Comparaison des comportements des processus communicants : application au langage FP2

Sylvie Rogé

THESE

présentée à

l' Institut National Polytechnique de Grenoble

pour obtenir le grade de

Docteur de l'INPG
(arrêté ministériel du 05/ 07 / 1984)

Spécialité: INFORMATIQUE

par

Sylvie ROGÉ

COMPARAISON des COMPORTEMENTS des PROCESSUS COMMUNICANTS.

APPLICATION au LANGAGE FP2

Thèse soutenue le 25 Novembre 1986 devant la commission d'examen

Messieurs	JP. VERJUS	Président
	J. SIFAKIS	Rapporteurs
	M. SINTZOFF	
	Ph. JORRAND	Examinateurs
	J. MOSSIERE	

préparée au sein du LIFIA

Laboratoire d'Informatique Fondamentale et d'Intelligence Artificielle

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président : Daniel BLOCH

Année 1987

Vice - Présidents : René CARRE
Jean-Marie PIERRARD

Professeurs des Universités

BARIBAUD Michel	ENSERG
BARRAUD Alain	ENSIEG
BAUDELET Bernard	ENSPG
BEAUFILS Jean-Pierre	ENSEEG
BESSON Jean	ENSEEG
BLIMAN Samuel	ENSERG
BLOCH Daniel	ENSPG
BOIS Philippe	ENSHMG
BONNETAIN Lucien	ENSEEG
BOUVARD Maurice	ENSHMG
BRISSONNEAU Pierre	ENSIEG
BRUNET Yves	IUFA
BUYLE-BODIN Maurice	ENSERG
CAILLERIE Denis	ENSHMG
CAVAIGNAC Jean-François	ENSPG
CHARTIER Germain	ENSPG
CHENEVIER Pierre	ENSERG
CHERADAME Hervé	UFR PGP
CHERUY Arlette	ENSIEG
CHIAVERINA Jean	UFR PGP
CHOVET Alain	ENSERG
COHEN Joseph	ENSERG
COUMES André	ENSERG
DARVE Félix	ENSHMG
DELLA-DORA Jean	ENSIMAG
DEPORTES Jacques	ENSPG
DOLMAZON Jean-Mar	ENSERG
DURAND Francis	ENSEEG
DURAND Jean-Louis	ENSIEG
FONLUPTJean	ENSIMAG
FOULARD Claude	ENSIEG
GANDINI Alessandro	UFR PGP
GAUBERT Claude	ENSPG
GENTIL Pierre	ENSERG
GREVEN Hélène	IUFA
GUERIN Bernard	ENSERG
GUYOT Pierre	ENSEEG
IVANES Marcel	ENSIEG
JAUSSAUD Pierre	ENSIEG
JOUBERT Pierre	ENSIEG
JOURDAIN Geneviève	ENSIEG
LACOUME Jean-Louis	ENSIEG
LESIEUR Marcel	ENSHMG
LESPINARD Georges	ENSHMG
LONGEQUEUE Jean-Pierre	ENSPG
LOUCHET François	ENSEEG
MASSE Philippe	ENSIEG
MASSELOT Christian	ENSIEG
MAZARE Guy	ENSIMAG
MOREAU René	ENSHMG
MORET Roger	ENSIEG
MOSSIERE Jacques	ENSIMAG
OBLED Charles	ENSHMG
OZIL Patrick	ENSEEG
PARIAUD Jean-Charles	ENSEEG
PAUTHENET René	ENSIEG
PERRET René	ENSIEG
PERRET Robert	ENSIEG
PIAU Jean-Michel	ENSHMG
POUPOT Christian	ENSERG
SAUCIER Gabrielle	ENSIMAG
SCHLENKER Claire	ENSPG
SCHLENKER Michel	ENSPG
SERMET PIERRE	ENSERG
SILVY Jacques	UFR PGP
SIRIEYS Pierre	ENSHMG
SOHM Jean-Claude	ENSEEG
SOLER Jean-Louis	ENSIMAG
SOUQUET Jean-Louis	ENSEEG
TROMPETTE Philippe	ENSHMG
VEILLON Gérard	ENSIMAG
ZADWORNY François	ENSERG

Professeur Université des Sciences Sociales (Grenoble II)

BOLLIET Louis

Personnes ayant obtenu le diplôme d'HABILITATION A DIRIGER DES RECHERCHES

BECKER Monique
BINDER Zdenek
CHASSERY Jean-Marc
COEY John
COLINET Catherine
COMMAULT Christian
CORNUEJOLS Gérard
DALARD Francis
DANES Florin
DEROO Daniel
DIARD Jean-Paul
DION Jean-Michel
DUGARD Luc
DURAND Robert
GALERIE Alain
GAUTHIER Jean-Paul
GENTIL Sylviane
PLA Fernand
GHIBAUDO Gérard
HAMAR Sylvaine
LADET Pierre
LATOMBE Claudine
LE GORREC Bernard
MADAR Roland
MULLER Jean
NGUYEN TRONG Bernadette
TCHUENTE Maurice
VINCENT Henri

Chercheurs du C.N.R.S

Directeurs de recherche 1ère Classe

CAILLET Marcel
CARRE René
FRUCHART Robert
JORRAND Philippe
LANDAU Ioan
MARTIN

Directeurs de recherche 2ème Classe

ALEMANY Antoine
ALLIBERT Colette
ALLIBERT Michel
ANSARA Ibrahim
ARMAND Michel
BINDER Gilbert
BONNET Roland
BORNARD Guy
CALMET Jacques
DAVID René
DRIOLE Jean
ESCUDIER Pierre
EUSTATHOPOULOS Nicolas
JOUD Jean-Charles
KAMARINOS Georges
KLEITZ Michel
KOFMAN Walter
LEJEUNE Gérard
MERMET Jean
MUNIER Jacques
SENATEUR Jean-Pierre
SUERY Michel
TEDOSIU
WACK Bernard

Personnalités agrées à titre permanent à diriger des travaux de recherche (décision du conseil scientifique)

E.N.S.E.E.G

BERNARD Claude
CHATILLON Catherine
CHATILLON Christian
COULON Michel
DIARD Jean-Paul
FOSTER Panayotis
HAMMOU Abdelkader
MALMEJAC Yves
MARTIN GARIN Régina
SAINTFORT Paul
SARRAZIN Pierre
SIMON Jean-Paul
TOUZAIN Philippe
URBAIN Georges

E.N.S.E.R.G

BOREL Joseph
CHOVET Alain
DOLMAZON Jean-Marc
HERAULT Jeanny

E.N.S.I.E.G

DESCHIZEAUX Pierre
GLANGEAUD François
PERARD Jacques
REINISCH Raymond

E.N.S.H.G

BOIS Daniel
DARVE Félix
MICHEL Jean-Marie
ROWE Alain
VAUCLIN Michel

E.N.S.I.M.A.G

BERT Didier
COURTIN Jacques
COURTOIS Bernard
DELLA DORA Jean
FONLUPT Jean
SIFAKIS Joseph

E.F.P.G

CHARUEL Robert

C.E.N.G

CADET Jean
COEURE Philippe
DELHAYE Jean-Marc
DUPUY Michel
JOUVE Hubert
NICOLAU Yvan
NIFENECKER Hervé
PERROUD Paul
PEUZIN Jean-Claude
TAIB Maurice
VINCENDON Marc

Laboratoires extérieurs

C.N.E.T

DEMOULIN Eric
DEVINE
GERBER Roland
MERCKEL Gérard
PAULEAU Yves

ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur : Monsieur M.MERMET
Directeur des Etudes et de la formation: Monsieur J. LEVASSEUR
Directeur des recherches : Monsieur J. LEVY
Secrétaire Général : Mademoiselle M. CLERGUE

PROFESSEURS DE 1ère CATEGORIE

COINDE Alexandre	Gestion
GOUX Claude	Métallurgie
LEVY Jacques	Métallurgie
LOWYS Jean-Pierre	Physique
MATHON Albert	Gestion
RIEU Jean	Mécanique-Résistance des matériaux
POUSTELLE Michel	Chimie
ORMERY Philippe	Mathématiques Appliquées

PROFESSEURS DE 2ème CATEGORIE

HABIB Michel	Informatique
PERRIN Michel	Géologie
VERCHERY Georges	Matériaux
TOUCHARD Bernard	Physique Industrielle

DIRECTEUR DE RECHERCHE

LESBATS Pierre	Métallurgie

MAITRE DE RECHERCHE

BISCONDI Michel	Métallurgie
DAVOINE Philippe	Géologie
FOURDEUX Angeline	Métallurgie
KOBYLANSKI André	Métallurgie
LALAUZE René	Chimie
LANCELOT Francis	Chimie
LE COZE Jean	Métallurgie
THEVENOT François	Chimie
TRAN MINH Canh	Chimie

Personalités habilitées à diriger des travaux de recherche

DRIVER Julian	Métallurgie
GUILHOT Bernard	Chimie
THOMAS Gérard	Chimie

Professeurs à l'UER de Sciences de Saint-Etienne

VERGNAUD Jean-Maurice	Chimie des Matériaux et Chimie Industrielle

Je tiens à remercier

Monsieur Jean-Pierre Verjus, Professeur à l'université de Rennes I, Directeur du pôle C3 du CNRS, de m'avoir fait l'honneur de présider le jury,

Monsieur Joseph Sifakis, Directeur de recherche au CNRS, d'avoir accepté de juger mon travail et de m'avoir suggéré des modifications intéressantes,

Monsieur Michel Sintzoff, Professeur à l'université catholique de Louvain, d'avoir jugé mon travail et de s'être montré intéressé par ce document,

Monsieur Jacques Mossière, Professeur à l'INPG, Directeur du LGI, d'avoir accepté de participer au jury,

Monsieur Philippe Jorrand, Directeur de recherche au CNRS, Directeur du LIFIA, de m'avoir accueillie dans son équipe et d'avoir dirigé ma thèse: les discussions que j'ai eues avec lui ont été des orientations essentielles et la confiance qu'il m'a témoignée m'a permis de travailler librement. Qu'il soit assuré de ma profonde gratitude.

mes collègues: Annick Marty, Eqerem Arkaxhiu, Jean-Charles Marty, Jean-Michel Hufflen, Juan-Manuel Pereira, Maria-Antonia Mozota, Maria-Blanca Ibañez, Patrick Lagnier, Saddek Bensalem et plus particulièrement Philippe Schnoebelen pour sa contribution intéressante sur certaines questions, Amelia Soriano pour les nombreuses discussions que nous avons eues ensemble et Mary Cisneros pour son intérêt constant à mon travail et son amitié.

S.Rogé

TABLE des MATIERES

Introduction

La conception d'un système parallèle peut être abordée de plusieurs manières: dans certains cas, (conception de réseaux systoliques, conception de systèmes temps réel ou de réseaux distribués) lorsque les entités devant évoluer en parallèle correspondent à une réalité du problème la conception parallèle peut être la plus naturelle. Dans d'autre cas (décomposition parallèle des entités élémentaires du cas précédent) le parallélisme peut être utilisé à d'autres fins, pour obtenir des gains d'efficacité par exemple. Dans ce cas , une spécification non parallèle de la solution peut être donnée plus naturellement, le parallélisme intervenant ensuite dans la construction d'un réseau qui doit réaliser correctement la spécification.

Dans un cas comme dans l'autre, on cherchera à prouver que les solutions proposées ont des fonctionnements corrects. Dans le premier cas, on essaiera, en général, de vérifier des assertions sur le fonctionnement du système (abscence d'interblocage, accès inévitable à un état de bonne terminaison...) que l'on exprimera par exemple dans un formalisme de logique temporelle. Dans le second cas, on essaiera de montrer qu'il existe une relation d'équivalence, ou au moins de réalisation correcte entre la spécification et l'implémentation.

C'est ce deuxième type de problème, couvert par le terme de comparaison observationnelle qui est étudié dans cette thèse.

L'étude de cette question a grandement bénéficié des progrès obtenus depuis CSP [Hoa 78] dans le domaine des langages de programmation, concernant l'expression de la synchronisation. En effet, le mécanisme de synchronisation du rendez-vous défini primitivement dans CSP permet de suivre pas à pas l'évolution d'un processus, de connaître à chaque instant ses capacités de communication et de synchronisation, et par conséquent de détecter les situations de deadlock d'un réseau de processus.

Ce mécanisme de synchronisation est fondé sur la notion de communication synchrone (la communication entre deux processus ne peut avoir lieu que si les deux processus sont prêts à communiquer et aura lieu dans ce cas au bout d'un temps fini). Un processus ne peut donc être utilisé que par un utilisateur qui soit lui même un processus communiquant selon ce mécanisme.

La comparaison observationnelle se définit alors informellement en fonction de l'indistingabilité par un environnement quelconque des processus comparés, ou encore comme la non perturbation du fonctionnement d'un réseau lorsqu'on remplace un des éléments du réseau par un processus qui lui est équivalent. Les critères de distingabilité des processus sont le fruit du mécanisme de synchronisation lui-même: deux processus ne peuvent avoir d'information sur leurs états respectifs que par les messages qu'ils ont échangés; les choix réalisés de façon interne par les processus ne sont donc pris en compte que dans la mesure où ils peuvent entraîner des modifications de comportement perçus par les processus avec lesquels ils communiquent.

Les premières études menées sur ce thème ont conduit à la parution de CCS (Calculus of Communicating Systems)[Mil 80], et ont été poursuivies par d'autres qui ont en général des visions différentes du problème qu'elles soient fondés sur CCS ou sur d'autres modèles.

Le calcul CCS définit une algèbre de processus communicants dont la description peut se ramener à l'utilisation d'opérateurs somme et préfixage utilisés également dans la théorie des langages formels.

La première étape, effectuée par CCS, a consisté à renoncer aux équivalences proposées par ces théories. En effet, celles-ci ignorent le problème du non-déterminisme alors que celui-ci a des conséquences essentielles sur le fonctionnement d'un processus communicant puisque les seules communications qu'un processus peut réaliser sont définies par l'état dans lequel il se trouve effectivement, les états dans lesquels il aurait pû de la même façon se trouver étant de ce point de vue totalement ignorés. La démarche adoptée dans CCS est donc une remise en cause de ces théories, mais elle reste cependant très voisine dans la forme.

D'autres théories développées ultérieurement abordent le problème de façon résolument nouvelle, notamment en se rapprochant de la formulation intuitive initiale du problème [HN 84]. Le premier pas consiste à formaliser cette intuition: on définit des observateurs qui communiquent avec les processus et on compare les processus en fonction de la façon dont ils permettent aux observateurs d'atteindre des états définis comme des états de succès.

Ces théories, plus récentes, conduisent à des résultats originaux par rapport à ceux de CCS et permettent en général d'obtenir des résultats correspondant à la notion d'indistingabilité par observation. En général, elles sont supportées par des algèbres de processus du type de celle définie dans CCS.

L'étude que nous avons menée se situe dans le cadre d'un projet qui s'articule autour d'un langage de programmation parallèle et fonctionnelle: FP2 (Functional Parallel Programming) [Cis 84][Jor 85], projet qui intègre également d'autres études liées aux systèmes communicants: implémentation de ce langage sur machine monoprocesseur [Sch 85][Huf 86], ou sur machine parallèle [MarA 86], analyse des comportements [Per 84], environnement de programmation pour un langage parallèle [Ark 84][MarJC 86], conception d'une machine à inférence parallèle, description de circuits à l'aide du langage.

Pour comparer le comportement de processus décrits en FP2, la première possibilité aurait été de traduire FP2 dans l'une des algèbres de processus communicants et d'utiliser ensuite les résultats de l'une ou l'autre des théories évoquées en cherchant à définir des algorithmes efficaces de comparaison.

Cette approche a plusieurs inconvénients.

D'une part du fait du langage FP2 lui-même:

Ce langage présente un mécanisme de synchronisation plus puissant que ces algèbres, bien qu'il soit fondé sur le même principe. Il était nécessaire d'en étudier les spécificités.

Par ailleurs, la première étape pour implémenter les relations fondées sur des algèbres de processus consiste en général à construire des systèmes de transitions à partir des termes de l'algèbre. FP2 décrit les processus par des systèmes de transition, il semblait donc inutile de passer par cette étape.

Enfin, ces algèbres utilisent pour définir des comportements infinis des opérateurs de récursion qui soulèvent des problèmes liés à la mauvaise définition de processus, ceux-ci étant absents du langage FP2.

D'autre part du fait des résultats obtenus par ces théories:

La théorie développée par CCS conduit à des résultats qui nous apparaissent comme contestables parce qu'elle amène à prendre en compte des distinctions entre processus qui sont inobservables quel que soit l'observateur.

Les théories les plus proches du problème sont plus récentes et conduisent à des résultats plus satisfaisants. Toutefois, elles se distinguent en général dans la façon d'aborder certains problèmes limites comme celui de la divergence des processus (processus qui ont la possibilité d'effectuer des séquences infinies de transitions internes). Sur ce problème particulier, les solutions envisagées sont d'une part à l'opposé de celles proposées dans CCS et d'autre part elles ne sont pas totalement satisfaisantes puisqu'aucune ne permet de comparer dans tous les aspects de leur comportement des processus ayant eu la possibilité de diverger. Enfin elles apparaissent comme trop rigides, l'interprétation proposée dans CCS pouvant s'avérer utile dans certains cas, tout comme celle qu'elles proposent peut l'être dans d'autres cas.

Nous avons donc choisi d'étudier pleinement le problème à partir du langage FP2.

Les résultats que nous avons obtenus sont voisins de ceux obtenus dans [HN 84] et [BHR 84], les différences essentielles se situant dans la façon de traiter le problème de la divergence. En effet, l'approche que nous avons développée traite séparément le non déterminisme dû aux transitions internes et celui issu de la possibilité d'atteindre plusieurs états en réalisant un même événement, en exprimant le premier à l'aide du second. De ce fait, le problème de la divergence a pû être isolé et les deux interprétations proposées d'une part dans CCS et d'autre part dans [HN 84] ou [BHR 84] sont possibles. De plus une fois que le phénomène a été interprété d'une façon ou de l'autre, il n'apparaît plus en tant que tel et les comportements au delà des points de divergence sont comparables, de la même façon qu'à partir de n'importe quel étape de l'évolution des processus.

Par ailleurs, nous avons défini pour chacune des relations proposées des algorithmes qui permettent de les calculer, mais qui restent à implémenter.

Enfin, ce travail nous a permis d'envisager des extensions intéressantes, en offrant la possibilité de définir des environnements utilisateurs qui ne seraient plus quelconque mais qui peuvent être partiellement définis, un des cas limites étant celui où l'environnement utilisateur est défini par un seul processus. Dans ce cas, on ne s'intéresse qu'aux distinctions effectuables par ce processus, en ignorant totalement le fait que d'autres processus pourraient les distinguer.

La présentation du travail réalisé est organisée en quatre chapitres:

Dans le premier chapitre, les différents formalismes de description de processus seront présentés. Nous y consacrons une première partie à CSP qui a été à l'origine de concepts nouveaux concernant l'expression de la synchronisation et qui a conduit à la définition d'une algèbre (TCSP) sur laquelle reposent les travaux de [BHR 84]. Une autre partie présente l'algèbre CCS et différentes algèbres qui en ont été dérivées sur laquelle reposent les résultats de CCS mais aussi ceux de [HN 84]. La dernière partie est consacrée au langage FP2 qui a été notre support de travail. Bien que nous n'ayons utilisé qu'un sous ensemble de ce langage, celui des processus réguliers, nous essaierons de le présenter dans sa généralité puisque c'est un langage novateur se situant au confluent de deux sources de développement des langages de programmation: les langages fonctionnels et les langages parallèles, notre ambition à plus long terme étant par ailleurs de pouvoir étendre les résultats obtenus à des domaines de processus moins restreints.

Dans le second chapitre, nous présenterons les théories qui étudient la comparaison observationnelle, en distinguant d'une part celles qui abordent le problème à la manière de CCS, qui sont fondées sur un même outil, la bisimulation des systèmes des transitions, et d'autre part, celles qui ont une approche globale du comportement des processus, particulièrement la théorie de l'équivalence par tests[HN84][Hen 83] et TCSP[Bro 83][BHR84].

Le troisième chapitre présente les résultats que nous avons obtenus à partir de FP2 dans le cas d'un contexte quelconque. Nous y présentons d'abord une étude menée dans un modèle asynchrone, puis comme cela a été fait à partir de CCS[Mil 83][Hen 83], nous étudions les conséquences de l'introduction d'opérateurs synchrones. Enfin nous abordons l'étude des propriétés algébriques de l'ensemble des processus, induites par les relations définies.

Dans le quatrième chapitre nous introduisons la notion de contexte utilisateur. Ces contextes sont définis par des ensembles finis de processus et sont moins généraux que ceux qui pourraient être définis en utilisant le support de la théorie d'équivalences par tests, bien que ce problème n'y soit pas traité. Nous traitons en particulier le problème de la comparaison des pouvoirs discriminants de tels contextes. Nous présentons quelques exemples intéressants qui nous permettent de définir le contexte "environnement quelconque" implicitement utilisé dans le troisième chapitre, ou encore de définir une notion d'implémentation correcte moins stricte, liée à la définition de la spécification.

1. Langages et modèles de processus communicants

Ce premier chapitre est consacré à la présentation de langages et de modèles qui ont conduit à des études sur la sémantique des processus communicants. Cette présentation est organisée autour de trois modèles:

Le modèle CSP défini par Hoare dans [Hoa 78] dont l'objectif était d'utiliser les opérations d'Entrées / Sorties comme primitives de programmation et qui définissait pour la première fois le " rendez-vous " entre processus communicants, sous sa forme la plus élémentaire: le rendez-vous de deux processus.

Le modèle CCS défini par Milner dans [Mil 80]. Les processus sont décrits par des termes d'une algèbre et le mécanisme de synchronisation des processus est analogue à celui défini dans CSP. Ce calcul avait en outre pour objectif d'étudier le comportement observable des processus en définissant des relations de congruence entre termes.

Le langage FP2 défini dans [Jor 85]. A la différence des précédents, ce langage permet à la fois la programmation parallèle et l'analyse des comportements. En outre, il se distingue par un mécanisme de synchronisation de plus haut niveau, obéissant toutefois aux mêmes principes et que l'on peut définir comme le rendez-vous de n processus.

1.1 Les modèles dérivés de CSP

Cette première partie est consacrée à la présentation des langages et modèles fondés sur CSP. Dans la première section, nous présenterons les principes d'un langage parallèle tels qu'ils sont proposés dans [Hoa 78] ainsi que le langage OCCAM [May 83] directement construit à partir de ce modèle. Dans la seconde section, nous exposerons les modèles définis pour CSP, notamment ceux présentés dans [Hoa 81], [Hoa 85] et [BHR 84] .

1.1.1 Communicating Sequential Processes [Hoa 78]

L'originalité du modèle CSP proposé dans [Hoa 78] résidait dans l'introduction de primitives de communication comme structures de contrôle et de synchronisation d'un langage de programmation parallèle. Son apport théorique essentiel a été de montrer qu'en donnant une sémantique précise aux échanges de messages, ceux-ci pouvaient être utilisés pour exprimer à la fois la coopération et la concurrence des processus s'exécutant en parallèle.

Le langage est un langage impératif organisé autour de quatre commandes simples et de quatre commandes structurées. Les commandes simples sont :

* L'affectation d'une valeur à une variable

* Une commande de signal de fin correcte de processus notée SKIP ou √

* Deux commandes d'Entrée / Sortie . Syntaxiquement, elles s'écrivent :

`rec ! e` pour l'émission du résultat d'une expression e vers le processus récepteur rec.

`em ? x` pour la réception dans une variable x d'une valeur émise par le processus émetteur em.

La signification de ces commandes qui définit le mécanisme du rendez-vous est la suivante : La communication entre processus émetteur et processus récepteur est instantanée et synchrone. Elle ne peut avoir lieu que lorsque les deux processus sont prêts à communiquer, ce qui implique que le premier processus prêt à communiquer doit attendre que le second le soit aussi.

Les commandes structurées permettent d'exprimer le parallélisme, la séquentialité et le non-déterminisme des processus :

* La commande parallèle (notée P1||...|| Pn, où les Pi sont des processus) permet l'exécution asynchrone des processus donnés en paramètre. Les processus ne peuvent travailler que sur des variables locales et ne peuvent communiquer et se synchroniser que par émission / réception de messages. Le processus global défini par la commande parallèle commence en lançant tous les processus et n'est terminé que lorsque tous les processus ont terminé.

* La commande alternative(notée G1 [] G2 ... [] Gn, où les Gi sont des commandes gardées) permet le choix non déterministe et s'exprime dans le style des commandes gardées de Dijkstra [Dij 75]. Toutefois, les gardes ne s'y limitent pas à des expressions booléennes mais peuvent également contenir une commande de réception. L'exécution de la commande alternative globale correspond à l'exécution d'exactement une commande gardée qui est choisie arbitrairement parmi toutes celles dont la garde est exécutable (expression booléenne vraie et communication possible). Si aucune garde n'est exécutable, la commande globale échoue.

* La commande répétitive(notée * A, où A est une commande alternative) spécifie l'exécution de n fois ($n \geq 0$) la commande alternative qu'elle prend en paramètre. Elle se termine lorsqu'aucune garde n'est exécutable (si une garde contient une commande de réception, elle n'est plus exécutable si le processus émetteur a terminé).

* La commande séquentielle(notée C1 ;...; Cn où les Ci sont des commandes) spécifie l'exécution en séquence de plusieurs commandes, simples ou structurées.

Plusieurs remarques peuvent être faites au sujet de ces propositions dont certaines ont conduit à des redéfinitions dans les modèles ultérieurs:

La critique la plus fréquente [AS 83] vient du nommage explicite du correspondant, qui peut s'avérer d'un usage assez lourd dans le cas de programmes du type client serveur. Bien que cette critique soit discutable (on peut imaginer d'introduire des processus relais pour gérer l'ensemble du système), c'est celle qui a amené par la suite les principales redéfinitions et l'utilisation de canaux(cf § suivants).

En second lieu, il faut remarquer que le mécanisme proposé ne permet pas de réunir sur un même point de synchronisation plus de deux processus. Toute synchronisation de plus de deux processus doit être approchée par une succession de rendez-vous, ce qui exige de la part de chacun des processus la connaissance de l'ensemble du système. Il semble donc exclu de pouvoir programmer un réseau de processus n'évoluant que de façon synchrone (où chaque processus n'effectue un pas que si tous les autres processus font de même).

Il faut aussi remarquer l'utilisation de la commande SKIP comme outil de synchronisation en plus du mécanisme du rendez-vous dont l'objectif était d'unifier toutes les formes d'expression de la synchronisation. Cette commande est par sa nature contraire aux principes généraux de CSP. Elle suppose en effet que tout processus peut avoir accès à une information particulière (la bonne terminaison d'un autre processus) dont il ne dispose pas et qui ne lui parvient pas par échange de messages.

1.1.2 Les modèles de C.S.P

A la suite des critiques formulées à l'encontre de CSP, plusieurs travaux ont tenté d'améliorer les propositions qui y étaient faites, dans deux directions principalement : modélisation de CSP [Hoa 81] [BHR 84] [Hoa 85] afin d'avoir un support se prêtant mieux à l'analyse des comportements des processus et définition d'un langage de programmation OCCAM [May 83].

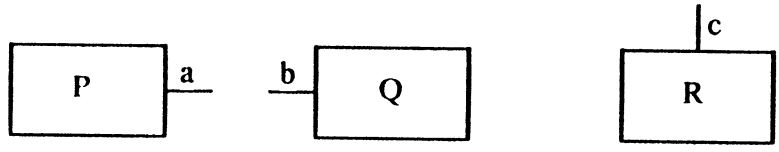
L'objectif des articles qui fournissent des modèles de CSP est plus de proposer des analyses d'ordre sémantique, en s'appuyant sur un formalisme algébrique dérivé, que de fournir des moyens d'expressions. On peut néanmoins noter dans tous ces modèles l'introduction de canaux ou d'événements partagés pour exprimer la synchronisation , au détriment du nommage explicite du correspondant.

Les canaux de communication sont introduits pour la première fois dans [Hoa 81]. L'apport principal est d'offrir un outil de synchronisation de plus de deux processus, par rendez-vous sur un même canal.

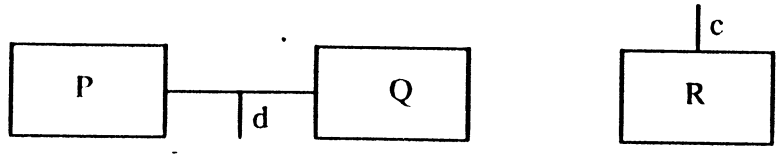
La communication est obtenue par identification de deux canaux appartenant à des processus différents et le rendez-vous à plusieurs est obtenu par nommage du nouveau canal, ce qui permet une liaison ultérieure du même type avec un processus supplémentaire.

Exemple

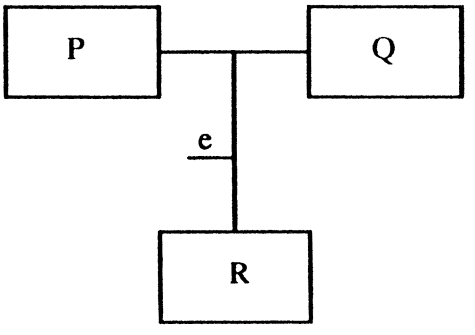
Soit un réseau R0 constitué par la composition parallèle de trois processus P, Q, R disposant des canaux a, b et c respectivement.



L'instruction `R1 = (chan d = ( a ↔ b ) in R0)` permet de définir le canal d comme le canal a lié au canal b.



L'opération peut être répétée entre le canal c du processus R et le canal d créé pour définir un nouveau canal e : `R2 = (chan e = ( c ↔ d )in R1.`



Fin-exemple

Le fonctionnement du canal détermine le rendez-vous de tous les processus, un message ne pouvant circuler sur le canal que si tous les processus concernés sont prêts à recevoir ou à émettre la valeur de ce message.

Il faut toutefois noter que la sémantique de la connexion de deux canaux n'est définie qu'entre deux canaux d'entrée (ce qui crée un canal d'entrée qui distribue la valeur reçue sur les deux canaux et qui n'est prêt que lorsque les deux sont prêts) ou entre un canal d'entrée et un canal de sortie (ce qui crée un canal de sortie qui émet la valeur émise par le canal de sortie et qui n'est prêt que lorsque les deux sont prêts) mais qu'elle n'est pas définie entre deux canaux de sortie (ceux-ci devraient en effet émettre la même valeur sur le canal créé). Ce problème a été soulevé par Hoare et nous nous contenterons ici de rapporter sa conclusion:

" ...There remains the case that both c and d are ready for output and the readiness of b depends on whether the values output are the same. This case is not very useful and should probably be excluded in a practical programming notation " ([Hoa 81] page 16)

Les autres modèles définis à partir de CSP [BHR84] et [Hoa 85] utilisent une autre approche en faisant dans un premier temps abstraction des notions de canal et de valeur transmise. Les processus y sont définis comme des termes construits sur un ensemble d'événements et un ensemble d'opérateurs, la synchronisation étant effectuée par partage d'événements.

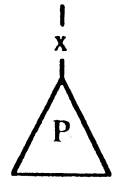
Chaque processus P est concerné par un ensemble d'événements A, son alphabet, qu'on note αP. On peut associer à chacun des opérateurs une représentation arborescente.

On notera 

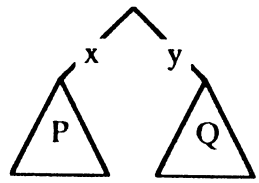
l'arbre de comportement d'un processus P

Les principaux opérateurs de description des processus sont les suivants:

Préfixage : $Q = x \rightarrow P$ où x est un événement et P un processus.
Q est un processus qui réalise initialement x et se comporte ensuite comme P.

$x \rightarrow P$: 

choix : $R = (x \rightarrow P) \mid (y \rightarrow Q)$
où x et y sont deux événements différents, P et Q deux processus.
R est un processus qui réalise initialement x ou y et se comporte ensuite comme P(si x est l'événement réalisé) ou comme Q (si y est le premier événement).

$(x \rightarrow P) \mid (y \rightarrow Q)$: 

Récursion : $X = F(X)$ ou $\mu X : \Lambda . F(X)$
où Λ est l'alphabet de X
Si F est une expression gardée(si la description est préfixée par un événement) X est le plus petit point fixe de la fonction F.

Plusieurs opérateurs sont définis entre les processus, notamment pour exprimer les choix non déterministes, la composition parallèle et la synchronisation.

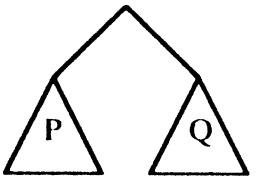
On peut citer par exemple

Le non déterminisme OR: $R = P \sqcap Q$

R est un processus qui peut se comporter indifféremment comme P ou comme Q, le choix du comportement qui sera celui de R est effectué de façon interne et est inaccessible à l'environnement. On peut rapprocher cet opérateur de l'opérateur choix défini précédemment. La différence tient à ce que pour cet opérateur la contrainte de préfixages différents n'est pas exigée:

$x \rightarrow (P \sqcap Q) = (x \rightarrow P) \sqcap (x \rightarrow Q)$

$P \sqcap Q$



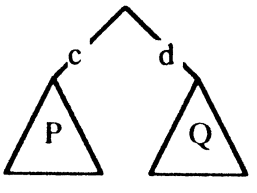
Le choix général : $R = P \; [] \; Q$

Cet opérateur est une combinaison des opérateurs "choix" et " non déterminisme OR". Le choix du comportement qui sera celui du processus R peut être déterminé par l'environnement si les processus sont préfixés différemment. Dans le cas où les processus sont préfixés par le même événement, le choix est effectué par le processus après occurence du premier événement. Cet opérateur est régi par la loi suivante:

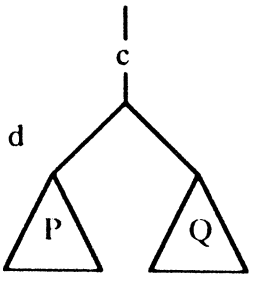
$(c \rightarrow P) [] (d \rightarrow Q) = (c \rightarrow P) | (d \rightarrow Q)$ si $c \neq d$

$(c \rightarrow P) [] (c \rightarrow Q) = c \rightarrow (P \sqcap Q)$

$(c \rightarrow P) [] (d \rightarrow Q)$ si $c \neq d$



si $c = d$



Le masquage: P \ C où P est un processus et C un ensemble d'événements.

Cet opérateur permet l'occurence de chacun des événements de C sans intervention de l'environnement. Il est défini ainsi:

$$(x \rightarrow P) \setminus C = x \rightarrow (P \setminus C) \quad \text{si } x \notin C$$

$$= P \setminus C \quad \text{si } x \in C$$

$(x \rightarrow P) \setminus C$: si $x \notin C$ 

si $x \in C$ 

Il faut remarquer que dans le cas où x est l'un des événements de C, le processus se réduit à P \ C et l'occurence d'un événement est oubliée, alors que dans la plupart des modèles ces événements sont notés par un symbole spécial, en général τ.

De ce fait, cette définition ne se suffit pas à elle même et la sémantique de cet opérateur doit être donnée en fonction des opérateurs à l'aide desquels le processus P est construit.

Soit R le processus $[(c \rightarrow P) | (d \rightarrow Q)] \setminus C$ où $P = (d \rightarrow P1) | (e \rightarrow P2)$, $C = \{c\}$

Seule l'occurence de c peut être automatique, mais l'occurence de d reste possible. Toutefois si l'événement d a lieu, ce peut être soit après occurence de c comme premier événement de P, soit comme alternative de c comme premier événement de R. Le développement du processus R à l'aide uniquement des opérateurs précédemment définis est le suivant:

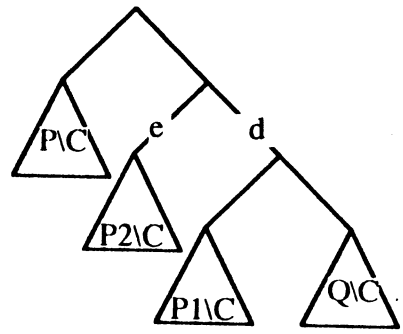
$$R = (P \setminus C) \lceil\rceil ((P \setminus C) ||| (d \rightarrow Q \setminus C))$$

$$= (P \setminus C) \lceil\rceil (d \rightarrow (P1 \setminus C \lceil\rceil Q \setminus C) | e \rightarrow P2 \setminus C)$$

Il faut ici remarquer que l'interprétation d'un événement interne qui est proposée se situe dans un modèle nécessairement asynchrone puisque la définition de l'opérateur de masquage ignore l'occurence de ces événements.

$((c \rightarrow P) \mid (d \rightarrow Q)) \setminus C$ où $C = \{ c \}$,

$P = (d \rightarrow P1) \mid (e \rightarrow P2)$



Synchronisation:

Dans ces modèles, la synchronisation est effectuée par partage de tous les événements, communs aux processus composés, grâce à l'opérateur d'interaction. Cet opérateur, noté || est défini ainsi dans [Hoa 85]:

(1) $(c \rightarrow P) \| (c \rightarrow Q) = c \rightarrow (P \| Q)$

(2) $c \neq d$; $c, d \in \alpha P \cap \alpha Q$: $(c \rightarrow P) \| (d \rightarrow Q) = STOP$

(3) $c \neq d$; $c \notin \alpha Q, d \in \alpha P$: $(c \rightarrow P) \| (d \rightarrow Q) = c \rightarrow (P \| d \rightarrow Q)$

(4) $c \neq d$; $c \notin \alpha Q, d \notin \alpha P$: $(c \rightarrow P) \| (d \rightarrow Q) = (c \rightarrow (P \| d \rightarrow Q)) \mid (d \rightarrow (c \rightarrow P \| Q))$

La synchronisation est effectuée en fonction des événements que les processus doivent simultanément réaliser:

(i) Si un événement ne concerne que l'un des processus alors il est initialement possible dans le processus résultat (3 et 4).

(ii) Si au contraire il concerne les deux processus alors les deux processus doivent être prêts à le réaliser pour qu'il puisse avoir lieu. De ce fait, il semble nécessaire de définir l'alphabet du processus STOP ($\alpha P \cup \alpha Q$), dans (2), de façon à ce qu'aucun des événements qui exigerait la participation de l'un des processus P ou Q ne puisse avoir lieu sans eux. Il faut remarquer qu'aucune contrainte ne porte sur les alphabets des différents processus composés et qu'il est donc possible de synchroniser plusieurs processus sur un même événement.

Après avoir défini un modèle, les auteurs introduisent en général la notion de communication, un événement étant identifié à la transmission d'une valeur sur un canal. Dans [Hoa 85] le problème posé par la connexion éventuelle de plusieurs émetteurs sur un canal est traité de la même façon que dans [Hoa 81]: un seul émetteur peut être connecté sur un canal et participer à l'événement. En notant c! v l'émission d'une valeur v sur le canal c et c ? x la réception sur la variable x de la valeur transmise par le canal c, l'opérateur d'interaction est étendu ainsi:

$$(c ! v \rightarrow P) \| (c ? x \rightarrow Q(x)) = c! v \rightarrow (P \| Q(v))$$

ce qui permet de composer ce nouveau processus avec un processus récepteur sur le canal c mais pas avec un processus émetteur puisque le test sur l'égalité des valeurs émises n'est pas inclus dans la définition de l'opérateur.

L'autre extension de CSP 78 fut la définition du langage OCCAM qui n'avait pas pour but de définir de nouveaux outils mais plutôt de faire un véritable langage de programmation à partir des principes développés dans CSP. Par exemple le rendez-vous n'y est toujours défini qu'entre deux processus mais des canaux sont introduits de façon à permettre une programmation plus modulaire. Ce langage est actuellement implémenté sur des machines parallèles: les transputers [Bar 83] .

De cette rapide présentation de CSP et des travaux qui lui ont succédé, nous pouvons tirer plusieurs conclusions:

Il semble nécessaire dans un langage de programmation parallèle de haut niveau d'étendre le rendez-vous pour exprimer la synchronisation de plusieurs processus. Si ceci est relativement facile dans les modèles, cette extension est plus délicate dans les langages qui, en introduisant les valeurs, n'introduisent pas dans le mécanisme de synchronisation de tests sur les valeurs transmissibles.

La notion de processus comme entité autonome de calcul et de communication doit être reconnue mais il semble difficile avec l'approche proposée ici de bien cerner cette notion (voir la définition de multiples opérateurs pour exprimer les choix non déterministes) et de définir un noyau d'opérateurs de composition de processus.

Les difficultés à définir un noyau d'opérateurs pour la définition de processus et la définition de structures plus complexes avec des processus élémentaires semblent avoir leurs origines parmi les raisons suivantes:

1) les notions de processus et de réseaux de processus sont identifiées, comme dans la plupart des modèles mais

2) des restrictions sont portées sur la description des processus qui deviennent caduques dès lors qu'il s'agit de réseaux de processus.

On peut citer, à l'appui de cette remarque, la multiplicité des opérateurs de choix non déterministes: si un processus élémentaire ne peut pas être décrit par l'expression $x \rightarrow P \mid x \rightarrow Q$, c'est à dire ne peut pas effectuer un événement et avoir ensuite deux comportements différents, en revanche, l'opérateur d'interfoliage permet d'avoir de telles alternatives qui doivent être exprimées à l'aide d'autres opérateurs. On pourrait également discuter dans l'expression du non déterminisme la contrainte qui exige que les processus ne puissent effectuer que des choix entre des événements externes ou des choix entre des événements internes. Ceci entraîne en effet une complication du calcul lorsque l'utilisation de l'opérateur de masquage introduit un événement interne comme alternative d'un choix. Puisque l'outil syntaxique n'est pas défini, on doit donner la défition sémantique de ces événements, c'est à dire faire des hypothèses sur la nature du modèle de synchronisation.

Beaucoup des difficultés apparaissant dans l'expression des processus semblent venir de choix a priori (issus d'une programmation séquentielle déterministe) qui ne sont résolus qu'après coup par introduction de nouveaux opérateurs. Au contraire, on peut admettre, comme le font la plupart des modèles que le non déterminisme est inhérent à l'étude du parallélisme et que son expression peut prendre des formes variées: évolution interne alternant avec des communications avec l'environnement, évolution unique pour l'environnement avec choix internes des processus, possibilités multiples de choix internes ...

1.2 Algèbres de processus dérivées de CCS

Cette seconde section est consacrée à la présentation d'algèbres de processus communicants. Trois calculs seront exposés: le calcul asynchrone CCS développé par Milner dans [Mil 80] et les deux calculs SCCS [Mil 83] et Meije [Bou 84] qui proposent à partir de CCS deux approches pour introduire le synchronisme de processus.

1.2.1 Calculus of Communicating Systems [Mil 80]

Les processus sont considérés dans ce calcul comme des boîtes noires communiquant avec leur environnement grâce à des ports de communication orientés (émission d'une valeur, réception d'une valeur sur une variable) et sont décrits comme par des termes d'une algèbre sur un ensemble Λ d'événements et un ensemble Σ d'opérateurs. Leur comportement est décrit par un par un arbre de synchronisation (ou de communication si le processus échange des valeurs).

- Les événements peuvent être une réception, une émission ou un événement interne noté τ

  Soit Δ un ensemble de noms de ports de communication tel que $\tau \notin \Delta$

  $\underline{\Delta} = \{ \underline{\alpha} \mid \alpha \in \Delta \}$ $\quad \alpha$: port d'entrée , $\underline{\alpha}$: port de sortie

  $\Lambda = \Delta \cup \underline{\Delta} \cup \{ \tau \}$

- L'ensemble Σ est constitué de 7 opérateurs.

  • NIL : processus qui ne fait rien.
  Il est représenté par une feuille dans l'arbre de synchronisation.

  • action : a.P
  Ce processus réalise initialement l'événement a et se comporte ensuite comme le processus décrit par P.

  • somme : P1+P2
  Cet opérateur permet d'exprimer le choix non déterministe. Il se traduit par l'enracinement des arbres de synchronisation associés aux processus P1 et P2.

Sa sémantique, à la Plotkin [Plo 81] est donnée par les deux règles suivantes et exprime que si l'un des processus peut se transformer en un autre processus en réalisant un événement alors le processus somme peut réaliser cet événement et se transformer dans le même processus:

$$\frac{P1 \text{ — } a \rightarrow P1'}{P1 + P2 \text{ —} a \rightarrow P1'} \qquad \frac{P2 \text{ — } a \rightarrow P2'}{P1 + P2 \text{ — } a \rightarrow P2'}$$

• Les définitions récursives de processus sont autorisées.

Ces définitions récursives peuvent être données sous forme d'équations sur des variables de comportement ou grâce à un opérateur rec dont la sémantique est la suivante:

$$\frac{E\ (\text{recx. } E / x) \text{ — } a \rightarrow E'}{\text{recx. } E \text{ — } a \rightarrow E'}$$

• Composition parallèle : P1 || P2.

Cet opérateur exprime la composition parallèle des deux processus. Il est caractéristique de la nature asynchrone du calcul: en dehors des points de synchronisation, les deux processus évoluent séparément (mais pas indépendamment car seul un processus à la fois peut réaliser un événement). La synchronisation est effectuée par rendez-vous lorsque l'un des processus est prêt à effectuer une émission et l'autre processus une réception sur des ports de communication complémentaires α et $\underline{\alpha}$. Cette synchronisation se traduit par l'occurence d'un événement noté τ qui remplace l'occurence synchrone de α et $\underline{\alpha}$.

La composition parallèle est traditionnellement exprimée grâce au non déterminisme, par interfoliage. La sémantique de l'opérateur est donnée par le "théorème d'expansion" :

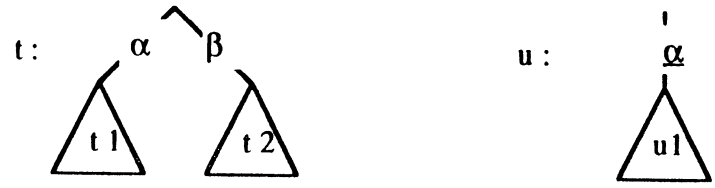
$$t = \Sigma\ \mu_i \,.\, t_i \qquad\qquad u = \Sigma\ \nu_j \,.\, u_j$$

$$t \,||\, u = \Sigma\ \mu_i \,.\, (t_i \,||\, u) + \Sigma\ \nu_j \,.\, (t \,||\, u_j) \quad \text{-- t seul ou u seul}$$

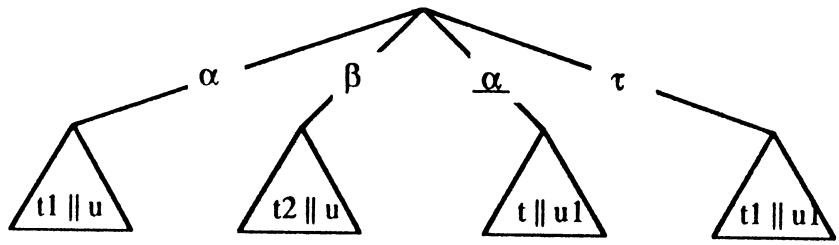
$$+ \Sigma_{\mu i = \underline{\nu j}}\ \tau\ (t_i \,||\, u_j) \quad \text{-- synchronisation de t et u}$$

Exemple:

$t = \alpha t_1 + \beta t_2$ $\quad u = \underline{\alpha} u_1$



Par = t || u



fin-exemple

• Renommage P[S]

Les processus étant écrits individuellement et la synchronisation s'effectuant par ports complémentaires, il est nécessaire de pouvoir changer les noms de ports de façon à avoir la possibilité de faire toutes les connexions utiles.

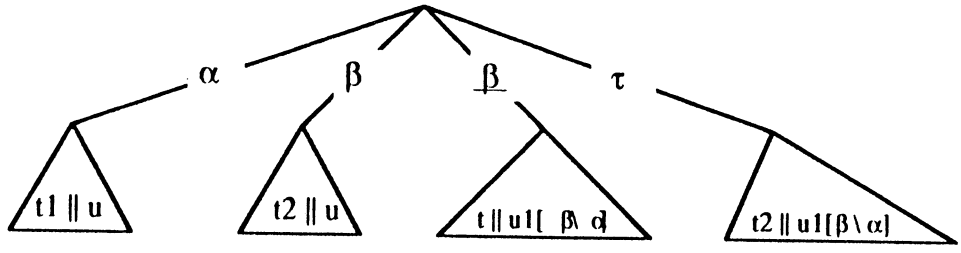
Le renommage S est une bijection , qui respecte les compléments ,de l'ensemble L des noms de ports de P dans un ensemble M:

$S(\underline{\alpha}) = \underline{S(\alpha)}$ pour tout $\alpha, \underline{\alpha} \in L$

$S(\tau) = \tau$

$(\Sigma \mu_i . t_i)[S] = \Sigma S(\mu_i) . t_i [S]$

Exemple: Ren = t || u [β \ α] (le renommage ne porte que sur u)



Fin-exemple

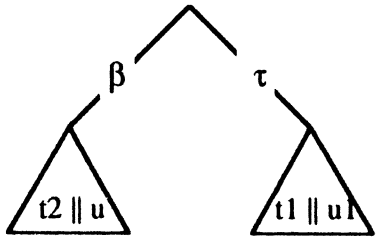
• Restriction P\C

L'opérateur de composition parallèle permet de synchroniser les processus. Toutefois, l'événement de synchronisation n'est que l'une des alternatives (les processus ayant toujours la possibilité d'évoluer séparément). Afin de rendre cet événement de synchronisation nécessaire, il faut pouvoir interdire l'usage des ports de communication par l'environnement (couper des branches dans l'arbre de synchronisation).

L'opérateur de restriction supprime tous les événements dont le nom est élément de C (C étant un sous ensemble des noms de ports de P).

$$(\Sigma \mu_i . t_i) \setminus C = \Sigma \mu_j . (t_j \setminus C) \qquad \text{si } \mu_j , \mu_j \notin C$$

Exemple : Abs = Par \ {α}



fin-exemple

Nous avons, jusqu'à présent, présenté les opérateurs fondamentaux de CCS dans le domaine des "arbres de synchronisation". Ce domaine est étendu aux processus capables de transmettre des valeurs qui sont représentés par des "arbres de communication" (terminologie définie dans CCS). Cet aspect n'apparaît pas comme fondamental dans la présentation de CCS. Il n'y est pas très développé et nous nous limiterons ici à en donner les principes.

1) Les valeurs sont des termes construits sur un ensemble de variables, de constantes et de fonctions.

2) Des identificateurs de comportement permettent de paramétrer des comportements par des variables.

3) Les ports de sortie permettent d'émettre des valeurs, les ports d'entrée de lier les variables.

Ces principes sont mis en évidence par la sémantique des opérateurs d'action et de composition parallèle.

action

$$\alpha\ x_1.....x_n.B \quad \text{——} \quad \alpha(v_1.....v_n) \quad \longrightarrow \quad B\{v_1\backslash x_1, \ldots, v_n\backslash x_n\}$$

où la notation $v_i \setminus x_i$ correspond à la substitution de la variable x_i par la valeur v_i

$$\underline{\alpha}\ v_1.....v_n.B \quad \text{——} \quad \underline{\alpha}(v_1.....v_n) \quad \longrightarrow \quad B$$

$$\tau.B \quad \text{——} \quad \tau \quad \longrightarrow \quad B$$

composition parallèle

$$\frac{B1 \xrightarrow{\mu v} B1'}{B1 \parallel B2 \xrightarrow{\mu v} B1' \parallel B2} \qquad \frac{B2 \xrightarrow{\mu v} B2'}{B1 \parallel B2 \xrightarrow{\mu v} B1 \parallel B2'}$$

$$\frac{B1 \xrightarrow{\underline{\lambda} v} B1' \qquad B2 \xrightarrow{\lambda x} B2'}{B1 \parallel B1' \xrightarrow{\tau} B1' \parallel B2' \{ v \setminus x\}}$$

En conclusion de cette présentation de CCS nous pouvons faire plusieurs remarques:

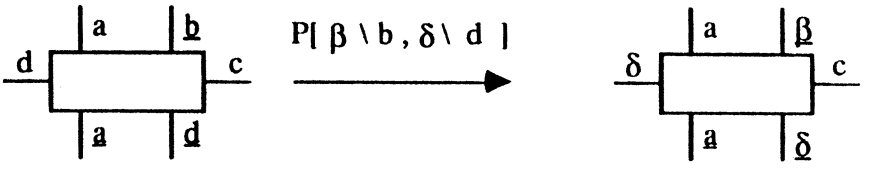
- En introduisant la possibilité de nommer les événements de synchronisation interne et en permettant qu'à partir d'un état plusieurs branches de l'arbre de synchronisation portent le même nom, le non déterminisme peut être exprimé sous toutes ses formes. L'ensemble des opérateurs proposés constitue de ce fait un ensemble suffisant pour exprimer à la fois les comportements et les compositions de processus.

- L'introduction des valeurs par liaison des variables lors des communications se fait harmonieusement dans un modèle initialement sans valeur ce qui permettra par la suite de définir des relations entre processus échangeant des valeurs.

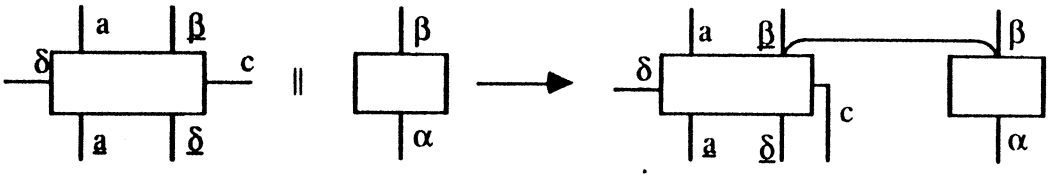
- Il faut remarquer qu'un processus n'a qu'un état initial et que la philosophie sous-jacente entraîne qu'un processus P = a . P1+ a . P2 peut se reconfigurer par réalisation de l'événement a en deux processus : P1 et P2.

- Parmi l'ensemble d'opérateurs Σ, on distingue deux sortes d'opérateurs: Ceux qui sont nécessaires pour décrire le comportement des processus (NIL, action, somme) et ceux qui sont utiles pour composer des entités indépendantes (composition parallèle, renommage, abstraction). Cette distinction est évidente si l'on cherche à représenter l'effet des opérateurs sur des schémas de réseaux. On peut représenter les opérateurs de

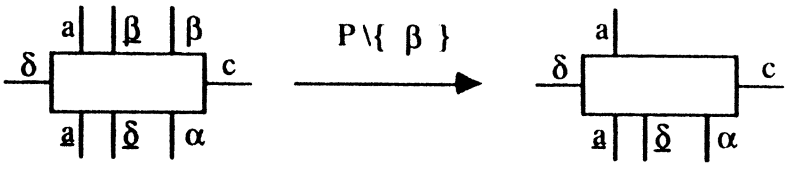
• renommage



• composition parallèle



• abstraction



En revanche, il est impossible d'associer des tels schémas aux trois premiers opérateurs.

Ces deux dernières remarques qui peuvent apparaître superficielles sont à notre avis essentielles pour la suite de cet exposé:

La première parce qu'elle a conduit à un raisonnement inductif particulier sur le comportement de processus que nous remettrons en cause (intuitivement, ce qui est mis en cause vient de la réponse qu'il faut donner à la question suivante: "Un processus après avoir effectué une séquence d'actions se transforme-t-il en plusieurs processus ou en un processus dont on ne sait pas quel sera son comportement parmi plusieurs possibles?", spécialement lorsqu'on remplace le mot processus par celui de boîte noire).

La seconde parce qu'elle a entrainé la prise en compte pour comparer le comportement des processus communicants de paramètres qui leur sont étrangers lorsqu'on les considère comme des entités autonomes. Un concept est en effet généralement admis comme bien défini par une relation d'équivalence si celle ci est une congruence pour les opérateurs qui s'appliquent au domaine [DK 85]. Les équivalences recherchées devront donc être des congruences pour tous les opérateurs de Σ, qu'ils soient du premier ou du second type, alors que si l'on ne s'intéresse qu'au comportement des processus comme élément de réseaux, seuls ceux du second type devraient être pris en considération.

La dernière remarque que nous ferons concerne le mécanisme de synchronisation. Seuls en effet deux processus peuvent participer à un rendez-vous. Ceci est une limitation des possibilités de synchronisation et a conduit à la définition ultérieure de deux calculs synchrones SCCS [Mil 83] et Meije [Bou 84] que nous allons présenter maintenant.

1.2.2 Les calculs Meije et SCCS.

Les calculs Meije [Bou 84] et SCSS [Mil 83] sont dérivés du calcul CCS en ce sens qu'ils adoptent, pour décrire les processus , un formalisme analogue (les processus sont des termes d'une algèbre) . Leur objectif commun est de pouvoir exprimer la synchronisation de plus de deux processus et leur apport principal défini dans SCCS et repris dans Meije tient à la définition de la notion d'action.

Dans CCS, une action est une transmission de valeur sur un port de communication et seules les actions sur des ports complémentaires peuvent être composées et produisent alors un événement de synchronisation. Au contraire, dans les calculs SCCS et Meije, les actions peuvent être "autre chose" que des actions de communication comme par exemple une action d'attente dans SCCS ou un produit d'actions ce qui constitue la différence essentielle avec CCS. En effet, une opération de produit est définie sur l'ensemble des actions, ce qui lui confère une structure de monoïde commutatif (de groupe commutatif si on limite les actions aux actions de communication).

- Si a et b sont deux actions alors a.b est une action qui réalise indivisiblement a et b.
- L'élément neutre est l'action d'attente dans SCCS et peut être obtenu par produit de deux actions complémentaires $a.\underline{a} = 1$
- L'élément inverse d'une action Πa_i où les a_i sont des communications est l'action $\Pi \underline{a}_i$.

Dès lors, ces deux calculs se distinguent essentiellement par le choix de leurs opérateurs. Ces choix permettent selon [Bou 84] de définir informellement SCCS comme un calcul qui peut "désynchroniser des processus synchrones" et Meije comme un calcul qui peut "synchroniser des processus asynchrones".

Les ***opérateurs communs à Meije et à SCCS*** sont les suivants: préfixage, abstraction, renommage (cet opérateur est un morphisme du monoïde et permet par conséquence de renommer une action par un paquet d'actions c'est à dire par une action élémentaire ou par une action résultant du produit d'actions élémentaires, en respectant l'orientation des actions de communication élémentaires) et définition récursive.

Les ***opérateurs spécifiques à SCCS*** sont:

- le produit: $\times$ -- composition parallèle synchrone

$$\frac{P \xrightarrow{a} P' \qquad Q \xrightarrow{b} Q'}{P \times Q \xrightarrow{a \times b} P' \times Q'}$$

Cet opérateur exige une participation simultanée des deux processus. Du point de vue des actions possibles le processus NIL de CCS est donc un élément absorbant tandis que le processus Y=1.Y est élément neutre.

- l'opérateur de délai : δ

- $\delta E \xrightarrow{1} \delta E$
- $$\frac{E \xrightarrow{a} E'}{\delta E \xrightarrow{a} E'}$$

Cet opérateur permet de définir à partir d'un processus E, un processus qui peut attendre, aussi longtemps que nécessaire, avant de se comporter comme E, sans pour autant bloquer le système. Il permet en outre de redéfinir les opérateurs de composition asynchrone.

- la somme infinitaire.

$$\frac{Ei \xrightarrow{a} E}{\Sigma_{i \in I} Ei \xrightarrow{a} E}$$

où les Ei constituent une famille indexée d'expressions de comportement

Les ***opérateurs de Meije*** sont les suivants:

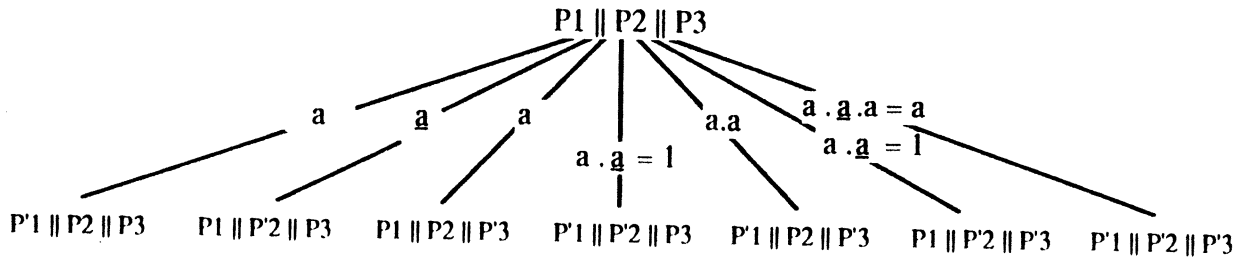
- la composition parallèle

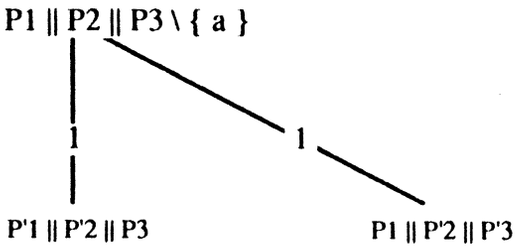
$$\frac{P1 \xrightarrow{a} P1'}{P1 \parallel P2 \xrightarrow{a} P1' \parallel P2} \qquad \frac{P2 \xrightarrow{b} P2'}{P1 \parallel P2 \xrightarrow{b} P1 \parallel P2'}$$

$$\frac{P1 \xrightarrow{a} P1' \quad P2 \xrightarrow{b} P2'}{P1 \parallel P2 \xrightarrow{a.b} P1' \parallel P2'}$$

Cet opérateur considère que la composition parallèle n'est pas traduite par l'interfoliage des comportements mais, comme cela était proposée dans [Jor 82], que les processus peuvent évoluer indépendamment ou simultanément. La connexion est automatique entre ports complémentaires mais n'existe qu'entre deux processus.

Exemple: $P1 \xrightarrow{a} P'1$, $P2 \xrightarrow{\underline{a}} P'2$, $P3 \xrightarrow{a} P'3$





Fin-exemple

Toutes les autres alternatives sont exclues parce qu'elles font intervenir directement l'une des actions a ou $\underline{a}$.

- l'opérateur de pilotage: e*P

Cet opérateur permet de lier l'occurrence de toute action d'un processus P à celle de l'action donnée en paramètre. Sa sémantique est définie par la règle suivante:

$$\frac{P \xrightarrow{\ e\ } P'}{a*P \xrightarrow{\ a.e\ } a*P'}$$

Nous avons présenté ici les choix d'opérateurs effectués pour constituer un noyau d'opérateurs, dans Meije d'une part, dans SCCS d'autre part. Dans [Bou 85] on montre que les opérateurs de construction de processus de Meije sont primitifs mais que l'on peut définir trois ensembles noyaux dont chacun est suffisant pour exprimer les compositions de processus (chacun des autres opérateurs peut être exprimé grâce aux opérateurs du noyau).

Ces 3 noyaux d'opérateurs sont les suivants:

- pilotage et composition parallèle (Meije)

- produit synchrone et désynchronisation (SCCS)

- produit synchrone et interfoliage

1.3 Le langage FP2

Dans le langage FP2[Cis 84][Jor 85][Pan 86], les processus apparaissent comme l'unité de base de programmation. C'est à l'intérieur des processus que sont effectués les calculs et, en général, un programme FP2 a la forme d'une construction de réseau de processus. Nous présenterons donc d'abord la définition des processus et le style adopté pour effectuer les calculs avant de montrer comment sont construits les réseaux de processus.

1.3.1 Notion de processus

Un processus est une entité autonome susceptible de réaliser des calculs et de communiquer avec son environnement grâce à des connecteurs qui lui sont propres et qui constituent sa seule interface avec son environnement, le seul moyen pour communiquer de l'information ou se synchroniser.

Vis à vis de son environnement, un processus a un comportement à la fois séquentiel, non déterministe et parallèle.

- le comportement est séquentiel dans la mesure où les échanges d'information qu'il effectue avec son environnement sont ordonnés dans le temps.

- le comportement est non déterministe : lors de la définition d'un processus, certains paramètres peuvent être ignorés, notamment lorsqu'un processus est constitué de processus plus élémentaires (voir § 1.3.2.2 sur l'opérateur de connexion). De ce fait, un processus garde la liberté d'effectuer certains choix contraignant son évolution ultérieure, l'environnement n'ayant aucun contrôle sur ces choix.

- le comportement est parallèle: on admet en effet qu'un processus peut réaliser en parallèle (indivisiblement) un ensemble de communication sur plusieurs connecteurs différents. De plus le style de programmation applicatif adopté pour le traitement des valeurs permet l'évaluation parallèle de termes fonctionnellement indépendants.

1.3.1.1 Définitions formelles

Les processus sont décrits par des systèmes de transitions d'états (éventuellement infinis), les valeurs sont des termes et les calculs sont effectués par réécriture de termes. Avant de donner la définition des processus, nous rapellerons quelques définitions préliminaires.

1.3.1.1.1 Définitions préliminaires

a) *termes de type t.*

Soit $\mathbb{T}=\{t_1,\ldots,t_n\}$ un ensemble de noms de type, $\mathbb{X}$ un ensemble de noms de variables et $\mathbb{F}$ un ensemble de noms de fonction (disjoint de $\mathbb{X}$).

Un opérateur $\alpha: \mathbb{F}\rightarrow\mathbb{T}^*$ représente l'arité d'une fonction de $\mathbb{F}$ et un opérateur $\beta:\mathbb{F}\cup\mathbb{X} \rightarrow \mathbb{T}$ le type d'une fonction ou d'une variable.

L'ensemble $\mathbb{T}_t$ des termes de type t construit sur l'ensemble des noms de variables $\mathbb{X}$ et des noms de fonctions $\mathbb{F}$ est défini par:

1. $\forall x\in \mathbb{X},\ \beta(x) = t \Rightarrow x\in \mathbb{T}_t$
2. $\forall f\in \mathbb{F},\ \alpha(f) = \varepsilon \wedge \beta(f)=t \Rightarrow f\in \mathbb{T}_t$
3. $\forall f\in \mathbb{F},\ \alpha(f)=<t_1,\ldots,t_n>\ n\geq 1 \wedge \beta(f)=t \Rightarrow$

$$\forall <u_1,\ldots,u_n>\in \mathbb{T}_{t_1}\times\ldots\times\mathbb{T}_{t_n}\ \ f(u_1,\ldots,u_n)\in \mathbb{T}_t$$

On appellera valeur de type t tout terme de type t sans variable.

b) *Etats*

Soit $\mathbb{Q} = \{q_1,\ldots,q_n\}$ un ensemble fini de symboles d'état et α un opérateur d'arité $\alpha: \mathbb{Q} \rightarrow \mathbb{T}^*$

L'ensemble $\mathbb{TE}_{\mathbb{Q}}$ des termes d'états définis sur $\mathbb{Q}$,$\mathbb{F}$et $\mathbb{X}$ est défini par:

1. $\forall q \in \mathbb{Q}, \alpha(q)=\varepsilon \Rightarrow q \in \mathbb{TE}_{\mathbb{Q}}$

2. $\forall q \in \mathbb{Q}, \alpha(q)=<t_1,\ldots,t_n> (n\geq 1) \Rightarrow$

$$\forall <u_1,\ldots,u_n> \in \mathbb{T}_{t_1}\times\ldots\times\mathbb{T}_{t_n} , q(u_1,\ldots,u_n) \in \mathbb{TE}_{\mathbb{Q}}$$

On appellera <u>état</u>, tout terme d'état sans variables et <u>profil d'état</u> tout terme $q(u_1,\ldots,u_n)$ tel que pour tout i, u_i soit une variable.

On définit le produit de deux termes d'états:

Soit q un constructeur d'états d'arité s et q' un constructeur d'états d'arité s', le produit de deux termes d'états $q(u_1 , \ldots , u_n)$ et $q'(v_1 , \ldots , v_m)$ est le terme d'état qu'on note $qq'(u_1 , \ldots , u_n , v_1 , \ldots , v_m)$ construit sur le constructeur d'états noté qq' dont l'arité est définie comme la concaténation de s et s'.

c) *Communications, événements*

Soit $\mathbb{K}=\{k_1,\ldots,k_n\}$ un ensemble de symboles de communications (connecteurs) et $\alpha : \mathbb{K} \rightarrow \mathbb{T}^+$ l'opérateur d'arité (à la différence des cas précédents, l'arité est définie dans $\mathbb{T}^+$ et ne peut pas être vide, une communication étant la circulation d'un message sur un connecteur).

L'ensemble $\mathbb{T}_{\mathbb{K}}$ des termes de communications (ou plus simplement <u>communications</u>) est défini par:

$$\forall k \in \mathbb{K}, \alpha(k)=<t_1,\ldots,t_n> n\geq 1$$
$$\Rightarrow \forall <u_1,\ldots,u_n> \in \mathbb{T}_{t1}\times\ldots\times\mathbb{T}_{tn} \quad k(u_1,\ldots,u_n) \in \mathbb{T}_{\mathbb{K}}$$

On appellera <u>événement</u> tout ensemble e (éventuellement vide) de termes de communication ne contenant pas deux termes construits sur le même symbole de communication.

On notera k : t1, ... , tn pour $\alpha(k)=<t_1,\ldots,t_n>$

Exemple :

$\mathbb{K} = \{k_1:t_1 , k_2:t_1 , t_2\}$

$e_1 = \emptyset$ -- e_1 est un événement qu'on notera τ

$e_2=\{k_1(u_1)\}$
$e_3=\{k_1(u_1) , k_2(v_1,v_2)\}$ -- e_2 et e_3 sont des événements si u_1 et v_1 sont des termes de type t_1 , v_2 un terme de type t_2

$e_4=\{k_1(u_1) , k_1(v_1)\}$ -- e_4 n'est pas un événement car il utilise deux fois le connecteur k_1

Fin-Exemple

Notre objectif n'est pas de détailler le traitement des valeurs dans le langage FP2. Disons simplement que les types sont définis comme des types abstraits algébriques et les fonctions par réécriture de termes [Jor 86] [Ber 79][Ech 86] . Toutes ces définitions (types et fonctions) sont faites à l'extérieur des processus et nous ne considérerons ici que des fonctions constructeurs (comme par exemple les fonctions 0 et succ pour les entiers naturels).

1.3.1.1.2 Processus = Système de transitions d'états

Un processus est décrit par un système de transitions d'états, éventuellement infini, exprimé par un quintuplet $< \mathbb{K}, \mathbb{Q}, \mathbb{V}, \mathbb{I}, \mathbb{R} >$ où:

- $\mathbb{K}$ est un ensemble fini de définitions de connecteurs

  $\mathbb{K} = \{(k_i, s_i)\}$ où $s_i \in \mathbb{T}^+$ est l'arité de k_i

- $\mathbb{Q}$ est un ensemble fini non vide de définitions de symboles d'états

  $\mathbb{Q} = \{(q_i, s_i)\}$ où $s_i \in \mathbb{T}^*$ est l'arité de q_i

- $\mathbb{V}$ est un ensemble fini de définitions de variables

  $\mathbb{V} = \{(x_i, t_i)\}$ où $t_i \in \mathbb{T}$ est le type de la variable

- $\mathbb{I}$ est un ensemble fini non vide d'états construits sur $\mathbb{Q}$

$\mathbb{I}$ représente l'ensemble des états initiaux possibles du processus

- $\mathbb{R}$ est un ensemble fini de règles de transition décrivant le comportement du processus.

Une règle de transition r est un triplet (s , e , s') qu'on notera s : e $\Rightarrow$ s' tel que

. s et s' sont des termes d'états construits sur $\mathbb{Q}$

. e est un événement construit sur $\mathbb{K}$

. s , s' et e vérifient $var(s') \subseteq var(s) \cup var(e)$

(On appelle var(t) l'ensemble des variables d'un terme t).

Le comportement d'un processus décrit par un quintuplet $< \mathbb{K}, \mathbb{Q}, \mathbb{V}, \mathbb{I}, \mathbb{R} >$ est le suivant: à tout instant, le processus est dans un état courant qui est initialement l'un des états de $\mathbb{I}$.

Lorsqu'un processus est dans un état q, il ne peut réaliser une transition (q , E , q') que s'il existe une règle de transition r=(s , e , s') dans R telle que

- il existe une substitution σ_s telle que $q = \sigma_s (s)$
- E est un ensemble de termes de communication sans variable et il existe une substitution σ_e telle que $E = \{ \sigma_e (\sigma_s (c)) \mid c \in e \}$
- q' est un état défini par $q' = \sigma_e (\sigma_s (s'))$

Lorsque plusieurs transitions sont possibles à partir d'un état q l'une d'entre elles est choisie de façon non déterministe. Au cours de son exécution, un processus parcourt séquentiellement un ensemble d'états, l'état q' d'une transition (q , E , q') étant atteint après l'état q et l'événement E étant réalisé indivisiblement.

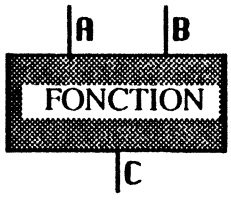
1.3.1.2 Exemples

On supposera dans cet exemple l'existence d'un type t , d'une fonction f d'arité $t \times t$ et de type t pour laquelle existe dans t un élément neutre e, et enfin d'un type signal à une valeur notée " sig ".

Nous allons présenter trois processus qui seront utilisés par la suite pour construire un réseau.

- Processus FONCTION

Ce processus, qui peut être représenté par le schéma suivant, reçoit séquentiellement , sur les connecteurs A et B deux valeurs et fournit ensuite, sur le connecteur C, le résultat de l'application de la fonction f à ces deux valeurs.



FONCTION= < $\mathbb{K}$ ={ A , B , C : t,}

$\mathbb{Q}$ ={ X : ε ; Y , Z: t},

$\mathbb{V}$ ={ n , m , p : t},

$\mathbb{I}$ = { X },

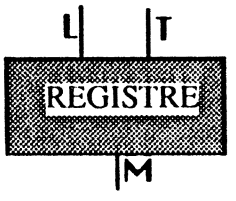
$\mathbb{R}$ = { X : A(n) $\Rightarrow$ Y(n),

Y(n) : B (m) $\Rightarrow$ Z(f (n , m)),

Z(p) : C(p) $\Rightarrow$ X } >

- Processus REGISTRE

Ce processus conserve une valeur de type t qu'il peut émettre sur le connecteur M tant que celle-ci n'a pas été invalidée par la réception d'un signal sur le connecteur T. Lorsque le signal a été reçu, le processus doit attendre la réception d'une nouvelle valeur sur le connecteur L.



REGISTRE= < $\mathbb{K}$ ={ L , M : t , T : signal},

$\mathbb{Q}$ ={ F: t , G : ε },

$\mathbb{V}$ ={ r : t},

$\mathbb{I}$ = { F(e) },

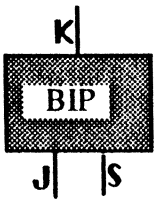
$\mathbb{R}$ = { F(r) : M(r) ⇒ F(r),

F(r) : T(sig) ⇒ G ,

G : L(r) ⇒ F(r) } >

- Processus BIP

Ce processus émet un signal sur le connecteur S lors du passage d'une même valeur de type t sur les connecteurs K et J.



BIP = < $\mathbb{K}$ ={ K , J : t , S : signal},

$\mathbb{Q}$ ={ H : ε} ,

$\mathbb{V}$ ={ x : t},

$\mathbb{I}$ = { H },

$\mathbb{R}$ = { H : K(x) J(x) S(sig) ⇒ H } >

1.3.2 Construction de réseaux de processus

Nous distinguerons deux parties dans l'ensemble des opérateurs de construction de processus qui sont définis dans [Jor 85]. La première se réduit à trois opérateurs et constitue un noyau. Elle permet en effet de définir exactement l'ensemble des opérations que l'on peut faire sur des processus lorsqu'on les considère comme des entités indépendantes ne se synchronisant que par échanges de messages. La seconde est un ensemble d'opérateurs exprimés dans le même formalisme et qui ont été introduits pour permettre une plus grande souplesse dans la programmation. Ces derniers opérateurs sont en général empruntés à d'autres modèles.

1.3.2.1 Un noyau d'opérateurs

Les opérateurs de construction de réseaux de processus qui seront présentés dans cette section constituent un noyau pour construire des réseaux statiques (dont la structure est figée au cours de l'exécution) dans un système a-temporel.

Par a-temporel, on entend ici un système dans lequel sont exclues les notions de temps: délai d'attente avant occurence d'un événement, contraintes temporelles intervenant dans les choix non-déterministes(temps de réponse ...), durées variables des communications. L'hypothèse est donc que toute communication et par conséquent tout événement, est instantanée ce qui permet de définir et d'utiliser la notion d'état d'un réseau de processus (caractérisé par l'ensemble des états des processus du réseau).

Il n'y a donc a priori, aucun obstacle à considérer un réseau de processus comme un processus. Toute application d'opérateur de composition de processus aura donc comme résultat un processus et la sémantique de ces opérateurs sera donnée par le quintuplet qui décrit le processus résultat.

Les opérateurs du noyau permettent, dans le cadre précédent, de respecter l'autonomie des processus, c'est à dire leur évolution indépendante en dehors des points de synchronisation. Le seul mécanisme de synchronisation exprimable à l'aide de ces opérateurs est celui du rendez-vous étendu à n interlocuteurs: pour qu'un événement à m communications faisant intervenir n processus puisse avoir lieu, il est nécessaire que tous les processus y participant soient dans un état qui leur permette de réaliser exactement leur part de l'événement global. Un processus reste donc bloqué dans un état si pour changer d'état il doit communiquer avec d'autres processus qui ne sont pas prêts.

Nous dirons du modèle défini par ce noyau d'opérateurs qu'il est asynchrone dans la mesure où deux processus n'ayant aucune contrainte de synchronisation pourront évoluer totalement indépendamment. Ceci n'exclut pas la possibilité de programmer des comportements totalement synchrones comme nous le verrons dans la section suivante.

Les opérateurs qui constituent le noyau sont au nombre de trois:

* La ***composition parallèle*** permet de construire un processus à partir de deux processus élémentaires et traduit leur évolution indépendante: chacun des processus peut évoluer seul ou les deux processus peuvent évoluer simultanément.
* La ***connexion*** de deux connecteurs établit un lien direct entre deux connecteurs d'un processus.
* L' ***abstraction*** sur un connecteur interdit l'usage de ce connecteur par l'environnement.

1.3.2.1.1 Composition parallèle : ||

Cet opérateur a pour opérandes deux processus et construit un nouveau processus dont les termes d'états sont le produit des termes d'état de chacun des processus. A partir de chaque état, le processus composé peut réaliser un événement de l'un des processus ou un événement de chacun des processus (l'union des événements des processus opérandes).

Si $P1 = < K1, Q1, V1, I1, R1 >$ et $P2 = < K2, Q2, V2, I2, R2 >$ alors

$P1 \| P2 = < K1 \cup K2,$

$Q1 * Q2,$

$V1 \cup V2 \cup W1 \cup W2,$ -- W1 et W2 sont des ensembles de variables nécessaires à la définition de profils d'états

$I1 * I2,$

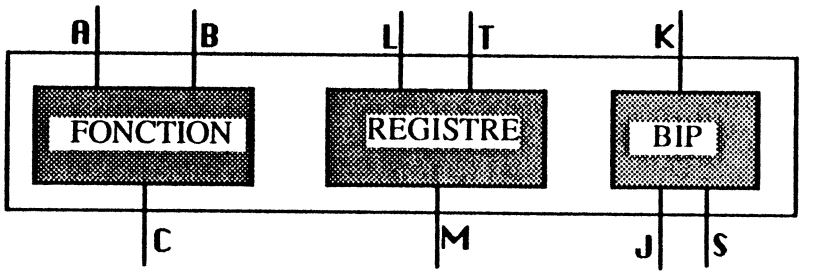
$\{ (s1 * s2 , e1 , s'1 * s2) \mid$ s2 est un profil d'état construit sur Q2 et W2 $(s1, e1 , s'1) \in R1 \}$

$\cup \quad \{ (s1 * s2 , e2 , s1 * s'2) \mid$ s1 est un profil d'état construit sur Q1 et W1 $(s2 , e2 , s'2) \in R2 \}$

$\cup \quad \{ (s1 * s2 , e1 \cup e2 , s'1 * s'2) \mid (s1, e1 , s'1) \in R1 \quad (s2 , e2 , s'2) \in R2 \} >$

Exemple

Par = (FONCTION || REGISTRE) || BIP



Par = < $\mathbb{K}$ = { A , B , C , L , M , K , J : t ; S, T : signal }

$\mathbb{Q}$ = { XFQ , YGQ , ZGQ : t

YFQ , Z FQ: t × t , XGQ : ε}

$\mathbb{V}$ = { n , m , p , q , r : t }

$\mathbb{I}$ = { XFQ(e) }

$\mathbb{R}$ = {

XFQ(r)	:	A(n)			⇒ YFQ(n , r)	1
XFQ(r)	:		M(r)		⇒ XFQ(r)	2
XFQ(r)	:		T (sig)		⇒ XGQ	3
XFQ(r)	:			K(x) J(x) S (sig)	⇒ XFQ(r)	4
XFQ(r)	:	A(n)	M(r)		⇒ YFQ(n , r)	5
XFQ(r)	:	A(n)	T (sig)		⇒ YGQ(n)	6
XFQ(r)	:	A(n)		K(x) J(x) S (sig)	⇒ YFQ(n , r)	7
XFQ(r)	:		M(r)	K(x) J(x) S (sig)	⇒ XFQ(r)	8
XFQ(r)	:		T (sig)	K(x) J(x) S (sig)	⇒ XGQ	9
XFQ(r)	:	A(n)	T (sig)	K(x) J(x) T (sig)	⇒ YGQ(n)	10
XFQ(r)	:	A(n)	M(r)	K(x) J(x) S (sig)	⇒ YFQ(n , r)	11

•
• 11 règles analogues à partir de YFQ (n , r) et ZFQ(n , r)
•

ZGQ(p)	:	C(p)			⇒ XGQ	A
ZGQ(p)	:		L(r)		⇒ ZFQ(p , r)	B
ZGQ(p)	:			K(x) J(x) S (sig)	⇒ ZGQ(p)	C

ZGQ(p)	:	C(p)	L(r)		⇒ XFQ (r)	D
ZGQ(p)	:	C(p)		K(x) J(x) S (sig)	⇒ XGQ	E
ZGQ(p)	:		L(r)	K(x) J(x) S (sig)	⇒ ZFQ(p , r)	F
ZGQ(p)	:	C(p)	L(r)	K(x) J(x) S (sig)	⇒ XFQ(r)	G

• 7 règles analogues à partir de XGQ et YGQ(n)

Fin-exemple

1.3.2.1.2 Connexion : +

Cet opérateur a deux opérandes: un processus et un couple de connecteurs de ce processus. Chaque fois qu'un événement contient une communication sur chacun des deux connecteurs, tout message susceptible de circuler sur les deux connecteurs peut être transmis directement sans intervention de l'environnement. Les messages pouvant ainsi être communiqués entre connecteurs doivent être des messages possibles sur chacun des connecteurs pris séparément et devront donc être multiples de l'unificateur le plus général (pgu(u , v)) des termes u et v communiqués.

Si $P = < \mathbb{K} , \mathbb{Q} , \mathbb{V} , \mathbb{I} , \mathbb{R} >$ et $\mathbb{K} \supseteq \{ A , B \}$ alors

$P + A.B = < \mathbb{K} , \mathbb{Q} , \mathbb{V} , \mathbb{I} , \mathbb{R} \cup R_+ >$

$$\text{où } R_+ = \{ g \mid (r , e - \{ A(u) , B(v) \} , s)] \mid g = pgu(u , v) \quad (r , e , s) \in R , \{ A(u) , B(v) \} \subseteq e \}$$

Cet opérateur peut produire des règles mal formées du type (s , e , s') ne vérifiant pas $var(s') \subseteq var(s) \cup var(e)$. Si l'on fait la connexion A.B sur un processus qui contient une règle

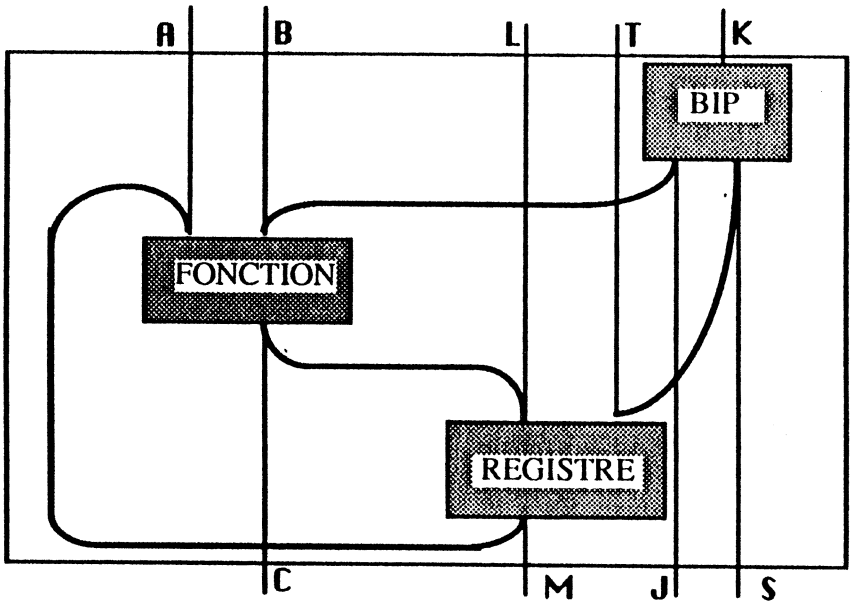
Q : A(x) B(x) ⇒ Y(x)

alors la règle suivante sera produite

Q : ⇒ Y(x)

qui est une règle mal formée puisque la variable qui apparaît dans le terme d'état final n'est pas définie. Une étude approfondie des questions de causalité soulevées par cet opérateur a été faite dans [Sch 85].

Exemple: Par-Cx = Par + CL + MA + JB + ST



Pour chaque connexion, nous présentons les règles produites.

La connexion CL envoie le résultat de l'évaluation de la fonction vers le registre — règle source

	ZGQ(p) :		⇒ XFQ(r)	D
	ZGQ(p) :	K(x) J(x) S (sig)	⇒ XFQ(r)	G

La connexion MA fournit le premier opérande de la fonction, la valeur du registre

	XFQ(r) :		⇒ YFQ(r , r)	5
	XFQ(r) :	K(x) J(x) S (sig)	⇒ YFQ(r , r)	11

La connexion JB fournit comme deuxième opérande une nouvelle valeur

	YFQ(n , r):		K(x)S (sig)	⇒ ZFQ(f(n , x) , r)	7
	YFQ(n , r):	M(r)	K(x)S (sig)	⇒ ZFQ(f(n , x) , r)	11
(*)	YFQ(n , r):	T (sig)	K(x)S (sig)	⇒ ZGQ(f(n , x))	10
	YGQ(n) :		K(x)S (sig)	⇒ ZGQ(f(n , x))	E
	YFQ(n , r):	L(r)	K(x)S (sig)	⇒ ZFQ(f(n , x) , r)	G

La connexion ST interrompt le registre lors du passage d'une nouvelle valeur

XFQ(r) :		K(x) J(x)	⇒ XGQ	9
XFQ(r) :	A(n)	K(x) J(x)	⇒ YGQ(n)	10
YFQ(n , r):		K(x)J(x)	⇒ YGQ(n)	9
YFQ(n , r):	B(m)	K(x) J(x)	⇒ ZGQ(f(n , m))	10
YFQ(n , r):		K(x)	⇒ ZGQ(f(n , x))	*
ZFQ(p , r):		K(x)J(x)	⇒ ZGQ(n)	9
ZFQ(p , r):	C(p)	K(x) J(x)	⇒ XGQ	10

1.3.2.1.3 Abstraction : -

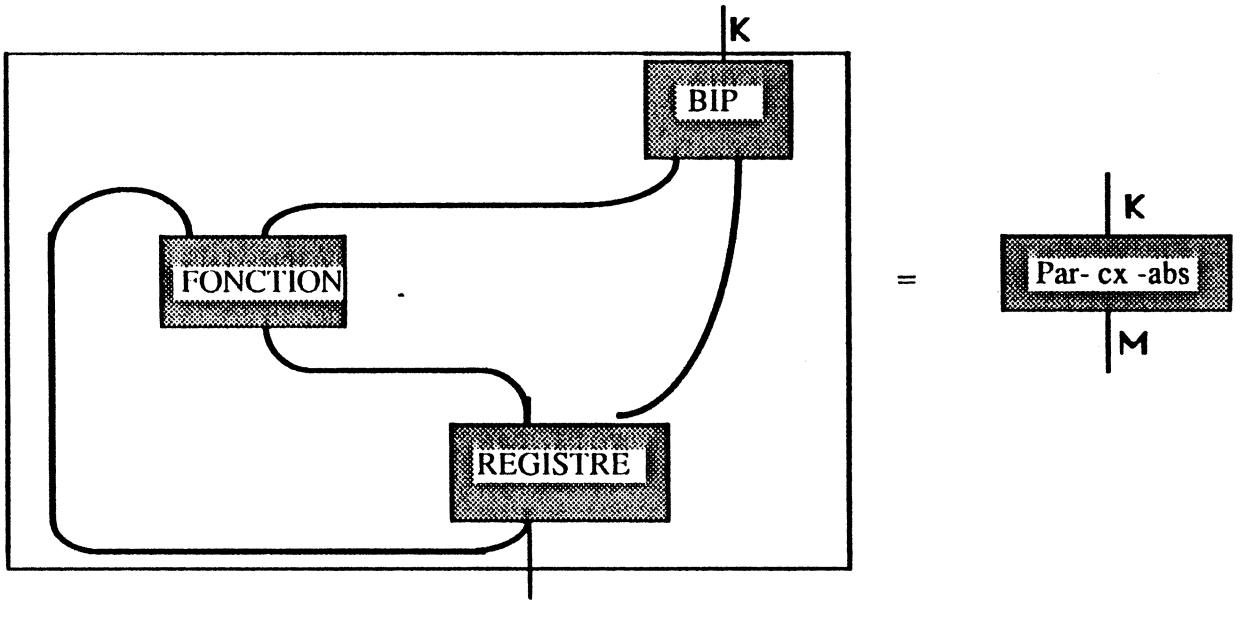
Cet opérateur a pour opérandes un processus et un connecteur. Il permet d'interdire à l'environnement l'usage du connecteur donné en paramètre. Il se traduit par la suppression de toutes les règles dont l'événement contient une communication sur ce connecteur.

Si $P = < \mathbb{K} , \mathbb{Q} , \mathbb{V} , \mathbb{I} , \mathbb{R} >$ et $k \in \mathbb{K}$ alors

$P - k = < \mathbb{K} - \{ k \}, \mathbb{Q} , \mathbb{V} , \mathbb{I} , \mathbb{R} - \{ (r , e , s) \in \mathbb{R} \mid k(u) \in e \} >$

Exemple

Par-cx-abs= Par-cx - A - C - L - S - T - J - B



Par -cx-abs = $< \mathbb{K} =$ $\{M , K: t \ ; \}$

$\mathbb{Q} =$ $\{ XFQ , YGQ , ZGQ : t \ ; \ YFQ , ZFQ : t \times t , XGQ : \varepsilon \}$

$\mathbb{V} =$ $\{ n , m , p , q , r : t \}$

$\mathbb{I} =$ $\{ XFQ(e) \}$

$\mathbb{R} =$ $\{$

$XFQ(r)$	:		$\Rightarrow YFQ(r , r)$
$XFQ(r)$	:	$M(r)$	$\Rightarrow XFQ(r)$
$YFQ(n , r)$	:	$K(x)$	$\Rightarrow ZGQ(f(n , x))$
$YFQ(n , r)$	:	$M(r)$	$\Rightarrow YFQ(n , r)$
$ZFQ(p , r)$	:	$M(r)$	$\Rightarrow ZFQ(p , r)$
$ZGQ(p)$	:		$\Rightarrow XFQ(p) \ \} >$

Ce réseau permet de communiquer sur M le résultat de l'insertion de la fonction f évaluée sur la séquence de valeurs ayant circulé sur le connecteur K. Ce réseau fait appel à la synchronisation des trois processus: Une nouvelle valeur ne peut circuler sur K que lorsque les événements JB (de BIP avec FONCTION) et ST (de BIP avec REGISTRE) sont possibles. Pour qu'une nouvelle valeur puisse être reçue,

1) le processus FONCTION doit être capable de recevoir, sur B, un nouvel opérande et par conséquent disposer de l'évaluation de la fonction f sur la séquence déjà traitée(l'avoir reçue précédemment sur A)

2) le processus registre doit invalider sa valeur, ce qui garantit qu'il ne peut pas émettre pendant le calcul de f, une valeur ignorant la dernière valeur reçue sur K.

1.3.2.2 D'autres opérateurs

Au delà du noyau d'opérateurs que nous venons de présenter et qui caractérise la philosophie du langage FP2, d'autres opérateurs, empruntés à d'autres calculs ont été définis. On y retrouve les opérateurs de choix, contrôlable et incontrôlable, définis dans CSP ainsi que des opérateurs de composition synchrone essentiellement issus des calculs Meije[Bou 84] et SCCS [Mil 83] (déclenchement, contrôle par événement(pilotage), time out, composition synchrone, interfoliage).

Ces opérateurs ont été introduits dans FP2 parce qu'ils apparaissent utiles dans certains types d'application (les applications dites temps réel, par exemple, nécessitent des références à des horloges ce qui est plus facile à exprimer dans un modèle synchrone [Ber 83]).

Toutefois, ces opérateurs ne doivent être considérés ici que comme des facilités de programmation . En effet, les mêmes comportements peuvent être obtenus à partir d'une programmation adéquate des processus élémentaires (qui doit inclure, si l'on veut associer le comportement d'un processus à l'évolution d'une horloge, l'existence d'une communication avec elle dans chaque événement) et d'une construction de réseaux n'utilisant que les opérateurs du noyau. Il faut aussi remarquer que ces nouveaux opérateurs violent le principe énoncé dans l'introduction selon lequel toute synchronisation devait se faire par échange de messages.

1.3.2.2.1 Définition d'opérateurs synchrones

Nous présenterons ici deux opérateurs: l'opérateur de contrôle par événement dont il a été montré dans [Aus 83] et [Sch 85] qu'il était un opérateur primitif pour ce genre de programmation et la composition synchrone qui est un opérateur révélateur de la philosophie de ce type de programmation.

a) *Contrôle par événement :* $e \Rightarrow$

Cet opérateur a pour opérandes un processus P et un événement e construit sur des connecteurs qui ne sont pas des connecteurs de P. Le processus $e \Rightarrow P$ se comporte comme P mais toute occurence d'un événement de P est liée indivisiblement à une occurence de l'événement e.

Si $P = < K , Q , V , I , R >$ et $\forall k , k(u) \in e$ implique $k \notin K$ alors

$$e \Rightarrow P = < K \cup K_e , Q , V \cup V_e , I , \{(r , e_p \cup e , s) \mid (r , e_p , s) \in R \} >$$

où K_e et V_e sont les ensembles de connecteurs et de variables utilisés pour définir l'événement e.

b) *Composition synchrone* |||

Cet opérateur a pour opérandes deux processus P1 et P2. Les deux processus ne peuvent évoluer que simultanément. De ce fait, les règles de transition du réseau expriment des changements d'états des deux processus par réalisation de l'union de deux événements; l'un de P1, l'autre de P2.

Si $P1 = < K1 , Q1 , V1 , I1 , R1 >$ et $P2 = < K2 , Q2 , V2 , I2 , R2 >$ alors

$$P1 \mid\mid\mid P2 = < K1 \cup K2, \; Q1*Q2 , \; V1 \cup V2 , \; I1*I2 , \; \{(r1*r2 , e1 \cup e2 , s1*s2) \mid (r1 , e1 , s1) \in R1 , (r2 , e2 , s2) \in R2 \} >$$

1.3.2.2.2 Programmation de comportements synchrones à l'aide des opérateurs du noyau de FP2

Nous allons montrer dans cette section comment les comportements définis par ces deux nouveaux opérateurs peuvent être programmés à l'aide des opérateurs de composition parallèle, de connexion et d'abstraction. Plus précisément, nous montrerons qu'il suffit, lorsqu'on veut utiliser un processus P dans un réseau construit à l'aide de ces opérateurs, d'utiliser un processus P_{ctrl} qui se déduit de P par l'ajout d'un connecteur de contrôle c_p transmettant des valeurs de type signal et d'une communication de contrôle dans tout événement. Par définition, ce processus est identique au processus $c_p \Rightarrow P$.

Pour montrer que toutes les constructions peuvent être obtenues par programmation, nous montrerons comment on peut retrouver un processus à partir du processus contrôlé et que toute construction peut être obtenue sous une forme contrôlée.

1- **Programmation de P:**

$P = P_{ctrl} \parallel UN + U.c_p - U - c_p$

où $UN = < \{ U \}, \{ Q \}, \emptyset, \{ Q \}, \{ (Q, \{ U(sig) \}, Q) \} >$

Le processus UN est toujours prêt à communiquer pour réaliser la communication de contrôle. Par conséquent, la construction proposée construit le processus P à partir du processus P_{ctrl}.

2- **Programmation de $c \Rightarrow (e \Rightarrow P)$:**

$c \Rightarrow (e \Rightarrow P) = P_{ctrl} \parallel OP + U.c_p - U - c_p$

où $c = k (sig)$

et $OP = < \{ U , k \} \cup K_e, \{ Q \}, \emptyset, \{ Q \}, \{ (Q, \{ U(sig) , c \} \cup e, Q) \} >$

Le connecteur U du processus OP permet de lier toute occurence d'un événement de P à une occurence de e . Le connecteur k permet de faire de cette construction un réseau contrôlable à son tour (que l'on peut décontrôler, pour obtenir $e \Rightarrow P$, grâce à UN comme ci-dessus).

3- **Programmation de $c \Rightarrow(P1 \parallel\parallel P2)$:**

$c \Rightarrow (P1 \parallel\parallel P2) = P1_{ctrl} \parallel P2_{ctrl} \parallel OP + U1.c_{p1} + U2.c_{p2} - U1 - U2 - c_{p1} - c_{p2}$

où $c = k (sig)$

et $OP = < \{ U1 , U2 , k \}, \{ Q \}, \emptyset, \{ Q \}, \{ (Q, \{ U1(sig) , U2(sig) , c \}, Q) \} >$

L'événement du processus OP est réalisé indivisiblement. Pour qu'un événement puisse avoir lieu dans P1, il est nécessaire qu'un événement ait lieu dans OP puisque les connecteurs Ui et c_{pi} sont connectés et inutilisables autrement du fait de l'abstraction. Pour la même raison , ceci ne peut être le cas que si P2 réalise également un événement. La construction proposée permet donc effectivement d'obtenir un réseau qui se comporte comme la composition synchrone contrôlée des processus P1 et P2. Pour obtenir le réseau P1 ||| P2 il suffit alors de construire un nouveau réseau selon la première construction proposée avec le processus UN.

4- **Programmation du contrôle des réseaux obtenus par application des opérateurs du noyau :**

a) **$c \Rightarrow$ (P1 || P2)**

$c \Rightarrow (P1 \| P2) = P1_{ctrl} \| P2_{ctrl} \| OP + U1.c_{p1} + U2.c_{p2} - U1 - U2 - c_{p1} - c_{p2}$

où $c = k\ (sig)$

et $OP = < \{ U1 , U2 , k \} , \{ Q \} , \emptyset , \{ Q \} , \{ (Q , \{ U1(sig) , c \} , Q) , (Q , \{ U2(sig) , c \} , Q) , (Q , \{ U1(sig) , U2(sig) , c \} , Q) \} >$

Le processus OP auquel sont liés les processus P1 et P2 est toujours prêt à communiquer avec les deux processus simultanément, ou séparément . Le processus obtenu par cette construction est donc le processus P1 || P2 contrôlé.

b) **$c \Rightarrow$ (P+ AB) , $c \Rightarrow$ (P - k)**

On a supposé que les connecteurs de contrôle étaient ajoutés a posteriori. Par conséquent , on supposera qu'on ne peut les utiliser pour des connexions ou des abstractions que pour réaliser des constructions du type des précédentes. De ce fait, le connecteur c_p qui permet de contrôler P permet également de contrôler $c \Rightarrow (P+ AB)$ et $c \Rightarrow (P - k)$.

Par ces différentes constructions, nous avons montré qu'il était suffisant de programmer les processus élémentaires comme des processus contrôlés pour pouvoir construire un réseau ayant un comportement décrit à l'aide d'autres opérateurs que les opérateurs du noyau.

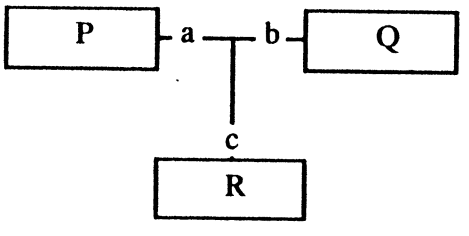
1.3.2.2.3 Programmation de canaux

Nous avons vu lors de la présentation de CSP, une alternative à l'expression du rendez-vous multiple tel qu'il est proposé dans FP2 qui est l'utilisation de canaux partagés, et non d'événements à plusieurs communications.

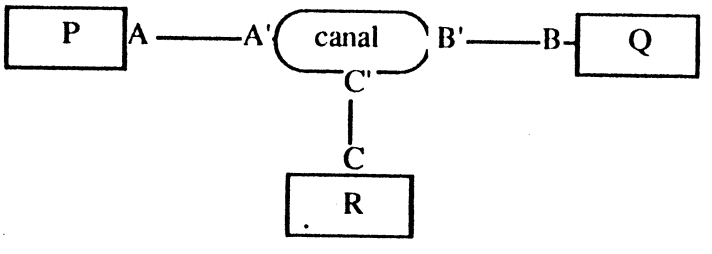
Il n'y a en FP2 aucune difficulté à définir de tels canaux. On peut en effet les considérer comme des processus et du fait du mécanisme de communication qui s'appuie sur l'unification des termes communicables, le problème soulevé dans [Hoa 81] ne se pose pas (on peut remarquer que la même propriété supprime la notion de processus émetteur et de processus récepteur au profit de celle de couple de processus d'accord pour échanger une valeur).

Exemple

Réseau CSP



Réseau FP2



Pour que le processus " Réseau FP2" fonctionne comme le processus " Réseau CSP" compte tenu de la sémantique attachée aux canaux, il suffit de programmer le processus canal ainsi:

Canal = < { A' , B' , C' : t }, où t est le type des valeurs circulant sur les connecteurs
{ X }, { x : t }, { X },
{ (X , { A'(x) , B' (x) , C' (x) }, X) } >

Ce processus impose aux trois processus d'échanger simultanément leurs valeurs et contrôle que les valeurs transmises sont les mêmes puisque la valeur transmise sur les connecteurs A', B' , C' doit être la même.

Fin-exemple

De la même façon que les opérateurs synchrones ont été introduits pour faciliter la programmation de certaines applications, l'opérateur de connexion a été étendu de façon à permettre les facilités de programmation qu'on peut avoir avec les canaux.

La première extension proposée est donc de redéfinir la connexion sur un ensemble de connecteurs. On note cet opérateur P + Cn où Cn est un sous-ensemble de l'ensemble des connecteurs de P.

Si $P = < K , Q , V , I , R >$ et $K \supseteq Cn$ alors

$P + Cn = < K , Q , V , I , R \cup R_+ >$

où $R_+ = \{ g((r , e - E , s)) \mid (r , e , s) \in R , \forall k \in Cn \ \exists k(u_k) \in e,$

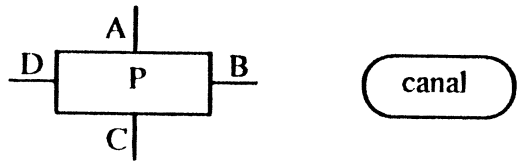
$E = \{ k(u_k) \mid k(u_k) \in e \wedge k \in Cn \},$

$g = pgu(\{ u_k \mid k(u_k) \in E \}) \}$

Exemples

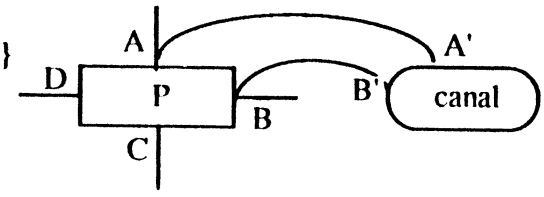
1) connexion d'un ensemble vide : Le processus résultat est identique au processus initial: Aucune interaction entre le processus et le canal

$P + \{ \}$



2) connexion d'un ensemble de deux connecteurs : le résultat est identique à celui obtenu par l'opérateur de connexion initial

$P + \{ A , B \}$

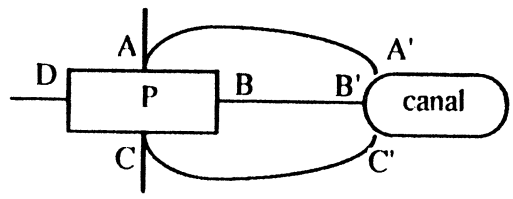


$X : A'(x) B'(x) \Rightarrow X$

3) Connexion d'un ensemble de trois connecteurs:

La connexion d'un ensemble de trois connecteurs crée un événement de synchronisation sur les trois connecteurs: Tout message qui circule sur deux de ces connecteurs circule nécessairement sur le troisième

$P + \{ A , B , C \}$

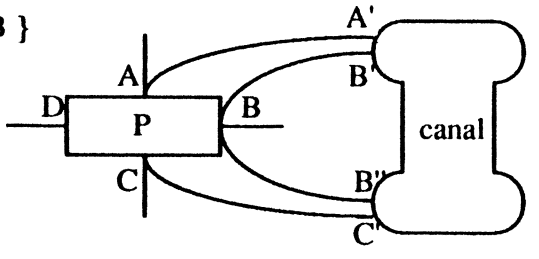


$X : A'(x) B'(x) C'(x) \Rightarrow X$

4) Plusieurs connexions à deux connecteurs.

Le résultat obtenu est différent de celui du cas précédent: le message qui circule sur le connecteur B circule dans ce cas soit sur A, soit sur B.

$P + \{A , B\} + \{C , B\}$



$X : A'(x)\,B'(x) \Rightarrow X$

$X : C'(x)\,B''(x) \Rightarrow X$

Fin-exemples

La deuxième extension permet de réutiliser le canal constitué par un ensemble de connecteurs, en créant un autre connecteur. On l'exprime par un opérateur qui s'écrit P + X : Cn et prend en paramètre un processus P un nom de connecteur X qui n'appartient pas à K(P) et un sous-ensemble Cn de K(P).

Si $< K, Q , V , I , R >$ définit le processus P, sa sémantique est donnée par le quintuplet suivant:

$P + X : Cn = < K \cup \{ X \}, Q , V , I , R \cup R_+ >$

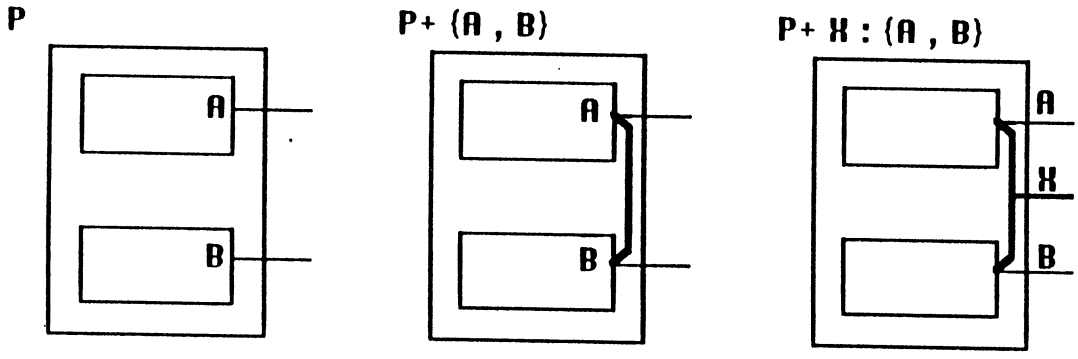
où $R_+ = \{ g ((r , e - E \cup \{ X(u) \}, s)) \mid (r , e , s) \in R , \forall k \in Cn \ \exists k(u_k) \in e$

$\exists k \in Cn, k(u) \in e, \ E = \{ k(u_k) \mid k(u_k) \in e \wedge k \in Cn \},$

$g = pgu(\{ u_k \mid k(u_k) \in E \wedge k \in Cn \}) \}$

Ces deux opérateurs construisent des processus ayant des comportements différents: avec le premier opérateur, l'ensemble des communications sur le canal peut avoir lieu dès lors qu'elles sont toutes possibles dans un même événement, avec le second opérateur pour que l'ensemble de communications puisse avoir lieu, il est nécessaire que le canal créé soit connecté à un autre canal. La communication sur le canal dans le premier cas est une communication interne tandis que dans le second cas, c'est une communication externe avec l'environnement.

Exemple



Comme le suggère cette figure , les processus P + { A , B} et P + X : { A , B} ont des comportements différents. En effet, un message ne peut circuler entre A et B dans le second processus que si celui-ci peut être reçu , par l'intermédiaire d'un connecteur lié à X par un autre processus. Au contraire, dans le premier la communication entre A et B est automatique et ne peut pas être liée à une autre communication.

Fin-exemple

Cet opérateur peut être utilisé pour programmer le renommage des connecteurs. Le processus P [X \ A] peut en effet être obtenu par l'expression suivante: P + X : {A } - A.

Le processus P + X : {A } est un processus qui dispose des deux connecteurs X et A tels que chacun d'eux peut être utilisé comme l'autre: s'il existe une règle dont l'événement contient une communication sur A alors la même règle existe avec une communication sur X remplaçant celle sur A et réciproquement.

L'abstraction sur le connecteur A ne concerne pas les règles qui contiennent X qui sont donc des règles du processus final. Ce dernier ne diffère donc de P que par le fait que le connecteur A a été remplacé par le connecteur X dans la description statique du processus ainsi que dans la description de son comportement.

1.4 Conclusion

L'étude que nous avons menée sur le thème de la compararaison des comportements de communication, qui fait l'objet des chapitres suivants a été faite à partir d'une réduction du langage FP2 aux processus finitaires d'états finis(processus décrits sans variables).

Malgré des formalismes de description des processus différents, nos résultats pourront être comparés à ceux obtenus à partir des différentes algèbres de processus qui ont été présentées (CCS , SCCS , MEIJE , TCSP) puisque tous ces formalismes décrivent le comportement des processus par des systèmes de transitions d'état étiquetées par des événements de communication-synchronisation et s'appuient sur le même principe fondamental concernant la synchronisation: le rendez-vous.

2. Comparaison Observationnelle

L'un des objectifs d'un langage de processus communicants tel que FP2 est de pouvoir construire un programme en associant plusieurs entités indépendantes entre elles, chacune de ces entités pouvant elle même être construite à partir d'éléments plus simples, de façon à intégrer le maximum de parallélisme dans la réalisation d'une tâche.

La question de savoir si processus ou un réseau peut "remplacer" un processus dans un réseau plus vaste apparaît alors comme centrale dans cette démarche, le processus remplaçant devant se comporter comme le processus remplacé, au moins dans le contexte du réseau dans lequel il doit être inséré.

On peut envisager plusieurs niveaux de réponse à cette question. Tout d'abord, il faut remarquer que le problème, tel qu'il vient d'être énoncé suppose une définition de l'équivalence des réseaux . Cette équivalence est, a priori fonction de l'utilisation que l'on veut avoir de ces réseaux, c'est à dire d'un autre contexte.

Un premier type de réponse consiste à faire abstraction des spécificités d'utilisation et à rechercher des équivalences valides quel que soit le contexte d'utilisation, c'est à dire chercher à définir une sémantique observationnelle des processus. Dans ce cas, les mêmes équivalences s'appliquent aux réseaux et aux éléments du réseau.

Un deuxième type de réponse consiste à prendre en compte explicitement le contexte d'utilisation, c'est-à dire pour le réseau général la "façon " dont il sera utilisé et pour le processus interne le reste du réseau dans son contexte. En fait, ces deux types de réponses peuvent être rassemblés par la notion de contexte si l'on définit pour le premier type un contexte représentant une utilisation quelconque.

Un type particulier de contrainte contextuelle vient de la nature même du système de communication. Nous verrons en effet que des processus indistingables dans un modèle asynchrone peuvent avoir des comportements très différents dans un modèle synchrone.

En résumé, le problème que nous voulons traiter, couvert par le terme de comparaison observationnelle, est celui de la comparaison d'une spécification et d'une implémentation. On considérera une spécification comme la donnée de trois éléments : une description de processus, un contexte utilisateur et un système de communication. On considérera une implémentation comme un autre processus dans le même contexte utilisateur et le même système de communication que la spécification.

L'objectif fixé est de pouvoir remplacer un processus par un autre processus ou par un réseau de processus. On ne s'intéresse donc ici aux processus qu'en tant que boîtes noires qu'on ne veut distinguer que s'ils communiquent différemment, en faisant par conséquent abstraction de leur structure interne. Le problème posé pour définir la notion de comportement de communication tient essentiellement à cette double caractéristique des processus communicants: ils sont en général non déterministes vis-à-vis de leur environnement et compte tenu du mécanisme de synchronisation, seules sont réalisables les communications possibles à partir de l'état courant du processus.

La manière de considérer le non-déterminisme et son influence sur la définition de relations de comparaison observationnelle est en soi un problème, qui a été abordé avec des intuitions différentes. Toutefois, parmi les travaux dans ce domaine, deux approches essentielles se dessinent, très différentes malgré une certaine unanimité par rapport à quelques cas.

Dans ce deuxième chapitre, consacré à la présentation des travaux de référence dans ce domaine nous exposerons d'abord les travaux fondés sur la notion de bisimulation des systèmes de transitions puis ceux qui ont une approche plus globale du comportement des processus non-déterministes.

2.1 Comparaison par bisimulation des systèmes de transitions

Dans cette première section, nous présenterons les calculs s'appuyant sur le même outil mathématique, celui de la bisimulation des systèmes de transitions [Par 81] .

2.1.1 Relation de Bisimulation [Par 81]

Nous allons dans cette première section rappeler la définition formelle de la notion de bisimulation des systèmes de transitions afin d'alléger par la suite les définitions des différentes relations qui seront proposées.

Un système de transitions d'états défini sur un ensemble d'états S et un ensemble d'action Λ est un triplet (S , A , R) où R est une partie de $S \times \Lambda \times S$ qu'on appelle relation de dérivation du système de transitions (on note $s\ R^a\ s'$ si $(s , a , s') \in R$).

Une relation d'équivalence B de $S \times S$ est une bisimulation de $p \in S$ et $q \in S$ ssi elle vérifie les propriétés suivantes:

(*) $(p, q) \in B$

(**) pour tout $a \in A$

(i) $p\ R^a\ p'$ implique $\exists q',\ q\ R^a\ q'$ et $(p', q') \in B$

et (ii) $q\ R^a\ q'$ implique $\exists p',\ p\ R^a\ p'$ et $(p', q') \in B$

Dans toute cette partie, l'expression "système de transitions associé au comportement d'un processus" se rapporte au système de transitions d'états que l'on peut construire ainsi:

- un état est associé à un terme de l'algèbre CCS
- si s est l'état associé au terme t alors il existe une transition (s , a , s') notée $s \rightarrow^a s'$ ssi s' est associé à un terme t' tel que $t = a.t' + u$

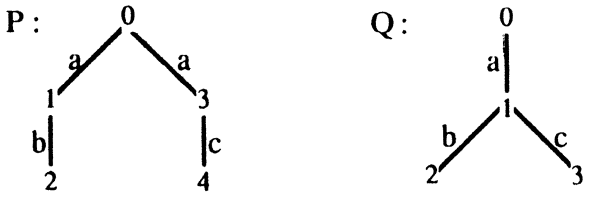
2.1.2 C . C . S . [Mil 80]

Dans cette section, nous présenterons d'abord les principes de CCS et nous discuterons ensuite certains des résultats obtenus par ce calcul.

2.1.2.1 Présentation de CCS

La philosophie de comparaison des processus communicants de CCS se résume en plusieurs points que nous rappellerons en suivant la démarche proposée dans [Mil 80] :

1- La théorie des Automates d'Etats Finis ne permet pas de traiter la comparaison de processus communicants. Lorsqu'un processus décrit par l'arbre P réalise l'événement a , il peut ensuite refuser de communiquer pour réaliser b ou c. Il a donc un comportement différent de celui dont le comportement est décrit par Q qui accepte nécessairement l'une ou l'autre des communications si elle lui est proposée.



2- Ces processus sont différents parce que le processus P, par réalisation de a peut se reconfigurer en deux processus (P1 et P3) dont aucun n'est équivalent à Q1.

3- En conséquence, si un processus peut se reconfigurer par réalisation d'une séquence s en un processus P', il est nécessaire pour qu'un processus Q puisse lui être équivalent qu'il puisse réaliser s et, ce faisant, se reconfigurer en un processus Q' équivalent au processus P'.

4- Les processus peuvent réaliser des transitions internes inobservables. La notion de séquence évoquée précédemment doit se restreindre aux séquences d'événements observables et, par conséquent, toute séquence $< e_1 , \ldots , e_n >$ d'événements observables peut être, pour le processus une séquence d'événements dans laquelle un nombre quelconque d'événements internes a pu avoir lieu entre deux événements observables. La séquence vide correspond donc à l'occurence d'un nombre quelconque de transitions internes (éventuellement nul : dans ce cas, on considère que le processus se reconfigure en lui-même).

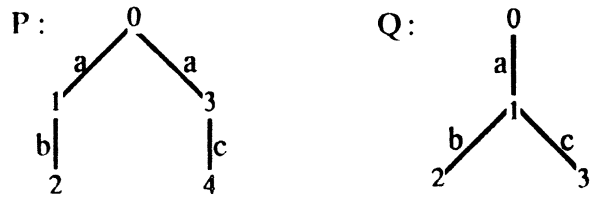
Formellement, on définit une relation de dérivation observationnelle, notée $\Rightarrow^a$, à partir de la relation de dérivation des systèmes de transitions :

$$s \Rightarrow^a s' \quad \text{ssi} \quad s \text{ - } \tau^* \, a \, \tau^* \text{ -> } s'.$$

Deux processus P et Q sont observationnellement équivalents ($P \approx Q$) ssi il existe une relation de bisimulation B entre P et Q dans le système de transitions dont la relation de dérivation est la relation dérivation observationnelle.

Cette relation d'équivalence permet effectivement de distinguer des processus dont les comportements de communication sont différents:

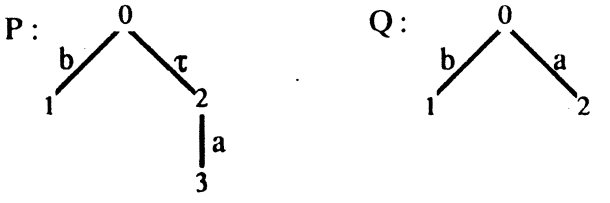
Exemples : Ex 1



Ces deux processus sont distingués puisque, après a, P refusera l'une des communications b ou c alors que Q acceptera les deux communications. Ce fait est reconnu par la relation d'équivalence observationnelle puisque ni P1, ni P3 ne bisimule Q1:

$Q1 \Rightarrow^b Q2 \ , \nexists P' \ P3 \Rightarrow^b P'$ $\qquad$ $Q1 \Rightarrow^c Q3 \ , \nexists P' \ P1 \Rightarrow^c P'$

Ex 2

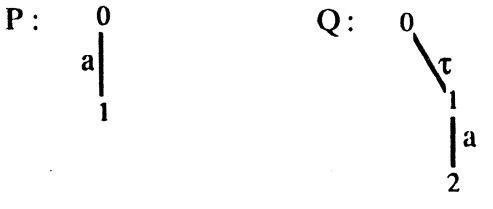


Ces deux processus n'ont pas le même comportement vis-à-vis de leur environnement si celui-ci tente de réaliser l'événement b. Le processus Q acceptera toujours la communication tandis-que le processus P pourra la refuser s'il choisit d'effectuer une transition interne. Ils sont distingués par la relation d'équivalence observationnnelle puisque P par la séquence vide peut se reconfigurer en P mais aussi en P2 qui n'est pas équivalent à Q.
Fin-exemples

Par ailleurs, les transitions internes sont inobservables. Les exemples 3, 4 ,5 montrent l'équivalence de processus illustrant les axiomes suivants de CCS:

A1	$\tau . X = X$	--axiome de l'équivalence observationnelle
A2	$a . \tau . X = a . X$	--axiome de la congruence observationnelle
A3	$a . (X + \tau . Y) = a . (X + \tau . Y) + a . Y$	"
A4	$X + \tau . X = \tau . X$	"

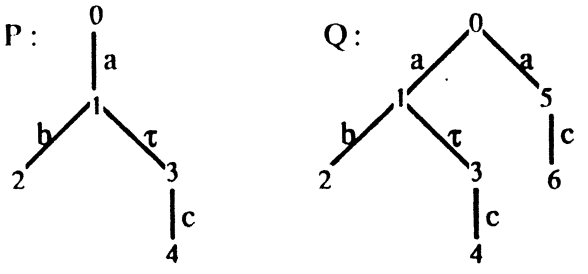
Exemples : Ex 3 (axiome A1)



Ces deux processus P et Q sont équivalents puisque le modèle est a-temporel. En effet l'événement interne aura lieu au bout d'un temps fini dans le processus Q et l'événement a sera toujours accepté, par Q comme par P.

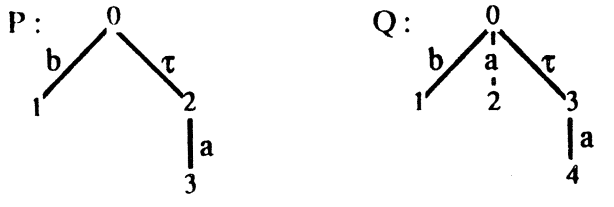
La relation d'équivalence qui contient les deux classes suivantes { P0 , Q0 , Q1 } et { P1 , Q2 } est une relation de bisimulation de systèmes de transitions si la relation de dérivation est la relation de dérivation observationnelle de CCS.

Ex 4 (axiome A3)



Ces deux processus sont équivalents. En effet, le processus P1 accessible dans P par la séquence < a > est équivalent au processus Q1 accessible dans Q par la même séquence et le processus P3 est équivalent aux processus Q3 et Q5.

Ex 5 (Axiome A4)



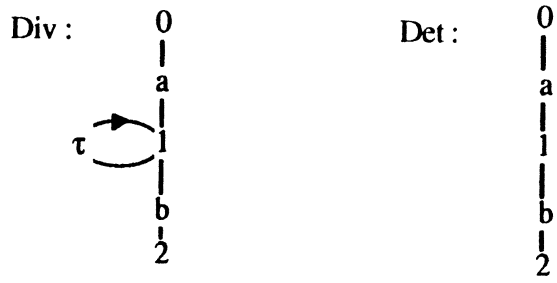
Ces deux processus sont équivalents: par la séquence < a >, P se reconfigure en P3et Q en Q2 ou Q4 qui sont tous deux équivalents à P3.

Fin-exemples

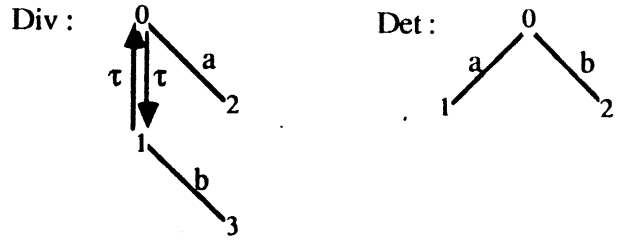
Traitement de la divergence

Cette relation d'équivalence implique une interprétation particulière du phénomène de divergence (chaînes infinies de transitions internes) comme l'ont montré Bergstra et Klop, dans [BK 84]. Cette interprétation est fondée sur le fait qu'on ne distingue pas dans la relation de dérivation observationnelle la possibilité d'avoir des chaînes infinies de transitions internes. Ceci implique l'équivalence des processus recX (τ .X) et nil de la même façon que celle de τ .nil et nil et conduit notamment à l'équivalence des processus présentés dans les exemples suivants.

Exemples



Ces deux processus sont équivalents selon CCS puisque Div1 bisimule Det1. En effet, le processus Div1, seul accessible par a dans Div peut réaliser b et se transformer en Div2, comme le processus Det.



Ces deux processus sont équivalents selon CCS. En effet, le processus Det0 bisimule les processus Div0 et Div1(par la relation de dérivation observationnelle Div0 peut dériver par b vers nil et Div1 par a vers nil), ces deux derniers étant équivalents.
Fin-exemples

Cette équivalence est une congruence pour tous les opérateurs de CCS à l'exclusion de l'opérateur + de choix non déterministe :

* $\tau . a . nil \approx a . nil$ -- Ex 3

* $b . nil + \tau . a . nil \not\approx b . nil + a . nil$ -- Ex 2

Cette propriété de non-congruence de la relation d'équivalence observationnelle pour l'opérateur + a conduit à la recherche d'une relation de congruence observationnelle($p \approx^c q$ ssi pour tout r, $p + r \approx q + r$). La caractérisation de cette relation de congruence [Sor 87] fait apparaître que deux processus ne peuvent être congrus que s'ils sont équivalents et s'ils vérifient des conditions plus stictes sur leur comportement initial, la non-congruence de la relation d'équivalence étant essentiellement dûe au fait qu'un processus stable et un processus instable peuvent être équivalents:

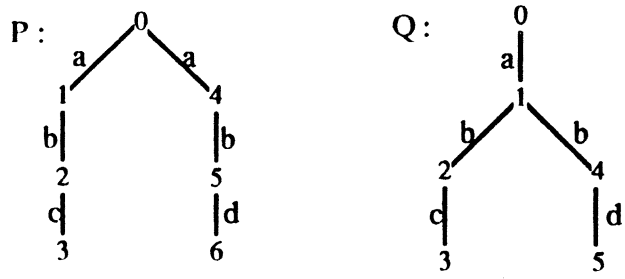
$$p \approx^c q \Leftrightarrow p \approx q$$
$$\wedge \quad p - \tau \rightarrow p' \Rightarrow \exists q' \quad q - \tau^+ \rightarrow q' \wedge p' \approx q'$$
$$\wedge \quad q - \tau \rightarrow q' \Rightarrow \exists p' \quad p - \tau^+ \rightarrow p' \wedge p' \approx q'$$

2.1.2.2 Discussion des résultats de CCS

Cette relation de congruence observationnelle fait du calcul CCS un calcul satisfaisant en ce sens que des équivalences peuvent être prouvées en raisonnant uniquement sur les termes de l'algèbre et que si deux processus sont reconnus équivalents alors ils sont indistingables par un observateur quelconque.

On peut toutefois discuter le bien-fondé de la philosophie proposée dans CCS et nous allons montrer à travers quelques exemples que tous les processus indistingables par un observateur quelconque dans le système de communication de CCS ne sont pas reconnus équivalents.

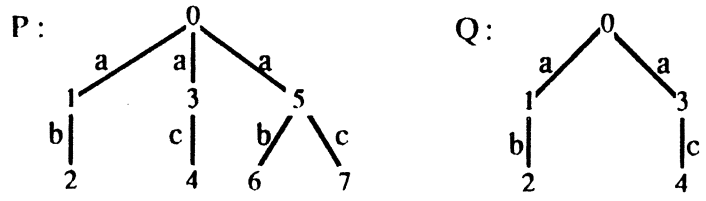
Exemples : Ex 6



Ces processus sont distingués par CCS puisque ni P1, ni P4 ne bisimule Q1.

Cet exemple, proposé dans [Dar 82] pose la question suivante : est il légitime de tenir compte du moment où les processus effectuent des choix non déterministes pour les distinguer? Il semble que la réponse à cette question soit non puisqu'un observateur de ces processus aura les mêmes réponses qu'il communique avec P ou avec Q : initialement, les deux processus accepteront de réaliser a, ensuite ils accepteront tous deux de réaliser b et enfin ils n'accepteront que c ou que d selon des choix qu'ils auront fait antérieurement mais sans que l'observateur en ait eu connaissance.

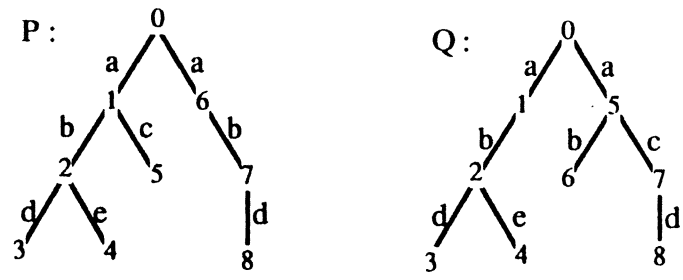
Ex 7



Ces processus sont distingués par la relation d'équivalence observationnelle de CCS puisque ni Q1, ni Q3 ne bisimule P5.

On peut cependant se demander s'il est légitime de prendre en compte le fait que le processus P puisse se reconfigurer en P5 et soit prêt à réaliser b et c pour distinguer ces processus. Ce pouvoir de l'état P5 qui permettrait de distinguer Q de P si P5 était le seul état accessible par a n'est pas une propriété du processus P lorsqu'il a réalisé a puisqu'il peut également atteindre les états P1 et P3 c'est à dire refuser, comme Q, de réaliser b s'il atteint P3 et c s'il atteint P1.

Ex 8



Ces processus P et Q sont distingués par CCS puisque ni P1, ni P6 ne bisimule Q1 ou Q5. Cependant, la seule distinction qui existe entre eux vient de la possibilité éventuelle dans les deux cas de réaliser c après a. Si cet événement a lieu alors l'observation des processus est terminée et aucun des deux processus n'a eu un comportement que l'autre n'aurait pas pû avoir. Si l'événement b a lieu alors la seule conclusion que l'on peut tirer de cette observation est que le processus était dans un état à partir duquel b était possible et donc qu'il se trouve en P2 ou P7 dans le cas de P, dans Q2 ou Q7 dans le cas de Q. Par conséquent, dans un cas comme dans l'autre, on pourra toujours ensuite réaliser d tandis que e ne sera qu'éventuellement possible.

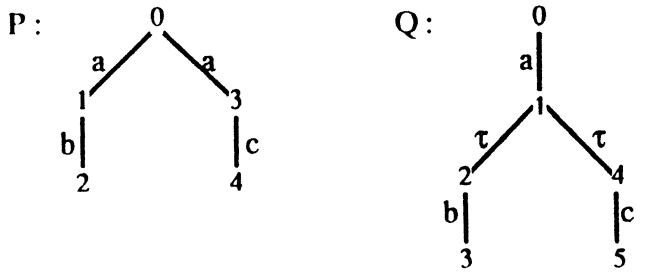
Si l'on se fonde uniquement sur des critères d'observabilité, ces deux processus devraient donc être équivalents.
Fin-exemples

Nous n'avons présenté jusqu'à présent que des exemples sur des processus rigides (processus qui ne peuvent se reconfigurer qu'en des processus stables, c'est à dire dont le comportement ne contient aucune transition interne).

Cependant, l'interprétation des processus non rigides pose aussi des problèmes :

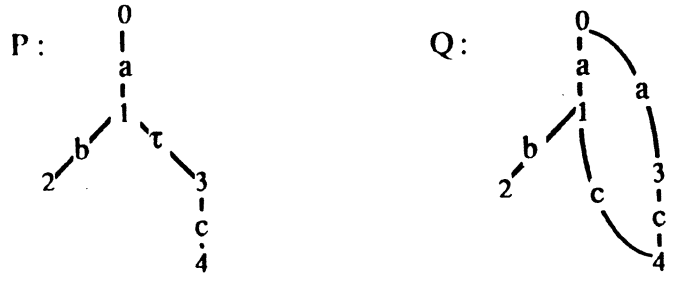
Exemples

Ex 9

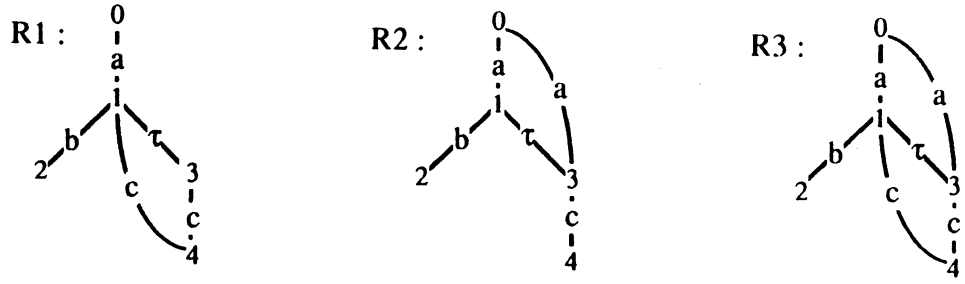


Ces deux processus ne sont pas équivalents d'après CCS puisque ni P1, ni P3 ne bisimule Q1. Cet exemple montre que les relations proposées sont trop fortes lorqu'il existe des transitions internes puisque ces processus ont manifestement le même comportement dans un modèle asynchrone. En effet ces deux processus doivent effectuer un choix non-déterministe pour pouvoir réaliser une nouvelle communication après a. Seule l'expression de ce choix est différente: accessibilité de deux états par réalisation de a dans P, réalisation de a puis choix entre deux transitions internes pour Q.

Ex 10



Ces deux processus ne sont pas équivalents d'après la relation d'équivalence observationnelle de CCS. Nous avons vu en effet (Ex 2) que le processus P1 n'était pas équivalent au processus Q1. Toutefois, il faut remarquer que le processus P est équivalent aux trois processus R1 (axiome A4), R2 (axiome A2), R3 (axiomes A3 et A4) suivants et que les processus R3 et Q ne diffèrent que par l'existence d'une transition interne entre les états 1 et 3 de R3.



Dans les deux cas, il est possible d'atteindre par a un état à partir duquel le seul événement observable soit c et un autre état à partir duquel les événements b et c soient possibles. Sans hypothèse de nature temporelle, notamment sur la durée des événements, il est impossible de distinguer par observation ces deux processus. De plus, si l'on faisait de telles hypothèses, il faudrait en général rejeter l'équivalence entre les processus P et R1, R2, R3 ainsi que l'axiome A1 de l'équivalence observationnelle.
Fin- exemples

Des exemples 6 , 7 , 8 , 9 et 10 que nous venons de présenter, nous pouvons conclure que les relations proposées dans CCS sont trop fortes si l'on étudie uniquement le comportement de communication des processus en modèle asynchrone. Cela tient au point trois du raisonnement initial qui parcellise le comportement d'un processus en transformant un processus non-déterministe en plusieurs processus. En effet, chacune des raisons invoquées pour dire que les processus cités dans les exemples étaient indistingables faisait intervenir une perception globale du comportement des processus. Au contraire, pour chacun de ces processus, CCS fait appel à des propriétés locales comme si celles-ci étaient conservées malgré le non-déterminisme.

2.1.3 Modèle synchrone : SCCS [Mil 83]

Le calcul SCCS [Mil 83] est une extension du calcul CCS [Mil 80] . La philosophie de comparaison des processus est celle de CCS adaptée de façon à prendre en compte les contraintes d'un modèle synchrone.

Ces contraintes sont liées à l'opérateur de produit synchrone qui oblige les processus à n'évoluer que simultanément et ne portent que sur les transitions internes.

Nous avons vu sur plusieurs exemples que l'interprétation des transitions internes de CCS était trop contraignante dans un modèle asynchrone. Elle est cependant trop faible pour pouvoir être reprise dans un modèle synchrone.

En effet, dans un modèle synchrone, lorsqu'un processus d'un réseau peut faire une transition interne, le reste du réseau peut effectuer une transition; au contraire, si un processus ne peut pas effectuer de transition le reste du réseau est bloqué. De ce fait, les processus nil et τ . nil ne sont pas équivalents dans un modèle synchrone (indépendamment de toute considération liée à l'opérateur +):

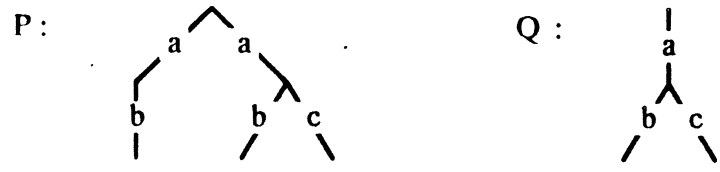
τ . nil	×	a . nil	=	a . nil
nil	×	a . nil	=	nil

La relation de dérivation observationnelle synchrone est donc définie comme la relation de dérivation des systèmes de transitions associés aux comportements des processus. Deux processus sont donc équivalents en modèle synchrone ssi les systèmes de transitions associés à leur comportement sont bisimulables.

2.1.4 Prise en compte de l'environnement [Lar 85]

Les calculs CCS et SCCS ne diffèrent que par la nature du système de synchronisation: dans ces deux calculs, les relations établies sont valides dans un environnement quelconque.

De ce fait, les processus P et Q suivants



ne peuvent pas être reconnus équivalents puisque le processus Q acceptera toujours de communiquer avec un environnement qui ne tente que c après a alors que le processus P pourra éventuellement refuser la communication. Cependant, il existe de nombreux réseaux dans lesquels on pourrait indifféremment utiliser P ou Q : un réseau qui n'essaie jamais de communiquer après a ou qui, après a n'essaie de communiquer que pour réaliser b par exemple.

La théorie proposée ici a pour objectif de raffiner la relation d'équivalence de CCS en la paramétrant par un environnement e. Formellement, les environnements sont décrits par des systèmes de transitions dont les actions "consomment" les événements "produits" par les processus.

La définition proposée pour paramétrer l'équivalence par un environnement est la suivante:

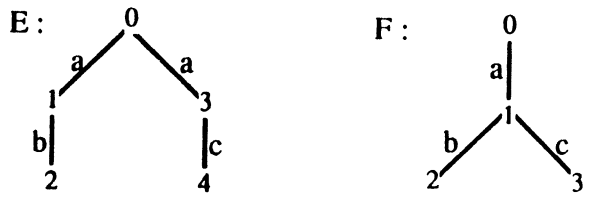
Soit SE = (E , A , $\Rightarrow$) un système de transitions d'environnements , où E est l'ensemble des environnements, A l'ensemble des événements et $\Rightarrow$ une relation de E $\times$ A $\times$ E

Une bisimulation R paramétrée par SE est une famille de relations binaires $R^e \subseteq P^2$ où P est un ensemble de processus telle que si p R^e q et e $\Rightarrow^a$ f alors

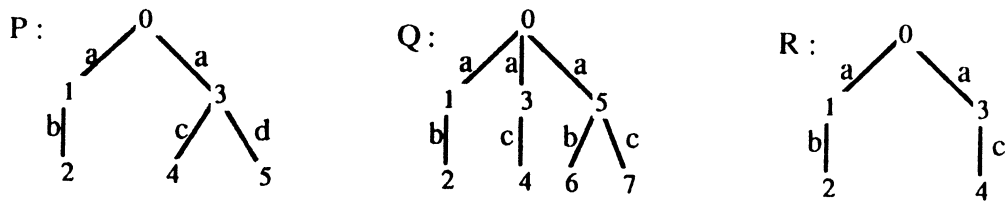
(i) p - a -> p' implique $\exists$ q' q - a -> q' et p' R^f q'
et (ii) q - a -> q' implique $\exists$ p' p - a -> p' et p' R^f q'

Ce type de relation permet essentiellement de ne pas tenir compte des alternatives qui distinguent deux processus lorsque celles-ci ne sont pas considérées par l'environnement. De plus, elle permet parfois d'ignorer certaines propriétés locales.

Exemple :
Soient E et F deux systèmes de transition d'environnement



P , Q , R les processus décrits par les arbres suivants



P, Q , R sont équivalents dans l'environnement E0. On a en effet les relations :

P1 $\approx_{E1}$ Q1 $\approx_{E1}$ Q5 $\approx_{E1}$ R1
P3 $\approx_{E3}$ Q3 $\approx_{E3}$ Q5 $\approx_{E3}$ R3

P et R sont équivalents dans l'environnement F0 puisqu'on a les relations:

P1 $\approx_{F1}$ R1 et P3 $\approx_{F1}$ R3

Par contre, le processus Q ne leur est pas équivalent dans l'environnement F0 puisqu'il n'existe ni dans P, ni dans R de processus accessible par a équivalent à Q5 dans l'environnement F1.
Fin-exemple

Une relation est définie entre les contextes qui les ordonne en fonction de leur pouvoir discriminant (l'environnement le moins discriminant est l'environnement nil et le plus discriminant est tel que pour tout a $e \Rightarrow^a e$).

En général un environnement e est plus discriminant qu'un environnement f si f est une simulation de e c'est - à - dire s'il existe une relation " $<$" entre e et f ($e < f$) telle que
$e \Rightarrow^a e'$ implique $\exists f'$ $f \Rightarrow^a f'$ et $e' < f'$

Plusieurs problèmes sont posés par la façon dont est traitée la prise en compte du contexte selon cette théorie :

(i) Il faut remarquer que les transitions internes ne peuvent pas être élément de l'ensemble Λ des événements. Il suffit de considérer l'exemple suivant pour s'en rendre compte:

E = a . nil
P = a . nil + b . nil Q = a . nil + c . nil

Les processus P et Q sont équivalents dans le contexte E . Toutefois cette équivalence serait inadmissible si l'un des événements b ou c était une transition interne, puisqu'alors la communication avec E ne serait plus certaine.

On pourrait envisager, pour prendre en compte les transitions internes, d'utiliser la relation de dérivation observationnelle entre les processus et d'étendre la relation de transition entre les environnements avec : pour tout e, $e \Rightarrow^{\varepsilon} e$. De ce fait on considérerait qu'un environnement par la séquence vide se reconfigure en lui-même alors qu'un processus se reconfigure en l'un des processus accessibles par une séquence de transitions internes de longueur quelconque.

Avec cette nouvelle définition, deux processus ne pourraient être équivalents que si leurs processus dérivés par τ^* étaient considérés dans l'environnement initial.

Les processus P = a . nil + b . nil et Q = a . nil + τ . nil ne seraient pas équivalents dans l'environnement E = a . nil puisque les processus P et nil ne le sont pas.

(ii) L'utilisation de la bisimulation, comme outil fondamental, est discutable indépendamment du problème posé par les transitions internes. En effet, à partir du dernier exemple présenté, il apparaît que les processus Q0 et R0 ne seraient pas considérés comme équivalents dans l'environnement

$$F' = a \ . \ (b \ . \ d \ . nil \ + \ c \ . \ e \ . \ nil)$$

Il est également très facile de voir que les processus F' || P et F' || Q sont observationnellement équivalents alors que l'objectif initial était précisément de définir des relations permettant d'obtenir de tels résultats. En fait la bisimulation paramétrée distingue des processus en fonction du nombre d'alternatives qui s'offrent au processus final pour effectuer un choix non déterministe (dans le cas de QO, après réalisation de a, le réseau aura un choix à faire entre b et c; au contraire dans le cas de P0, le réseau aura fait ce choix initialement).

2.1.5 Equivalences selon des critères d'observation [Bou 84]

Cet article propose de paramétrer les relations de comparaison observationnelle par des "critères d'observation " . Un critère d'observation y est défini par un ensemble d' "observables ", un observable étant un ensemble de séquences d'actions considérées a priori comme équivalentes.

Il est par exemple proposé d'introduire la perception de la durée de réalisation des transitions internes (notées ici 1 car elles sont éléments neutre du monoïde d'action M) par le critère suivant:

$$E = \{ \{ 1^n . a \} \mid n \in \mathbf{N} \text{ et } a \in M - \{ 1 \} \}$$

qui distingue toutes les séquences d'actions qui diffèrent soient par l'événement a, soit par la valeur n du nombre de transitions internes qui précèdent l'occurence de a.

Les critères d'observation de l'équivalence observationnelle (CCS) et de la congruence forte (SCCS) sont respectivement définis par :

$E_{cf} = \{ \{ a \} \mid a \in M \}$

Toutes les actions sont distinguées , y compris l'action 1 puisqu'en système synchrone son occurence est observable.

$E_{eo} = \{ \pi^{-1}(\omega) \mid \omega \in (M - \{ 1 \})^* \}$

où $\pi(1) = \varepsilon$ et $\pi(a) = < a >$ pour $a \in M - \{ 1 \}$

Chaque observable définit comme équivalentes à une séquence observable $< e_1 \ldots e_n >$ toutes les séquences $< \tau^{k0}, e_1, \tau^{k1} \ldots e_n, \tau^{kn} >$.

On peut aussi définir comme équivalentes une action a et une séquence d'actions $< b_1 \ \ldots \ b_k >$ par l'observable suivant $\{ a, < b_1 \ \ldots \ b_k > \}$.

Tout critère d'observation E induit une relation de dérivation sur les systèmes de transitions, définie à partir des observables du critère d'observation et non plus à partir des actions: la transition d'un processus p vers un processus p' se déduit de la relation de dérivation des systèmes de transition:

$$\forall e \in E, \quad p \Rightarrow^e p' \text{ ssi } \exists u \in e \quad p \ -u-> p'$$

Un critère d'observation E induit alors une relation d'équivalence telle que deux processus sont équivalents modulo E ssi les systèmes de transitions sont bisimulables modulo la relation de dérivation induite par E:

$p \approx_E q$ ssi

(i) $p \Rightarrow^e p'$ implique $\exists q'$ $q \Rightarrow^e q'$ et $p' \approx_E q'$

et (ii) $q \Rightarrow^e q'$ implique $\exists p'$ $p \Rightarrow^e p'$ et $p' \approx_E q'$

L'approche proposée ici permet d'envisager plusieurs applications intéressantes:

Nous avons repris l'exemple proposé dans l'article qui permet de définir comme équivalentes une action et une séquence d'actions. Ce point de vue peut être utile si l'on envisage une implémentation du rendez-vous multiple tel qu'il est proposé dans FP^2 sur une machine parallèle telle que le transputer. S'il est en effet indéniable que le rendez-vous multiple soit un apport important au niveau de l'expression de la synchronisation, il n'en reste pas moins vrai que les transputers ne communiquent que par rendez-vous à deux. Il est donc évident qu'un événement constitué de plusieurs communications entre des processus implantés sur des transputers différents devra être simulé par un protocole constitué d'une séquence de communications. On peut penser que la preuve de la validité d'un tel protocole pourrait être facilitée si l'on était capable de prendre en compte des critères tels que celui qui est proposé.

L'autre critère que nous avons présenté permet aussi d'envisager la comparaison des processus d'un point de vue original. En effet, les notions liées à la durée des actions sont d'une façon générale absentes des modèles actuels. On peut envisager de définir des critères d'observation qui distinguerait les actions à la fois par leur nom et par leur durée.

Toutefois cette approche mériterait d'être approfondie, notamment pour étudier l'ensemble des critères que l'on peut réellement définir. En effet, on peut ici définir l'équivalence d'une séquence d'événements et d'un événement, la séquence d'événements étant divisible alors que l'événement ne l'est pas.

De ce fait, l'observable qui définit l'équivalence des séquences <a , b > et < b , a > , s'il est le seul observable du critère d'observation conduit à l'équivalence des processus a . b . nil et b . a . nil + a. nil, ce qui d'une façon générale ne répond pas à ce que l'on peut souhaiter puisque l'événement a peut avoir lieu sans être suivi de l'événement b.

Si l'on ajoute à ce critère l'ensemble des observables qui distinguent les actions, alors les processus a. b . nil et b . a . nil ne seront pas équivalents ce qui correspond précisément à ce que l'on voulait autoriser en introduisant l'observable initial.

2.1.6 Conclusion

Les résultats que nous avons présentés ici abordent dans leur ensemble la plupart des questions liées à la comparaison observationnelle des processus communicants. Ils ont en commun d'utiliser comme principe de base la bisimulation des systèmes de transitions.

D'une manière générale, les résultats ne reflètent pas fidèlement la notion de comportement de communication des processus: Nous avons vu sur de nombreux exemples que des processus indistingables par un environnement quelconque pouvaient ne pas être équivalents. De plus les relations d'équivalence qui sont définies ne sont pas induites par un préordre. On pourrait, cependant penser que la relation d'équivalence se déduit d'une relation de préordre reflétant une notion de réalisation correcte (P est équivalent à Q si P peut remplacer Q pour un contexte quelconque et réciproquement).

Toutefois ces résultats ne sont, en général (pour des processus non divergents), discutables que parce qu'ils établissent trop de distinction entre les processus: en effet, deux processus reconnus équivalents sont indistingables par un observateur quelconque. De ce fait, de nombreux systèmes sont actuellement développés pour rendre opérationnels les résultats obtenus par ces différents calculs (ECRIN développé à partir de MEIJE [MV 86] , VENUS développé au LGI à partir de CCS [Sor 87], SCAN [Naj 85]).

2.2 Comparaison globale des processus

L'approche par bisimulation des systèmes de transition associés au comportement des processus s'avérant inadaptée, plusieurs calculs ont été développés pour définir autrement la sémantique observable des processus communicants non-déterministes.

Les problèmes posés par les résultats présentés au cours de la première partie viennent essentiellement du fait qu'on y considère qu'après avoir réalisé une séquence d'événements, un processus peut se transformer en plusieurs processus: qu' il aura, non pas le comportement d'un processus initialement non-déterministe, mais celui d'un processus initialement déterministe, parmi plusieurs possibles. Au contraire, tous les travaux qui seront présentés maintenant adoptent une démarche différente en considérant qu'un processus, déterministe ou non, reste, tout au cours de son évolution, un seul processus. De ce fait, les capacités de communication d'un processus, si elles sont fonctions de l'ensemble des comportements qu'il est susceptible d'avoir, ne se réduisent pas à une somme de comportements comme cela est le cas lorsqu'on considère l'approche par bisimulation.

Parmi les travaux qui ont abordé le problème de la comparaison observationnelle sous cet angle, on peut citer ceux de Darondeau sur les processus à comportement finis [Dar 82], ceux de Kennaway [Ken 80] à partir d'un modèle fondé sur la notion de machine déterministe, ceux de Hennessy et De Nicola à partir des algèbres CCS et SCCS [HN 84] , ceux de Brookes [Bro 83], [BHR 84] ou Olderog [Old 84]à partir du modèle TCSP ou encore ceux que nous avons développés à partir du langage FP2 (chapitres 3 et 4) .

Malgré la diversité des modèles de description et des approches intuitives différentes, il est remarquable de constater que tous ces travaux conduisent à des résultats similaires pour l'essentiel, c'est à dire si les processus comparés sont non divergents et ont le même langage de traces.

2.2.1 Equivalences par test : [HN 84], [Hen 83], [Hen 85]

La comparaison des processus est définie à partir de la capacité des processus à satisfaire certains tests définis par un observateur. Un observateur est une entité capable de communiquer avec les processus selon le mécanisme de communication du système (comme les processus entre eux) et de décider, lorsqu'il atteint certains états, que le test de l'expérimentation est satisfaisant.

Puisque les processus ont des comportements en général non-déterministes, un observateur quelconque pourra donc faire d'un même processus plusieurs expérimentations qui aboutiront tantôt à un succès, tantôt à un échec (si le processus est bloqué dans un état qui ne permet pas à l'observateur d'atteindre un état de succès ou s'il boucle sur un état qui ne correspond pas à un succès pour l'observateur). Pour un observateur donné, l'ensemble des expérimentations d'un processus peut donc soit toujours conduire à un succès, soit toujours conduire à un échec, soit conduire parfois à un succès, parfois à un échec.

2.2.1.1 Définitions

Ces principes ont été formalisés dans [Hen 83] à partir des calcul CCS et SCCS:

* Les processus sont soit décrits par des termes de l'algèbre CCS, soit le processus indéfini noté Ω.

* Les observateurs sont décrits par des termes de l'algèbre CCS sur un alphabet augmenté d'un symbole spécial ω chargé de rapporter le succès de l'expérimentation. On dira qu'un observateur est dans un état de succès s'il peut produire ω, dans un état d'échec sinon.

* Une observation d'un processus p par un observateur o (notée $p \mid o$) est définie par la composition parallèle des deux termes p et o sur laquelle on fait l'abstraction de tous les ports de communication. Suivant le cas, l'opérateur de composition utilisé est l'opérateur de composition parallèle asynchrone ou de produit synchrone.

* Une expérimentation de p par o est alors définie par une suite

  $$p \mid o \ -\tau\text{->} \ p_1 \mid o_1 \ \ldots$$

  obtenue par développement de l'observation telle que si cette suite est finie et d'élément terminal $p_k \mid o_k$ alors il n'existe pas de couple (p', e') tel que $p_k \mid o_k \ -\tau\text{->} \ p' \mid o'$.

* Une expérimentation de p par o est un succès ssi
  1) L'observateur peut atteindre un état de succès (produire ω):

  $$\exists n \quad p \mid o \ -\tau\text{->}^* \ p_n \mid o_n \ \text{ et } \ o_n \ -\omega\text{->}$$

  2) Il est certain qu'un état de succès pourra être atteint:

  $$\text{si } (p_k \mid o_k)\uparrow \text{ alors } \exists k' \leq k \quad o_{k'} \ -\omega\text{->}$$

  où le prédicat $\uparrow$ est caractéristique des états partiellement définis tels que les états de divergence ou les états à branchement infini.

* Le résultat d'une expérimentation de p par *o* est donc une valeur dans l'ensemble Résultats={ succès , échec } et celui d'une observation noté Obs(p , *o*) dans l'ensemble $\mathbf{P}$(Résultats) - { ø }. On dit que l'observation de p par *o* "peut" conduire à un résultat *res* si *res* ∈ Obs(p , *o*) et que l'observation de p par *o* " doit " conduire à un résultat *res* si {*res* }= Obs(p , *o*).

2.2.1.2 Trois critères de comparaison de processus

Trois critères de comparaison des processus, fondés sur un ordre dans l'ensemble des résultats d'observation, sont proposés. Des relations de préordre sont définies pour un observateur *o* (notées $\leq_i^o$, pour i = 1 , 2 , 3) et étendues à un ensemble d'observateurs *O* si elles sont vérifiées pour chaque observateur de *O*. Sans précision de l'ensemble *O*, ces relations s'entendent pour l'ensemble des observateurs le plus général(l'ensemble des termes de l'algèbre).

2.2.1.2.1 Le critère "peut"

Le premier type de relations, indicées par 3, ne repose que sur la capacité des processus à satisfaire au moins une fois les observateurs, c'est à dire sur leur capacité à réaliser des séquences d'événements qui permettent à l'observateur d'atteindre des états de succès.

$p \leq_3^o q$: Si l'observation de p par *o* peut conduire à un succès alors l'observation de q par *o* peut conduire à un succès:

succès ∈ Obs(p , *o*) <u>implique</u> succès ∈ Obs(q , *o*)

Les relations $\leq_3$ et $\approx_3$ comparent les processus en fonction de leurs ensembles de traces (en modèle asynchrone une séquence d'événements observables $t = < e_1 \ldots e_n >$ est une trace de P s'il existe P' tel que P = t => P' ; en modèle synchrone s'il existe P' tel que $P \text{ — } t \rightarrow P'$, les relations =t => et —t → sont les relations de dérivation définies dans CCS). Elles sont assimilables aux relations de comparaison d'automates.

$p \leq_3 q \Leftrightarrow$ Traces (p) ⊆ Traces (q)

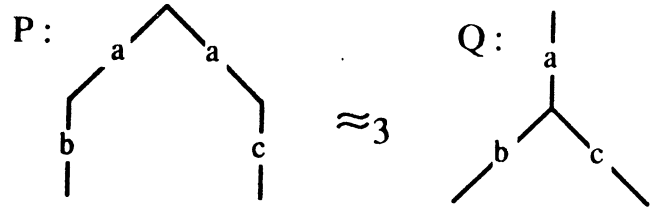
Puisque le seul élément de comparaison est la possibilité de réaliser une séquence d'événements, deux observations qui conduisent l'une toujours à un succès, l'autre à un succès ou éventuellement à un échec seront considérées comme équivalentes.

Au contraire, lorsqu'une expérimentation ne peut conduire qu'à un échec, cette trace n'est pas réalisable par le processus. Par conséquent, ces relations sont fondées sur l'ordre suivant dans l'ensemble des résultats d'observation:

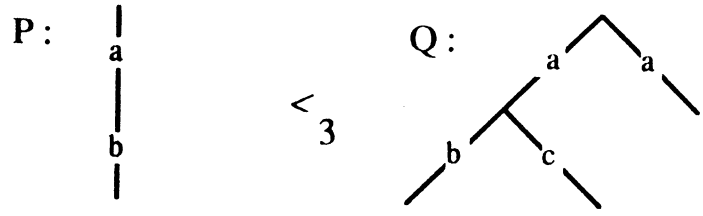
{échec} < { échec, succès } = { succès }

$p <_3^o q \Leftrightarrow obs(p, o) = \{ \text{échec} \}$ et $obs(q, o) \supseteq \{ \text{succès} \}$

Exemple :



Face à tout observateur, ces deux processus sont équivalents par la relation $\approx_3$. En effet, tout observateur qui peut atteindre un état de succès par l'une des séquences suivantes : < >, < a >, < a . b >, < a. c > pourra avoir avec ces processus une expérimentation le conduisant à un état de succès. Dans le cas où l'observateur n'atteint pas un état de succès avec l'une de ces traces alors il ne pourra avoir avec aucun des deux processus d'expérimentation le conduisant à un état de succès.



L'observateur décrit par a . c . ω . nil, peut avoir avec le processus Q une expérimentation qui soit un succès, ce qui est impossible avec le processus P.
Fin-exemple

2.2.1.2.2 Le critère "doit"

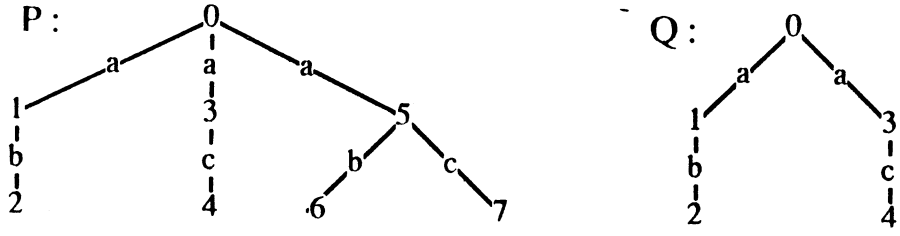
Ce critère de comparaison dont les relations sont indicées par 2, est fondé sur la capacité des processus à toujours satisfaire les observateurs (toute expérimentation est un succès).

$p \leq_2^o q$: Si l'observation de p par *o* conduit toujours à un succès alors l'observation de q par *o* conduit toujours à un succès.

Obs(p , *o*) = { succès } implique Obs(q , *o*) = { succès }

Ce critère de comparaison permet essentiellement de prendre en compte l'influence du non déterminisme et sur plusieurs exemples, nous montrerons comment ces relations se distinguent de CCS.

Exemples: Ex 1 (Exemple 7 de CCS)



$o1$ = a . b . ω . nil — Obs(P , $o1$) = Obs(Q , $o1$) = { échec, succès }

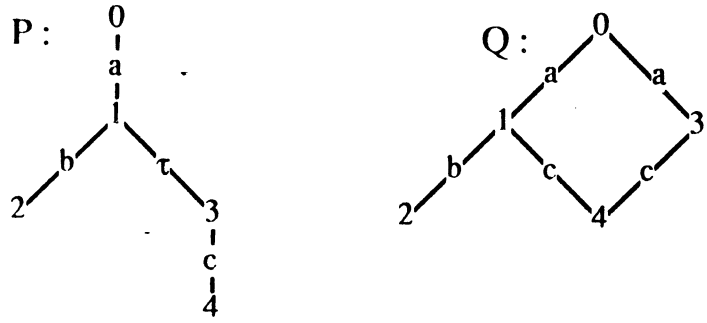
$o2$ = a . c . ω . nil — Obs(P , $o2$) = Obs(Q , $o2$) = { échec, succès }

$o3$ = a . (b . ω . nil + c . ω . nil) — Obs(P , $o3$) = Obs(Q , $o3$) = { succès }

$o4$ = a . b . ω . nil + a . c . ω . nil — Obs(P , $o4$) = Obs(Q , $o4$) = { échec, succès }

Face à ces quatre observateurs "délicats" les processus fournissent les mêmes résultats. Plus généralement, on a $P \approx_2 Q$.

Ex 2



$o1$ = a . b . ω . nil — Obs(P , $o1$) = Obs(Q , $o1$) = { échec, succès }

$o2$ = a . c . ω . nil

$o2$ | P - τ -> c . ω . nil | (b.nil + τ . c . nil) - τ -> c . ω . nil | c . nil
- τ -> ω . nil | nil - ω ->

$o2$ | Q - τ -> c . ω . nil | b . nil + c . nil - τ -> ω . nil | nil - ω ->
ou - τ -> c . ω . nil | c . nil - τ -> ω . nil | nil - ω ->

d'où Obs(P , $o2$) = Obs(Q , $o2$) = { succès }

Fin-exemples

Nous n'avons examiné que deux des exemples présentés dans la section précédente (un sur l'équivalence de processus rigides, un sur l'équivalence entre processus rigides et processus non rigides). Cependant, les exemples 6, 7, 8, 9 et 10 que nous avons étudiés conduisent à l'équivalence selon ce critère.

Ces relations ont été axiomatisées dans [HN 84]. Sans en donner la liste exhaustive, nous mentionnerons quelques axiomes qui mettent en évidence l'originalité de ces résultats par rapport aux précédents:

A1: $\mu . X + \mu . Y = \mu . (\tau . X + \tau . Y)$
Ceci était l'exemple 9 qui montrait que les deux formes de non-déterminisme ne sont pas équivalentes selon CCS

A2: $\mu . X + \tau . (\mu . Y + Z) = \tau . (\mu . X + \mu . Y + Z)$
Les relations seront , à l'évidence moins fines que celles de CCS puiqu'il est, en général, exclu de trouver dans le terme de droite un équivalent à $\mu . Y + Z$ accessible par la séquence vide.

Un processus doit satisfaire un observateur si toute expérimentation est un succès, c'est à dire si le résultat de l'observation est { succès }. Ces relations sont donc fondées sur un ordre dans l'ensemble des résultats d'observation qui assimile échec et possibilité d'échec:

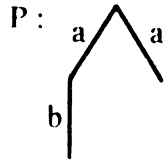
{échec} = { échec, succès } < { succès }

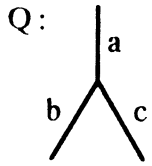
$p \not\leq_2^o q \Leftrightarrow obs(p , o) = \{ \text{succès} \}$ et $obs(q , o) \supseteq \{ \text{échec} \}$

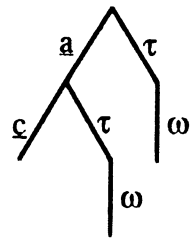
Les observateurs sont des termes bien définis de l'algèbre. Comme l'a montré Brookes dans [Bro 83], certains observateurs (notés *non(t , e)*) définis à partir d'une trace t sont nécessairement satisfaits par un processus P si t n'est pas une trace de P. Pour une séquence t et un événement e, un observateur non(t , e) peut être défini qui n'est satisfait que par les processus qui n'ont pas t .e pour trace. Ces processus non(t , e) atteignent un état de succès pour toute trace u préfixe de t et un état d'échec pour t . e :

$non (< > , e) = \tau . \omega . nil + \underline{e} . nil$
$non (a . t , e) = \tau . \omega . nil + \underline{a} . non (t , e)$

Exemple 3

P : 

Q : 

o : 

$o = \tau.\omega.nil + \underline{a} . (\tau.\omega.nil + \underline{c}.nil)$

P " doit satisfaire " o : $P \mid o = \tau . \omega . nil + \tau . \tau . \omega . nil$

Q " doit satisfaire " o : $Q \mid o = \tau . \omega . nil + \tau . (\tau . \omega . nil + \tau . nil)$

Fin-exemple

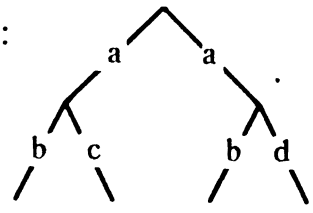
Des observateurs not (t , X), où X est un ensemble d'événements, ont été définis dans [Bro 83] qui testent l'influence du non-déterminisme des processus ayant réalisé une séquence t : ces observateurs, à partir desquels nous avons définis les observateurs non(t , e), sont satisfaits par tous les processus qui ne peuvent pas réaliser t ou par les processus qui peuvent réaliser t et ensuite accepter toujours de communiquer pour réaliser l'un des événements de X. La définition des observateurs "not" que nous donnons n'est pas exactement celle donnée dans [Bro 83]. Il est en effet nécessaire d'ajouter dans le premier terme de la somme de (2) un préfixage par τ, faute de quoi tout expérimentation serait nécessairement un succès.

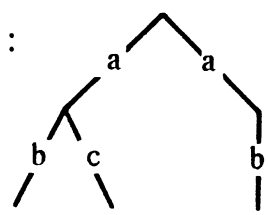
$$not (< > , X) = \Sigma_{x \in X} \underline{x} . \omega .nil + nil \qquad (1)$$

$$not (a . t , X) = \tau . \omega . nil + \underline{a} . not (t , X) \qquad (2)$$

donc $P \leq_2 Q \Rightarrow traces(Q) \subseteq traces(P)$

Exemple 4

P : 

Q : 

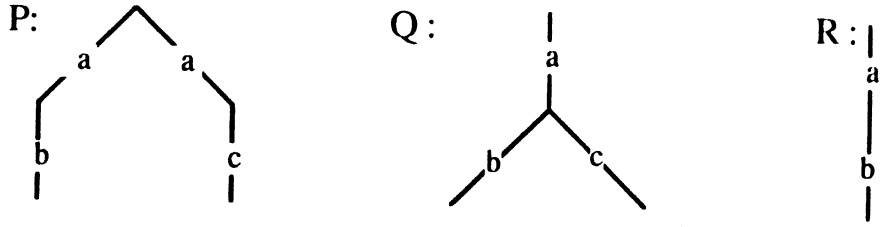
Ces deux processus ne sont pas liés par la relation de préordre $\leq_2$:

< a , d > est une trace de P et n'est pas une trace de Q donc $Q \not\leq_2 P$:

Q "doit" satisfaire $\underline{a} . (\tau . \omega + \underline{d})$; P ne "doit" pas satisfaire $\underline{a} . (\tau . \omega + \underline{d})$ { c , d } permet à P' d'évoluer après $\underline{a}$ alors que Q' peut rester bloqué donc $P \not\leq_2 Q$

P "doit" satisfaire $\underline{a} . (\underline{c} . \omega + \underline{d} . \omega)$; Q "doit satisfaire" $\underline{a} . (\underline{c} . \omega + \underline{d} . \omega)$

Exemple 5



Le processus P est inférieur aux processus Q et R. En effet, il présente le même ensemble de traces que le processus Q, mais, étant moins déterministe, il existe des observateurs que Q "doit" satisfaire alors que P ne "doit" pas les satisfaire. Par ailleurs, l'ensemble de traces de P est strictement inclus dans celui de R et tout observateur satisfait par P est satisfait par R.

Par contre, les processus Q et R ne sont pas comparables, puisque Q doit satisfaire $\underline{a} \,.\, \underline{c}.\, \omega$ et que R doit satisfaire $\underline{a} \,.\, (\underline{c} + \omega)$.

Fin-exemples

En conclusion, sur le domaine des processus non divergents la relation $\leq_2$ est équivalente à celle définie dans TCSP (voir section suivante) et peut s'exprimer uniquement à l'aide des observateurs not .

2.2.1.2.3 Le critère " peut et doit "

Ce critère de comparaison dont les relations sont indicées par 1, est défini comme le produit des deux critères précédents:

$p \leq_1^o q$: Si l'observation de p par o doit conduire à un succès alors l'observation de q par o doit conduire à un succès.

et Si l'observation de p par o peut conduire à un succès alors l'observation de q par o peut conduire à un succès.

succès $\in$ Obs(p , o) implique succès $\in$ Obs(q , o)

et Obs(p , o) ={ succès } implique Obs(p , o) ={ succès }

$p \leq_1^o q$ ssi $p \leq_2^o q$ et $p \leq_3^o q$

Les processus sont distingués par un observateur o si les résultats des observations sont différents. Les relations sont donc fondées sur l'ordre suivant dans l'ensemble des résultats d'observation:

$\leq_1$: {échec} < { échec, succès } < { succès }

La relation $p \leq_3 q$ est satisfaite si l'ensemble de traces de p est inclus dans celui de q. La relation $p \leq_2 q$ implique au contraire que l'ensemble de traces de q soit inclus dans celui de p. La relation $\leq_1$ est donc une restriction de la relation $\leq_2$ aux couples de processus ayant des ensembles de traces identiques.
Les processus P et R de l'exemple 5 ne sont pas liés par la relation $\leq_1$.

2.2.1.3 Traitement de la divergence

En général, dans la littérature, le terme de divergence est utilisé pour désigner la possibilité de trouver dans le langage du processus un mot contenant une sous-chaîne infinie de transitions internes. Ce problème est posé quel que soit le modèle de description des processus dès lors que les comportements peuvent être infinis.

Les auteurs de la théorie par équivalence de tests utilisent ce terme pour désigner des comportements partiellement définis, c'est à dire en général des comportements décrits par des expressions récursives mal gardées telles que *rec x . (x + a . nil)*.

Par ailleurs, dans [HN 84] et [Hen 83], deux prédicats, notés $\downarrow$ et $\uparrow$ sont définis pour caractériser les comportements respectivement convergents et divergents:

(i) $nil\downarrow$, $a . p\downarrow$

(ii) $p\downarrow$, $q\downarrow$ implique $(p+q)\downarrow, (p|q)\downarrow, (p[S])\downarrow$

(iii) $t(rec\ x.t/x)\downarrow$ implique $rec\ x.t/x\downarrow$

Tout processus pour lequel on ne peut pas établir par ces règles qu'il est convergent est divergent ($\Omega\uparrow$, rec x . (x + a . nil) $\uparrow$).

Dans [HN 84] et [Hen 83], il n'est pas précisé si l'action a est une action visible ou éventuellement invisible. Ceci est a priori important puisque la relation définie à partir du critère "doit" fait appel à ce prédicat.

Dans [HN 84], a est utilisé comme nom de variable courante de l'ensemble des actions visibles ($\neq \tau$). Dans ce cas, le processus rec x . (τ . x + a . nil) est un processus divergent, conformément à l'acceptation traditionnelle de ce terme. En effet, (τ . rec x(τ.x + a .nil) + a.nil)$\uparrow$ puisqu'on a pour tout p, τ . p $\uparrow$ (ce qui est moins traditionnel) .

Au contraire, dans [Hen 83], a est utilisé comme nom de variable courante du groupe d'actions Λ qui contient τ. Dans ce cas, pour tout p, le processus τ . p converge (i) ainsi que le processus rec x . (τ . x + a . nil) (iii).

Malgré cette ambiguité sur la définition du prédicat de divergence, nous allons maintenant présenter comment le problème de l'existence de chaînes infinies de transitions internes est traité selon cette théorie.

La relation " peut satisfaire" est indépendante du problème de la divergence:

$$p \text{ " peut satisfaire " } o \Leftrightarrow \exists q , (p \mid o) \text{---} \tau^* \rightarrow q \text{ et } q \rightarrow^{\omega}$$

La relation "doit" satisfaire est au contraire définie avec le prédicat de divergence:

p " doit " satisfaire " o $\Leftrightarrow$

si $(p \mid o) = (p_0 \mid o_0) \text{---} \tau \rightarrow (p_1 \mid o_1) \text{---} \tau \rightarrow \ldots$ alors

1- $\exists n \geq 0 , o_n \rightarrow^{\omega}$

et 2- $(p_k \mid o_k) \uparrow$ implique $\exists k' \leq k , o_{k'} \rightarrow^{\omega}$

ce qui signifie qu'une expérimentation ne peut être un succès que si l'observateur peut atteindre un état de succès avant d'atteindre un état de divergence. Il est en effet légitime de considérer que rien ne peut être affirmé sur ce que sera le comportement d'un processus partiellement défini avec un observateur.

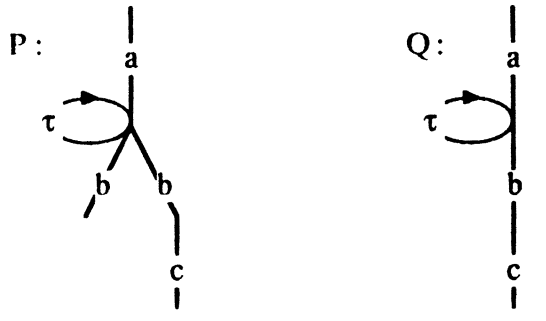
Pour étudier le phénomène des "chaînes infinies de transitions internes", il semble a priori, qu'il faudrait savoir si un tel comportement est associé ou non au prédicat de divergence. Cependant, ceci n'a pas d'influence si on suppose que les observateurs ont des comportements bien définis (ce qui est une hypothèse implicite de la définition précédente).

En effet, si p est un processus susceptible de réaliser τ^* et si on le considère comme un processus convergent alors, quel que soit l'observateur o , l'observation de p par o contiendra l'expérimentation $(p \mid o) \text{---} \tau \rightarrow (p_1 \mid o) \text{---} \tau \rightarrow \ldots \rightarrow (p_k \mid o) \text{---} \tau \rightarrow \ldots$ qui ne peut être un succès que si o est un état de succès. Si, au contraire, on considère que p diverge alors $p \mid o \uparrow$ et p ne peut satisfaire o que si $o_{k'} \rightarrow^{\omega}$ pour k' = 0.

Quelle que soit l'interprétation donnée de τ^*, on aura donc p " doit " satisfaire o , si et seulement si $o \rightarrow^{\omega}$.

Puisqu'un processus ne "doit" satisfaire que les observateurs qui sont dans des états de succès lorsque le processus peut diverger, il est impossible de distinguer des processus divergents qui ont des possibilités de communication différentes au dela des états à partir desquels ils sont susceptibles de diverger.

Exemple:



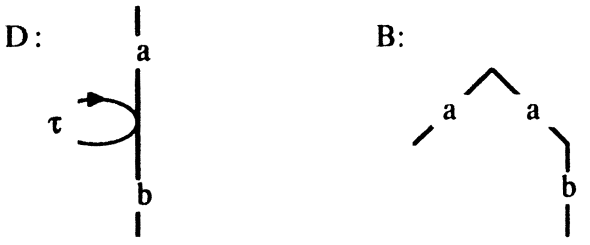
Ces processus sont équivalents pour les trois relations: En effet, ces deux processus ne se distinguent que par leur réaction à l'événement c qui est certaine dans le cas du processus P tandis-qu'elle ne l'est pas dans le cas du processus Q. Puisqu'aucune expérimentation ne peut être certaine d'atteindre ce point, toute expérimentation de l'un ou l'autre des processus est susceptible de ne jamais atteindre d'état de succès.
Fin-exemple

Il reste encore à préciser comment se situent les processus divergents par rapport aux processus non divergents pour les relations de préordre. Pour cela, nous considérerons les deux processus suivants:

Exemple

D "doit" satisfaire un observateur o ssi tout état de o atteint par a est un état de succès. Dans ce cas, B "doit" satisfaire o. (Si après a o atteint par b un état d'échec alors ni B, ni D ne "doit" satisfaire o). On a donc la relation $D \leq_2 B$.

Au contraire, B peut toujours satisfaire l'observateur $o' = a \,.\, \tau \,.\, \omega$; D ne "doit" pas satisfaire o' puisque l'état initial de o' atteint par a n'est pas un état de succès.
On a donc la relation $D <_2 B$.



Fin-exemple

Cette propriété peut être étendue puisque tout processus non-divergent pourra toujours satisfaire les observateurs qui atteignent un état de succès après avoir réalisé une séquence de transitions internes.

On peut formellement montrer que les processus rec x (τ . x + u) et Ω sont équivalents selon le critère de comparaison "doit" en utilisant les axiomes et les règles d'induction proposées dans [HN 84]:

1- $\Omega \leq X$ -- axiome

2- $t = \text{rec } x (\tau . x + u)$

Nous allons montrer que pour tout i , $t^i \leq \Omega$ où par définition $t^0 = \Omega$, $t^{i+1} = t (t^i / x)$.

$t^1 = \tau . \Omega + u \leq \tau . (\Omega + u)$ -- axiome $X + \tau . Y \leq \tau . (X + Y)$

$\leq \Omega + u$ -- axiome $\tau . X \leq X$

$\leq \Omega$ -- axiome $\Omega + X = \Omega$ (spécifique au critère "doit ")

$t^{i+1} = \tau . t^i + u$ --définition

$= \tau . (\tau . t^{i-1} + u) + u$ -- règle de substitution

$= \tau . (\tau . t^{i-1} + u)$ --axiome $X + \tau . (X + Y) = \tau . (X + Y)$

$= \tau . t^i \leq t^i$ -- axiome $\tau . X \leq X$

L'ensemble des termes t^i (pour $i \geq 1$) constitue une chaîne décroissante, pour la relation $\leq_2$. Par conséquent pour tout i, $t^i \leq \Omega$. Par la règle d'induction suivante sur les termes récursifs, on peut déduire $t \leq_2 \Omega$.

$$\frac{d \leq u , \forall d \in FIN (t)}{t \leq u}$$

En résumé l'étude du problème de la divergence identifié comme "chaînes infinies de transitions internes" fait apparaître que:

1- Les processus divergents sont distingués, au dela des points de divergence,suivant leurs ensembles de traces par la relation d'inclusion

2- Il n'est pas possible de distinguer des processus différant par leur non-déterminisme au delà des points de divergence

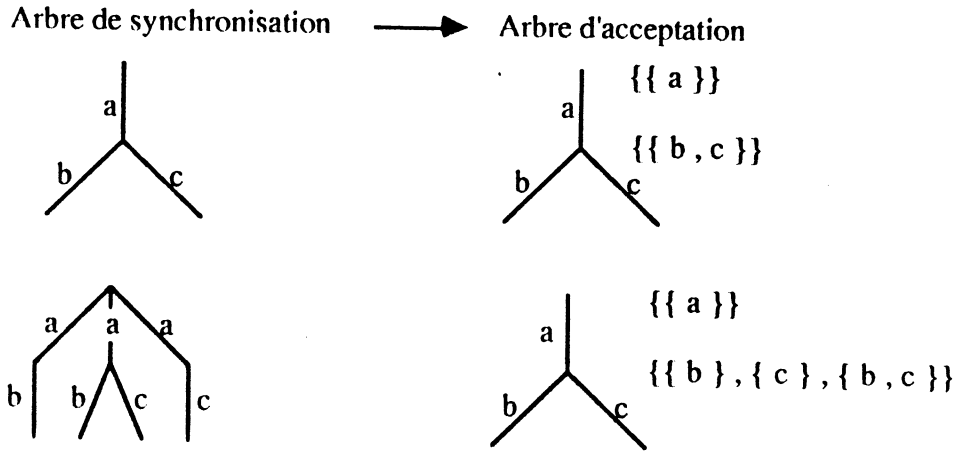
3- Lorsqu'un processus atteint un point de divergence, il est strictement inférieur à tous les processus qui ne divergent pas et qui lui sont équivalents par la relation $\approx_3$.

2.2.1.4 Modèle des arbres d'acceptation

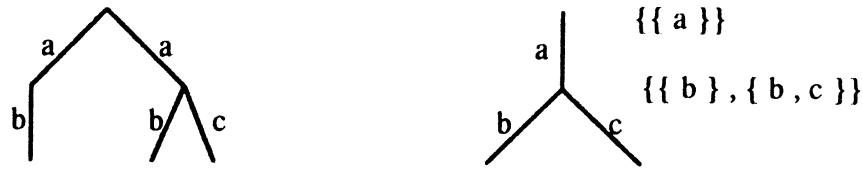
Un modèle de description des processus a été associé à cette théorie: le modèle des arbres d'acceptation [Hen 85]. Ce modèle définit la sémantique observationnelle d'un processus par un arbre déterministe, qui permet de connaître le langage du processus et dont chaque noeud est étiqueté par un attribut caractéristique du non-déterminisme du processus lorsqu'il a exécuté la séquence d'événements correspondante.

Chaque ensemble d'événements réalisables à partir d'un état du processus représente une possibilité locale. Le non déterminisme du processus lorsqu'il a exécuté une séquence s est caractérisé par l'ensemble des possibilités locales des états qu'il peut atteindre par cette séquence.

Exemple



Arbre de synchronisation ⟶ Arbre d'acceptation



Fin exemple

Un noeud de l'arbre est associé à une séquence unique s qui permet d'y accéder. On notera un noeud n par t(s) si s est la séquence qui permet d'accéder au noeud n de l'arbre t. Le non déterminisme est représenté par des ensembles d'acceptation notés A (t (s)).

Les ensembles d'acceptation A(t (s)) caractéristiques du non déterminisme d'un processus représenté par l'arbre t après la séquence s sont saturés, c'est à dire

fermés par union: $X, Y \in A(t(s)) \Rightarrow X \cup Y \in A(t(s))$

et par convexité : $X, Z \in A(t(s)),\ X \subseteq Y \subseteq Z \Rightarrow Y \in A(t(s))$

Sur des processus dont tous les états sont définis la relation générale entre les arbres d'acceptation se réduit à :

$t1 \leq t2$ <u>ssi</u> $L(t1) = L(t2)$

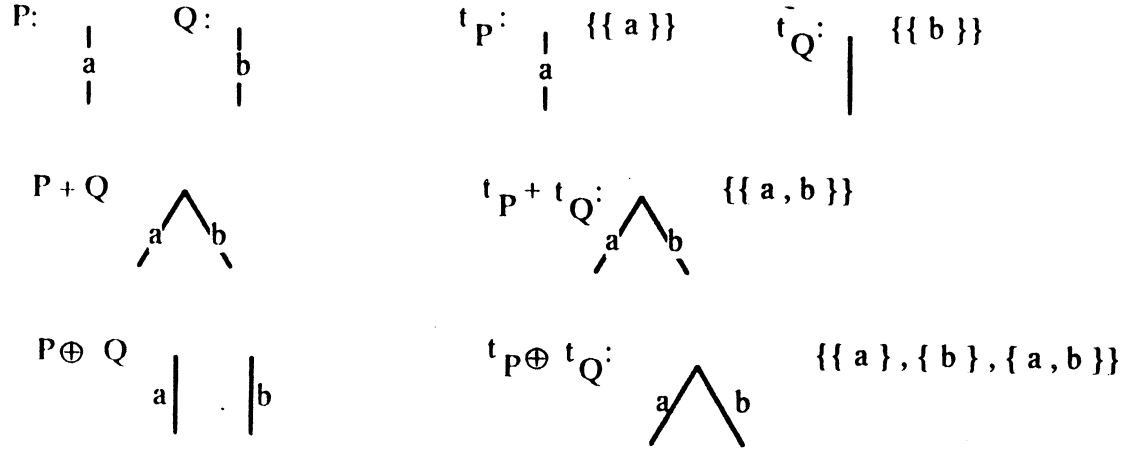
et pour tout $s \in L(t1)$ $A(t2(s)) \subseteq A(t1(s))$

Ceci signifie qu'un processus P2 caractérisé par un arbre t2 est plus déterministe qu'un processus P1 caractérisé par un arbre t1 ssi

1) Il peut réaliser exactement les mêmes séquences d'événements

2) Pour chaque séquence s de P1, on peut associer à tout état de P2 accessible par s un état de P1 accessible par s offrant au plus autant de possibilités locales de communication (puisque les ensembles caractéristiques sont fermés par convexité).

Les arbres d'acceptation sont décrits à l'aide de trois opérateurs : deux opérateurs classiques, le préfixage et la somme correspondant au choix non-déterministe externe et un opérateur plus original qu'on pourrait qualifier de choix non-déterministe interne.



La divergence est représentée, dans les arbres d'acceptation par un type de noeud particulier: les noeuds dits "ouverts" par opposition aux noeuds "fermés" des comportements bien définis. Par définition, tous les fils d'un noeud ouvert sont ouverts. La relation de comparaison des arbres d'acceptation sous leur forme la plus générale est la suivante:

$t1 \leq t2$ ssi 1- $L(t1) \subseteq L(t2)$

2- $CL(t1) \subseteq CL(t2)$

(où CL(t) est l'ensemble des noeuds fermés d'un arbre t.)

3- pour tout $s \in CL(t1)$ $A(t2(s)) \subseteq A(t1(s))$

Un des résultats présentés dans [Hen 85] est que ces arbres peuvent être utilisés comme modèle sémantique des processus , la relation $\leq$ entre les arbres étant équivalente à la relation $\leq_I$ entre les processus.

L'inclusion des langages des processus exprimée par la première condition ne s'applique qu'aux comportements divergents car, dans le cas où les comportements ne sont pas divergents, il faut prendre en compte la troisième condition ce qui conduit à exiger l'égalité des langages des processus en relation.

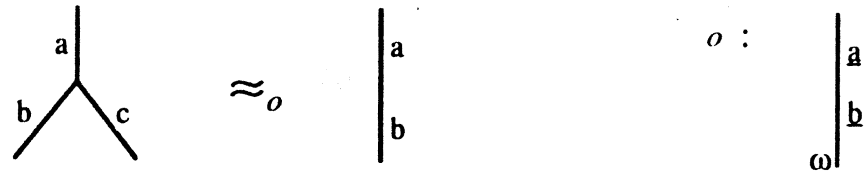
En effet, compte tenu de la structure des ensembles d'acceptation qui sont fermés par union, tout événement successeur dans t2 d'une trace conduisant à un noeud fermé de t1 doit être successeur dans t1 de la même trace. Dans le cas contraire l'élément de A (t2 (s)) qui contient tous les événements possibles après t ne serait pas un élément de A(t1 (s)).

Cette condition traduit la même interprétation de la divergence que le critère de comparaison "peut et doit". Nous avons vu précédemment que ce critère se réduit au critère "peut" puisque le critère "doit" reconnaît comme équivalents tous les processus divergents. La deuxième condition implique que les processus divergents sont strictement inférieurs à tous les processus non divergents ayant les mêmes ensembles de traces.

2.2.1.5 Conclusion

Par rapport aux objectifs initiaux affectés au problème de la comparaison observationnelle, la nature du modèle est prise en compte et des relations fonction des contextes peuvent être définies si l'on assimile la notion de contexte à celle d'ensemble d'observateurs qui ne serait pas le plus général. Toutefois, l'influence des ensembles d'observateurs sur la finesse des relations n'est pas étudiée.

L'outil proposé, expérimentation et report de succès permettrait pourtant d'obtenir des résultats intéressants, puisqu'on peut penser qu'il permet de fusionner les approches proposées dans [Lar 85] et [Bou 84]. Sur des exemples élémentaires, il apparaît qu'on pourrait ignorer certaines différences si celles-ci ne sont pas utilisées par l'environnement:



ou encore qu'on pourrait définir l'équivalence de certaines séquences d'événements:

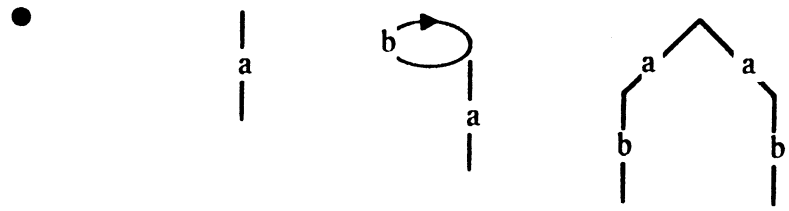
2.2.2 A Theory of Communicating Sequential Processes [BHR 84]

L'objectif poursuivi par cette théorie est de munir l'ensemble des processus définis sur un alphabet d'événements A d'une relation d'ordre pour mesurer le non déterminisme.

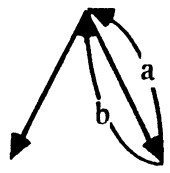
Les processus non déterministes sont, à chaque pas, susceptibles d'effectuer certains événements (définis à partir de leur langage) mais n'ont la capacité de n'en réaliser qu'une partie (en fonction de l'état dans lequel ils se trouvent). Le degré de non-déterminisme proposé ici est en quelque sorte le rapport entre les capacités du processus (évaluées globalement) et ses possibilités.

Ainsi, sur un alphabet A les processus les plus déterministes sont ceux qui à chaque pas sont capables de réaliser tous les événements possibles compte tenu de leur langage, c'est à dire ceux qui laissent tout le contrôle de l'évolution à l'utilisateur. Un processus est défini comme le moins déterministe: celui dont le langage est A^*, qui est donc capable a priori de réaliser n'importe quel événement, mais qui peut également à chaque pas refuser de réaliser n'importe quel événement.

Sur l'alphabet A = { a , b } les processus suivants sont déterministes:



le processus le plus non déterministe est (on le note CHAOS ({ a , b })):



Pour pouvoir définir cet ordre, plusieurs notions sont introduites:

- Une trace d'un processus P est une séquence d'événements de A réalisable par P:

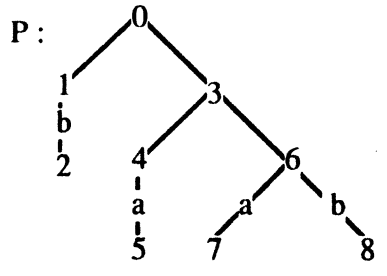
  Traces (P) = { $s \in A^*$ | $\exists$ Q P - s -> Q }

-Les événements initiaux d'un processus P sont ceux de l'alphabet A qui sont tous des premiers événements observables de P:

$$\text{initiaux}(P) = \{\, a \in A \mid \exists\, Q \quad P \text{ -}\!<\! a \!>\!\text{-> } Q \,\}$$

Il faut noter ici que la relation de dérivation utilisée s'applique aux séquences d'événements. Du fait de la loi $< > . < a > = < a >$, tous les événements initiaux des processus successeurs de P par une séquence d'événements internes (répresentés ici par $< >$) sont des événements initiaux de P.

Exemple



initiaux (P0) = {a, b}
initiaux (P1) = { b }
initiaux (P3) = {a, b}
initiaux (P4) = { a }
initiaux (P6) = {a , b}

- L'ensemble des refus est défini pour caractériser le non-déterminisme. Un refus de P est un sous-ensemble X de A tel que le processus P puisse se transformer en P' et qu'à partir de P' aucun événement de X ne soit possible:

$$\text{refus}(P) = \{\, X \mid X \text{ fini } \underline{\text{et}}\ \exists\, Q \quad P \text{ -}\!<>\!\text{-> } Q \ \underline{\text{et}}\ \text{initiaux}(Q) \cap X = \emptyset \,\}$$

Exemple : Sur l'exemple précédent,

refus (P1) = { X | X fini & b ∉ X }	si alphabet (P) = {a , b}	{ ø, {a}}
refus (P3) = { X | X fini & a ∉ X }		{ ø, {b}}
refus (P6) = { X | X fini & a , b ∉ X }		{ ø }
refus (P4) = { X | X fini & a ∉ X }		{ ø, {b}}
refus (P0) = { X | X fini & {a,b} ⊆ X }		{ ø, {a}, {b} }

Fin-exemple

- Un échec est défini comme un couple associant une trace à un refus de l'un des processus accessibles par cette trace.

L'ensemble des échecs définit entièrement le processus

$$\text{Echec}(P) = \{\, (s, X) \mid \exists\, Q \quad P \text{ - } s \text{ -> } Q \ \underline{\text{et}}\ X \in \text{refus}(Q) \}$$

- La relation de pré-ordre partiel entre les processus qu'il faut lire $P \leq Q$ ssi P est plus non déterministe que Q est alors définie par l'inclusion des ensembles d'échecs.

$$P \leq Q \Leftrightarrow \text{échec}(P) \supseteq \text{échec}(Q)$$

- Cette relation définit une structure de demi-treillis complet sur l'ensemble de processus (cf preuve de [BHR 84])

- L'élément minimal est le processus CHAOS:

$$\text{CHAOS} = \{ (s , X) \mid s \in A^* \text{ et } X \text{ fini et } X \subseteq A \}$$

qui est comme nous l'avions dit le processus qui, à chaque pas peut tout faire mais aussi tout refuser.

-Les processus déterministes sont les éléments maximaux:
Un processus P peut être défini comme initialement déterministe si aucun refus de P ne contient un événement initial de P et comme déterministe si tous ses successeurs par la relation - s -> sont initialement déterministes:

$$\text{Pour tout } (s, X) \in \text{échec}(P), a \in X \text{ implique } \langle s . a \rangle \notin \text{Traces}(P)$$

Il est facile de montrer que ces processus sont maximaux en remarquant qu'à toute trace s est associée le refus (s , ∅).

La comparaison de processus ayant des ensembles de traces différents fournit les mêmes résultats que ceux obtenus par les relations fondées sur le deuxième critère de la théorie par équivalence de tests(ce résultat est formellement démontré dans [Bro 83]).

L'approche proposée conduit à des résultats discutables sur certains points délicats. Les processus divergents sont définis comme ayant un non-déterminisme maximal, et sont équivalents de ce point de vue au deadlock. Ici, le processus minimal est le processus CHAOS. De ce fait, le processus Y: (X = b.X + a) \ b est équivalent à CHAOS, ce qui rend éventuellement possibles tous les événements de l'alphabet simplement du fait de l'abstraction sur b qui fait de X un processus divergent.

2.3 Conclusion

Des différentes théories que nous avons présentées, nous pouvons conclure que dans le domaine des processus non divergents deux approches s'affrontent, l'une fondée sur l'outil mathématique de la bisimulation des systèmes de transition, l'autre fondée sur les notions d'expérimentation ou de comportement vis-à-vis du deadlock, la première étant plus contraignante que la seconde.

La nature synchrone ou asynchrone du système de communication est en général prise en compte et l'influence d'un système synchrone ne porte que sur la sémantique des transitions internes (seul le modèle TCSP est inadapté pour en prendre en compte les contraintes d'un modèle synchrone).

L'influence d'un contexte d'utilisation particulier n'est que très peu étudiée, bien que ce problème soit soulevé de différentes façons [Lar 85] [Bou 84] [HN 84].

Si une frontière franche apparaît entre deux approches principales, en revanche des différences existent entre les travaux qui ont une vision globale des processus, notamment sur le problème de la divergence des processus.

L'approche que nous avons développée à partir du langage FP2, et que nous présentons dans les deux chapitres suivants se situe parmi celles qui ont une vision globale du comportement des processus. Toutefois elle s'en distingue par une approche originale du problème de la divergence. Par ailleurs, contrairement à ce qui est proposé dans les deux théories que nous venons de présenter, nous n'établirons a priori aucune relation entre des processus ayant des ensembles de traces différents, en considérant que ces différences doivent essentiellement être prises en compte par le contexte d'utilisation, ce qui fera l'objet du dernier chapitre.

3. Comparaison de processus FP2

Dans ce chapitre, nous présentons les résultats de l'étude que nous avons menée à partir du langage FP2 pour définir et comparer les comportements de communication de processus non-déterministes.

Le domaine des processus a été réduit aux processus d'états finis finitaires. En FP2, ce domaine peut être défini comme l'ensemble des processus dont le comportement est entièrement décrit sans variable. Le comportement des processus est décrit par un ensemble fini de règles de transition, on supposera donc que les processus n'échangent que des signaux (les communications seront représentées uniquement par le connecteur qu'elles utilisent) et que les symboles d'états ne sont pas paramétrés et définissent des états.

Notre objectif est de définir, en fonction du système de synchronisation, la notion d'implémentation correcte d'une spécification (de réalisation correcte d'un processus par un autre) et de définir des algorithmes qui permettent de calculer ces relations sur des processus décrits en FP2. Intuitivement, un processus est une implémentation correcte si, placée dans un environnemnt quelconque l'implémentation peut "remplacer" la spécification c'est à dire si elle permet les mêmes séquences de communication que la spécification et n'introduit pas de refus de communication qui seraient impossibles si la spécification était utilisée.

Dans la première section, le problème sera étudié à partir d'un modèle asynchrone de FP2, dont le seul mécanisme de synchronisation est celui du rendez-vous étendu, c'est à dire avec un ensemble d'opérateurs réduit aux opérateurs de composition parallèle, de connexion et d'abstraction.

Dans la deuxième section, le problème sera étudié dans le cadre d'un modèle synchrone qu'on définit à partir de FP2 en introduisant les opérateurs de composition synchrone et de contrôle par événement en plus des opérateurs du modèle asynchrone.

Dans la dernière section, nous étudierons certaines propriétés algébriques de l'ensemble des processus relativement aux relations qui auront été définies.

3.1 Modèle asynchrone

L'étude du comportement de communication des processus en modèle asynchrone repose sur la résolution de deux problèmes: d'une part, le non-déterminisme qui rend l'environnement incapable de savoir dans quel état se trouve le processus et par conséquent quels sont les événements que le processus est capable de réaliser. L'étude de ce problème fera l'objet de la première section; d'autre part, le problème de la divergence qui provient de limites du modèle de description des processus pour représenter les phénomènes réels. Ce problème sera étudié dans la seconde section.

Notre objectif est de décrire le comportement des processus tel qu'il est perçu par l'environnement, c'est à dire en n'utilisant que les événements de communication avec l'environnement. La troisième section présente un algorithme de calcul du comportement des processus qui est défini dans les deux premières sections.

Dans la quatrième section, nous proposerons des relations de comparaison de comportements et nous étudierons leurs propriétés vis à vis des opérateurs du modèle puisque ces relations ont pour objet la comparaison de processus dans un environnement quelconque, construit avec ces opérateurs.

Enfin, dans une dernière section nous rapprocherons ces résultats de ceux obtenus par les différentes théories présentées au cours du deuxième chapitre.

3.1.1 Etude du non-déterminisme

Dans un modèle asynchrone, deux types de comportement sont source de non-déterminisme: le premier vient de la possibilité d'atteindre plusieurs états tout en réalisant pour l'environnement le même événement, le second est associé à la possibilité de réaliser des transitions internes et donc, en changeant d'état, de changer de capacités de communication sans que l'environnement en ait connaissance.

Les processus sont comparés en fonction de la façon dont ils sont perçus par l'environnement. Celui-ci ne peut influencer l'évolution du système que par les événements qu'il peut réaliser avec le processus. Le comportement de communication des processus devra donc être défini uniquement par les événements auxquels peut participer l'environnement, c'est à dire dans un modèle asynchrone par les événements externes.

Dans le premier paragraphe, nous caractériserons l'influence du non-déterminisme sur les possibilités de communication avec l'environnement et nous définirons formellement un comportement de processus par un automate étiqueté. Cet aspect sera étudié sur le domaine des processus rigides, dont toutes les transitions sont étiquetées par des événements de communication avec l'environnement.

Dans la seconde section, nous montrerons que l'existence de transitions internes dans la description des processus est, pour un observateur, une forme de non-déterminisme interprétable sous forme de non-déterminisme des processus rigides. Nous proposerons un algorithme de rigidification qui permet de calculer un processus rigide indistingable d'un processus non-rigide non-divergent donné.

3.1.1.1 Non-déterminisme des processus rigides

L'objectif de cette première section est d'étudier l'influence du non-déterminisme sur les possibilités de communication des processus afin de définir formellement la notion de comportement des processus.

Pour un observateur quelconque, un processus apparaît comme une boîte noire avec laquelle il peut communiquer mais qui peut ne pas être capable de réaliser tous les événements qui sont a priori possibles, puisque le processus peut atteindre des états à partir desquels seule une partie des événements possibles sera réalisable.

Pour caractériser l'influence du non-déterminisme, sur les possibilités de communication d'un processus, on définira, sur les ensembles d'états accessibles par réalisation d'une séquence d'événements, des ensembles d'évolution certaine. Un ensemble d'événements E sera une "évolution certaine" d'un ensemble d'états S d'un processus P si d'une part, il ne peut pas y avoir deadlock entre un environnement prêt à réaliser E et le processus P lorsque son état courant est l'un des états de S (si pour tout état s de S l'un des événements possibles à partir de s est élément de E) et si d'autre part tout événement de E est susceptible d'avoir lieu.

Nous utiliserons les notations suivantes pour caractériser les ensembles d'événements possibles à partir des états, ou des ensembles d'états. (*)

(*) On notera en minuscules ce qui se rapporte aux états, en majuscules ce qui se rapporte aux ensembles d'états.

evts (s , P) l'ensemble des événements possibles à partir de l'état s du processus P:

$$evts(s, P) = \{ e \mid \exists s' \quad (s, e, s') \in R(P) \}$$

EVTS(S , P) l'ensemble des événements possibles à partir de l'ensemble d'états S du processus P

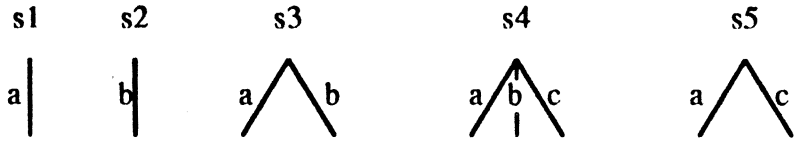
$$EVTS(S, P) = \cup_{s \in S} evts(s, P)$$

Les ensembles d'évolution certaine seront calculés par la fonction *D* suivante qui s'applique à un couple (S , P) où S est un ensemble d'états du processus P:

$$D = \lambda(S, P) . \{ E \subseteq EVTS(S, P) \mid \forall s \in S \quad E \cap evts(s, P) \neq \emptyset \}$$

Exemple

Soient s1, s2 , s3, s4, s5 cinq états d'un processus P représentés ci-dessous



D({ s1 , s2 } , P) = {{ a,b }}
D({ s1, s2, s3 } , P) = {{ a,b }}
D({ s2 , s4 } , P) = { { b } , {a , b} , {b , c} , {a , b , c} }
D({ s2 , s3, s4 } , P) = { { b } , {a , b} , {b , c} , {a , b , c} }
D({ s2 , s5 } , P) = { {a , b} , {b , c} , {a , b , c} }
D({ s3 , s5 } , P) = { { a } , {a , b} , {a , c} , {b , c} , {a , b , c} }

Fin-exemple

Le comportement de communication d'un processus rigide sera alors défini par un automate fini obtenu par déterminisation du graphe de comportement issu de la description du processus FP2. Un état de l'automate sera associé à un ensemble d'états du processus, l'état initial de l'automate étant associé à l'ensemble d'états initiaux du processus. Chaque état de l'automate de comportement sera étiqueté par le résultat de l'évaluation de la fonction D sur l'ensemble d'états du processus.

Pour définir les transitions de l'automate, on utilisera les fonctions successeurs *succ* et *SUCC*, d'un état ou d'un ensemble d'états, par une séquence d'événements:

$$succ(s, <>, P) = \{ s \}$$

$$succ(s, t.e, P) = \{ s' \mid \exists q \in succ(s, t, P), (q, e, s') \in R(P) \}$$

$$SUCC(S, t, P) = \cup_{s \in S} succ(s, t, P)$$

On définit un prédicat *acces*, d'accessibilité d'un état par une trace dans un processus:

$$* \; acces(s, t, P) = \lambda(s, t, P).(\exists i \in I(P), s \in succ(i, t, P))$$

Ce prédicat est étendu aux ensembles d'états:

$$* \; ACCES(S, t, P) = \lambda(S, t, P).(S = \{ s \mid acces(s, t, P) \})$$

* On notera P(t) le couple (S , P) tel que ACCES(S , t , P).

On définit un ***automate de comportement*** par un quintuplet $< Q, \Delta, i, V, T >$

où Q , i , V, T sont respectivement l'ensemble d'états, l'état initial, le vocabulaire, l'ensemble de transitions de l'automate,

Δ est une application qui définit les étiquettes des états: $\Delta : Q \longrightarrow \mathbf{P}(\mathbf{P}(V))$

L'automate de comportement d'un processus P, qu'on note @P est défini ainsi:

- un état de l'automate est associé à un ensemble d'états accessibles du processus

  $Q(@P) = \{ q_S \mid \exists t, S = P(t) \}$, on note @P(t) l'état $q_{P(t)}$
- l'étiquette d'un état est définie par la fonction D appliquée à l'ensemble d'états associé

  $\Delta(@P) = \lambda(q_S).(D(S, P))$
- l'état initial est associé à l'ensemble d'états initiaux du processus

  $i(@P) = q_{I(P)}$
- le vocabulaire est l'ensemble des événements observables définissables sur K(P)

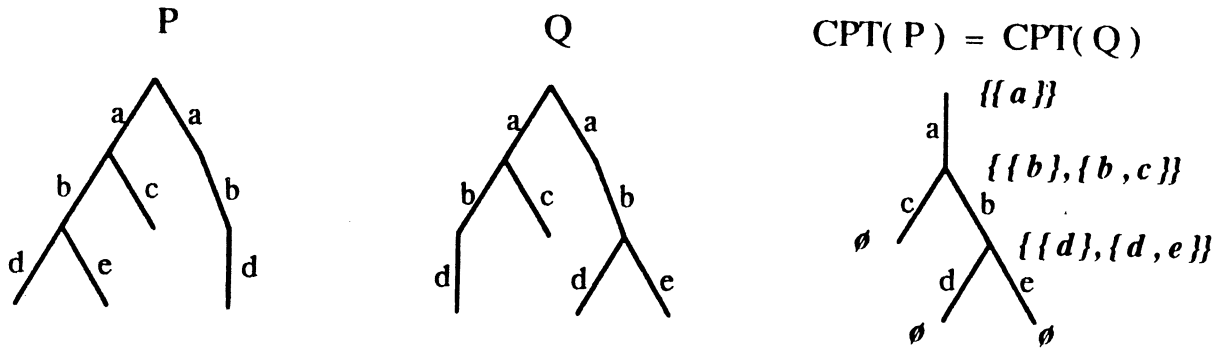
  $V(@P) \models \mathbf{P}(K(P)) - \{ \emptyset \}$
- l'ensemble de transitions est défini par les fonctions successeurs entre ensembles d'états

  $T(@P) = \{ (q_S, e, q_{S'}) \mid S' = SUCC(S, <e>, P) \}$

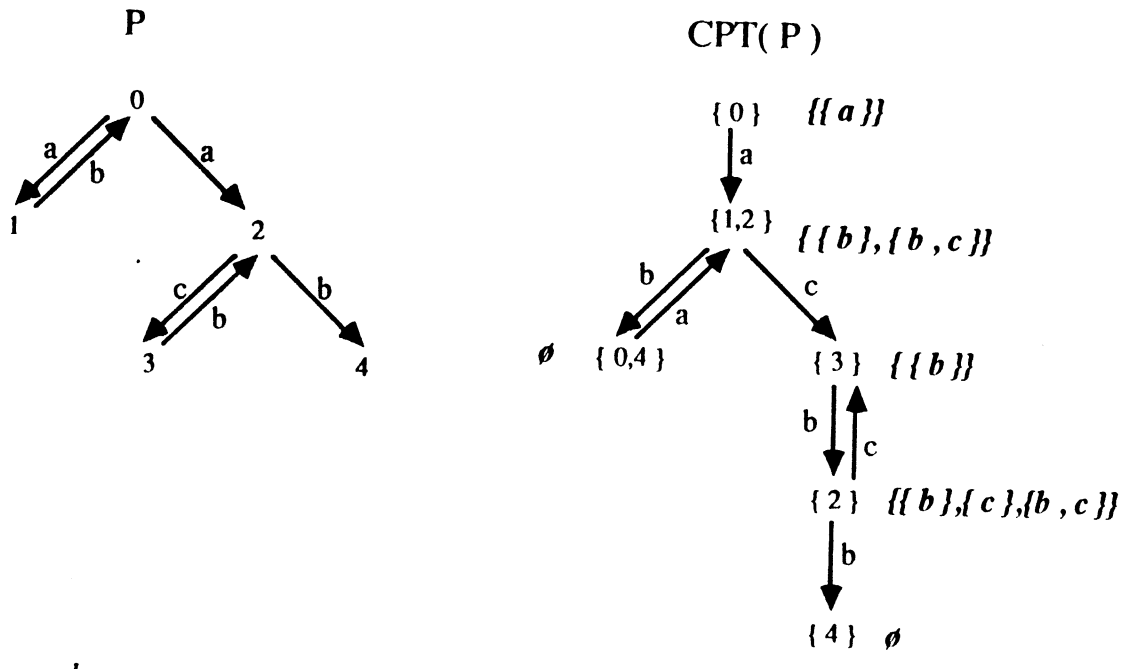
On notera *Traces(P)*, l'ensemble des séquences d'événements réalisables par le processus: le langage formel défini par l'automate @P dont tous les états sont terminaux.

Exemples

Ce premier exemple a été présenté au cours du deuxième chapitre (ex 8, §2.1.2.2). Il montre comment avec l'approche proposée, on conclut que les processus P et Q ont le même comportement.



Ce deuxième exemple présente le calcul du comportement d'un processus ayant un comportement infini régulier.



Fin-exemples

3.1.1.2 Non-déterminisme dû aux transitions internes: Rigidification des processus non divergents

Les transitions internes peuvent être obtenues par synchronisation de processus à l'intérieur d'un réseau et doivent donc obéir aux principes définis par le mécanisme du rendez-vous. Dans un système asynchrone, un processus ne peut pas rester bloqué dans un état instable (à partir duquel au moins une transition interne est possible). Par ailleurs, le changement d'état lié à l'occurence d'une transition interne ne fait pas intervenir l'environnement du processus. Pour que l'environnement soit certain de pouvoir communiquer avec le processus, il est donc nécessaire, et suffisant, qu'il puisse communiquer lorsque le processus atteint des états stables (à partir desquels aucune transition interne n'est possible).

Un problème particulier est donc posé lorsqu'a priori un processus peut réaliser des transitions internes sans atteindre d'états stables. Ce phénomène est celui de la divergence et sera étudié dans la section suivante.

Les transitions internes peuvent avoir lieu sans que l'environnement du processus évolue (toute règle (X , τ , X') d'un processus P génère pour tout état Y d'un processus Q la règle (X.Y , τ , X'.Y) dans le réseau P || Q). Leur occurence se traduit donc par une modification des possibilités de communication du processus qui les réalise.

Lorsqu'un processus atteint un état instable, l'environnement doit considérer que le processus est soit dans l'état initial de la transition interne, soit dans son état final ou encore que ces deux états sont accessibles compte tenu de l'observation du processus qui a eu lieu. Par ailleurs, tous les événements qui peuvent avoir lieu à partir de l'état final de la transition interne peuvent être considérés comme possibles à partir de l'état initial puisque la transition interne aura lieu au bout d'un temps fini.

On peut donc en conclure que les transitions internes ont un double effet sur le comportement des processus qui est d'induire une notion particulière de l'accessibilité des états et de modifier les capacités de communication des états instables des processus.

Ces conséquences des transitions internes sur l'observabilité des processus avaient été formulées dans [HM 80] par l'axiome suivant sur les termes d'une algèbre de processus:

$$\mu . (X + \tau . Y) = \mu . (X + Y) + \mu . Y$$ --axiome A6

où μ est une action de Λ , X et Y deux expressions de comportement.

Parce qu'il induisait des résultats incompatibles pour l'opérateur de composition parallèle avec des principes analogues à ceux de la bisimulation, dans le même article cet axiome avait été abandonné au profit des deux axiomes de CCS

$$\mu . (X + \tau . Y) = \mu . (X + \tau . Y) + \mu . Y$$

$$\mu . \tau . X = \mu . X$$

Dans le premier paragraphe, nous développons un algorithme de rigidification fondé sur ces principes et nous étudions dans le paragraphe suivant les propriétés de la fonction de rigification calculée par l'algorithme précédent par rapport aux trois opérateurs du modèle.

3.1.1.2.1 Algorithme de rigidification

L'opération élémentaire de l'algorithme consiste à remplacer une transition interne par un ensemble de transitions conformément aux principes énoncés dans l'introduction:

Pour toute transition (q , τ , q')

1) l'état q' devient accessible comme l'état q: à partir des mêmes états, par les mêmes événements:
   * toute transition (s , e , q) produit une transition (s , e , q')
   * si q est un état initial alors q' est un état initial

2) Toute transition possible à partir de q' est possible à partir de q:
   * toute transition (q' , e , s) produit une transition (q , e , s')

3) Toute transition possible de q' vers q est possible de q vers q' :
   * toute transition (q' , e , q) produit une transition (q , e , q')

Remarque

L'événement e des transitions qui sont à l'origine d'autres transitions est un événement quelconque, éventuellement un événement interne (sauf dans le troisième cas puisque cela correspondrait à des processus divergents). Des transitions internes peuvent donc être générées au cours de l'exécution de l'algorithme.

Cette opération élémentaire est formalisée par une fonction S sur le domaine des processus non divergents qu'on notera Π_{ndiv} (Π_r étant le domaine des processus rigides) :

$$S : \Pi_{ndiv} \rightarrow \Pi_{ndiv}$$

$$S = \lambda P .\ \underline{si}\ P \in \Pi_r\ \underline{alors}\ P\ \underline{sinon}\ S_\tau(ch_\tau(P), P)$$

La fonction ch_τ appliquée à un processus P non-rigide a pour résultat l'une des transitions internes de P. La fonction S_τ appliquée à une transition interne t et un processus P a pour résultat un processus dans lequel le non-déterminisme dû à la transition t a été traduit par l'accès de plusieurs états, la transition t étant supprimée de la description du processus .

$$S_\tau = \lambda(t, P).\ (K(P), S(P), I_\tau(t, P), R(P) \cup Aj(t, P) - \{t\})$$

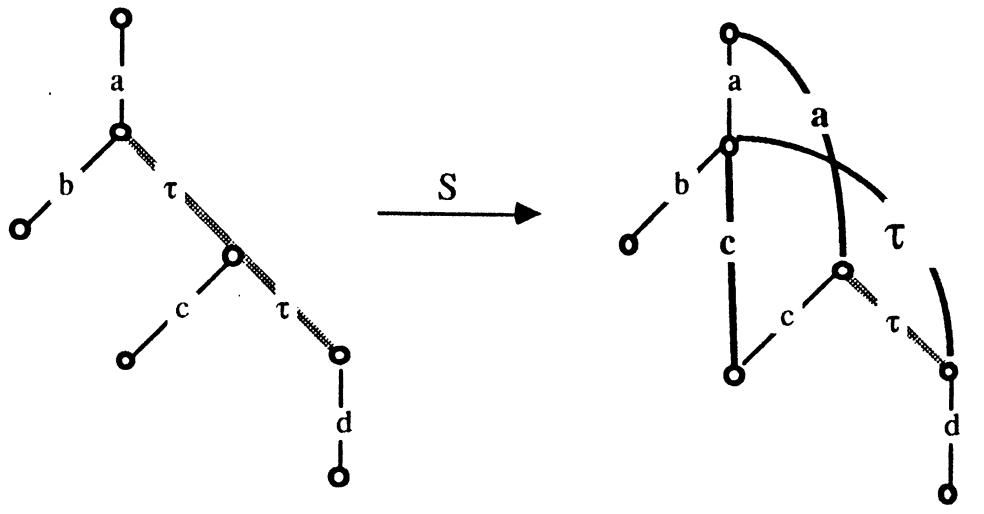
où

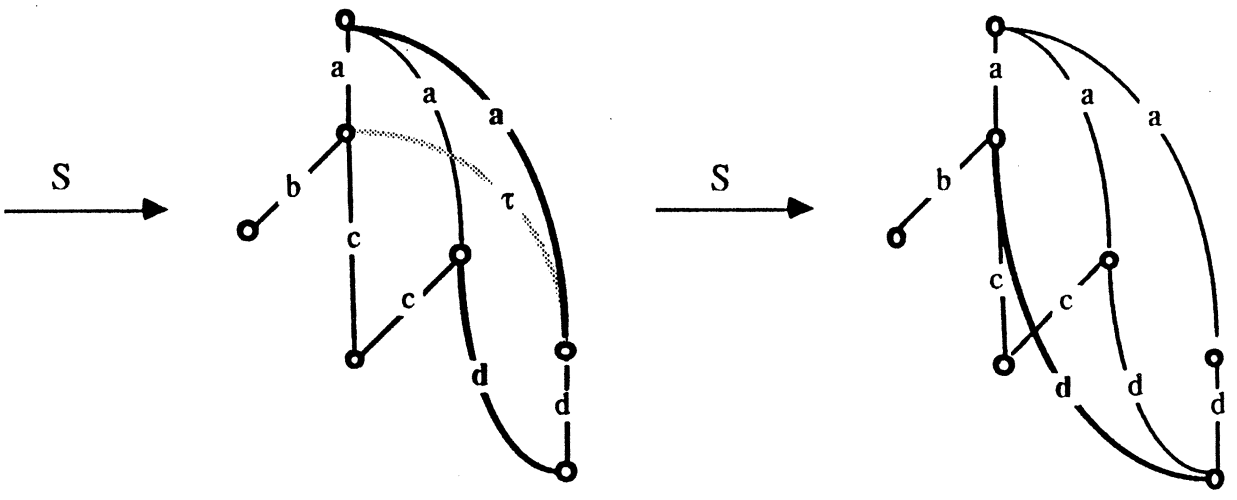
$$I_\tau = \lambda((q, \tau, q'), P).\ I(P) \cup \underline{si}\ q \in I(P)\ \underline{alors}\ \{q'\}\ \underline{sinon}\ \emptyset$$

$$Aj = \lambda((q, \tau, q'), P).\ \{(s, e, s') \mid q = s \wedge (q', e, s') \in R(P) \vee q' = s' \wedge (s, e, q) \in R(P) \vee q = s \wedge q' = s' \wedge (q', e, q) \in R(P)\}$$

Exemple

Cet exemple présente l'exécution de l'algorithme de rigidification sur un processus qui contient deux transitions internes consécutives. On remarquera qu'une transition interne est créée lors de la première application de la fonction S.





Fin-exemple

La fonction S peut être appliquée itérativement sur le domaine des processus non divergents. Nous allons maintenant montrer que son application itérative s'arrête et permet de calculer, quelle que soit la fonction ch_τ, la fonction *Rig*. Cette fonction peut être rapprochée de l'opération de saturation proposée dans [BK 84] avec toutefois deux différences essentielles: elle n'est définie que sur le domaine des processus non divergents et <u>*toutes*</u> les transitions internes sont supprimées:

$$\begin{aligned} &Rig : \Pi_{ndiv} \rightarrow \Pi_r \\ &Rig = \lambda P . (K(P) , S(P) , SUCC\tau(I(P) , P) , R_{obs}(P)) \\ \text{où } &R_{obs} = \lambda P. \{ (s , e , s') \mid \exists\ s_1,s_2 \ \ s_1 \in succ\tau (s , P) , s' \in succ\tau (s_2 , P) \\ &\qquad\qquad (s_1 , e , s_2) \in R (P) \} \end{aligned}$$

Les fonctions *succ*τ et *SUCC*τ sont des extensions des fonctions *succ* et *SUCC* définies précédemment et définissent l'accès par une séquence de transitions internes de longueur quelconque:

$$\begin{aligned} succ\tau (s , P) &= \{ s' \mid \exists k \geq 0 , \ s' \in succ(s , \tau^k , P) \} \\ SUCC\tau (S , P) &= \cup_{s \in S} succ\tau (s , P) \end{aligned}$$

Les propriétés 1 et 2 qui suivent sont des résultats qui nous permettront d'établir le premier lemme, qui prouve que si l'application itérative de S s'arrête, alors le résultat est le processus Rig(P).

Propriété1

$$u' \in succ\tau(u, P) \text{ et } u' \notin succ\tau(u, S_\tau(t, P)) \Rightarrow t = (u, \tau, u')$$

preuve:

$$u' \in succ\tau(u, P) \Rightarrow \exists k \geq 0, u' \in succ(u, \tau^k, P)$$

Si la chaîne de transitions internes entre u et u' dans P est de longueur strictement supérieure à 1 alors, dans S_τ (t,P), il existe entre u et u' une chaîne de transitions internes de longueur k ou k-1.

Si la chaîne de transitions internes entre u et u' dans P est de longueur un alors, par définition, (u, τ, u') est une transition interne de P. Si $u' \notin succ\tau(u, S_\tau(t, P))$ alors la transition $(u, \tau, u') \notin R(S_\tau(t, P))$. Puisque la seule transition de P qui n'est pas une transition de $S_\tau(t, P)$ est la transition t, il faut en conclure que $t = (u, \tau, u')$.

Fin-preuve

Propriété2

$$u' \in succ\tau(u, S_\tau(t, P)) \Rightarrow u' \in succ\tau(u, P)$$

preuve: Par induction sur la valeur de k: longueur de la chaîne de transitions internes entre u et u' dans $S_\tau(t, P)$.

<u>k = 0</u> : u' = u

<u>k = 1</u> : $(u, \tau, u') \in R(S_\tau(t, P))$

1- $(u, \tau, u') \in R(P) \Rightarrow u' \in succ\,\tau(u, P)$

2- $(u, \tau, u') \notin R(P), t = (q, \tau, q')$

a) $q = u \wedge (q', \tau, u') \in R(P)$

$(u, \tau, q'), (q', \tau, u') \in R(P) \Rightarrow u' \in succ\,\tau(u, P)$

b) $q' = u' \wedge (u, \tau, q) \in R(P)$

$(u, \tau, q), (q, \tau, u') \in R(P) \Rightarrow u' \in succ\,\tau(u, P)$

c) $q = u \wedge q' = u' \wedge (q', \tau, q) \in R(P)$: cas de divergence

<u>k > 1</u> : $\exists p \in succ(u, \tau^{k-1}, S_\tau(t, P)) \wedge (p, \tau, u') \in R(S_\tau(t, P))$

Par hypothèse d'induction, on suppose $p \in succ\,\tau(u, P)$,

Par le cas précédent, il vient $u' \in succ\,\tau(p, P)$,

donc $u' \in succ\,\tau(u, P)$

Fin-preuve

Lemme 1

Soit $t = (q, \tau, q')$ une transition interne d'un processus P,

$$Rig(P) = Rig(S_\tau(t, P)) = Rig(S(P))$$

Preuve :

A - Egalité des ensembles de transitions

a) Soit (s, e, s') une transition de $Rig(P)$

$\exists\ s_1, s_2 \quad s_1 \in succ\tau(s, P), s' \in succ\tau(s_2, P)\ (s_1, e, s_2) \in R(P)$

a1: $s_1 \in succ\tau(s, S_\tau(t, P))$, $s' \in succ\tau(s_2, S_\tau(t, P))$

$(s_1, e, s_2) \in R(S_\tau(t, P))$ puisque $e \neq \tau$ donc $(s, e, s') \in R(Rig(S_\tau(t, P)))$

a2: $s_1 \in succ\tau(s, S_\tau(t, P))$, $s' \notin succ\tau(s_2, S_\tau(t, P))$

$t = (s_2, \tau, s')$ -- propriété1

$(s_1, e, s_2) \in R(P) \quad \Rightarrow \quad (s_1, e, s') \in R(S_\tau(t, P))$

$s_1 \in succ\tau(s, S_\tau(t, P)) \Rightarrow \quad (s, e, s') \in R(Rig(S_\tau(t, P)))$

a3: $s_1 \notin succ\tau(s, S_\tau(t, P))\ s' \in succ\tau(s_2, S_\tau(t, P))$

$t = (s, \tau, s_1)$ -- propriété1

$(s_1, e, s_2) \in R(P) \quad \Rightarrow \quad (s, e, s_2) \in R(S_\tau(t, P))$

$s' \in succ\tau(s_2, S_\tau(t, P)) \Rightarrow \quad (s, e, s') \in R(Rig(S_\tau(t, P)))$

a4: $s_1 \notin succ\tau(s, S_\tau(t, P))\ s' \notin succ\tau(s_2, S_\tau(t, P))$

$t = (s, \tau, s_1) = (s_2, \tau, s') \Rightarrow s' = s_1\ , s = s_2$ -- propriété1

$(s_1, e, s_2) \in R(P) \quad \Rightarrow \quad (s, e, s') \in R(S_\tau(t, P))$

$\Rightarrow \quad (s, e, s') \in R(Rig(S_\tau(t, P)))$

donc $\boxed{R(Rig(P)) \subseteq R(Rig(S_\tau(t, P)))}$

b) Soit (s, e, s') une transition de $Rig(S_\tau(t, P))$

$\exists\ s_1, s_2 \quad s_1 \in succ\tau(s, S_\tau(t, P)), s' \in succ\tau(s_2, S_\tau(t, P))$

$(s_1, e, s_2) \in R(S_\tau(t, P))$

Compte tenu de la propriété 2, on a $s_1 \in succ\tau(s, P), s' \in succ\tau(s_2, P)$

Si $(s_1, e, s_2) \notin R(P)$ alors avec $t = (q, \tau, q')$

1) $s_1 = q$, $(q', e, s_2) \in R(P)$

$s_1 \in succ\tau(s, P) \wedge t = (q, \tau, q') \Rightarrow q' \in succ\tau(s, P)$

donc $(s, e, s') \in R(Rig(P))$

2) $s_2 = q'$, $(s_1, e, q) \in R(P)$

$s' \in succ\tau(s_2, P) \wedge t = (q, \tau, q') \Rightarrow s' \in succ\tau(q, P)$

donc $(s, e, s') \in R(Rig(P))$

3) $s_1 = q$, $s_2 = q'$, $(s_2, e, s_1) \in R(P)$

$s_1 \in succ\tau(s, P) \wedge t = (q, \tau, q') \Rightarrow s_2 \in succ\tau(s, P)$

$s' \in succ\tau(q', P) \wedge t = (q, \tau, q') \Rightarrow s' \in succ\tau(s_1, P)$

donc $(s, e, s') \in R(Rig(P))$

donc $\boxed{R(Rig(P)) \supseteq R(Rig(S_\tau(t, P)))}$

donc $\boxed{R(Rig(P)) = R(Rig(S_\tau(t, P)))}$

B- Egalité des ensembles d'états initiaux

On peut considérer que l'initialisation d'un processus P est faite à partir d'un état d'initialisation Init(P) par un événement σ, cet état et cet événement n'étant utilisés que pour l'initialisation des processus: Init(P) et σ ne peuvent apparaître que dans des transitions (Init(P) , σ , q) telles que q est un état initial de P.

On peut alors traiter l'initialisation des processus sur un ensemble de transitions. En effet, si l'on traite ces transitions comme les autres par la fonction S_τ, les ensembles d'états initiaux seront ceux définis par $I(S_\tau(t, P))$.

L'égalité des ensembles d'états initiaux est alors un résultat immédiat à partir du précédent.

Fin-preuve

Lemme 2

L'application itérative de la fonction S s'arrête:

$$\forall P \in \Pi_{ndiv}, \quad \exists k \qquad S^k(P) \in \Pi_r$$

Preuve :

- Les processus sont d'états finis et non divergents. Il n'existe donc pas de cycle de transitions internes dans le graphe de comportement des processus et toutes les chaînes sont de longueur finie.
- L'application de la fonction S peut créer de nouvelles transitions internes. Toutefois, une transition (s , τ , s') ne peut être créée que s'il existe deux transitions (s , τ , s_1) et (s_1 , τ , s'), c'est à dire entre s et s' une chaîne de transitions internes de longueur strictement supérieure.
- Il est donc possible d'exhiber une fonction positive strictement décroissante à chaque application de la fonction S :

  Soit n_P le nombre de chaînes d'un graphe complet à Card(S (P)) sommets.

  * $h(R (P)) = \sum p_i . n_P{}^i$, pour i = 1, ... , n_P -1

    où p_i est le nombre de chaînes de longueur i. Cette fonction est par définition positive (ou nulle dans le cas des processus rigides).
  * Soit k la longueur de la plus longue chaîne de transitions internes qui contient t.

$$h(R (S_\tau(t , P))) \leq \sum p_i . n_P{}^i \text{ , pour } i = 1, \ldots , n_P - 1, i \neq k + (p_k - 1) n_P{}^k + \sum p'_j . n_P{}^j$$

La suppression d'une transition interne ne peut pas engendrer de chaîne de longueur supérieure ou égale à celle de la plus longue chaîne qui la contient . Par conséquent pour tout j de la formule, j est strictement inférieur à k. Compte tenu de la définition de n_P, p'_j est inférieur à n_P quel que soit j .

donc $h (R (S_\tau(t , P))) < h (R (P))$

Fin-preuve

Théorème

L'itération de la fonction S calcule la fonction Rig quelle que soit la fonction ch_τ :

$$\forall\ k,\ S^k(P) \in \Pi_r \Rightarrow S^k(P) = Rig(P)$$

$$\exists\ k,\ S^k(P) \in \Pi_r$$

Preuve :

- Le premier lemme ne fait aucune hypothèse sur le choix de la transition interne et permet de déduire que pour tout k $Rig(S^k(P)) = Rig(S^{k+1}(P)) = Rig(P)$
- Le deuxième lemme permet de déduire que l'itération de S s'arrête

Fin-preuve

Grâce à la fonction de rigidification, le non-déterminisme dû aux transitions internes a été traduit par l'accessibilité de plusieurs états ayant des possibilités de communication différentes. Le comportement de communication d'un processus non-rigide non-divergent pourra donc être obtenu en calculant celui du processus rigide obtenu par rigification.

3.1.1.2.2 Propriétés de la fonction de rigidification dans le modèle

La fonction de rigidification a été définie sur le domaine des processus non divergents avec l'objectif d'établir une équivalence entre processus rigides et processus non rigides. Les propriétés de cette fonction par rapport aux trois opérateurs de FP2 sont étudiées dans ce paragraphe: nous montrons que tout réseau non divergent construit avec le processus Rig(P) a un comportement identique à celui construit avec le processus P. Plus précisément, nous montrerons tout d'abord que pour les opérateurs de connexion et d'abstraction, on obtient l'égalité syntaxique des descriptions de processus utilisant P d'une part, Rig(P) d'autre part. Nous étudierons ensuite le cas de l'opérateur de composition parallèle.

Théorème:

$\forall P, \forall A, B \quad P + AB \in \Pi_{ndiv} \Rightarrow Rig(P + A.B) = Rig(Rig(P) + A.B)$

$\forall P, \forall k \quad Rig(P - k) = Rig(P) - k$

preuve:

Pour alléger l'écriture de cette preuve, on utilisera une notation étendue de la fonction succτ : succE où E est un ensemble d'événements.

$s' \in succE(s, P)$

$\Leftrightarrow \quad s' = s \quad$ ou $\quad \exists q \in succE(s, P), \exists e \in E, (q, e, s') \in R(P)$

$(s, e, s') \in R(Rig(P + A.B))$

$\Leftrightarrow \quad \exists\ s_1, s_2 \quad s_1 \in succ\tau(s, P + A.B),$

$s' \in succ\tau(s_2, P + A.B),$

$(s_1, e, s_2) \in R(P + A.B)$

$\Leftrightarrow \quad \exists\ s_1, s_2 \quad s_1 \in succ\{\tau, \{A, B\}\}(s, P),$

$s' \in succ\{\tau, \{A, B\}\}(s_2, P),$

$(s_1, e', s_2) \in R(P) \quad$ où $\quad e' = e$ ou $e' = e \cup \{A, B\}$

$\Leftrightarrow \quad \exists\ s_1, s_2 \quad s_1 \in succ\{\{A, B\}\}(s, Rig(P)),$

$s' \in succ\{\{A, B\}\}(s_2, Rig(P)),$

$(s_1, e', s_2) \in R(Rig(P)) \quad$ où $\quad e' = e$ ou $e \cup \{A, B\}$

$\Leftrightarrow \quad \exists\ s_1, s_2 \quad s_1 \in succ\tau(s, Rig(P) + A.B),$

$s' \in succ\tau(s_2, Rig(P) + A.B),$

$(s_1, e, s_2) \in R(Rig(P) + A.B)$

$\Leftrightarrow \quad (s, e, s') \in R(Rig(Rig(P) + A.B))$

L'égalité des ensembles d'états initiaux des processus est prouvée par un argument identique à celui utilisé dans la preuve du lemme1 (§31121).

La deuxième partie du théorème, relative à l'opérateur d'abstraction, est immédiate puisque la fonction succτ n'est pas affectée par cet opérateur: $s_1 \in succ\tau(s, P) \Leftrightarrow s_1 \in succ\tau(s, P - k)$.

Fin-preuve

Les propriétés de la fonction de rigidification pour l'opérateur de composition parallèle doivent être étudiés plus finement. En effet, contrairement au résultat obtenu pour les opérateurs de connexion et d'abstraction, en général, les ensembles de transitions des processus Rig(P || Q) et Rig(P) || Rig(Q) sont différents.

En effet, le processus Rig(Q) est par définition un processus rigide et il ne peut y avoir dans le processus Rig(P) || Rig(Q) de transition correspondant à un changement d'état de Q que si ce changement d'état est associé à la réalisation par Q d'un événement avec l'environnement.

Si le processus Q n'est pas rigide, c'est à dire s'il existe une transition (s_Q , τ , s'_Q), on aura pour toute transition (s_P , e_P , s'_P) les transitions suivantes dans P || Q:

$$(s_P.s_Q , \tau , s_P.s'_Q) \qquad \text{et} \qquad (s_P.s'_Q , e_P , s'_P.s'_Q)$$

qui conduisent aux transitions $(s_P.s_Q , e_P , s'_P.s'_Q)$ dans Rig (P || Q) qui correspondent à un changement d'état de Q (de s_Q à s'_Q) sans événement observable de Q (On peut remarquer que cette transition est le résultat du produit des deux transitions de P et Q mais qu'elle est obtenue dans Rig (P || Q) à partir des évolutions indépendantes des deux processus. Par conséquent, ce problème existerait aussi avec un opérateur d'interfoliage qui refuserait l'évolution simultanée des processus composés en parallèle).

Toutefois, plusieurs résultats permettent de préciser les différences existant entre les ensembles de transitions des processus et d'étudier leur incidence sur le comportement observable des processus.

Premièrement, nous pouvons montrer que les ensembles d'états initiaux des processus Rig (P || Q) et Rig (P) || Rig (Q) sont identiques :

Propriété 1:

$$\forall P , Q \in \Pi_{ndiv} \qquad I (Rig (P \| Q)) = I (Rig (P) \| Rig (Q))$$

Preuve:

$s_P.s_Q \in I (Rig (P) \| Rig (Q))$

$\Leftrightarrow s_P \in I (Rig (P)) , s_Q \in I (Rig (Q))$

$\Leftrightarrow \exists i_P \in I (P) , \exists i_Q \in I (Q) , s_P \in succ\tau (i_P , P) s_Q \in succ\tau (i_Q , Q)$

$\Leftrightarrow s_P.s_Q \in succ\tau (i_P.i_Q , P \| Q)$

$\Leftrightarrow s_P.s_Q \in I (Rig (P \| Q))$

Fin-preuve

Deuxièmement, il est possible de caractériser les différences existant entre les ensembles de transitions des processus Rig (P) || Rig (Q) et Rig (P || Q). La propriété qui suit, montre que l'ensemble de transitions de Rig (P) || Rig (Q) est inclus dans celui de Rig (P || Q). De plus, toute transition qui n'est pas une transition de Rig (P) || Rig (Q) correspond à un changement d'état de l'un des processus par réalisation d'un événement observable, l'autre processus changeant d'état en réalisant une séquence de transitions internes.

Propriété 2:

$\forall P, Q \in \Pi_{ndiv}$

A: $R(Rig(P) \| Rig(Q)) \subseteq R(Rig(P \| Q))$

B: $R(Rig(P \| Q)) - R(Rig(P) \| Rig(Q)) = \{ (s_P.s_Q, e, s'_P.s'_Q) \mid$
$(s_P, e, s'_P) \in R(Rig(P)) \wedge s'_Q \in succ\tau(s_Q, Q) - \{s_Q\}$
$\vee (s_Q, e, s'_Q) \in R(Rig(Q)) \wedge s'_P \in succ\tau(s_P, P) - \{s_P\} \}$

Preuve:

A - $(s_P.s_Q, e, s'_P.s'_Q) \in R(Rig(P) \| Rig(Q))$

1) $e = e_P \qquad (s_P, e_P, s'_P) \in R(Rig(P)) \;,\; s_Q = s'_Q$

$\Rightarrow \exists s_{P1}, s_{P2},\; s_{P1} \in succ\tau(s_P, P), s'_P \in succ\tau(s_{P2}, P), (s_{P1}, e, s_{P2}) \in R(P)$

or $\forall u_Q,\; u'_P \in succ\tau(u_P, P) \Rightarrow u'_P.u_Q \in succ\tau(u_P.u_Q, P \| Q)$

donc $s_{P1}.s_Q \in succ\tau(s_P.s_Q, P \| Q)$ et $s'_P.s_Q \in succ\tau(s_{P2}.s_Q, P \| Q)$

et par définition de l'opérateur $\| (s_{P1}.s_Q, e, s_{P2}.s_Q) \in R(P \| Q)$

$\Rightarrow (s_P.s_Q, e, s'_P.s'_Q) \in R(Rig(P \| Q))$

2) idem si $e = e_Q$

3) $e = e_P \cup e_Q \qquad (s_P, e_P, s'_P) \in R(Rig(P)) \;,\; (s_Q, e_Q, s'_Q) \in R(Rig(Q))$

$\Rightarrow \exists s_{P1}, s_{P2},\; s_{P1} \in succ\tau(s_P, P), s'_P \in succ\tau(s_{P2}, P), (s_{P1}, e_P, s_{P2}) \in R(P)$
idem pour Q

$\Rightarrow s_{P1}.s_{Q1} \in succ\tau(s_P.s_Q, P \| Q) \qquad s'_P.s'_Q \in succ\tau(s_{P2}.s_{Q2}, P \| Q)$
$(s_{P1}.s_{Q1}, e_P \cup e_Q, s_{P2}.s_{Q2}) \in R(P \| Q)$

$\Rightarrow (s_P.s_Q, e, s'_P.s'_Q) \in R(Rig(P \| Q))$

$\Rightarrow \forall e \;\; (s_P.s_Q, e, s'_P.s'_Q) \in R(Rig(P \| Q))$

B- $(s_P.s_Q, e, s'_P.s'_Q) \in R(Rig(P \| Q))$, $e = e_P \cup e_Q$

$\Rightarrow \exists\, s_{P1}.s_{Q1}, s_{P2}.s_{Q2}$, $s_{P1}.s_{Q1} \in succ\tau(s_P.s_Q, P \| Q)$, $s'_P.s'_Q \in succ\tau(s_{P2}.s_{Q2}, P \| Q)$

$(s_{P1}.s_{Q1}, e_P \cup e_Q, s_{P2}.s_{Q2}) \in R(P \| Q)$

$\Rightarrow$ $s_{P1} \in succ\tau(s_P, P)$ $\quad$ $s_{Q1} \in succ\tau(s_Q, Q)$

$s'_P \in succ\tau(s_{P2}, P)$ $\quad$ $s'_Q \in succ\tau(s_{Q2}, Q)$

$(s_{P1}, e_P, s_{P2}) \in R(P)$ $\quad$ $(s_{Q1}, e_Q, s_{Q2}) \in R(Q)$

si $e_Q \neq \tau$ et $e_P \neq \tau$ alors

$(s_P, e_P, s'_P) \in R(Rig(P))$ $\quad$ $(s_Q, e_Q, s'_Q) \in R(Rig(Q))$

$(s_P.s_Q, e, s'_P.s'_Q) \in R(Rig(P) \| Rig(Q))$

si $e_Q = \tau$ et $e_P \neq \tau$ alors $(s_P, e_P, s'_P) \in R(Rig(P))$ et $s'_Q \in succ\tau(s_Q, Q)$

si $s'_Q = s_Q$ alors $(s_P.s_Q, e, s'_P.s'_Q) \in R(Rig(P) \| Rig(Q))$

si $s'_Q \neq s_Q$ alors $(s_P.s_Q, e, s'_P.s'_Q) \notin R(Rig(P) \| Rig(Q))$

donc $(s_P.s_Q, e, s'_P.s'_Q) \in R(Rig(P) \| Rig(Q)) - R(Rig(P) \| Rig(Q))$

$\Rightarrow$ $s'_Q \in succ\tau(s_Q, Q)$, $s'_Q \neq s_Q$, $(s_P, e, s'_P) \in R(Rig(P))$

Fin-preuve

Troisièmement, à partir de chaque état, les mêmes ensembles d'événements sont possibles dans les processus $Rig(P) \| Rig(Q)$ et $Rig(P \| Q)$.

Propriété 3 :

$\forall\, s_P.s_Q$, $\quad evts(s_P.s_Q, Rig(P) \| Rig(Q)) = evts(s_P.s_Q, Rig(P \| Q))$

preuve:

$R(Rig(P) \| Rig(Q)) \subseteq R(Rig(P \| Q))$

$\Rightarrow evts(s_P.s_Q, Rig(P) \| Rig(Q)) \subseteq evts(s_P.s_Q, Rig(P \| Q))$

$(s_P.s_Q, e, s'_P.s'_Q) \notin R(Rig(P) \| Rig(Q))$

$\Rightarrow (s_P, e, s'_P) \in R(Rig(P))$ et $s'_Q \in succ\tau(s_Q, Q) - \{s_Q\}$

$(s_P.s_Q, e, s'_P.s_Q) \in R(Rig(P) \| Rig(Q)) \Rightarrow e \in evts(s_P.s_Q, Rig(P) \| Rig(Q))$

Fin-preuve

Enfin, si un état s est accessible par une trace t dans l'un des processus alors il est accessible par cette trace dans l'autre processus.

Propriété 4 :

$$\forall s,\ acces(s, t, Rig(P) \parallel Rig(Q)) = acces(s, t, Rig(P \parallel Q))$$

preuve: induction sur la longueur de t avec l'hypothèse suivante: Pour tout $k \leq n, |t| = k$

1: $acces(s, t, Rig(P) \parallel Rig(Q)) = acces(s, t, Rig(P \parallel Q))$

2: $s'_Q \in succ\tau(s_Q, Q), s'_P \in succ\tau(s_P, P), acces(s_P.s_Q, t, Rig(P) \parallel Rig(Q))$

$$\Rightarrow \quad acces(s'_P.s_Q, t, Rig(P) \parallel Rig(Q))$$

$$acces(s_P.s'_Q, t, Rig(P) \parallel Rig(Q))$$

Fin-preuve

Les quatre propriétés qui viennent d'être établies fournissent essentiellement les deux résultats suivants:

1) Si un état peut être atteint après une séquence d'événements entre le processus et son environnement dans l'un des processsus alors il peut être atteint par la même séquence dans l'autre processus(prop 1 et 4).

2) Tout état offre les mêmes possibilités de communication dans les deux processus (prop 3).

Compte tenu de ces résultats, il est clair que ces deux processus seront caractérisés par des automates de comportement identiques.

3.1.2 Etude de la divergence

Le cas des processus divergents doit être isolé du cas précédent. En effet, si la divergence est aisément détectable sur des processus d'états finis (elle se traduit dans le graphe issu de la description FP2 des processus par l'existence d'un cycle de transitions internes), plusieurs interprétations peuvent en être données selon que l'on fait ou non une hypothèse d'équité.

Cette hypothèse peut informellement s'exprimer ainsi: une alternative infiniment souvent possible ne peut pas être infiniment souvent refusée.

Si l'on fait cette hypothèse, comme cela est fait dans CCS, alors un processus ne peut exécuter une chaîne infinie de transitions internes que s'il n'existe pas d'autre alternative sur aucun des états parcourus. Dans le cas contraire, l'un des événements possibles à partir de l'un des états du cycle de divergence et tenté par l'environnement, aura lieu au bout d'un temps fini.

Cette hypothèse est généralement faite lorsqu'on étudie les protocoles de communication sur des supports non fiables. En faisant cette hypothèse, on admet qu'au bout d'un temps fini chaque message émis est reçu correctement par le destinataire.

Si l'on refuse cette hypothèse, comme cela est fait dans TCSP par exemple, alors il n'existe aucune garantie pour que le processus accepte de communiquer avec son environnement et l'accès à un cycle de divergence peut être assimilé à l'accès possible d'un état de deadlock.

Cette hypothèse doit être faite par exemple lorsqu'on étudie le problème des philosophes défini par E.W Dijkstra et que l'on s'intéresse aux possibilités de coalition d'un groupe de philosophes contre un autre. Cette coalition et l'ensemble du système de gestion peuvent être vus par le dernier philosophe comme un processus avec lequel il veut communiquer et les transitions internes qui constituent la divergence comme les échanges entre les autres philosophes dans le respect des règles définies. L'étude du problème de la famine de l'un des philosophes est donc incompatible avec l'hypothèse d'équité puisque cela revient à éliminer le problème en supposant l'existence de règles supplémentaires non formulées explicitement.

Puisque les deux interprétations de la divergence, équitable et non équitable, ont chacune un domaine d'application particulier, il faut admettre que le modèle de description des processus est insuffisant. Il est donc nécessaire de l'enrichir en définissant le ***comportement d'un processus*** par ***une description FP2 et un prédicat d'équité*** de façon à préciser dans chaque cas particulier, l'interprétation qu'il faut donner de la divergence (*).

Notre objectif sera d'interpréter la divergence en définissant une fonction Div qui associe à chaque processus un processus non-divergent indistingable compte tenu de la valeur du prédicat d'équité. Cette fonction sera définie grâce aux fonctions Div_{eq} (§3121) et Div_{neq} (§3122):

$$Div = \lambda P . (\text{si } eq(P) \text{ alors } Div_{eq}(P) \text{ sinon } Div_{neq}(P))$$

(*) On notera *eq* ce prédicat, dont les valeurs seront notées *eq* pour la valeur vrai et *neq* pour la valeur faux.

Dans cette section, on utilisera les notations suivantes:

Le *point de divergence* d'un état s est l'ensemble des états qui peuvent être parcourus si le processus diverge à partir de s. Il est associé à un ensemble d'états d'un cycle maximal de transitions internes dans le graphe de comportement issu de la description initiale:

$$pdiv(s, P) = \{ s' \mid s' \in succ\tau^+(s, P) \wedge s \in succ\tau^+(s', P) \}$$

où $succ\tau^+(s, P) = \{ s' \mid \exists k > 0, s' \in succ(s, \tau^k, P) \}$

div (*DIV*) est un prédicat caractéristique de la divergence d'un état (d'un d'ensemble d'états):

$div = \lambda(s, P).(pdiv(s, P) \neq \emptyset)$

$DIV = \lambda(S, P).(\exists s \in S, div(s, P))$

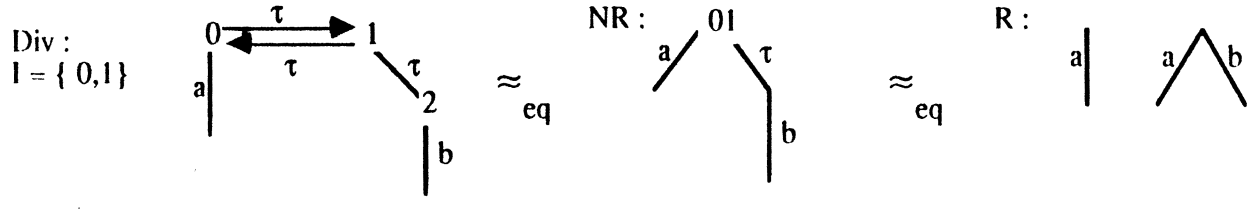
3.1.2.1 Interprétation équitable de la divergence

Si des processus d'états finis ont un comportement équitable, alors tout événement possible à partir de l'un des états à partir duquel un processus peut diverger, s'il est le seul tenté par l'environnement, aura lieu au bout d'un temps fini.

En effet, sous cette hypothèse, en cas de comportement infini, tous les états d'un point de divergence doivent être parcourus une infinité de fois par une trace infinie, faute de quoi il existerait nécessairement une transition interne à partir de l'un des états qui serait infiniment refusée. Puisque tous les états sont parcourus une infinité de fois, si l'environnement tente de réaliser l'un des événements d'un cycle ce choix sera infiniment possible et n'étant pas infiniment refusé, aura lieu au bout d'un temps fini.

Puisque le seul critère est de garantir la communication entre le processus et son environnement, l'accès d'un point de divergence peut être assimilé à l'accès d'un seul état capable de réaliser toutes les transitions de chacun des états du point de divergence.

Exemple



Sous hypothèse d'équité, les processus Div et NR sont indistingables.

En effet, cette hypothèse implique seulement que l'un des événements a ou τ aura lieu au bout d'un temps fini. Ceci n'implique pas que si l'événement a est le seul événement tenté par l'environnement celui-ci aura lieu au bout d'un temps fini, puisque l'occurence de l'événement interne est possible de la même façon. Si au contraire l'événement b est le seul tenté par l'environnement, ceci implique qu'un événement interne à partir de l'état Div1 doit avoir lieu au bout d'un temps fini.
Fin-exemple

Lorsqu'on fait une hypothèse de comportement équitable, on admet que tous les états d'un point de divergence sont équivalents c'est à dire qu'ils sont accessibles de la même façon et qu'ils peuvent réaliser les mêmes transitions. L'interprétation équitable sera donc fournie par une fonction Div_{eq} qui rend équivalents les états d'un même point de divergence et supprime toutes les transitions internes qui sont source de divergence.

$$Div_{eq}(P) = (K(P), S(P), I_{eq}(P), R_{eq}(P))$$

où

$$I_{eq}(P) = \cup_{i \in I(P)} pdiv(i, P) \cup I(P)$$

$$R_{eq}(P) = R(P)$$

$$\cup \{ (q, e, q') \mid \exists s \in S \quad (q, e, s) \in R(P) \text{ et } q' \in pdiv(s, P)$$

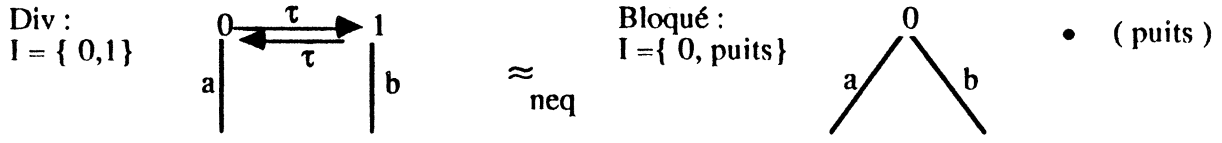
$$\text{ou } (s, e, q') \in R(P) \text{ et } q \in pdiv(s, P) \}$$

$$- \{ (s, \tau, s') \mid s \in pdiv(s', P) \}$$

3.1.2.2 Interprétation inéquitable de la divergence

Si l'on ne fait pas l'hypothèse que les processus ont un comportement équitable, alors il faut interpréter un cycle de transitions internes comme la possibilité réelle d'un comportement infini du processus excluant toute communication avec l'environnement.

Dans ce cas, l'accès à un point de divergence peut être assimilé à l'accès de deux états: un état puits équivalent à l'exécution d'une chaîne infinie de transitions internes et un état offrant toutes les possibilités de transition des états du point de divergence (identique à celui qui caractérise la divergence lorsqu'on fait l'hypothèse d'équité).



L'interprétation inéquitable sera donc fournie par une fonction Div_{neq} qui se limite à traduire l'accessibilité d'un point de divergence par l'accès possible à un état puits pour représenter le refus éventuel de communication des processus.

$$Div_{neq}(P) = (K(P), S(P) \cup \{ puits \}, I_{neq}(P), R_{neq}(P))$$

où

$$I_{neq}(P) = I_{eq}(P) \cup \text{si } DIV(I(P)) \text{ alors } \{ puits \} \text{ sinon } \emptyset$$

$$R_{neq}(P) = R_{eq}(P) \cup \{(s, e, puits) \mid \exists s', (s, e, s') \in R(R_{eq}(P)) \wedge div(s', P)\}$$

3.1.2.3 Redéfinition des opérateurs

Le modèle de description des processus a été enrichi en associant à chaque description de processus FP2 une interprétation de la divergence. Ceci signifie qu'une description FP2 de processus divergent définit ***deux*** processus: le processus divergent équitable et le processus divergent inéquitable.

Les opérateurs sur les processus doivent donc être redéfinis de façon à prendre en compte ce nouvel aspect de la description du comportement des processus. Nous avons montré dans les sections précédentes comment la divergence pouvait être interprétée par une fonction Div qui associe à tout processus divergent un processus non divergent ayant le même comportement observable.

Les opérateurs sont donc redéfinis sur des processus dont la divergence a été interprétée, c'est à dire en utilisant les processus Div(P).

3.1.2.3.1 Composition parallèle

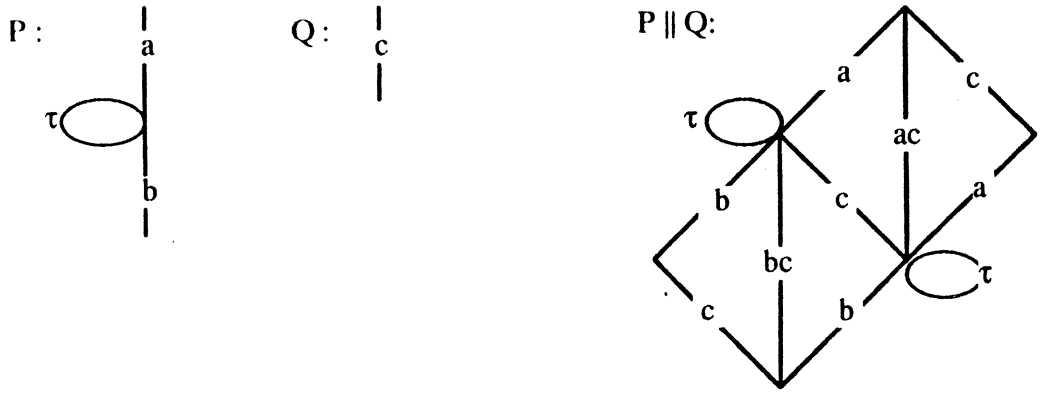
En FP2, comme dans tous les modèles que nous avons présentés, les opérateurs qui traduisent la composition parallèle définissent un réseau de processus comme un processus en exprimant le parallélisme par le non-déterminisme.

Il est généralement admis que la traduction de la composition parallèle par le non-déterminisme est une opération inéquitable dans la mesure où elle ne respecte pas l'identité des processus sur les comportements infinis ce qui a été exprimé par la formule suivante,

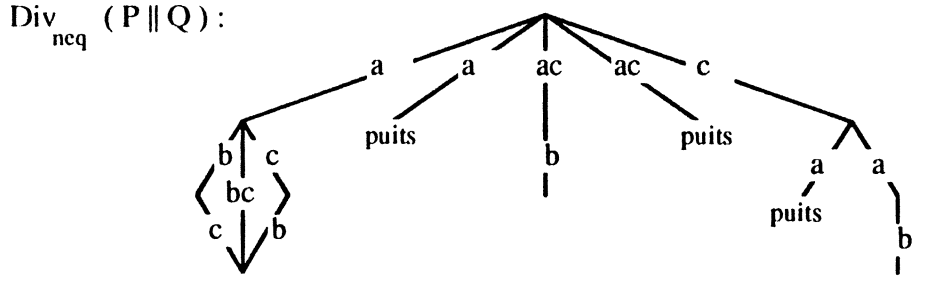
" PARALLELISME = EQUITE ≠ NON DETERMINISME " dans [DK 85] .

Cette traduction du parallélisme a en effet pour conséquence de faire de la divergence une propriété globale des réseaux de processus alors qu'elle n'est qu'une propriété spécifique des processus qui composent les réseaux. En particulier, l'influence de la divergence sur la réalisabilité des événements, qui ne devrait porter que sur les événements des processus divergents qui composent un réseau, porte au contraire sur tous les événements du réseau quelle que soit le processus qui est à l'origine de ces événements.

On peut en effet considérer l'exemple suivant:



Dans le cadre d'un calcul inéquitable, qui interpréterait systématiquement la divergence inéquitablement, le processus P || Q aurait le même comportement que le processus Div_{neq}(P || Q), représenté ci-dessous , qui peut après a refuser de réaliser c.



Au contraire, en proposant de redéfinir l'opérateur de composition parallèle sur les processus pour lesquels la divergence a été préalablement interprétée, ce problème se trouve écarté, la divergence devenant par définition une propriété dont les effets sont limités au processus qui diverge.

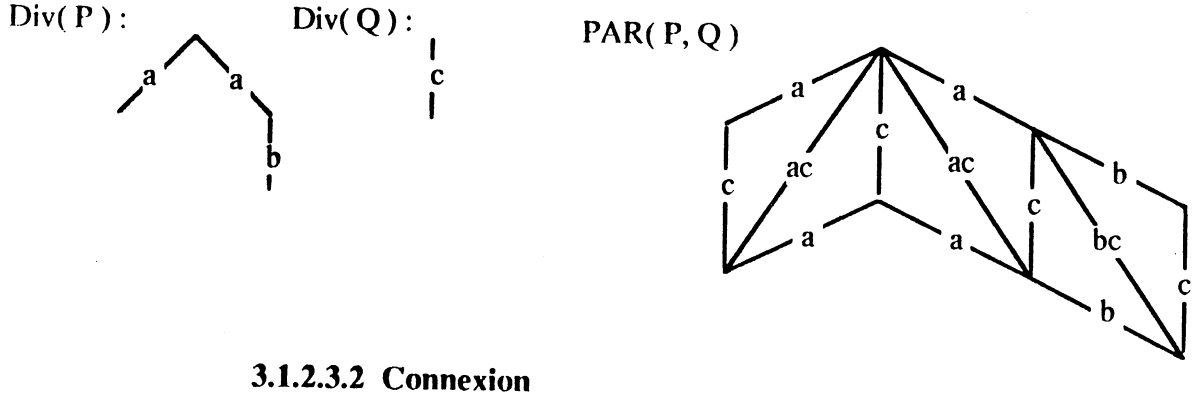
On notera PAR, l'opérateur de composition parallèle défini à partir des processus dont la divergence a été interprétée:

$$PAR(P, Q) = Div(P) \parallel Div(Q)$$

Remarque:

Tout processus obtenu par application de l'opérateur PAR est un processus non divergent puisque la composition parallèle ne peut pas créer de divergence. La valeur du prédicat d'équité d'un processus obtenu par l'opérateur PAR est donc indifférente.

Le processus PAR(P || Q), résultat de la composition du processus P divergent avec le processus Q non divergent est décrit par le graphe suivant. On notera qu'après a il peut refuser de réaliser l'événement b, qui est alternatif dans P d'un comportement divergent, mais qu'il devra toujours accepter de réaliser l'événement c, réalisé par le processus Q non divergent.



3.1.2.3.2 Connexion

Contrairement à l'opérateur de composition parallèle, l'opérateur de connexion est source de divergence. En effet, la connexion de deux connecteurs A et B crée une boucle de transitions internes chaque fois qu'il existe une boucle de { A , B }. Il est donc nécessaire de définir pour les processus connectés une valeur du prédicat d'équité.

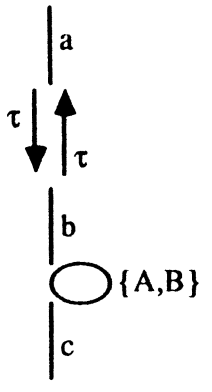
Toutefois, l'interprétation qu'on donne de la divergence qui a été créée par cet opérateur ne doit pas remettre en cause celle qui avait été donnée du processus initial, celui-ci étant inchangé. L'opérateur de connexion qui est défini en utilisant les processus dont la divergence a été interprétée doit donc permettre de distinguer la divergence créée de celle qui existait initialement.

On notera © l'opérateur de connexion sur les processus dont la description est enrichie. Contrairement à l'opérateur initial qui ne prenait en paramètre que le processus et un couple de connecteurs, cet opérateur prend un paramètre supplémentaire: le prédicat d'équité du processus résultat, qu'on note f dans la formule suivante:

$$©(P , \{A,B\}, f) = Div_f (Div(P) + A.B)$$

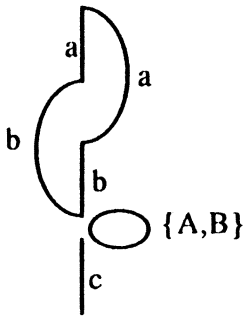
Exemple

CP :
(graphe issu d'une description FP2)

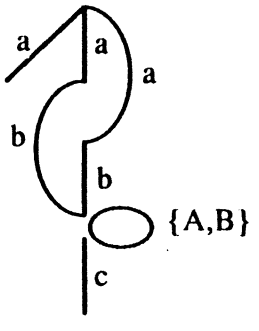


Cette description FP2 d'un processus divergent définit deux processus: le processus E qui interprète équitablement la divergence et le processus nE qui l'interprète inéquitablement.

E = (CP , eq)

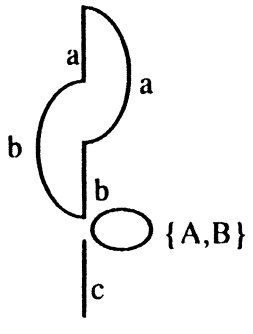


nE = (CP , neq)

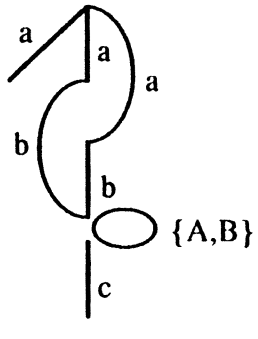


La connexion des connecteurs A et B crée une boucle de divergence qui peut, pour chaque processus, être interprétée équitablement (EE , EnE) ,

EE =
©(E , AB, eq)

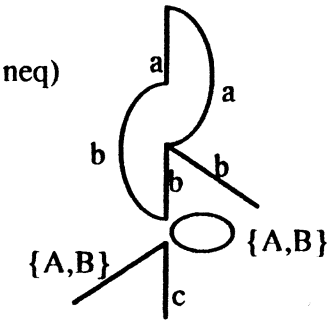


EnE =
©(nE , AB, eq)

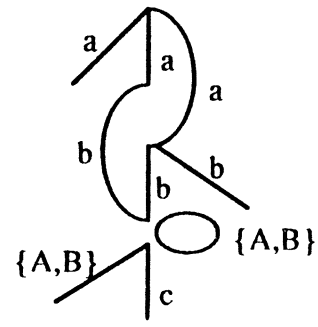


ou inéquitablement (nEE , nEnE)

nEE =
©(E , AB, neq)



nEnE =
©(nE , AB, neq)



Les quatre processus EE , EnE , nEE, nEnE correspondent aux quatre interprétations possibles des deux boucles de divergence de la description P+AB.
Fin-exemple

3.1.2.3.3 Abstraction

L'opérateur d'abstraction ne crée pas de divergence. Sa définition n'est donc pas modifiée du fait de la divergence, si l'on donne au processus résultat la même valeur d'équité qu'au processus initial: $eq(P - k) = eq(P)$

3.1.3 Calcul du comportement de processus

Au cours des deux sections précédentes, nous avons étudié séparément les différents paramètres qui conditionnent le comportement des processus: non-déterminisme sous toutes ses formes, divergence. Le comportement d'un processus quelconque sera donc défini par celui du processus obtenu par rigidification du processus non divergent résultat de l'application de la fonction Div.

Dans cette section, nous proposons un algorithme permettant de calculer directement l'automate de comportement d'un processus quelconque sans calculer explicitement les fonctions Div et Rig précédemment présentées, c'est à dire en particulier sans réaliser l'opération de saturation responsable d'une explosion du nombre de transitions qui décrivent le processus [Sor 87].

Dans cet algorithme, les notions d'événements possibles et d'accès sont substituées par celles d'événements observables et d'accès par observation.

Tout d'abord, on peut définir une fonction "observation d'une séquence d'événements" qui extrait d'une séquence d'événements réalisée par un processus la séquence d'événements de communication avec l'environnement.

obs (< >) = < >
obs (t . e) = si e = τ alors obs (t) sinon obs (t) . e

On dira qu'une séquence d'événements *o* est une *observation d'une séquence* t si *o* = obs(t).

Des fonctions successeurs par observation peuvent être définies relativement à cette notion d'observation:

$$succobs(s , o , P) = \{ s' \mid \exists t , o = obs(t) \wedge s' \in succ(s , t , P) \}$$

$$SUCCOBS(S , < e > , P) = \cup_{s \in S} succobs(s , < e > , P)$$

Si la séquence d'événements est réduite à un événement, cette fonction s'exprime par:

$$succobs(s , < e > , P) = SUCC\tau(SUCC(succ\tau(s , P) , < e > , P) , P)$$

On dira qu'un événement de communication est observable à partir d'un état s, s'il est possible à partir d'un état s' accessible depuis s par une séquence de transitions internes:

$$evtsobs(s , P) = \cup_{s' \in succ\tau(s , P)} evts(s' , P) - \{ \tau \}$$
$$EVTSOBS(S , P) = \cup_{s \in S} evtsobs(s , P)$$

L'automate de comportement @P d'un processus régulier P peut alors être défini à partir de la description du processus:

* L'***ensemble d'états*** est défini par les ensembles d'états accessibles par observation:

$$Q(@P) = \{ q_S \mid \exists o , S = SUCCOBS(I(P) , o , P) \}$$

L'étiquette d'un état de l'automate, résultat de l'évaluation de la fonction D sur l'ensemble d'états associé du processus rigide peut être calculée sur le processus initial en utilisant la notion d'événement observable à partir d'un état. En effet, l'ensemble des événements observables à partir d'un état s du processus initial est identique, par définition, à l'ensemble des événements possibles de cet état s dans le processus rigide équivalent.

On peut alors reformuler la définition de la fonction D pour qu'elle soit utilisable sur des ensembles d'états éventuellement instables:

$$D = \lambda(S, P). \quad \text{si } \neg eq(P) \wedge DIV(S, P) \text{ alors } \emptyset$$
$$\text{sinon } \{ E \subseteq EVTSOBS(S, P) \mid \forall s \in S \quad E \cap evtsobs(s, P) \neq \emptyset \}$$

L'*état initial* est associé à l'ensemble d'états accessible par observation de la séquence vide:

$$i(@P) = q_{SUCC\tau(I(P), P)}$$

L'*ensemble de transitions* est obtenu grâce aux fonctions *succobs*:

On a une transition $(q_S, e, q_{S'})$ dans l'automate de comportement si l'ensemble d'états S' du processus rigide est successeur de S par $< e >$. Compte tenu de la définition de la rigidification cette condition peut également s'exprimer par: S' est l'ensemble d'états successeur de S par observation de $< e >$.

$$T(@P) = \{ (q_S, e, q_{S'}) \mid S' = SUCCOBS(S, < e >, P) \}$$

Le calcul de l'automate de comportement reposera donc essentiellement sur le calcul de la fonction succτ pour les états du processus. En effet les transitions de l'automate sont définies par des extensions de ces fonctions aux ensembles d'états, et le calcul de l'étiquette d'un état de l'automate qui utilise la fonction evtsobs repose également sur ce calcul.

3.1.4 Comparaison de comportements

Dans cette section nous définirons formellement la notion d'implémentation correcte d'un processus par un autre, dans un environnement inconnu. Informellement, un processus IMPL sera une implémentation correcte d'un processus SPEC ssi toute séquence d'événements possible entre SPEC et un environnement quelconque est possible entre IMPL et le même environnement et si lorsque SPEC peut communiquer avec un environnement prêt à réaliser un ensemble d'événements alors IMPL peut aussi communiquer.

Selon cet énoncé informel, la notion d'implémentation correcte doit se traduire par une relation de préordre (puisque les contraintes exigées pour IMPL par rapport à SPEC ne le sont pas à l'inverse) qui exigera que deux conditions soient satisfaites entre les processus.

D'une part, les processus devront avoir les mêmes ensembles de traces. En effet, d'après l'énoncé l'ensemble des traces de la spécification devra être inclus dans celui de l'implémentation, et par ailleurs si l'ensemble de traces de l'implémentation contenait strictement celui de la spécification, il existerait une possibilité pour que l'implémentation ait un comportement qui soit différent de celui de la spécification. Ceci se traduira par une égalité des langages de traces des automates de comportement des processus comparés.

D'autre part, les limitations des possibilités de communication immédiates dûes au non déterminisme devront être moins fortes dans l'implémentation que dans la spécification. Celles-ci sont caractérisées par les ensembles d'évolution certaine qui étiquettent les états de l'automate de comportement. La condition s'exprimera donc par l'inclusion de l'étiquette de la spécification dans celle de l'implémentation pour chaque couple d'états (p , q) des automates tels que ces états soient accessibles par une même trace.

La définition de la relation d'implémentation correcte entre comportements de processus qu'on note $\leq$ est donc la suivante:

$$P \leq Q \quad \Leftrightarrow \exists\, T\,,\ TRACES(P) = TRACES(Q) = T$$
$$\underline{\text{et}} \quad \forall t \in T,\ D(P(t)) \subseteq D(Q(t))$$

La relation $\leq$ est une relation de préordre. Elle induit une relation d'équivalence, qu'on notera $\approx$ qui exprime l'indistingabilité de deux processus dans un environnement quelconque.

La relation $D(P(t)) \subseteq D(Q(t))$ peut s'exprimer de façon équivalente par une relation entre les événements observables des états des processus. En effet, pour qu'elle soit vérifiée il est nécessaire, et suffisant qu'à tout état q de Q(t) on puisse associer un état p de P(t) tel que tout événement observable à partir de p soit observable à partir de q: si cette condition n'était pas vérifiée, alors on pourrait trouver, à partir de chaque état de P(t) un événement qui serait impossible à partir de q. L'ensemble de ces événements serait naturellement un élément de D(P(t)) qui n'appartiendrait pas à D(Q(t)).

Propriété:

$$D(P(t)) \subseteq D(Q(t)) \Leftrightarrow EVTSOBS(P(t)) = EVTSOBS(Q(t))$$

$$\underline{et} \quad \forall q \in Q(t),\ \exists p \in P(t),\ evtsobs(p, P) \subseteq evtsobs(q, Q)$$

Preuve

Condition nécessaire: par l'absurde

$$\exists q \in Q(t), \forall p \in P(t)\ evtsobs(p, P) \not\subseteq evtsobs(q, Q)$$

$$\forall p_i \in P(t) \qquad \exists e_i \in evtsobs(p_i, P), \quad e_i \notin evtsobs(q, Q)$$

$$\{ e_i \} \in D(P(t)) \underline{et}\ \{ e_i \} \notin D(Q(t))$$

Condition suffisante: évident

Fin-preuve

3.1.5 Propriétés des relations entre comportements dans le modèle.

Ces relations ont pour objet de définir l'équivalence des comportements de communication, dans le cadre d'un modèle asynchrone défini par FP2 muni des opérateurs de composition parallèle, de connexion et d'abstraction. L'objet de cette section est donc d'étudier les propriétés de précongruence de la relation $\leq$ pour les trois opérateurs définis en 3123.

3.1.5.1 Composition parallèle

L'étude des propriétés de la relation d'implémentation correcte par rapport à la composition parallèle est faite en deux temps: nous montrons d'abord que cette relation est une relation de précongruence pour l'opérateur || sur le domaine des processus non divergents. Ce résultat est ensuite étendu au domaine des processus divergents.

Lemme

$$\forall P, Q \in \Pi_{ndiv},\quad P \leq Q \;\Rightarrow\; \forall R \in \Pi_{ndiv},\quad P \,\|\, R \;\leq\; Q \,\|\, R$$

Preuve :

Soit $s_Q.s_R$ un état de Q || R accessible par une trace t.

Cette trace t est le résultat de la fusion (merge) de deux traces t_Q et t_R telles que s_Q soit accessible par t_Q dans Q et s_R soit accessible par t_R dans R.

Puisque la propriété $P \leq Q$ est vérifiée, il existe un état s_P accessible par t_Q dans P vérifiant evtsobs(s_P , P) $\subseteq$ evtsobs(s_Q, Q) .

$s_P.s_R$ est accessible par t dans P || R et vérifie evtsobs($s_P.s_R$, P || R) $\subseteq$ evtsobs($s_Q.s_R$, Q || R)

Fin-preuve

Théorème

La relation d'implémentation correcte est une relation de précongruence pour l'opérateur PAR (§31231)

$$\forall P, Q \in \Pi,\qquad P \leq Q \;\Rightarrow\; \forall R \in \Pi,\quad PAR(P, R) \;\leq\; PAR(Q, R)$$

preuve: conséquence immédiate du lemme précédent puisque cet opérateur est défini par les processus non divergents équivalents aux processus divergents.

3.1.5.2 Connexion

L'opérateur de connexion peut être source de divergence. Les propriétés de la relation $\leq$ par rapport à cet opérateur sont étudiées en plusieurs temps. Tout d'abord, nous étudions le cas où l'opérateur de connexion ne crée pas de divergence. Ensuite, dans le cas où l'opérateur est source de divergence, nous distinguons deux cas selon l'interprétation qui est donnée de la divergence créée.

Lemme1

$$\forall P, Q \in \Pi_r,\ P \leq Q \Rightarrow \forall A, B \qquad P + A.B \in \Pi_{ndiv} \Rightarrow P + AB \;\leq\; Q + AB$$

preuve:

De l'égalité des ensembles de traces des processus P et Q, on déduit immédiatement celle des processus P + A.B et Q + A.B

Il reste à montrer que pour toute observation o l'ensemble d'évolution certaine du processus P + A.B est inclus dans celui du processus Q + A.B. Cette propriété sera établie en utilisant la propriété entre les ensembles d'événements observables des états établie au § 314 et se décompose en deux cas selon que les états sont stables ou non dans Q + A.B.

Soit q un état de Q + A.B accessible dans Q + A.B par une observation o. Cet état est accessible dans Q par une trace t qui produit une séquence d'événements t' dans Q + A.B dont l'observation est o.

1) q est un état stable:

Il existe alors un état p dans P accessible par t vérifiant $evts(p, P) \subseteq evts(q, Q)$. Dans P+A.B, cet état p est accessible par la séquence d'événements t' et donc par l'observation o. La propriété d'inclusion des événements possibles dans les processus initiaux est transmise par application de l'opérateur de connexion puisque les états sont stables dans les processus connectés.

2) q n'est pas un état stable:

Puisque, par hypothèse, la connexion ne crée pas de divergence, il existe dans Q + A.B un état stable q' successeur de q par une séquence de transitions internes et donc accessible par observation de o dans Q+A.B. Par définition des ensembles d'événements observables à partir d'un état, on a la relation $evtsobs(q', Q+A.B) \subseteq evtsobs(q, Q+A.B)$.

Puisque q' est un état stable il existe, d'après le point précédent, un état p, accessible dans P + A.B par l'observation o, vérifiant $evtsobs(p, P+A.B) \subseteq evtsobs(q', Q+A.B)$.

On a donc, par transitivité, la relation $evtsobs(p, P+A.B) \subseteq evtsobs(q, Q+A.B)$

Théorème1

$\forall P, Q \in \Pi,$

$$P \leq Q \Rightarrow \forall A, B, f \quad Div(P) + A.B \in \Pi_{ndiv} \Rightarrow \copyright(P, \{A,B\}, f) \leq \copyright(Q, \{A,B\}, f)$$

preuve:

Par définition, pour tout P, les processus P, Div(P) et Rig(Div(P)) sont équivalents.

La preuve est immédiate en utilisant le lemme1 et le théorème du § 31122 sur le domaine des processus non divergents:

$P, Q \in \Pi_{ndiv}$,

$P \leq Q \Rightarrow Rig(P) \leq Rig(Q) \Rightarrow Rig(P) + A.B \leq Rig(Q) + A.B$

$\Rightarrow Rig(Rig(P) + A.B) \leq Rig(Rig(Q) + A.B) \Rightarrow Rig(P + A.B) \leq Rig(Q + A.B)$

$\Rightarrow P + A.B \leq Q + A.B$

Puisque l'opérateur © est défini en utilisant les processus non divergents, la propriété est immédiate:

$P \leq Q \Rightarrow Div(P) \leq Div(Q)$

$\Rightarrow Div(P) + A.B \leq Div(Q) + A.B$ (puisque la connexion ne crée pas de divergence)

$\Rightarrow ©(P, \{A,B\}, f) \leq ©(Q, \{A,B\}, f)$ (lorsque la connexion ne crée pas de divergence les opérateurs © et + sont identiques sur le domaine des processus non divergents)

Lemme2

$\forall P, Q \in \Pi_{ndiv}, \quad P \leq Q \Rightarrow \forall A, B \quad ©(P, \{A,B\}, neq) \leq ©(Q, \{A,B\}, neq)$

Preuve

L'égalité des ensembles d'observations des processus connectés se déduit de celle des processus initiaux. On peut également déduire de l'égalité des ensembles d'observations qu'un des processus connectés ne diverge que lorsque l'autre diverge également.

Par ailleurs, le lemme 1 permet de conclure que lorsque les ensembles d'états atteints par observation ne sont pas divergents, la propriété d'inclusion des ensembles d'évolution certaine est vérifiée.

Puisque l'hypothèse faite sur la divergence dûe à la connexion, est l'hypothèse d'inéquité les ensembles d'évolution certaine de tous les ensembles d'états divergents seront vides et l'inclusion des ensembles d'évolution certaine sera vérifiée.

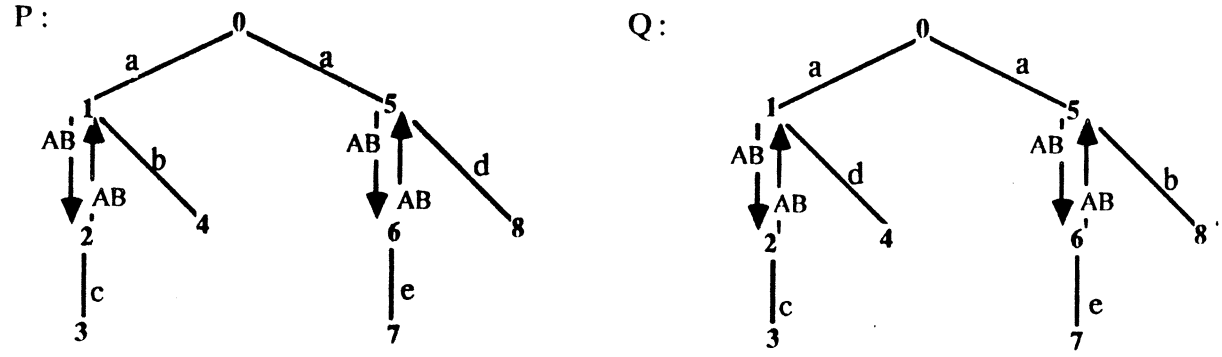
Fin-preuve

Théorème2

$\forall P, Q \in \Pi, \; P \leq Q \Rightarrow \forall A, B \quad ©(P, \{A,B\}, neq) \leq ©(Q, \{A,B\}, neq)$

Preuve: immédiat à partir du lemme 2

Ce théorème établit la propriété de précongruence de la relation d'implémentation correcte pour l'opérateur de connexion ©, lorsque l'hypothèse d'équité faite pour la divergence créée par connexion est l'hypothèse d'inéquité. Cependant, la relation d'implémentation correcte n'est pas une relation de précongruence pour l'opérateur de connexion si l'on donne une interprétation équitable de la divergence créée par la connexion. On peut en effet considérer les processus P et Q suivants qui sont équivalents:



La connexion des connecteurs A et B crée dans les deux processus deux points de divergence: les états 1et 2 d'une part, les états 5 et 6 d'autre part. Si l'on donne une interprétation équitable de la divergence les événements observables à partir des états 1 et 2 sont, dans P {b,c,AB}, dans Q {c,d,AB}; à partir des états 5 et 6 {d,e,AB} dans P, {b,e,AB} dans Q. Après avoir réalisé a, le processus P + AB équitable pourra donc communiquer avec un environnement prêt à réaliser {b , e} ou {c , d}, ce qui n'est pas le cas du processus Q + AB équitable. Par contre, le processus Q + AB équitable pourra communiquer avec un environnement qui pourra réaliser {d , e} ou {b , c}, ce qui n'est pas le cas du processus P + AB équitable.

3.1.5.3 Abstraction

La relation d'implémentation correcte est une relation de précongruence pour l'opérateur d'abstraction puisque celui-ci se limite à supprimer certains comportements possibles du processus initial.

Théorème:

$\forall P, Q \in \prod,\ P \leq Q \ \Rightarrow\ \forall k \quad P - k \leq Q - k$

Preuve:

L'égalité des ensembles d'observations est immédiate, toute observation d'un processus P – k étant une observation du processus P dans laquelle aucun des événements ne contient une communication sur le connecteur k.

Si un état q est accessible par une observation *o* dans Q – k, il existe un état p accessible dans P par *o* vérifiant evtsobs(p , P) $\subseteq$ evtsobs(q , Q). Cette propriété est conservée par l'opérateur d'abstraction puisque les mêmes événements sont supprimés de evtsobs(p , P) et de evtsobs(q , Q).

Fin-preuve

3.1.5.4 Conclusion

Les résultats établis dans les sections précédentes ont montré que la relation d'implémentation correcte est une précongruence pour les opérateurs de composition parallèle, d'abstraction et de connexion lorsque celui-ci ne crée pas de divergence.

Lorsque l'opérateur de connexion est source de divergence, la relation n'est conservée que lorsqu'on fait l'hypothèse d'inéquité sur les comportements divergents créés par connexion. Ceci tient au fait que les relations proposées ne comparent les processus qu'en fonction de leur réaction immédiate à une proposition de l'environnement sans prendre en compte des différences possibles ultérieures, liées aux états atteints. Lorsque la connexion crée de la divergence, il devient nécessaire de prendre en compte, au moins partiellement, ces différences liées aux états atteints.

3.1.6 Comparaison avec les autres travaux

L'approche qui a été développée à partir du langage FP2 peut être rapprochée de celles qui ont été présentées au cours du deuxième chapitre.

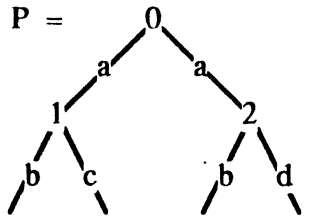
Tout d'abord, comme nous l'avons vu sur plusieurs exemples dans les deux premières sections, elle se situe parmi les théories qui ont une vision globale du comportement des processus par opposition à celles du type CCS qui ont une vision plus morcelée du comportement de communication.

Nous pouvons toutefois comparer nos résultats à ceux qui ont été obtenus par ailleurs. Sur l'essentiel, c'est-à-dire sur le domaine des processus non divergents, ces résultats traduisent la même notion de comportement observable et il peut être intéressant de remarquer comment la caractérisation du non déterminisme est faite dans les différentes théories:

Toutes ces théories évaluent le non déterminisme par une fonction sur le comportement qui peut immédiatement suivre l'observation d'une trace.

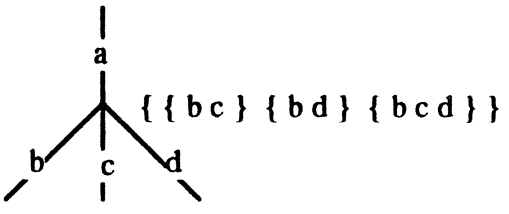
Sur un exemple, nous pouvons montrer comment et pourquoi les caractérisations par ensemble d'acceptation, ensemble de refus ou ensemble d'évolution certaine sont équivalentes.

Soit P un processus décrit par l'arbre de comportement suivant. Nous nous limiterons à présenter la caractérisation du non déterminisme du processus P ayant réalisé $< a >$.



Dans le modèle des Arbres d'acceptation, ce processus est exprimé par l'arbre déterministe étiqueté:

Acceptation



Refus Dans la théorie des refus, un ensemble de refus lié à la séquence a est défini :

$$(<a>, X) \in \text{Refus}(P) \Leftrightarrow b \notin X \wedge (c \notin X \vee d \notin X)$$

Sur un alphabet $A = \{ a, b, c, d \}$

$$\{ X \} = \{ \emptyset, \{ a \}, \{ a c \}, \{ a d \}, \{ c \}, \{ d \} \}$$

Evolution certaine

$$S = \{ P1, P2 \}$$

$$D(S, P) = \{ \{ b \}, \{ b c \}, \{ bd \}, \{ cd \}, \{ bcd \} \}$$

Les ensembles de refus et d'évolution certaine correspondent à des approches complémentaires mais sont évalués sur des ensembles différents. Dans le second cas, la fonction est calculée sur l'ensemble EVTSOBS(S , P) tandis que dans le premier cas elle est calculée sur un alphabet A. Toutefois, étendue à l'ensemble des événements possibles, la seconde conduirait à un résultat exactement complémentaire puisque la propriété suivante est vérifiée

$$X' \in D(S, P) \wedge X' \subseteq X'' \Rightarrow \forall s \in S \quad X'' \cap \text{evtsobs}(s, P) \neq \emptyset$$

Par ailleurs, les interprétations données de ces caractéristiques sont complémentaires:

$$P \leq Q \Leftrightarrow \text{refus}(P) \supseteq \text{refus}(Q)$$

$$D(S, P) \subseteq D(S', Q)$$

Les ensembles d'acceptation peuvent également être rapprochés des ensembles d'évolution certaine. En effet, ces ensembles sont définis directement à partir des ensembles d'états:

$$\text{Acceptation}(S, P) = \{ \text{evtsobs}(s, P) \mid s \in \text{Sat}(S, P) \}$$

où Sat est une fonction qui calcule la saturation d'un ensemble S (§ 2.2.1.3)

Compte tenu de la propriété que nous avons établie en 3.1.2.1 il est naturel que l'interprétation donnée des ensembles d'acceptation conduise aux mêmes résultats. Rappelons que ce résultat était le suivant :

$$D(S, P) \subseteq D(S', Q) \Leftrightarrow \forall s' \in S', \exists s \in S, \text{evtsobs}(s, P) \subseteq \text{evtsobs}(s', Q)$$

Il est donc évident qu'un ensemble d'états S et l'ensemble d'états obtenu par saturation de S seront caractérisés par le même ensemble d'évolution certaine. Il est également évident que sur des ensembles d'états saturés la propriété précédente peut s'exprimer par:

$$\forall s' \in S' \quad \exists s \in S \quad \text{evtsobs}(s, P) = \text{evtsobs}(s', P)$$

ce qui correspond à l'inclusion des ensembles d'acceptation.

En ce qui concerne les transitions internes, l'interprétation est essentiellement identique: grâce aux axiomes D4 et N1 de [HN 84],

$$X + \tau Y = \tau(X + Y) + \tau Y \qquad \text{[axiome D4]}$$

$$\mu X + \mu Y = \mu(\tau X + \tau Y) \qquad \text{[axiome N1]}$$

on peut retrouver l'axiome A6 de [HM 80]

$$\mu(X + \tau Y) = \mu(X + Y) + \mu Y \qquad \text{[axiome A6]}$$

Le modèle des refus, fondé sur TCSP, adopte également le même point de vue dans un formalisme qui est toutefois différent. Nous avons vu au cours du premier chapitre (§ 1.1.2) que les transitions internes ne sont pas explicitement nommées dans les modèles dérivés de CSP. Toutefois, la sémantique des transitions internes en modèle asynchrone est fournie lorsque de tels événements sont produits par abstraction.

$$[(c \rightarrow P) | (d \rightarrow Q)] \setminus \{c\} = (P \setminus \{c\}) \lceil\rceil ((P \setminus \{c\}) [] (d \rightarrow Q \setminus \{c\}))$$

De plus la sémantique de l'opérateur de choix non déterministe interne est identique à celle que l'on obtient en considérant le comportements pris en paramètre comme s'ils étaient accessibles à partir d'un même état par des transitions internes.

Ces trois théories traduisent donc la même notion de comportement observable des processus réguliers non divergents.

Toutefois, elles se distinguent sur plusieurs points notamment: le domaine des processus étudiés et l'interprétation du phénomène de la divergence.

Le domaine des processus que nous avons étudié est celui des processus réguliers sur un alphabet d'événements fini incluant une action privilégiée: le τ.

Les processus insuffisamment définis du fait d'une récursion non gardée ne sont donc pas étudiés contrairement à ce qui est fait dans [HN 84] où il est par ailleurs étudié sous le nom de divergence. Le problème de la divergence a été étudié ici d'une façon originale en permettant d'associer à chaque processus (donc à chaque opération effectuée pour construire un réseau)un prédicat définissant ou non l'équité de son comportement (de la construction). Ceci a conduit à définir des relations de comparaison plus fines que celle des théories totalement équitables du type CCS ou inéquitables du type TCSP, permettant de plus d'aborder certains aspects ignorés par les autres modèles(comparaison des processus au-delà des points de divergence).

3.2 Modèle synchrone

On peut définir un modèle synchrone à partir de FP2 en étendant l'ensemble des opérateurs avec des opérateurs de composition parallèle synchrone et de contrôle par événement.

Dans la première section nous étudierons les conséquences de ces opérateurs sur l'observabilité du comportement des processus et dans la deuxième section nous définirons et étudierons les propriétés d'une relation d'implémentation correcte dans un modèle synchrone . Dans la dernière section, nous rapprocherons ces résultats de ceux obtenus à partir du modèle synchrone SCCS par Hennessy [Hen 83] d'une part ou Milner [Mil 83] d'autre part.

3.2.1 Non déterminisme et synchronisme

Les opérateurs synchrones introduisent une forme du synchronisation supplémentaire des processus. De ce fait, les processus sont plus contraints dans leur évolution que dans un modèle asynchrone et par conséquent moins non déterministes. Dans cette première section nous montrerons que ces opérateurs synchrones éliminent uniquement la source de non déterminisme provenant des transitions internes des processus.

3.2.1.1 Les transitions internes sont observables

Dans un réseau construit à l'aide de l'opérateur de composition parallèle synchrone, il ne suffit pas pour que deux processus puissent communiquer qu'ils soient dans des états à partir desquels la communication soit possible. Il faut encore que tous les processus du réseau soient capables d'effectuer une transition. Ainsi, dans un réseau P1 ||| P2 ||| P3 où P1 et P2 seraient susceptibles de communiquer si toute synchronisation n'était faite que par rendez-vous, cette communication peut être rendue impossible par le processus P3 si celui-ci se trouve dans un état de deadlock. C'est donc seulement si P3 peut effectuer une transition interne que les processus P1 et P2 pourront communiquer. De ce fait, les processus nil et τ . nil ne sont pas équivalents.

Les transitions internes deviennent donc observables parce que leur occurence est liée à des changements d'états observables des autres processus du réseau.

3.2.1.2 La relation d'implémentation correcte est valide sur les processus rigides

La synchronisation induite par les opérateurs synchrones rythme l'évolution d'un réseau de processus sur une horloge centrale. Il est par conséquent possible de connaître les évolutions internes des processus. Au contraire, le non déterminisme dû à l'accessibilité de plusieurs états par réalisation d'un même événement n'est a priori pas influencé par cette nouvelle forme de synchronisation qui ne contraint pas les choix internes des processus.

La propriété qui suit établit formellement ce fait puisqu'elle montre que la relation d'implémentation correcte définie sur les processus rigides en 3.1.2.1 est une précongruence pour les opérateurs synchrones:

Propriété

$$\forall P, Q \in \Pi_r , \quad P \leq Q \Rightarrow \forall R \quad P|||R \leq Q|||R$$

$$\forall e \quad e \Rightarrow P \leq e \Rightarrow Q$$

preuve:

Par définition de l'opérateur, les connecteurs utilisés dans l'événement de contrôle ne sont pas des connecteurs des processus contrôlés. Il existe donc entre les graphes de comportement des processus e $\Rightarrow$ Proc et Proc un isomorphisme dû uniquement au renommage des événements. La relation e $\Rightarrow$ P $\leq$ e $\Rightarrow$ Q se déduit donc immédiatement de l'hypothèse P $\leq$ Q.

La propriété de la relation $\leq$ par rapport à la composition parallèle synchrone se déduit de la propriété de précongruence de $\leq$ par rapport aux opérateurs $\|, +, -, e \Rightarrow$ et de la définition de cet opérateur $|||$ que nous avons énoncée au cours du premier chapitre.

Rappelons que la composition synchrone de deux processus peut être définie à partir des quatre opérateurs précédents:

$$P1 ||| P2 = (H_1 \Rightarrow P1) \| (H_2 \Rightarrow P2) \| H + H'_1.H_1 + H'_2.H_2 - \{ H_1, H'_1, H_2, H'_2 \}$$

où $H = I(H) = \{ T \}$, $R(H) = \{ H : \{ H_1 , H_2 \} \Rightarrow H \}$

Par conséquent

$$P ||| R = (H_1 \Rightarrow P) \| (H_2 \Rightarrow R) \| H + H'_1.H_1 + H'_2.H_2 - \{ H_1 , H'_1 , H_2 , H'_2 \}$$

Par ailleurs $\leq$ est une précongruence pour l'opérateur de contrôle :

$$P \leq Q \ \underline{impl} \quad H_1 \Rightarrow P \leq H_1 \Rightarrow Q$$

Par conséquent

$$(H_1 \Rightarrow P) \| (H_2 \Rightarrow R) \| H + + H'_1.H_1 + H'_2.H_2 - \{ H_1, H'_1, H_2, H'_2 \} = P ||| R$$
$$\leq (H_1 \Rightarrow Q) \| (H_2 \Rightarrow R) \| H + + H'_1.H_1 + H'_2.H_2 - \{ H_1, H'_1, H_2, H'_2 \} = Q ||| R .$$

3.2.2 Relations entre processus

Comme nous venons de le voir, l'introduction des opérateurs synchrones modifie uniquement le non déterminisme lié aux transitions internes. Plus précisément , alors qu'elles étaient source de non déterminisme en modèle asynchrone, elles ont dans un tel modèle le même rôle que tout événement de communication-synchronisation puisqu'elles ne peuvent avoir lieu dans un processus élément d'un réseau synchrone, que si les autres processus peuvent évoluer sans communiquer avec le processus.

La notion d'implémentation correcte dans un modèle utilisant des opérateurs synchrones se déduit donc, dans sa forme, de la relation définie sur les processus rigides:

$$P \leq_S Q \Leftrightarrow \exists\, T = TRACES(P) = TRACES(Q)$$
$$\forall\, t \in T , \ D_S(P(t)) \subseteq D_S(Q(t))$$

où $D_S = \lambda(S, P) \{ E \subseteq EVTS(S, P) \mid E \cap evts(s, P) \neq \emptyset \}$

Sur le domaine des processus rigides cette relation est identique à celle qui a été définie en modèle asynchrone puisque dans ce cas $\tau \notin EVTS(S, P)$. Par contre, elle est définie sur le domaine des processus rigides et non.rigides et il est évident que toute équivalence entre processus rigides et non rigides est maintenant exclue.

Au cours du premier chapitre, nous avons montré que les opérateurs du noyau de FP2 étaient suffisants pour définir des compositions quelconques de comportements si on utilisait des processus contrôlés. On peut montrer facilement que:

$$P \leq_S Q \quad \underline{ssi} \quad P_{ctrl} \leq Q_{ctrl}$$

puisqu'il existe entre les graphes de comportement des processus et des processus contrôlés un isomorphisme de renommage.

Enfin, on peut montrer que la relation d'implémentation correcte en modèle synchrone est une précongruence pour les opérateurs du noyau de FP2 ainsi que pour les opérateurs synchrones.

Propriété:

$$\forall P, Q \in \Pi$$

$$P \leq_S Q \Rightarrow \begin{array}{llll} \forall R & P \parallel R & \leq_S & Q \parallel R \quad (1) \\ \forall A,B & P + A.B & \leq_S & Q + A.B \quad (2) \\ \forall k & P - k & \leq_S & Q - k \quad (3) \\ \forall R & P \parallel\parallel R & \leq_S & Q \parallel\parallel R \quad (4) \\ \forall e & e \Rightarrow P & \leq_S & e \Rightarrow Q \quad (5) \end{array}$$

preuve

La relation $\leq_S$ a la même structure que la relation $\leq$. Elle implique de ce fait une propriété voisine de celle qui a été établie en 3.1.2.1 relativement aux états des processus comparés. La preuve des propriétés de précongruence de la relation $\leq_S$ est donc analogue à celle qui a permis de conclure à la propriété de précongruence de la relation $\leq$ en modèle asynchrone.

Fin-preuve

3.3 Propriétés algébriques de l'ensemble des processus

Quel que soit le mécanisme de synchronisation du système, on peut dans un premier temps partitionner l'ensemble des processus en classes d'équivalences définies par un ensemble de traces observables puisque les relations ne sont définies qu'entre des processus satisfaisant cette équivalence.

L'objet de cette section est d'étudier plus finement la structure de ces classes d'équivalence en fonction des différentes relations de préordre et d'équivalence qui ont été définies dans les sections précédentes (implémentation correcte en modèle synchrone et en modèle asynchrone).

Comme cela a été présenté précédemment, l'essentiel de la comparaison de deux processus repose sur des comparaisons d'ensembles d'états. Il est donc naturel, pour étudier la structure des ensembles de processus d'étudier celle des ensembles caractéristiques du non déterminisme des ensembles d'états susceptibles de réaliser immédiatement un ensemble d'événements donné.

Dans une première section, nous montrerons que pour un ensemble d'événements donné E, le sous-ensemble de $\mathbf{P}(\mathbf{P}(E))$ (on notera $\mathbf{P}(X)$ l'ensemble des parties d'un ensemble X) qui peuvent caractériser un ensemble d'états (être leur D(S), chaque D(S) est un élément de $\mathbf{P}(\mathbf{P}(EVTS(S)))$) forment un treillis distributif (*). Nous en déduirons, dans une seconde section, la structure de treillis distributif des ensembles de processus caractérisés par un ensemble de traces.

3.3.1 Treillis des ensembles caractéristiques des ensembles d'états

On dira qu'un ensemble X, élément de $\mathbf{P}(\mathbf{P}(E))$ est caractéristique d'un ensemble d'états si et seulement si il existe un processus d'états finis P et un ensemble d'états S inclus dans l'ensemble d'états de P tels que $D(S, P) = X$.

Le non déterminisme d'un processus susceptible de réaliser un ensemble d'événements E est exprimé par la fonction D qui prend ses valeurs dans l'ensemble $\mathbf{P}(\mathbf{P}(E))$. Toutefois, cette fonction n'est pas surjective, puisque tous les éléments de $\mathbf{P}(\mathbf{P}(E))$ ne sont pas caractéristiques d'un ensemble d'états.

(*) Dans cette section qui n'étudie que les ensembles d'états, on pourra omettre, pour simplifier l'écriture, la référence au processus dans les formules utilisant les fonctions D, evts ou EVTS.

Considérons par exemple un ensemble d'événements E = {a , b } et X= {{ a }} un élément de $\mathbf{P}(\mathbf{P}(E))$. X n'est pas caractéristique d'un ensemble d'états. En effet, si à partir de tout état de S la communication est certaine avec un environnement qui tente a alors elle est certaine avec un environnement qui tente a et b. Par conséquent, si { a } appartient à un D(S), alors { a , b } appartient au même D(S).

L'ensemble $\mathbf{D}(E)$, image par D de l'ensemble des ensembles d'états suceptibles de réaliser E peut être formellement défini par:

$$\mathbf{D}(E) = \{ X \mid \exists S, \exists P \ EVTS(S, P) = E \ \text{et} \ D(S, P) = X \}$$

On peut caractériser les éléments de $\mathbf{D}(E)$. Ils présentent en effet une propriété analogue aux propriétés de fermeture des ensembles d'acceptation définis dans [Hen 85].

Soient E un ensemble d'événements et F un élément de $\mathbf{P}(\mathbf{P}(E))$,

$$F \in \mathbf{D}(E) \Leftrightarrow \quad \emptyset \notin F$$

$$\text{et} \quad X \in F \ \wedge \ X \subseteq X' \subseteq E \Rightarrow \ X' \in F$$

preuve:

Condition nécessaire : immédiat

Condition suffisante :

On montre qu'à tout ensemble F satisfaisant la double condition, on peut associer un ensemble d'états caractérisé par F.

1) $F = \emptyset$: $S = \{ s_b , s_t \}$ où $evts(s_b) = \emptyset$ et $evts(s_t) = E$

2) $F \neq \emptyset$

Nous nous proposons de définir un ensemble d'états à partir de chaque ensemble F satisfaisant la condition, en associant une interprétation logique à tout ensemble d'événements X, élément de F. Nous montrerons ensuite que l'ensemble d'états ainsi construit est caractérisé par l'ensemble F qui a permis de le construire.

Tout élément $X = \{ x_1 , x_2 , \ldots , x_n \}$ de F doit garantir l'évolution certaine à partir de S, donc tout état de S dérive par l'un des x_i de X.

$$I(X) = \forall s, \ \Sigma_{x \in X} \ p_x(s) \quad \text{où } p_x = \lambda s \, . \, x \in evts(s)$$

L'ensemble F peut alors être interprété par le produit logique de toutes les propositions I(X) associées à ses éléments.

$$I(F) = \Lambda_{X \in F}\, I(X) \quad (1)$$

$$= \Lambda_{X \in F}\, [\, \forall s,\ \Sigma_{x \in X}\, p_x(s)\,)\,] \quad (2)$$

$$= \forall s,\ \Lambda_{X \in F}\ (\Sigma_{x \in X}\, p_x(s)\,) \quad (3)$$

$$= \forall s\ ,\ \Sigma_{Y \in P(F)}\ (\ \Lambda_{y \in Y}\ \ p_y(s)\,) \quad (4)$$

où P(F) est le produit "cartésien" des éléments de F défini par: $P(F) = \{ G \mid \forall X \in F, X \cap G \neq \emptyset \}$

L'interprétation de F permet de définir un ensemble d'états en associant un état à chaque proposition $\Lambda_{y \in Y}\ p_y(s)$ pour tous les ensembles Y tels que $Y \in Pdt(F)$.

Chaque proposition $\Lambda_{y \in Y}\, p_y(s)$ est interprétable par "tout y, élément de Y, est possible à partir de s" ou encore evts(s) = Y. Globalement l'ensemble S(F) défini par I(F) est défini ainsi:

$$S(F) = \{ s \mid \exists Y, Y \in P(F) \wedge evts(s) = Y \}$$

Il reste à montrer que l'ensemble d'états ainsi construit est caractérisé par l'ensemble F

1) $s \in S(F) \Leftrightarrow (\forall X \in E)(\exists x \in X)\ p_x(s)$ donc $F \subseteq D(S(F))$ (cf (3))

2) Montrons que pour tout $Y \in D(S(F))$ il existe $X \in F$ tel que $X \subseteq Y$

Supposons qu'il existe Y tel que Y soit élément de D(S(F)) et Y ne soit pas élément de F.

Par définition, Y est un ensemble d'événements inclus dans E et Y ne contient aucun des éléments de $F(\forall X \in F, X \not\subseteq Y)$. Il existe donc dans chaque ensemble X élément de F un événement e_X qui n'est pas élément de Y. L'ensemble de ces événements e_X définit un état s_0 de S(F). L'intersection de evts(s_0) avec Y étant vide, Y ne peut pas être élément de D(S(F)).

Fin-preuve

Exemple 1: E = { a, b, c }

F = { { a } , { a , b } , { a , c } , { b , c } , { a , b , c } }

F satisfait les conditions du lemme:

S = { s1 , s2 | evts(s1) = { a , c } et evts(s2) = { a , b } }

Exemple 2: E = { a, b, c }

F = { { a , b } , { a , c } , { a , b , c } }

F satisfait les conditions du lemme

S = { s1 , s2 | evts(s1) = { a } et evts(s2) = { b , c } }

Exemple 3 : E = { a, b, c }

F = { { a } , { a , b } , { a , b , c } }

L'ensemble d'états que l'on obtient en interprétant l'ensemble F est le suivant:

S = { s1 , s2 , s3 , s4 | evts(s1) = { a }, evts(s2) = { a , b }, evts(s3) = { a , c },
evts(s4) = { a , b , c }}

E ne satisfait pas le lemme, D(S) contient { a , c }. Il est, en fait, impossible de construire un ensemble d'états S dont le D(S) contienne { a } et pas { a , c }.

Remarque : $F \neq \emptyset \Rightarrow E \in F$

Compte tenu de la propriété de caractérisation des éléments de **D**(E), il apparaît que tout ensemble caractéristique F contient des éléments privilégiés, ceux qui ne sont sur-ensemble d'aucun autre élément de F({ a } et {b , c } dans l'exemple 1, {a , b } et {a , c } dans l'exemple 2). Nous allons maintenant reformuler les propriétés précédentes à partir de ces éléments privilégiés.

Nous définirons tout d'abord la notion de famille d'un sous ensemble F d'un ensemble E comme l'ensemble des sur-ensembles de F inclus dans E.

Définition:

Soient E un ensemble, **P**(E) l'ensemble des parties de E, X un sous-ensemble de **P** (E) et F un élément de **P**(E) ; X est la famille de F dans E, on notera X= famille(F , E) ssi

$$[\ F' \in \mathbf{P}\ (E) \wedge F \subseteq F' \Rightarrow F' \in X\ ]$$

Propriété :

Soient E un ensemble d'événements et F un élément de **P** (**P** (E)). F est caractéristique d'un ensemble d'états vérifiant EVTS(S) = E ssi F est union de n(≥0) familles de sous-ensembles non-vides de E.

(on suppose que l'union n-aire est définie pour n=0 par ø et pour n = 1 par $\cup$(X) = X)

Corollaire :

Les opérateurs d'union et d'intersection ensemblistes sont stables sur **D**(E)

Théorème

Etant donné un ensemble d'événements E, l'ensemble **D**(E) muni de l'intersection et de l'union comme opérations inf et sup et de la relation d'ordre d'inclusion est un treillis distributif.

Preuve : Le corollaire précédent permet de déduire que **D**(E) est un sous-treillis du treillis distributif **P**(**P**(E)).

Définition : (rappel)
Un élément "a" d'un treillis (T,inf, sup, ≤) est join-irréductible ssi

(∀ x , y ∈ T) [sup(x , y) = a ⇒ (x = a) ou (y = a)]

Théorème :

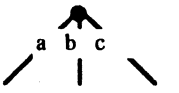
Un élément F d'un treillis (**D**(E) , ∩ , ∪ , ⊆) est join-irréductible ssi

(F= ø) ∨ (∃ f ∈ F, F = famille(f , E))

preuve : Conséquence du corollaire puisque ce théorème exprime qu'un élément irréductible de **D**(E) est soit vide soit réductible à une seule famille.

Exemple : Calcul du treillis **D**(E) où E = { a , b , c }

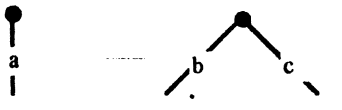
{ } *



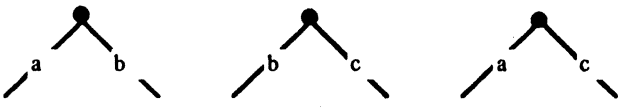
{{a,b,c}} *

{{a,b},{a,b,c}} *
{{a,c},{a,b,c}}
{{b,c},{a,b,c}}

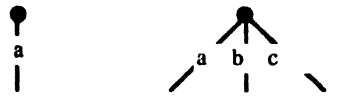
{{a,b},{a,c},{a,b,c}} *
{{a,b},{b,c},{a,b,c}}
{{a,c},{b,c},{a,b,c}}



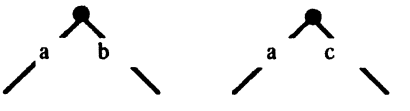
{{a,b},{a,c},{b,c},{a,b,c}} *



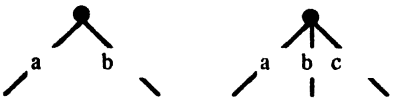
{{a},{a,b},{a,c},{a,b,c}} *
{{b},{a,b},{b,c},{a,b,c}}
{{c},{a,c},{b,c},{a,b,c}}



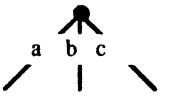
{{a},{a,b},{a,c},{b,c},{a,b,c}} *
{{b},{a,b},{a,c},{b,c},{a,b,c}}
{{c},{a,b},{a,c},{b,c},{a,b,c}}



{{a},{b},{a,b},{a,c},{b,c},{a,b,c}} *
{{a},{c},{a,b},{a,c},{b,c},{a,b,c}}
{{b},{c},{a,b},{a,c},{b,c},{a,b,c}}

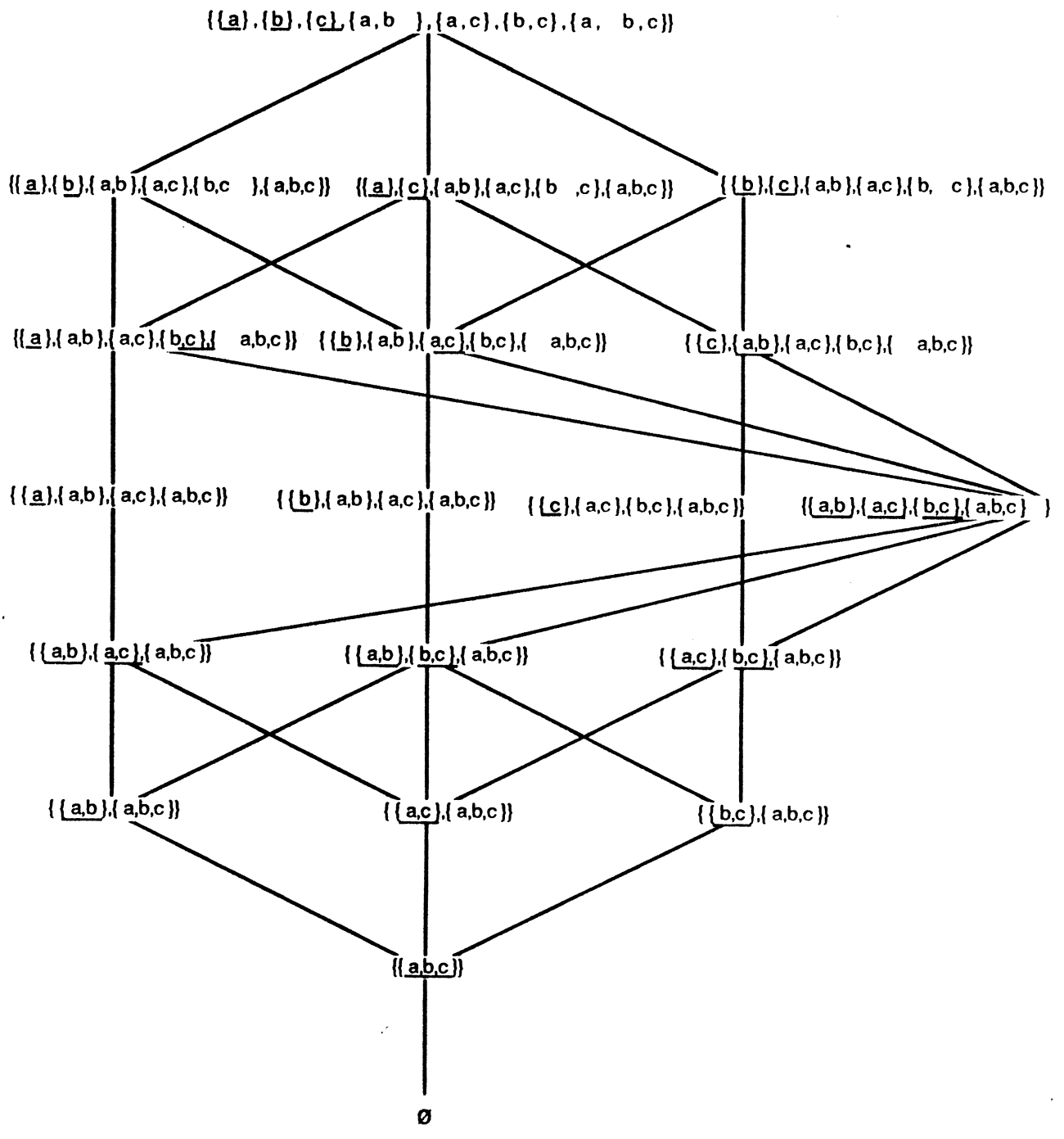


{{a},{b},{c},{a,b},{a,c},{b,c},{a,b,c}} *



Remarque : Les ensembles d'états représentés sont caractérisés par les ensembles marqués *.

Représentation du treillis **D**(E) où E = { a , b , c }



Remarque:

Chaque D(S) est union des familles des éléments soulignés:

{ { a } , { a , b } , { a , c } , { b , c } , { a , b , c } } = famille({a}) ∪ famille({b,c})

3.3.2 : Treillis de processus

L'ensemble Π des processus d'états finis est partitionné en classes d'équivalence, chaque classe étant caractérisée par un ensemble de traces(ou de traces observables).

Définition:

Soit P un processus d'états finis. La classe de P notée Cl(P) est définie par:

$$Cl(P) = \{ P' \in \Pi \mid TRACES(P') = TRACES(P) \}$$

Deux opérateurs binaires sup et inf peuvent être définis entre deux processus ayant des ensembles de traces identiques. Le maximum (resp le min) de deux processus est caractérisé après chaque trace par le maximum(resp le min) des caractéristiques des processus après cette trace.

Définitions :

$\forall P1 , P2 \in Cl(P)$, on définit deux opérations binaires inf et sup:

$sup(P1,P2) = X \Leftrightarrow \forall t,\ D(X(t)) = D(P1 (t)) \cup D(P2 (t))$

$inf(P1,P2) = N \Leftrightarrow \forall t, D(N (t)) = D(P1 (t)) \cap D(P2 (t))$

Théorème :

Toute classe Cl(P) quotientée par la relation $\approx$, munie des opérations inf et sup et de la relation d'ordre $\leq$ est un treillis distributif.

preuve: conséquence du 3.3.1.1.

On peut noter que tout treillis est borné inférieurement par le processus le moins déterministe (tout ensemble d'états accessible présente un état de blocage) et supérieurement par le processus le plus déterministe (à partir de tout état s d'un ensemble d'états accessible S, l'ensemble EVTS(S) est possible: evts(s , P) = EVTS(S , P)).

Les éléments join-irréductibles de ces treillis peuvent être caractérisés comme ceux qui, par une trace au plus, peuvent ne pas atteindre un état de blocage et qui dans ce cas sont caractérisés par un élément join-irréductible du treillis $\mathbf{D}(E)$.

Théorème:

Un processus R d'un treillis Cl(P) est join-irréductible ssi

1) Card({ t | D(R (t)) ≠ ø }) ≤ 1

2) D(R (t)) ≠ ø ⇒ D(S , R) est join-irréductible dans **D**(EVTS(S , R))

Preuve:

1) Tout processus vérifiant ces deux propriétés est irréductible:

Si Card({ t | D(R (t)) ≠ ø }) = 0 alors PROC= P_{min}

Sinon ∃ t_0, ∀ t ≠ t0 D(R (t)) = ø ∧ D(R (t 0)) ≠ ø

Supposons qu'il existe deux processus P et Q tels que R = sup(P , Q). Ces deux processus sont inférieurs à R donc il doivent être caractérisés, pour toutes les traces par des ensemble inférieurs à ceux qui caractérisent R. Pour toutes les traces, à l'exception de la trace t0, ils sont donc caractérisés par l'ensemble vide. Pour la trace t0, puisque R est caractérisé par un élément irréductible, l'un des deux processus doit être caractérisé par le même ensemble. Par conséquent, l'un des deux processus est caractérisé pour toutes les traces par des ensembles identiques à ceux de R et lui est donc équivalent.

C.Q.F.D

2)La réciproque est évidente:

En effet, tout processus ne vérifiant pas l'une de ces deux conditions peut s'exprimer soit comme maximum de processus ayant moins d'ensembles d'états accessibles caractérisés par des ensembles vides, soit comme maximum de processus ayant des ensembles d'états caractérisés par des ensembles plus élémentaires.

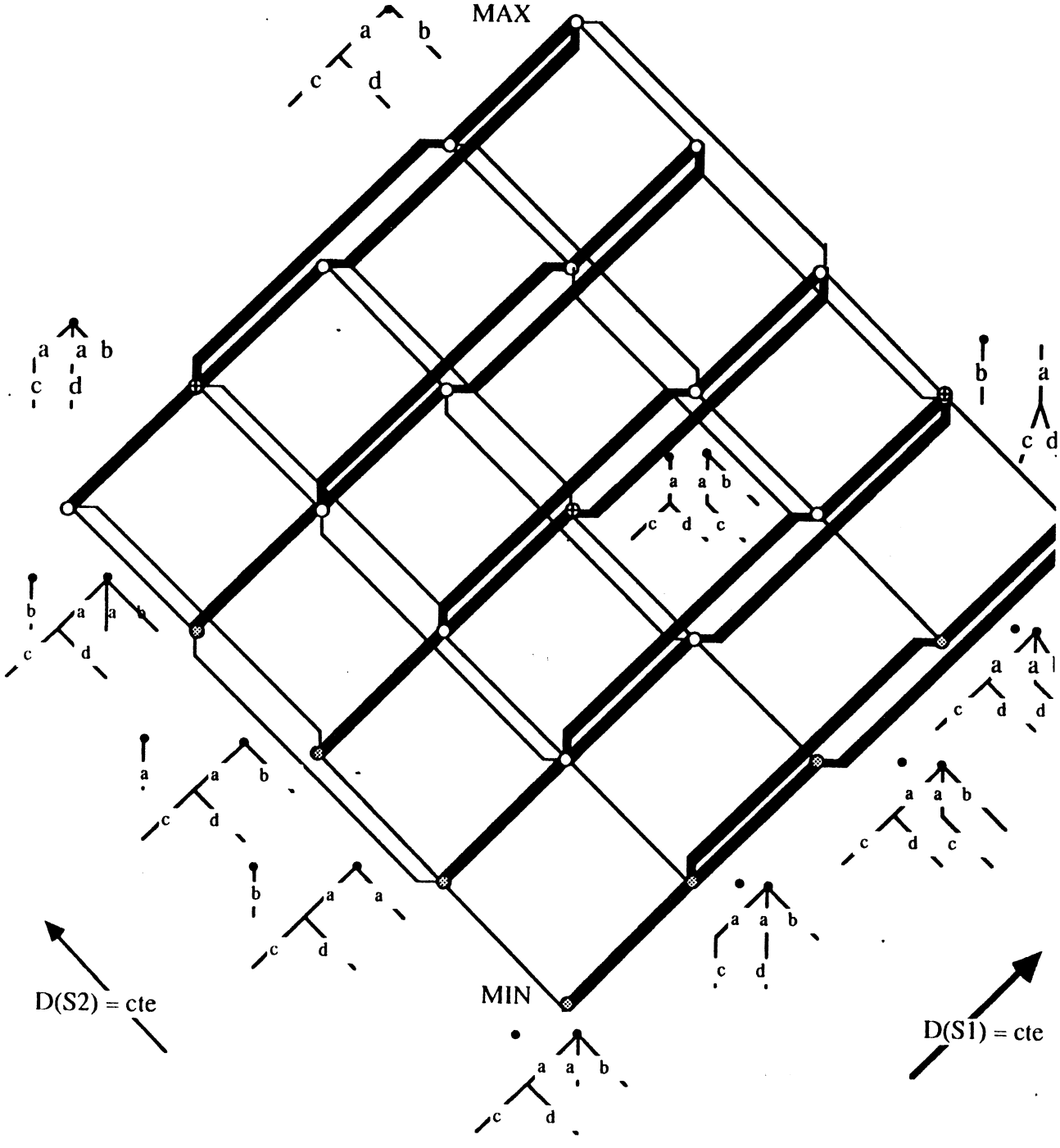
C.Q.F.D

Fin-preuve

Exemple : Représentation du treillis Cl(P) tel que

TRACES(P) = { < >, < a >, < b >, < a , c >, < a , d> }

(On note S1 l'ensemble d'états atteint par la trace < > et S2 celui atteint par la trace < a >)



Remarque :

Nous n'avons représenté que quelques éléments du treillis:

- Tous les éléments irréductibles (fond gris)
- Les éléments extrêmaux (MAX et MIN)
- 3 éléments internes (+ entouré)

3.3.3 Conclusion

Les résultats que nous avons présentés font apparaître l'ensemble des processus d'états finis comme un ensemble de treillis de processus, chaque treillis étant caractérisé par un ensemble de traces.

Il faut remarquer que nous n'avons pas fait référence à une notion particulière de trace et que ces résultats sont donc valides que l'on utilise les notions de trace et d'événements possibles ou celles de trace observable et événements observables (modèle asynchrone). La partition de l'ensemble des processus sera plus fine dans le premier cas que dans le second, chaque treillis obtenu dans le deuxième cas étant union de treillis obtenus dans le premier.

Enfin, compte tenu de la structure des relations qui exigent que les processus comparés aient des ensembles de traces identiques, les structures induites par les relations que nous avons définies sont naturellement différentes de celles obtenues à partir des relations qui incluent comme élément de comparaison la différence de l'ensemble des traces: ensemble de treillis dans notre cas, inf demi treillis dans le cas des relations de TCSP.

4. Comparaison de processus FP2 en fonction du contexte d'utilisation

Au cours du troisième chapitre, nous avons étudié des relations entre processus avec l'objectif de caractériser le comportement de communication des processus.

Dans ce chapitre, nous définirons la notion de contexte d'utilisation des processus. Cette notion a pour objet de prendre en compte une connaissance partielle du comportement de l'environnement utilisateur comme paramètre de la comparaison. On obtiendra alors une forme générale d'expression des relations de telle sorte que les relations du chapitre trois apparaîtront comme un cas particulier (elles seront définies relativement à un contexte particulier) et seront les relations les plus fines, les autres relations étant plus spécifiques dans le cadre d'un problème donné.

On peut envisager deux cas limites de connaissance sur le contexte utilisateur. D'une part, celui où aucune information particulière n'est connue sur son comportement: c'est le cas du contexte implicitement utilisé dans le troisième chapitre. D'autre part, le cas où les processus à comparer sont connectés à un autre processus pour constituer un réseau fermé, c'est à dire un réseau dont les processus ne communiquent qu'entre eux.

Il faut remarquer que seuls les réseaux fermés peuvent évoluer puisqu'une communication exige la participation de tous les interlocuteurs. C'est d'ailleurs le point de vue qui a été développé dans [Sch 86b] pour réaliser une implémentation du langage FP2, les utilisateurs des programmes FP2 étant eux-mêmes considérés, à travers des processus "clavier" et "écran", comme des processus d'un réseau fermé.

Dans la première section, nous définirons la notion d'implémentation correcte d'un processus par un autre vis à vis d'un processus fixe d'un réseau fermé.

Dans les seconde, troisième et quatrième sections, nous étudierons une notion de contexte plus générale. Dans le cas de réseaux fermés, l'environnement est complètement défini. La notion de contexte que nous définirons dans la seconde section, permettra d'introduire plus d'incertitude sur le comportement de l'environnement. Dans la troisième section, nous étudierons plus précisément le pouvoir discriminant des processus afin d'établir une relation entre les contextes. Enfin, dans la quatrième section, nous donnerons plusieurs exemples de contextes.

Dans la dernière section, nous rapprocherons cette notion de contexte de la notion d'équivalence liée à un ensemble d'observateurs définie dans [HN 84] et des différentes propositions qui ont été faites pour paramétrer les relations de comparaison des comportements.

4.1 Relations entre processus dans un réseau fermé

4.1.1 Présentation informelle

On considère les réseaux R1 et R2 représentés par les schémas suivants, le processus R et les liaisons entre Pi et R étant identiques dans les deux réseaux:

On cherche à définir des relations qui traduisent l'équivalence des processus P1 et P2 "vus par" le processus R. Les événements qui ont lieu dans un réseau fermé sont par définition des événements internes. Toutefois, dans ce cadre, les événements de communication entre P et R ne seront plus représentés anonymement par τ. Au contraire, on notera une communication sur un connecteur interne A.A' par ce connecteur et donc un événement par l'ensemble des connecteurs utilisés pour les différentes communications.

Les événements internes de chacun des processus P1, P2, R sont toujours notés τ et interprétés dans un modèle asynchrone. On pourra donc supposer, sans perte de généralité, que tous les processus sont des processus rigides.

On dira que les processus P1 et P2 sont équivalents pour R ssi les deux conditions suivantes sont réalisées:

1) Toutes les séquences d'événements réalisables entre P1 et R sont réalisables entre P2 et R et réciproquement.

2) Les deux réseaux présentent les mêmes situations de deadlock: R1 (R2) ne peut se bloquer après avoir réalisé une séquence d'événements que si R2 (R1) peut se bloquer.

On dira que le processus P2 est une implémentation correcte du processus P1 pour R ssi:

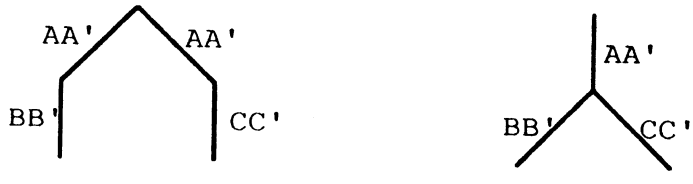
1) La condition sur l'égalité des séquences réalisables par les réseaux est satisfaite.

Cette condition est naturelle. Il semble en effet normal d'exiger de l'implémentation qu'elle permette au moins les mêmes possibilités que la spécification. Par ailleurs, une inclusion stricte signifierait que certains comportements sont possibles entre l'implémentation et R et qu'ils sont impossibles entre la spécification et R. A priori, ces comportements différents peuvent correspondre à des cas d'erreur.

2) Le réseau R2 présente au plus les mêmes situations de deadlock que le réseau R1:

Il est légitime de limiter ainsi les contraintes sur les comportements des réseaux, bien que cela conduise à des résultats qui peuvent sembler en contradiction avec ce qui a été présenté dans les chapitres précédents. La contradiction en fait n'existe pas si l'on considère plus précisément l'observateur: un observateur de processus, tel qu'il est considéré dans le cadre de la comparaison observationnelle, participe à l'évolution du réseau et est par conséquent sensible au non déterminisme du processus avec lequel il communique; un observateur de réseau fermé, au contraire, est extérieur au système et n'intervient pas dans son évolution. Nous montrerons formellement dans la quatrième section que la relation d'implémentation correcte de processus définie au troisième chapitre correspond à la relation définie selon les critères que nous venons de présenter, pour tous les processus-contextes.

Par ailleurs, lorsqu'on construit un réseau fermé indéterministe, on lui laisse la décision du choix de l'événement qui aura lieu lorsque plusieurs événements sont possibles compte tenu des états dans lesquels se trouvent les différents processus. Par conséquent, il n'y a pas lieu de tenir compte de la façon dont sont résolus ces choix. En particulier, nous considérerons que les deux réseaux fermés suivants sont équivalents.



Le premier réseau résoud le choix au moment où il réalise le premier événement tandis que le second le résoud après avoir réalisé le premier événement. Un observateur de ces réseaux verra d'abord une communication sur AA' puis une autre sur BB' ou sur CC' en fonction de choix qui auront été effectués par le réseau.

4.1.2 Définition de relations paramétrées par un processus-contexte

Avant de définir formellement ces relations, nous adopterons quelques conventions de notation pour alléger leur formulation.

* On notera P_R_C un réseau constitué des processus P et R et dont les connexions sont définies par C.

* Comme cela a été exposé plus haut, les relations utilisent la notion de séquence d'événements réalisables entre les processus P et R. On notera Traces(P_R_C) l'ensemble de ces séquences pour le réseau P_R_C.

* Compte tenu de la façon dont sont notées les communications (par le connecteur interne, c'est à dire par les deux connecteurs), il n'y a aucune ambiguité sur les événements des processus P et R qui sont sources d'un événement entre P et R (un événement {A.A', B.B'} ne peut être produit que par les événements { A , B } dans P et { A' , B' } dans R). Par abus de notation, si t est une trace de P_R_C, on notera également t les traces de P et de R qui sont à l'origine de t.

Le processus R est a priori un processus non-déterministe. Il peut donc atteindre, en réalisant une trace t avec P, l'un des états d'un ensemble d'états. Il est alors prêt à réaliser l'ensemble d'événements de l'état qu'il a atteint. Pour qu'après réalisation d'une trace t, la communication soit certaine entre P et R, il est nécessaire que, quel que soit l'état s atteint par R en réalisant t, l'ensemble d'événements qu'il est prêt à réaliser (evts(s , R)) permette la communication avec le processus P: evts(s , R) contient le "complément" d'un élément de D(P(t)). La double caractérisation du non déterminisme, par la fonction D d'une part et par l'ensemble des ensembles d'événements possibles à partir d'un ensemble d'états d'autre part, est donc ici très utile pour formaliser ce type de relations.

On définit alors une relation de préordre pour traduire la notion d'implémentation correcte pour un processus R. Celle-ci s'exprime par deux conditions: la première exige que les ensembles de traces des réseaux soient identiques, la seconde que s'il n'y a pas deadlock avec le plus petit alors il n'y a pas dealock avec le plus grand.

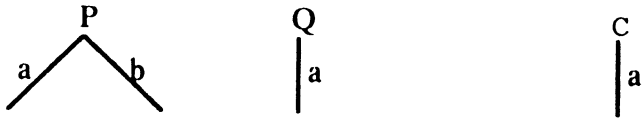
$$P \leq_R Q \Leftrightarrow$$

$$\exists\, T,\ Traces(P\_R\_C) = Traces(Q\_R\_C) = T$$

et $$\forall\, t \in T \mid (\forall\, s \in R(t))\ (\exists\, E \in D(P(t)))\ (E \subseteq evts(s, R))$$

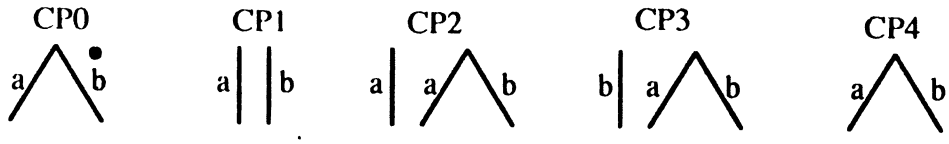
$$\Rightarrow (\forall\, s \in R(t))\ (\exists\, E \in D(Q(t)))\ (E \subseteq evts(s, R)) \mid$$

Exemples: Ex1



Les processus P et Q sont équivalents dans le contexte C qui ne peut pas communiquer pour réaliser l'événement b.

Ex2



On note Ci (ou Pi) le processus-contexte (ou le processus) dont le comportement est décrit par CPi.

Le contexte C0 induit une seule classe d'équivalence : { P0 , P1, P2, P3, P4 }.
Face à C0 tous les processus sont équivalents puisque tous les réseaux peuvent se bloquer.
Les contextes C1, C2,C3,C4 induisent deux classes d'équivalence :
La classe d'équivalence des processus qui conduisent à des réseaux qui peuvent se bloquer:
{ P0 , P1, P2, P3}pour C1, { P0 , P1, P3}pour C2, { P0 , P1, P2}pour C3, {P0 }pour C4
La classe d'équivalence des processus qui conduisent à des réseaux qui ne peuvent pas se bloquer:
{ P4 }pour C1, { P2 , P4}pour C2, { P3 , P4}pour C3, {P1, P2, P3, P4 }pour C4
Fin-exemple

4.1.3 Relations entre processus-contextes

La discrimination faite entre les processus vis-à-vis d'un processus R fixe dans un réseau fermé distingue à chaque étape, parmi l'ensemble des processus capables de réaliser avec R un même ensemble d'événements ceux qui peuvent toujours communiquer avec R de ceux pour lesquels il peut exister une situation de deadlock avec R. On peut établir, entre les processus considérés comme des contextes, une équivalence en s'appuyant sur le fait que si un processus accepte toujours de communiquer face à un ensemble d'événements alors il accepte toujours de communiquer face à un ensemble d'événements plus grand.

L'équivalence des contextes est définie par l'égalité des équivalences induites:

$$C1 \equiv C2 \text{ ssi } \forall P, Q \; P \approx_{C1} Q \Leftrightarrow P \approx_{C2} Q$$

Elle est caractérisée par la relation suivante:

$$C1 \equiv C2 \Leftrightarrow \exists T, Traces(C1) = Traces(C2) = T \quad (1)$$

et $$\forall t \in T \; [\; (\forall s1 \in C1(t)) \; (\exists s2 \in C2(t)) \; (evts(s2, C2) \subseteq evts(s1, C1)) \quad (2,1)$$

$$\wedge (\forall s2 \in C2(t)) \; (\exists s1 \in C1(t)) \; (evts(s1, C1) \subseteq evts(s2, C2)) \;] \quad (2,2)$$

Preuve:

Condition nécessaire

Si la première partie de la condition (1) n'est pas satisfaite, il existe deux processus qui ont des ensembles de traces identiques avec l'un des contextes et différents avec l'autre contexte tout en ayant des possibilités de communication identiques pour les traces communes. Ces processus sont équivalents dans l'un des contextes et différents dans l'autre.

Si la seconde partie de la condition n'est pas satisfaite alors pour une trace t l'un des deux contextes est nécessairement discriminant. Supposons que la condition (2,1) soit fausse. Le contexte C2 est discriminant pour t. Si le contexte C1 est discriminant alors il existe un état s_0 pour lequel la condition n'est pas satisfaite et un processus $\Delta 12$ qui par t ne peut atteindre qu'un état dont l'ensemble d'événements est égal à:

$$\cup_{s2 \in C2(t)} evts(s2, C2) - evts(s0, C1)$$

Le processus le plus déterministe est distingué du processus $\Delta 12$ par le contexte C1 mais ne l'est pas par le contexte C2. Le contexte C1 ne peut donc pas être discriminant comme le contexte C2 (de même que dans le cas où le contexte C1 ne serait pas discriminant).

Condition suffisante

De l'égalité des ensembles de traces des contextes on déduit l'égalité des ensembles de traces des réseaux fermés.

Il reste à montrer que si les conditions (2,1) et (2,2) sont satisfaites alors on a:

$$[(\forall s1 \in C1(t))(\exists E \in D(P(t)))(E \subseteq evts(s1, C1)) \Leftrightarrow (\forall s1 \in C1(t))(\exists E \in D(Q(t)))(E \subseteq evts(s1, C1))]$$
$$\Rightarrow [(\forall s2 \in C2(t))(\exists E \in D(P(t)))(E \subseteq evts(s2, C2)) \Leftrightarrow (\forall s2 \in C2(t))(\exists E \in D(Q(t)))(E \subseteq evts(s2, C2))]$$

Par hypothèse

$$(\forall s2 \in C2(t)) \; (\exists E \in D(P(t))) \; (E \subseteq evts(s2, C2))$$
$$\text{et} \; (\forall s1 \in C1(t)) \; (\exists s2 \in C2(t)) \; (evts(s2, C2) \subseteq evts(s1, C1))$$

donc $(\forall s1 \in C1(t))\ (\exists E \in D(P(t)))\ (E \subseteq evts(s1, C1))$
donc $(\forall s1 \in C1(t))\ (\exists E \in D(Q(t)))\ (E \subseteq evts(s1, C1))$
or $(\forall s2 \in C2(t))\ (\exists s1 \in C1(t))\ (evts(s1, C1) \subseteq evts(s2, C2))$
donc $(\forall s2 \in C2(t))\ (\exists E \in D(Q(t)))\ (E \subseteq evts(s2, C2))$
L'implication dans l'autre sens se montre de la même façon.
Fin-preuve

Cette équivalence entre les contextes peut être engendrée par un préordre fondé sur une inclusion des pouvoirs discriminants:

$$C1 \geq C2 \quad \underline{ssi} \quad \forall P, Q \quad P \approx_{C1} Q \Rightarrow P \approx_{C2} Q$$

La relation d'équivalence des processus considérés en fonction de leur pouvoir discriminant est identique à celle définie à partir de leur comportement de communication (ch 3): si deux processus ont le même comportement alors ils discriminent de la même façon et s'ils discriminent de la même façon alors ils ont des comportements identiques.

Toutefois, le préordre qui induit la relation entre les processus-contextes est différent de celui qui induit l'équivalence par identité des comportements:

En effet, un processus Q est une implémentation correcte d'un processus P ssi ils reconnaissent les mêmes traces et si lorsqu'après t, P accepte de communiquer face à un ensemble d'événements alors Q accepte aussi de communiquer face au même ensemble. Si la relation entre P et Q est stricte, alors le processus Q peut accepter de communiquer face à des ensembles d'événements qui conduiraient éventuellement à un refus de communication de la part de P. Ceci signifie que lorsqu'on considère P et Q comme des contextes des processus qui se trouvent dans la classe d'équivalence "blocage après t" avec P, sont dans la classe "non blocage après t" avec Q.

Deux processus P et Q non équivalents déterminent donc en général, sur un ensemble de processus caractérisé par un ensemble de traces, deux classes d'équivalence distinctes pour chaque trace et leurs pouvoirs discriminants ne sont donc pas comparables. Il faut cependant noter que la classe d'équivalence "blocage" est toujours définie puisqu'on peut toujours connecter à un processus qui présente un état de blocage. La classe d'équivalence "non blocage" n'est, au contraire, définie que si le processus-contexte ne présente pas lui même un état de blocage. Dans le cas contraire, tous les processus auxquels on peut le connecter sont équivalents et appartiennent à la classe "blocage" et ce contexte est strictement moins discriminant que tous les autres.

On peut se convaincre de ce fait en examinant les contextes proposés dans l'exemple précédent: C1, C2, C3, C4 définissent des classes d'équivalence différentes sur l'ensemble des processus { P0, P1, P2, P3, P4 } tandis que C0 ne définit qu'une classe d'équivalence.

Pour pouvoir être exprimée simplement, cette caractérisation doit faire appel à des définitions que nous introduirons ultérieurement. Nous n'en donnerons donc pas ici une caractérisation, celle-ci étant donnée dans la section 4.3 dans le cas de contextes plus généraux.

4.2 Relations dans un contexte

Dans cette section, nous introduisons une notion de contexte plus générale pour prendre en compte des environnements dont le comportement est seulement partiellement connu. Dans le premier paragraphe, un contexte est défini à partir des processus-contextes qui ont été étudiés dans la section précédente. Nous montrons ensuite qu'un tel contexte peut être représenté par un automate d'états finis étiqueté. Dans le dernier paragraphe, nous exprimons la notion d'implémentation correcte dans un contexte en utilisant la représentation des contextes par des automates.

4.2.1 Définition d'un contexte

Lorsqu'on considère un processus fixe dans un réseau fermé, l'environnement est totalement défini. Cependant, le comportement de l'environnement peut être seulement partiellement défini: l'ensemble des processus auxquels doivent être connectés les processus comparés peut être constitué de plusieurs processus, on peut définir certaines restrictions au comportement le moins défini (celui qui induit les relations du troisième chapitre): on peut par exemple considérer que lorsqu'on compare une spécification et une implémentation, l'implémentation est correcte dès lors qu'on se limite à l'utiliser conformément à la spécification.

D'une façon générale, un contexte peut apparaître comme un ensemble de processus. Les résultats qui seront présentés ne s'appliquent qu'à des contextes qui peuvent s'exprimer à l'aide d'un ensemble fini de processus.

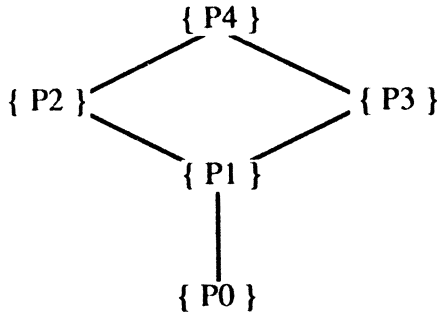
Si C est un contexte défini par un ensemble de processus {T1, ... , Tn} alors la relation d'implémentation correcte dans C est définie par

$$P \leq_C Q \quad \Leftrightarrow \quad \forall i \quad P \leq_{Ti} Q$$

Exemples: Ex 3

Les processus Pi ainsi que les processus-contextes Ci sont ceux qui ont été définis dans l'exemple 1 de la première section.C = { C0, C1, C2, C3, C4 }.

La relation de préordre engendrée par le contexte C sur l'ensemble de processus { P0, P1, P2, P3, P4 } lui confère la structure de teillis représenté sur le schéma suivant



Soit C' le contexte composé des processus T0,T1et T2:

$$C' = \{\ T0 : a\ \big|\ ,\ T1 : b\ \big|\ ,\ T2 : a \,\|\, b\ \}$$

Les préordres engendrés séparément par les processus-contextes T0,T1et T2 sont les suivants:

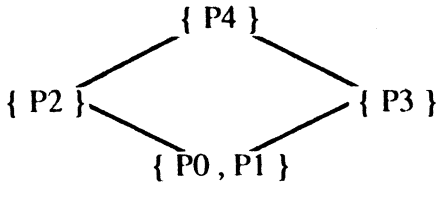
T0 : { P0, P1, P3 } < { P2, P4 }

T1 : { P0, P1, P2 } < { P3, P4 }

T2 : { P0, P1, P2, P3 } < { P4 }

(on peut remarquer que cette relation n'est pas induite par les deux précédentes)

d'où la structure suivante pour le préordre engendré par C' :



Fin-exemple

4.2.2 Représentation d'un contexte

4.2.2.1 Contexte = Automate d'états finis étiqueté

La notion de contexte définie par un ensemble de processus est difficile à manipuler formellement puisqu'elle suppose d'exprimer toute opération que l'on voudrait définir en termes de description de processus.

Dans cette section, nous proposons de définir une structure plus simple pour décrire les contextes, fondée sur les faits suivants:

1- Un contexte teste un ensemble de traces, l'union des ensembles de traces testées séparément par chacun des processus qui le définissent. Puisque cet ensemble est fini et que les processus sont réguliers, ce test peut être fait par un automate d'états finis.

2- Pour chaque trace, un contexte effectue un ensemble de tests sur les possibilités de communication: chaque processus définit un test. L'ensemble des traces est a priori infini. Cependant, en limitant les contextes à des ensembles finis de processus réguliers, il existe une régularité sur les tests effectués.

On pourra donc définir un contexte par un automate d'états finis dont les états sont étiquetés par un ensemble de tests sur les possibilités de communication. Le test effectué à chaque étape par un processus est exprimé par l'ensemble d'états qu'il peut atteindre. Chaque état n'est considéré qu'en tant que proposition de communication et peut donc se réduire dans ce cadre à un ensemble d'événements. On représentera donc un test par un ensemble d'ensembles d'événements.

L'***automate d'un contexte*** est défini de manière analogue aux automates de comportement par un quintuplet $< Q , \Omega , i , V , T >$, la seule différence étant la fonction Ω qui étiquette les états de l'automate (remplace la fonction Δ des automates de comportement).

$$\Omega : Q \longrightarrow \mathbf{P}(\mathbf{P}(\mathbf{P}(V)))$$

4.2.2.2 Calcul de l'automate d'un ensemble fini de processus

L'objet de cette section est de définir un algorithme permettant de construire l'automate d'un contexte à partir de ceux des processus qui le définissent. On notera $C l'automate d'un contexte C.

4.2.2.2.1 Automate d'un processus-contexte

L'automate associé à un processus-contexte P, noté, pour alléger l'écriture, $P et non ${P}, est défini par

- un état de l'automate est associé à un ensemble d'états du processus

$$Q(\$P) = \{ q_S \mid \exists t,\ S = P(t) \}$$

- l'étiquette d'un état associé à un ensemble d'états de P est définie à partir des ensembles d'événements possibles à partir des états

$$\Omega(\$P) = \lambda(q_S).\{E \mid E = \{ evts(s) \mid s \in S \}\}$$

- L'état initial est associé à l'ensemble d'états initiaux du processus

$$i(\$P) = q_{I(P)}$$

- Le vocabulaire de l'automate est défini par l'ensemble des événements observables de P

$$V(\$P) = \mathbf{P}(K(P)) - \{ \emptyset \}$$

- L'ensemble de transitions est défini par les fonctions successeurs entre ensembles d'états:

$$T(\$P) = \{ (q_S, e, q_{S'}) \mid S' = SUCC(S, <e>, P) \}$$

4.2.2.2.2 Automate d'un contexte

Il suffit maintenant de définir l'opération de composition d'automates qui permet de définir l'automate d'un contexte à partir de ceux des processus qui le composent.

Un état de l'automate correspond au test d'un ensemble de traces testant les mêmes possibilités de communication. Il sera étiqueté par l'ensemble des tests effectués séparément par chacun des processus susceptibles de tester ces traces. Les états successeurs sont définis à partir de l'ensemble des événements qui peuvent continuer cette trace.

Les ensembles de traces testés par les différents processus peuvent être distincts. On supposera qu'on peut étendre l'automate d'un contexte grâce à un état indéfini qu'on notera #, tel que si e n'est pas un événement possible à partir d'un état s de l'automate alors à partir de s l'automate atteint cet état par e. Par définition, puisque cet état est indéfini, l'ensemble de tests qui lui est associé est vide, c'est à dire que cet état est non discriminant.

Nous pouvons maintenant définir l'opération de composition d'automates qui permet de construire $C à partir de T_1$, ... T_n$, si C= {T_1, ... , T_n}:

- un état de l'automate est défini par un n-uplet d'états des automates qui composent le contexte. Un état correspond au test d'une trace par l'un au moins des processus qui composent le contexte. Puisque les processus Ti peuvent avoir des ensembles de traces différents, l'état # sera utilisé pour tous les processus qui ne réalisent pas une trace t dès lors que t est une trace de l'un des processus.

$$Q(\$C) = \{ [s_1, \dots , s_n] \mid \exists t \in \cup_i \text{Traces}(\$T_i), \quad \forall i, (s_i = \$T_i (t) \vee (t \notin \text{Traces}(\$T_i) \wedge s_i = \{\#\}) \}$$

- l'étiquette d'un état [s_1, ... , s_n] est définie par l'union des étiquettes associées aux états s_i

$$\Omega ([s_1, \dots , s_n]) = \cup_i \Omega(s_i) \quad \text{avec } \Omega(\#) = \emptyset$$

- L'état initial est défini par le n-uplet des états initiaux

$$i(\$C) = [i(\$T_1), \dots , i(\$T_n)]$$

- Le vocabulaire de l'automate est défini par l'ensemble des vocabulaires des T_i.

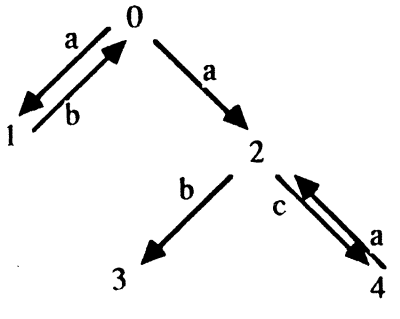
$$V(\$C) = \cup_i V(\$T_i)$$

- L'ensemble de transitions est défini de manière à ce que l'ensemble de traces du contextes soit l'union des ensembles de traces des processus qui composent le contexte.

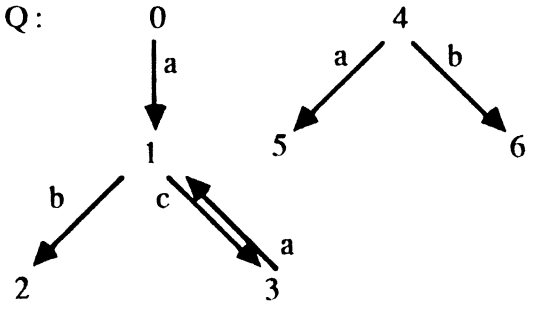
$$T(\$C) = \{ ([s_1, \dots ,s_n] , e , [s'_1, \dots , s'_n]) \mid \exists i, (s_i , e , s'_i) \in T(\$T_i) \wedge \forall i [(s_i , e , s'_i) \in T(\$T_i) \text{ ou } s'_i = \# \wedge e \notin \text{evts}(s_i , \$T_i)] \}$$

Exemple

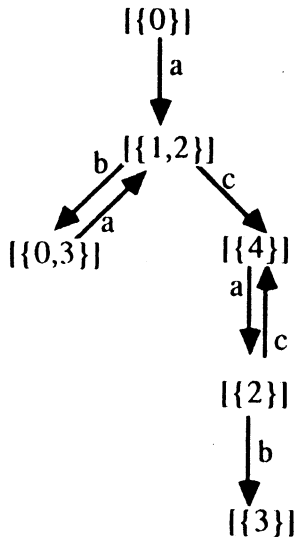
P :



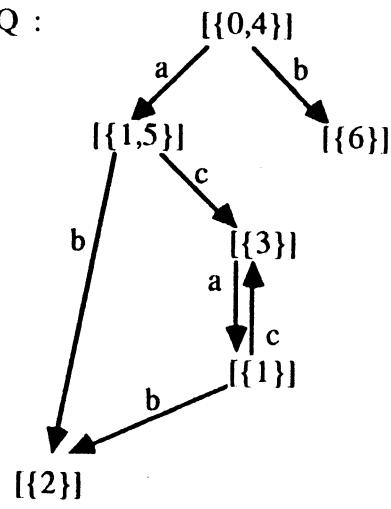
Q :



$P :

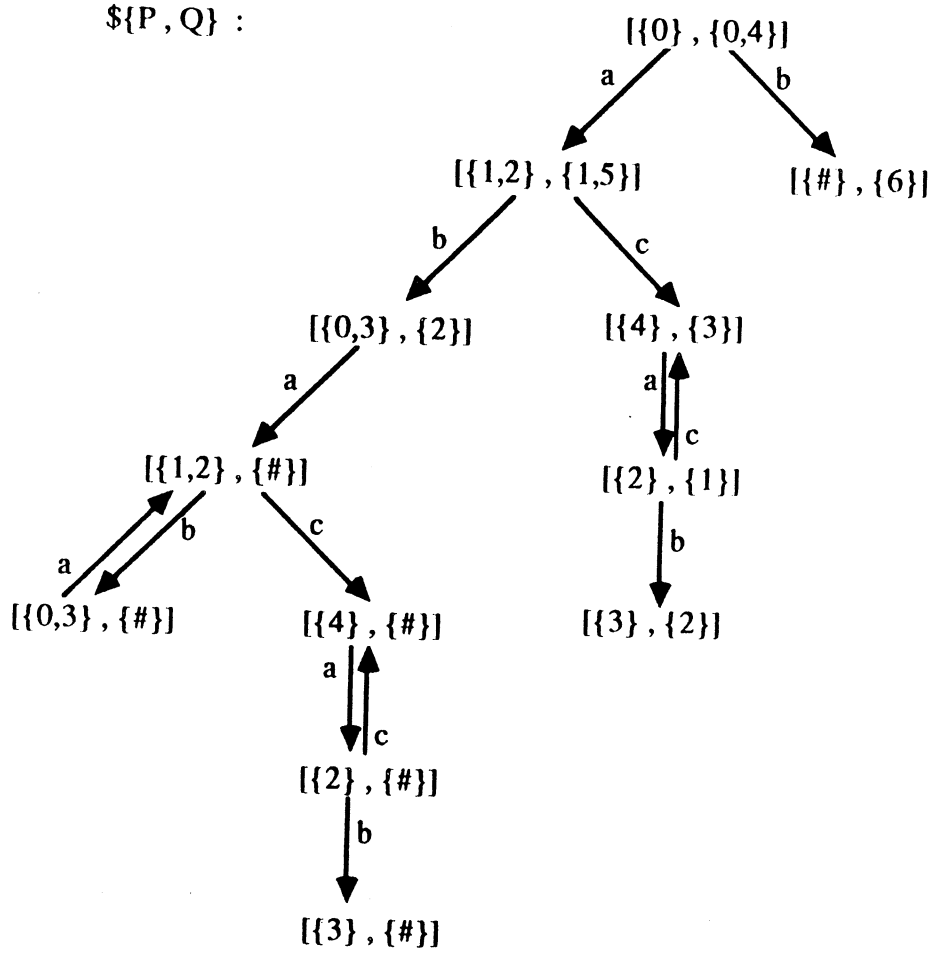


$Q :



$P	état : p_S	étiquette: $\Omega(p_S)$
	[{0}]	{{{a}}}
	[{1,2}]	{{{b},{b,c}}}
	[{0,3}]	{{ }}
	[{4 }]	{{{a }}}
	[{2}]	{{ {b}, {c},{b,c}}}
	[{3}]	{{ }}

$Q	état : q_S	étiquette: $\Omega(q_S)$
	[{0,4}]	{{{a} , {a,b}}}
	[{1,5}]	{{ }}
	[{2}]	{{ }}
	[{3}]	{{{ a }}}
	[{1}]	{{ {b}, {c},{b,c}}}
	[{6}]	{{ }}



$\{ P,Q \}$

état : p_S	étiquette: $\Omega(p_S)$
[{0}, {0,4}]	{{{a}}, {{a} , {a,b}} }
[{1,2}, {1,5}]	{{{b},{b,c}}, { } }
[{0,3}, {2}]	{{ }}
[{4 }, {3}]	{{{a }}}
[{2}, {1}]	{{ {b}, {c},{b,c}}}
[{3}, {2}]	{{ }}
[{1,2}, {#}]	{{{b},{b,c}}}
[{0,3}, {#}]	{{ }}
[{4 }, {#}]	{{{a }}}
[{2}, {#}]	{{ {b}, {c},{b,c}}}
[{3}, {#}]	{{ }}
[{#}, {6}]	{{ }}

Fin-exemple

4.2.3 Implémentation correcte dans un contexte

La relation d'implémentation correcte dans un contexte se déduit de celle qui a été définie dans la première section pour des contextes réduits à un seul processus. La condition qui doit être satisfaite à chaque étape est la conjonction de celles définies par les tests qui étiquettent l'état de l'automate.

$P \preceq_C Q \Leftrightarrow$

$\exists T, Traces(P\_C) = Traces(Q\_C) = T$

$\forall t \in T$

$\forall U \in \Omega(\$C(t)),$

$(\forall u \in U, \exists E \in D(P(t)), E \subseteq u) \Rightarrow (\forall u \in U, \exists E \in D(Q(t)), E \subseteq u)$

Compte tenu de la définition de l'automate associé au contexte, la relation définie ainsi correspond à la définition de la notion de contexte. En effet, l'automate de $C = \{ T_i \}$ a pour langage l'union des langages des processus T_i et si t est une trace de T_i alors le test sur les possibilités de communication effectué par T_i est l'un des tests effectués par le contexte.

4.3 Comparaison de contextes en fonction de leur pouvoir discriminant

Chaque contexte définit un préordre sur l'ensemble des processus. Il est intéressant de pouvoir comparer les contextes en fonction de leur pouvoir discriminant de façon à pouvoir utiliser une relation plus discriminante qui serait satisfaite par deux processus ou encore pour pouvoir déterminer si un ensemble de processus définit un contexte donné par ailleurs.

4.3.1 Définition

De la même façon que précédemment (§413) on peut définir un préordre sur les pouvoirs discriminants des contextes par l'inclusion des relations d'équivalences induites: C est plus discriminant que C' ($C \geq C'$) ssi $\approx_C \subseteq \approx_{C'}$. La propriété suivante montre que cette relation peut également être définie à partir de la relation d'implémentation correcte dans les contextes.

Propriété

$$C \geq C' \Leftrightarrow \forall P, Q \; P \leq_C Q \Rightarrow P \leq_{C'} Q$$

Preuve

Condition suffisante: évident puisque les équivalences sont engendrées par les préordres.

Condition nécessaire:

Soit à démontrer $(\forall P, Q \; P \approx_C Q \Rightarrow P \approx_{C'} Q) \Rightarrow (\forall P, Q \; P \leq_C Q \Rightarrow P \leq_{C'} Q)$.

On montre que $(\exists P, Q \; P \leq_C Q \wedge P \not\leq_{C'} Q) \Rightarrow (\exists P', Q' \; P' \approx_C Q' \wedge P' \not\approx_{C'} Q')$

Soient P et Q tels que $P \leq_C Q \wedge P \not\leq_{C'} Q$

P' et Q' tels que $D(P'(t)) = D(P(t))$ étendu à $EVTS(P(t)) \cup EVTS(Q(t))$

$D(Q'(t)) = D(P(t)) \cap D(Q(t))$ étendu à $EVTS(P(t)) \cup EVTS(Q(t))$

où E' "étendu à" E est la fermeture par convexité de $E' \cup \{ E \}$

1. Montrons que $P' \not\leq_{C'} Q'$

$P \not\leq_{C'} Q : \exists U \in \Omega(\$C'(t)),$

$$\forall u \in U, \exists E \in D(P(t)), E \subseteq u \quad \wedge \quad \exists u \in U, \forall E \in D(Q(t)), E \not\subseteq u$$

$* D(P(t)) \subseteq D(P'(t)) \Rightarrow \forall u \in U, \exists E \in D(P'(t)), \; E \subseteq u$

$*\forall E' \in D(Q'(t)), \exists E \in D(Q(t)), E \subseteq E' \Rightarrow \exists u \in U, \forall E \in D(Q'(t)), \; E \not\subseteq u$

donc $P' \not\leq_{C'} Q'$

2. Montrons que $P' \approx_C Q'$

$* P' \leq_C Q'$: Soit à démontrer $\forall t \in T, \forall U \in \Omega(\$C(t))$

$$\forall u \in U, \exists E \in D(P'(t)), \; E \subseteq u \Rightarrow \forall u \in U, \exists E \in D(Q'(t)), \; E \subseteq u$$

$\forall u \in U, \exists E \in D(P'(t)), \; E \subseteq u$

$\Rightarrow \forall u \in U, \exists E_P \in D(P(t)), \; E_P \subseteq u$

$\Rightarrow \forall u \in U, \exists E_Q \in D(Q(t)), \; E_Q \subseteq u$ -- $P \leq_C Q$

$\Rightarrow \forall u \in U, \exists E(=E_P \cup E_Q) \in D(Q'(t)), \; E \subseteq u$ --définition de $D(Q'(t))$

* $Q' \leq_C P'$: Soit à démontrer $\forall t \in T, \forall U \in \Omega(\$C(t))$

$\forall u \in U, \exists E \in D(Q'(t)),\ E \subseteq u \Rightarrow \forall u \in U, \exists E \in D(P'(t)),\ E \subseteq u$

$\forall u \in U, \exists E \in D(Q'(t)),\ E \subseteq u$

$\Rightarrow \forall u \in U, \exists E_P \in D(P(t)),\ E_P \subseteq u$ --définition de $D(Q'(t))$

$\Rightarrow \forall u \in U, \exists E_{P'} \in D(P'(t)),\ E_{P'} \subseteq u$ -- $D(P(t)) \subseteq D(P'(t))$

Fin-preuve

4.3.2 Quelques exemples

La difficulté essentielle pour caractériser cette relation entre les contextes tient à la structure complexe des tests portant sur les possibilités de communication des processus. Comme cela a été montré dans la première section deux tests discriminants ne sont pas comparables s'ils ne sont pas équivalents. Cependant, on peut voir qu'il existe des contextes qui sont strictement plus discriminants que d'autres.

Exemple1

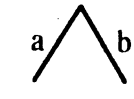
$C = \{ T : a \mid b \mid \}$ $\quad C' = \{ U : a \mid , U' : b \mid \}$

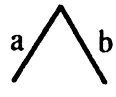
$\forall P, Q\ ,\ P \approx_{C'} Q \Rightarrow P \approx_C Q$:

$P \approx_{C'} Q : [a \in D(P) \Leftrightarrow a \in D(Q)] \wedge [b \in D(P) \Leftrightarrow b \in D(Q)]$

$\Rightarrow [(a \in D(P) \wedge b \in D(P)) \Leftrightarrow (a \in D(Q) \wedge b \in D(Q)] \Rightarrow P \approx_C Q$

Le contexte C n'est pas plus discriminant que le contexte C' comme le montre l'exemple des deux processus P et Q suivants: P et Q sont équivalents dans C et ne le sont pas dans C'.

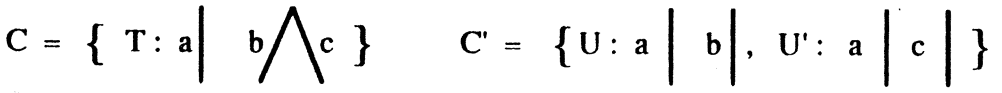
P1 : 

P2: 

Fin-exemple

Sur l'exemple précédent, il apparaît que les deux états du test T de C sont des états de l'un des tests U ou U' du contexte C'. Ceci peut être généralisé lorsqu'on considère parmi les états d'un test ceux qui sont nécessaires pour la discrimination (ceux qui ne sont pas sur-ensemble d'un autre état du même test). En effet, il est nécessaire pour qu'un test T soit moins discriminant qu'un ensemble de tests { U1, ... , Un } que chaque état discriminant de T soit élément d'un test Ui.

Exemple2



Le contexte C' n'est pas plus discriminant que le contexte C bien qu'il effectue plus de tests individuellement car l'état {b,c} du test T du contexte C n'est ni un état de U ni un état de U': il n'existe pas de tests de C' contenant cet état.

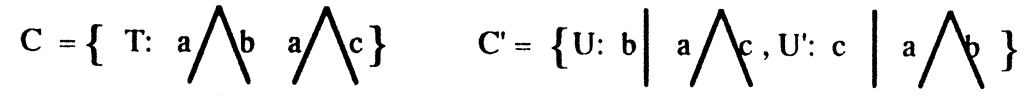
P1 et P2 ne sont pas équivalents dans le contexte C puisque D(P1) $\supseteq$ {{a},{b,c}} alors que {b,c} $\notin$ D(P2). Au contraire, ces processus sont équivalents dans C' puisque ni l'un ni l'autre ne satisfait b ou c séparément.

Par ailleurs le contexte C ne peut pas être plus discriminant que le contexte C' puisque C ne contient qu'un test alors que C' en contient deux non-équivalents.

Fin-exemple

Pour qu'un test T soit effectué par parties par un ensemble de tests C', il est nécessaire qu'on puisse associer à chaque état s de T un test T' de C' qui contienne s. Toutefois, le test T' de C' auquel doit correspondre s ne doit contenir que des états "compatibles" avec T.

Exemple3



Le contexte C' n'est pas plus discriminant que le contexte C bien que l'on retrouve pour chaque état de T un test de C' qui le contienne. Cependant, chaque test de C' associe à l'état de T une condition qui n'est pas prise en compte par T.

Ces deux processus ne sont pas équivalents dans le contexte C puisque {a,c} $\notin$ D(P2). Cependant P1 et P2 sont équivalents dans C' car la partie { a , c } du test T n'existe dans C' que dans le test U où elle est associée au test sur b qui n'est satisfait ni par P1, ni par P2.

Fin-exemple

Les tests du contexte le plus discriminant, auxquels on doit associer chaque état d'un test T du contexte le moins discriminant ne sont cependant pas nécessairement inclus dans le test T. Ils doivent uniquement contenir des états qui sont satisfaits dès lors que T est satisfait.

Exemple4

$C = \{ T: a \mid b \mid c \mid \}$ $\quad C' = \{U: a \mid b \bigwedge c , U': b \mid a \bigwedge c , U'' : c \mid\}$

Le contexte C est strictement moins discriminant que le contexte C'. En effet, pour que le test de C soit satisfait par un processus P, celui-ci doit être tel que $\{\{a\},\{b\},\{c\}\} \subseteq D(S, P)$. Compte tenu de la structure des ensembles définis par la fonction D, D(S , P) contient aussi {b , c} et {a , c}. Par conséquent, deux processus quelconques ne peuvent pas être équivalents dans C sans l'être dans C'.

Au contraire, les tests U1 et U2 ne sont pas réalisés par parties par C; C' ne peut donc pas être équivalent à C comme le montre l'exemple des deux processus de l'exemple 2.

Fin-exemple

A partir de ce dernier exemple, nous pouvons déduire la forme que prendra la relation caractéristique du pouvoir discriminant des contextes.

1- Puisque les relations dans les contextes sont fondées sur une conjonction de relations définies par des tests, pour qu'un contexte C soit plus discriminant qu'un contexte C', C devra satisfaire globalement une condition vis-à-vis de chaque test de C'.

2- Si T' est un test de C' alors à chaque état s' de T', on peut associer un test $T_{s'}$ de C satisfaisant la condition suivante: s' est un état de $T_{s'}$ et tous les états de $T_{s'}$ sont satisfaits si T' est satisfait.

4.3.3 Caractérisation de la relation de préordre

Définition

Un test U est réduit s'il satisfait la condition $\forall s \in U, \forall s' \in U, s \neq s' \Rightarrow s \not\subset s'$

On dira qu'un test U est équivalent à un test U' si le remplacement de U par U' dans l'étiquette d'un état d'automate de contexte ne modifie pas la relation d'équivalence engendrée par le contexte.

Propriété

Tout test U est équivalent à un test réduit $U' = \{ u \in U | \forall u' \in U, u \neq u' \Rightarrow u' \not\subset u \}$

Preuve : Evident puisque les tests ne permettent que de comparer les posssibilités de communication.

Théorème

Soient C et C' deux contextes dont toutes les étiquettes sont des ensembles de tests sous forme réduite,

$C \geq C' \Leftrightarrow$ Traces(C) $\supseteq$ Traces (C')

et $\forall t' \in$ Traces(C')

$\forall U' \in \Omega(\$C'(t))$,

$\emptyset \notin U' \Rightarrow \forall u' \in U', \exists U \in \Omega(\$C(t)), u' \in U \wedge \forall v \in U, \exists v' \in U', v' \subseteq v$

Preuve

Condition nécessaire

1- La première condition implique directement que si deux processus P et Q ont des ensembles de traces identiques dans le contexte C alors ils ont des ensembles de traces identiques dans le contexte C'.

2- Soient P et Q deux processus vérifiant $P \leq_C Q$. Montrons par l'absurde que $P \leq_{C'} Q$.

Supposons $P \not\leq_{C'} Q$.

$\exists t \in$ Traces(C'),$\exists U' \in \Omega(\$C'(t)), \forall u' \in U', \exists E \in D(P(t)), E \subseteq u'$

$\wedge \quad \exists w \in U', \forall E \in D(Q(t)), E \not\subseteq w$

D'après la propriété, il existe $U \in \Omega(\$C(t))$ tel que $w \in U$. Le test U du contexte C est donc un échec pour le processus Q.

Par ailleurs, U vérifie la propriété $\forall v \in U, \exists v' \in U', v' \subseteq v$. Par conséquent, la propriété qui définissait que P passait le test U' avec succès ($\forall u' \in U', \exists E \in D(P(t)), E \subseteq u'$) est conservée avec le test U du contexte C.

De ce fait, le test U du contexte C est passé avec succès par le processus P et est un échec pour le processus Q. Le processus Q n'est donc pas une implémentation correcte du processus P dans le contexte C.

Condition suffisante:

1- Si la condition Traces(C) $\supseteq$ Traces (C') n'est pas satisfaite alors il existe deux processus équivalents dans C et non équivalents dans C': pour une trace t de C' qui n'est pas une trace de C, un processus P qui n'a pas t comme trace et un processus Q qui ne diffère de P que par le fait qu'il contient t (et toutes les traces nécessaires pour avoir t) sont équivalents dans C et ne sont pas équivalents dans C'.

2- Nous allons montrer que si la seconde partie de la condition est fausse, alors il existe deux processus P et Q tels que Q soit une implémentation correcte de P dans C et ne le soit pas dans C'.

Si la condition est fausse alors:

$\exists$ t' $\in$ Traces(C'), $\exists$ U' $\in$ Ω(\$C'(t)),

$\emptyset \notin$ U' $\wedge \exists$ u'(=w) $\in$ U',$\forall$ U $\in$ Ω(\$C (t)), u' $\notin$ U'$\vee \exists$ v $\in$ U,$\forall$ v' $\in$ U', v' $\not\subseteq$ v

Soient P et Q les deux processus définis par

a) P et Q ont des comportements identiques sauf pour la trace t':

traces(P) = traces(Q) = T, t' $\in$ T

$\forall$ t($\neq$t') $\in$ T, D(P(t)) = D(Q(t))

EVTS(P(t')) = EVTS(Q(t')) = EVTS(C(t')) =**E**

b) Pour la trace t', P et Q ont des comportements de communication différents:

$D(P(t')) = \cup_{u' \in U'}$famille(u' , E)

$D(Q(t')) = \cup_{U \in F(C, U')}(\cup_{u \in U}$famille(u , E))

où F(C , U')= {U $\in$ C(t') | $\forall$ u $\in$ U,$\exists$ u' $\in$ U', u' $\subseteq$ u }

commentaires:

* $D(Q(t')) \subseteq D(P(t'))$
* Par définition, P satisfait le test U'
* Q ne satisfait que les tests de C qui seront satisfaits par P si P satisfait U'.

Montrons que $P \leq_C Q$:

Soit U un test de C(t').

$U \in F(C, U')$: U est satisfait par P et par Q

$U \notin F(C, U')$: $\exists u_0 \in U, \forall u' \in U', u' \not\subseteq u_0$

$\forall E \in D(P(t')), \exists u' \in U', u' \subseteq E$, donc $\forall E \in D(P(t')), E \not\subseteq u_0$

Puisque $D(Q(t')) \subseteq D(P(t'))$, il vient $\forall E \in D(Q(t')), E \not\subseteq u_0$

donc $P \approx_C Q$

Montrons que $P \not\leq_{C'} Q$:

$\forall u' \in U', \exists E \in D(P(t')), E \subseteq u'$

$w \in U', \forall E \in D(Q(t')), E \not\subseteq w$. En effet,

$E \in D(Q(t')) \Leftrightarrow \exists Y \in F(C, U'), \exists y \in Y, y \subseteq E$;

puisque $Y \in F(C, U'), \exists u' \in U', u' \subseteq y$

or par définition de F, $\forall U \in F(C, U'), w \notin U$.

Donc s'il existe $E \in D(Q(t'))$ tel que $E \subseteq w$, on a $u' \subseteq y \subseteq E \subseteq w$ où u' et w sont des éléments de U'. Ceci est contraire aux hypothèses: C' est sous forme réduite et $\emptyset \notin U'$.
Par conséquent, Q n'est pas une implémentation correcte de P dans C'.
Fin-preuve

4.4 Exemples de contextes

Nous avons souligné dans la première section que la notion de contexte avait pour objet de prendre en paramètre de la comparaison des processus une connaissance a priori sur l'environnement des processus, un cas limite étant celui où le comportement de l'environnement est totalement inconnu.

4.4.1 Contexte des relations du troisième chapitre

Le premier exemple que nous présenterons est celui de l'environnement inconnu, permettant de retrouver les relations définies au troisième chapitre.

La première chose à préciser pour définir un contexte est l'ensemble des événements à partir duquel sera défini l'ensemble des traces testées. Puisqu'un contexte est décrit par un automate fini, cet ensemble d'événements est nécessairement fini.

L'ensemble des événements de communication d'un processus ayant un ensemble de connecteurs K est $\mathbf{P}(K) - \{\emptyset\}$. Pour comparer des processus P et Q ayant des ensembles de connecteurs K_P et K_Q, on utilisera $EV_{P,Q} = \mathbf{P}(K_P) \cup \mathbf{P}(K_Q) - \{\emptyset\}$ comme ensemble d'événements.

Toute trace d'un processus est un mot du langage EV*. L'automate du contexte décrit donc ce langage.

Il reste à définir l'ensemble des possibilités de communication qui seront testées. Cet ensemble sera indépendant de la trace testée et sera représenté par l'ensemble des propositions de communications possibles. Par conséquent, un test sera défini par un seul ensemble d'événement, sous-ensemble de $EV_{P,Q}$.

Exemple: CAB : Contexte inconnu pour des processus définis sur l'ensemble de connecteurs {A,B}
L'ensemble d'événements est { a = {A} , b = {B} , c = {A , B} }
L'automate $CAB de ce contexte est défini par

Q($CAB) = { s }
Ω(s) = { {{ a }}, {{ b }}, {{ c }},
{{ a,b }}, {{ a,c }}, {{ b,c }}, {{ a,b,c }} }
T($CAB) = { (s , a , s) , (s , b , s) , (s , c , s) }
i($CAB) = s

Il est facile de vérifier, en utilisant la propriété du § 4.3 que tout processus défini sur un ensemble de connecteurs K est moins discriminant que le contexte défini sur ce principe pour le même ensemble de connecteurs (l'ensemble des traces est nécessairement inclus dans l'ensemble des traces testées et la condition sur les tests de possibilités de communication est nécessairement réalisée par parties).
Fin-exemple

Pour un ensemble d'événements Ev, le contexte CEv décrivant ces relations est défini par l'automate $CEv :

$Q(\$CEv) = \{ s \}$

$\Omega(s) = \{ X \mid \exists Y \in \mathbf{P}(Ev) - \emptyset , X = \{ Y \} \}$

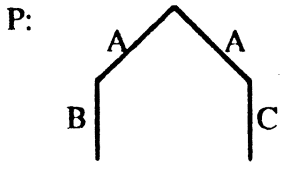
$T(\$CEv) = \{ (s , e , s) \mid e \in Ev \}$

4.4.2 Contexte "utilisateur de machines automatiques"

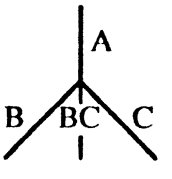
Le deuxième contexte que nous décrirons définit des relations qui ont été étudiées en tant que telles dans [Rog 85]. Ces relations répondent au problème suivant:

On considère , comme cela est fait par de nombreux auteurs [Hoa 85][Mil 80] pour introduire la notion de processus communicant, que les processus sont des machines automatiques avec lesquelles on communique en appuyant sur des boutons.

A partir de FP2, les connecteurs peuvent être considérés comme les boutons et les communications comme le fait d'appuyer sur les boutons. Un problème particulier est posé par l'observabilité des événements multiples.
Considérons un processus P non-déterministe



Après A, pour être sûr de pouvoir communiquer avec le processus, il faut être prêt à réaliser les événements B et C et par conséquent, l'utilisateur doit appuyer sur les deux boutons B et C.
Face au processus suivant, si l'environnement appuie sur B et C, les événements B et C peuvent donc être réalisés séparément comme dans le cas précédent.



De ce fait il faut admettre en particulier l'équivalence des processus suivants

La caractéristique de ce contexte est donc essentiellement de rendre éventuels à partir d'un état tous les événements multiples qui sont réalisables en partie à partir de cet état.

L'automate qui décrit ce contexte MEv est défini à partir du même langage sur les traces que le précédent et l'état de l'automate est étiqueté par un sous-ensemble du précédent: chaque test contenant un événement multiple doit contenir sur le même état tous les événements qui en sont une partie non vide.

$Q(\$M_{Ev}) = \{ s \}$

$\Omega(s) = \{ Y \mid \exists E \in H(EV), Y = \{ E \} \}$

où $H(EV) = \{ E \in \mathbf{P}(EV) - \{\emptyset\} \mid \forall X \in E, \emptyset \subset X' \subseteq X \Rightarrow X' \in E \}$

$T(\$M_{Ev}) = \{ (s, e, s) \mid e \in Ev \}$

$i(\$M_{Ev}) = s$

Pour l'ensemble des processus définis sur l'ensemble de connecteurs { A , B }, ce contexte est défini par:

$Q(\$M_{AB}) = \{ s \}$

$\Omega(s) = \{ \{\{ a \}\}, \{\{ b \}\}, \{\{ a,b,c \}\} \}$

$T(\$M_{AB}) = \{ (s, a, s), (s, b, s), (s, c, s) \}$

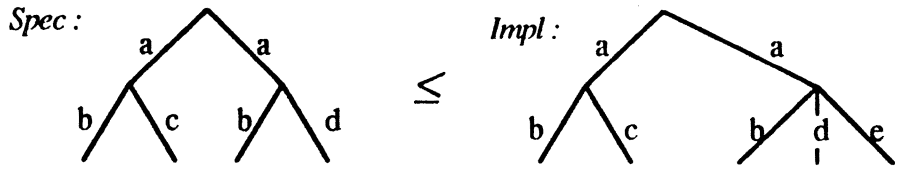
$i(\$M_{AB}) = s$

4.4.3 Contextes liés à la définition d'un processus.

On peut donner, à l'aide de la notion de contexte une interprétation particulière d'un processus donné comme spécification. Dans ce cas, le but est de fournir une méthode générale de construction de contexte propre à chaque type d'interprétation. Nous présentons dans cette section un exemple de méthode générale de construction de contexte.

Les relations de comparaison que nous proposons dans ce paragraphe sont fondées sur l'idée qu'un processus *Spec* donné comme spécification définit un "mode d'emploi" et par conséquent qu'un processus *Impl* peut en être considéré comme une implémentation correcte dans la mesure où l'environnement reste dans le cadre de ce "mode d'emploi".

Exemple



Le processus *Impl* peut réaliser l'événement e que ne peut pas réaliser le processus *Spec*. Toutefois, si un utilisateur essaie de réaliser e, il sort du mode d'emploi défini par *Spec* et le comportement n'est pas garanti.

Fin-exemple

Ces relations seront donc définies par une fonction *ME* qui construit l'automate du contexte à partir de l'automate de comportement du processus spécification.

Q($ME(spec)) = Q(@spec)

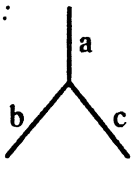
i($ME(spec))= i(@spec)

T ($ME(spec))= T(@spec)

$\Omega(q) = \{ \{ E \} \mid E \in \Delta(q) \}$

Exemple

$ME(Spec) :



{{{ a }}}

{{{ b }} ,{{ b,c }} ,{{ b,d }} ,{{ b,c,d }} ,{{ c,d }}}

Fin-exemple

4.5 Comparaison avec d'autres travaux

Dans cette section, nous rapprocherons la notion de contexte que nous avons définie de celles qui ont été définies à partir d'autres modèles et que nous avons présentées dans les chapitres précédents.

4.5.1 Comparaison avec les travaux de Larsen

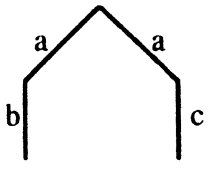
Les objectifs informellement assignés à l'étude présentée dans [Lar 85] d'une part et à celle que nous avons exposée d'autre part sont sensiblement identiques: il s'agit de prendre en compte une définition partielle de réseau pour comparer des processus qui doivent y être insérés.

Les différences essentielles dans la façon de traiter ce problème se situent à deux niveaux: d'une part la définition de la notion de contexte, d'autre part le principe fondamental de comparaison qui est dans le cas de [Lar 85] la bisimulation tandis que nos propositions reposent sur une notion globale de comportement des processus.

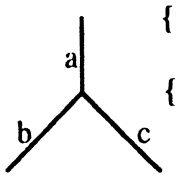
Nous avons utilisé une structure plus complexe pour définir les contextes, les tests définis par un processus se situant à un niveau interne dans la structure du test effectué par le contexte. Au contraire, la structure définie dans [Lar 85] est identique à celle des processus (systèmes de transitions) et les deux niveaux sont identifiés, ce qui signifie en fait que l'un des deux n'existe pas.

Plus précisément, la paramétrisation proposée dans [Lar 85] ne permet pas de rendre compte uniquement des contraintes d'un élément de réseau déjà défini, c'est à dire de la notion d'équivalence de réseaux fermés.

Considérons par exemple, l'environnement présenté au chapitre deux décrit par l'arbre :

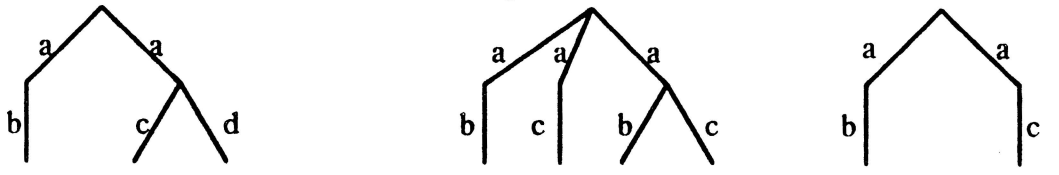


En tant que contexte, il serait représenté par l'automate suivant:

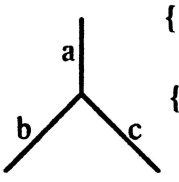


--deux tests à un état

et on obtiendrait l'équivalence des trois processus ci-dessous.



En tant que processus-contexte, il serait représenté par l'automate suivant

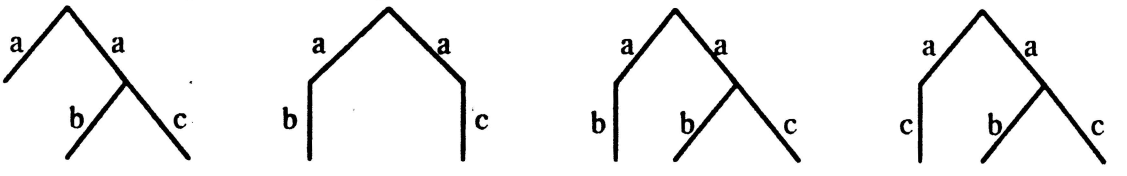


{{{ a }}}

{{{ b } , { c }}}

--un test à deux états

et on aurait l'équivalence des processus suivants,



ce qui est exclu par les relations proposées dans [Lar 85].

D'autre part, la paramétrisation proposée dans [Lar 85] est fondée sur la bisimulation des systèmes de transition tandis que celle que nous avons proposée repose sur une vision globale du comportement des processus. Par conséquent, alors que les relations que nous proposons reposent sur un double critère égalité des ensembles de traces et deadlock dans un réseau fermé, celles qui sont obtenues par bisimulation paramétrée exigent que les réseaux fermés aient des comportements "beaucoup plus voisins". Il faut toutefois noter que les relations proposées dans [Lar 85] ne reposent pas sur une définition de l'équivalence de réseaux fermés.

4.5.2 Comparaison avec les "critères d'observation"

L'apport essentiel de la notion de critère d'observation est de proposer un nouveau problème: celui de l'équivalence de séquences d'événements (voir la discussion du chapitre deux sur la difficulté à définir un critère d'observation qui autorise l'équivalence de deux séquences d'événements <a,b> et <b,a> tout en distinguant les autres événements).

Tels que nous les avons définis , les contextes ne permettent pas de prendre en compte de tels paramètres de compraison. On pourrait envisager de modifier la notion d'action utilisée pour définir l'automate du contexte en utilisant la notion d'observable définie dans [Bou 84] (action=ensemble de séquences d'événements).

Toutefois, un problème essentiel reste posé: celui de la possibilité de réaliser une séquence d'événements. En effet, lorsqu'on limite les actions à des événements, leur réalisation est indivisible et par conséquent, possible ou impossible à partir d'un état. Au contraire, lorsque les actions peuvent être des séquences d'événements, elles deviennent possibles, impossibles ou commencées et non terminées.

4.5.3 Equivalences par tests

Les relations obtenues par cette théorie sont définies à partir d'un ensemble d'observateurs qui peuvent atteindre des états de succès après une suite de communications avec les processus. Les seules relations qui ont été étudiées sont celles qui sont définies à partir de l'ensemble d'observateurs le plus général.

Le problème de la comparaison de contextes définis par des ensembles d'observateurs O1 et O2 n'a pas été étudié. Toutefois, on peut essayer de situer l'ensemble des relations définissables par la notion de contexte dans l'ensemble des relations définissables par la notion d'ensemble d'observateurs.

Les notions d'observateur et de processus-contexte semblent voisines mais celle d'observateur est strictement plus puissante lorsqu'on limite la comparaison au domaine des processus non divergents. En effet, toute relation exprimée en terme d'ensemble de processus-contexte peut être exprimée par un ensemble d'observateurs, l'inverse étant faux. Les observateurs peuvent être considérés comme des entités capables de détecter des résultats identiques indépendamment de la façon dont ils sont calculés alors que les processus-contextes ne permettent que de comparer des façons de calculer identiques.

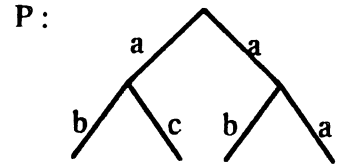
A tout processus-contexte, on peut donc associer un ensemble d'observateurs de telle manière qu'un observateur soit associé à une trace t et à un ensemble de tests sur les possibilités de communication:

Soit t une trace d'un processus P et S l'ensemble des états accessibles par t dans P. On définit un observateur $o(t , S)$ à partir des observateurs not définis dans [Bro 83]:

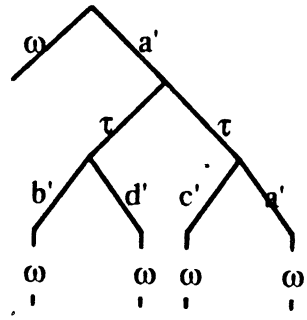
$$o(<>, S) = \Sigma_{s \in S} \tau \Sigma_{x \in evts(s , P)} x'.\omega.nil$$

$$o(a . t, S) = \omega.nil + a'. o(t, S)$$

Exemple



o (< a > , { P1 , P2 })



Fin-exemple

Le concept d'observateur est strictement plus puissant que celui de processus-contexte comme nous l'avons vu par rapport à la possibilité de prendre en compte la notion de séquence d'événements.

Il faut cependant rappeler les problèmes posés par la notion d'observateur pour prendre en compte le problème de la divergence. Une étude intéressante devrait pouvoir être menée en utilisant d'une part les fonctions de rigidification définies au troisième chapitre et d'autre part, la notion de contexte définie par un ensemble d'observateurs.

Conclusion

Pour résumer les travaux présentés dans cette thèse, nous dirons qu'ils sont fondés sur un formalisme de description des processus communicants particulier –le langage FP2– que l'approche adoptée pour traiter le problème de la comparaison observationnelle se situe parmi celles qui ont une vision globale du comportement des processus et a permis de traiter certains problèmes de façon originale. Nous avons également étudié l'influence d'un modèle synchrone, les propriétés algébriques de l'ensemble des processus et abordé la notion de contexte d'utilisation.

FP2 , contrairement à la plupart des formalismes algébriques qui supportent les études menées sur le thème de la comparaison observationnelle des processus (CCS , Meije , AT , TCSP) distingue expression de comportement et processus. Les processus, systèmes de transitions d'états, sont décrits par des ensembles de règles de transition et les opérateurs entre processus définissent une algèbre sur le domaine des systèmes de transition.

Le langage FP2 nous a donc permis d'aborder le problème de la comparaison observationnelle avec une notion précise de processus: entité autonome de calcul et de communication. De ce fait certains problèmes dûs à des opérateurs de description des comportements ont pu être évités.

Par ailleurs, les processus étant décrits par des systèmes de transition nous avons défini des algorithmes et des fonctions (rigidification , caractérisation du non déterminisme) sans chercher d'axiomatisation sur les comportements.

La notion de comportement de communication des processus telle que nous l'avons utilisée se situe parmi celles qui ont une approche globale du comportement des processus. Nous avons montré son équivalence avec celle de Hennessy d'une part et de Brookes d'autre part dans le cas des processus non divergents ayant des ensembles de traces identiques. Sur ces domaines, on peut la considérer comme une troisième formalisation aboutissant aux mêmes conclusions.

Néanmoins, l'approche que nous avons développée traite de façon originale le non déterminisme en interprétant d'abord le non déterminisme dû aux transitions internes. Ceci nous a permis d'isoler le problème de la divergence des processus en modèle asynchrone et d'en proposer une interprétation plus fine que celles de [HN 84] ou [BHR 84].

En effet, le modèle des équivalences par tests permet de comparer des processus en fonction de leurs ensembles de traces au delà des points de divergence mais, comme nous l'avons montré au cours du deuxième chapitre, il ne permet pas de comparer le non déterminisme des processus au delà de ces points. Le modèle TCSP réduit la divergence au processus CHAOS, qui peut accepter ou refuser tout événement. La divergence y est donc traitée comme une possibilité de deadlock et de plus il est impossible de comparer les processus au delà des points de divergence, que ce soit par leurs ensembles de traces ou par leurs possibilités de communication.

En traitant d'abord le non déterminisme des transitions internes, nous avons isolé le problème de la divergence et proposé deux interprétations de la divergence (équitable ou non). Ceci nous a permis de ne pas nécessairement assimiler divergence et deadlock. De plus, la divergence ayant été interprétée équitablement ou non, le comportement des processus est alors exprimé uniquement par des événements de communication avec l'environnement et il est possible de comparer les processus au delà des points de divergence en fonction de leurs ensembles de traces mais aussi en fonction de leurs possibilités de communication.

FP2, par la notion d'événement qu'il utilise (ensemble indivisible de communications) permet la définition d'opérateurs synchrones. En distinguant, dans l'étude fondée sur un modèle asynchrone non déterminisme dû aux transitions internes et non déterminisme dû à l'accessibilité de plusieurs états par un même événement, nous avons pu montrer que l'influence des opérateurs synchrones se limitait à rendre déterministe l'occurence des transitions internes.

L'étude menée pour définir le comportement observable des processus est entièrement centrée sur l'influence du non déterminisme et n'a pas pour objet de définir des critères de comparaison a priori sur les ensembles de traces des processus. Ces relations sont donc définies uniquement entre des processus ayant des ensembles de traces identiques.

Pour aborder le problème des processus ayant des ensembles de traces différents, nous avons choisi de définir une notion de contexte utilisateur des processus. Cette notion a pour objet de prendre en compte un contexte partiellement défini et par conséquent d'ignorer dans les comportements potentiellement différents des processus tous ceux qui sont effectivement ignorés par les utilisateurs des processus.

Cette notion nous a permis en particulier de montrer que les relations caractéristiques du comportement des processus étaient entièrement définies par le fait qu'un processus quelconque, utilisateur de deux processus équivalents, doit pouvoir avoir avec ces processus les mêmes séquences d'événements et les mêmes situations de deadlock.

De plus, nous avons montré sur plusieurs exemples comment on pouvait définir des contextes intéressants, liés ou non à l'un des processus.

Le cadre dans lequel se situe cette étude permet d'envisager des extensions et des applications de ces résultats.

Tout d'abord, au niveau des extensions, il faut remarquer que nous avons restreint le domaine des processus définissables en FP2 de manière draconienne: processus d'états finis n'échangeant que des valeurs de type signal. Ceci nous a permis de définir des relations qui sont calculables alors qu'elles ne le seraient pas dans le cas général.

Le problème de la divergence en particulier , est ici facile à cerner alors qu'il est très complexe dans le cas général. Le problème posé est alors celui de l'arrêt d'un système de réécriture et dans [Per 84], JM Pereira a cherché à définir le domaine des processus sur lequel l'arrêt d'une règle serait décidable.

On peut néanmoins envisager d'étendre les résultats qui ont été présentés ici à des processus qui présenterait une certaine régularité dans leurs comportements.

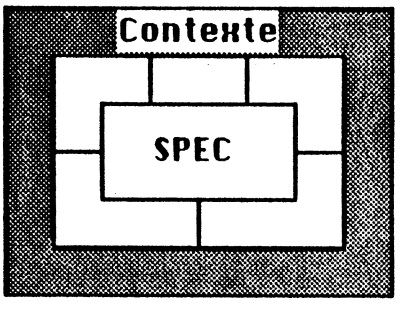
Déjà, on peut noter que ces résultats peuvent être appliqués à des processus qui n'échangeraient entre eux que des messages appartenant à des types finis: il suffit d'associer un connecteur à un couple connecteur-valeur et un état à un couple symbole d'état valeur de paramètre.

Une application intéressante à laquelle pourraient conduire ces résultats est la synthèse de processus. Dans [Mar.A 86], ce problème est abordé de façon à pouvoir décomposer un processus ou un réseau FP2, sous sa forme la plus générale (pas uniquement un processus d'états finis) en un réseau de processus devant s'exécuter sur un réseau de processeurs ayant une configuration particulière (maillage régulier du type hypercube).

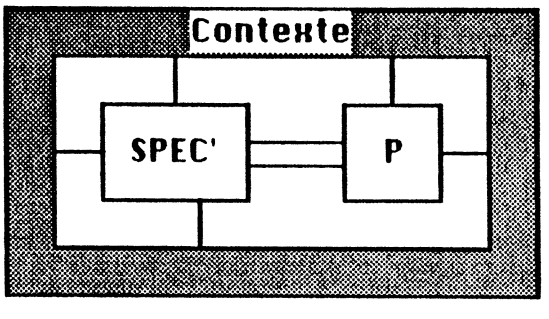
Le problème de la synthèse pourrait être défini autrement ainsi: étant donnés d'une part une spécification donnée par un processus FP2 et un contexte d'utilisation et d'autre part un ensemble de processus P1 ... Pn, comment construire à partir de l'ensemble des processus Pi une implémentation de la spécification qui soit satisfaisante dans le contexte de la spécification.

Grâce à la notion de contexte d'utilisation, on peut envisager une formulation récursive de ce problème:

Soit une spécification

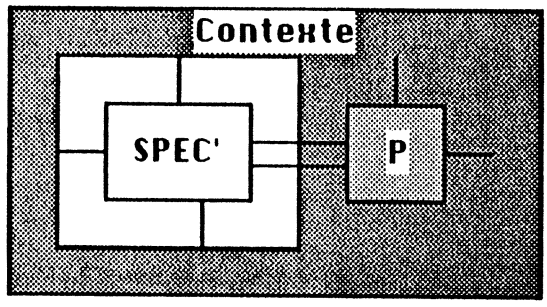


Supposons que le processus P soit l'un des processus de l'implémentation. Dans ce cas, l'implémentation se décompose de la façon suivante: IMPL = P || SPEC' + cnx - abs



qu'on peut considérer comme le problème précédent si on l'envisage sous la forme

qu'on peut considérer comme le problème précédent si on l'envisage sous la forme



c'est à dire si l'on peut définir le contexte de SPEC' par le contexte de SPEC précisé par le processus P.

Pour pouvoir aborder le problème de cette façon deux problèmes devraient être résolus: le premier est de pouvoir définir un nouveau contexte à partir d'un contexte et d'un processus. Puisqu'on a initialement défini un contexte comme un ensemble fini de processus, on devrait pouvoir dériver des opérateurs définis entre les processus des opérateurs entre les contextes.

Le deuxième problème est plus ardu et consiste à calculer SPEC' à partir du contexte et des processus SPEC et P.

Un premier élément de réponse peut être apporté en suivant le comportement des processus à chaque pas. En effet, on peut envisager de définir des algorithmes du type de ceux qui ont été utilisés dans cette thèse pour répondre à des questions du type: le processus P peut-il participer à un réseau qui implémente correctement la spécification, c'est à dire décider de l'existence possible de SPEC' sans calculer SPEC'.

La difficulté est alors de définir les processus" internes" de l'implémentation, c'est à dire le sous-réseau qui ne contient pas de connecteurs de la spécification (le processus FONCTION du réseau présenté au cours du premier chapitre, par exemple). En effet, du fait des abstractions et de l'anonymat des événements de communication internes d'un réseau de processus, les solutions peuvent être nombreuses. On peut penser que la caractérisation des éléments manquant pour réaliser un réseau qui soit une implémentation correcte devrait faire appel à l'énoncé de propriétés nécessaires plus qu'à la définition d'ensembles de processus SPEC'. En ce sens des approches unificatrices des différents formalismes d'analyse et de comparaison des processus telles que celles développées récemment [GS 86] devraient se révéler utiles.

REFERENCES

[Ark 84] E. ARKAXHIU
"Un environnement et un langage graphique pour la spécification de processus parallèles communicants".
Thèse 3° cycle LIFIA INPG 1984

[AS 83] G.R.ANDREWS, F.B. SCHNEIDER
"Concepts and notations for concurrent programming"
Computing Surveys Vol 15 No 1 Mars 83

[Aus 83] D. AUSTRY
"Aspects syntaxiques de Meije,un calcul pour le parallélisme"
Thèse 3° cycle ParisVII 1983

[Bar 83] J.BARRON, P.CAULL , D.MAY , P WILSEN
"Transputer does 5 or more MIPS even when not used in parallel"
Electronics Nov 1983

[Ber 79] D.BERT
"La programmation générique: construction de logiciel, spécification algébrique et vérification"
Thèse d'état, USMG 1979

[Ber 83] G. BERRY, S.MOISAN, JP RIGAULT
"ESTEREL: Towards a synchronous and semantically sound high level language for real time applications"
Proc of IEEE real time symposium 1983

[BHR 84] S.BROOKES, C.A.R HOARE, A.W ROSCOE
"A theory of communicating sequential processes"
J ACM Juillet 1984

[BK 84] J.A BERGSTRA , J.W KLOP
" A complete inference system for regular processes with silent moves"
Center for mathematics and computer science Amsterdam
Report CSR 8420 Nov 1984

[Bou 84] G. BOUDOL, D. AUSTRY
" Algèbre de processus et synchronisation"
TCS 30 - 1984

[Bou 85] G. BOUDOL
"Le calcul Meije"
dans " Parallélisme, communication et synchronisation" Edité par JP Verjus et G. Roucairol(Edition du CNRS) pp 23-45 1985

[Bro 83] S.BROOKES
"A model for communicating sequential processes"
Thèse Ph.D Oxford 1983

[Cis 84] M. CISNEROS
"Programmation parallèle et programmation fonctionnelle: propositions pour un langage"
Thèse 3° cycle LIFIA INPG 1984

[Dar 82] Ph. DARONDEAU
"An elarged definition and complete axiomatisation of observational congruence of finite processes"
LNCS 137 1982

[Dij 75] E.W DIJKSTRA
"Guarded commands, nondeterminacy and a calculus for the derivation of programs"
C.ACM Aout 1975

[DK 85] Ph. DARONDEAU, L. KOTT
" Equivalence observationnelle du parallélisme équitable"
dans " Parallélisme, communication et synchronisation" Edité par JP Verjus et G. Roucairol (Edition du CNRS) pp 47-63 1985

[Ech 86] R.ECHAHED, D. BERT
"Design and implementation of a generic, logic and functional programming language"
RRIMAG 560 LIFIA 32 Novembre 85 - Paru dans LNCS 213 pp 119-132

[SG 86] S.GRAF , J.SIFAKIS
"A Logic for the Description of Non-deterministic Programs and Their Properties"
Inf.control 68 -1986

[Hen 83] M. HENNESSY
" synchronous and asynchronous experiments on processes"
Inf. control 59 -1983

[Hen 85] M. HENNESSY
"Acceptance trees"
J.ACM vol 32 -Oct 1985

[Hoa 78] C.A.R HOARE
"Communicating Sequential Processes"
C.ACM Aout 1978

[Hoa 81] C.A.R HOARE
"A calculus of total correctness for communicating processes"
PRG 23 Oxford 1981

[Hoa 85] C.A.R HOARE
"Communicating sequential processes"
Prentice-Hall International 1985

[HN 84] M. HENNESSY, R. De NICOLA
"Testing Equivalences for processes"
TCS 34 1984

[HM 80] M. HENNESSY, R. MILNER
"On observing nondeterminism and concurrency"
Proc. ICALP 1980 LNCS 85

[Huf 86] JM. HUFFLEN
"un exemple d'utilisation de FP2: Description d'une architecture parallèle et pipe-line pour l'unification"
RR IMAG 576 LIFIA 43

[Jor 82] Ph. JORRAND
"Specification of communicating processes and process implementation correctness"
LNCS 137 1982

[Jor 85] Ph. JORRAND, JM. HUFFLEN, A.MARTY, JC.MARTY, Ph . SHNOEBELEN
"FP2 : The langage and its formal definition"
RR LIFIA 26 IMAG 537 Mai 85

[Jor 86] Ph. JORRAND
"Term rewriting as a basis for the design of a functional and parallel langage:
A case study : The langage FP2"
RR IMAG 576

[Ken 80] JR. KENNAWAY
" A theory of nondeterminism"
Springer LNCS vol 85 1980

[Lar 85] K.G LARSEN
"A context dependent equivalence between processes"
Automata, Languages and programming 12th colloquium 1985 LNCS 194

[Mar.A 86] A.MARTY
" Placement d'un réseau de processus communicants décrit en FP2 sur une structure de grille en vue d'une implémentation parallèle de ce langage"
RR IMAG 606 LIFIA 49 Mai 1986

[Mar.JC 86] JC. MARTY
"Un environnement de programmation pour le langage FP2"
A paraître

[May 83] D. MAY
"OCCAM"
SIGPLAN notices Vol 18 Avril 1983

[Mil 80] R. MILNER
"A calculus of Communicating Systems"
LNCS 92 1980

[Mil 83] R. MILNER
"Calculi for synchrony and asynchrony"
TCS 25 1983

[MV 86] E.MADELAINE, D. VERGAMINI
"système Ecrin manuel d'utilisation"
présenté aux journées pôle sémantique de C3 SMH avril 86

[Naj 85] E.NAJM
"SCAN un outil de modélisation et de vérification des systèmes distribués"
Rapport Rhin/FDT 7577 Février 1985

[Old 85] E.R . OLDEROG
"Specification oriented programming in TCSP"
Advanced Seminar 'logic and models for verification and specification of concurrent systems', La Colle sur Loup, 1984

[Pan 86] X.PANDOLFI
"Etude de l'inference de types en FP2, langage fonctionnel et parallèle fondé sur la réécriture de termes"
Rapport interne LIFIA

[Par 81] D. PARK
" Concurrency and automata on infinite sequences""
Proc of 5th GI conference LNCS 104 1981

[Per 84] JM.PEREIRA
"Processus communicants: un langage formel et ses modèles. Problèmes d'analyse"
Thèse 3° cycle LIFIA -INPG Juin1984

[Plo 81] PLOTKIN
" A structured approach to operational semantics"
DAIMI FN 19 Computer Science Department Aarhus University 1981

[Rog 85a] S. ROGÉ
"Systèmes PARallèles Communicants: Etude du non déterminisme,
comparaison des comportements de processus"
RR IMAG 541, LIFIA27 , Juin 85

[Rog 85b] S. ROGÉ
"Comparaison de processus dans des modèles synchrones et asynchrones"
RR IMAG 572, LIFIA40 , Novembre 85

[Rog 85c] S. ROGÉ
"Propriétés algébriques de l'ensemble des processus communicants réguliers"
RR IMAG 569, LIFIA38 , Décembre 85

[Sch 85] Ph . SHNOEBELEN
"Sémantique du parallélisme en FP2"
Rapport interne LIFIA

[Sch 86a] Ph . SHNOEBELEN
"μFP2: a prototype interpreter for FP2"
RR IMAG 573 LIFIA 41 Janvier 1986

[Sch 86b] Ph . SHNOEBELEN
"About the implementation of FP2"
RR IMAG 579 LIFIA 42 Janvier 1986

[Sor 87] A.SORIANO
"VENUS: Un outil d'aide à la vérification des systèmes communicants"
Thèse 3° cycle LGI-INPG 1987 A paraître

INPG département des études doctorales

AUTORISATION de SOUTENANCE

VU les dispositions de l'article 15 Titre III de l'arrêté du 5 juillet 1984 relatif aux études doctorales

VU les rapports de présentation de Messieurs

. J. SIFAKIS, Directeur de recherche
. M. SINTZOFF, Professeur

Mademoiselle Sylvie ROGE

est autorisée à présenter une thèse en soutenance en vue de l'obtention du diplôme de DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE, spécialité "Informatique".

Fait à Grenoble, le 17 novembre 1986

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

P.O. le Vice-Président,

RESUME

Le problème étudié est celui de la comparaison des comportements des processus communicants, à partir d'un langage de programmation parallèle: FP2 (Functional Parallel Programming) sur le domaine restreint des processus réguliers.

Dans un premier temps, nous présentons des modèles permettant de décrire des systèmes de processus communicants, synchronisés par rendez-vous, ainsi que les différentes théories qui traitent les problèmes de la comparaison observationnelle.

Nous abordons ensuite le problème à partir du langage FP2. Nous proposons une démarche qui permet de faire totalement abstraction des événements internes des processus et d'exprimer le comportement de communication des processus en n'utilisant que les événements de communication avec l'environnement. Cette approche nous permet, en particulier, de proposer une interprétation plus fine de la divergence que celles rencontrées dans la littérature.

Enfin, une notion de contexte est définie et étudiée qui permet d'envisager la comparaison des processus dans un environnement dont le comportement est partiellement connu.

MOTS - CLES

parallélisme, processus communicants,
synchronisation par rendez-vous, divergence,
comparaison observationnelle, contexte utilisateur