations sont caractéristiques d'un
 type de communication. Nous en verrons des exemples au
 paragraphe 2-1-4-, où chaque type de communication est résumé
 par une présentation de ce genre:

pré-cons (<SP,A,SC>) = ...
val-cons (<SP,A,SC>) = ...
post-cons (<SP,A,SC>) = ...

En fait, dans la définition de l'opération post-cons,
 seul l'ensemble consommable résultant (le deuxième terme A'
 du triplet) est caractéristique du type de communication.
 Dans la suite nous noterons, par abus de langage, seulement

ce second terme comme résultat de l'opération post-cons.

Remarque-

La présentation intuitive faite au chapitre 1 et le
 style impératif pour l'expression des processus peuvent
 suggérer au lecteur un point de vue dynamique -voire
 d'implantation- qui le conduirait à une interprétation
 erronée des opérations ci-dessus, en particulier de
 l'asymétrie apparente entre la "production" et la
 "consommation".

En fait, il faut rappeler qu'un type de communication
 est une structure abstraite pour exprimer des relations
 statiques entre des ensembles de valeurs (ensembles produit,
 consommable et consommé). Ces ensembles sont construits par
 les opérations

- init
- prod (qu'on pourrait noter post-prod par goût de la
 symétrie)
- post-cons

qui sont définies à partir des constructeurs "créer" et
 "apparaître" du type "ensemble-indexé".

Le constructeur "apparaître" a un argument de type
 "elt":

- dans le cas de l'opération prod cet argument est explicite,
- dans le cas de l'opération post-cons il est le résultat de
 la fonction val-cons, dont le domaine de définition est
 déterminé par la fonction pré-cons.

2-1-2-Le type "ensemble-indexé"-

Nous définissons le type "ensemble-indexé" à partir
 duquel sont définis les types de communication.

Un objet de ce type est constitué de couples (i,v), où "i"
 est un "entier_positif" et "v" est de type "elt". On suppose
 dans la suite que le type "entier_positif" désigne l'ensemble
 $\mathbb{N} \cup \{+\infty\}$, dont "+∞" est le plus grand élément, c'est-à-dire
 $\forall i \in \mathbb{N}, i < +\infty$. Intuitivement, dans la définition d'un type de
 communication, "i" est l'indice (entier positif) de "v"
 (valeur de la donnée communiquée) dans la suite produite.

Parce que, dans certaines situations, ces couples doivent être directement accessibles, nous définissons le type "ensemble-indexé" à partir du constructeur de type "table" classique, paramétré par les types des éléments des couples:

```
type ensemble-indexé = table ( entier_positif , elt )
```

Définition du type "table"-

Nous définissons le type paramétré table(t1,t2), où "t1" est un ensemble muni d'une relation d'ordre total admettant un plus petit et un plus grand élément.

Nous exprimons cette définition sous la forme classique d'un type abstrait algébrique (voir, parmi une littérature nombreuse (LiZ_75), (GTW_77), (GuH_78), (Ber_79'), (Gau_80), (GuH_80), etc.):

```
type table ( t1 , t2 )
```

opérations

```
table_vide : ---> table
mettre : table * t1 * t2 ---> table
enlever : table * t1 ---> table
accès : table * t1 ---> t2 ∪ {ω}
-- ω est supposé ne pas être de
-- type t2 --
domaine : table * t1 ---> ens_ord(t1)
-- ce constructeur "ens_ord"
-- paramétré par t1 définit le type des
-- sous-ensembles finis de l'ensemble
-- ordonné t1 --
```

axiomes soient $e_1, e'_1 \in t_1; e_2 \in t_2; t \in \text{table}(t_1, t_2)$

```
enlever(table_vide(), e1) = table_vide()
enlever(mettre(t, e1, e2), e'1) = si e1=e'1 alors t
sinon mettre(enlever(t, e'1), e1, e2)
fsi
accès(table_vide(), e1) = ω
accès(mettre(t, e1, e2), e'1) = si e1=e'1 alors e2
sinon accès(t, e'1)
fsi
domaine(table_vide()) = ens_vide()
domaine(mettre(t, e1, e2)) = union(domaine(t), e1)
```

fin type.

Cette définition importe les opérations du type ens_ord(t1), en particulier "ens_vide" et "union".

Définition du type "ens_ord"-

Nous définissons le type ens_ord(t), paramétré par un type "t" totalement ordonné (On note "<" sa relation d'ordre), ayant un plus petit élément noté "1" et un plus grand élément noté "T". Il définit l'ensemble des sous-ensembles finis de "t":

```
type ens_ord ( t )
```

opérations

```
ens_vide : ---> ens_ord
union : ens_ord * t ---> ens_ord
différence : ens_ord * t ---> ens_ord
appartient_à : ens_ord * t ---> booléen
est_vide : ens_ord ---> booléen
card : ens_ord ---> entier
min : ens_ord ---> t
max : ens_ord ---> t
```

axiomes soient $e, e' \in t; E \in \text{ens_ord}(t)$

```
différence(ens_vide(), e) = ens_vide()
différence(union(E, e), e') = si e=e'
alors différence(E, e')
sinon union(différence(E, e'), e)
fsi
appartient_à(ens_vide(), e) = faux
appartient_à(union(E, e), e') = si e=e' alors vrai
sinon appartient_à(E, e')
fsi
est_vide(ens_vide()) = vrai
est_vide(union(E, e)) = faux
card(ens_vide()) = 0
card(union(E, e)) = si appartient_à(E, e)
alors card(E)
sinon card(E)+1
fsi
min(ens_vide()) = 1
min(union(E, e)) = si e<min(E) alors e
sinon min(E)
fsi
max(ens_vide()) = 1
```

```

max(union(E,e)) = si max(E) < e alors e
                  sinon max(E)
                  fsi

```

fin type.

Opérations du type "ensemble-indexé"

Sur la structure "table(entier_positif,elt)" on définit les opérations suivantes:

```

- créer : ---> ensemble-indexé
- apparaître : ensemble-indexé * entier_positif * elt --->
                  ensemble-indexé
  -- cette opération ajoute à l'ensemble l'élément dont
  -- l'indice est donné --
- disparaître : ensemble-indexé * entier_positif --->
                  ensemble-indexé
  -- cette opération supprime de l'ensemble l'élément
  -- dont l'indice est donné --
- extraire : ensemble-indexé * entier_positif ---> elt
  -- cette opération rend l'élément dont l'indice est
  -- donné --
- dans? : ensemble-indexé * entier_positif ---> booléen
- vide? : ensemble-indexé ---> booléen
- cardinal : ensemble-indexé ---> entier
- maximum : ensemble-indexé ---> entier_positif
- minimum : ensemble-indexé ---> entier_positif

```

en fonction des opérations importées du type "table", par:

```

- créer() = table_vide()
- apparaître(s,i,v) = mettre(s,i,v)
- disparaître(s,i) = enlever(s,i)

  et les cas particuliers:

  - ajouter(s,v) = apparaître(s,max(domaine(s))+1,v)
  - décapiter(s) = disparaître(s,min(domaine(s)))

- extraire(s,i) = accès(s,i)

  et les cas particuliers:

  - premier(s) = extraire(s,min(domaine(s)))
  - dernier(s) = extraire(s,max(domaine(s)))

- dans?(s,i) = appartient_à(domaine(s),i)

```

```

- vide?(s) = (card(domaine(s))=0)
- cardinal(s) = card(domaine(s))
- maximum(s) = max(domaine(s))
- minimum(s) = min(domaine(s))

```

Cet ensemble d'opérations considère un objet de type "ensemble-indexé"

- comme une "table" lorsqu'on veut accéder à un élément quelconque- par exemple pour l'utilisation de l'ensemble consommable A pour une opération de consommation- (opérations "disparaître", "extraire", "dans?" ou "vide"),

- ou comme une "suite ordonnée" lorsqu'on ajoute un élément en fin de suite (opération "ajouter")- pour l'utilisation de l'ensemble produit SP pour une opération de production, ou de l'ensemble consommé SC pour une opération de consommation, ou dans des cas particuliers d'utilisation de l'ensemble A pour certaines opérations de consommation (opérations "décapiter", "premier", "dernier").

On trouvera dans cette thèse d'autres exemples d'opérations définies sur le type "ensemble-indexé" à partir de ces opérations et adaptées aux solutions de différents problèmes.

Remarque-

La notion de type algébrique, dans un domaine voisin de celui de l'étude présentée ici, est également utilisée dans (Ber_82) pour la spécification de protocoles de communication pour des systèmes répartis:

- Un protocole est spécifié, comme un type de communication, par un type produit cartésien:

- . un ensemble de sélecteurs (analogues à "SP", "A" ou "SC") permettent d'accéder aux composantes des objets de ce type,
- . ces objets sont construits par des opérations analogues à prod ou post-cons.

- Deux différences "techniques" avec l'approche développée ici:

- . le nombre de composantes est variable selon le protocole spécifié. Ceci est justifié par le domaine traité: échanges complexes de signaux pour réaliser une communication,
- . les composantes sont définies à partir d'un type

"séquence de messages", moins riche que le type "ensemble-indexé" ci-dessus. Ceci est également justifié par le domaine traité: la définition d'un type de communication peut requérir un accès aléatoire, voire à certaines valeurs des ensembles.

L'objectif de ces deux approches semble bien complémentaire:

- d'une part valider la notion générale de spécification algébrique comme un niveau d'abstraction pertinent pour la définition de protocoles,
- d'autre part proposer un cadre formel précis pour induire un mode de raisonnement pour la conception de systèmes parallèles.

2-1-3-Exemples de types de communications-

On propose ici une suite d'exemples de types de communications relativement simples, montrant l'utilisation des opérations précédentes. Ils correspondent, pour la plupart, à des situations tout à fait réelles, qui seront illustrées dans cette thèse.

- Type de communication "égalité":

Ce type exprime une relation d'égalité, telle que celle mise en évidence au paragraphe 1-4-2-, entre les suites produites et consommées: c'est-à-dire que toute valeur émise est reçue une et une seule fois, et l'ordre des réceptions est identique à l'ordre des émissions. Ce type est caractérisé par:

```

pré-cons(<SP,A,SC>) = non vide?(A)
val-cons(<SP,A,SC>) = premier(A)
post-cons(<SP,A,SC>) = décapiter(A)

```

- Type de communication "aléatoire":

Ce type exprime une relation telle que chaque valeur reçue est la dernière émise à cet instant: ainsi, certaines valeurs ne sont pas reçues et d'autres peuvent l'être

plusieurs fois, selon le "rythme" aléatoire de chacun des processus producteur ou consommateur.

Pour cela on définit une nouvelle opération sur le type "ensemble-indexé":

```

effacer : ensemble-indexé * entier_positif --->
                                         ensemble-indexé

```

qui supprime tous les éléments d'indice inférieur ou égal à l'entier positif donné. Cette opération est définie par:

```

effacer(s,i) = si i=minimum(s) alors décapiter(s)
               sinon si dans?(s,i)
                 alors effacer(décapiter(s),i)
                 sinon si i<minimum(s)
                   alors s
                   sinon effacer(s,i-1)
               fsi
               fsi

```

Dont des cas particuliers intéressants sont:

- finir(s) = effacer(s,maximum(s)-1)
- tuer(s) = effacer(s,maximum(s))

On définit alors le type "aléatoire" par:

```

pré-cons(<SP,A,SC>) = non vide?(A)
val-cons(<SP,A,SC>) = dernier(A)
post-cons(<SP,A,SC>) = finir(A)

```

- Type de communication "très-aléatoire":

Ce type est une généralisation du précédent en permettant une consommation avant la première opération de production. Il est défini par:

```

pré-cons(<SP,A,SC>) = vrai
val-cons(<SP,A,SC>) = si vide?(A) alors ω
                     sinon dernier(A)
                     fsi
post-cons(<SP,A,SC>) = finir(A)

```

- Type de communication "rafraichie":

Ce type exprime une relation semblable à celle décrite par le type "aléatoire", mais où toute valeur reçue doit être "rafraichie": c'est-à-dire qu'entre deux consommations successives de la donnée communiquée il doit y avoir au moins une nouvelle production de valeur de cette donnée.

```

pré-cons(<SP,A,SC>) = non vide?(A)
val-cons(<SP,A,SC>) = dernier(A)
post-cons(<SP,A,SC>) = tuer(A)

```

- Type de communication "égalité-presque-partout":

Ce type exprime que toute valeur émise est reçue une fois, et une seule, si l'opération de production survient "assez tôt". Plus précisément, il y a égalité des suites produites et consommées, si ce n'est que le ième élément produit est remplacé par "w" dans la suite consommée, si la production de cet élément est en retard sur la ième consommation.

```

pré-cons(<SP,A,SC>) = vrai
val-cons(<SP,A,SC>) = extraire(A,cardinal(SC)+1)
post-cons(<SP,A,SC>) = effacer(A,cardinal(SC)+1)

```

- Type de communication "rafraichie-avec-trous":

Ce type exprime qu'au cas où la donnée communiquée n'aurait pas été rafraichie, la valeur reçue est w ; dans le cas contraire la dernière valeur produite est consommée:

```

pré-cons(<SP,A,SC>) = vrai
val-cons(<SP,A,SC>) = si vide?(A) alors w
                        sinon dernier(A)
                        fsi
post-cons(<SP,A,SC>) = si vide?(A) alors A
                        sinon tuer(A)
                        fsi

```

- Type de communication "égalité-avec-trous":

Différent de "l'égalité-presque-partout", ce type est

tel que toute valeur émise est reçue; "entre-temps" le processus consommateur peut recevoir la valeur w, lorsqu'aucune nouvelle valeur n'est émise.

```

pré-cons(<SP,A,SC>) = vrai
val-cons(<SP,A,SC>) = si vide?(A) alors w
                        sinon premier(A)
                        fsi
post-cons(<SP,A,SC>) = si vide?(A) alors A
                        sinon décapiter(A)
                        fsi

```

- Type de communication "à-retard-borné":

Dans la situation décrite par ce type, toute valeur reçue est celle émise le plus récemment. Cependant le "retard" pris par le processus producteur reste borné: c'est-à-dire que le nombre d'occurrences de consommation moins le nombre d'occurrences de production est borné par un entier "p":

```

pré-cons(<SP,A,SC>) = non vide?(A)
val-cons(<SP,A,SC>) = dernier(A)
post-cons(<SP,A,SC>) = si cardinal(SC)-maximum(A)<p
                        alors finir(A) sinon tuer(A)
                        fsi

```

- Type de communication "échantillon":

La relation décrite est telle que le processus consommateur prélève la plus récente valeur produite, une seule fois, en n'ignorant pas plus de "p" valeurs produites consécutivement.

On définit le prédicat suivant sur le produit cartésien ensemble-indexé * entier_positif:

```

P(s,i) = minimum(s)<i<maximum(s)
          et non dans?(s,i)
          et j ∈ [i+1,maximum(s)] ==> dans?(s,j)

```

-- définit le plus grand entier "i" dans le domaine de "s", s'il existe, correspondant à une consommation antérieure

--

On définit la fonction suivante de "ensemble-indexé" dans "entier":

```
q(s) = si existe(i) telque P(s,i) alors i
      sinon minimum(s)-1
      fsi
```

```
-- définit le plus grand entier correspondant à une
-- consommation antérieure
```

Le type de communication "échantillon" est alors défini par:

```
pré-cons(<SP,A,SC>) = non vide?(A)
val-cons(<SP,A,SC>) = si maximum(A)-q(A)<p
                      alors dernier(A)
                      sinon extraire(A,q(A)+p)
                      fsi
post-cons(<SP,A,SC>) = si maximum(A)-q(A)<p
                      alors retirer(A,maximum(A))
                      sinon retirer(A,q(A)+p)
                      fsi
```

- Type de communication "élastique":

La relation décrite par ce type de communication est telle que toute valeur émise est reçue une fois, et une seule, dans l'ordre des émissions, sauf lorsque le processus consommateur prend un retard supérieur à "p": dans ce cas toutes les valeurs émises sont "sautées" jusqu'à la dernière.

```
pré-cons(<SP,A,SC>) = non vide?(A)
val-cons(<SP,A,SC>) = si cardinal(A)<p
                      alors premier(A) sinon dernier(A)
                      fsi
post-cons(<SP,A,SC>) = si cardinal(A)<p
                      alors décapiter(A) sinon tuer(A)
                      fsi
```

- Type de communication "circulaire":

La relation décrite par ce type de communication est

semblable à la précédente, mais où, en cas de retard du processus consommateur, la valeur reçue est la plus anciennement émise "modulo" un cycle de taille "p":

```
pré-cons(<SP,A,SC>) = non vide?(A)
val-cons(<SP,A,SC>) =
  extraire(A,minimum(A)+k(maximum(A)-minimum(A),p)*p),
  avec k(x,p) = si p>=x alors 0
               sinon k(x-p,p)+1
               fsi
```

```
post-cons(<SP,A,SC>) =
  moduler(A,minimum(A)+k(maximum(A)-minimum(A),p)*p)
```

où l'opération "moduler"

```
moduler : ensemble-indexé * entier_positif --->
          ensemble-indexé
```

est définie sur tout objet "s" de type "ensemble-indexé" et tout entier_positif "i" par:

```
moduler(s,i) = si dans?(s,i)
               alors moduler(disparaître(s,i),i-p)
               sinon s
               fsi
```

- Type de communication "pile":

La relation décrite par ce type est telle que la plus récente valeur, parmi les valeurs consommables, est consommée une seule fois et retirée de cet ensemble:

```
pré-cons(<SP,A,SC>) = non vide?(A)
val-cons(<SP,A,SC>) = dernier(A)
post-cons(<SP,A,SC>) = disparaître(a,maximum(A))
```

- Type de communication "croissante":

On suppose que le type "elt" est muni d'une relation d'ordre, notée "<": ce type de communication est tel que les valeurs émises sont reçues une seule fois, dans l'ordre de leur production, à la condition de former une suite croissante de valeurs de type "elt".

On définit le prédicat suivant, sur le produit cartésien ensemble-indexé * ensemble-indexé * entier_positif:

```
P(s1,s2,i) = non vide?(s2)
              et dans?(s1,i)
              et dernier(s2)<extraire(s1,i)
              et 7 j<i dans?(s1,j) ==>
                  extraire(s1,j)=<dernier(s2)
```

On définit la fonction suivante de ensemble-indexé * ensemble-indexé dans booléen:

```
q(s1,s2) = existe(i) telque P(s1,s2,i)
```

et la fonction suivante, de ensemble-indexé * ensemble-indexé dans entier_positif:

```
r(s1,s2) = si existe(i) telque P(s1,s2,i) alors i fsi
```

```
-- détermine le plus petit indice "i" du domaine de
-- "s1", s'il existe, tel que l'élément correspondant de
-- "s1" soit supérieur au dernier élément de "s2"
```

Le type de communication "croissante" est alors défini par:

```
pré-cons(<SP,A,SC>) = q(A,SC)
val-cons(<SP,A,SC>) = ou vide?(SC) et dans?(A,1)
                    si q(A,SC)
                    alors extraire(A,r(A,SC))
                    sinon premier(A)
                    fsi
post-cons(<SP,A,SC>) = si q(A,SC)
                    alors effacer(A,r(A,SC))
                    sinon décapiter(A)
                    fsi
```

- Type de communication "non-répétitive":

La relation définie par ce type de communication est telle qu'il n'y ait pas de série de plus de "p" fois la même valeur dans la suite consommée, ces valeurs étant reçues dans l'ordre de leur émission.

On définit pour cela l'opération suivante:

- même : ensemble-indexé --> entier

qui compte le nombre d'éléments consécutifs de l'ensemble, identiques à son dernier élément.

Soit s de type "ensemble-indexé",

```
- même(s) = si cardinal(s)>1
              alors si extraire(s,maximum(s)-1)=dernier(s)
                  alors même(disparaitre(s,maximum(s)))+1
                  sinon 1
              fsi
              sinon cardinal(s)
              fsi
```

Nous introduisons les prédicats suivants, sur le produit cartésien ensemble-indexé * ensemble-indexé:

```
P(s1,s2) = non vide?(s1)
            et premier(s1)<>dernier(s2)
            ou cardinal(s2)<p
            ou cardinal(s2)>=p et même(s2)<p
```

```
-- P est à vrai si "s2" ne se termine pas par "p"
-- éléments identiques au premier élément de "s1"
```

```
Q(s1,s2) = si non P(s1,s2) et non vide?(décapiter(s1))
            alors Q(décapiter(s1),s2)
            sinon P(s1,s2)
            fsi
```

```
-- Q est à vrai s'il existe un élément dans "s1" qui ne
-- soit pas égal aux "p" derniers éléments de "s2",
-- supposés identiques
```

Le type de communication "non-répétitive" est alors défini par:

```
pré-cons(<SP,A,SC>) = P(A,SC) ou Q(A,SC)
val-cons(<SP,A,SC>) = extraire(A,k(A,SC))
```

```
avec k(s1,s2) = si premier(s1)<>dernier(s2)
                alors minimum(s1)
                sinon k(décapiter(s1),s2)
                fsi
```

```
post-cons(<SP,A,SC>) = effacer(A,k(A,SC))
```

2-2- EXPRESSION D'UN SYSTEME PARALLELE-

L'exemple introductif de ce chapitre a montré intuitivement la forme générale d'expression d'un système parallèle. Cette expression est précisée ici.

Les processus élémentaires constituant un système parallèle sont des processus séquentiels non-déterministes classiques (énoncés de séquence, choix non-déterministe, itération, ...), dont l'activation en parallèle est définie explicitement par une instruction de "composition parallèle".

La composition parallèle de processus définit un nouveau processus: une liste de nom de ports, et pour chacun d'eux le type de communication associé, définit les relations de communication de ces processus.

Un processus communicant comporte une "liste d'entrée" et/ou une "liste de sortie":

- Une liste d'entrée est constituée de noms de données typées, et pour chaque donnée le port utilisé pour transmettre les valeurs de cette donnée.

- Une liste de sortie est constituée de noms de données typées, et pour chaque donnée, la liste des ports utilisés pour transmettre les valeurs de cette donnée.

On appelle donnée communiquée une donnée figurant dans une liste d'entrée ou une liste de sortie.

Toute communication est exprimée par l'activation d'une opération "produire" ou "consommer" sur une donnée communiquée.

2-2-1-Définition sommaire du langage-

On donne une partie de la grammaire définissant la

syntaxe du langage de description des systèmes parallèles, dont la sémantique sera précisée au paragraphe 2-2-3- et au chapitre 4.

```

PROCESSUS ---> SYSTEME | PROC_COMMUNICANT
SYSTEME ---> processus IDENT :: SYST
PROC_COMMUNICANT ---> processus IDENT :: PROC

SYST ---> (COMMUNICATION)01 PROC_COMMUNICANT
          (PROC_COMMUNICANT)* COMPOSITION
COMMUNICATION ---> port LISTE_PORTS ;
                  -- précédé de la définition des types de
                  -- communication nécessaires -si ceux-ci
                  -- ne sont pas prédéfinis- sous la forme
                  -- présentée ci-dessus

PROC ---> PROC_SEQ | SYST
PROC_SEQ ---> DONNEES_COM DECLARATIONS CORPS
CORPS ---> corps ENONCES fin IDENT ;
DONNEES_COM ---> ENTREE [ SORTIE ] ENTREE SORTIE
ENTREE ---> entrée LISTE_ENTREE ;
SORTIE ---> sortie LISTE_SORTIE ;
LISTE_PORTS ---> ID_PORT ( ID_PORT ) * ; TYPE_COM ;
                  (ID_PORT ( ID_PORT ) * ; TYPE_COM ; ) *
                  -- on admettra des listes avec nom
                  -- indicé, l'indice variant dans un
                  -- intervalle précis

LISTE_ENTREE ---> ID_DON : TYPE via ID_PORT ;
                  (ID_DON : TYPE via ID_PORT ; ) *
                  -- un même nom de port peut apparaître
                  -- dans des listes d'entrée de différents
                  -- processus communicants: ceci définit
                  -- implicitement plusieurs ports (un par
                  -- processus consommateur) sur lesquels
                  -- sont diffusées les valeurs transmises,
                  -- avec les mêmes propriétés

LISTE_SORTIE ---> ID_DON : TYPE vers LISTE_ID_PORT ;
                  (ID_DON : TYPE vers LISTE_ID_PORT ; ) *
                  -- les clauses via ou vers peuvent être
                  -- omises: dans ce cas, la donnée
                  -- communiquée et le port utilisé ont le
                  -- même nom

COMPOSITION ---> composition IDENT (// IDENT) *
                  fin IDENT ;
DECLARATIONS --->

```

```

-- déclarations classiques de données,
-- procédures, etc.
--
ENONCES --->
-- séquence, notée i d'énoncés du type
-- suivant:
--
VIDE --->
AFFECTATION ---> ID := EXP
PRODUCTION ---> produire( ID_DON )
CONSUMMATION ---> consommer( ID_DON )
CHOIX ---> [ GARDE --> ENONCES
( [ GARDE --> ENONCES ] * ) ]
-- où GARDE est une expression booléenne
--
ITERATION ---> * [ GARDE --> ENONCES ]
-- une GARDE constante à vrai peut être
-- omise (cycle infini)
--

```

Nous avons illustré cette grammaire, en avant-propos de ce chapitre, pour un exemple simple de système producteur-consommateur. Nous pouvons maintenant le préciser comme suit:

```

processus PROD_CONS ::
  port a: aléatoire;
  processus PROD ::
    sortie x: ... vers a;
    corps
    .
    .
    produire(x) ;
    .
  fin PROD;
  processus CONS ::
    entrée y: ... via a;
    corps
    .
    .
    consommer(y) ;
    .
  fin CONS;

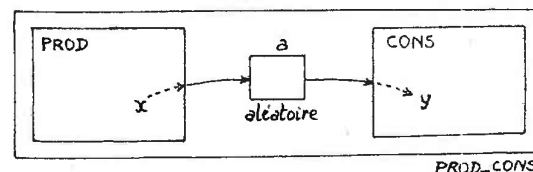
```

composition

PROD // CONS

fin PROD_CONS;

et le schématiser ainsi:



2-2-2-Hiérarchisation de processus-

Revenons sur quelques points concernant la hiérarchisation des processus, c'est-à-dire sur le fait qu'un système parallèle peut, à son tour, être considéré comme un processus communicant dans un système "englobant".

Situation de diffusion-

Les exemples précédents concernent des communications dites "bipoint" (Ray_81) entre processus, éventuellement généralisées à une même donnée communiquée vers plusieurs ports auxquels pourraient être associés des types de communication différents. En tout état de cause, ce cas de "diffusion" est défini comme un ensemble de communications "bipoint", de la manière suivante:

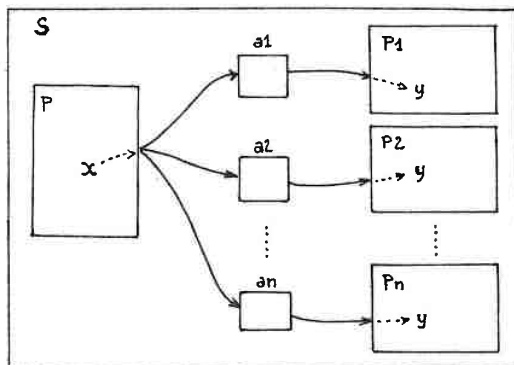
Premier cas:

Diffusion vers "n" processus, les types de communication associés aux "n" ports sont identiques:

```

processus S ::
  port (ai: -- type de communication ; i=1,n);
  processus P ::
    sortie x: ... vers (ai;i=1,n);
    corps
    .
    .
    produire(x) ;
    .
  fin Pi
  pour tout i de 1 à n:
    processus Pi ::
      entrée y i ... via ai;
      corps
      .
      .
      consommer(y) ;
      .
    fin Pi;
  composition
    P (// Pi;i=1,n);
fin Si

```



Remarque-

Si aucune confusion ne peut en résulter on pourra noter un seul nom de port. De plus on pourra confondre le nom de ce port et des données communiquées qui l'utilisent:

```

processus S ::
  port a: ... ;
  processus P ::
    sortie a: ... ;
    corps
    .
    .
    produire(a) ;
    .
  fin Pi
  pour tout i de 1 à n:
    processus Pi ::
      entrée a: ... ;
      corps
      .
      .
      consommer(a) ;
      .
    fin Pi;
  composition
    P (// Pi;i=1,n);
fin Si

```

Ce système définit "n" ports distincts, auxquels sont associés "n" objets "tci", de même type "TC", caractérisés par un triplet $tci = \langle SPi, Ai, SCi \rangle$, pour tout i de 1 à n .

Deuxième cas:

Le nombre d'interlocuteurs est défini statiquement; les types de communication peuvent être différents.

Exemple:

```

processus S ::
  port a: ... i
    b: ... i
  processus P ::
    sortie x: ... vers a, b;
    .
  fin P;
  processus Q ::
    entrée y: ... via a;
    .
  fin Q;
  processus R ::
    entrée z: ... via b;
    .
  fin R;
  composition
    P // Q // R;
fin S;

```

Situation de divergence-

Utilisant le fait qu'un système parallèle peut être considéré comme un processus communicant, composant d'un autre système parallèle, il peut définir des données communiquées permettant sa liaison avec d'autres processus. Chaque nom apparaissant dans sa "liste d'entrée" définit une donnée communiquée associée à un port auquel est associé un type de communication généralisé, dont tout objet est formé

- . d'un ensemble produit "SP",
- . d'un ensemble consommable "A",
- . d'un n-uplet d'ensembles consommés "SCi", $i \in [1, n]$, chaque ensemble consommé étant associé à un des processus formant le

système.

Du point de vue sémantique, les processus constituant un système respectent les règles suivantes:

. ils peuvent comporter des opérations de communication "internes" au système,

. ils peuvent comporter des opérations de communication concernant des ports définis au niveau du système: le ième processus modifie alors les éléments "SP", "A" et "SCi" de l'objet de type "TC" associé, selon les règles précédentes.

Exemple:

Le système producteur-n consommateurs suivant décrit une situation de "divergence", où toutes les valeurs produites sont consommées par un des processus consommateurs, au hasard de leur activation:

```

processus S ::
  port x: égalité;
  processus P ::
    sortie x: ... i
    corps
      .
      produire(x) i
      .
  fin P;
  processus C ::
    pour tout i de 1 à n:
      processus Ci ::
        entrée x: ... i
        corps
          .
          consommer(x) i
          .
      fin Ci;

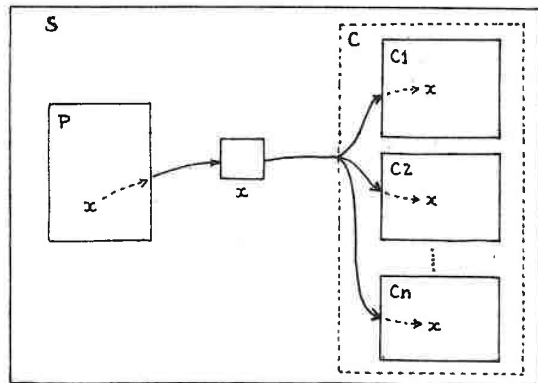
```

```

composition
    C1 (// Ci;i=2,n);
fin Ci
composition
    P // C
fin Si

```

Remarquons que la signification de ce type de communication "égalité" est globale pour le système "C": il y a égalité entre l'ensemble des valeurs produites et la réunion des ensembles de valeurs consommées. Il ne s'agit pas d'une égalité pour chaque processus.



Situation de convergence:

La description d'un système "à convergence" ne se pose pas en des termes identiques: en effet, on doit exprimer la sélection d'un élément consommé dans l'ensemble des éléments consommables définis pour chaque processus producteur. Ceci peut être exprimé de deux façons différentes:

- soit par regroupement des processus producteurs dans un système producteur et l'introduction d'un type de

communication généralisé, où est défini un n-uplet d'ensembles produits et consommables,

- soit par un "mélange" de ces ensembles en un ensemble unique, pour former un type de communication classique: la sélection dans cet ensemble suppose d'y faire figurer, pour chaque élément, un identificateur discriminant associé à chaque processus producteur. Une telle expression est illustrée au paragraphe 3-2-2-.

2-2-3-Sémantique intuitive d'un système parallèle-

Nous présenterons au chapitre 4 une sémantique formelle du langage défini ci-dessus. Nous nous contentons ici d'une approche intuitive permettant de comprendre la signification des exemples présentés au chapitre 3 suivant.

Rappelons qu'un "canal de communication" définit l'association de deux données communiquées "x" et "y" et d'un port "a":

$c=(x,a,y)$

où "x" est une donnée communiquée figurant dans une liste de sortie

sortie x: ... vers a;

"y" est une donnée communiquée figurant dans une liste d'entrée

entrée y: ... via a;

"a" est un port défini dans une clause

port a: TC;

où "TC" est un type de communication.

Initialisation:

L'ensemble des déclarations de "x", "y", et "a" ont pour effet la création d'un objet "tc" de type "TC" par l'opération init.

Production:

Si "tc" est l'objet de type "TC" associé au port "a" avant l'activation de l'énoncé produire(x), l'état résultant de cette opération est le suivant:

- . l'état mémoire du processus producteur est inchangé,
- . si VAL.x dénote la valeur associée à la donnée "x", prod(tc,VAL.x) est le nouvel objet associé au port "a".

Consommation:

Soit "tc" l'objet de type "TC" associé au port "a" au moment d'une requête d'activation de l'énoncé consommer(y)

- si la valeur pré-cons(tc) est vrai l'opération de consommation peut être activée,
- si la valeur pré-cons(tc) est faux cette activation est retardée jusqu'à la définition -par l'activation d'autres opérations du système- d'un objet "tc'" tel que pré-cons(tc') est vrai: la définition de cette "attente" est précisée au chapitre 4.

L'état résultant de l'activation de l'opération de consommation est le suivant (on note dans tous les cas "tc'" l'objet associé au port "a" avant l'activation de cette opération):

- . l'état mémoire du processus consommateur est modifié comme suit:

VAL.y = val-cons(tc')

- . post-cons(tc') est le nouvel objet associé au port "a".

CHAPITRE 3- EXEMPLES DE SYSTEMES PARALLELES

Dans ce chapitre, on propose un ensemble d'exemples pour illustrer les notions introduites au chapitre précédent. Ces exemples vont de problèmes "jouets" à des algorithmes dont l'intérêt pratique est réel. Ils sont volontairement simples, mais représentatifs de classes de problèmes différentes et de solutions algorithmiques pertinentes. Ils sont seulement une illustration de la notion de type de communication (ils reprennent certains des types "classiques" présentés au chapitre 2 ou en introduisent de nouveaux): les aspects complémentaires de conception et de vérification seront traités dans les chapitres ultérieurs.

3-1- EXEMPLE 1: CALCUL DE FACTORIELLE-

Etant donné "n", entier strictement positif, le nombre n! peut être défini comme le terme de rang "n" d'une des suites suivantes:

- f définie par
$$\begin{cases} f_0 = 1 \\ f_{i+1} = f_i * (i+1) \quad \forall i \geq 0 \end{cases}$$

(on a en effet $f_i = i! \quad \forall i \geq 0$)

- q définie par $\begin{cases} q_0 = 1 \\ q_{i+1} = q_i * (n-i) \quad \forall i \geq 0 \end{cases}$
 (on a en effet $q_i = n! / (n-i)! \quad \forall i \geq 0$)

On peut répartir les calculs sur deux intervalles disjoints, par exemple:

- le calcul des termes f_i de la suite f pour $0 \leq i \leq \lfloor n/2 \rfloor$
 (ce calcul a pour résultat $\lfloor n/2 \rfloor!$)
- le calcul des termes q_i de la suite q pour $0 \leq i \leq n - \lfloor n/2 \rfloor$
 (ce calcul a pour résultat $n! / \lfloor n/2 \rfloor!$)

On a alors l'énoncé suivant:

- soit f la suite définie par $\begin{cases} f_0 = 1 \\ f_{i+1} = f_i * (i+1) \quad \forall i \geq 0 \end{cases}$
- soit q la suite définie par $\begin{cases} q_0 = 1 \\ q_{i+1} = q_i * (n-i) \quad \forall i \geq 0 \end{cases}$
- le résultat $n!$ est égal à $f_{\lfloor n/2 \rfloor} * q_{n - \lfloor n/2 \rfloor}$

On peut donc construire une solution qui consiste en un système constitué de deux processus parallèles indépendants, qui calculent respectivement les termes des deux suites f et q . Le résultat est le produit de ces résultats.

Cet algorithme peut être généralisé de la manière suivante:

- le premier processus, disons "P1", calcule les termes de la suite f ,
 f_i pour $0 \leq i \leq n_1$
- le second processus, disons "P2", calcule les termes de la suite q ,
 q_j pour $0 \leq j \leq n_2$
- la condition d'arrêt est $n_1 \geq n - n_2$

Pour évaluer cette condition, les valeurs successives de " n_1 " et de " n_2 " doivent être échangées:

On note n_1 la suite des valeurs de " n_1 " produites par "P1" (le raisonnement qui suit est analogue pour la suite n_2 des valeurs produites par "P2"),

La suite des valeurs de " n_1 " consommées par "P2" est extraite de n_1 . On peut imaginer plusieurs relations de communication pourvu qu'elles satisfassent la condition (c), basée sur l'observation suivante: pour l'évaluation de la condition d'arrêt chacun des processus a seulement besoin de la dernière valeur déterminée par l'autre.

La condition (c) s'énonce alors:

- (c) lorsque "P1" termine, la valeur de " n_1 " consommée par "P2" est la dernière valeur produite par "P1".

L'association du type de communication "aléatoire" aux ports des données communiquées " n_1 " et " n_2 " satisfait évidemment la condition (c). On peut alors observer une exécution "asynchrone" des processus.

Du fait que chaque processus est à la fois producteur et consommateur, le type "égalité" (ou d'autres encore) convient également. Il a pour conséquence une exécution "synchrone" des processus.

Nous montrerons au chapitre 6 que ces deux solutions sont effectivement correctes. Elles définissent un système parallèle dont la post-condition est, si on note " f_1 " et " f_2 " les résultats respectifs de "P1" et "P2", et avec le changement de variable $n_2 = n - n_1$:

$$\{ f_1 = (n_1 - 1)! \text{ et } f_2 = n! / n_1! \text{ et } n_2 = n_1 = n_2 + 1 \}$$

Le nombre $n!$ est alors la valeur du résultat " f " du système parallèle suivant, présenté dans sa forme "asynchrone" (la forme "synchrone" ne différant que par la déclaration des types de communication "égalité" substituée à la déclaration des types "aléatoire" pour les ports de communication):

processus S ::

port n1, n2: aléatoire;
f1, f2: égalité;

processus P1 ::

entrée n2: entier ;
sortie n1, f1: entier ;
corps
f1:=1; n1:=1;
produire(n1) ; consommer(n2) ;
*[n1<n2 --> f1:=f1*n1; n1:=n1+1;
produire(n1) ; consommer(n2)

];
produire(f1)
fin P1;

processus P2 ::

entrée n1: entier ;
sortie n2, f2: entier ;
entier n;
corps
f2:=1; n2:=n;
produire(n2) ; consommer(n1) ;
*[n1<n2 --> f2:=f2*n2; n2:=n2-1;
produire(n2) ; consommer(n1)

];
produire(f2)
fin P2;

processus Q ::

entrée n1, n2, f1, f2: entier ;
entier f;
corps
consommer(f1) ; consommer(f2) ;
consommer(n1) ; consommer(n2) ;
[n1=n2 --> f:=f1*f2*n1
Q n1>n2 --> f:=f1*f2
Q

fin Q;

composition

P1 // P2 // Q

fin S;

3-2- EXEMPLE 2: CALCUL DU PLUS GRAND COMMUN DIVISEUR DE N NOMBRES-

3-2-1-Décomposition en facteurs premiers-

On propose dans (Per_81) l'énoncé suivant, pour le calcul du plus grand commun diviseur de "n" nombres entiers strictement positifs "a1", "a2", ..., "an", par décomposition de ces nombres en facteurs premiers:

$$- \text{pgcd} = \prod_{j=1, k} p_j^{t_{sj}}$$

- p: suite des nombres premiers (p1=2)

- s: suite des exposants de la décomposition de "pgcd" en facteurs premiers, définie par

$$s_j = \max\{s \text{ entier } \geq 0 \text{ tel que } p_j^s \text{ divise } a_i \forall i \in [1, n]\}$$

- k: entier tel que $\forall i \in [1, n] \quad k' > k \Rightarrow \text{non } (p_k' \text{ divise } a_i)$

La réalisation de cet énoncé peut être effectuée par un système parallèle constitué des processus suivants:

- un processus "P" calcule la suite "p" des nombres premiers,
- un processus "S" calcule la suite "s" des exposants,
- un processus "Q" calcule le résultat "pgcd".

Par propriété de l'opérateur "max" et de la conjonction, on peut substituer à la définition du terme général de la suite "s", les définitions suivantes:

$$- s_j = \min\{r_{ij}, i \in [1, n]\}$$

- $\forall i \in [1, n]$, r_i : suite des exposants de la décomposition de "ai" en facteurs premiers, définie par

$$r_{ij} = \max\{r \text{ entier } \geq 0 \text{ tel que } p_j^r \text{ divise } a_i\}$$

Le système parallèle obtenu comporte alors "n" processus "Pi" qui calculent les suites " r_i " des exposants.

La définition des suites dans l'énoncé précédent suppose l'introduction des communications suivantes:

- Chaque terme "pj" calculé par le processus "P" est utilisé une fois au plus par les processus "Pi" et "Q", dans l'ordre de sa production: cette suite détermine donc n+1 données communiquées, utilisant chacune un port associé au type de communication "égalité",

- De même, la suite "ri" détermine, pour tout i de 1 à n, une donnée communiquée utilisant un port associé au type de communication "égalité",

- Enfin, on détermine la valeur de "k" par la communication d'une suite de valeurs booléennes "ci" des processus "Pi" vers le processus "Q". La terminaison du processus "P" est obtenue par "consultation" d'une valeur booléenne "c" émise par "Q": ceci est réalisé par une donnée communiquée utilisant un port associé au type de communication "aléatoire".

On obtient finalement le système suivant:

processus S ::

port p₁ (ri, ci; i=1, n): égalité;
ci aléatoire;

processus P ::

entrée c: booléen;

sortie pi: entier;

corps

pi=2;

produire(p) ; consommer(c) ;

*if non c -->

-- calcul nombre premier suivant --
produire(p) ; consommer(c)

];

fin P;

pour tout i de 1 à n:

processus Pi ::

entrée p: entier;

sortie r: entier vers ri;

ci booléen vers ci;

entier ai;

corps

c:= faux;

consommer(p) ;

*if p < ai -->

ri:=0;

*if p divise ai --> ai:=ai/pi;

ri:=ri+1

];

produire(r) ; produire(c) ;

consommer(p)

];

c:= vrai;

produire(c)

fin Pi;

processus Q ::

entrée p₁ (ri; i=1, n): entier;

(ci; i=1, n): booléen;

sortie c: booléen;

entier pgcd, si;

corps

pgcd:=1; c:= faux;

consommer(p) ; (consommer(ci); i=1, n);

*if $\bigwedge_{i=1}^n$ non ci --> (consommer(ri); i=1, n);

s:=min(ri; i=1, n);

pgcd:=pgcd*(p s);

consommer(p) ;

(consommer(ci); i=1, n)

];

c:= vrai;

produire(c)

fin Q;

composition

P (// Pi; i=1, n) // Q

fin S;

On propose également dans (Per 81), une variante pour cet algorithme, en introduisant des communications entre les processus "Pi": pour chaque nombre premier "pj", chaque processus "Pi" transmet aux autres la valeur de l'exposant "rij". Alors, si la valeur de l'exposant correspondant au même nombre premier a déjà été transmise par un autre processus, le calcul d'un exposant peut être arrêté lorsque cette valeur est atteinte.

Cette situation est décrite par l'introduction de n données communiquées " ti " pour chaque suite " ri ", chacune utilisant un port associé au type de communication "égalité_presque_partout", où la valeur ω est ici la valeur maximum du type entier.

On complète la clause "port" par:

($ti; i=1, n$): égalité_presque_partout;

Les processus " Pi " sont modifiés, de la façon suivante:

```

processus Pi ::
  entrée pi: entier i
  (rj: entier via tj; j=1, n; j <> i);
  sortie ri: entier vers ri; (tj; j=1, n; j <> i);
  c: booléen vers ci;
  entier ai, si;
  corps
    c := faux;
    consommer(p);
    * [ p < ai --> (consommer(rj); j=1, n; j <> i);
      si := min(rj; j=1, n; j <> i);
      r := 0;
      * [ r < s et p divise ai -->
        ai := ai / p;
        r := r + 1;
      ];
    ];
    produire(r);
    consommer(p);
    ];
    c := vrai;
    produire(c);
fin Pi;

```

Ces communications sont semblables aux communications dites "optionnelles" dans (Pet_81'). Nous les retrouverons dans l'exemple 3 ci-dessous.

3-2-2-Algorithme d'Euclide-

On propose dans (AFR_80) une autre solution à ce même problème, basée sur l'algorithme classique d'Euclide.

Pour tout i de 1 à n , on associe un processus " Pi " au nombre entier strictement positif " ai ". Ces processus sont organisés "en cercle", c'est-à-dire que chacun d'eux, disons " Pi ", communique à ses deux "voisins" immédiats, c'est-à-dire $Pi-1$ et $Pi+1$, modulo n , sa valeur courante de " ai ". Lorsqu'il reçoit la valeur courante " $ai-1$ " ou " $ai+1$ " de l'un de ses voisins, et si cette valeur est inférieure à " ai ", " ai " est modifié par l'algorithme d'Euclide, et ainsi de suite. Le système décrit par (AFR_80) termine sur un état de blocage de tous les processus, qui ont tous la même valeur courante " ai " égale au plus grand commun diviseur des nombres " ai " initiaux.

Ces communications sont "contrôlées" par des communications "parasites", au sens déjà évoqué au chapitre 1 (paragraphe 1-4-2-), communications de valeurs booléennes, qui garantissent que les valeurs ne sont transmises que si elles sont demandées et si un nouveau calcul a eu lieu depuis la demande précédente émanant du même processus.

Autrement dit, pour chaque processus " Pi " la relation entre l'ensemble des valeurs successives de " ai " qu'il calcule et l'ensemble des valeurs utilisées par un de ses voisins est telle que:

- toute valeur consommée est la dernière calculée,
- deux consommations successives supposent au-moins une production intermédiaire.

Cette propriété est décrite par le type de communication "rafraichie". Pour tenir compte de la possibilité pour la valeur de provenir indifféremment de l'un quelconque des deux processus voisins, on définit le type de communication "mélange" suivant:

Ce type est défini à partir du type "ensemble-indexé" (cf. paragraphe 2-1-2-), dont le paramètre "elt" est supposé être un produit cartésien, dont on note " p " et " x " les projections, par les opérations suivantes:

```

pré-cons(<SP,A,SC>) = non vide?(A)
val-cons(<SP,A,SC>) = extraire(A,maximum(A))
post-cons(<SP,A,SC>) = gommer(A,maximum(A))

```

où intuitivement, l'opération "gommer" retire de l'ensemble consommable tous les éléments produits antérieurement par le même processus que celui dont l'indice est indiqué, ici le dernier élément. Cette opération est définie sur un objet "s" de type "ensemble-indexé" (satisfaisant l'hypothèse précédente) par:

```
gommer : ensemble-indexé * entier_positif
      --> ensemble-indexé

gommer(s,i) = si dans?(s,i)
             alors
               gommer(disparaître(s,i),réplique(s,i))
             sinon s
             fsi
```

où la fonction "réplique" détermine le plus grand indice inférieur à un entier positif donné, correspondant à un élément de même valeur:

```
réplique(s,i) = si i>minimum(s)
                alors si extraire(s,pred(s,i)).p=
                      extraire(s,i).p
                alors pred(s,i)
                sinon
                  réplique(disparaître(s,pred(s,i)),i)
                fsi
            sinon i
            fsi
```

et où la fonction "pred" est définie par:

```
pred(s,i) = si dans?(s,i-1)
            alors i-1 sinon pred(s,i-1)
            fsi
```

Remarque: sur cet exemple le premier champ "p" désigne un numéro de processus entre 1 et n, origine de la donnée "ai" communiquée, et le second champ "x" désigne cette donnée. On réalise ainsi simplement une situation classique de liaison "n producteurs- un consommateur" de type "convergence".

Le système parallèle s'écrit alors:

processus S ::

```
port (vi: mélange;i=1,n);

pour tout i de 1 à n:
processus Pi ::
  entrée v: l..n * entier via vi;
  sortie v: l..n * entier vers vi-1, vi+1;
  entier ai;
  corps
    v.p:=i;
    v.x:=ai; produire(v) ;
    *[ consommer(v) ;
      [ ai<v.x -->
        [ ai>v.x -->
          [ v.x divise ai --> ai:=v.x
          [ non (v.x divise ai) -->
            ai:=ai mod v.x
          ]
        ]
      ]
    ]
    v.x:=ai; produire(v)
  ]
fin Pi;

composition

P1 (// Pi;i=2,n)

fin S;
```

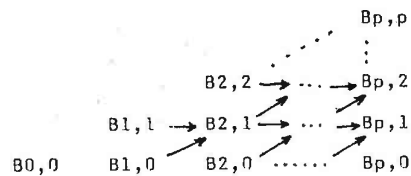
En la comparant à celle présentée dans (AFR_80), le lecteur appréciera certainement l'élégante lisibilité de cette solution, permise par la modularité offerte par la notion de type de communication. Modularité également appréciable pour la conception ou pour la vérification du système.

3-3- EXEMPLE 3: CALCUL DES COEFFICIENTS DU BINÔME-

On souhaite calculer les coefficients du binôme d'ordre $p>0$: pour cela on définit la suite de valeurs suivantes, par récurrence:

$$\forall n \in [0, p] \quad \forall m \in [0, n] \quad B_{n,m} = B_{n-1,m-1} + B_{n-1,m} \text{ si } m < n \text{ et } m > 0 \\ \text{1 si } m = n \text{ ou } m = 0$$

Cette suite peut être ordonnée, selon le triangle de Pascal suivant:



On conçoit alors deux processus parallèles indépendants:

- le premier "P1" calcule les p+1 lignes successives,
- le second "P2" calcule les p+1 diagonales successives.

L'efficacité de ce système peut être améliorée par l'introduction de communications qui transmettent les résultats de chaque processus pour éviter des calculs inutiles à l'autre. L'intérêt de ce type d'algorithmes, dits "itératifs asynchrones simples" dans (Kun_76), au-delà de cet exemple "jouet", est étudié en particulier dans (Bau_78).

Ces communications de type "optionnelles", également présentées dans (Pet_81'), sont ici décrites par le type de communication "égalité_presque_partout" (où la valeur ω est le plus grand élément "max_ent" du type entier) associé aux données communiquées suivantes:

- Pour tout n de 0 à p une donnée communiquée "Dn" transmet les valeurs successives de la nième "diagonale", c'est-à-dire Bn,0; Bn+1,1; Bn+2,2; ...,
- Pour tout m de 0 à p une donnée communiquée "Lm" transmet les valeurs successives de la mième "ligne", c'est-à-dire Bm,m; Bm+1,m; Bm+2,m;

On obtient le système suivant:

processus S ::

port (Di, Li; i=0,p): égalité_presque_partout;

processus P1 ::

entrée (Dn; n=0,p): entier i

sortie (Lm; m=0,p): entier i

entier (Bn, m; n=0, p; m=0, p);

0..p n, m, q;

corps

m:=0;

* if m < p --> n:=m;

* if n < p -->

q:=n-m;

consommer(Dq) i

if Dq < max_ent -->

Bn, m := Dq

if Dq = max_ent -->

-- calculer Bn, m --

fi

Lm := Bn, m;

produire(Lm) i

n:=n+1

fi;

m:=m+1

fin P1;

processus P2 ::

entrée (Lm; m=0,p): entier i

sortie (Dn; n=0,p): entier i

entier (Bn, m; n=0, p; m=0, p);

0..p n, m, q;

corps

n:=0;

* if n < p --> m:=0;

* if m < p-n -->

q:=n+m;

consommer(Lm) i

if Lm < max_ent -->

Bq, m := Lm

if Lm = max_ent -->

-- calculer Bq, m --

fi

Dn := Bq, m;

produire(Dn) i

m:=m+1

fi;

n:=n+1

fin P2;

composition

P1 // P2

fin Si3-4- EXEMPLES 4: ALGORITHMES ITERATIFS EN CALCULS SCIENTIFIQUES-

L'étude des algorithmes itératifs, qui représentent une part importante des algorithmes utilisés en calculs scientifiques, permet de présenter quelques exemples de systèmes parallèles simples utilisant les types de communication présentés au chapitre 2 (cf. (Per_82')).

A partir d'une forme récurrente générale

$$x^{(p)} = \varphi(x^{(p-1)}, x^{(p-2)}, \dots, x^{(p-d)})$$

où " $x^{(p)}$ " est le vecteur produit à la p ème itération, on peut proposer un algorithme parallèle par deux types classiques de décomposition (DuB_82):

- décomposition "vectorielle": le vecteur " x " est décomposé en sous-vecteurs $(x_1, x_2, \dots, x_n)$ sur lesquels est projetée la fonction " φ ",
- décomposition "fonctionnelle": la fonction " φ " est décomposée en fonctions partielles dont les résultats sont "re"-composés.

Dans les deux cas on trouve dans la littérature différents algorithmes mettant en oeuvre différentes situations de communications intéressantes. Nous présentons ici deux problèmes classiques: le calcul d'un point fixe, sur lequel nous appliquons une décomposition vectorielle et la recherche du zéro d'une fonction continue, par l'algorithme de Newton sur lequel nous appliquons une décomposition fonctionnelle, et par l'algorithme de partitionnement, sur lequel nous appliquons une décomposition vectorielle.

3-4-1- Calcul d'un point fixe-

On propose dans (Bau_76) l'algorithme "itératif asynchrone" suivant, de calcul d'un point fixe d'une application " F " de $\mathbb{R}^h$ dans $\mathbb{R}^n$:

$$\forall i \in [1, n] \quad x_i^{(p)} = \begin{cases} F_i(x_1^{(s_1(p))}, x_2^{(s_2(p))}, \dots, x_n^{(s_n(p))}) & \text{si } i \in J_p \\ x_i^{(p-1)} & \text{si } i \notin J_p \end{cases}$$

où, pour tout i de 1 à n , " F_i " est la i ème projection de " F " sur $\mathbb{R}$,
et où, pour tout p , entier strictement positif:

- " J_p " est un sous-ensemble non vide de $\{1, 2, \dots, n\}$ tel que tout entier $i \in [1, n]$ apparait un nombre infini de fois dans l'ensemble des " J_p ", $p \in \mathbb{N}^+$,

- $\forall i \in [1, n]$ $s_i(p)$ est un entier tel que:

$$\begin{aligned} & \cdot s_i(p) \leq p-1 \\ & \cdot \lim_{p \rightarrow +\infty} s_i(p) = +\infty \end{aligned}$$

Cet algorithme peut être exprimé sous la forme de " n " processus parallèles, " P_1 ", " P_2 ", ..., " P_n ", chaque processus " P_i " calculant les valeurs itérées successives de " x_i " (Dij_78) (il existe également des algorithmes "par blocs" dans lesquels chaque processus traite un sous-ensemble de composantes du vecteur " x " (JPS_80), (Sta_82)). On suppose que tous ces processus "progressent", c'est-à-dire que chaque pas d'itération termine dans un temps fini.

Les différentes mises en oeuvre possibles de cet algorithme, qu'on peut formuler par une explicitation des ensembles " J_p " et des suites " $s_i(p)$ ", correspondent en fait à un algorithme identique du point de vue de la description des calculs à opérer, et où seules les relations de communication des données communiquées " x_i " associées à chaque processus diffèrent (JuP_81).

Algorithme de Jacobi:

Les conditions:

$$J_p = \{1, 2, \dots, n\} \quad \forall p \in \mathbb{N}^+$$

$$\text{et si } (p) = p-1 \quad \forall p \in \mathbb{N}^+, \quad \forall i \in [1, n]$$

qui caractérisent l'algorithme de Jacobi, peuvent être exprimées en disant que toute valeur "xi" calculée par le processus "Pi" est utilisée une fois et une seule par les autres processus et dans l'ordre de ces calculs.

Autrement dit la relation entre les ensembles produit et consommé de toute donnée communiquée "xi" est décrite par le type de communication "égalité", qui est donc associé à ces données dans le système suivant, où une donnée communiquée "c" est associée à chaque processus "Pi", pour indiquer la convergence de l'itération. La valeur de "c" est déterminée par un processus "C", qui consomme les différentes valeurs "xi":

processus S ::port c₁ (xi;i=1,n): égalité;

pour tout i de 1 à n:

processus Pi ::entrée (xj;j=1,n;j<>i): réel ;

c: booléen ;

sortie xi: réel ;corps

xi := -- valeur initiale --

produire(xi) ;

(consommer(xj);j=1,n;j<>i); consommer(c) ;

*C non c -->

-- calculer xi --

produire(xi) ;

(consommer(xj);j=1,n;j<>i);

consommer(c)

]

fin Pi;processus C ::entrée (xi;i=1,n): réel ;sortie c: booléen ;corpsc := faux ; produire(c) ;

*C non c --> (consommer(xi);i=1,n);
-- calculer c --
produire(c)

]

fin Cicomposition

C (// Pi;i=1,n)

fin SiAlgorithme de relaxation chaotique (ChM_69):

Lorsque "F" est un opérateur affine dont la matrice est non négative et de rayon spectral inférieur à 1, on montre que la condition

$$\exists s \in \mathbb{N}^+ \text{ tel que } \forall p \in \mathbb{N}^+, \forall i \in [1, n] \quad p - si(p) < s$$

est suffisante pour la convergence de l'algorithme. Dans ces conditions, une relation de communication dite "semi asynchrone" dans (Kun_76), entre les ensembles produit et consommé par chaque processus "Pi" convient. Elle est décrite par le type de communication "à_retard_borné" présenté au chapitre 2. Pour les données communiquées "c" en entrée des processus "Pi", et "xi" en entrée du processus "Q", on utilise alors des ports associés au type de communication "aléatoire".

On obtient un système identique au précédent, où seule la clause "port" est modifiée comme suit:

port c: aléatoire;

(xi;i=1,n): à_retard_borné;

Algorithme de relaxation chaotique avec retards (Mie_75), (Cou_78):

Si "F" est un opérateur linéaire, ou non linéaire vérifiant l'hypothèse de contraction (Tar_81), un algorithme "asynchrone" converge, qui est caractérisé par une relation de communication entre les ensembles produit et consommé associés à chaque donnée communiquée "xi" définie par le type de communication "aléatoire", ou même "très-aléatoire". On

obtient encore un système identique au précédent, où seule la clause "port" est modifiée comme suit:

```
port ci (xi;i=1,n); aléatoire;
-- ou très-aléatoire --
```

Ce type d'algorithme, ainsi qu'une variante intéressante où le type de communication "rafraichie" est substitué au type "aléatoire" des données communiquées "xi" des processus "Pi", a été appliqué à différents problèmes aux dérivées partielles linéaires ou non linéaires. Citons, en particulier la résolution du problème de Poisson, avec des conditions de Dirichlet

$$\begin{cases} -\Delta u + \sigma u = \phi & \text{sur } \Omega \subset \mathbb{R}^n \\ u = 0|_{\Gamma} & \text{sur } \Gamma, \text{ frontière de } \Omega \end{cases}$$

par la méthode alternée de Schwarz ((Mie_82), (MPS_84)).

Pour certains de ces algorithmes, nous avons réalisé des simulations d'implantation des différents types de communication évoqués. On trouvera l'exposé du principe de ces simulations et leurs résultats dans (Jul_81) et (JPS_81). D'autres algorithmes du même type ont également donné lieu à ces mêmes simulations: citons en particulier (Tar_81), (Nak_84) et (Spi_84).

3-4-2-Recherche du zéro d'une fonction-

Soit "F" une fonction réelle continue sur un intervalle $[a,b]$, telle que $F(a)*F(b) < 0$: alors il existe au moins un point "x" de $[a,b]$ tel que $F(x)=0$. Nous considérons deux algorithmes de calcul d'un tel point "x".

Algorithme de Newton:

Si "F" est une fonction dérivable sur $[a,b]$, cet algorithme "asynchrone avec mémoire" (terminologie de (Mie_75') et (Bau_76)), définit la suite d'itérés suivante:

$$x^{(p)} = H(x^{(p-1)}, x^{(s(p))}) \text{ avec } s(p) = p-1 \\ \text{et } H(x,y) = x - F'(y)^{-1} * F(x)$$

Par décomposition fonctionnelle, on peut exprimer cet algorithme par deux processus parallèles:

- "P1" qui calcule la dérivée $F'(y)$,
- "P2" qui calcule la valeur de la fonction $F(x)$ et l'itéré suivant.

Différentes mises en oeuvre en sont possibles, qui diffèrent par les types de communication associés aux ports utilisés par les données communiquées "f'" et "x" des processus "P1" et "P2":

- Une version "synchrone", telle que $s(p)=p-1$, $\forall p \in \mathbb{N}^*$ est définie par le système suivant:

```
processus S ::
port xf f': égalité; ci aléatoire;

processus P1 ::
entrée xi réel i
ci booléen i
sortie f': réel i
corps
consommer(c) i
*[ c --> consommer(x) i
-- calcul de la dérivée --
produire(f') i
consommer(c)
]
fin P1;

processus P2 ::
entrée f': réel i
sortie xi réel i
ci booléen i
corps
c := vrai i
produire(c) i
x := -- valeur initiale --
produire(x) i
*[ c --> consommer(f') i
-- calcul de l'itéré suivant --
-- calcul de la condition d'arrêt --
produire(x)
]
c := faux;
produire(c)
fin P2;
```

composition

P1 // P2

fin S;

- On obtient une version "asynchrone", présentée dans (Kun_76), en substituant le type de communication "aléatoire" au type "égalité" pour les ports "x" et "f".

Algorithme de partitionnement:

Cet algorithme itératif réduit l'intervalle de recherche du zéro à chaque itération, en calculant la valeur de la fonction en un point "x" convenablement choisi de l'intervalle:

si $F(a)*F(x) \leq 0$ alors le nouvel intervalle est $[a, x]$
tandis que si $F(x)*F(b) \leq 0$ le nouvel intervalle est $[x, b]$.

On propose dans (ErS_83) une version "vectorielle" de cet algorithme en calculant la fonction "F" sur "n" points "x1", "x2", ..., "xn" de l'intervalle.

On considère donc "n" processus "P1", "P2", ..., "Pn", chaque processus "Pi" calculant la valeur de la fonction au ième point "xi" de l'intervalle courant. Cet intervalle est déterminé par un processus "Q", qui dans la version "asynchrone" de cet algorithme, "re"-calcule cet intervalle à chaque fois qu'un nouveau couple (xi, F(xi)) lui parvient: il détermine alors les nouveaux points "xi", qui sont transmis aux processus "Pi" correspondants. Ces données communiquées utilisent des ports associés au type de communication "rafraichie", qui assure que la valeur de la fonction est calculée, autant que faire se peut, en un point de l'intervalle courant, et pas deux fois au même point.

Le système a la forme suivante:

processus S ::

port (xi;i=1,n): rafraichie;
c: aléatoire;
p: égalité;
-- ou tout autre type assurant que toute donnée
produite est consommée, mais pas nécessairement
dans le même ordre --

pour tout i de 1 à n:

processus Pi ::

entrée x: réel via xi;
c: booléen i;
sortie u: réel * réel vers pi
corps
 consommer(c) i;
 *[c --> consommer(x);
 u:= (x, F(x));
 produire(u) i;
 consommer(c)

]

fin Pi;processus Q ::

entrée (u,v): réel * réel via pi;
sortie (xi;i=1,n): réel i;
c: booléen i

réel a, b, f1, f2, eps;

intervalle inter;

corps

 inter:= (a,b) i;
 f1:= F(a) i; f2:= F(b) i;
 c:= vrai i;
 produire(c) i;
 -- déterminer les points xi --
 (produire(xi); i=1,n) i;
 *[long(inter) < eps -->
 consommer((u,v)) i;
 [f1*v < 0 --> b:= ui; f2:= v
 f2*v < 0 --> a:= ui; f1:= v
] i;
 [réduit(inter, a, b) -->
 inter:= (a,b) i;
 -- déterminer les xi --
 (produire(xi); i=1,n)

]

]

c:= faux i;
produire(c)

fin Q;

composition

```

Q (// Pi;i=1,n);
fin Si

```

On trouve dans (Per_85) la dérivation d'un programme CSP à partir de cet énoncé. On procède de la façon suivante (analogue à celle présentée au chapitre 7 de cette thèse, paragraphe 7-3-3-):

- représentation de la manipulation des objets de type "TC" par des processus CSP,
- transformations par des implantations particulières des types de communication (cf. chapitre 5),
- fusion de processus.

3-5- EXEMPLE 5: PLUS COURT CHEMIN DANS UN GRAPHE-

Le problème est celui du calcul de la plus petite distance "di" d'un noeud "vl" à un noeud "vi", $i=2,n$ sur un graphe orienté dont les arcs sont valués, avec des poids entiers tels qu'il n'y ai pas de cycle négatif: on note w_{ik} le poids de l'arc reliant un noeud "vi" à un successeur "vk".

On a l'énoncé suivant:

```

. dl = 0
. di = min{dk+wki, k tel que i ∈ succ(k)}  ∀ i>1

```

On peut définir chaque valeur "di" comme le dernier terme d'une suite "di" définie comme suit:

```

. dlj = 0 pour tout j
. pour i>1:
  . si i ∈ succ(1) di0 = wli
  . dij = min{dkm+wki, k tel que i ∈ succ(k) et m<j}
. di = lim dij  ∀ i>0
      j

```

Ordonnancement des calculs:

Pour déterminer l'intermédiaire d_{ij} on doit avoir calculé d_{km} , pour tout noeud "vk" prédécesseur de "vi" et pour tout $m < j$. L'algorithme suivant satisfait cette condition:

```

dl:=0; pour i de 2 à n: di:=max_ent;
*[pour i de 1 à n:
  pour tout k ∈ succ(i):
    [dk > di+wik --> dk:=di+wik]
]

```

Cet algorithme peut être vectorisé en "n" processus parallèles "Pi", chacun étant associé à un noeud "vi": pour tout $i > 1$, "Pi" calcule la plus petite distance "d" de "vl" à "vi". Il reçoit de chacun de ses prédécesseurs une suite décroissante de valeurs de "d". Si cette valeur est inférieure à sa valeur courante, elle devient la nouvelle distance minimum courante, et est transmise, augmentée du poids de l'arc correspondant, à tous les processus successeurs de "Pi", c'est-à-dire aux processus associés aux noeuds successeurs de "vi" dans le graphe.

Une mise en oeuvre "asynchrone" de cet algorithme, suggérée dans (ChM_82), est réalisée en associant le type de communication "mélange", présenté ci-dessus en 3-2-2-, aux données communiquées "mi" qui sont des couples formés du numéro du processus producteur et d'une distance (on note "p" et "x" les projections correspondantes).

On obtient le système suivant:

processus S ::

```

port (mi;i=1,n): mélange;

```

```

-- wik est le poids de l'arc reliant vi et un
successeur vk --
-- succ(i) est l'ensemble des numéros des
successeurs de vi --

```

processus Pl ::

```

sortie (mk;k ∈ succ(1)): 1..n * entier i
entier (wik;k ∈ succ(1));

```

```

corps
  (mk:=1,wlk);
  produire(mk);k:=succ(1))
fin P1;

pour tout i>1:
processus Pi ::
  entrée m: 1..n * entier via mi;
  sortie (mk;k:=succ(i)): 1..n * entier;
  entier d_i (wik;k:=succ(i))_i
  corps
    d:=max_ent;
    *E consommer(m) i
    E m.x<d --> d:=m.x;
                  (mk:=(i,wik+d));
                  produire(mk);k:=succ(i))
  ]
fin Pi;

composition

P1 (// Pi;i=2,n);

fin S;

```

On peut simplifier cet énoncé en constatant que l'origine des valeurs n'est pas utilisée par les processus consommateurs: on peut donc substituer au type de communication "mélange" le type de communication "valeur-minimum", qui extrait la plus petite valeur de l'ensemble consommable et en supprime toutes les valeurs. Il est défini comme suit:

```

pré-cons (<SP,A,SC>) = non vide?(A)
val-cons (<SP,A,SC>) = extraire(A,j(A))
post-cons (<SP,A,SC>) = tuer(A)

```

```

où j(s) = si cardinal(s)=1 alors minimum(s)
          sinon si dernier(s)>extraire(s,pred(s,maximum(s)))
              alors j(disparaître(s,maximum(s)))
              sinon j(disparaître(s,pred(s,maximum(s))))
          fsi
fsi

```

CHAPITRE 4- SEMANTIQUE D'UN SYSTEME PARALLELE

Nous proposons dans ce chapitre une définition sémantique du langage succinct présenté au chapitre 2.

Dans la littérature, on retrouve à ce propos les différentes approches de définition sémantique, classiques pour les langages d'expression de systèmes séquentiels, étendues aux problèmes du parallélisme.

La sémantique dénotationnelle, ou fonctionnelle, définit un système comme l'ensemble des fonctions qu'il calcule: cette approche est rendue très complexe dans le domaine du parallélisme (voir (Sch_79), (BaZ_82)).

La sémantique opérationnelle définit un système comme une machine à états (voir, en particulier (Plo_82)), généralement sous la forme d'un système de transitions (cf. chapitre 1).

Ce même type d'approche peut être exprimé, sous forme algébrique, en définissant le système de transitions comme un type abstrait, dont les opérations sont les constructions du

langage (voir (Pai_82), et pour des systèmes parallèles (BrW_80) et (FiO_83)).

Enfin les sémantiques axiomatiques procèdent par dérivation d'assertions logiques, soit dans la logique de "Hoare", étendue aux opérations de communication de processus (EIF_82), soit dans des logiques "dynamiques", modale ou temporelle ((Abr_79), (Pnu_81)), sur lesquelles nous reviendrons dans des chapitres ultérieurs de cette thèse.

La définition présentée ici procède en deux temps:

- nous proposons une sémantique opérationnelle non déterministe pour la définition des systèmes parallèles. Cette approche simple est en particulier adaptée au type de preuves que nous développons au chapitre 6.

- pour ce qui concerne la partie "communication" des systèmes, nous précisons cette sémantique sous une forme déterministe de type fonctionnelle. Cette fonction sémantique est définie à partir du concept de "suite temporelle": à une suite temporelle d'opérations de production et une suite temporelle de requêtes de consommation, elle associe une suite temporelle d'opérations de consommation. Nous observerons au chapitre 7 que ce concept est bien adapté à l'aspect spécification de problèmes et méthode de construction de programmes.

4-1- SEMANTIQUE OPERATIONNELLE D'UN SYSTEME PARALLELE -

Nous proposons une sémantique opérationnelle du langage présenté au chapitre 2 (cf. (Per_83') et (Per_83'')). Pour cela on ignore, dans un premier temps, les conséquences des opérations de communication: un processus est considéré comme une entité sémantique autonome. Puis nous composons les différents processus d'un système, dont nous déduisons la sémantique.

4-1-1-Sémantique d'un processus séquentiel-

On suppose que tout énoncé, dérivant du symbole non terminal ENONCES, peut être exprimé sous une forme normale

elie2, où el est l'opération "vide", ou une affectation, ou une production, ou une consommation, ou un choix.

(On suppose donc que l'itération vérifie:

$$*[b \rightarrow e] = [b \rightarrow e] ; *[b \rightarrow e]$$

Dans ces conditions, il suffit d'exprimer la fonction de transition à partir de ces énoncés élémentaires. On obtient la définition suivante:

A tout processus est associé un automate présenté comme un triplet $\langle Q, q_0, T \rangle$, où

- Q est un ensemble infini d'états: un état est un couple (m, e) , où

- . m est "l'état mémoire" du processus, c'est-à-dire une application de l'ensemble des données typées (locales ou communiquées) du processus dans l'ensemble de leurs valeurs. On note "VAL" l'application qui associe une valeur à l'identificateur d'une donnée,
- . e est un énoncé du langage ou une "erreur".

- $q_0 \in Q$ est l'état initial du processus:

$$q_0 = (\mu, \text{"vide"}), \text{ où } \mu \text{ est un état mémoire indéfini.}$$

- T est un ensemble de transitions, définissant la relation binaire "→" suivante sur Q :

- . $(m, \text{"vide"}) \rightarrow (m, \text{"vide"})$
- . $(m, \text{"erreur"}) \rightarrow (m, \text{"erreur"})$
- . $(m, id := exp) \rightarrow (m', \text{"vide"})$
où $m' = m(\text{VAL.exp substitué à VAL.id})$
- . $(m, \text{produire}(x)) \rightarrow (m, \text{"vide"})$
-- l'opération "produire" ne modifie pas l'état mémoire du processus --
- . $(m, \text{consommer}(x C)) \rightarrow (\mu, \text{"vide"})$
-- l'opération "consommer" laisse le processus dans un état mémoire indéfini --
- . $(m, elie2) \rightarrow (m', e2)$
où m' est l'état résultant de el , c'est-à-dire:
 $(m, el) \rightarrow (m', \text{"vide"})$
- . $(m, [b1 \rightarrow] el \square \dots \square bn \rightarrow en [1]) \rightarrow$
si existe(i) telque bi
alors (m, ei) sinon $(m, \text{"erreur"})$
fsi

Remarque-

Le fait de considérer dans cette fonction de transition, au demeurant très classique, uniquement les états résultant d'une opération de l'énoncé du processus peut s'interpréter, pour ce qui nous concerne, de deux manières équivalentes: ou bien ces opérations sont considérées comme "atomiques", c'est-à-dire instantanées par rapport à l'échelle de temps retenue, ou bien on parle en fait des "fins d'opération". Cette remarque est fondamentale pour la suite, où on compose des processus pour définir la sémantique d'un système parallèle. Ceci est fait en deux temps: on définit d'abord le "produit" de processus, puis leur composition par "communication".

4-1-2-Produit de processus-

Le "produit" de processus définit un système non-déterministe constitué de l'entrelacement ou de la simultanéité (compte-tenu de la remarque ci-dessus) des opérations de chacun des processus, et défini par l'automate suivant:

Considérons un ensemble de processus $P_1, P_2, \dots, P_p$.

- Soit $S_i = \langle Q_i, q_{i0}, T_i \rangle$ l'automate associé au processus P_i , pour i de 1 à p .

Rappelons qu'un "canal de communication" est un triplet

$$c = (x, a, y)$$

où

- . a est un port de communication,
- . x est une donnée communiquée figurant dans une liste de sortie sous la forme:

sortie x_i ... vers a_i

. y est une donnée communiquée figurant dans une liste d'entrée sous la forme:

entrée y_i ... via a_i

- Soit $C = \{c_1, c_2, \dots, c_r\}$ l'ensemble des canaux de communication définis par les processus $P_1, P_2, \dots, P_p$.

- L'automate $\prod S_i = \langle \prod Q_i, (q_{i0}, i \in [1, p]), \prod T_i \rangle$

où - $\prod Q_i$ est le produit cartésien des ensembles $Q_1, Q_2, \dots, Q_p$,

- toute transition de $\prod T_i$ est de la forme:

$$((m_1, e_1), \dots, (m_i, e_i), \dots, (m_p, e_p)) \rightarrow ((m'_1, e'_1), \dots, (m'_i, e'_i), \dots, (m'_p, e'_p))$$

avec $(m_i, e_i) \rightarrow (m'_i, e'_i) \in T_i, i \in [1, p]$

ou $m_i = m'_i$ et $e_i = e'_i$

ou si $e_i = \text{produire}(x)$

et $e_j = \text{consommer}(y)$

et $(a, x, y) \in C$

alors $e_i = e'_i$ ou $e_j = e'_j$

et $\exists i \in [1, p] \quad m_i \neq m'_i \text{ ou } e_i \neq e'_i$

-- c'est-à-dire que la simultanéité de deux opérations, l'une de production et l'autre de consommation concernant le même canal de communication n'est pas définie

--

exprime la sémantique du processus appelé le produit des processus $P_1, P_2, \dots, P_p$.

Remarquons que l'état mémoire initial ou résultant d'une opération de consommation n'est pas défini par l'opération "produit". Ainsi que le montre l'exemple suivant:

Exemple-

Le système producteur-consommateur présenté au paragraphe 2-2-1- est défini par l'ensemble des transitions suivantes:

. $((m, \text{"vide"}), (m', e')) \rightarrow ((m, \text{"vide"}), (m', e'))$
 . $((m, \text{"erreur"}), (m', e')) \rightarrow ((m, \text{"erreur"}), (m', e'))$
 . $((m, \text{id} := \text{exp}), (m', e')) \rightarrow ((m, \text{"vide"}), (m', e'))$
 où $m = m(\text{VAL.exp substitué à VAL.id})$
 . $((m, \text{produire}(x)), (m', e')) \rightarrow ((m, \text{"vide"}), (m', e'))$
 . $((m, e_1, e_2), (m', e')) \rightarrow ((m, e_2), (m', e'))$
 où $(m, e_1) \rightarrow (m, \text{"vide"})$
 . $((m, \text{cbl} \rightarrow e_1 \dots \text{cbr} \rightarrow e_2), (m', e')) \rightarrow$

```

      si existe(i) telque bi
      alors ((m,ei),(m',e'))
      sinon ((m,"erreur"),(m',e'))
      fsi
. ((m,e),(m',"vide")) ---> ((m,e),(m',"vide"))
. ((m,e),(m',"erreur")) ---> ((m,e),(m',"erreur"))
. ((m,e),(m',id:=exp')) ---> ((m,e),(m',"vide"))
  où m=m'(VAL.exp' substitué à VAL.id')
. ((m,e),(m', consommer(y))) ---> ((m,e),(μ,"vide"))
. ((m,e),(m',e'lie'2)) ---> ((m,e),(m',e'2))
  où (m',e'1) ---> (m',"vide")
. ((m,e),(m',[b'l-->e'lQ...Qb'n-->e'nJ])) --->
  si existe(j) telque b'j
  alors ((m,e),(m',e'j))
  sinon ((m,e),(m',"erreur"))
  fsi
. ((m,id:=exp),(m',id:=exp')) --->
  ((m,"vide"),(m',"vide"))
  où m=m'(VAL.exp' substitué à VAL.id)
  m'=m'(VAL.exp' substitué à VAL.id')
. ((m,id:=exp),(m', consommer(y))) --->
  ((m,"vide"),(μ,"vide"))
  où m=m'(VAL.exp' substitué à VAL.id)
. ((m,id:=exp),(m',e'lie'2)) ---> ((m,"vide"),(m',e'2))
  où m=m'(VAL.exp' substitué à VAL.id)
  (m',e'1) ---> (m',"vide")
. ((m,id:=exp),(m',[b'l-->e'lQ...Qb'n-->e'nJ])) --->
  si existe(j) telque b'j
  alors ((m,"vide"),(m',e'j))
  sinon ((m,"vide"),(m',"erreur"))
  fsi
  où m=m'(VAL.exp' substitué à VAL.id)
. ((m, produire(x)),(m',id:=exp')) --->
  ((m,"vide"),(m',"vide"))
  où m'=m'(VAL.exp' substitué à VAL.id')
. ((m, produire(x)),(m',e'lie'2)) --->
  si e'1<> consommer(y)
  alors ((m,"vide"),(m',e'2))
  fsi
  où (m',e'1) ---> (m',"vide")
. ((m, produire(x)),
  (m',[b'l-->e'lQ...Qb'n-->e'nJ])) --->
  si existe(j) telque b'j
  alors ((m,"vide"),(m',e'j))
  sinon ((m,"vide"),(m',"erreur"))
  fsi
. ((m,e'lie'2),(m',id:=exp')) ---> ((m,e2),(m',"vide"))
  où (m,e1) ---> (m,"vide")
  m'=m'(VAL.exp' substitué à VAL.id')
. ((m,e'lie'2),(m', consommer(y))) --->

```

```

      si e1<> produire(x)
      alors ((m,e2),(μ,"vide"))
      fsi
  où (m,e1) ---> (m,"vide")
. ((m,e'lie'2),(m',e'lie'2)) --->
  si (e1,e'1)<>
  (produire(x), consommer(y))
  alors ((m,e2),(m',e'2))
  fsi
  où (m,e1) ---> (m,"vide")
  (m',e'1) ---> (m',"vide")
. ((m,e'lie'2),(m',[b'l-->e'lQ...Qb'n-->e'nJ])) --->
  si existe(j) telque b'j
  alors ((m,e2),(m',e'j))
  sinon ((m,e2),(m',"erreur"))
  fsi
  où (m,e1) ---> (m,"vide")
. ((m,[b'l-->e'lQ...Qb'n-->e'nJ]),(m',id:=exp')) --->
  si existe(i) telque bi
  alors ((m,e1),(m',"vide"))
  sinon ((m,"erreur"),(m,"vide"))
  fsi
  où m'=m'(VAL.exp' substitué à VAL.id')
. ((m,[b'l-->e'lQ...Qb'n-->e'nJ]),(m', consommer(y))) --->
  si existe(i) telque bi
  alors ((m,e1),(μ,"vide"))
  sinon ((m,"erreur"),(μ,"vide"))
  fsi
. ((m,[b'l-->e'lQ...Qb'n-->e'nJ]),(m',e'lie'2)) --->
  si existe(i) telque bi
  alors ((m,e1),(m',e'2))
  sinon ((m,"erreur"),(m',e'2))
  fsi
  où (m',e'1) ---> (m',"vide")
. ((m,[b'l-->e'lQ...Qb'n-->e'nJ]),
  (m',[b'l-->e'lQ...Qb'n-->e'nJ])) --->
  si existe(i) telque bi
  et existe(j) telque b'j
  alors ((m,e1),(m',e'j))
  sinon si existe(i) telque bi
  alors ((m,e1),(m',"erreur"))
  sinon si existe(j) telque b'j
  alors
  ((m,"erreur"),(m',e'j))
  sinon
  ((m,"erreur"),(m',"erreur"))
  fsi
  fsi
  fsi

```

4-1-3-Opérateur de communication-

Cet opérateur agissant sur un système non-déterministe a pour effet de définir son état mémoire initial, ou résultant d'une opération de consommation. Il est paramétré par le type de communication associé aux ports caractérisant les canaux de communication. On suppose dans la suite que l'ensemble "C" des canaux de communication est non vide, c'est-à-dire que le système est "au-moins" obtenu par produit d'un processus producteur et d'un processus consommateur sur le même canal.

Cet opérateur définit la sémantique d'un système exprimé syntaxiquement par:

```

processus S ::=
  composition
    P1 // P2 // ... // Pp
fin S;

```

Soit $\langle Q, q_0, T \rangle$ l'automate associé au produit des processus $P_1, P_2, \dots, P_p$, et les objets "tcl" de type "TC.c1", "tc2" de type "TC.c2", ..., "tcr" de type "TC.cr", caractérisant les canaux de communication "c1", "c2", ..., "cr" de "C",

L'opérateur de communication appliqué à ce système définit le "système parallèle", décrit par l'automate suivant:

$\mathcal{S} = \langle Q, \xi_0, \mathcal{E} \rangle$

- où - Q est le produit cartésien $Q * tcl * tc2 * \dots * tcr$
- ξ_0 est l'état (q_0 , init(), init(), ..., init())
- $\mathcal{E}$ est l'ensemble T , modifié comme suit:
- toute transition où n'apparaît pas une opération "produire" ou "consommer" ne modifie pas les objets de type "TC".
 - les transitions où apparaissent ces opérations sont modifiées dans le sens suivant:
- pour tout "cij" dans "C" (on note $cij = (xi, aij, yj)$ les

noms respectifs du port et des données communiquées définissant un canal),

```

. (((m1,e1),...,(mk, produire(xi)),...,(mp,ep)),
  tcl,...,tcij,...,tcr) --->
  (((m'1,e'1),...,(mk,"vide"),...,(m'p,e'p)),
  tc'1,..., prod(tcij,VAL.xi),...,tc'r)

```

avec $((m1,e1),...,(mp,ep)) \rightarrow ((m'1,e'1),...,(m'p,e'p)) \in T$
et $tc'n$ déduit de tcn par une éventuelle opération de communication, pour tout n de 1 à r , $n \neq k$

-- une même donnée communiquée en entrée "xi" peut être associée à plusieurs canaux de communication "cij" (cf. paragraphe 2-2-2-). Pour simplifier la présentation on indique un seul nom de canal "cij" dans cette transition

```

. pré-cons(tcij) ==>
  (((m1,e1),...,(mk, consommer(yj)),...,(mp,ep)),
  tcl,...,tcij,...,tcr) --->
  (((m'1,e'1),...,(m'k,"vide"),...,(m'p,e'p)),
  tc'1,..., post-cons(tcij),...,tc'r)

```

où $m'k = mk(\text{val-cons}(tcij) \text{ substitué à } VAL.yj)$

avec $((m1,e1),...,(mp,ep)) \rightarrow ((m'1,e'1),...,(m'p,e'p)) \in T$

Sur l'exemple précédent, dans le cas où le port "a" est déclaré de type "aléatoire", on obtient, en explicitant "tc" sous la forme du triplet d'ensembles $\langle SP, A, SC \rangle$:

```

. (((m, produire(x)), (m', e')), <SP, A, SC>) --->
  (((m, "vide"), (m', e')),
  <ajouter(SP, VAL.x), apparaitre(A, maximum(SP)+1, VAL.x), SC>)

. non vide?(A) ==>
  (((m, e), (m', consommer(y))), <SP, A, SC>) --->
  (((m, e), (m', "vide")),
  <SP, finir(A), ajouter(SC, dernier(A))>)
  où  $m' = m'(\text{dernier}(A) \text{ substitué à } VAL.y)$ 

. (((m, produire(x)), (m', id:=exp')), <SP, A, SC>) --->
  (((m, "vide"), (m', "vide")),
  <ajouter(SP, VAL.x), apparaitre(A, maximum(SP)+1, VAL.x), SC>)
  où  $m' = m'(VAL.exp' \text{ substitué à } VAL.id')$ 

```

```

. ((m, produire(x)), (m', e'1; e'2)), <SP, A, SC> --->
  si e'1 <> consommer(y)
  alors ((m, "vide"), (m', e'2)),
<ajouter(SP, VAL.x), apparaitre(A, maximum(SP)+1, VAL.x), SC>
  fsi
  où (m', e'1) ---> (m, "vide")

. ((m, produire(x)), (m', [b'1-->e'1 l0...l0 b'n-->e'n])),
  <SP, A, SC> --->
  si existe(j) telque b'j
  alors ((m, "vide"), (m', e'j)),
<ajouter(SP, VAL.x), apparaitre(A, maximum(SP)+1, VAL.x), SC>
  sinon ((m, "vide"), (m', "erreur")),
<ajouter(SP, VAL.x), apparaitre(A, maximum(SP)+1, VAL.x), SC>
  fsi

. non vide?(A) ==>
  (((m, id:=exp), (m', consommer(y))),
   <SP, A, SC> --->
   (((m, "vide"), (m', "vide")),
    <SP, finir(A), ajouter(SC, dernier(A))>))
  où m=m(VAL.exp substitué à VAL.id)
      m'=m'(dernier(A) substitué à VAL.y)

. non vide?(A) ==>
  (((m, e1; e2), (m', consommer(y))),
   <SP, A, SC> --->
   si e1 <> produire(x)
   alors ((m, e2), (m', "vide")),
   <SP, finir(A), ajouter(SC, dernier(A))>))
  fsi
  où (m, e1) ---> (m, "vide")
      m'=m'(dernier(A) substitué à VAL.y)

. non vide?(A) ==>
  (((m, b1-->e1...bn-->en),
   (m', consommer(y))), <SP, A, SC> --->
   si existe(i) telque bi
   alors ((m, ei), (m', "vide")),
   <SP, finir(A), ajouter(SC, dernier(A))>))
  sinon ((m, "erreur"), (m', "vide")),
  <SP, finir(A), ajouter(SC, dernier(A))>))
  fsi
  où m'=m'(dernier(A) substitué à VAL.y)

```

4-2- INTERPRETATION D'UN TYPE DE COMMUNICATION-

Nous avons observé au paragraphe 1-4-2- que les modèles formels classiques, de type ensembliste, ne conviennent pas pour interpréter l'expression d'une relation de communication, par exemple comme une relation statique entre des ensembles.

Pour réaliser une telle interprétation nous proposons

- d'enrichir ces modèles en y incluant la notion de temps,
- de faire apparaitre la notion de requête d'opération.

Nous définissons alors la sémantique d'un type de communication comme une fonction qui, à une suite de données produites (notée "SP" dans la suite) et une suite de requêtes d'opération de consommation (notée "SC" dans la suite), associe une suite de données consommées. Ces suites sont des "suites temporelles".

4-2-1-Notion de suite temporelle- ((FiJ_80), (FJJ_84))

Définition-

On appelle "CHRONO" un ensemble infini totalement ordonné, dont les éléments sont des "instants". On note "<" la relation d'ordre sur cet ensemble.

On appelle suite temporelle d'objets de type "elt" une application de $\mathbb{N}$ dans le produit cartésien $\text{elt} * \text{CHRONO}$, application qui conserve l'ordre des instants, c'est-à-dire:

si on note $S_i = (E_i, t_i)$, $i \in \mathbb{N}$, le terme général d'une telle suite "S",

$$\forall i, j \in \mathbb{N} \quad i < j \implies t_i < t_j$$

Notation:

Nous noterons désormais $ti_instant(Si)$.

Remarque:

Dans les exemples traités dans cette thèse, et pour l'interprétation qui suit, il suffit de considérer un ordre total sur "CHRONO". Pour traiter de problèmes dits "temps réel" on devrait définir, en outre, une distance sur cet ensemble. On peut, à ce propos, lier ce cadre formel à celui développé dans le langage LUSTRE (BCH_85) pour l'expression des problèmes temps réel. Dans ce langage "à la LUCID" (AsW_76) on exprime des relations entre objets "asynchrones": chaque objet -analogue à une suite temporelle- est constitué . d'une suite de valeurs, . d'un objet de type booléen, appelé "horloge": cet objet définit donc une suite de valeurs booléennes. Des opérateurs "temporels" (when, current, ...) permettent de définir des expressions combinant ces objets à la condition que leurs horloges aient la même horloge, c'est-à-dire qu'on dispose d'une référence "horloge universelle". Celle-ci est exprimée par l'ensemble "CHRONO" utilisé ici pour définir toute suite temporelle.

Nous nous proposons donc d'interpréter un type de communication dans le domaine "ST" des suites temporelles d'objets de type "ensemble-indexé". Nous définissons ce domaine:

Définition:

On note "EC" le produit cartésien ensemble-indexé * CHRONO.

On appelle suite temporelle associée à un objet "S" de type "ensemble-indexé", une suite temporelle "S", à valeurs dans "EC", telle que:

si on note "ensemble(Si)" le premier élément, de type "ensemble-indexé", du terme général "Si" de "S",

. S0 = (créer(), 1)

où 1 est un élément quelconque de "CHRONO"

. $\forall i > 0 \quad Si = (op(ensemble(S_{i-1})), ti)$

où $ti-1 < ti$
et "op" dénote une opération quelconque sur les objets de type "ensemble-indexé", à résultat de type "ensemble-indexé" (apparaître, disparaître, ajouter, effacer, finir, gommer, ...).

Exemple:

A l'ensemble produit "SP" d'un objet de type "TC" on associe la suite temporelle "S1" dont les termes sont définis comme suit:

. S10 = ({}, 1)
. S11 = (ajouter({}, el), t1)

qu'on notera ({p1}, t1), "p1" représentant en fait un élément d'un objet de type "ensemble-indexé", c'est-à-dire le couple (1, el) dans entier_positif*elt,

. S12 = (ajouter({p1}, e2), t2)
= ({p1, p2}, t2)

avec les mêmes conventions de notation,

. S13 = (ajouter({p1, p2}, e3), t3)
= ({p1, p2, p3}, t3)

. etc.

4-2-2-Domaine d'interprétation d'un type de communication-

Comme intermédiaire à l'interprétation d'un type de communication nous introduisons le domaine $EC^3 = EC * EC * EC$, constitué de l'ensemble des triplets de termes de suites temporelles, éléments de "ST".

Définition 1-

Soient S_1 , S_2 et S_3 des suites temporelles éléments de "ST",

A ces trois suites temporelles on associe une suite quelconque, notée $[S_1, S_2, S_3]$ et appelée "suite-triple", extraite de la suite à trois indices dont le terme général, de type EC3, est le triplet (S_{1i}, S_{2j}, S_{3k}) , noté sous la forme $[S_{1i}, S_{2j}, S_{3k}]$.

Tout terme de cette suite-triple est construit par une des opérations suivantes:

Définition 2-

L'opération INIT: $---> EC3$, définit le terme initial de $[S_1, S_2, S_3]$ de la manière suivante:

INIT () = $[S_{10}, S_{20}, S_{30}]$

avec $S_{10} = S_{20} = S_{30} = (\{\}, \perp)$

Définition 3-

L'opération PROD: $EC3 * ST ---> EC3$, appliquée au terme général $[S_{1i}, S_{2j}, S_{3k}]$ d'une suite-triple et à un objet "SP" de type "ST" définit le terme $[S_{1i+1}, S_{2j}, S_{3k}]$ suivant:

PROD($[S_{1i}, S_{2j}, S_{3k}], SP$) = $[S_{1i+1}, S_{2j}, S_{3k}]$

avec .ensemble(S_{1i+1})=ajouter(ensemble(S_{1i}), ei+1)
.instant(S_{1i+1})=instant(S_{1i})

Définition 4-

L'opération CONS: $EC3 * ST ---> EC3$, appliquée au terme général $[S_{1i}, S_{2j}, S_{3k}]$ d'une suite-triple et à un objet "SC" de type "ST" vérifiant l'hypothèse

(H) instant(S_{3k+1}) > instant(S_{3k})

définit le terme $[S_{1i}, S_{2j+1}, S_{3k+1}]$ suivant:

CONS($[S_{1i}, S_{2j}, S_{3k}], SC$) = $[S_{1i}, S_{2j+1}, S_{3k+1}]$

avec les définitions suivantes:

- Définition de S_{3k+1} :

. ensemble(S_{3k+1}) = ajouter(ensemble(S_{3k}),
valeur(S_{1i}, S_{2j+1}, S_{3k}))
. instant(S_{3k+1}) = instant(S_{2j+1})

où "valeur" est une fonction quelconque associant un objet de type "elt" à un objet de type "EC3".

- Définition de S_{2j+1} :

. ensemble(S_{2j+1}) = $\begin{matrix} \text{si pre}(S_{1i}, B_j, S_{3k}) \\ \text{alors ensemble}(B_j) \\ \text{sinon ensemble}(BC_{imin}) \end{matrix}$
fsi
. instant(S_{2j+1}) = $\begin{matrix} \text{si pre}(S_{1i}, B_j, S_{3k}) \\ \text{alors instant}(B_j) \\ \text{sinon instant}(BC_{imin}) \end{matrix}$
fsi

où "pre" est une fonction quelconque associant un booléen à un objet de type "EC3",

"B" et "BC" sont des éléments de "ST", fonctions de S_1 , S_2 , S_3 et S_C , définies ci-dessous,

"imin" est le plus petit entier positif $m(i, j, k) \geq i$ tel que
pre($S_{1m}(i, j, k), BC_m(i, j, k), S_{3k}$).

Définition 5-

La suite temporelle intermédiaire BC est définie par:

. ensemble($BC_m(i, j, k)$) = ensemble(B_j)
 $\cup$ ensemble($S_{1m}(i, j, k)$) - ensemble($S_{1lmax}(i, j, k)$)

où " $l_{\max}(i,j,k)$ " est le plus grand entier positif
 " $l(i,j,k)$ " tel que

$\text{instant}(S_{1l}(i,j,k)) \leq \text{instant}(S_{3k})$

. $\text{instant}(B_{Cm}(i,j,k)) = \text{instant}(S_{1m}(i,j,k))$

-- intuitivement, ce terme intermédiaire $B_{Cm}(i,j,k)$ exprime l'ensemble des valeurs "ajoutées" à l'objet de type "ensemble-indexé" défini dans S_{1i} (par des opérations "PROD") depuis l'instant défini dans S_{3k} jusqu'à l'instant défini dans $S_{1m}(i,j,k)$;

"ensemble(B_{Cimin})", fonction de $\llbracket S_{1i}, S_{2j}, S_{3k} \rrbracket$ et de la suite SP , exprime donc l'ensemble des valeurs "ajoutées" à "ensemble(S_{1i})" depuis "instant(S_{3k})" jusqu'à "instant(S_{1imin})", qui est le plus petit instant suivant "instant(S_{3k})", auquel la fonction "pre" rend la valeur vrai

--

Définition 6-

La suite temporelle intermédiaire B est définie par:

. $\text{ensemble}(B_j) = \text{op}(\text{ensemble}(S_{2j}))$
 $\quad \cup \text{ensemble}(S_{1\max}(i,j,k))$
 $\quad - \text{ensemble}(S_{1l\max}(i,j,k))$

où " $l_{\max}(i,j,k)$ " est le plus grand entier positif
 " $m(i,j,k)$ " tel que

$\text{instant}(S_{3k}) < \text{instant}(S_{1m}(i,j,k))$
 $\quad = < \text{instant}(S_{3k+1})$

"op" dénote une opération quelconque sur les objets de type "ensemble-indexé", à résultat de type "ensemble-indexé".

. $\text{instant}(B_j) = \text{instant}(S_{3k+1})$

-- intuitivement, ce terme intermédiaire B_j , fonction de $\llbracket S_{1i}, S_{2j}, S_{3k} \rrbracket$, exprime l'ensemble défini dans " S_{2j} " à l'instant défini dans " S_{3k+1} ": il est déterminé à partir du résultat de la dernière opération "op" sur "ensemble(S_{2j})" et des valeurs "ajoutées", par des opérations "PROD", depuis l'instant de cette opération "op", jusqu'à

"instant(S_{3k+1})"

--

4-2-3-Interprétation d'un type de communication-

Définition-

Considérons un objet de type "TC", défini par le triplet $\langle SP, A, SC \rangle$. La représentation des opérations init, prod, pré-cons, val-cons et post-cons sur un tel objet est réalisée à partir de la définition des suites temporelles suivantes:

. S_1 associée à l'ensemble produit "SP" est définie ci-dessus en 4-2-1-, avec les instants $t_1, t_2, t_3, \dots$ donnés, définis par l'activation des opérations de production par le processus producteur, définissant la suite temporelle:

$SP = \{(e_1, t_1), (e_2, t_2), \dots, (e_i, t_i), \dots\}$

. A l'ensemble consommé "SC" on associe deux suites temporelles:

S_C suite des "requêtes de consommation". Les éléments de type "CHRONO" de cette suite sont des données définies par l'activation des opérations de consommation par le processus consommateur,

$S_C = \{(\{\}, t'_1), (\{\}, t'_2), \dots, (\{\}, t'_k), \dots\}$

S_3 suite de "consommations", telle que:

$\forall k \geq 0 \quad \text{instant}(S_{3k}) \leq \text{instant}(S_{3k})$
 et $\text{instant}(S_{3k}) < \text{instant}(S_{3k+1})$

(cf. hypothèse (H) de la définition 4 ci-dessus)

-- l'inégalité entre "instant(S_{3k})" et "instant(S_{3k})" exprime une situation d'attente du processus consommateur

--

. La suite temporelle S_2 associée à l'ensemble consommable "A" est telle que:

$\forall j \geq 0 \quad \text{instant}(S_{2j}) = \text{instant}(S_{3j})$

Remarquons que l'identité des indices entre les suites temporelles S_2 , SC et S_3 , et l'égalité des instants des termes S_{2j} et S_{3j} signifient qu'on ne "tient compte" des productions dans l'ensemble "A" qu'au moment d'une nouvelle consommation, et non à chaque production.

On définit la fonction booléenne "REPRESENTE" sur $EC3 * ST * ST * TC$ par:

$REPRESENTE((S_{1i}, S_{2j}, S_{3k}), SP, SC, \langle SP, A, SC \rangle) =$
 $(ensemble(S_{1i}) = SP \text{ et } ensemble(S_{2j}) = A$
 $\text{et } ensemble(S_{3k}) = SC)$

Cette fonction est construite par les règles suivantes:

- . $REPRESENTE(INIT(), SP, SC, init()) = \text{vrai}$
- . $REPRESENTE(PROD([S_{1i}, S_{2j}, S_{3j}], SP),$
 $SP, SC, prod(\langle SP, A, SC \rangle, p)) =$
 $(REPRESENTE([S_{1i}, S_{2j}, S_{3j}], SP, SC, \langle SP, A, SC \rangle)$
 $\text{et } p = elt(S_{3i+1}))$
- . $REPRESENTE(CONS([S_{1i}, S_{2j}, S_{3j}], SC),$
 $SP, SC, post-cons(\langle SP, A, SC \rangle)) =$
 $REPRESENTE([S_{1i}, S_{2j}, S_{3j}], SP, SC, \langle SP, A, SC \rangle)$

où les fonctions "valeur" et "pre" sont définies par:

$valeur(S_{1i}, S_{2j+1}, S_{3j}) = val-cons(\langle SP, A, SC \rangle)$
 $\text{et } REPRESENTE([S_{1i}, S_{2j+1}, S_{3j}],$
 $SP, SC, \langle SP, A, SC \rangle) = \text{vrai}$

$pre(S_{1i}, S_{2j}, S_{3j}) = pre-cons(\langle SP, A, SC \rangle)$
 $\text{et } REPRESENTE([S_{1i}, S_{2j}, S_{3j}],$
 $SP, SC, \langle SP, A, SC \rangle) = \text{vrai}$

Cette représentation définit en fait un choix d'interprétation de la précondition de l'opération de consommation: dans un univers séquentiel, une opération est activable si et seulement si sa précondition est satisfaite. Si elle apparaît dans un choix non-déterministe, c'est une autre opération qui est activée -si possible. Si elle apparaît dans une séquence, le système est dans un état "d'erreur".

Dans l'univers parallèle que nous décrivons, une précondition peut être "tantôt" vraie, "tantôt" fausse: lorsqu'une opération gardée par cette précondition doit être activée, le processus concerné est dans une situation "d'attente" que la précondition soit mise à vrai par un autre processus du système, attente dont on doit généralement assurer qu'elle a une durée finie.

Exemples-

L'interprétation du parallélisme induit par un type de communication consiste donc essentiellement en la définition de la suite temporelle " S_2 ", d'où on déduit la suite " S_3 ", dont une part est triviale, ainsi que la suite " S_1 " déduite de la suite " SP " donnée. La suite " SC " des requêtes de consommation est donnée et vérifie l'hypothèse (H).

Pour s'en convaincre, étudions l'exemple du type de communication "aléatoire", présenté ci-dessus en 2-1-4- et dont on rappelle la définition:

$pre-cons(\langle SP, A, SC \rangle) = \text{non vide?}(A)$
 $val-cons(\langle SP, A, SC \rangle) = \text{dernier}(A)$
 $post-cons(\langle SP, A, SC \rangle) = \text{finir}(A)$

Définition de la suite temporelle S_2 :

D'après la définition générale de S_{2j+1} , pour $j > 0$, on obtient:

- $ensemble(S_{2j+1}) = \text{si } ensemble(B_j) < \{ \}$
 $\text{alors } ensemble(B_j)$
 $\text{sinon } \{P_i\} \text{ tel que}$
 $instant(S_{3i-1}) = \langle instant(SC_{j+1}) \langle instant(S_{3i}) \rangle$
 fsi
- $instant(S_{2j+1}) = \text{si } ensemble(B_j) < \{ \}$
 $\text{alors } instant(SC_{j+1})$
 $\text{sinon } instant(S_{3i}) \text{ tel que}$
 $instant(S_{3i-1}) \langle instant(SC_{j+1}) = \langle instant(S_{3i}) \rangle$
 fsi

avec B_j défini, pour $j > 0$, par:

- ensemble(Bj)=finir(ensemble(S2j))
 U ensemble(S1imax(i,j,j+1))
 - ensemble(S1lmax(i,j,j+1))

- instant(Bj)=instant(SCj+1)

On déduit que pour tout $j > 0$ ensemble(Bj) $\neq \{\}$, ce qui simplifie la définition précédente qui devient:

$$S2j+1 = B_j$$

Et pour $j=0$ on obtient:

- ensemble(S21)= si ensemble(B0) $\neq \{\}$ alors ensemble(B0)
 sinon { P1 }
 fsi

- instant(S21)=si ensemble(B0) $\neq \{\}$ alors instant(SC1)
 sinon instant(S11)
 fsi

- ensemble(B0)={Pi tels que instant(SPi)= $<$ instant(SC1)}

- instant(B0)=instant(SC1)

Une situation décrite par ce type de communication peut être illustrée par la figure-1 ci-dessous, qui correspond au schéma intuitif donné à la fin du chapitre 1.

elt(<u>S1i</u>)	ensemble(<u>S1i</u>)	ensemble(<u>Bj</u>)	ensemble(<u>S2j</u>)	ensemble(<u>S3k</u>)	<u>SCk</u>
$\vdots$ P_1^i	$S1^0 = \{\}$ $S1^1 = \{\dots, P_1^i\}$ $S1^4 = \{\dots, P_1^i, P_{i+1}^i\}$	$B_{j-1} = \{\dots, P_1^i\}$ $B_j = \{P_1^i, P_{i+1}^i\}$ $B_{j+1} = \{P_{i+1}^i\}$	$S2^0 = \{\}$ $S2^1 = \{\dots, P_1^i\}$ $S2^{j+1} = \{P_1^i, P_{i+1}^i\}$ $S2^{j+2} = \{P_{i+1}^i\}$	$S3^0 = \{\}$ $S3^j = \{\dots, P_1^i\}$ $S3^{j+1} = \{\dots, P_1^i, P_{i+1}^i\}$ $S3^{j+2} = \{\dots, P_1^i, P_{i+1}^i, P_{i+1}^i\}$ $S3^{j+3} = \{\dots, P_1^i, P_{i+1}^i, P_{i+1}^i, P_{i+1}^i\}$	$\vdots$ SCj $SCj+1$ $SCj+2$ $SCj+3$ $\vdots$
P_{i+2}^i	$S1^{i+2} = \{\dots, P_1^i, P_{i+1}^i, P_{i+2}^i\}$				
P_{i+3}^i	$S1^{i+3} = \{\dots, P_1^i, P_{i+1}^i, P_{i+2}^i, P_{i+3}^i\}$				
$\vdots$					

instants croissants

Figure - 1

On n'observe pas sur cet exemple de situation "d'attente" du processus consommateur: on a toujours $\text{instant}(\underline{SCj}) = \text{instant}(\underline{Sj})$, sauf éventuellement pour $j=1$, mais cette situation n'est pas représentée ici. Il n'en n'est pas de même dans l'exemple qui suit.

Considérons l'exemple du type de communication "croissante", dont on rappelle la définition:

```

pré-cons( $\langle SP, A, SC \rangle$ ) = q(A, SC)
                        ou vide?(SC) et dans?(A, 1)
val-cons( $\langle SP, A, SC \rangle$ ) = si q(A, SC)
                        alors extraire(A, r(A, SC))
                        sinon premier(A)
                        fsi
post-cons( $\langle SP, A, SC \rangle$ ) = si q(A, SC)
                        alors effacer(A, r(A, SC))
                        sinon décapiter(A)
                        fsi

```

L'interprétation dans le modèle présenté est définie à partir des éléments suivants:

Pour tout $j > 0$:

```

- ensemble( $\underline{S2j+1}$ ) = si q(ensemble( $\underline{Bj}$ ), ensemble( $\underline{SC1j+1}$ ))
                    alors ensemble( $\underline{Bj}$ )
                    sinon ensemble( $\underline{BCimin}$ )
                    fsi
    où  $\underline{imin}$  est le plus petit  $m(i, j, j+1)$  tel que:
     $q(\text{ensemble}(\underline{BCm}(i, j, j+1)), \text{ensemble}(\underline{SCj+1}))$ 

- instant( $\underline{S2j+1}$ ) = si q(ensemble( $\underline{Bj}$ ), ensemble( $\underline{SCj+1}$ ))
                    alors instant( $\underline{SCj+1}$ )
                    sinon instant( $\underline{BCimin}$ )
                    fsi

```

où la suite intermédiaire $\underline{B}$ est définie par:

```

- ensemble( $\underline{Bj}$ ) = extraire(ensemble( $\underline{S2j}$ ),
                        r(ensemble( $\underline{S2j}$ ), ensemble( $\underline{SCj+1}$ )))
     $\cup$  ensemble( $\underline{S1imax}(i, j, j+1)$ ) - ensemble( $\underline{S1lmax}(i, j, j+1)$ )
- instant( $\underline{Bj}$ ) = instant( $\underline{SCj+1}$ )

```

Et pour $j=0$:

```

- ensemble( $\underline{S21}$ ) = si ensemble( $\underline{B0}$ ) <> {} alors ensemble( $\underline{B0}$ )
                    sinon { P1 }
                    fsi

- instant( $\underline{S21}$ ) = si ensemble( $\underline{B0}$ ) <> {} alors instant( $\underline{SC1}$ )
                    sinon instant( $\underline{S11}$ )
                    fsi

```

Et $\underline{B0}$ défini comme précédemment.

Une situation décrite par cette interprétation peut être illustrée par la figure-2 ci-dessous, qui met en évidence des situations "d'attente" du processus consommateur.

elt($\underline{SE}_i$)	ensemble ($\underline{SE}_i$)	ensemble ($\underline{E}_i$)	ensemble ($\underline{SE}_i$)	ensemble ($\underline{SE}_i$)	$\underline{SE}_i$
$\underline{P}_1$	$\underline{SE}_0 = \{ \}$		$\underline{SE}_0 = \{ \}$	$\underline{SE}_0 = \{ \}$	
$\underline{P}_2$	$\underline{SE}_1 = \{ P_1 \}$	$\underline{E}_0 = \{ P_1, P_2 \}$	$\underline{SE}_1 = \{ P_1, P_2 \}$	$\underline{SE}_1 = \{ P_1 \}$	$\underline{SE}_1$
$\underline{P}_3$	$\underline{SE}_2 = \{ P_1 < P_2 \}$	$\underline{E}_1 = \{ P_2, P_3 \}$	$\underline{SE}_2 = \{ P_2, P_3 \}$	$\underline{SE}_2 = \{ P_1, P_2 \}$	$\underline{SE}_2$
$\underline{P}_4$	$\underline{SE}_3 = \{ P_1 < P_2 > P_3 \}$	$\underline{E}_2 = \{ P_3 \}$		attente	$\underline{SE}_3$
$\underline{P}_5$	$\underline{SE}_4 = \{ P_1 < P_2 > P_3 > P_4 \}$		$\underline{SE}_3 = \{ P_3, P_4, P_5 \}$	$\underline{SE}_3 = \{ P_1, P_2, P_5 \}$	$\underline{SE}_3$
$\underline{P}_6$	$\underline{SE}_5 = \{ P_1 < P_2 > P_3 > P_4 < P_5 \}$	$\underline{E}_3 = \{ \}$		attente	$\underline{SE}_4$
$\vdots$	$\underline{SE}_6 = \{ P_1 < P_2 > P_3 > P_4 < P_5 < P_6 \}$		$\underline{SE}_4 = \{ P_6 \}$	$\underline{SE}_4 = \{ P_1, P_2, P_5, P_6 \}$	$\vdots$

instants croissants

Figure - 2

CHAPITRE 5- PROPRIETES D'UNE COMMUNICATION

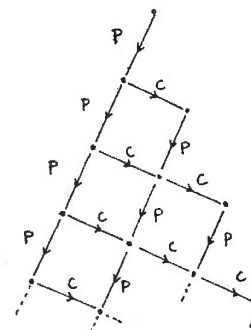
5-1- AUTOMATE ASSOCIE-

5-1-1-Définition-

A tout type de communication on associe un automate figuré par un graphe orienté représentant les transitions possibles d'un système réduit à l'activation d'opérations "produire" ou "consommer" sur une donnée communiquée associée à un port de ce type.

Ce graphe est constitué de noeuds représentant les états du dit système tels qu'ils sont définis au chapitre 4. Les arcs sont étiquetés par le nom d'une opération sur la donnée communiquée: on note "P" pour une opération de production et "C" pour une opération de consommation.

Exemple:



Ce graphe est défini comme suit:

- il respecte l'axiome suivant (analogue à l'axiome "du séquençement bien formé" dans (Rou_78')): conformément à l'interprétation définie au chapitre précédent, une opération "C" est activable si et seulement si, à l'état associé au noeud origine, l'opération pré-cons($\langle SP, A, SC \rangle$) a pour résultat vrai,

- Les étiquettes "P" et "C" sont indicées de la manière suivante:

- . tout arc "P" est indicé par l'indice, dans l'ensemble produit, de l'élément résultant de l'occurrence de production associée,

- . tout arc "C" est indicé par

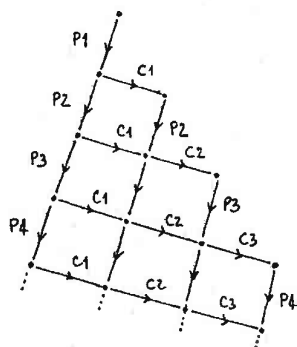
- s'il existe, l'indice, dans l'ensemble consommable, de l'élément résultant de l'occurrence de consommation associée
(indice "j" tel que val-cons($\langle SP, A, SC \rangle$) = extraire(A,j)),

- ω , si ω est le résultat de cette opération.

- Les productions sont étiquetées dans l'ordre croissant des indices.

Exemple:

Ainsi complété le graphe ci-dessus est associé au type de communication "égalité":



Remarque 1:

L'ensemble des états d'un tel automate est infini. On représentera généralement le graphe associé et on étudiera ses propriétés par sa "restriction à l'ordre i ", c'est-à-dire ne comportant pas de chemin constitué de " i " productions consécutives.

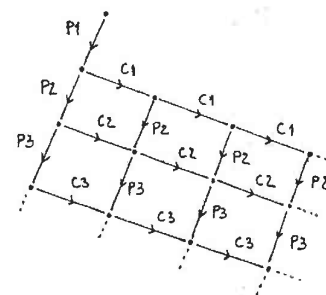
Remarque 2:

Un tel graphe n'est défini que pour des types de communications où l'axiome du séquençement bien formé peut être vérifié statiquement. Ce n'est pas toujours le cas: en particulier pour les types "croissante" et "non-répétitive", présentés au chapitre 2, où l'opération pré-cons fait intervenir les valeurs des ensembles consommable et consommé. Cependant nous verrons que, pour ces situations particulières, des propriétés intéressantes peuvent être observées à l'aide de considérations très partielles sur l'automate correspondant.

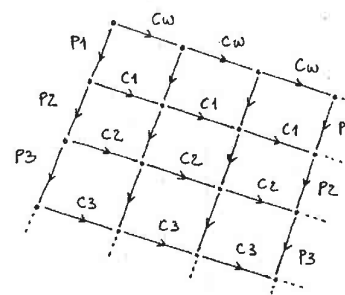
5-1-2-Exemples-

Nous reprenons les exemples de types de communication présentés au chapitre 2.

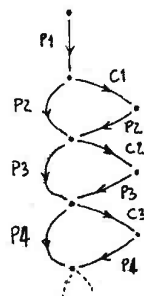
type "aléatoire":



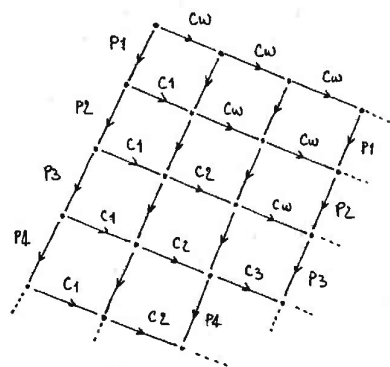
type "très-aléatoire":



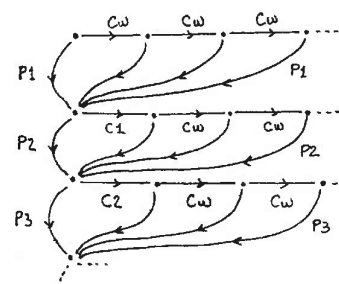
type "rafraichie":



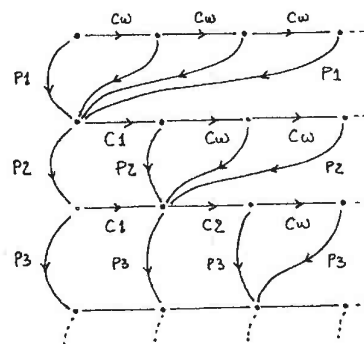
type "égalité-presque-partout":



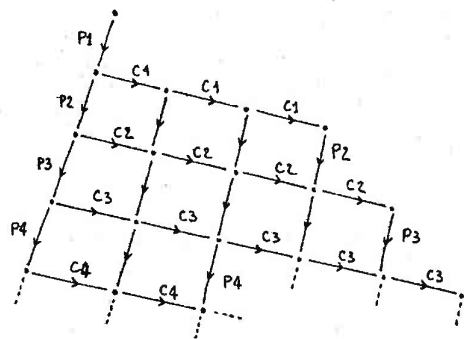
type "rafraichie-avec-trous":



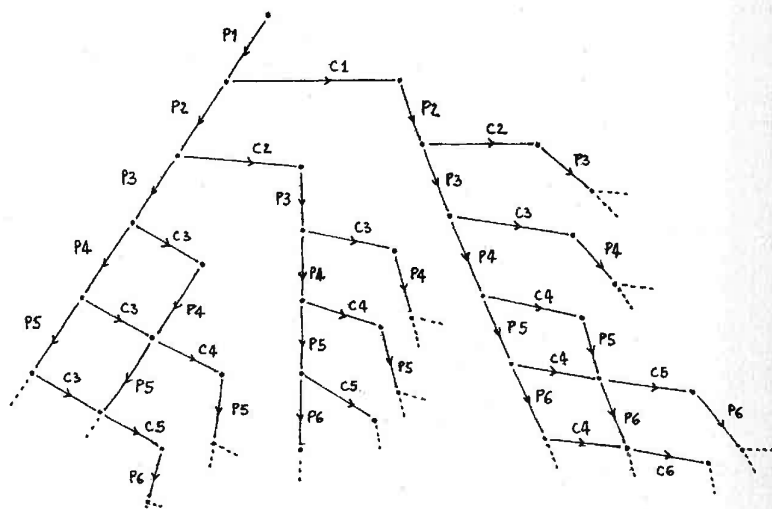
type "égalité-avec-trous":



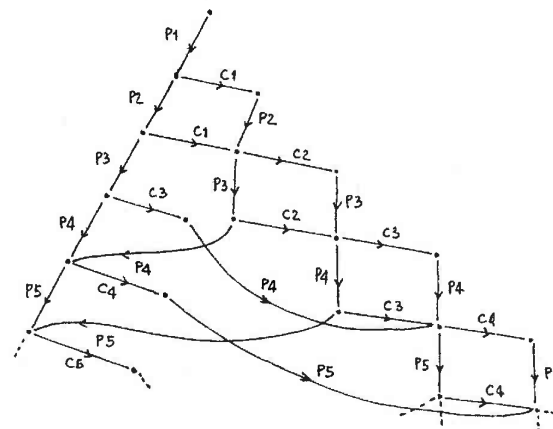
type "à-retard-borné": (cas $p=2$)



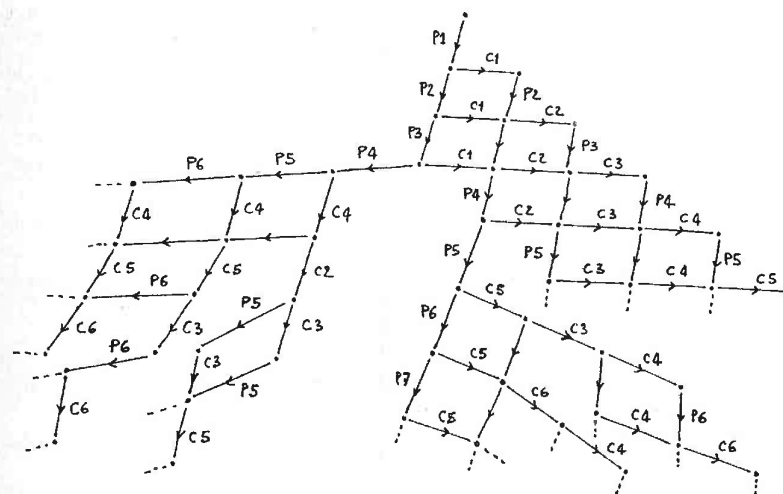
type "échantillon": (cas $p=2$)



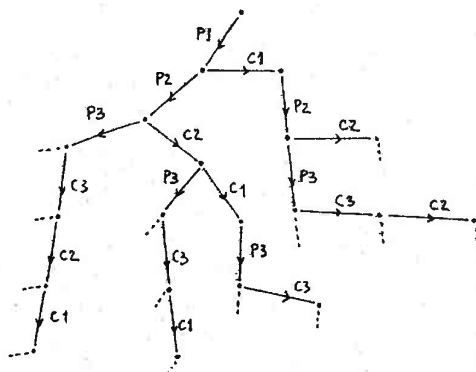
type "élastique": (cas $p=2$)



type "circulaire": (cas $p=3$)



type "pile":



5-1-3-Ordres de communications-

La définition de ces graphes permet d'introduire deux relations d'ordre partiel sur l'ensemble des types de communication. Ces relations sont telles que si, pour une donnée communiquée, on substitue un type de communication "TC2" à un type de communication "TC1", tels que "TC1" et "TC2" soient ordonnés dans cet ordre dans l'une de ces relations, le système devient "plus asynchrone", dans un certain sens que nous préciserons ci-dessous.

Définition 1:

On définit la relation "as<" suivante sur l'ensemble des types de communication:

$TC1 \text{ as} < TC2 \text{ ssi}$

- tout chemin dans le graphe associé à TC1 est un chemin du graphe associé à TC2,

- sur les arcs correspondants, les indices du graphe de

TC1 sont inférieurs ou égaux aux indices du graphe de TC2 (par hypothèse ω est inférieur à tout indice).

Propriété:

La relation "as<" est une relation d'ordre partiel sur l'ensemble des types de communication.

Exemples:

Pour les types de communication définis au chapitre 2, on a les relations suivantes:

- "égalité" as< "aléatoire"
- "rafraichie" as< "échantillon" as< "aléatoire"
- etc.

Définition 2:

On définit la relation "op<" suivante sur l'ensemble des types de communication:

$TC1 \text{ op} < TC2 \text{ ssi}$

- tout chemin sur le graphe associé à TC1 est un chemin du graphe associé à TC2,

- sur les arcs correspondants les indices de TC1 et de TC2 sont égaux.

Propriété:

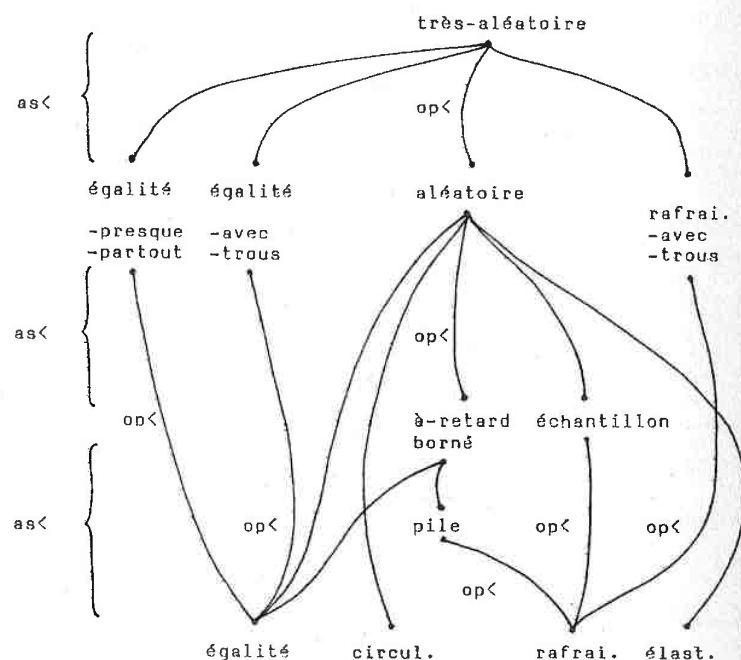
La relation "op<" est une relation d'ordre partiel sur l'ensemble des types de communication, telle que:

$$TC1 \text{ op} < TC2 \implies TC1 \text{ as} < TC2$$

Exemples:

- "égalité" op< "égalité-presque-partout"
- "rafraichie" op< "rafraichie-avec-trous"
- etc.

Pour les types de communication définis au chapitre 2, on obtient le diagramme suivant:

5-1-4-Application aux modifications de programmes-

Aux deux relations d'ordre précédentes correspondent deux types différents de modifications de programmes parallèles: par "modification", nous entendons modification des textes de programmes (en l'espèce, des types de communication) impliquant des modifications des séquences d'opérations exécutées par ces programmes. Il ne s'agit pas des techniques habituelles de "transformation" de programmes préservant la sémantique (fusion, pliage, dépliage, etc.).

Pour une donnée communiquée, lorsqu'on substitue un type de communication IC2 à un type de communication IC1, tel que IC1 as< IC2 et tout arc de IC2 étiqueté par est également dans IC1, sans changer les opérations dans les processus, on obtient un programme "plus asynchrone" au sens suivant: toutes les valeurs produites ne sont pas nécessairement consommées et les situations d'attente du processus consommateur sont plus rares. L'algorithme obtenu n'étant plus défini sur les mêmes ensembles de données, on doit démontrer qu'il est correct vis-à-vis des spécifications du problème.

C'est ce type d'"asynchronisation" qui est utilisé dans les exemples suivants:

- factorielle, en substituant le type "aléatoire" au type "égalité",
- calcul de point fixe, en substituant "aléatoire" à "à-retard-borné", lui-même substitué à "rafraichie", lui-même substitué à une forme particulière de "égalité" (voir ci-dessous en 5-2-2-),
- algorithme de Newton, en substituant "aléatoire" à "égalité".

Pour une donnée communiquée, lorsqu'on substitue un type de communication IC2 à un type de communication IC1, tels que IC1 op< IC2, en modifiant éventuellement certaines opérations des processus (pour tenir compte de la consommation de la valeur ω), on obtient un programme défini sur les mêmes ensembles de données, "plus asynchrone" que le précédent, au sens suivant: les situations d'attente du processus consommateur sont plus rares.

Cette philosophie d'"asynchronisation", qui ne modifie pas les ensembles de données traitées par l'algorithme, est appliquée dans les situations suivantes:

- substitution de cas particuliers d'implantations du type "égalité" (voir ci-dessous en 5-2-2-) pour exprimer des échanges de messages sans mémorisation, via un tampon borné, ou une file non bornée.

- substitution du type "égalité-presque-partout" au type "égalité" dans un système du type suivant:

processus S ::

port x: égalité_i

processus P ::

sortie x: ... i

entier i;

tableau(1..?) de ... a;

corps

i:=1;

*[-- calculer ai=F(ai-1) --

x:=ai; produire(x);

-- utiliser ai --

i:=i+1

]

fin P;

processus Q ::

entrée x: ... i

entier i;

tableau(1..?) de ... a;

corps

i:=1;

*[consommer(x);

ai:=x;

-- utiliser ai --

i:=i+1

]

fin Q;

composition

P // Q

fin S;

La substitution du type "égalité-presque-partout" au type "égalité" associé à la donnée communiquée "x" et la substitution de l'énoncé

$$\begin{array}{l} \boxed{x \langle \rangle \omega} \quad \text{-->} \quad \text{ai:=x} \\ \boxed{x = \omega} \quad \text{-->} \quad \text{--calculer ai=F(ai-1) --} \\ \boxed{} \end{array}$$

à l'énoncé

ai:=x

définit un algorithme asynchrone, sans attente du processus Q. Une version "symétrique" de ce type d'algorithme a été présentée avec les exemples du calcul du PGCD ou des coefficients du binôme, ou, plus généralement les algorithmes itératifs asynchrones simples (Kun_76).

Pour l'une et l'autre de ces philosophies de modification, l'intérêt du diagramme présenté plus haut est de mettre en évidence les seules voies possibles pour l'"asynchronisation" des programmes; la validité de ces modifications devant être vérifiée comme il est dit ci-dessus.

5-2- GRAPHE D'IMPLANTATION-

Les graphes présentés au paragraphe précédent sont suffisants pour définir les relations d'ordre "as<" et "op<". Ils ne rendent pas compte du schéma de contrôle dans son ensemble, qui exprime le parallélisme d'un système: en particulier qu'une opération n'est pas toujours activable et peut engendrer une "attente".

Nous précisons donc ces graphes en définissant l'automate sur un alphabet composé, pour chaque opération de communication, d'un symbole de "requête" et d'un symbole de "terminaison". Nous retrouvons ainsi la représentation des schémas de contrôle présentés dans (Rou_78'), à partir desquels nous déduisons les implantations possibles d'un type de communication.

5-2-1-Définition-

Soit $\Sigma = \Sigma_r \cup \Sigma_t$,

où $\Sigma_r = \bigcup_{A \in PC} \bar{A}$ est l'ensemble des requêtes d'opérations,

$\Sigma_t = \bigcup_{A \in PC} A$ l'ensemble des terminaisons.

(PC est l'ensemble des occurrences d'opérations "produire" ou "consommer" activées sur une donnée communiquée).

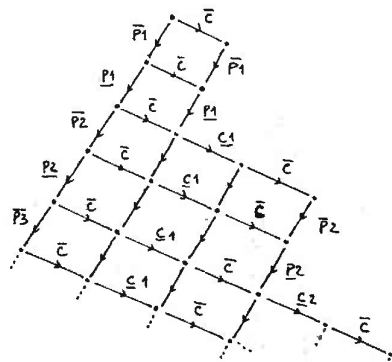
On appelle graphe d'implantation d'une communication, le graphe représentant l'automate associé à un type de communication, défini sur l'alphabet par les règles précisées en 5-1-1-, auxquelles on ajoute l'axiome suivant (dit d'activation préalable dans (Rou_78')): toute terminaison d'opération est précédée de sa requête.

Remarque:

L'indiquage des "arcs de consommation" est défini sur les terminaisons de consommation.

Exemple:

Graphe d'implantation associé au type "égalité"



5-2-2-Protocole et priorité-

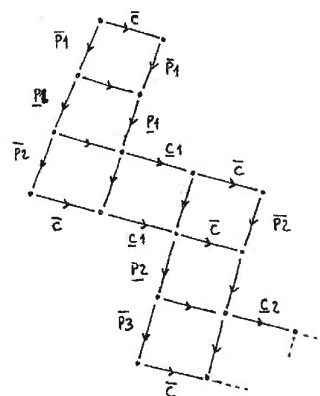
Etant donné un graphe d'implantation d'un type de communication, tout sous-graphe définit une sur-spécification de ce type de communication telle que si "TC" est le type de communication donné et "STC" le sous-graphe considéré: $STC \text{ op} < TC$.

Ces sur-spécifications définissent des représentations particulières du type de communication en introduisant des synchronisations supplémentaires, qui conduisent éventuellement à de nouvelles situations d'attente des processus producteur ou consommateur. Opposée à la technique d'"asynchronisation" des programmes traitée précédemment, il s'agit là d'une technique de "synchronisation" de programme à des fins de représentations particulières dues à des contraintes d'implantation.

Deux situations sont, à cet égard, exemplaires:

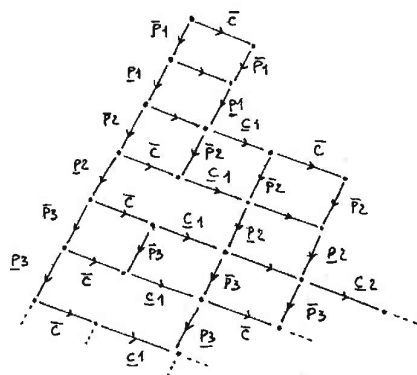
- la définition d'un "protocole" pour la communication, qui introduit des synchronisations pour le processus producteur: elle consiste à ne considérer qu'un sous-graphe "d'ordre n", c'est-à-dire ne comportant pas de chemins constitué de "n" terminaisons de production consécutives.

Exemple: type de communication "égalité" avec protocole d'ordre 2:

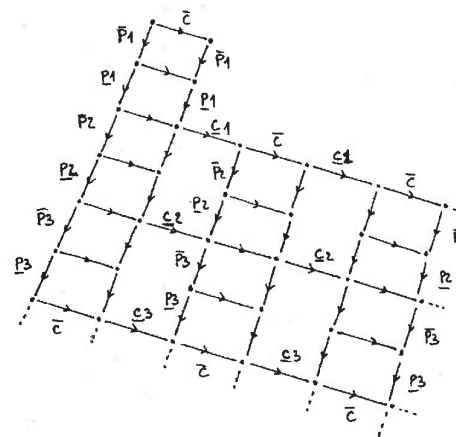


- la définition d'une "priorité", en cas de requêtes de production et de consommation. Ce type de sur-spécification est exprimé en supprimant certains chemins dans le graphe d'implantation, induisant d'éventuelles attentes de l'un ou l'autre des processus.

Exemple 1: type de communication "égalité" avec priorité au consommateur:



Exemple 2: type de communication "aléatoire" avec priorité au producteur:

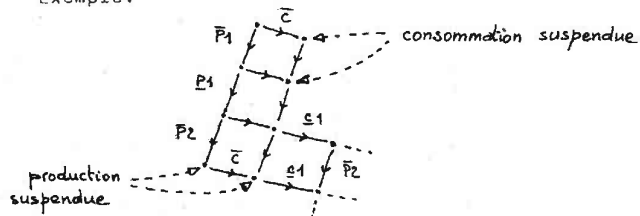


5-2-3-Equité et non-blocage-

Définition:

Une occurrence d'opération A est dite "suspendue", pour un graphe d'implantation donné, si les arcs étiquetés $\bar{A}$ et $\underline{A}$ ne sont pas contigus dans le graphe.

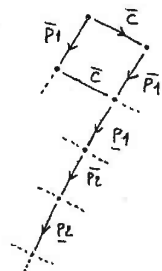
Exemple:



Axiome d'équité: (Mar_81)

Lorsqu'une occurrence d'opération est suspendue il n'existe pas de chemin de longueur non bornée étiqueté seulement par des occurrences de l'opération de communication symétrique.

Exemple: cette partie de graphe ne satisfait pas l'axiome d'équité:



Axiome de non blocage: (Mar_81)

En tout noeud il y a au plus une occurrence d'opération suspendue.

5-3- IMPLANTATION D'UNE COMMUNICATION-

5-3-1-Noeuds d'observation-

Etant donné un graphe d'implantation, on appelle noeud d'observation un noeud tel que

. ou bien tout arc en entrée est étiqueté par une terminaison d'opération,

. ou bien il existe deux arcs en entrée, l'un étiqueté par une terminaison d'opération, l'autre correspondant à une

occurrence d'opération suspendue,

. ou bien il existe un seul arc en entrée, correspondant à une occurrence d'opération suspendue.

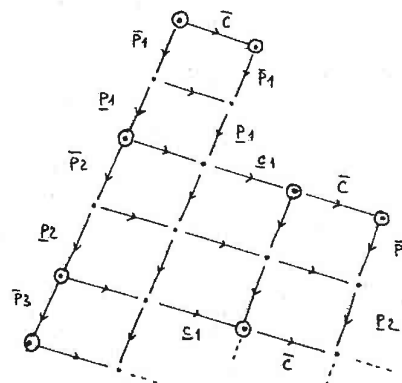
Remarque 1

La racine du graphe est un noeud d'observation.

Remarque 2:

Les autres noeuds d'un graphe d'implantation sont sans signification supplémentaire quant aux propriétés de synchronisation d'un système parallèle. Ils expriment seulement qu'une opération est en cours, en incluant implicitement une notion de durée des opérations, pendant laquelle l'observation du système n'est pas pertinente (cf. la notion "d'état imparfait" dans (FiO_83)).

Exemple: sur le graphe suivant les noeuds d'observation sont marqués d'un "O":



Définition:

En tout noeud d'observation d'un graphe d'implantation, on définit les compteurs entiers $\#P$ et $\#C$ par les règles suivantes:

- le compteur $\#P$ est un compteur classique du nombre d'activations d'opérations de production. Par contre, le compteur $\#C$ est caractérisé par l'indice des éléments considérés dans l'ensemble consommable du type de communication. Il n'est pas défini comme le nombre d'activations d'opérations "consommer", comme cela est le cas dans les modèles du parallélisme basés sur le concept de synchronisation
-
- . à la racine $\#P = \#C = 0$
- . s'il existe un arc d'entrée étiqueté par une production " P_i " (requête P_i ou terminaison P_i)
 - $\#P = i$
 - $\#C = j$ tel que j est le plus petit indice d'une étiquette C_j de terminaison de consommation rencontrée à partir de ce noeud, s'il en existe une, en suivant une branche de "consommations", = $\#P$ s'il n'en existe pas ou si elle est étiquetée par ω .
- . si aucun arc en entrée n'est étiqueté par une production,
 - . si un arc d'entrée est étiqueté par une terminaison de consommation C_j
 - $\#P$ est inchangé: il est égal au nombre $\#P$ d'un noeud d'observation précédent (Ils ont tous la même valeur),
 - $\#C = j$
 - . si l'arc en entrée est étiqueté par $C\omega$,
 - $\#C = \#P$
 - . si le seul arc en entrée est une requête de consommation $\bar{C}$
 - $\#P$ et $\#C$ sont inchangés.

Axiomes:

Tout graphe d'implantation vérifie les axiomes suivants:

Axiome de l'écart: (cf. the "boundedness requirement" dans (Mar_81))

En tout noeud d'observation d'un graphe d'implantation on a

$$0 \leq \#P - \#C \leq M \quad \text{où } M \text{ est une constante entière, éventuellement infinie.}$$

D'où trois situations caractéristiques d'une implantation d'un type de communication selon que M (the "slack" dans (Mar_81)) est nul, fini ou infini (cf. le théorème ci-dessous).

Axiome du non blocage:

On peut réécrire cet axiome de la manière suivante: si PS et CS sont le nombre d'occurrences d'opérations suspendues à un noeud on a

$$(0 \leq PS + CS \leq 1) \text{ et } (PS = 0 \text{ ou } \#P = \#C + M) \text{ et } (CS = 0 \text{ ou } \#P = \#C)$$

5-3-2-Théorème-

Etant donné un graphe d'implantation " I ", dont on note " T_n " tout sous-graphe d'ordre n (c'est-à-dire ne comportant pas de chemin constitué de " n " terminaisons d'opérations de production consécutives),

si pour tout $n > 0$, M_n est tel que, sur T_n :

$$0 \leq \#P - \#C \leq M_n$$

$$M = \limsup_{n \rightarrow +\infty} M_n$$

est la taille nécessaire et suffisante du "tampon de communication",

C'est-à-dire que le type de communication, éventuellement sur-spécifié par un sous arbre d'implantation, peut être réalisé par des procédures manipulant une structure de donnée partagée composée de "M" éléments.

Concrètement,

- si $M=0$, la communication est réalisée sans mémorisation de donnée, c'est-à-dire par un "rendez-vous",
- si M est fini, le "buffer" de communication contient au plus M éléments,
- si M est infini, les processus communiquent via une file non bornée.

Démonstration:

On applique un théorème démontré dans (Mar_81) où les compteurs représentent le nombre d'activations de primitives symétriques de synchronisation.

• si $M=0$ alors $\#P = \#C$: $\#C$ représente donc le nombre d'activations de l'opération "consommer", et la propriété est démontrée.

• si $M>0$ on substitue une séquence d'opérations "recevoir" à l'opération "consommer", telle que tous les éléments de l'ensemble consommable, jusqu'à l'élément "consommé", soient "reçus": $\#C$ est alors égal au nombre d'activations de l'opération "recevoir", et la propriété est démontrée pour le couple d'opérations symétriques "produire" et "recevoir".

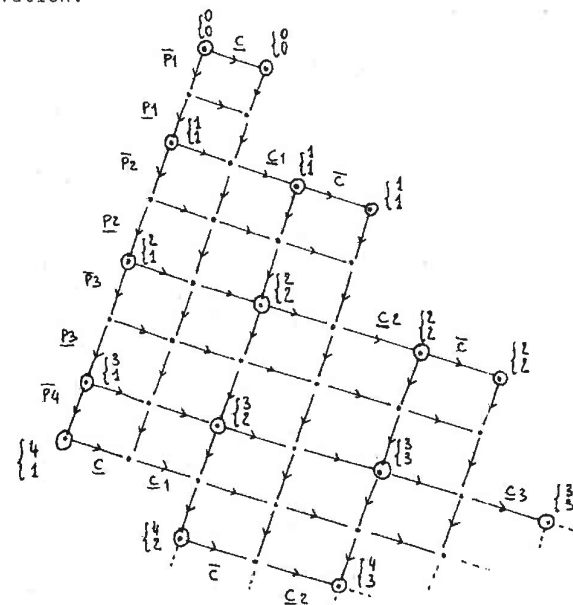
Remarque:

Pour tout graphe T_n d'ordre n , on obtient la valeur maximum de $\#P - \#C$ en suivant le chemin comportant le plus de productions, et sur ce chemin, au noeud correspondant à une attente de production. On peut donc se limiter à l'étude de ce noeud.

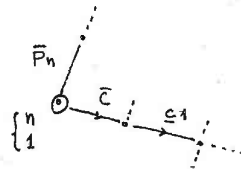
Exemples:

Implantations du type "égalité":

Considérons le graphe suivant, associé au type "égalité", sur-spécifié par un protocole d'ordre n (ici $n=4$), sur lequel on indique les valeurs $\#P$ et $\#C$ en tout noeud d'observation:



De même, pour tout $n>0$:

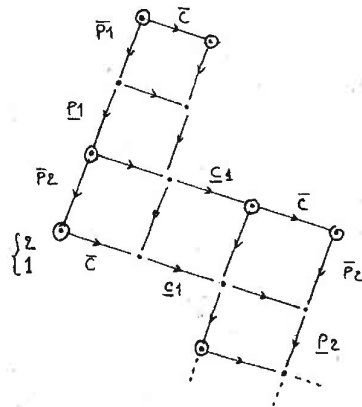


$$0 \leq \#P - \#C \leq n-1$$

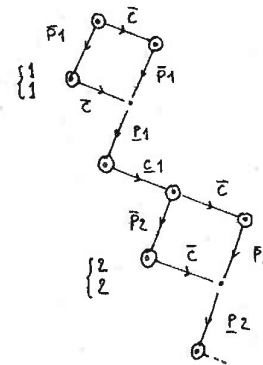
d'où, pour le type de communication "égalité", $M = +\infty$, les processus communiquent via une file non bornée.

On déduit également que ce type sur-spécifié par un protocole d'ordre $n > 0$ est implanté par un "buffer" de taille $n-1$, avec les deux cas particuliers suivants:

- communication via un tampon à une place, décrite par le sous graphe suivant:

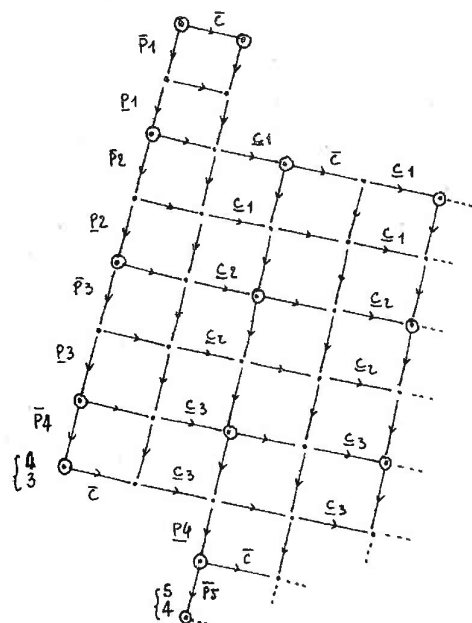


- communication par "rendez-vous", décrite par le sous graphe suivant:



Implantation du type "aléatoire":

Pour tout $n > 0$ (ici $n=4$), on définit le sous arbre d'ordre n suivant:



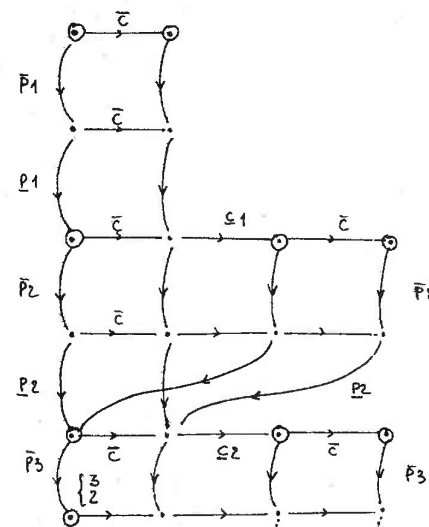
qui est tel que $0 \leq P - C \leq 1$

d'où on déduit que pour le type "aléatoire" $M=1$, et que ce type de communication peut être réalisé par un tampon à une place.

Note: on a la même propriété pour le type "très-aléatoire".

Implantation du type "rafraichie":

De même, de l'arbre suivant on déduit que $M=1$, c'est-à-dire une implantation par un tampon de taille un.



Remarque:

On peut alors observer les nouvelles relations suivantes entre communications:

"rendez-vous" $op <$ "tampon-à-une-place" $op <$ "égalité"
 $op <$ "rafraichie"

Théorème:

Les types de communication "rendez-vous" et "très-aléatoire" sont respectivement le plus petit et le plus grand élément de l'ensemble des types de communication pour la relation " $as <$ ".

Démonstration:

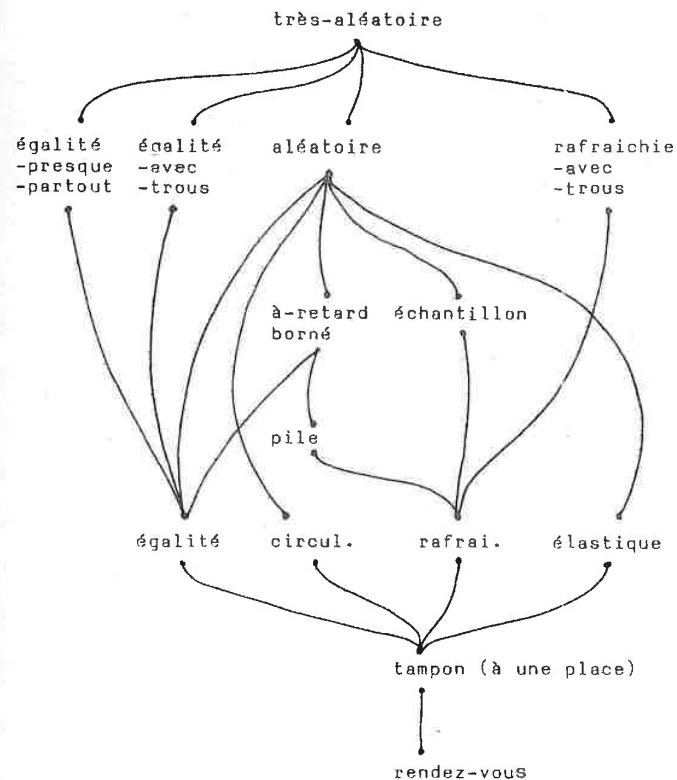
. Tout chemin du graphe associé au type "rendez-vous" est un chemin de tout graphe d'implantation de tout type de communication.

Les indices des arcs de consommation sont les minima des ensembles d'indices de production (on ne considère pas ici un type de communication définissant "a" comme résultat de toute consommation). Ce qui démontre la propriété du plus petit élément.

. Le graphe du type "très-aléatoire" contient les chemins du graphe de tout type de communication.

Les indices des arcs de consommation sont les maxima des ensembles d'indices de production (on ne considère pas ici un type de communication permettant la consommation d'un élément qui ne soit pas dans l'ensemble consommable). Ce qui démontre la propriété du plus grand élément.

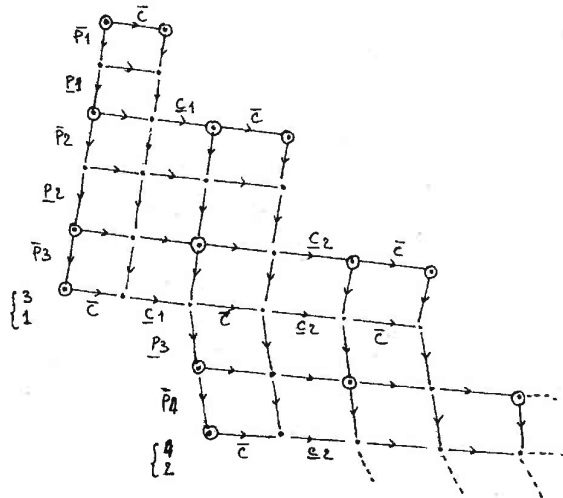
On peut alors compléter le diagramme présenté en 5-1-3-, pour former le treillis suivant:



Autres types de communication:

On montre facilement les résultats suivants:

- . type "égalité-presque-partout": $M=+\infty$
- . type "rafraichie-avec-trous": $M=+\infty$
- . type "égalité-avec-trous": $M=+\infty$
- . type "à-retard-borné": $M=1$
- . type "échantillon": $M=+\infty$
- . type "pile": $M=+\infty$
- . type "élastique": cas($p=2$)

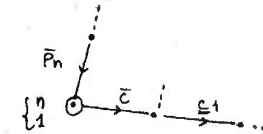


d'où $M=2$

De manière générale: $M=p$

. de même, pour le type "circulaire": $M=p$

. pour les types où l'opération pré-cons fait intervenir les valeurs des éléments des ensembles consommable ou consommé (exemples des types "croissante" et "non-répétitive"), il est impossible de définir un graphe d'implantation. Cependant on a toujours



c'est-à-dire $0 \leq \#P - \#C \leq n-1$, d'où on déduit $M=+\infty$.

CHAPITRE 6- VERIFICATION D'UN SYSTEME PARALLELE

Nous abordons dans ce chapitre le problème de la vérification des systèmes parallèles, exprimés sous la forme de processus communicants présentée dans cette thèse.

Nous rappelons d'abord les différentes approches présentées dans la littérature: extensions de la "logique de Hoare", utilisation de la logique temporelle, etc. Ces approches sont associées à une expression de la synchronisation d'opérations complémentaires par "rendez-vous", ou à une expression de la concurrence par l'accès à des données partagées. Elles conduisent à la définition de règles d'inférence particulières au "domaine parallèle" considéré.

Au contraire, le modèle que nous avons présenté conduit à la définition d'un système de preuve "relativement" classique des programmes non déterministes. Pour ce, nous définissons la "sérialisation" d'un système parallèle, puis nous proposons les axiomes et les règles d'inférence d'un système de preuve "à la Hoare".

Une dernière partie est une illustration sur différents exemples déjà traités au chapitre 3.

6-1- RAPPELS BIBLIOGRAPHIQUES-6-1-1-Introduction-

La vérification, ou plus généralement, l'étude des propriétés d'un système parallèle comporte deux aspects:

- l'étude de propriétés invariantes, ou de "sûreté" ("safety properties" dans la littérature anglo-saxonne) du système, qui spécifient qu'il ne doit pas atteindre un état "catastrophique". Des exemples de telles propriétés sont:

- . la correction partielle: s'il termine, un programme satisfait sa post-condition,
- . l'absence de blocage: le système ne peut pas atteindre un état à partir duquel il ne peut plus évoluer,
- . etc.,

- l'étude de propriétés non invariantes, ou d'"accessibilité" ("liveness properties") du système, qui spécifient que le système doit évoluer jusqu'à atteindre un état satisfaisant. Des exemples de telles propriétés sont:

- . la terminaison: le système doit atteindre, dans un laps de temps fini, son état "final",
- . l'équité ("fairness"): toute opération activable une infinité de fois dans une exécution infinie, doit être activée dans un laps de temps fini,
- . le "délai fini" ("finite delay property"): toute requête de service est satisfaite dans un laps de temps fini,
- . etc

Plus intuitivement, et selon une autre classification, la vérification d'un système parallèle concerne deux aspects complémentaires:

1. la vérification que le système "calcule" les résultats attendus, ou rend le service spécifié (correction

partielle),

2. la vérification que le système a un "bon comportement" (non blocage et propriétés non invariantes ci-dessus).

Il faut noter d'une part, que selon le type de problème traité, la distinction entre ces deux aspects est plus ou moins nette (par exemple, le comportement du système peut être considéré comme étant une contrainte spécifiée du service attendu), d'autre part, que selon les auteurs et les classes de problèmes traités, on privilégie dans la littérature l'un ou l'autre de ces aspects.

Une analyse chronologique des techniques de preuves de programmes parallèles fait apparaître différentes approches:

- extension de la "logique de Hoare" (Hoa_69) à des programmes parallèles avec données partagées: nous rappelons les travaux de Owicki ((OwG_76), (OwG_76'), etc.),

- extension de cette même logique à des processus communicants par échange de messages: l'essentiel des travaux considèrent des systèmes de preuve de CSP. La "référence" à ce sujet sont les travaux de Apt: citons, parmi une littérature abondante (Apt_80), (Apt_83), (Apt_84). D'autres langages à "rendez-vous" sont étudiés avec un point de vue voisin: par exemple ADA (cf. (GeR_84) ou (GeR_84')).

Le point commun de ces travaux est une démarche en deux temps dans la vérification des programmes: preuve des différents processus d'une part, dans un système classique, et "coopération" (sous différentes formes selon les auteurs) de ces preuves d'autre part, grâce aux règles d'inférence particulières de l'univers parallèle considéré.

- introduction d'autres logiques, de type "temporelles": la "coopération" des preuves, dans les approches précédentes, exprime la "non-interférence" d'un processus sur la preuve d'un autre processus parallèle. Cette idée peut être directement traduite dans le système formel de preuve, qui permet l'expression de l'évolution dans le temps de l'état global du programme. C'est ce qui est proposé par Lamport avec la logique GHL ("Generalized Hoare Logic") dans (Lam_80) appliquée à différentes expressions du parallélisme (LaS_82). C'est également l'intérêt de la "logique temporelle" (Pnu_77) appliquée à des programmes parallèles avec données partagées

(cf. par exemple (Pnu_81) ou (MaP_81)).

6-1-2-Extension de la "logique de Hoare" à des systèmes avec données partagées-

Le système de preuves défini dans (OwG_76) ou (OwG_76') est proposé pour un langage d'expression du parallélisme, dans lequel on note

cobegin S1 // S2 // ... // Sn coend

la composition parallèle de processus séquentiels, qui coopèrent par des énoncés de la forme

await b then S

où "b" est une expression booléenne et "S" un énoncé ne contenant ni composition parallèle, ni énoncé du même type. Lorsqu'un processus est prêt à exécuter un tel énoncé, il est suspendu jusqu'à ce que la condition "b" soit vraie. L'énoncé "S" est considéré comme une action indivisible.

Ces processus traitent des données partagées, dans les conditions ci-dessus.

Ce système de preuves est une extension du système de Hoare défini pour les programmes séquentiels (Hoa_69), que nous ne rappelons pas ici. Notons simplement les deux règles d'inférence suivantes, liées au modèle de parallélisme retenu:

R1.
$$\frac{\{p \wedge b\} \quad S \quad \{q\}}{\{p\} \quad \text{await } b \text{ then } S \quad \{q\}}$$

R2.
$$\frac{\{p_i\} \quad S_i \quad \{q_i\} \quad , \quad i=1,n \text{ sont sans interférence}}{\{p_1 \wedge \dots \wedge p_n\} \text{cobegin } S_1 // \dots // S_n \text{coend} \{q_1 \wedge \dots \wedge q_n\}}$$

où la non interférence est définie de la façon suivante:

- étant donnée une preuve de $\{p\} \quad S \quad \{q\}$ et un énoncé "I", dont on note pré(I) la précondition, I n'interfère pas sur $\{p\} \quad S \quad \{q\}$ si:

• $\{q \wedge \text{pré}(T)\} \quad T \quad \{q\}$

et . pour tout énoncé S' interne à S (mais pas dans un "await"):

$\{\text{pré}(S') \wedge \text{pré}(T)\} \quad T \quad \{\text{pré}(S')\}$

- les preuves de $\{p_1\} S_1 \{q_1\}, \dots, \{p_n\} S_n \{q_n\}$ sont sans interférence si, pour tout énoncé " T " dans un processus S_i , énoncé du type await ou affectation d'une donnée partagée, pour tout $j \neq i$, T n'interfère pas sur la preuve de $\{p_j\} S_j \{q_j\}$.

Rappelons enfin la démarche de vérification associée à ce système, qui procède en deux temps:

- d'abord la preuve indépendante de chacun des processus séquentiels, par induction d'assertions dans le système de Hoare classique,

- ensuite la vérification de la non-interférence de ces preuves.

La même idée est développée ci-dessous, sous la forme de "coopération de preuves", pour des programmes parallèles sans donnée partagée, avec communications par "rendez-vous".

6-1-3-Coopération de preuves-

On propose dans (AFR_80) un système de preuves pour CSP: démonstration de correction partielle et d'absence de blocage. La consistance (the "soundness") et la complétude (the "completeness") de ce système sont démontrés dans (Apt_83). Un système du même type est proposé dans (GeR_84) et (GeR_84') pour le langage ADA.

Le point commun de ces approches est la notion de coopération de preuves et d'invariant global: la première est une façon d'exprimer la non-interférence des preuves des processus séquentiels; la seconde permet de spécifier les opérations de communication sémantiquement complémentaires, c'est-à-dire les paires

$P_i!x \quad // \quad P_j?y$

qui peuvent donner lieu à un "rendez-vous" à l'exécution.

On propose dans (LeG_81) un système de preuve de correction totale de programmes CSP, très voisin, où la non-interférence des preuves est définie par une "règle de satisfaction" pour des paires d'opérations de communication sémantiquement complémentaires, définies par des "variables auxiliaires" figurant dans ces opérations.

Le système de preuves de CSP-

Nous ne présentons pas ici l'ensemble des axiomes et des règles d'inférence de ce système. Nous rappelons seulement ceux et celles qui concernent le modèle de parallélisme retenu, soit au niveau de chaque processus, pour tenir compte des opérations d'émission et de réception dans les processus non-déterministes, soit pour exprimer la "coopération" des preuves dans le programme parallèle.

- Axiome de la réception: $\{p\} \quad P_i?x \quad \{q\}$

- Axiome de l'émission: $\{p\} \quad P_i!y \quad \{q\}$

où " q " est déterminé la coopération des preuves (voir ci-dessous).

Remarque-

A ces post-assertions arbitraires, finalement déterminées par la règle de coopération, d'autres auteurs (voir (MiC_81)) opposent des assertions complètement déterminées portant sur des "traces" de l'histoire des messages émis ou reçus par chaque processus. La preuve de chaque processus utilise alors de façon autonome ses propres traces considérées comme des "variables auxiliaires".

Revenons au système décrit ici. Aux deux axiomes précédents, on ajoute les axiomes et règles suivantes, qui définissent la coopération des preuves:

- Axiome de la communication:

$$\{\text{vrai}\} \text{Pi?x} // \text{Pj!y} \{x=y\}$$

- Règle de la composition parallèle:

$$\frac{\text{les preuves de } \{p_i\} \text{ Pi } \{q_i\}, i=1, n \text{ coopèrent}}{\{p_1 \wedge \dots \wedge p_n \wedge I\} \text{ P1} // \dots // \text{Pn } \{q_1 \wedge \dots \wedge q_n \wedge I\}}$$

où la coopération des preuves et l'invariant global "I" sont définis comme suit.

Qu'une paire d'opérations de communication soient "syntaxiquement" complémentaires, c'est-à-dire telles que dans un processus "Pi" se trouve l'opération "Pj!..." et, dans le processus "Pj", se trouve l'opération "Pi?...". n'implique pas nécessairement que cette communication soit possible à l'exécution, c'est-à-dire que ces opérations soient "sémantiquement" complémentaires.

Les paires d'opérations sémantiquement complémentaires sont déterminées par un invariant global, portant sur des données des processus ou sur un ensemble de "variables auxiliaires", invariant qui doit être vérifié en certains points des processus.

L'ensemble des points où cet invariant doit être vérifié définit, dans chaque processus, des "régions parenthésées" - en un certain sens assimilables à des régions critiques - à l'intérieur desquelles il n'est pas nécessairement vérifié.

Soient "Si" et "Sj" deux régions parenthésées de "Pi" et "Pj". Nous dirons que ces deux régions sont complémentaires si elles contiennent des opérations de communication complémentaires.

Dans ces conditions, les preuves de $\{p_i\} \text{Pi} \{q_i\}$, $i=1, n$ coopèrent si

. les assertions utilisées dans la preuve de $\{p_i\} \text{Pi} \{q_i\}$ n'ont pas de variables libres qui peuvent être modifiées dans P_j , pour $j \neq i$,

. pour toute paire de régions parenthésées complémentaires Si et Sj, on a

$$\{\text{pré}(Si) \wedge \text{pré}(Sj) \wedge I\} Si // Sj \{\text{post}(Si) \wedge \text{post}(Sj) \wedge I\}$$

Le problème du blocage-

La consistance et la complétude de ce système (démontrées dans (Apt_83)) impliquent le résultat très général suivant:

Etant donnée une preuve de $\{p\} \text{P1} // \dots // \text{Pn} \{q\}$,

. si, à partir d'un état initial satisfaisant "p", dans l'exécution de $\text{P1} // \dots // \text{Pn}$ chaque processus "Pi" atteint une opération dont la pré-condition est "préi", l'état courant satisfait

$$\bigwedge_{i=1, n} \text{pré}_i$$

. si Pi est terminé, $\text{post}(Pi)$ est vérifié,

. si aucun processus n'est dans une région parenthésée, l'invariant global "I" est vérifié.

De ce résultat on déduit une caractérisation des systèmes "sans blocage", c'est-à-dire tels que, à partir d'un état initial satisfaisant "p", on ne peut pas atteindre un état de blocage, défini par les propriétés suivantes:

. il n'existe pas de paires de régions parenthésées complémentaires,

. si un processus a atteint une commande répétitive, au moins une des gardes est une opération de communication (émission ou réception), dont le processus interlocuteur n'a pas terminé,

. tous les processus n'ont pas terminé.

Dans ces conditions, on a le théorème suivant:

Théorème-

Etant donnée une preuve de $\{p\} P1//...//Pn \{q\}$, avec un invariant global "I", le programme $P1//...//Pn$ est sans blocage, relativement à "p", si pour tout n-uple d'assertions $\langle p1, \dots, pn \rangle$, associé à une situation de blocage,

$$\neg \left(\bigwedge_{i=1,n} p_i \wedge I \right) \text{ est vrai}$$

où les assertions "pi", associées aux processus "Pi" sont définies de la manière suivante:

. si "Pi" a atteint une opération de communication, en dehors d'une garde, "pi" est la pré-condition de la région parenthésée qui contient cette opération,

. si "Pi" a atteint une garde, dans une commande alternative ou répétitive,

$$p_i = \text{pré}(S) \wedge \bigwedge_{l \neq j} b_l$$

où "S" est la commande (alternative ou répétitive) et "bj" la partie booléenne de la j^{ème} garde, qui est la garde atteinte par "Pi",

. si "Pi" a terminé, $p_i = \text{post}(Pi)$.

Le système de preuve de ADA-

Le même type d'approche est développé dans (Ger_82), (Ger_84), (Ger_84'), etc., pour le langage ADA -du moins pour la partie concernant les tâches parallèles. Les axiomes et règles d'inférence du système de preuves, propres à l'aspect "communication", concernent les instructions d'appel et d'acceptation de procédures "points d'entrée", le "rendez-vous" et la composition parallèle sous l'hypothèse de la coopération des preuves.

La consistance et la complétude de ce système sont déduits d'une traduction de ce fragment de ADA en CSP, traduction qui conserve les preuves.

6-1-4-Logiques de type "temporelles"-

La caractéristique des raisonnements précédents est l'utilisation qu'ils font de logiques "séquentielles", soit en considérant des opérations comme étant indivisibles, soit par la définition de régions "critiques": la récupération du savoir-faire qui leur est associé présente naturellement quelque avantage, mais suppose l'introduction de règles (non-interférence ou coopération des preuves), qui peuvent se révéler coûteuses.

Cette idée est la base même des travaux répertoriés dans la catégorie "logiques temporelles", que nous rappelons maintenant et qui cherchent à exprimer directement dans la logique utilisée l'évolution dans le temps de l'état global d'un programme parallèle.

On notera qu'une des justifications de l'utilisation de ce type de logique est qu'elle permet la démonstration de propriétés non-invariantes d'"accessibilité" des programmes, jusqu'alors peu traitées. En outre, selon leurs auteurs, ce type d'approche se prête d'avantage à une spécification et une vérification "modulaires" des programmes ((Lam_83), (Ha0_83)).

La logique GHL ((Lam 80), (LaS 82))-

GHL est une généralisation de la logique de Hoare permettant de dériver des propriétés invariantes des programmes, en particulier des programmes parallèles. Le principe de base en est le suivant: alors que le système de Hoare caractérise un énoncé "S" par deux prédicats "p" et "q" vérifiés respectivement à l'entrée et à la sortie de "S" -ce qui conduit aux notions d'interférence ou de coopération pour les systèmes de preuve de programmes parallèles construits à partir de ce point de vue- on interprète dans cette nouvelle logique l'écriture $\langle p \rangle S \langle q \rangle$ ainsi: si l'exécution de "S" débute avec le prédicat "p" à vrai, "p" reste à vrai pendant toute la "durée" de cette exécution et "q" devient vrai quand "S" termine.

On est ainsi conduit à définir trois prédicats permettant de caractériser les points de contrôle pour tout

énoncé "S":

- at("S") égal à vrai ssi le contrôle est à l'entrée de "S",
- in("S") égal à vrai ssi le contrôle est quelquepart dans "S", y compris à l'entrée,
- after("S") égal à vrai ssi le contrôle est au point immédiatement derrière "S".

On propose dans (Lam_80) un système de preuve, dont on rappelle ici quelques règles concernant le parallélisme:

- Pour tous énoncés "Si" et "Sj" figurant dans des processus distincts d'un énoncé cobegin ... coend,

"Si"// "Sj"

 $\langle p("Sj") \rangle \text{ Si } \langle p("Si") \rangle$

où $p("Sj")$ dénote l'un quelconque des prédicats at("Sj"), in("Sj"), after("Sj") ou leurs négations.

- Si "S" dénote le programme parallèle

cobegin S1//...//Sn coend

$\langle p>S1\langle p>, \dots, \langle p>Sn\langle p>$

 $\langle p> S \langle p>$

$\text{in}("S") = \frac{[\text{in}("S1") \vee \text{after}("S1")] \wedge \dots \wedge [\text{in}("Sn") \vee \text{after}("Sn")]]}{\wedge \neg [\text{after}("S1") \wedge \dots \wedge \text{after}("Sn")]}$

$\text{at}("S") = \text{at}("S1") \wedge \dots \wedge \text{at}("Sn")$

$\text{after}("S") = \text{after}("S1") \wedge \dots \wedge \text{after}("Sn")$

pour tout $i \neq j$, "Si"// "Sj"

Ce système de preuves permet de démontrer des propriétés de "sûreté" d'un programme parallèle "S" par l'expression d'un prédicat "q" toujours vrai. Pour cela, on

détermine un prédicat "p" tel que:

$\text{at}("S") \Rightarrow p$

$\langle p \rangle S \langle \text{vrai} \rangle$

$p \Rightarrow q$

Ce prédicat a la forme $\bigwedge_{i=1,n} (\text{in}("Si") \Rightarrow pi)$

où les prédicats "pi" sont tels que $\langle pi \rangle Si \langle \text{vrai} \rangle$.

On montre alors qu'il suffit de vérifier:

pour tout $j \neq i$, $\langle pi \wedge pj \rangle Si \langle \text{vrai} \rangle$

En décomposant chaque processus "Si" en opérations élémentaires, cette propriété est équivalente à la condition de non-interférence de (DwG_76).

Logique temporelle des programmes parallèles-

On propose dans (Pnu_81) une application de la logique temporelle des programmes séquentiels (Pnu_77) aux programmes parallèles, constitués de processus partageant un ensemble de données.

Ces programmes sont interprétés dans un modèle du non déterminisme, sous la forme d'un système de transitions étiquetées, dans lequel un état est composé d'un "point de contrôle" et d'un "état mémoire", constitué de l'ensemble des valeurs des données.

Cette interprétation est définie par une sémantique temporelle, qui à tout programme "P" associe une formule "W(P)" d'une logique temporelle, telle que "W(P)" exprime, dans cette logique, les propriétés de toutes les séquences d'exécution possibles associées à "P". Alors pour prouver une propriété "R" de ce programme (correction partielle, non blocage, terminaison, ...), il suffit de démontrer que

$W(P) \Rightarrow R$

est un théorème de la dite logique temporelle.

Définition d'une logique temporelle-

Différentes logiques temporelles sont proposées dans la littérature: logique du temps "linéaire", logique du temps "arborescent", etc. (pour une étude approfondie de ces différents systèmes on pourra lire (Gui_82)). Le modèle du non déterminisme étant retenu, il existe différentes logiques de temps linéaire, selon les opérateurs considérés:

1. Définition de deux opérateurs ((OwL_82), (Lam_83), etc.)

- . $\Box$ signifiant "maintenant et toujours"
- . $\Diamond$ signifiant "maintenant ou plus tard"

Plus précisément, les formules d'une telle logique sont définies de la manière suivante, par composition à l'aide de ces opérateurs, à partir de "propositions élémentaires", c'est-à-dire de fonctions à valeur booléenne, utilisant les opérateurs $\wedge$, $\vee$ et $\neg$:

soit $S = (S_0, S_1, \dots, S_i, \dots)$ une suite temporelle (cf. paragraphe 4-1-), on note $iS = (S_i, S_{i+1}, \dots)$ la sous-suite suffixe de S débutant à l'indice i ,

. une proposition élémentaire "P" est vraie pour "S", et on note

- $S \models P$ ssi $P(S_0) = \text{vrai}$
- . pour toute formule "W"

$$S \models \Box W \text{ ssi pour tout } i \geq 0, iS \models W$$

$$S \models \Diamond W \text{ ssi il existe } i \geq 0, iS \models W$$

2. Définitions d'autres opérateurs:

D'autres opérateurs temporels sont proposés, selon les auteurs (cf. (Pnu_81), (Man_81), (MaP_81'), etc.)

- . l'opérateur $\bigcirc$, défini par $S \models \bigcirc W$ ssi $iS \models W$
- . l'opérateur binaire U , défini par

$$S \models W_1 U W_2 \text{ ssi il existe } i \geq 0 \text{ tel que } iS \models W_2 \text{ et } S_j \models W_1 \text{ pour tout } j \text{ dans } [0, i[$$

. etc.

Les formules valides de ces logiques sont définies par un ensemble d'axiomes et de règles d'inférence, variant selon les auteurs, et que nous ne rappelons pas ici. Rappelons seulement quelques exemples de théorèmes:

$$\begin{aligned} & \Box (W_1 \wedge W_2) = (\Box W_1 \wedge \Box W_2) \\ & \Diamond (W_1 \vee W_2) = (\Diamond W_1 \vee \Diamond W_2) \\ & \Box (W_1 \vee W_2) \Rightarrow (\Box W_1 \vee \Diamond W_2) \\ & \text{etc.} \end{aligned}$$

Propriétés des programmes-

Ces logiques permettent d'exprimer des propriétés d'un programme, en considérant des suites temporelles dont chaque terme définit l'état de ce programme.

- Une propriété invariante a la forme suivante:

$$W_1 \Rightarrow \Box W_2$$

où W_1 et W_2 sont des propositions élémentaires.

Exemple: la correction partielle d'un programme "S" dont la pré-condition est "p" et la post-condition est "q" est définie par la formule

$$\text{at}("S") \wedge p \Rightarrow \Box (\text{after}("S") \Rightarrow q)$$

- Une propriété non-invariante est exprimée par une formule de "fatalité" de la forme suivante:

$$\Box (W_1 \Rightarrow \Diamond W_2)$$

Exemple: correction totale sous les mêmes hypothèses que ci-dessus,

$$\square (\text{at}("S") \wedge p \Rightarrow \Diamond (\text{after}("S") \wedge q))$$

6-2- SERIALISATION D'UN SYSTEME PARALLELE-

Par référence à l'interprétation du parallélisme dans un modèle du non-déterminisme on peut, comme on le fait dans les systèmes de preuves utilisant la logique temporelle rappelés ci-dessus, prouver qu'un programme parallèle a la propriété "R" en démontrant que pour toute suite d'états "s" associée à une exécution de ce programme on a $s \models R$.

D'où l'idée de déterminer, pour un programme donné, l'ensemble des suites d'états, c'est-à-dire l'ensemble des exécutions possibles, sous forme de programmes non-déterministes, et que nous appelons sérialisations d'un programme parallèle.

Cette idée est classique dans l'étude des propriétés d'un système de base de données, notamment pour la vérification de la cohérence des données lors de transactions simultanées : voir en particulier la notion de "consistent schedule" dans (EGL_76) ou de "serializability" dans (Pap_79), (BSW_79), (SiK_79), (BeG_81), etc.

Nous présentons d'abord la notion de "trace d'exécution", puis de "sérialisation".

6-2-1-Trace d'exécution-

Dans tout ce qui suit nous notons "S" un système parallèle défini comme la composition $P1 // P2 // \dots // Pn$, et dont la sémantique est définie au chapitre 4. Les processus "Pi" sont supposés séquentiels: la généralisation à un système "hiérarchisé" (cf. paragraphe 2-2-2-) ne pose pas de problème à condition d'utiliser les définitions et propriétés suivantes de façon "modulaire".

Nous notons $C = \{c1, c2, \dots, cr\}$ l'ensemble des canaux de communication dans "S" (cf. paragraphes 2-1- et 2-2-3-).

Pour illustrer les définitions et propriétés qui suivent, nous considérerons l'exemple des processus parallèles "P1" et "P2" formant un système "P" dans le calcul de factorielle(n) (cf. paragraphe 3-1-), avec les données communiquées de type "égalité":

```
processus P ::
port n1, n2: égalité;

processus P1 ::
  entrée n2: entier ;
  sortie n1: entier ;
  corps
    f1:=1; n1:=1;
    produire(n1) ; consommer(n2) ;
    *f n1<n2 ---> f1:=f1*n1; n1:=n1+1;
    produire(n1) ; consommer(n2)
  ]
fin P1;

processus P2 ::
  entrée n1: entier ;
  sortie n2: entier ;
  entier n;
  corps
    f2:=1; n2:=n;
    produire(n2) ; consommer(n1) ;
    *f n1<n2 ---> f2:=f2*n2; n2:=n2-1;
    produire(n2) ; consommer(n1)
  ]
fin P2;

composition
  P1 // P2

fin P;
```

L'ensemble des canaux de communication dans "P" est

$$C = \{n1, n2\}$$

```
-- On confond dans cet exemple le noms des données
-- communiquées et du port associé
```

-- Nous avons supprimé du texte de ces processus tout ce qui concerne les communications de "f1" et "f2": ces données communiquées sont prises en compte lors de la vérification du système complet défini par la composition P1 // P2 // Q

--

Définition 1-

Dans un processus "Pi" on appelle unité d'exécution une "région" qui communique avec d'autres processus ou bien par une opération de production, ou bien par une opération de consommation.

Exemple-

Le ième pas d'itération du processus "P1" définit deux unités d'exécution:

eli1 : f1:=f1*n1; n1:=n1+1;
produire(n1) ;

eli2 : consommer(n2)

De même, le jème pas d'itération du processus "P2" définit deux unités d'exécution:

e2j1 : f2:=f2*n2; n2:=n2-1;
produire(n2) ;

e2j2 : consommer(n1)

Définition 2-

On appelle unité initiale du système "S", et on note "e0", l'ensemble des définitions des canaux de communication du système.

Exemple-

L'unité initiale du système "P" est définie par

e0 : canal n1
canal n2
tous deux de type "égalité".

Définition 3-

On appelle trace d'exécution d'un processus "Pi", et on note "TPi", l'ensemble des suites possibles d'unités d'exécution, définies par le schéma de contrôle de ce processus.

Cet ensemble est exprimé sous la forme d'une expression régulière, du type "expression de chemin", sur l'alphabet des noms d'unités d'exécution. On note:

- . ";" la trace formée de la séquence de deux unités d'exécution,
- . "(...)" la répétition finie d'une trace d'exécution,
-- le symbole "+" remplaçant "*" signifie que la trace est présente au-moins une fois --
- . "," le choix non-déterministe, mais exclusif dans un ensemble fini de traces d'exécution,
- . "[...]" l'éventualité d'une trace d'exécution.

Remarque-

Toute trace d'exécution commence par "e0".

Exemples-

. La trace d'exécution du processus "P1" est:

TP1 = e0 ; +(eli1 ; eli2)

. De même:

TP2 = e0 ; +(e2j1 ; e2j2)

Pour illustrer un cas d'éventualité, la trace d'exécution d'un processus "Pi" du système "S" présenté au paragraphe 3-5-, pour le calcul du plus court chemin dans un graphe, est:

$\text{TPi} = e0 ; \text{ } \underset{j}{*} (\text{eij1} [; \text{eij2}])$

où, au jème pas d'itération:

eij1 : consommer(m) ;

eij2 : $\underset{(\text{mk} := (i, \text{wik} + d))}{d := m \cdot x_i}$; produire(mk) ; kessucc(i)

6-2-2-Assertions d'une unité d'exécution-

Dans un processus "Pi" chaque unité d'exécution est caractérisée par une pré- et une post-assertion précisant sa situation vis-à-vis de l'ensemble des données communiquées du système. Elles sont définies de la manière suivante:

Définition 4-

On appelle pré-assertion d'une unité d'exécution "ei", $i \neq 0$, le r-uple de variables à valeurs booléennes

$\text{pi} = (\text{préi1}, \text{préi2}, \dots, \text{préir})$

où pour tout j de 1 à r, préij est définie de la manière suivante:

. pour tout j tel que l'opération consommer(x) apparait dans ei, avec $x = x_j$:

préij = pré-cons(tcj)

où tcj est l'objet de type "TC.cj" associé à cj (canal associé à x_j)

. pour les autres valeurs de j:

préij = vrai

Exemples-

La pré-assertion de ei1 ci-dessus est:

pli1 = (vrai , vrai)

La pré-assertion de ei2 est:

pli2 = (vrai , non vide?(A2))

La pré-assertion de e2j1 est:

p2j1 = (vrai , vrai)

La pré-assertion de e2j2 est:

p2j2 = (non vide?(A1) , vrai)

-- On note "A1" et "A2" les ensembles consommables respectifs des données communiquées "n1" et "n2" --

Définition 5-

On appelle post-assertion d'une unité d'exécution "ei", $i \neq 0$, le r-uple de variables à valeurs booléennes

$\text{qi} = (\text{posti1}, \text{posti2}, \dots, \text{postir})$

où pour tout j de 1 à r, postij est défini de la manière suivante:

. pour tout j tel que l'opération consommer(x) apparait dans "ei", avec $x = x_j$:

postij = pré-cons (post-cons(tcj))

. pour tout j tel que l'opération produire(x) apparait dans "ei", avec $x = x_j$:

postij = pré-cons (prod(tcj, VAL.xj))

. pour les autres valeurs de j:

postij = faux

Exemples-

Pour les unités d'exécution ci-dessus on a:

$q_{111} = (\text{non vide?}(A1) , \text{faux})$

$q_{112} = (\text{faux} , \text{vrai})$

$q_{211} = (\text{faux} , \text{non vide?}(A2))$

$q_{212} = (\text{vrai} , \text{faux})$

Définition 6-

Les assertions de l'unité initiale "e0" sont:

$p_0 = (\text{vrai} , \text{vrai}, \dots , \text{vrai})$

$q_0 = (\text{post01} , \text{post02}, \dots , \text{post0r})$

où pour tout j de 1 à r:

$\text{post0j} = \text{pré-cons}(\text{init})$

init créant l'objet "tcj" de type "TC.c.j".

Exemple-

Pour le système P ci-dessus, on a:

$p_0 = (\text{vrai} , \text{vrai})$

$q_0 = (\text{vide?}(A1) , \text{vide?}(A2))$

6-2-3-Sérialisation d'un ensemble d'unités d'exécution-

On considère maintenant l'ensemble des unités d'exécution des processus "Pi", $i=1,n$, formant un système parallèle "S": on définit par induction une sérialisation de cet ensemble.

Définition 7-

On appelle sérialisation, et on note $(e_0;e_1;\dots;e_i)$, une suite ordonnée d'unités d'exécution, dont le premier terme est "e0" et telle que:

- (e0) est une sérialisation,
- pour $i>0$, $(e_0;e_1;\dots;e_i)$ est une sérialisation si et seulement si
 - . $(e_0;\dots;e_{i-1})$ est une sérialisation,
 - . $(e_0;\dots;e_{i-1})$ est inférieure à "ei".

Définition 8-

Pour $i>0$, une sérialisation $(e_0;\dots;e_{i-1})$ est inférieure à une unité d'exécution "ei", et on note $(e_0;\dots;e_{i-1}) < e_i$, si et seulement si:

- . pour tout j de 1 à n, la projection de $e_0;\dots;e_{i-1}$ sur "Pj" satisfait la trace d'exécution "IPj" (au sens de la vérification de l'"adequacy" dans (LaS_78)),
 - le terme projection signifie qu'on retire de la suite $e_0;\dots;e_i$ les unités ne figurant pas dans le processus "Pj" --
- . $\tilde{q}_{i-1} \Rightarrow p_i$, où " $\tilde{q}_{i-1}$ " est la post-condition de la sérialisation $(e_0;\dots;e_{i-1})$.
 - $(b_1,b_2,\dots,b_r) \Rightarrow (b'_1,b'_2,\dots,b'_r)$ ssi pour tout j de 1 à r $b_j \Rightarrow b'_j$, c'est-à-dire $b_j = b'_j \wedge b_j$
 -

Définition 9-

La post-condition " $\tilde{q}_i$ " d'une sérialisation $(e_0;e_1;\dots;e_i)$ est définie comme suit:

- . $\tilde{q}_0 = q_0$
- . pour tout $i>0$, " $\tilde{q}_i$ " est obtenue à partir de la

post-assertion "qi" de "ei" en substituant à toutes ses composantes égales à faux leur valeur dans "qi-1".

Exemples-

Pour le système P ci-dessus, on a :

- $\tilde{q}_0 = (\text{vide?}(A1) , \text{vide?}(A2))$

$p_{111} = (\text{vrai} , \text{vrai})$

d'où on déduit (e_0) < e_{111} , et donc ($e_0; e_{111}$), qui satisfait les traces "TP1" et "TP2", et dont la post-condition est :

$\tilde{q}_{111} = (\text{non vide?}(A1) , \text{vide?}(A2))$

- Démontrons ($e_0; e_{111}; e_{211}$) :

. $p_{211} = (\text{vrai} , \text{vrai})$, d'où $\tilde{q}_{111} \Rightarrow p_{211}$

. $e_0; e_{111}; e_{211}$ satisfait les traces "TP1" et "TP2"

on a donc ($e_0; e_{111}$) < e_{211} , ce qui démontre la sérialisation.

- Démontrons ($e_0; e_{111}; e_{211}; e_{112}; e_{212}$)

. on a $\tilde{q}_{211} = (\text{non vide?}(A1) , \text{non vide?}(A2))$

et $p_{112} = (\text{vrai} , \text{non vide?}(A2))$

d'où $\tilde{q}_{211} \Rightarrow p_{112}$

et d'après "TP1" et "TP2" ($e_0; e_{111}; e_{211}; e_{112}$) dont la post-condition est :

$\tilde{q}_{112} = (\text{non vide?}(A1) , \text{vrai})$

On a $p_{212} = (\text{non vide?}(A1) , \text{vrai})$

d'où $\tilde{q}_{112} \Rightarrow p_{212}$,

et d'après "TP1" et "TP2" :

($e_0; e_{111}; e_{211}; e_{112}; e_{212}$)

dont la post-condition est :

$\tilde{q}_{212} = (\text{vrai} , \text{vrai})$

Définition 10-

Etant donné un système $P1//...//Pn$, on appelle sérialisation de ce système, et on note $\mathcal{S}(P1//...//Pn)$, une sérialisation de toutes les unités d'exécution composant ce système.

6-3- SYSTEME DE PREUVE-

On présente maintenant un système formel qui permet de prouver la correction partielle d'un système parallèle. On montre que ce système est consistant et complet pour le langage d'expression du parallélisme, et sa sémantique, présentée au chapitre 4.

6-3-1-Axiomes et règles d'inférence-

Du fait de l'interprétation dans un modèle non-déterministe, le système formel proposé est une extension des systèmes de preuves classiques des programmes non-déterministes (voir par exemple (Apt_84)).

Axiomes-

. A1 - Axiome de l'opération vide

$\{p\} \text{ vide } \{p\}$

. A2 - Axiome de l'affectation

$\{p[t/x]\} \quad x := t \quad \{p\}$

. A3 - Axiome de la production

$\{p\} \text{ produire}(x) \quad \{p \wedge Q_p\}$

avec $Q_p = (\text{post}_1, \text{post}_2, \dots, \text{post}_r)$

où . "r" est le cardinal de l'ensemble des canaux de communication $\{c1, c2, \dots, cr\}$,
 . pour tout j de 1 à r:

$$\text{post}_j = \begin{cases} \text{pré-cons}(\text{prod}(tc_j, \text{VAL}.x_j)) & \text{si } x=x_j \\ \text{faux} & \text{si } x \neq x_j \end{cases}$$

-- avec les mêmes notations que ci-dessus en 6-2-2-

--

. A4 - Axiome de la consommation

$$\{p[\text{val-cons}(tc)/x] \wedge Pc\} \text{ consommer}(x) \{p \wedge Qc\}$$

avec "tc" est l'objet de type "TC" associé à x,

$$Pc = (\text{pré}_1, \text{pré}_2, \dots, \text{pré}_r)$$

où pour tout j de 1 à r:

$$\text{pré}_j = \begin{cases} \text{pré-cons}(tc_j) & \text{si } x=x_j \\ \text{vrai} & \text{si } x \neq x_j \end{cases}$$

$$Qc = (\text{post}_1, \text{post}_2, \dots, \text{post}_r)$$

où pour tout j de 1 à r:

$$\text{post}_j = \begin{cases} \text{pré-cons}(\text{post-cons}(tc_j)) & \text{si } x=x_j \\ \text{faux} & \text{si } x \neq x_j \end{cases}$$

-- Ces deux derniers axiomes expriment

. d'une part, les pré- et post-conditions de production et de consommation précisées par le type de communication associé aux données communiquées,

. d'autre part, la valeur échangée par cette opération.

Comparés aux approches rappelées au paragraphe 6-1-, ces axiomes sont un mode d'expression de la "complémentarité sémantique" des opérations de communication et aussi de l'axiome de la communication du paragraphe 6-1-3-. En effet, qu'il s'agisse de l'écriture d'un invariant global, de la définition de variables auxiliaires ou de traces de communications, l'idée est toujours d'étudier une certaine "coopération" des preuves lors des rendez-vous sémantiquement possibles. Ici, les r-uplets

définissant "Qp", "Pc" et "Qc" peuvent être considérés comme des "variables auxiliaires" exprimant l'état des données communiquées: alors, à la notion de paires d'opérations sémantiquement complémentaires, nous substituons naturellement la notion de paire de valeurs sémantiquement liées. Notre but n'est pas de dire que telle opération d'émission peut être associée à telle opération de réception; il est de préciser, pour chaque valeur reçue d'une donnée, à quelle valeur émise elle correspond. Ceci est bien l'objet des pré- et post-conditions des axiomes A3 et A4.

--

Règles d'inférence-

. R1 - Règle de la composition séquentielle

$$\frac{\{p\} S1 \{r\} , \{r\} S2 \{q\}}{\{p\} S1;S2 \{q\}}$$

. R2 - Règle du choix non-déterministe

$$\frac{\{p \wedge b_i\} Si \{q\} , i=1, n}{\{p\} [b1 \rightarrow S1 \square \dots \square b_n \rightarrow S_n] \{q\}}$$

. R3 - Règle de l'itération

$$\frac{\{p \wedge C\} S \{p\}}{\{p\} * [C \rightarrow S] \{p \wedge \neg C\}}$$

. R4 - Règle de l'implication

$$\frac{p \Rightarrow p1 , \{p1\} S \{q1\} , q1 \Rightarrow q}{\{p\} S \{q\}}$$

. R5 - Règle de la composition parallèle

$$\frac{\{p\} \mathcal{S}(P1//\dots//Pn) \{q\} , \text{ pour toute sérialisation } \mathcal{S}}{\{p\} P1//\dots//Pn \{q\}}$$

-- Cette dernière règle établit le lien entre l'ensemble des sérialisations, ou suites possibles d'exécutions d'unités de P1, P2, ..., Pn et le

système parallèle.

La vérification de cette propriété est naturellement coûteuse, comme toutes les propositions rappelées ci-dessus: c'est le prix à payer pour s'être ramené à un modèle du non-déterminisme.

On remarquera cependant, dans les exemples cités ci-après, qu'on aboutit à une démarche très modulaire permettant de démontrer des propriétés "structurelles" des processus et, par conséquent, de réduire sensiblement le coût apparent d'une telle méthode.

--

6-3-2-Propriétés d'un système parallèle-

Dans ce système formel on peut démontrer des propriétés invariantes, ou non, d'un programme parallèle.

Correction partielle-

Les axiomes et règles d'inférence présentées ci-dessus permettent la preuve de correction partielle des programmes. La règle R5 précise en effet que si un programme parallèle $P_1 // \dots // P_n$, dont l'exécution commence en vérifiant le prédicat "p", termine son exécution, alors le prédicat "q" est valide. Deux exemples sont traités au paragraphe 6-4-ci-dessous.

Blocage-

Du système précédent on déduit trivialement la propriété suivante: un programme parallèle est sans blocage si et seulement si pour toute sérialisation "S" d'un ensemble d'unités d'exécution, dont la post-condition est " $\tilde{q}_i$ ", ou bien "S" est la sérialisation du programme, ou bien il existe une unité d'exécution "e_j" telle que "S" est inférieure à "e_j", c'est-à-dire $\tilde{q}_i \Rightarrow p_j$.

Correction totale-

La preuve de correction totale $\{p\} S \{q\}$ signifie

que toute exécution du programme "S" satisfaisant initialement "p" termine et satisfait "q". Cette preuve suppose

- que toute unité d'exécution termine: pour cela on peut modifier de manière classique (voir (Har_79) ou (Apt_83)) les règles R2 et R3, qui deviennent

$$p \Rightarrow \bigvee_{i=1,n} b_i, \{p \wedge b_i\} \text{ si } \{q\}, i=1,n$$

$$R'2. \text{-----}$$

$$\{p\} [b_1 \rightarrow S_1 0 \dots 0 b_n \rightarrow S_n] \{q\}$$

$$R'3. \text{-----}$$

$$p(m) \wedge m > 0 \Rightarrow C, p(0) \Rightarrow \neg C, \{p(m) \wedge m > 0\} S \{p(m-1)\}$$

$$\{ \exists m p(m) \} * [C \rightarrow S] \{ p(0) \}$$

- et que le programme "S" est sans blocage.

6-3-3-Consistance et complétude du système de preuve-

Nous montrons que le système formel présenté ci-dessus est consistant et complet vis-à-vis de la sémantique des programmes parallèles (notée "Sem" dans la suite), sous forme de systèmes non déterministes, présentée au paragraphe 4-1-.

Consistance-

Intuitivement, la consistance (the "soundness", voir entre autres (Owi_76), (Apt_83), (Apt_84)) d'un système formel vis-à-vis d'une interprétation donnée, signifie que toute formule démontrée vraie en utilisant les axiomes et les règles d'inférence du système formel, est vraie dans l'univers de l'interprétation.

Ici, les formules sont des expressions de la forme

$$\{p\} S \{q\}.$$

Une telle formule est vraie dans l'interprétation "Sem" rappelée plus haut si pour toute suite d'états $s_0, s_1, \dots, s_m$, associée à toute exécution possible de "S", telle que " s_0 " satisfait "p", " s_m " satisfait "q".

On a la propriété suivante:

Théorème:

Si une formule $\{p\} S \{q\}$ peut être prouvée en utilisant les axiomes A1 à A4 et les règles R1 à R5, alors $\{p\} S \{q\}$ est vraie pour l'interprétation "Sem".

Schéma de démonstration:

On démontre cette propriété en montrant que chaque axiome est valide, et que chaque règle est consistante pour l'interprétation "Sem".

On se limite ici aux axiomes A3 et A4 et à la règle R5, représentatifs du domaine parallèle considéré. Pour le reste, nous renvoyons aux résultats classiques pour les programmes non déterministes.

Axiome A3-

La formule $\{p\} \text{produire}(x) \{p \wedge q\}$ est vraie dans "Sem". En effet, si "s" est l'état du système avant l'activation de cette opération, "s" est caractérisé par un objet "tc" et satisfait "p". D'après la transition définie en 4-1-3-, l'état résultant "s'" est caractérisé par l'objet $\text{prod}(tc, \text{VAL}.x)$ et satisfait donc $\{p \wedge q\}$.

Axiome A4-

De même, la formule

$$\{p[\text{val-cons}(tc)/x] \wedge p\} \text{consommer}(x) \{p \wedge q\}$$

est vraie dans "Sem".

Règle R5-

Soit $S = P1//...//Pn$. Considérons une exécution

quelconque de "S", caractérisée par une suite d'états $s0, s1, \dots, sm$ telle que "s0" satisfait "p". Cette exécution est définie par une sérialisation de "S", dont les assertions intermédiaires définissent les états "si", $i=0, m$.

Si "q" est la post-condition de cette sérialisation, l'application de la règle R5 conduit à $q = \tilde{q}_m$. Par définition "sm" satisfait " $\tilde{q}_m$ " et, par conséquent il satisfait "q". La formule $\{p\} S \{q\}$ est donc vraie dans l'interprétation "Sem", ce qui achève la démonstration.

Complétude-

Intuitivement, la complétude (the "completeness") d'un système formel vis à vis d'une interprétation signifie que toute formule vraie dans cette interprétation peut être prouvée en utilisant les axiomes et les règles d'inférence du dit système.

Ici, notre propos est de montrer que le système de preuves précédent est complet vis-à-vis de l'interprétation "Sem". Comme précédemment, nous nous limitons aux expressions $\{p\} S \{q\}$, où "S" est un système parallèle $P1//...//Pn$, en renvoyant aux résultats classiques pour les programmes non déterministes.

On a la propriété suivante:

Théorème:

Si la formule $\{p\} S \{q\}$ est vraie dans l'interprétation "Sem", alors $\{p\} S \{q\}$ peut être prouvée en utilisant les axiomes A1 à A4 et les règles R1 à R5.

Schéma de démonstration:

Considérons donc une expression $\{p\} S \{q\}$ vraie dans l'interprétation "Sem".

On définit des pré- et post-assertions "pi" et "qi" de toute unité d'exécution "ei" de "S": ces assertions sont composées des assertions "pi" et "qi" définies ci-dessus en 6-2-2-,

caractérisant "l'état des communications", et d'assertions "classiques" caractérisant l'état des processus séquentiels.

En utilisant les axiomes et les règles R1 à R4, on peut prouver toutes les formules $\{p_i\}$ et $\{q_i\}$. Ce point n'est développé d'avantage ici.

En utilisant la règle R5, on prouve alors $\{p\} S \{q\}$. En effet, pour toute sérialisation, caractérisée par les pré- et post-conditions "p0" et "qm" (définies comme indiqué ci-dessus), on a
 . $p \Rightarrow p0$
 . $qm \Rightarrow q$,
 en effet, si "qm" est vérifié après la sérialisation de "S", "p" est vérifié avant cette sérialisation; $\{p\} S \{q\}$ étant supposée vraie dans "Sem", on déduit que "q" est vérifié après "S", et par conséquent $qm \Rightarrow q$, ce qui termine la démonstration.

6-4- EXEMPLES ET PROPRIETES-

Nous illustrons le système de preuves ci-dessus par l'étude de deux exemples de programmes présentés au chapitre 3, représentatifs de deux situations intéressantes et pour lesquelles nous démontrons des propriétés générales.

6-4-1-Calcul de factorielle-

Revenons sur l'étude commencée au paragraphe 6-2-1- ci-dessus, en considérant le système P composé de P1//P2;

Rappelons les notations:

```
pour tout i>0
  e1i1 : f1:=f1*n1; n1:=n1+1;
        produire(n1);
  e1i2 : consommer(n2);

pour tout j>0
  e2j1 : f2:=f2*n2; n2:=n2-1;
        produire(n2);
  e2j2 : consommer(n1);
```

```
-- En fait, pour i=1, e1i1 : f1:=1; n1:=1;
                        produire(n1);
et, pour j=1, e2j1 : f2:=1; n2:=n;
                        produire(n2);
--
```

Communications en "égalité"-

Nous avons montré que $(e0; e1i1; e2j1; e1i2; e2j2)$ est une sérialisation. Généralisons ce résultat pour déduire la propriété structurale suivante:

Propriété:

Toute sérialisation de "P" est de la forme

$SeP = (e0; \uparrow (ekil; ekil; eli2; eli2))$

avec $k, \bar{k} \in [1,2]$ et $k+\bar{k}=3$
 $l, \bar{l} \in [1,2]$ et $l+\bar{l}=3$

où la notation " $\uparrow(\dots)$ " a la signification indiquée en 6-2-1-.

La démonstration de cette propriété est déduite des lemmes suivants:

Lemme 1-

$(e0; \uparrow (eli1; e2i1; eli2; e2i2))$ est une sérialisation de "P".

Démonstration-

Par induction sur "i": la propriété est démontrée pour $i=1$.

De $\widetilde{q2i2} = (\text{vrai}, \text{vrai})$

on déduit que cette assertion est un invariant de l'itération. On a donc

$\widetilde{q2i2} = (\text{vrai}, \text{vrai})$

pour tout $i>0$, ce qui termine la démonstration.

Lemme 2-

Toute sérialisation de la forme "SeP" est une sérialisation de "P".

Démonstration-

Pour tout $k, \bar{k}, l$ et $\bar{l}$ vérifiant les hypothèses indiquées, la post-assertion invariante est identique:

$$\tilde{qli2} = (\text{vrai} , \text{vrai})$$

Lemme 3-

Toute autre séquence n'est pas une sérialisation de "P".

Démonstration-

On généralise facilement le contre-exemple suivant:

($e0; e111$) < $e112$ est faux.

En effet, $\tilde{qli1} = (\text{nonvide?}(A1) , \text{vide?}(A2))$
 $\neq \text{pli2} = (\text{vrai} , \text{nonvide?}(A2))$

Remarque-

La conséquence "observable" de cette propriété est une exécution "synchrone" de cet algorithme.

Preuve du programme "P"-

D'après la règle R5, la post-condition de "P" est identique à la post-condition des sérialisations de la forme "SeP". Toute sérialisation de la forme "SeP" est équivalente à l'algorithme séquentiel suivant:

```
f1:=1; f2:=1; n1:=1; n2:=n;
* [ n1<n2 --> f1:=f1*n1; f2:=f2*n2;
    n1:=n1+1; n2:=n2-1
  ]
```

dont la post-condition est:

$$\{ f1=(n1-1)! \wedge f2=n!/n2! \wedge n2 \leq n1 \leq n2+1 \}$$

Preuve du système complet P1//P2//Q-

En ajoutant "f1" et "f2" à l'ensemble de données communiquées, l'ensemble des sérialisations de "S" est équivalent à l'ensemble des sérialisations de la forme SeP;e, où "e" est le "corps" de "Q":

```
e : consommer(f1) ; consommer(f2) ;
    consommer(n1) ; consommer(n2) ;
    [ n1=n2 --> f:=f1*f2*n1
      0 n1>n2 --> f:=f1*f2
    ]
```

On en déduit que la post-condition de "S" est $\{ f=n! \}$

Communications en "aléatoire"-

Dans la seconde version de ce système, on a les pré- et post-conditions suivantes pour les unités d'exécution des processus "P1" et "P2":

```
p0 = ( vrai , vrai )
q0 = ( vide?(A1) , vide?(A2) )

pli1 = ( vrai , vrai )
qli1 = ( nonvide?(A1) , faux )

pli2 = ( vrai , nonvide?(A2) )
qli2 = ( faux , nonvide?(A2) )

p2i1 = ( vrai , vrai )
q2i1 = ( faux , nonvide?(A2) )

p2i2 = ( nonvide?(A1) , vrai )
q2i2 = ( nonvide?(A1) , faux )
```

On en déduit la propriété suivante:

Propriété-

Toute sérialisation de "P" est de la forme

```
SeP = ( e0; ek11; ek11; e112; e112;
        ( ( eli1; eli2 ),
          ( e2i1; e2i2 ),
          ( ek11; ek11; eki2; eki2 ),
          ( ekil; ekil; eki2; eki2 )
        ) )
```

avec $k, \bar{k} \in [1, 2]$ et $k + \bar{k} = 3$
 $l, \bar{l} \in [1, 2]$ et $l + \bar{l} = 3$

Démonstration-

Les pré-assertions étant toujours vérifiées pour $i, j > 1$, il suffit de montrer que toute sérialisation de la forme "SeP" est telle que sa projection sur "P1" et "P2" satisfait les traces "TP1" et "TP2", ce qui est trivial. D'autre part, le début de ces sérialisations exprime la nécessité d'une première production; la suite exprime que toute sérialisation convient.

Remarque-

La conséquence "observable" de cette propriété est une exécution "asynchrone" de cet algorithme.

Preuve du programme-

Toute sérialisation de "P" est équivalente à l'algorithme séquentiel suivant:

```

f1:=1; f2:=1; n1:=1; n2:=n;
* [ n1 < n2 --> [ vrai --> f1:=f1*n1; n1:=n1+1
                | vrai --> f2:=f2*n2; n2:=n2-1
                | vrai --> f1:=f1*n1; f2:=f2*n2;
                  n1:=n1+1; n2:=n2-1
                ]
      ]

```

dont la post-condition est, comme précédemment:

$$\{ f1 = (n1-1)! \wedge f2 = n! / n2! \wedge n2 = \langle n1 = \langle n2 + 1 \rangle \}$$

qui est donc la post-condition de "P".

On déduit également que la post-condition du système "S" est $\{ f = n! \}$

Remarque-

On observe sur cet exemple que l'ensemble des sérialisations correspondant au premier système (version

"synchrone") est inclus dans l'ensemble des sérialisations correspondant au second (version "asynchrone"). Ceci suggère le résultat général suivant:

Théorème-

. Soit "S1" un système dans lequel on associe à une donnée communiquée le type de communication "TC1",
 . Soit "TC2" un type de communication tel que $TC2 \text{ as} < TC1$ et tel que, pour toute suite d'opérations de consommation dans "TC2", il existe un chemin dans "TC1" admettant les mêmes opérations, dans le même ordre (cette hypothèse "raisonnable" est en particulier vérifiée si $TC2 \text{ op} < TC1$), . Soit "S2" le système issu de "S1" en substituant "TC2" à "TC1",

On a $\{p\} S1 \{q\} \Rightarrow \{p\} S2 \{q\}$

Démonstration-

Par définition de la relation d'ordre "as<", toute sérialisation de "S2" est une sérialisation de "S1". Si "q" est la post-condition de toute sérialisation de "S1", "q" est donc la post-condition de toute sérialisation de "S2". De plus, les valeurs traitées dans cette sérialisation de "S2" le sont, dans le même ordre, dans une sérialisation de "S1" dont "q" est également la post-condition. D'où le résultat.

Exemples-

Ce type de propriété peut être appliquée dans des situations du type:

rendez-vous --> tampon --> égalité --> aléatoire
 (à une place) rafraichie ↗

caractéristiques de "représentation" d'un type par un autre.

Remarque-

Il s'agit bien ici de preuve de correction partielle,

puisque "S1" peut être sans blocage et "S2" ne pas l'être.

6-4-2-Calcul des coefficients du binôme-

Nous reprenons l'exemple développé au paragraphe 3-3-, qui est représentatif des classes de systèmes dans lesquels apparaissent des communications "optionnelles". Nous commençons par le résultat général suivant:

Théorème-

. Soient "TC1" et "TC2" deux types de communication tels que $TC2 \leq TC1$, et " ω " est l'indice des arcs du graphe de "TC1" qui ne sont pas dans le graphe de "TC2",
 . Soit S composé de P1/P2 un système dans lequel on associe à une donnée communiquée "x" de "P1" vers "P2" un objet "tc" de type "TC",
 . On a $\{p\} S \{q\}$ si et seulement si

- (1) Pour toute unité d'exécution " e_{2i} " de "P2" dans laquelle apparaît consommer(x), q_{2i} est indépendante de la consommation, ou non, de la valeur " ω ",
- (2) Pour toute sérialisation $\mathcal{S}$ de "S" telle que "P2" consomme toujours la valeur " ω ", on a $\{p\} \mathcal{S} \{q\}$.

Démonstration-

- Condition nécessaire: on a $\{p\} S \{q\}$ si pour toute sérialisation $\mathcal{S}$, $\{p\} \mathcal{S} \{q\}$; en particulier pour toute sérialisation vérifiant (2).

(1) se montre par l'absurde: si (1) n'est pas vérifié, soit q_{2i} (resp. q'_{2i}) la post-condition de e_{2i} lorsque la consommation a pour résultat " ω " (resp. une valeur différente de " ω "). $q_{2i} \neq q'_{2i}$ doit être vérifiée dans la preuve de $\{p\} \mathcal{S} \{q\}$, et q'_{2i} vérifie (1).

- Condition suffisante: considérons une sérialisation $\mathcal{S}$ de "S" qui ne vérifie pas (2)

$\mathcal{S} = (e_0; \dots; e_{1j}; \dots; e_{2i}; \dots)$
 avec " e_{2i} " telle que consommer(x) délivre une valeur différente de " ω ", produite par " e_{1j} ".
 Soit $\mathcal{S}' = (e_0; \dots; e_{2i}; e_{1j}; \dots)$. $\mathcal{S}'$ vérifie la condition (2) et, d'après (1) elle est telle que sa post-condition est

identique à celle de $\mathcal{S}$, qui est donc égale à " q ", ce qui, d'après la règle R5, termine la démonstration.

Conséquence-

Pour ce type de système on pourra donc procéder en deux étapes:

- 1- vérifier que la post-assertion d'une unité d'exécution dans laquelle on consomme une valeur de la donnée communiquée est indépendante du résultat obtenu (" ω " ou pas " ω "),
- 2- déterminer l'ensemble des sérialisations correspondant au cas où le processus consommateur consomme toujours la valeur " ω ".

Application à l'exemple-

Première étape:

La suite de valeurs, différentes de "max_ent", consommées par le processus "P2" est extraite de la suivante:

$$s = \underbrace{B0,0; B1,1; \dots; Bp,p}_{D0} \underbrace{B1,0; B2,1; \dots; Bp,p-1; \dots; Bp,0}_{D1} \underbrace{\dots}_{Dp}$$

Considérons alors une unité d'exécution du processus "P2" correspondant à une valeur quelconque de " n " et de " m ":

. si consommer(Lm) a pour résultat "max_ent":

$$\dots; Bn,0; Bn+1,1; \dots; Bn+m-1,m-1; \text{"max_ent"}; \dots$$

Dn

"P2" calcule $Dn = Bn+m,m$

. si consommer(Lm) n'a pas pour résultat "max_ent", $Dn = Lm$, qui, d'après "s" est la valeur de " $Bn+m,m$ ".

Cette unité d'exécution vérifie donc la propriété (1). Par symétrie, on a de même que toute unité d'exécution du processus "P1" vérifie (1).

Deuxième étape:

L'ensemble des sérialisations correspondant au cas où "P1" consomme toujours "max_ent" est équivalent au programme séquentiel suivant:

```
m:=0;
* [ m<p -->   n:=m;
  * [ n<p -->   -- calculer Bn,m --
                n:=n+1
  ];
  m:=m+1
]
```

L'ensemble des sérialisations correspondant aux cas où "P2" consomme toujours "max_ent" est équivalent au programme séquentiel suivant:

```
n:=0;
* [ n<p -->   m:=0
  * [ m<p-n --> -- calculer Bn+m,m --
                m:=m+1
  ];
  n:=n+1
]
```

Ces deux programmes calculent les coefficients du binôme d'ordre "p". On peut donc conclure que le système "S" proposé au paragraphe 3-3- est correct.

CHAPITRE 7 - LA COMMUNICATION DANS LA CONCEPTION DE PROGRAMMES

Dans ce mémoire nous avons proposé un outil -la notion de relation de communication-

- . exprimé par un type de communication,
- . dont on donne la sémantique,
- . dont on déduit quelques propriétés des programmes,
- . auquel on associe une méthode de preuve.

Dans ce chapitre nous revenons à l'objectif initial de notre travail: que cet outil induise un mode de raisonnement pour la construction de programmes parallèles.

. Les techniques d'"asynchronisation" vues au chapitre 5, . Le système de preuves développé au chapitre 6, sont des éléments d'aide à cette construction. Nous montrons maintenant en quoi l'introduction de la notion de communication est un intermédiaire qui permet, à différents niveaux, une démarche méthodique et modulaire. Nous illustrons plus particulièrement les points suivants:

- . Différents choix de types de communication, comme outil de modification de programmes,

- . Implantations de types de communication, comme outil de construction de programmes -y compris de programmes séquentiels,

- . Définition d'un type de communication, comme outil de spécification d'un problème.

Auparavant, nous rappelons quelques propositions en matière de spécification et de construction de programmes parallèles.

7-1- SPECIFICATION ET CONSTRUCTION DE PROGRAMMES PARALLELES-

Dans la littérature il y a souvent ambiguïté sur le terme "spécification" utilisé pour dire "conception dans un langage de très haut niveau". En fait la spécification concerne le problème traité en définissant ce qu'un programme doit calculer, le service que doit satisfaire un système, les propriétés qu'ils doivent vérifier, etc. La conception -ou construction, d'un système ou d'un programme définit comment ce système ou ce programme calcule les résultats spécifiés, rend les services attendus ou satisfait les propriétés voulues.

Plusieurs difficultés apparaissent alors, accrues dans le domaine du parallélisme:

- . Quels outils et quelle méthodologie pour la spécification?
- . Synthèse ou dérivation méthodique des programmes à partir d'une spécification?
- . Outils de programmation?

Nous rappelons quelques propositions dans ce domaine.

7-1-1-Spécification des propriétés d'un système-

On propose dans (ChY 83) une méthode de spécification basée sur la notion d'événement, considéré comme une transition atomique instantanée.

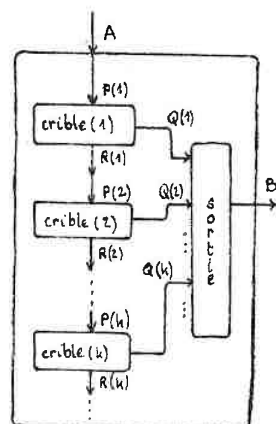
On définit deux relations d'ordre partiel entre événements:

. la "précédence", qui exprime une relation temporelle entre deux événements: $e1 \rightarrow e2$ ssi "e1" arrive avant "e2",

. la "causalité", qui exprime une implication logique entre deux événements: $e1 \rightsquigarrow e2$ ssi le fait que "e1" arrive implique que "e2" arrivera dans le futur.

La démarche présentée est descendante: à chaque étape, on prouve informellement que l'ensemble des spécifications satisfait la spécification de l'étape précédente.

Exemple: Spécification d'un générateur des nombres premiers par le crible d'Eratosthènes.



Première étape-

entrée A: suite des nombres entiers ≥ 2 dans l'ordre croissant

sortie B: suite des nombres premiers

$\forall a \in A \quad \neg (\exists a' \in A, a' \rightarrow a \wedge a'.mess \text{ div } a.mess)$
 $\Leftrightarrow \exists b \in B, a \rightsquigarrow b$

Deuxième étape-

On considère le système constitué d'un ensemble infini de modules "crible" et d'un module "sortie". Pour tout k, "crible(k)" est spécifié par:

. le premier message "p" reçu est envoyé à "sortie":
 $\forall p \in P(k) \text{ ord}(p)=1 \Leftrightarrow \exists q \in Q(k), p \rightsquigarrow q$

. tout message reçu, non multiple de "p" est envoyé au crible suivant:

$\forall p1, p2 \in P(k) \text{ ord}(p1)=1 \wedge \text{ord}(p2)>1 \wedge \neg (p1.mess \text{ div } p2.mess)$
 $\Leftrightarrow \exists r \in R(k), p2 \rightsquigarrow r$

. les messages sont reçus dans l'ordre de leur émission:
 $\forall p1, p2 \in P(k), q \in Q(k), r \in R(k)$
 $p1 \rightsquigarrow q \wedge p2 \rightsquigarrow r \Rightarrow (p1 \rightarrow p2 \wedge q \rightarrow r)$
 $\vee (p2 \rightarrow p1 \wedge r \rightarrow q)$

Le module "sortie" est un multiplexeur spécifié par:

- . pas de perte de message:
 $\forall k, \forall q \in Q(k) \exists b \in B \quad q \rightsquigarrow b$
- . pas de génération de message:
 $\forall b \in B \exists k, \exists q \in Q(k) \quad q \rightsquigarrow b$
- . conservation de l'ordre des messages:
 $\forall k1, k2, \forall a1, b1 \in Q(k1), \forall a2, b2 \in Q(k2)$
 $(a1 \rightsquigarrow a2 \wedge b1 \rightsquigarrow b2) \Rightarrow (a1 \rightarrow b1 \wedge a2 \rightarrow b2)$
 $\vee (b1 \rightarrow a1 \wedge b2 \rightarrow a2)$
 $\vee (a1=b1 \wedge a2=b2)$
- . identité des messages:
 $\forall k, \forall q \in Q(k), \forall b \in B \quad q \rightsquigarrow b \Rightarrow b.mess=q.mess$

Remarques-

1. Ce type de spécification n'est pas éloigné de l'approche "objet" que nous proposons, dans la mesure où on établit une relation entre l'ensemble des messages émis et l'ensemble des messages reçus: l'ensemble des propriétés ci-dessus serait exprimé par le type de communication "égalité".

2. Les relations entre événements sont des relations "temporelles" qu'on retrouve dans les méthodes suivantes, basées sur la logique temporelle.

Diverses propositions existent, portant sur l'état global du système ou sur les "histoires" de ce système: voir par exemple (Lam_83'), (Ha0_83) ou (RaK_83). Nous rappelons ici la méthode proposée dans (Lam_83').

On considère que l'état d'un système définit tous les comportements possibles futurs de ce système: un état est donc composé de l'ensemble des valeurs des données, des points de contrôle atteints dans chaque processus, des valeurs des paramètres dans la pile, du contenu des canaux de communication, etc.

On définit sur ces états un ensemble de fonctions, dites

"fonctions d'état": valeur des variables, points de contrôle (définis au paragraphe 6-1-3-: at(S), in(S), after(S)), etc.

La spécification d'un système a alors la forme suivante:

- . définition de l'ensemble des fonctions d'état,
- . conditions initiales,
- . propriétés du système: sûreté et/ou accessibilité.

Exemple- Spécification d'une file "fifo" de taille "m", accessible à deux processus "PUT" et "GET":

- . fonctions d'état:
 queue: suite de elt
 parg: elt U {ω}
 gval: elt U {ω}
- . conditions initiales:
 queue = ∅
- . propriétés de sûreté:
 1. at(PUT) ==> parg = PUT.par
 after(PUT) ==> parg = ω
 2. at(GET) ==> gval = ω
 after(GET) ==> gval = GET.par
 3. changements d'état de queue:
 . parg quand in(PUT)
 . gval quand in(GET)
 in(PUT) ∧ parg ≠ ω --> parg' = ω ∧ queue' = queue * parg
 in(GET) ∧ gval = ω ∧ |queue| > 0 -->
 gval' ≠ ω ∧ queue = gval * queue'
- . propriétés de vivacité:
 4. □(in(PUT) ∧ |queue| < m ==> ◇ after(PUT))
 5. □(in(GET) ∧ |queue| > 0 ==> ◇ after(GET))

On propose ensuite une implémentation dont on vérifie qu'elle satisfait cette spécification.

7-1-2-Synthèse de processus communicants-

On propose dans (Wol_82) et (MaW_84) une méthode de synthèse de processus communicants de type CSP, à nombre fini d'états, à partir de spécifications utilisant la logique temporelle.

Nous rappelons cette méthode à l'aide de l'exemple de l'expression de l'exclusion mutuelle de deux processus "P1" et "P2", qui communiquent avec un "synchroniseur", "S",

gestionnaire d'une ressource commune: les processus "Pi", i=1,2, envoient des signaux S!begini (entrée en section critique), et S!endi (sortie de la section critique). Le synchroniseur doit garantir que "P1" et "P2" ne sont pas simultanément dans leur section critique.

Spécification:

. Vi=1,2, processus "Pi":

```
S!begini
-- le premier signal émis est l'entrée en section
   critique --
□( S!begini ==> ○ S!endi )
□( S!endi ==> ○ S!begini )
-- les signaux d'entrée et de sortie de section
   critique sont consécutifs --
```

. synchroniseur "S":

```
□( P1?begin1 ==> ((¬P2?begin2) U (P1?end1)))
□( P2?begin2 ==> ((¬P1?begin1) U (P2?end2)))
-- les opérations ○ et U ont le sens défini en
   6-1-3- --
□( ◇ P1?begin1 ∨ ◇ P1?end1 )
□( ◇ P2?begin2 ∨ ◇ P2?end2 )
-- un processus n'est pas bloqué dans sa section
   critique --
```

A partir de cette spécification, la synthèse procède en trois étapes:

- Transformation des spécifications de chaque processus en une spécification globale du système,
- Application d'un algorithme qui produit un graphe d'états finis (si la spécification a une solution),
- Dérivation des programmes réalisant les processus.

Etape 1-

Il s'agit essentiellement de tenir compte de la

sémantique des communications pour les paires d'opérations complémentaires, et de spécifier les entrelacements (ou sérialisations) possibles des opérations de chaque processus. Pour l'exemple considéré on obtient:

. Soit $\text{in}(P1) = \text{begin1} \vee \text{end1}$
 $\text{in}(P2) = \text{begin2} \vee \text{end2}$

-- La proposition begini (resp. endi) dénote le rendez-vous
 $S! \text{begini} // P_i? \text{begini}$
 (resp. $S! \text{endi} // P_i? \text{endi}$)

. Spécification de "Pi", $i=1,2$:

$\neg \text{in}(P_i) \vee \text{begini}$
 $\square(\text{begini} \Rightarrow \bigcirc(\neg \text{in}(P_i) \vee \text{endi}))$
 $\square(\text{endi} \Rightarrow \bigcirc(\neg \text{in}(P_i) \vee \text{begini}))$

. Spécification de "S":

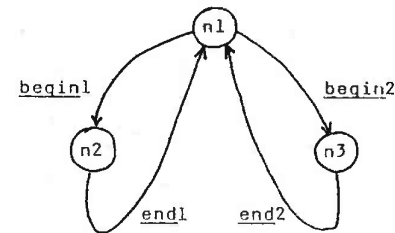
$\square(\text{begin1} \Rightarrow \neg \text{begin2} \vee \text{end1})$
 $\square(\text{begin2} \Rightarrow \neg \text{begin1} \vee \text{end2})$

. Non blocage:

$\square \diamond \text{in}(P1)$
 $\square \diamond \text{in}(P2)$

Etape 2-

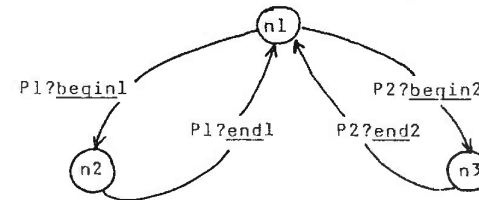
L'algorithme transforme itérativement l'ensemble des formules de la spécification jusqu'à des propositions, ou des formules ne contenant que l'opérateur temporel " $\bigcirc$ ": initialement, l'ensemble des formules de la spécification définit le noeud d'un graphe. A chaque étape de l'algorithme, l'ensemble des formules qui ne sont pas du type indiqué, obtenues à partir d'une proposition et des formules définissant un noeud du graphe, définit un nouveau noeud (ou un noeud existant déjà). L'arc reliant ces deux noeuds est étiqueté par la proposition utilisée. Sur l'exemple, on obtient le graphe suivant:



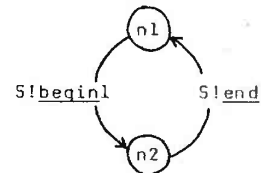
Etape 3-

De ce graphe, on dérive un graphe associé à chaque processus: les arcs sont étiquetés par les opérations propres à chaque processus. On obtient:

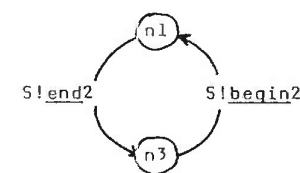
Pour "S":



Pour "P1":



Pour "P2":



On obtient finalement le programme CSP suivant, en représentant chaque noeud des graphes par un entier 1, 2 ou 3:

S:: [N:=1;
 * [N=1; P1?begin1 --> N:=2

```

    0 N=1; P2?begin2 --> N:=3
    0 N=2; P1?end1 --> N:=1
    0 N=3; P2?end2 --> N:=1
  ]
]

P1:=[N:=1;
  * [ N=1; S!begin1 --> N:=2
    0 N=2; S!end1 --> N:=1
  ]
]

P2:=[N:=1;
  * [ N=1; S!begin2 --> N:=2
    0 N=2; S!end2 --> N:=1
  ]
]

```

On propose dans (ClE_82) une méthode analogue pour une logique temporelle arborescente et des systèmes parallèles à données partagées.

7-1-3-Construction de programmes-

Un certain nombre de "méthodes", ou de représentations d'une démarche de conception de programmes parallèles, ou distribués, dans un modèle formel, ou de transformations de programmes, sont proposées dans la littérature.

Du point de vue conception des programmes, on peut repérer trois types de "méthodes":

- Les méthodes "descendantes", où, à chaque niveau, on affine des modules du niveau antérieur. La difficulté de ce type d'approche est la conception de niveaux "fermés" du point de vue des communications de données:

Citons une représentation de type réseaux de Petri (YaC_83), où ces modules sont considérés comme des transitions non instantanées, et une représentation, sous forme de modules avec ports de communication, d'une démarche basée sur l'énoncé de propriétés invariantes des systèmes ((KrC_78), (GoK_79), (KMS_82)).

Une autre approche est proposée dans (DeZ_83), dans laquelle la décomposition en modules est fondée sur l'étude des communications du système spécifié. Un langage est associé à

cette méthodologie, adapté notamment à la conception d'applications réparties de conduite de procédés (Zak_84).

Enfin, une méthodologie de ce type, pour des systèmes utilisant des données partagées, est proposée dans (LeH_82) et (Len_82), où on présente une démarche par raffinements successifs, qui, à partir d'une spécification des résultats du problème, spécifie à chaque étape des "relations sémantiques" (idempotence, commutativité, non interférence et indépendance) entre les énoncés introduits: le but est ici seulement de spécifier les contraintes que devra respecter toute implémentation -éventuellement sous la forme d'un programme parallèle- du problème considéré.

- L'application d'autres méthodes classiques aux problèmes du parallélisme. Citons une démarche "à la Jackson" (Jac_75) représentée sous forme de processus CSP (KeM_80).

- La recherche d'une décomposition d'une spécification en un ensemble de "sous-spécifications" le plus indépendantes possibles. Citons (Elf_82) où on propose une décomposition en "couches" non communicantes.

Du point de vue transformation de programmes, nous rappelons les quelques propositions suivantes:

- Transformation d'un programme séquentiel en un programme parallèle:

A partir d'un programme non déterministe, on dérive dans (Sin_79) un programme distribué, selon les principes suivants: substitution de variables locales aux variables initiales, et définition d'une fonction "totalisante", comparable à la définition de "variables auxiliaires", et maintenant des conditions équivalentes pour les énoncés parallèles.

Une autre approche, basée sur les réseaux de Petri, est présentée dans (Jan_81).

- Transformations de programmes parallèles:

Dans (MaS_83) on présente trois types de transformations de programmes CSP:

. substitution de processus (et communications) à des variables,

. transformations préservant les pré et post-conditions

initiales: à titre d'exemple, on synthétise de manière informelle le programme de calcul du PGCD présenté au chapitre 3, dans la forme (AFR₈₀),

. modification des liaisons entre processus.

En un certain sens, cette méthode peut être rapprochée de celle présentée dans cette thèse, puisqu'elle considère initialement une relation entre données et résultats des processus: cette relation est établie par l'introduction de processus supplémentaires (premier type de transformation), d'où un aspect modulaire analogue à celui introduit par la notion de type de communication; la définition de ces processus et de leurs communications (choix du "bon" type de communication) peut être faite par approximations successives, pour garantir des propriétés invariantes (deuxième type de transformation); enfin, l'aspect modulaire obtenu favorise une réflexion sur des transformations (troisième type) de l'ensemble des liaisons du système.

Nous illustrons maintenant, à l'aide de trois exemples, l'apport de la notion de type de communication dans la conception de programmes:

- le premier exemple, dans le domaine des calculs scientifiques, montre différentes modifications de programme, par la définition et différentes implantations de différentes relations de communication,

- le deuxième exemple traite un problème de cinématique de fichiers. Il a pour but de montrer comment la notion de type de communication permet de développer un raisonnement pour passer d'une spécification -très simple, sur l'exemple- à un ensemble de solutions: la spécification est identifiée à la définition d'une relation de communication exprimée en utilisant la notion de suite temporelle. Cette relation est ensuite réalisée en utilisant un type de communication simple, dont différentes implantations conduisent à différentes solutions, y compris un programme séquentiel.

- le troisième exemple, sur un problème de commande de procédé, illustre la notion de relation de communication, comme outil de spécification de problèmes. Là aussi, la spécification initiale est exprimée en utilisant la notion de suite temporelle et montre que le problème revient à définir un type de communication particulier: la construction d'une solution de ce problème revient à la construction de ce type. Celle-ci est menée de façon déductive. Le type de

communication est ensuite représenté en ADA.

7-2- EXEMPLE 1- RESOLUTION D'UN SYSTEME D'EQUATIONS AUX DERIVEES PARTIELLES- (Per₈₃)

Nous montrons sur cet exemple les conséquences de la notion de type de communication sur les transformations de programmes parallèles, et, en particulier l'intérêt des choix d'implémentation définis au chapitre 5.

7-2-1- Définition du problème-

Le problème traité est celui de la résolution d'un système d'équations aux dérivées partielles à forte non linéarité, avec des conditions de Dirichlet, du type suivant:

$$(a) \quad \begin{cases} -\Delta u + e^u = f \text{ sur } \Omega \subset \mathbb{R}^n \\ u = 0 \text{ sur } \Gamma, \text{ frontière de } \Omega \end{cases}$$

Soit donc à trouver le zéro de la fonction

$$-\Delta u + e^u - f$$

Par application de la méthode de Newton, il vient:

$$(b) \quad -\Delta u^{(p)} + e^{u^{(p-1)}} \cdot u^{(p)} = f + e^{u^{(p-1)}} \cdot (u^{(p-1)} - 1)$$

Par discrétisation du problème (a) par différences finies, on se ramène à un problème matriciel de la forme:

$$(a') \quad A \cdot v + B(v) = F \\ \text{où } B(v) \text{ est le vecteur } (\dots, e^{v_i}, \dots)$$

L'algorithme (b) devient alors:

$$(b') \quad A \cdot v^{(p)} + D^{(p-1)} \cdot v^{(p)} = S^{(p-1)}$$

où $v^{(p)}$ est un vecteur de $\mathbb{R}^n$

$D^{(k)}$ est une matrice diagonale dont

les éléments non nuls sont les $e^{v_i^{(k)}}$

. $s^{(k)}$ est un vecteur de la forme
 $F + D^{(k)} \cdot (v^{(k)} - 1)$

On peut alors considérer que chaque pas de l'algorithme (b'), qui est un analogue discret du problème de Poisson (cf. paragraphe 3-4-1-), s'exprime, avec une numérotation "rouge-noir" (SaS_81) des points de discrétisation, comme un système linéaire de la forme suivante:

$$(c) \quad \begin{pmatrix} TR & U \\ U^T & TN \end{pmatrix} \begin{pmatrix} wR \\ wN \end{pmatrix} = \begin{pmatrix} gR \\ gN \end{pmatrix}$$

On propose dans (SaS_81) de résoudre (c) par l'algorithme cyclique de Chebychev (qui est une accélération de la méthode de Jacobi par blocs):

$$(d) \quad \left\{ \begin{array}{l} \cdot xR^{(0)} \text{ est une approximation initiale de } wR \\ \cdot xN^{(1)} = TN^{-1} \cdot (gN - U^T \cdot xR^{(0)}) \\ \cdot \text{pour tout } k \geq 0: \\ \quad xR^{(2k)} = (1 - \omega^{(2k-2)}) xR^{(2k-2)} \\ \quad \quad + \omega^{(2k)} TR^{-1} \cdot (gR - U \cdot xN^{(2k-1)}) \\ \quad xN^{(2k+1)} = (1 - \omega^{(2k-1)}) xN^{(2k-1)} \\ \quad \quad + \omega^{(2k+1)} TN^{-1} \cdot (gN - U^T \cdot xR^{(2k)}) \end{array} \right.$$

où les coefficients $\omega^{(j)}$ sont calculés à partir de bornes encadrant les valeurs propres d'une matrice définie par une factorisation LDL^T des blocs TN et TR.

La convergence de cet algorithme peut être accélérée par l'implantation du $\mathcal{E}$ -algorithme vecteur de Wynn:

$$(e) \quad \left\{ \begin{array}{l} \cdot z_{(-1)}^{(k)} = 0 \text{ pour tout } k \geq 0 \\ \cdot z_{(0)}^{(k)} = xR^{(2k)} \text{ pour tout } k \geq 0 \\ \cdot z_{(j+1)}^{(k)} = z_{(j-1)}^{(k+1)} + \frac{z_{(j)}^{(k+1)} - z_{(j)}^{(k)}}{\|z_{(j)}^{(k+1)} - z_{(j)}^{(k)}\|_2^2} \end{array} \right.$$

pour tout $k, j \geq 0$

qui est tel que la suite $z_{(2j)}^{(k)}$ converge plus vite que la suite $z_{(0)}^{(k)}$ vers la solution wR .

7-2-2-L, Différentes solutions-

L'implantation parallèle de l'algorithme (b') peut être réalisée par la définition de trois processus:

- P1: calcul de la dérivée $(\dots, e^{v_i(p-1)}, \dots)^T$
- P2: calcul du second membre et du pième itéré $v^{(p)}$ par l'algorithme cyclique de Chebychev (d)
- P3: accélération de la convergence par le $\mathcal{E}$ -algorithme vecteur (e)

La solution présentée dans (SaS_81) est dérivée de la forme suivante:

processus S ::

port vR, xR : égalité; vN : aléatoire;
 D : égalité;

processus P1 ::

entrée vN, vR : demi_vecteur;
sortie D : matrice;
corps

```

.
.
* [ .
.
consommer(vR) ;
consommer(vN) ;

-- calcul de  $D = D^{(p)}$  --
produire(D) ;
.
.
]
.
.
fin P1;

```

```

processus P2 ::
  entree D: matrice;
  sortie vN, xR: demi_vecteur;
  corps
  .
  vN:=vN(0) i produire(vN) i
  * └ consommer(D) i
    .
    -- calcul de S(P) --
    * └ .
      .
      -- calcul de xR=xR(2k) --
      produire(xR) i
      .
      -- calcul de vN=xN(2k+1) --
      produire(vN) i
      .
    ]
  .
]
.
fin P2;

processus P3 ::
  entree xR: demi_vecteur;
  sortie vR: demi_vecteur;
  corps
  .
  vR:=vR(0) i produire(vR) i
  * └ .
    .
    * └ .
      .
      consommer(xR) i
      * └ .
        .
        -- calcul de vR=z (w) --
        (j)
        .
      ] i
    .
  ] i
  .
  ] i
  produire(vR) i

```

```

fin P3;

composition

    P1 // P2 // P3

fin S;

```

La version présentée dans (SaS_81) est obtenue en restreignant les types de communication "égalité" au type "tampon à une place" présenté au paragraphe 5-3-2-. Dans l'algorithme obtenu, on peut observer une exécution parallèle synchrone des processus "P2" et "P3": pendant que "P2"

calcule $x_R^{(2k)}$ et $x_N^{(2k+1)}$, le processus "P3" calcule un certain nombre d'itérés $z_{(j)}^{(k)}$. Lorsque ce processus "P3" a convergé, il produit la moitié "vR" du vecteur "v" au processus "P1", qui consomme alors la plus récente valeur de l'autre moitié "vN" de ce vecteur: le nouveau vecteur "v" est alors une donnée de "P1", qui produit la matrice "D" par un nouveau pas d'itération de l'algorithme (b').

On peut rendre cet algorithme asynchrone en deux points, et par là améliorer son efficacité:

- entre "P2" et "P3". En effet, en l'absence d'hypothèses ou de contraintes sur les performances des processeurs exécutant les processus "P2" et "P3", on peut maintenir le type

"égalité" dans sa généralité. Le nombre d'itérés $xR^{(2k)}$ dans le "tampon de communication" est alors quelconque.

- entre "P3" et "P1". Le calcul de la dérivée peut être désynchronisé et effectué en parallèle avec les calculs de "P2" et "P3". A l'algorithme (b') on substitue alors l'algorithme suivant:

$$(b'') \quad A.v(p) + D(s(p)).v(p) = S(s(p))$$

où $D(s(k))$ est une matrice diagonale dont les

des fichiers "hd" et "hr" par:

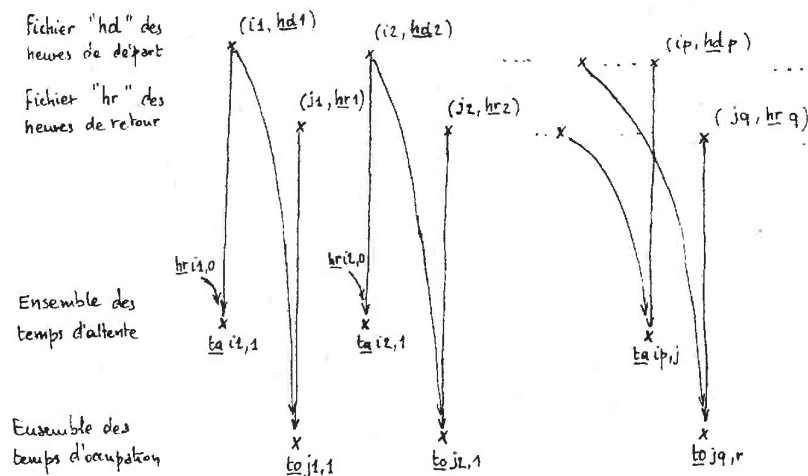
$\forall i \in [1, n], \forall j \in [1, k_i]$
 $\text{hd}_{ij} = h \text{ tel que } (i, h) \in \text{hd}$
 $\text{et card}\{(i, h') \in \text{hd}, h' < h\} = j - 1$
 $\text{hr}_{ij} = h \text{ tel que } (i, h) \in \text{hr}$
 $\text{et card}\{(i, h') \in \text{hr}, h' < h\} = j - 1$

et $\text{hd}_{ik_i+1} = \text{"fin"}$
 $\text{hr}_{i0} = \text{"début"}$

où "début" et "fin" sont les heures respectives de début et de fin de la période de location dans la journée considérée (on suppose que tous les bateaux loués sont de retour au plus tard à la fin de la période de location).

On peut résumer cette spécification par le schéma présenté sur la figure 1 ci-dessous: les ensembles d'attentes et d'occupations sont ordonnés de gauche à droite dans l'ordre de calcul, qui dépend de la "disponibilité" des données extraites des fichiers "hd" et "hr".

Ce faisant, à partir de la spécification fonctionnelle ci-dessus, on entreprend une démarche de construction de solutions: l'idée est ici de substituer au rang des éléments des fichiers "hd" et "hr" un "instant de disponibilité" de ces éléments relativement à un algorithme de parcours séquentiel de ces fichiers. Cette idée peut être concrétisée par l'introduction de suites temporelles associées à ces fichiers: on en déduit alors une définition de suites temporelles, substituées aux suites ta et to , comme le montre le paragraphe suivant.



- figure 1 -

7-3-2-Définition de communications-

On peut "lire" sur le schéma précédent un ensemble de relations de communication: pour tout i de 1 à n , la suite des calculs

$$\{t_{ai,j}; j=1, k_i+1\}$$

(respectivement, la suite des calculs $\{t_{oi,j}; j=1, k_i\}$) nécessite une suite de "consommations" des valeurs " $h_{di,j}$ " et " $h_{ri,j}$ " "produites" par les fichiers "hd" et "hr". On peut donc compléter ce schéma en introduisant les ensembles de valeurs consommables " H_{Di} " et " H_{Ri} " adéquats.

Plus formellement, on peut spécifier ces relations par la définition des suites temporelles suivantes:

Pour tout i de 1 à n , la suite temporelle " TA_i " des temps d'attente du i ème bateau est formée de couples

$$(valeur(TA_{ij}), instant(TA_{ij}))$$

définis de la manière suivante:

Soient " t_j " l'instant auquel le j ème terme (ip, h_{dp}) tel que $ip=i$ est disponible,

" $H_{Ri,j}$ " l'ensemble des heures de retours du i ème bateau, disponibles à l'instant du calcul de " TA_{ij} ":

$$H_{Ri,j} = \begin{cases} \text{si } H_{Ri,j-1} \neq \emptyset \text{ alors } H_{Ri,j} \\ \text{sinon premier}((q, h_{rq}), t_j) \text{ tel que} \\ q=i \text{ et } t_j < t_{j-1} \end{cases}$$

où " $H_{Ri,j}$ " est l'ensemble des heures de retour du i ème bateau, disponibles à l'instant " t_j ":

$$H_{Ri,j} = H_{Ri,j-1} - ((i, hr), t_{j-1}) \cup \{(q, h_{rq}), tk \text{ tel que } q=i \text{ et } t_{j-1} < tk < t_j\}$$

Alors $valeur(TA_{ij}) = h_{dp} - h_{rj}$

où h_{rj} est tel que $((i, h_{rj}), t_j) \in H_{Ri,j}$

$$\text{et } \forall ((i, hr), tk) \in H_{Ri}, t_j' < tk$$

$$\text{instant}(TA_{ij}) = \min(t_j, t_j')$$

On définit symétriquement la suite " TO_i " à partir des ensembles " $H_{Di,j}$ ".

Ces ensembles peuvent donc être considérés comme les ensembles consommables des types de communication des données échangées entre les processus "HD" et "HR" et les processus " TA_i " et " TO_i ", pour tout i de 1 à n : d'après la définition de ces ensembles, les types de communication sont identiques au type "égalité". On obtient donc le système suivant, dans lequel on a trivialement fusionné les processus " TA_i " et " TO_i " en un seul processus " P_i ", pour tout i de 1 à n , et inclus le calcul des maxima.

processus bateau ::

port ($h_{di}, h_{ri}; i=1, n$): égalité;

processus HD ::

sortie (h_{di} : heure; $i=1, n$);

corps

*[non fdf(hd) --> lire(i, h);
hdi:=h; produire(hdi)

] ;

(hdi:="fin"; produire(hdi); $i=1, n$)

fin HD;

processus HR ::

sortie (h_{ri} : heure; $i=1, n$);

corps

*[non fdf(hr) --> lire(i, h);
hri:=h; produire(hri)

] ;

fin HR;

pour tout i de 1 à n :

processus P_i ::

entrée h_{di} : heure via h_{di} ;

h_{ri} : heure via h_{ri} ;

booléen bd, br ;

corps

$bd:=\text{faux};$

$h_{ri}:=\text{"début"}; br:=\text{vrai};$

*[$bd \wedge br$ --> [$hd < hr$ --> tom(i); $bd:=\text{faux}$

[$hd > hr$ --> tam(i); $br:=\text{faux}$

```

    ]
    0 7 bd --> consommer(hd); bd:=vrai
    0 7 br --> consommer(hr) ; br:=vrai
  fin Pi;
  composition
    HD // HR (// Pi; i=1,n)
  fin bateau;

```

7-3-3-Représentations du système obtenu-

Nous étudions deux représentations, selon qu'on implante le type de communication "égalité" dans toute sa généralité par une file non bornée, ou en le restreignant par un tampon à une place.

Dans le premier cas, on obtient un algorithme CSP du type suivant:

```

BATEAU :: [ HD // HR // (FDi // FRi // Pi; i=1,n) ]
  -- les processus "FDi" et "FRi" représentent les
  -- ensembles consommables des types de
  -- communication "égalité"

HD :: [
  .
  * [ lire(i,hd); FDi!hd ]
  .
  ]

HR :: [
  .
  * [ lire(i,hr); FRi!hr ]
  .
  ]

```

Pour tout i de 1 à n:

```

FDi :: [ var a: ...;
  .

```

```

  .
  queue:=1; tete:=1;
  * [ HD?a[queue] --> queue:=queue+1
    0 Pi?bd; queue>tete --> Pi!a[tete];
      tete:=tete+1
    ]
  ]
  FRi :: [ var a: ...;
  .
  .
  queue:=1; tete:=1;
  * [ HR?a[queue] --> queue:=queue+1
    0 Pi?br; queue>tete --> Pi!a[tete];
      tete:=tete+1
    ]
  ]
  Pi :: [
  .
  * [ bd^br --> [ hd<hr --> tom(i); bd:=faux
                0 hd>hr --> tam(i); br:=faux
                ]
    0 7 bd; FDi!bd; FDi?hd --> bd:=vrai
    0 7 br; FRi!br; FDi?hr --> br:=vrai
    ]
  ]

```

On transforme cet algorithme en fusionnant les processus "Pi", "FDi" et "FRi", par identification des variables communiquées. On obtient, pour tout i de 1 à n:

```

Pi :: [
  .
  * [ bd^br --> [ hd<hr --> tom(i); bd:=faux
                0 hd>hr --> tam(i); br:=faux
                ]
    0 7 bd; qd>td --> hd:=ad[td]; td:=td+1;
      bd:=vrai
    0 7 br; qr>tr --> hr:=ar[tr]; tr:=tr+1;
      br:=vrai
    0 HD?ad[qd] --> qd:=qd+1
    0 HR?ar[qr] --> qr:=qr+1
    ]
  .
  ]

```

A partir de cet énoncé on dérive une solution

séquentielle classique, en choisissant systématiquement les deux dernières alternatives de l'itération: les fichiers "hd" et "hr" sont alors triés par bateau. Puis, pour chaque bateau on parcourt les listes triées des heures de départ et de retour.

Revenons au système "bateau" du paragraphe 7-2-2-, pour lequel on choisit une implantation de chaque type de communication "égalité" par un tampon à une place, réalisé par les processus CSP suivants:

Pour tout i de 1 à n:

```
VDi :: [ .
        .
        plein:=faux;
        * [ r plein; HD?hd --> plein:=vrai
            [] Pi?bd; plein --> Pi!hd; plein:=faux
          ]
      ]
```

```
VRi :: [ .
        .
        plein:=faux;
        * [ r plein; HR?hr --> plein:=vrai
            [] Pi?br; plein --> Pi!hr; plein:=faux
          ]
      ]
```

Par fusion des processus "Pi", "VDi" et "VRi", il vient:

```
P'i :: [ .
        .
        * [ bd^hr --> [ hd<hr --> tom(i); bd:=faux
                      [] hd>hr --> tam(i); br:=faux
                    ]
          [] r bd; pd --> bd:=vrai; pd:=faux
          [] r br; pr --> br:=vrai; pr:=faux
          [] r pd; HD?hd --> pd:=vrai
          [] r pr; HR?hr --> pr:=vrai
        ]
      ]
```

Remarque-

On peut simplifier cet énoncé (l'énoncé obtenu n'est pas équivalent) en spécifiant un rendez-vous HD?hd (respectivement HR?hr) seulement lorsque "bd" est à faux (resp. "br" est à faux). Dans ces conditions, les booléens "bd" et "br" sont équivalents aux booléens "pd" et "pr", et on obtient:

```
P'i :: [ .
        .
        * [ bd^hr --> [ hd<hr --> tom(i); bd:=faux
                      [] hd>hr --> tam(i); br:=faux
                    ]
          [] r bd; HD?hd --> bd:=vrai
          [] r br; HR?hr --> br:=vrai
        ]
      ]
```

A partir de l'énoncé précédent, on dérive une solution séquentielle en fusionnant les processus "HD", "HR" et "P'i", pour i de 1 à n:

- un tableau de variables "hd[i]", i=1,n (resp. "hr[j]", j=1,n) représente les données "hd" (resp. "hr") des processus "Pi",

- la communication des données est réalisée par identification des variables communiquées "hd" (élément courant du fichier "hdep") et "hr" (élément courant du fichier "hret"),

- on choisit de privilégier un des fichiers, par exemple "hdep": pour chaque élément de ce fichier, dont l'heure de départ est "hd", l'ensemble des booléens "pr" (dans l'intervalle [1,n]), mis à vrai par lecture de l'élément correspondant sur le fichier "hret", est déterminé par les éléments correspondants à une heure de retour "hr" inférieure à "hd". Pour ces éléments on est dans la situation où on calcule "tom(i)". De plus pour l'élément courant dont l'heure de départ est "hd" on est dans la situation où on calcule "tam(i)".

- le programme se termine par la lecture des derniers

éléments du fichier "hret", puis par les derniers calculs de "tam(i)" pour chaque bateau "i".

Expression de ce programme en PASCAL:

```

program bateau;
const n=...;
type fichier = file of record num: 1..n;
                        time: heure
                    end;
var hdep,hret: fichier;
    hd,h,am,om: array[1..n] of heure;
    i,j: 1..n;
procedure tam(x,y: heure; var z: heure);
.
.
procedure tom(x,y: heure; var z: heure);
.
.
begin
  for j:=1 to n do begin hr[j]:="debut";
                        om[j]:=0
                    end;
  reset(hdep); reset(hret);
  while not eof(hdep) do
    begin
      while hret^.time<hdep^.time do
        begin
          j:=hret^.num;
          hr[j]:=hret^.time;
          tom(hd[j],hr[j],om[j]);
          get(hret)
        end;
      i:=hdep^.num; hd[i]:=hdep^.time;
      tam(hd[i],hr[i],am[i]);
      get(hdep)
    end;
  while not eof(hret) do
    begin
      j:=hret^.num; hr[j]:=hret^.time;
      tom(hd[j],hr[j],om[j]);
      get(hret)
    end;
  for i:=1 to n do
    begin
      hd[i]:="fin";
      tam(hd[i],hr[i],am[i])
    end
end.

```

7-4- EXEMPLE 3- CONTROLEUR D'UNITE DE DISQUE-

7-4-1-Présentation informelle-

L'exemple traité est celui de la spécification d'un contrôleur d'unité de disque: il s'agit de gérer les différentes demandes d'accès au disque en optimisant les déplacements du bras selon la stratégie, dite "de l'ascenseur" (cet exemple est traité dans (AbS_79)).

Le disque est composé de pistes concentriques numérotées en croissant vers l'intérieur. Une demande d'accès est composée d'un numéro de piste à atteindre et d'un code d'opération à effectuer (lecture ou écriture). Dans la suite nous négligerons ce dernier code. En réponse aux demandes d'accès le système déplace le bras de la manière suivante:

- à partir d'une position donnée du bras, la prochaine piste atteinte est la plus proche pour laquelle il existe une demande d'accès, et dans la direction courante du bras (vers l'intérieur ou vers l'extérieur),

- la direction est inversée si la première ou la dernière piste est atteinte, ou s'il n'existe plus de demande d'accès dans la direction courante et s'il en existe dans la direction opposée,

- deux demandes d'accès à la même piste sont satisfaites dans leur ordre d'arrivée.

L'ensemble des données du problème forme donc une suite, à priori infinie, de demandes d'accès de pistes, et l'ensemble des résultats est la suite infinie des satisfactions des demandes de piste.

7-4-2-Spécification fonctionnelle- (cf. (FiJ_80), (FJJ_84), (JuP_85))

Nous cherchons à définir une relation entre données et résultats du problème, de la façon suivante:

- les données sont la suite infinie des demandes d'accès,

$$\underline{d} = (\underline{d}_1, \underline{d}_2, \dots, \underline{d}_j, \dots)$$

où chaque terme " $\underline{d}_j$ " est un numéro de piste,

- les résultats sont la suite infinie des satisfactions,

$$\underline{r} = (\underline{r}_1, \underline{r}_2, \dots, \underline{r}_i, \dots)$$

où chaque terme " $\underline{r}_i$ " est un numéro de piste,

- la relation entre ces suites est définie comme une bijection φ de " R ", domaine de " $\underline{r}$ ", sur " D ", domaine de " $\underline{d}$ ", telle que deux demandes d'accès à la même piste sont satisfaites dans leur ordre d'arrivée, soit:

$$\forall j_1, j_2 \in D \text{ tels que } \underline{d}_{j_1} = \underline{d}_{j_2} \text{ et } j_1 < j_2, \\ \varphi(i_1) = j_1 \text{ et } \varphi(i_2) = j_2 \Rightarrow i_1 < i_2$$

Un exemple trivial de bijection pourrait être de choisir:

$$\forall i \in R \quad \varphi(i) = i.$$

En termes de relation de communication, si la suite $\underline{d}$ est l'ensemble des demandes produites, et la suite $\underline{r}$ est l'ensemble consommé (des demandes satisfaites), cette bijection est exprimée par le type de communication "égalité".

On constate bien alors qu'une telle solution ne répond pas au problème posé, en ignorant la contrainte d'optimisation indiquée dans l'énoncé.

Pour construire une solution satisfaisante, il s'agit de définir une relation qui considère "à un moment donné" (celui du choix de la demande à satisfaire) l'état du système, c'est-à-dire la position courante du bras, sa direction et l'ensemble des demandes en attente. On doit donc spécifier la relation d'une façon constructive, qui définit le terme général de la suite $\underline{r}$ par une équation récurrente de la forme suivante:

$$\forall i \in R - \{1\} \quad \underline{r}_i = \text{asc}(\underline{r}_{i-1}, \underline{d}_{i-1}, \text{CAND}_i)$$

où . la fonction "asc" définit le terme " $\underline{r}_i$ " en fonction des arguments suivants,

. $\underline{d}_{i-1}$ est la direction courante du bras (vers l'intérieur ou vers l'extérieur),

. CAND_i est l'ensemble des demandes d'accès à satisfaire au "moment" de définir " $\underline{r}_i$ ".

En substituant des suites temporelles aux suites " $\underline{d}$ " et " $\underline{r}$ ", on obtient:

$\forall i \in R - \{1\}$:

$$\underline{r}_i = \text{asc}(\underline{r}_{i-1}, \underline{d}_{i-1}, \text{CAND}_i)$$

$$\text{CAND}_i = \begin{array}{l} \text{si } \text{NSERV}_{i-1} \neq \{\} \text{ alors } \text{NSERV}_{i-1} \\ \text{sinon } \{(d_j, \text{instant}(d_j))\} \text{ tel que} \\ \text{instant}(d_{j-1}) < \underline{t}_{si} < \text{instant}(d_j) \\ \text{fsi} \end{array}$$

où " $\underline{t}_{si}$ " est la date à laquelle le système est prêt à satisfaire la demande " $\underline{d}_j$ " avec $j = \varphi(i)$,
 $\underline{t}_{si} = \underline{t}_{si-1} + \text{SYST}(\underline{r}_{i-2}, \underline{r}_{i-1})$

-- SYST est une fonction à résultat dans l'ensemble "CHRONO", représentant le temps de déplacement du bras

$\forall i \in R$:

$$\text{NSERV}_i = \text{CAND}_i - \{(r_i, \text{instant}(r_i))\} \\ \cup \{(d_j, \text{instant}(d_j)) \text{ tel que } \text{instant}(r_i) < \text{instant}(d_j) < \underline{t}_{si} + 1\}$$

-- ensemble des demandes non satisfaites au moment où le système est prêt à satisfaire le i -ième service, c'est-à-dire à la date " $\underline{t}_{si} + 1$ "

--

$$\underline{r}_1 = \begin{array}{l} \text{si } \text{NSERV}_1 \neq \{\} \text{ alors } \underline{d}_{si} \\ \text{sinon } \text{instant}(d_j) \text{ tel que} \\ \text{instant}(d_{j-1}) < \underline{t}_{si} < \text{instant}(d_j) \\ \text{fsi} \end{array}$$

Et pour le premier terme de la suite $\underline{r}$:

$$\underline{r}_1 = \text{asc}(n0, "inter", \text{CAND}_1)$$

où "n0" est le plus petit numéro de piste (piste la plus extérieure),

"inter" est la direction initiale de déplacement du bras,

CAND1 = si NSERVO <> {} alors NSERVO
sinon { (dl, instant(dl)) }
fsi

NSERVO = { (dj, instant(dj)) tel que instant(dj) <= tsl }

tsl = 1 indéfini: par exemple, le plus petit élément de l'ensemble CHRONO

• instant(rl) = si NSERVO <> {} alors tsl
sinon instant(dl)
fsi

Le problème se réduit en fait à la spécification d'une relation de communication entre une suite infinie de demandes d'accès (opérations de production) et une suite infinie de satisfactions de ces demandes (opérations de consommation).

La fonction "asc" définit le ième élément de la suite résultat, choisi parmi l'ensemble "CANDi" des demandes non satisfaites: dans les termes du modèle formel présenté dans cette thèse, cet ensemble est représenté par l'ensemble des données consommables. Le résultat de la fonction "asc" est défini par l'opération "val-cons" du type de communication suivant:

Première spécification-

On note <SP,A,SC> le triplet du type de communication défini par

pré-cons (<SP,A,SC>) = non vide?(A)
val-cons (<SP,A,SC>) = extraire(A, j(A, SC))
post-cons (<SP,A,SC>) = disparaître(A, j(A, SC))

où la fonction "j" retourne un entier positif défini par les règles suivantes:

(1) cardinal(SC)=0 ==> j(A, SC) ∈ mini{ i tel que dans?(A, i) }

(2) dir(A, SC) = "inter" ∧ supérieur(A, dernier(SC)) ≠ ∅ ==>
j(A, SC) ∈ mini(supérieur(A, dernier(SC)))

(3) dir(A, SC) = "exter" ∧ inférieur(A, dernier(SC)) ≠ ∅ ==>
j(A, SC) ∈ maxi(inférieur(A, dernier(SC)))

Dans ces définitions "supérieur", "inférieur", "mini" et "maxi" sont des opérations sur le type "ensemble-indexé", définies ci-dessous.

La fonction "dir", à résultat dans {"inter", "exter"} représente la direction choisie pour le déplacement du bras. Elle est définie de la manière suivante:

(4) cardinal(SC)=0 ==> dir(A, SC) = "inter"

(5) (inférieur(A, dernier(SC)) = ∅ ∨ $\overline{\text{dir}}(\text{SC}) = \text{"inter"}$) ∧
supérieur(A, dernier(SC)) ≠ ∅ ==> dir(A, SC) = "inter"

(6) (supérieur(A, dernier(SC)) = ∅ ∨ $\overline{\text{dir}}(\text{SC}) = \text{"exter"}$) ∧
inférieur(A, dernier(SC)) ≠ ∅ ==> dir(A, SC) = "exter"

On a fait l'hypothèse qu'initialement le bras est positionné sur la piste de plus petit numéro, et que la direction de déplacement est "inter". Dans le cas général, la direction choisie pour le déplacement du bras dépend de sa direction courante déterminée par la fonction "dir":

(7) si existe(k) tel que dans?(SC, k)
∧ dernier(SC) < extraire(SC, k)
∧ dans?(SC, l) ∧ l > k ==> dernier(SC) = extraire(SC, l)
alors dir(SC) = "exter"
sinon dir(SC) = "inter"
fsi

Dans les définitions précédentes, si "s" est un objet de type "ensemble-indexé" et $I \subset \{i \text{ tel que } \text{dans?}(s, i)\}$ on utilise les opérations suivantes:

inférieur(s, x) = { i tel que dans?(s, i) ∧ extraire(s, i) = x }

supérieur(s, x) = { i tel que dans?(s, i) ∧ extraire(s, i) > x }

mini(I) = { j ∈ I tel que $\forall i \in I$ extraire(s, j) = < extraire(s, i) }

$$\text{maxi}(T) = \{i \in I \text{ tel que } \forall i \in I \text{ extraire}(s, j) \geq \text{extraire}(s, i)\}$$

Remarque-

Ce type de communication rend compte d'une situation simplifiée par rapport à la spécification informelle du problème: il ne fait apparaître que le choix du numéro de piste parmi l'ensemble des demandes non satisfaites. Pour satisfaire les demandes d'accès à la même piste dans leur ordre d'arrivée, on enrichit la spécification précédente, en précisant la définition de "j":

- (1') $\text{cardinal}(SC) = 0 \implies$
 $j(A, SC) = \text{ainé}(\text{mini}\{i \text{ tel que dans?}(A, i)\})$
- (2') $\text{dir}(A, SC) = \text{"inter"} \wedge \text{supérieur}(A, \text{dernier}(SC)) \neq \emptyset \implies$
 $j(A, SC) = \text{ainé}(\text{mini}(\text{supérieur}(A, \text{dernier}(SC))))$
- (3') $\text{dir}(A, SC) = \text{"exter"} \wedge \text{inférieur}(A, \text{dernier}(SC)) \neq \emptyset \implies$
 $j(A, SC) = \text{ainé}(\text{maxi}(\text{inférieur}(A, \text{dernier}(SC))))$

où la fonction "ainé" détermine le plus petit élément d'un ensemble d'indices entiers positifs ("ainé" est mis pour rappeler que ce plus petit indice représente la plus ancienne demande).

Seconde spécification-

Pour prévenir les risques de coalitions, on impose que la satisfaction des demandes d'accès à une même piste ne soit pas obtenue "Nmax" fois consécutivement. On enrichit la spécification précédente, en modifiant ainsi la définition de "j", qui utilise l'opération "même" définie sur le type "ensemble-indexé" au paragraphe 2-1-3-:

- (1'') $\text{cardinal}(SC) = 0 \implies$
 $j(A, SC) = \text{ainé}(\text{mini}\{i \text{ tel que dans?}(A, i)\})$
- même(SC) < Nmax $\wedge$
 (2'') $(\text{dir}(A, SC) = \text{"inter"} \wedge$
 $\text{supérieur}(A, \text{dernier}(SC)) \neq \emptyset \implies$
 $j(A, SC) = \text{ainé}(\text{mini}(\text{supérieur}(A, \text{dernier}(SC))))$
- $\vee$
 (3'') $(\text{dir}(A, SC) = \text{"exter"} \wedge$
 $\text{inférieur}(A, \text{dernier}(SC)) \neq \emptyset \implies$

$j(A, SC) = \text{ainé}(\text{maxi}(\text{inférieur}(A, \text{dernier}(SC))))$

• même(SC) = Nmax $\wedge$
 (a) $(\text{dir}(A, SC) = \text{"inter"} \wedge$
 $\text{sup_strict}(A, \text{dernier}(SC)) \neq \emptyset \implies$
 $j(A, SC) = \text{ainé}(\text{mini}(\text{sup_strict}(A, \text{dernier}(SC))))$

$\vee$
 (b) $(\text{dir}(A, SC) = \text{"exter"} \wedge$
 $\text{inf_strict}(A, \text{dernier}(SC)) \neq \emptyset \implies$
 $j(A, SC) = \text{ainé}(\text{maxi}(\text{inf_strict}(A, \text{dernier}(SC))))$

$\vee$
 (c) $(\text{inf_strict}(A, \text{dernier}(SC)) =$
 $\text{sup_strict}(A, \text{dernier}(SC)) \neq \emptyset \wedge$
 $\text{supérieur}(A, \text{dernier}(SC)) \neq \emptyset \implies$
 $j(A, SC) = \text{ainé}(\text{mini}(\text{supérieur}(A, \text{dernier}(SC))))$

où les opérations "inf_strict" et "sup_strict" sont définies sur un objet "s" de type "ensemble-indexé" par:

$\text{inf_strict}(s, x) = \{i \text{ tel que dans?}(s, i) \wedge \text{extraire}(s, i) < x\}$

$\text{sup_strict}(s, x) = \{i \text{ tel que dans?}(s, i) \wedge \text{extraire}(s, i) > x\}$

Le système ainsi spécifié assure que toute demande d'accès à une piste est satisfaite dans un délai fini. Pour démontrer cette propriété, il suffit de montrer que toute demande la plus ancienne d'accès à une piste est satisfaite dans un délai fini, même lorsque le bras est positionné sur une autre piste. Supposons que cette dernière a un numéro inférieur à celle demandée.

Lemme 1- Le bras choisira la direction de la piste demandée dans un délai fini.

Justification: s'il est déjà dans cette direction ($\text{dir}(SC) = \text{"inter"}$), il n'en changera pas (d'après (5)). S'il n'y est pas ($\text{dir}(SC) = \text{"exter"}$), tant que la condition (6) est vraie, cette direction ne change pas: cette condition devient fausse dans un délai fini, d'après (b) et (5).

Lemme 2- Lorsque le bras a choisi la direction demandée, il satisfait la demande de piste dans un délai fini.

Justification: on est dans les conditions $\{\text{dir}(SC) = \text{"inter"} \wedge \text{supérieur}(A, \text{dernier}(SC)) \neq \emptyset\}$ tant que la piste correspondant à la demande n'est pas

atteinte: d'après (5), le déplacement du bras garde la même direction. D'après (a), la piste souhaitée est atteinte dans un délai fini.

7-4-3-Construction d'une solution-

Nous proposons une programmation naïve de cet exemple en ADA, par un programme constitué de deux tâches:

- une tâche "producteur", qui délivre une suite de demandes,
- une tâche "consommateur", qui déplace le bras vers le numéro de piste demandée,

utilisant une procédure "ascenseur", qui réalise le type de communication défini ci-dessus. Nous développons seulement la construction de cette procédure, les autres parties du programme étant triviales.

Remarque préliminaire-

La solution présentée ici conduit à un "programme erroné" selon (IcA_79), puisqu'il ne respecte pas les règles d'accès aux données communes aux tâches "prod" et "cons". On pourrait lui préférer une solution du type de celle présentée dans (Lev_82), où on définit une seule tâche, à deux points d'entrée, et dans le corps de laquelle un énoncé "select" permet d'activer l'une ou l'autre de ces entrées. Celle-ci aurait l'avantage d'être correcte vis-à-vis de toute implémentation du langage. Elle a pour inconvénient de forcer une exclusion mutuelle abusive sur l'ensemble de la représentation du type de communication.

Structures de données représentant le type de communication-

Le triplet $\langle SP, A, SC \rangle$ est représenté par une variable, notée "tc", composée des éléments suivants:

- trois variables: "dir", de type ("inter", "exter"), initialisée à "inter", représente le résultat de la fonction dir,
- "ds", de type num_piste, initialisée à "n0", représente la position courante du

bras (dernière demande servie, sauf cas initial),
"même", de type 0..Nmax, initialisée à 0, représente le nombre de services successifs concernant la piste de la dernière demande servie.

- D'après la spécification précédente, ces trois variables sont suffisantes pour la définition de l'indice de la valeur consommée
-

- une variable "t", de type tableau de (nbr_piste) files_non_bornées, où une file_non_bornée représente la suite des demandes associées à une piste.

- Cette variable représente les ensembles consommables "A". Ce choix de représentation permet de décomposer l'opération de consommation en deux parties:

- . détermination du numéro de piste de la demande à servir: ce numéro représente l'ensemble

```
mini(supérieur(A, dernier(SC))
ou maxi(inférieur(A, dernier(SC))
```

- . consommation dans la file correspondante: une organisation "fifo" réalise très simplement la spécification de l'opération "ainé".

L'élément "SP" du type de communication n'a pas lieu d'être représenté dans la procédure "ascenseur", puisque la stratégie décrite n'est pas définie à partir de l'état de cet ensemble. "SP" est en fait représenté par une variable simple, de type num_piste, dans la tâche "producteur".

Représentation des opérations du type ensemble-indexé-

On se restreint aux opérations utilisées dans la spécification, à leurs conditions d'application, et particularisées selon les structures de données définies ci-dessus.

- . "créer" est réalisée par les initialisations des structures de données,

. les opérations suivantes sont réalisées par des opérations (non rappelées ici) sur le type `file_non_bornée`:

- "ajouter" est réalisée par l'opération "AJOUT" d'adjonction en queue dans la file dont l'indice est donné,

- "extraire" est réalisée par l'opération "TETE" sur la file sélectionnée,

- "disparaître" est réalisée par l'opération "QUEUE" sur la file sélectionnée, qui supprime son élément de tête,

. "vide?" est réalisée par une opération "VIDE?", définie comme la conjonction d'opérations "FILE_VIDE?" sur le tableau de files.

Schéma de la procédure ADA-

procedure ascenseur is

-- ici se placent les déclarations du type "file_non_bornée", de la variable "tc" et des opérations du type "ensemble-indexé"

task prod is

entry produire(x: in num_piste);
end prod;

task cons is

entry consommer(x: out num_piste);
end cons;

task body prod is

loop accept produire(x: in num_piste) do
AJOUT(tc.t x);
end produire;
end loop;

end prod;

task body cons is

suisant, suiv_int, suiv_ext: num_piste;
loop
select
when not VIDE?(tc.t) =>
-- représentation de la précondition,
impliquant une éventuelle attente --

accept consommer(x: out num_piste) do
-- calcul du numéro de piste à servir
"suisant", et de la nouvelle
direction "tc.dir":

--
if tc.même < Nmax then
suiv_int := SUP_EGAL(tc.ds);
-- réalise l'opération
mini(supérieur(A, dernier(SC))),
c'est-à-dire calcule le plus
petit numéro de piste supérieur
ou égal à "tc.ds" pour lequel
il existe une demande (on
retourne 0 s'il n'y a pas de
demande)

--
suiv_ext := INF_EGAL(tc.ds);
-- réalise dans les mêmes
conditions l'opération
maxi(inférieur(A, dernier(SC)))

--
if (suiv_int = 0 and tc.dir = "inter") or
(suiv_ext ≠ 0 and tc.dir = "exter")
then suivant := suiv_ext;
tc.dir := "exter";
else suivant := suiv_int;
tc.dir := "inter";
end if;

else

-- idem avec "SUP" et "INF" à la place
de "SUP_EGAL" et "INF_EGAL", pour
réaliser les opérations
mini(sup_strict(A, dernier(SC)))
et
maxi(inf_strict(A, dernier(SC)))

--
end if;
if tc.ds <> suivant then tc.même := 1;
else tc.même := tc.même + 1;
end if;

-- calcul du nombre de services
successifs correspondants à cette
piste

--
x := TETE(tc.t[suisant]);
QUEUE(tc.t[suisant]);

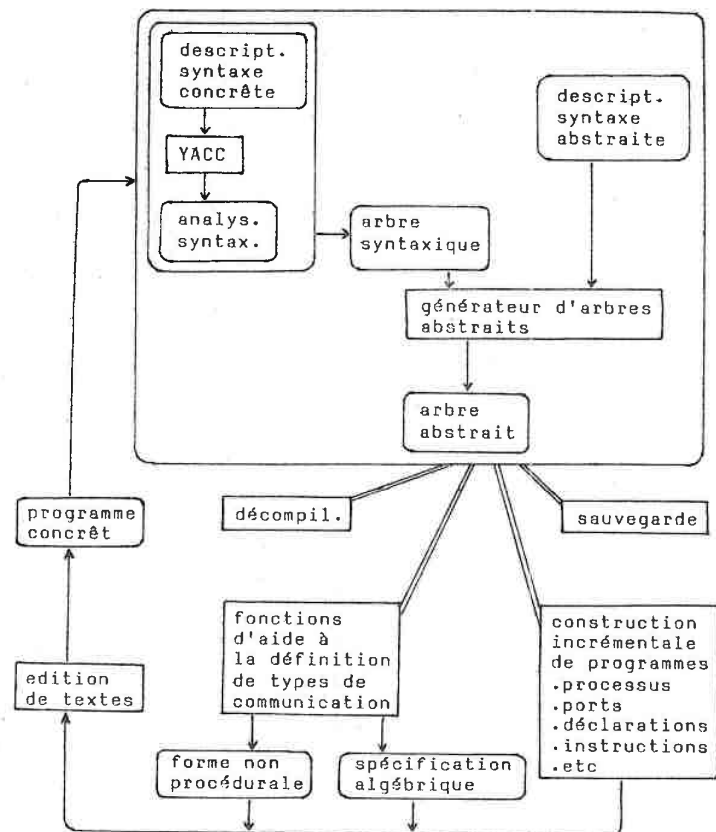
```
        tc.ds:=x  
        end consommer;  
    else null;  
    end select;  
    end loop;  
end cons;  
  
end ascenseur;
```

CONCLUSION

Pour conclure ce mémoire, nous présentons le travail en cours prolongeant les différents points traités et quelques directions de recherche pour l'avenir.

L'essentiel du travail mené autour des idées présentées ici réside dans la réalisation d'un logiciel permettant de valider le concept de type de communication pour la spécification, la construction et la vérification de systèmes parallèles.

Ce logiciel, appelé COMEDIE (Jul_85), est conçu comme un environnement de programmation bâti autour d'un éditeur syntaxique du langage présenté au chapitre 2, écrit en CEYX et organisé comme suit:



Cet éditeur a pour buts:

- de fournir des fonctions d'aide à la définition de types de communications

- soit sous une forme non-procédurale (FJP_83), pour laquelle un traducteur a été réalisé,
 - soit sous forme de spécification algébrique avec introduction progressive des intermédiaires nécessaires,
- de permettre une construction incrémentale des textes des processus, incluant des vérifications de sémantique statique.

Il permet de constituer une bibliothèque de types de communication et de processus utilisables dans la définition d'un système parallèle.

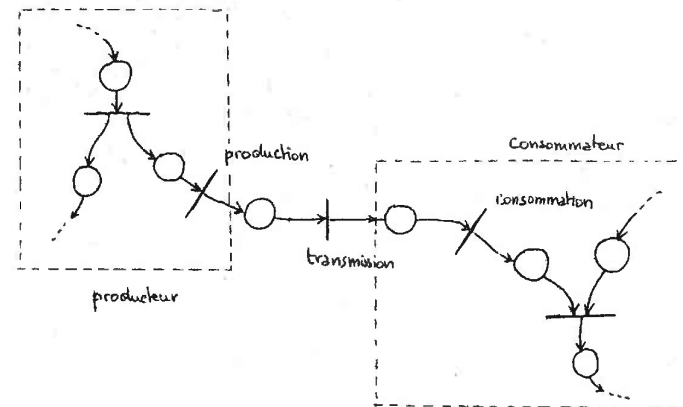
L'ensemble du logiciel COMEDIE comportera les fonctions suivantes:

1. Interprétation d'un système parallèle avec visualisation graphique: le but essentiel est de simuler le comportement des processus et de visualiser une implémentation des ports de communication. Cette partie est en cours de réalisation.
2. Vérification des propriétés d'un système parallèle: les quelques logiciels de vérification effectivement réalisés utilisent généralement l'expression des systèmes parallèles sous forme de réseaux de Petri (citons, par exemple le système CESAR (Que_82)). Cette expression, adaptée à l'étude des propriétés des synchronisations (blocage et propriétés de viabilité), est moins significative en matière de propriétés des communications.

Définie dans un tel contexte, une communication serait en effet caractérisée par:

- les synchronisations qu'elle requiert,
- la structure de donnée utilisée, et les opérations sur cette structure.

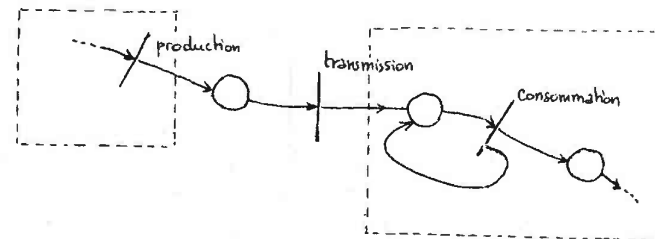
On propose dans (Gui_82) le modèle général suivant, qui exprime un système distribué producteur-consommateur:



où les transitions de production et de consommation sont réduites à des affectations sur une variable simple: il s'agit donc en fait d'une communication de type "égalité" implémentée sur un tampon à une place.

Si on se restreint à ce type de structure de donnée simple, d'autres types de communication ont une expression moins triviale en termes de réseaux de Petri.

Ainsi, les synchronisations associées au type "aléatoire" pourraient être représentées par:



Inversement, on peut utiliser une structure de données plus complexe, manipulée par des opérations portant sur les valeurs de cette structure, non représentables par des réseaux de Petri ordinaires: c'est précisément l'objet du travail développé dans cette thèse.

Nous devons donc réaliser un logiciel implémentant le système de preuves présenté au chapitre 6, comportant en particulier la preuve de correction partielle.

3. Transformation d'un système parallèle par modification des types de communication selon le treillis présenté au chapitre 5.
4. Compilation en langage objet ADA.

D'un point de vue plus fondamental les directions de recherche que nous nous sommes définies, à partir de ce travail, sont les suivantes:

1. Enrichissement du cadre formel proposé: il est bien clair que ce travail a voulu mettre l'accent sur un aspect négligé dans les approches classiques basées sur le concept de synchronisation. En conséquence, il n'est qu'une première idée qui doit encore être explorée quant aux classes de problèmes et de solutions informatiques à atteindre.
2. Etude plus approfondie des propriétés des communications, du type de celles présentées au chapitre 5, et de leurs conséquences sur la construction, la transformation et la preuve des algorithmes.
3. Travail "en aval" par l'étude des relations avec les langages de programmation "classiques", et en particulier des problèmes posés par une implantation répartie.
4. Travail "en amont" sur l'aspect spécification d'un problème et passage à la construction d'une solution, passage dans lequel les notions développées dans cette thèse semblent de bons repères: l'aspect temporel précisé dans une spécification et sa traduction dans une solution ne sont qu'ébauchés au chapitre 7. D'une manière générale, nous devons passer du niveau "outil" et son illustration, développés ici, au niveau "méthode" et "raisonnement". Ceci pose deux classes de problèmes que nous développons ci-dessous:

- a) la puissance et les limites de la notion de relation de

communication,

- b) la définition d'un "véritable" langage de spécification.

Problème a)-

La définition de ce qui relève de l'expression de relations de communication ou de relations de calcul dans la spécification d'un problème n'est pas toujours aisée. Nous évoquons ce point à travers un exemple: celui de la génération de la suite des nombres premiers par le crible d'Eratosthènes (cf. paragraphe 7-1-1-).

L'énoncé définissant le ième nombre premier "pi" est le suivant:

donnée di: suite d'entiers, résultat du crible précédent (pour $i > 2$), ou suite des entiers naturels (pour $i = 2$),

résultat (pi, di+1), où
 . pi est le premier terme de di,
 . di+1: suite d'entiers non divisibles par pi.

La relation entre la suite di et la suite di+1 peut être vue comme une relation de communication, exprimée par le type de communication "crible" suivant:

pré-cons($\langle SP, A, SC \rangle$) = ($q(A, SC) < 0$)
val-cons($\langle SP, A, SC \rangle$) = extraire(A, q(A, SC))
post-cons($\langle SP, A, SC \rangle$) = effacer(A, q(A, SC))

où la fonction "q" est définie par:

```
q(s,t) = si vide?(s) alors 0
         sinon si vide?(t) alors minimum(s)
              sinon si premier(t) divise premier(s)
                  alors q(décapiter(s),t)
              sinon minimum(s)
         fsi
         fsi
```

L'algorithme est donc constitué d'un ensemble de processus "PREMi", $i > 1$. Chaque processus "PREMi" détermine le ième

nombre premier:

```

processus eratos ::
  port (ci: crible; i>1);
  processus gen_entiers ::
    sortie p: entier vers c2;
    corps
      p:=2;
      *[ produire(p) ; p:=p+1 ]
  fin gen_entiers;

pour tout i>1:
  processus PREMi ::
    entrée p: entier via ci;
    sortie p: entier vers ci+1;
    corps
      consommer(p) ;
      -- p est le ième nombre premier --
      *[ consommer(p) ; produire(p) ]
  fin PREMi;

composition
  gen_entiers (// PREMi; i>1)

fin eratos;

```

On constate alors que ce point de vue consiste finalement à "réduire" le problème à la définition du type de communication "crible". Les processus eux-mêmes étant triviaux, limités à un rôle de "tampon".

Une représentation du type de communication par un processus communicant par rendez-vous, puis sa fusion avec les processus "PREMi" (dans une démarche analogue à celle présentée au chapitre 7) conduit à une solution extrême par rapport au point de vue initial: le système est constitué de processus "CRIBLEi", réalisant à la fois la fonction de crible et de mémorisation des suites. Les communications sont exprimées par des rendez-vous.

Une troisième voie consiste à regrouper les fonctions de mémorisation et de communication, sous la forme d'une réalisation asynchrone du type de communication "égalité". C'est ce point de vue qui est adopté dans (Cer_85).

Problème b)-

Il s'agit en fait de définir un langage permettant de spécifier les résultats d'un problème, les propriétés d'un système, etc. -en un mot, le "quoi"- plutôt que de décrire une solution -c'est-à-dire le "comment"- . Etant entendu que ce langage doit avoir au moins les deux caractéristiques suivantes:

- être adapté à l'expression des ces spécifications,
- permettre une dérivation systématique, ou pour le moins justifiée, de solutions.

L'utilisation des suites temporelles, ou d'un formalisme analogue, est une réponse partielle entrevue dans cette thèse. Elle est un cadre pour l'identification de relations entre des suites de données. La spécification, dans ce type de formalisme, des propriétés de ces relations doit être une seconde étape; propriétés à partir desquelles on doit déduire une définition constructive -sous forme algébrique- des opérations de communication et des processus de calcul.

Pour ce qui concerne la partie "communication", plusieurs voies sont possibles pour aborder ce travail:

- la définition de primitives évoluées pour la manipulation de suites, permettant de caractériser une relation de communication (cf. (FJP_83)),
- une expression de prédicats caractérisant un type de communication: un exemple est traité dans ce sens dans (JuP_85),
- une spécification des événements et des relations de causalité, du type de celle présentée dans (ChY_83), et rappelée ci-dessus en 7-1-1- pour le type "égalité",
- la spécification des "transferts de connaissance" (cf. (HaM_84), (FHV_84)) caractérisant une relation de

communication. Dans ce type d'approche, on considère qu'un système (notamment un système réparti) est caractérisé par l'état de ses connaissances, c'est-à-dire l'ensemble des propriétés qu'on peut prouver. Chaque composante du système est caractérisée par une connaissance partielle (propriétés locales des processus, par exemple); l'ensemble constitue la "connaissance implicite" du système.

Toute opération de communication a pour effet de modifier cet état, par exemple de telle façon qu'une propriété connue d'un processus soit connue de l'ensemble du système:

- transmission de valeurs,
- propriétés globales déduites de propriétés locales: blocage, ...
- etc.

Associée à l'utilisation d'opérateurs de la logique temporelle, la définition de ces changements d'état de la connaissance permet de spécifier des propriétés caractérisant les relations de communication.

Par exemple, la transmission, sans perte de message, entre un processus "i" et un processus "j" s'écrirait:

$$K_i p \Rightarrow \Diamond K_j p$$

où $K_i p$ signifie: le processus "i" connaît la propriété "p" définissant un message.

Ce type d'écriture serait particulièrement adapté aux relations de communication "optionnelles" (égalité-presque-partout, rafraichie-avec-trous, etc.) qui utilisent la connaissance implicite du problème.

BIBLIOGRAPHIE-

(Abr_78) J.R.ABRIAL

Z: a specification language.
Proc. int. conf. on Mathematical Studies of Information Processing, 1978.

(Abr_79) K.A. AHAMSON

Modal logic of concurrent non deterministic programs.
Lect. Notes in Computer Sc., vol. 70, (1979), pp 21-33.

(Abs_79) J.R.ABRIAL, S.A.SCHUMAN

Non deterministic system specification.
Lect. Notes in Computer Sc., vol.70, (1979), pp 34-50.

(AFR_80) K.R.APT, N.FRANCEZ, W.P.DE ROEVER

A proof system for Communicating Sequential Processes.
TOPLAS, ACM, vol. 2, n.3, (1980), pp 359-385.

(AGP_77) ARVIND, K.P.GOSTELOW, W.PLOUFFE

Indeterminacy, monitors and data flow.
Proc. 6th ACM Conf. on Operating Systems Principles, 1977.

(AmA_77) AMBER et ALL

GYPSY: a language for specification and implementation of verifiable programs.
SIGPLAN Notices, vol. 3, 1977.

(And_81) G.R.ANDREWS

Synchronizing Resources.
ACM, TOPLAS, vol. 3, n. 4, 1981.

(Apt_81) K.R.APT

Recursive assertions and parallel programs.
Acta Informatica, vol. 15, (1981), pp 219-232.

(Apt_83) K.R.APT

Formal justification of a proof system for Communicating Sequential Processes.
Journal of the ACM, vol. 30, n.1 (1983), pp 197-216.

(Apt_84) K.R.APT

Ten years of Hoare logic: a survey. Part 2: non determinism.
ICS, n. 28 (1984), pp 83-109.

(Arn_79) A.ARNOLD

Sémantique des processus communicants.

- Rapport de recherche n.1, Univ. Poitiers, 1979.
- (AsW_76) E.A.ASHCROFT, W.N.WADGE
LUCID: a formal system for writing and proving programs.
SIAM journal on computing, 1976.
- (AuB_83) D.AUSTRY, G.BOUUDOL
Algèbre de processus et synchronisation.
Rapport de recherche INRIA, n.187, 1983.
- (Bau_76) G.BAUDET
Asynchronous iterative methods for multiprocessors.
Research Report, CMU, 1976.
- (Bau_78) G.BAUDET
The design and analysis of algorithms for asynchronous multiprocessors.
PHD CMU, 1978.
- (BaZ_82) J.W.DE BAKKER, J.I.ZUCKER
Denotational semantics of concurrency.
Proc. 14th ACM Symp. on Theory of Computing, (1982), pp 153-158.
- (BCH_95) J.L.BERGERAND, P.CASPI, N.HALBWACHS et ALL
Outline of a real time data flow language.
Rapport de recherche n.520, IMAG, Univ. Grenoble, 1985.
- (BeG_81) P.A.BERNSTEIN, N.GOODMAN
Concurrency control in distributed database systems.
Computing Surveys, vol. 13, n.2, (1981), pp 185-221.
- (Ber_78) G.BERTHELOT
Vérification de réseaux de Petri.
Thèse de 3ème cycle, Univ. Paris VI, 1978.
- (Ber_79) B.BERTHOMIEU
Analyse structurelle des réseaux de Petri: méthodes et outils.
Thèse Doct. Ing., Univ. Toulouse, 1979.
- (Ber_79') D.BERT
La programmation générique: construction de logiciel, spécification et vérification.
Thèse d'Etat, Univ. Grenoble, 1979.
- (Ber_82) B.BERTHOMIEU
Techniques algébriques pour la spécification et la vérification de protocoles de communication.
Actes Colloque Génie Logiciel, Paris, (1982), pp. 345-354.

- (Ber_85) B.BERTHOMIEU
Le langage et l'interprète LCS: une implantation de CCS.
Colloque C3, Angoulême, 1985.
- (Bou_85) G.BOUUDOL
Le calcul MEIJE.
Parallélisme, communication et synchronisation, Ed. CNRS, 1985.
- (BPW_79) F.L.BAUER, P.PEPPER, H.WOSSNER
Notes on the project CIP: outline of a transformation system.
Lect. Notes in Computer Sc., vol. 69, 1979.
- (BrH_72) P.BRINCH HANSEN
Operating system principles.
Prentice Hall Ed., 1973.
- (BrW_80) M.BROY, M.WIRSING
Programming languages as abstract data types.
Proc. 5th CAAP, 1980.
- (BSW_79) P.A.BERNSTEIN, D.W.SHIPMAN, W.S.WONG
Formal aspects of serializability in database concurrency control.
IEEE Trans. on Software Engineering, vol. 5, n.3, (1979), pp 203-216.
- (CaH_74) R.H.CAMPBELL, A.N.HABERMAN
The specification of process synchronizations by path expressions.
Lect. Notes in Computer Sc. n.16, (1974).
- (Cer_85) P.LE CERTEN
LC3, un langage de programmation parallèle.
Rapport de recherche IRISA, 1985.
- (ChM_69) D.CHAZAN, M.L.MIRANKER
Chaotic relaxation.
Linear Algebra and its applications, vol. 2, (1969), pp 199-222.
- (ChM_82) K.M.CHANDY, J.MISRA
Distributed computations on graphs: shortest path algorithms.
Com. ACM, vol. 25, n.11, (1982), pp 833-837.
- (ChY_83) B.S.CHEN, R.T.YEH
Formal specification and verification of distributed systems.
IEEE, Trans. on Software Engineering, vol.9, n.6, (1983), pp 710-722.
- (ClE_82) E.M.CLARKE, E.A.EMERSON
Design and synthesis of synchronization skeletons using branching time temporal

- logic.
Lect. Notes in Computer Sc., vol. 131,
(1982), pp 52-71.
- (Cou_78) P.COUSOT
Méthodes itératives de construction et
d'approximation de points fixes
d'opérateurs monotones sur un treillis;
analyse sémantique des programmes.
Thèse d'Etat, Univ. Grenoble, (1978).
- (Den_75) J.B.DENNIS
First version of a data-flow procedure
language.
Lect. Notes in Computer Sc., vol. 19,
(1975), pp 362-376.
- (DeZ_83) J.C.DERNIAME, A.ZAKARI
Spécification d'application de conduite
d'un atelier flexible.
Convention Informatique Latine,
Barcelone, 1983.
- (Dij_72) E.W.DIJKSTRA
Co-operating sequential processes.
Programming languages, NY Academic Press,
(1978), pp 43-112.
- (Dij_78) E.W.DIJKSTRA
Finding the correctness proof of a
concurrent program.
Lect. Notes in Computer Sc., n.64,
(1978), pp 31-38.
- (DLM_78) E.W.DIJKSTRA, L.LAMPORT, A.J.MARTIN et ALL
On-the-fly garbage collection: an
exercise in cooperation.
Com. of the ACM, vol.21, n.11, (1978),
pp. 966-975.
- (DuB_82) M.DUBOIS, F.A.BRIGGS
Performance of synchronized iterative
processes in multiprocessor systems.
IEEE, Trans. on Software Engineering,
SE-8, 4 (1982), pp 419-431.
- (ElF_80) T.ELRAD, N.FRANCEZ
A weakest precondition semantics for
communicating processes.
Lect. Notes in Computer Sc., n.137, 1981.
- (ElF_82) T.ELRAD, N.FRANCEZ
Decomposition of distributed programs
into communication-closed layers.
Research Report RC 9760, IBM T.J.Watson
Research Center, 1982.
- (EGL_76) K.P.ESWARAN, J.N.GRAY, R.A.LORIE, I.L.TRAIGER
The notions of consistency and predicate
locks in a database system.

- com. ACM, vol. 19, n.11, (1976), pp
624-633.
- (ErS_83) O.ERIKSEN, J.STAUNSTRUP
Concurrent algorithms for root searching.
Acta Informatica, vol. 18, (1983), pp
361-376.
- (Fel_79) J.A.FELDMAN
High level programming language for
distributed programming.
Com. ACM, vol. 22, n.6, 1979.
- (FHV_84) R.FAGIN, J.Y.HALPERN, M.Y.VARDI
A model theoretic analysis of knowledge:
preliminary report.
dans "Logiques et modèles pour la
vérification et la spécification des
systèmes concurrents", séminaire INRIA,
La Colle-sur-Loup, 1984.
- (FiJ_80) J.P.FINANCE, J.JARAY
Towards a methodology to specify and
construct concurrent programs.
Rapport de recherche CRIN, 1980.
- (Fin_79) J.P.FINANCE
Etude de la construction des programmes:
méthodes et langages de spécification et
de résolution de problèmes.
Thèse d'Etat, Univ. Nancy, 1979.
- (Fin_83) J.P.FINANCE
Un essai de construction méthodique
d'algorithmes parallèles.
Rapport de recherche CRIN, 83-R-094,
1983.
- (Fio_83) J.P.FINANCE, M.S.OVERGHI
On the algebraic specification of
concurrency and communication.
Proc. ICPP'83, 1983.
- (FJJ_84) J.P.FINANCE, J.JARAY, J.JULLIAND, M.MARMONIER,
G.R.PERRIN
How to construct parallel programs: an
example.
Rapport de recherche CRIN, 84-R-065,
1984.
- (FJP_83) J.P.FINANCE, J.JULLIAND, G.R.PERRIN
Conception et expression de types de
communication entre processus.
Rapport de recherche CRIN, 83-R-069,
1983.
- (FlH_76) L.FLON, A.N.HABERMAN
Towards the construction of verifiable
software systems.
Proc. conf. on Data, SIGPLAN Notices,

- (Gau_80) M.C. GAUDEL vol. 2, (1976), pp 135-148.
Génération et preuve de compilateurs basées sur une sémantique formelle des langages de programmation. Thèse d'Etat, Inst. Nat. Polytec. de Lorraine, 1980.
- (Ger_82) R. GERTH A sound and complete Hoare axiomatization of the ADA-rendezvous. Research Report, RUU-CS-82-5, Utrecht Univ., 1982.
- (Ger_84) R. GERTH, W.P. DE ROEVER A proof system for concurrent ADA programs. Science of Computer Programming, vol. 4, n.2 (1984), pp 159-204.
- (Ger_84') R. GERTH, W.P. DE ROEVER Essentials of a proof system for the concurrency fragment of ADA. Research Report, Univ. of Utrecht, in Proc. Advanced NATO study institute on logics and models for verification and specification of concurrent systems, La Colle-sur-Loup (1984), pp 80-108.
- (GoK_79) S.J. GOLDSACK, J. KRAMER The use of invariants in the application-oriented specification of real time control systems. Research Report, Imperial College of Sc. and Techn., London, 1979.
- (GTW_77) J.A. GOGUEN, J.W. TATCHER, E.G. WAGNER, J.B. WRIGHT Initial algebra semantics and continuous algebras. Jour. ACM, vol. 1, (1977), pp 68-95.
- (GuH_78) J.V. GUTTAG, J.J. HORNING The algebraic specification of abstract data types. Acta Informatica, vol 10, (1978), pp 27-52.
- (GuH_80) J.V. GUTTAG, J.J. HORNING Formal specification as a design tool. ACM conf. on Principles of Programming Languages, (1980), pp 251-261.
- (Gui_82) G.F. GUIDACCI DA SILVEIRA Sur la preuve des systèmes parallèles et distribués par la logique temporelle. Thèse d'état, Univ. Toulouse, 1982.
- (HaM_84) J.Y. HALPERN, Y. MOSES Knowledge and common knowledge in a

- distributed environment. Proc. of the 3rd ACM conf. on Principles of Distributed Computing, 1984.
- (HaO_83) B. HAILPERN, S. OWICKI Modular verification of computer communication protocols. IEEE, Trans. on Communication, vol. 31, n.1, (1983), pp 56-68.
- (Har_79) D. HAREL First-order dynamic logic. Lect. Notes in Computer Sc., n.68, 1979.
- (Hoe_69) C.A.R. HOARE An axiomatic basis for computer programming. Com. ACM, vol. 12 (1969), pp 576-580.
- (Hoe_74) C.A.R. HOARE Monitor: an operating system structuring concept. Com. ACM, vol. 17, n. 12, (1974), pp 549-557.
- (Hoe_78) C.A.R. HOARE Communicating Sequential Processes. Com. ACM, vol. 21, n. 8, 1978.
- (IcA_79) J.D. ICHBIAH et AL Preliminary ADA reference manual and Rationale for the design of the ADA programming language. SIGPLAN Notices, vol. 14, n.16, part A and B, 1979.
- (Jac_75) M.A. JACKSON Principles of program design. Academic Press, 1975.
- (Jan_81) R. JANICKI On the design of concurrent systems. Proc. IEEE Int. Conf. on Concurrent Programming, (1981), pp 455-466.
- (Jor_82) P. JORRAND Specification of communicating processes. Lect. Notes in Computer Sc., n.137, (1982), pp 242-255.
- (JPS_80) J. JULLIAND, G.R. PERRIN, P. SPITERI Simulations d'exécutions parallèles d'algorithmes de relaxation asynchrone. Rapport de recherche, ERA n. 654, Univ. Besançon, 1980.
- (JPS_81) J. JULLIAND, G.R. PERRIN, P. SPITERI Simulation de types de communication appliqués à des algorithmes numériques. Rapport de recherche, ERA n.906, Univ. Besançon, 1981.

- (Jul_81) J.JULLIAND
Expression des communications entre processus d'un programme parallèle par des types abstraits.
Thèse 3ème cycle, Univ. Besançon, 1981.
- (Jul_83) J.JULLIAND
Spécification algébrique de la communication entre processus parallèles. TSI, vol. 2, n. 4, (1983), pp 257-269.
- (Jul_85) J.JULLIAND
COMEDIE: manuel d'utilisation. Rapport CRIN, 85-R-075, 1985.
- (JuP_81) J.JULLIAND, G.R.PERRIN
Design and development of concurrent programs. Proc. CONPAR'81 in Lect. Notes in Comp. Sc. n.111 (Springer Verlag), (1981), pp. 464-479.
- (JuP_85) J.JULLIAND, G.R.PERRIN
Towards a methodology for concurrent programming. Proc. Parallel Computing 85, Berlin (Elsevier Sc. Pub.), 1985.
- (Kel_76) R.KELLER
Formal verification of parallel programs. Com. ACM vol. 19, n.7, (1976), pp 371-384.
- (KeM_80) R.M.McKEAG, P.MILLIGAN
An experiment in parallel program design. Software: Practice and Experience, vol. 10, (1980), pp 687-696.
- (KeS_75) K.KENNEDY, J.SCHWARTZ
An introduction to the set theoretic language SETL. Computers and Mathematics with Applications, vol. 1, n.97, 1975.
- (KMS_82) J.KRAMER, J.MAGEE, M.SLOMAN
A software architecture for distributed computer control systems. IFAC Symp. on Theory and applications of digital control, 1982.
- (KrC_78) J.KRAMER, R.J.CUNNINGHAM
Towards a notation for the functional design of distributed processing systems. Proc. IEEE Int. Conf. on Parallel Processing, (1978), pp 69-76.
- (Kun_76) H.T.KUNG
Synchronized and asynchronous parallel algorithms for multiprocessors. Algorithms and complexity: new directions

- and recent results, J. Traub Ed., Academic Press, 1976.
- (Kun_79) H.T.KUNG
The structure of parallel algorithms. Advances in Computers, vol. 19, Academic Press, 1976.
- (Lam_80) L.LAMPORT
The "Hoare Logic" of concurrent programs. Acta Informatica, vol. 14, (1980), pp 21-37.
- (Lam_83) L.LAMPORT
What good is temporal logic? Proc. IFIP congress, Information Processing, (1983), pp 657-668.
- (Lam_83') L.LAMPORT
Specifying concurrent program modules. ACM, TOPLAS, vol. 5, n.2, (1983), pp 190-222.
- (LaS_78) P.E.LAUER, M.W.SHIELDS
Abstract specification of resource accessing disciplines: adequacy, starvation, priority and interrupts. SIGPLAN Notices, vol. 13, n.12, 1978.
- (LaS_82) L.LAMPORT, F.B.SCHNEIDER
The "Hoare Logic" of CSP, and all that. Techn. Report 82-490, Cornell Univ., 1982.
- (LeG_81) G.M.LEVIN, D.GRIES
A proof technique for Communicating Sequential Processes. Acta Informatica, vol. 15, (1981), pp 281-302.
- (LeH_82) C.LENGAUER, E.C.R.HEHNER
A methodology for programming with concurrency: an informal presentation. Science of Computer Programming, vol. 2, (1982), pp 1-18.
- (Len_82) C.LENGAUER
A methodology for programming with concurrency: the formalism. Science of Computer Programming, vol. 2, (1982), pp 19-52.
- (Lev_82) D.LE VERRAND
Le langage ADA, manuel d'évaluation. Dunod Ed, 1982.
- (LiA_81) B.LINDT, T.AGERWALA
Communication issues in the design and analysis of parallel algorithms. IEEE, Trans. on Software Engineering, SE-7, 2 (1981), pp 174-188.

- (LiZ_75) B.LISKOV, S.ZILLES
Specification techniques for data abstractions.
Proc. Int. Conf. on Reliable Software, (1975), pp 72-87.
- (LTS_79) P.E.LAUER, P.R.TORRIGIANI, M.W.SHIELDS
COSY: A system specification language based on paths and processes.
Acta Informatica, vol. 12, 1979.
- (LyF_81) N.A.LYNCH, M.J.FISCHER
On describing the behavior and implementation of distributed systems.
TCS, vol. 13, (1981), pp 17-43.
- (Man_81) Z.MANNA
Verification of sequential programs: temporal axiomatization.
Theoretical Foundations of Programming Methodology, NATO Scientific Series, 1981.
- (MaP_81) Z.MANNA, A.PNUELI
Verification of concurrent programs: temporal proof principles.
Research Report CS81-23, Stanford Univ., 1981.
- (MaP_81') Z.MANNA, A.PNUELI
Verification of concurrent programs: the temporal framework.
The correctness problem in computer science, Int. Lect. Series in Computer Sc., Academic Press, 1981.
- (Mar_81) A.J.MARTIN
An axiomatic definition of synchronization primitives.
Acta Informatica, 16, (1981), pp 219-235.
- (MaS_83) S.MAHADEVAN, R.K.SHYAMASUNDAR
Correctness preserving transformations for distributed programs.
Proc. IFIP Congress Information Processing, (1983), pp 307-313.
- (MaW_84) Z.MANNA, P.L.WOLPER
Synthesis of communicating processes from temporal logic specifications.
ACM, TOPLAS, vol. 6, n.1, (1984), pp 68-93.
- (MiC_81) J.MISRA, K.M.CHANDY
Proofs of networks of processes.
IEEE, Trans. on Software Engineering, vol. 7, n.4, (1981), pp 417-426.
- (Mie_75) J.C.MIELLOU
Algorithmes de relaxation chaotique à

- retard.
RAIRO Analyse Numérique, 9ème année, R1, (1975), pp 55-82.
- (Mie_75') J.C.MIELLOU
A mixt relaxation algorithm applied to quasi-variational inequations.
Col. IFIP Optimisation, Math. Lect. Notes, 1975.
- (Mie_82) J.C.MIELLOU
Variantes synchrones et asynchrones de la méthode alternée de Schwarz.
Rapport contrat STPA, Univ. Besançon, 1982.
- (Mil_80) R.MILNER
A Calculus for Communicating Systems.
Lect. Notes in Computer Sc., n.92, Springer-Verlag, 1980.
- (Mil_80') R.MILNER
On relating synchrony and asynchrony.
Research report, Edinburgh Univ., 1980.
- (Mil_83) R.MILNER
Calculi for synchrony and asynchrony.
TCS vol 25, 1983.
- (Mon_85) C.MONGENET
Une méthode de conception d'algorithmes systoliques: résultats théoriques et réalisation.
Thèse, INPL, 1985.
- (MPS_84) J.C.MIELLOU, G.R.PERRIN, P.SPITERI
A cheap method of performance evaluation for subdomain decomposition parallel algorithms of three-dimentional elliptic problems.
Proc. Inter. Workshop on Modeling and Performance evaluation of parallel systems, Grenoble (Elsevier Sc. Pub.), 1984.
- (Nak_84) M.NAKECHBANDI
Algorithmes parallèles dans les problèmes combinatoires.
Thèse d'Etat, Univ. Besançon, 1984.
- (NiA_82) M.NIVAT, A.ARNOLD
Comportements de processus.
Rapport de recherche 82-12, LIIP, 1982.
- (Niv_80) M.NIVAT
Systèmes de transition permanents et équitables.
Congrès AFCET- Informatique, Nancy, 1980.
- (OwG_76) S.OWICKI, D.GRIES
An axiomatic proof technique for parallel



- programs.
Acta Informatica, vol. 6, (1976), pp 319-340.
- (OwG_76') S.OWICKI, D.GRIES
Verifying properties of parallel programs: an axiomatic approach.
Com. ACM, vol. 19, n.5, 1976.
- (Owi_76) S.OWICKI
A consistent and complete deductive system for the verification of parallel programs.
Research Report, Cornell Univ., 1976.
- (Owl_82) S.OWICKI, L.LAMPORT
Proving liveness properties of concurrent programs.
ACM, TOPLAS, vol. 4, n. 3, (1982), pp 455-495.
- (Pai_82) C.PAIR
Abstract data types and algebraic semantics of programming languages.
TCS, n.1, (1982), pp 1-38.
- (Pap_79) C.H. PAPADIMITRIOU
The serializability of concurrent database updates.
Journal of ACM, vol. 26, n.4, (1979), pp 631-653.
- (PDI_83) G.R.PERRIN, J.C.DERNIAME, J.P.THOMESSE
Caractérisation de la communication pour la conception de systèmes répartis.
Proc. CIL'83, (1983), pp 171-186.
- (Per_81) G.R.PERRIN
Conception, écriture et vérification de programmes parallèles.
RAIRO Informatique, vol. 15, n.3, (1981), pp 235-253.
- (Per_82) G.R.PERRIN
Specification and verification of communications of processes.
Rapport de Recherche, CRIN, n. 82-R-020, 1982.
- (Per_82') G.R.PERRIN
De l'expression des synchronisations dans un système parallèle.
Rapport de recherche, contrat STPA n.81 95-51500, 1982.
- (Per_83) G.R.PERRIN
De l'expression des communications dans la conception d'un algorithme parallèle.
Proc. col. Calcul vectoriel et parallèle, Bulletin EDF, série C, n.1, (1983), pp

- 109-116.
- (Per_83') G.R.PERRIN
Formal model of parallel programs.
Proc. Parallel Computing 83, Berlin (Elsevier Sc. Pub.), (1984), pp 469-474.
- (Per_83'') G.R.PERRIN
Sémantique d'un système parallèle.
Rapport de recherche CRIN, 83-R-024, 1983.
- (Per_85) G.R.PERRIN
La spécification des communications: un outil pour la conception méthodique de programmes parallèles.
Cours INRIA "Calcul parallèle à usage scientifique", St Cyprien, 1985.
- (Pet_81) J.L.PETERSON
Petri net theory and the modeling of systems.
Prentice-Hall Ed., 1981.
- (Pet_81') A.PETTOROSI
An approach to communications and parallelism in applicative languages.
Lect. Notes in Computer Sc., n.107, (1981), pp 432-446.
- (Pet_83) G.L.PETERSON
Concurrent reading while writing.
TOPLAS, ACM, vol. 5, n.1, (1983), pp 46-55.
- (Plo_81) G.D.PLOTKIN
An operational semantics for CSP.
Research Report, Edinburgh Univ., 1981.
également dans
- (Plo_82) G.L.PLOTKIN
Proc. IFIP-TC2, (1982), pp 185-208.
- (Pnu_77) A.PNUELI
The temporal logic of programs.
Proc. 19th an. Symp. on Foundations of Computer Science, 1977.
- (Pnu_81) A.PNUELI
The temporal semantics of concurrent programs.
TCS, vol. 13, (1981), pp 45-60.
- (Que_82) J.P.QUEILLE
Le système CESAR: description, spécification et analyse des applications réparties.
Thèse Doct. Ing., Univ. Grenoble, 1982.
- (RaK_83) K.RAMAMRITHAM, R.M.KELLER
Specification of synchronizing processes.
IEEE, Trans. on Software Engineering,

- (Ray_81) M.RAYNAL
vol. 9, n.6, (1983), pp 722-733.
Contribution à l'étude de la coopération entre processus dans les langages et les systèmes informatiques.
Thèse d'Etat, Univ. Rennes, 1981.
- (Ray_82) M.RAYNAL
Une analyse de la spécification de la coopération entre processus par variables partagées.
TSI, vol. 1, n.3, (1982), pp 201-210.
- (Rou_78) G.ROUCAIROL
Mots de synchronisation.
RAIRO Informatique, vol. 12, n.4, (1978), pp 277-290.
- (Rou_78') G.ROUCAIROL
Contribution à l'étude des équivalences syntaxiques et transformations de programmes parallèles.
Thèse d'Etat, Univ. Paris VI, 1978.
- (RoV_77) P.ROBERT, J.P.VERJUS
Towards autonomous descriptions of synchronization modules.
Proc. IFIP congress, Information Processing, (1977), pp 981-986.
- (SaS_81) Y.SAAD, A.SAMEH
Iterative methods for the solution of elliptic difference equations on multiprocessors.
Proc. CONPAR'81, Lect. Notes in Computer Sc., vol.111, (1981), pp 395-411.
- (Sch_79) J.SCHWARZ
Denotational semantics of parallelism.
Lect. Notes in Computer Sc., n.70, (1979), pp 191-202.
- (Sif_79) J.SIFAKIS
Le contrôle des systèmes asynchrones: concepts, propriétés, analyse statique.
Thèse d'Etat, Univ. Grenoble, 1979.
- (SiK_80) A.SILBERSCHATZ, Z.KEDEM
Consistency in hierarchical database systems.
Journal of ACM, vol. 27, n.1, (1980), pp 72-80.
- (Sin_79) M.SINTZOFF
Principles For distributing programs.
Lect. Notes in Computer Sc., n.70, (1979), pp 337-347.
- (Spi_84) P.SPITERI
Contribution à l'étude des grands

- systèmes non linéaires.
Thèse d'Etat, Univ. Besançon, 1984.
- (Sta_82) A.STAPHYLOPATIS
Performance considerations in the parallel execution of numerical algorithms on two processors.
Acta Informatica, vol. 17, (1982), pp 311-325.
- (SyA_76) J.C.SYRE et ALL
Etude et spécifications d'une architecture de calculateurs à contrôle décentralisé, exploitant le concept d'assignation unique.
Rapport CERT-ONERA, Univ. Toulouse, 1976.
- (Tar_81) M.N.EL TARAZI
Contraction et ordre partiel pour l'étude d'algorithmes synchrones et asynchrones en analyse numérique.
Thèse d'Etat, Univ. Besançon, 1981.
- (Tho_80) J.P.THOMESSE
Sygare: une structuration pour la conception d'applications en temps réel et réparties.
Thèse d'Etat, INPL, 1980.
- (TSI_85) TSI
Numéro spécial "Réseaux de Petri".
TSI, vol.4, n.1, 1985.
- (Wol_82) P.L.WOLPER
Synthesis of communicating processes from temporal logic specifications.
Research report STAN-CS 82-925, Stanford Univ., 1982.
- (YaC_83) S.S.YAU, M.U.CAGLAYAN
Distributed software system design representation using modified Petri nets.
IEEE, Trans. on Software Engineering, vol. 9, n.6, (1983), pp 733-745.
- (Zak_84) A.ZAKARI
Flexi: langage de conception d'application de conduite de procédés industriels.
Thèse 3ème cycle, Univ. Nancy, 1984.

NOM DE L'ETUDIANT : PERRIN Guy-René

NATURE DE LA THESE : Doctorat d'Etat ès sciences

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le 26 SEP. 1985 2781

LE PRESIDENT DE L'UNIVERSITE DE NANCY I



RESUME

Le travail présenté dans cette thèse est relatif à la spécification, la construction et la vérification des systèmes parallèles. L'objectif est de proposer un cadre formel pour l'expression des dépendances entre les entités constituant un système parallèle, induisant un mode de raisonnement pour la construction de ces systèmes. Ce cadre peut être résumé ainsi :

- les entités manipulent des suites de données,
- leur dépendance est exprimée par des "relations de communication" entre ces suites,
- un "type de communication" est une structure abstraite définie algébriquement : les propriétés spécifiées par une relation de communication sont exprimées par des opérations sur un objet de ce type.

La spécification des relations de communication, leur expression sous la forme d'un type abstrait algébrique, puis sa représentation dans un langage concret constituent les étapes d'une démarche méthodique de construction de systèmes parallèles. L'aspect modulaire et déductif de ces niveaux d'abstraction est comparé aux approches classiques, essentiellement fondées sur la notion de synchronisation d'opérations.

Enfin, ce formalisme est étudié d'un point de vue théorique sous deux aspects :

- la vérification des systèmes,
- les propriétés d'implantation des types de communication, dont on déduit quelques résultats concernant la transformation des programmes.

Mots-clés : communication, conception de programmes, parallélisme, représentation, suites temporelles, spécification, système de preuve, transformation, type algébrique.