THESE

présentée à

l'Université de la Méditerranée - Aix-Marseille II

Ecole Doctorale de Mathématiques et Informatique

par

Cristian Ene

pour obtenir le grade de **DOCTEUR**

DE L'UNIVERSITÉ D'AIX-MARSEILLE II

spécialité : **INFORMATIQUE**

Un modèle formel pour les systèmes mobiles à diffusion

Date de soutenance : 3 décembre 2001

Composition du jury :

- Yassine LAKHNECH, Prof., Université Joseph Fourier, Grenoble, Rapporteur
- Jean-Jacques LEVY, Dir. de Recherche, INRIA Rocquencourt
- Jean-François MAURRAS, Prof., Université d'Aix-Marseille II
- Traian MUNTEAN, Prof., Université d'Aix-Marseille II, Directeur de thèse
- Laurence PIERRE, MdC-HdR, Université d'Aix-Marseille I
- K.V.S. PRASAD, Prof., Chalmers University of Technology, Gothenburg, Rapporteur

à Iuliana,

à mes parents,

à Daniela.

- Remerciements -

Tout d'abord, je remercie chaleureusement Traian Muntean, mon directeur de thèse, qui m'a accueilli dans son groupe, et m'a fourni un bon cadre de travail. Les precieux conseils qu'il m'a donnés régulièrement et l'examen de mes travaux qu'il a effectué tout au long de ces années ont permis à cette thèse de voir le jour.

Je souhaite ensuite remercier les personnes qui ont accepté de participer à mon jury de thèse : Yassine Lakhnech, Jean-Jacques Levy, Jean-François Maurras, Laurence Pierre et K.V.S. Prasad. Tout particulièrement, mes remerciements vont aux rapporteurs, K.V.S. Prasad pour ses commentaires détaillés et sa grande disponibilité, et Yassine Lakhnech, pour ses remarques qui m'ont permis d'améliorer le manuscrit.

Je tiens à remercier les membres de l'équipe Systèmes Parallèles et Communicants du LIM: Marc Gengler, Tin N'Guyen, Arnaud Février pour les remarques faites sur le manuscrit, et Léon Mugwaneza pour le temps qu'il m'a consacré dans ses lectures "très critiques" sur mes travaux. L'immense aide scientifique et moral que Léon m'a apportée tout au long de ces années, en commençant par mes premiers mois de DEA en France, sont pour beaucoup dans cette thèse.

Un grand merci aux anciens et actuels T.H.E. pour leur accueil chaleureux à Luminy et pour leur amitié: Bich, Billl, Bruno, Cédric, Hervé, Ludmila, Marjorie, Matthieu, Nicolas, Pedro, Stéphane et Yann. Leur présence a grandement contribué à rendre le travail agréable. Je remercie tout particulièrement Matthieu Martel pour ses commentaires détaillés sur le manuscrit.

J'adresse aussi de très vifs remerciements à Ferucio Laurenţiu Ţiplea, qui a su me donner le goût de faire de la recherche. Sa rigueur scientifique restera pour moi un exemple à suivre.

Finalement, je remercie ma femme, Iuliana, pour son précieux soutien moral tout au long de ces années de thèse. Sa patience et sa disponibilité m'ont aidé à dépasser les moments difficiles qui ont pu apparaître pendant ces quatre derniers années.

Table des matières

1 Introduction **3**

1.1 Construction correcte des systèmes parallèles/distribués 3

1.2 Contribution et plan de cette thèse 7

2 Une hiérarchie de modèles pour les systèmes distribués **9**

2.1 Modèles statiques pour les systèmes concurrents 10

2.1.1 Calculus of Concurrent Systems (CCS) 10

2.1.2 Communicating Sequential Processes (CSP) 13

2.1.3 I/O automata 14

2.1.4 Calculus of Broadcasting Systems (CBS) 15

2.2 Modèles dynamiques pour les systèmes concurrents 17

2.2.1 Systèmes d'acteurs 17

2.2.2 Linda 18

2.2.3 π - calcul 19

2.2.4 Higher Calculus of Broadcasting Systems ($HOBS$) 21

2.3 Pourquoi un autre modèle pour la mobilité? 21

3 Le $b\pi$-calcul **25**

3.1 Syntaxe et Sémantique 25

3.2 Exemples 34

3.3 Conclusion 42

4 Bisimulations **43**

4.1 Équivalences 43

4.1.1 Les équivalences barbelées 44

4.1.2 Équivalences transitionnelles 47

4.1.3 Les bisimulations étiquetées 50

4.1.4 Application 61

4.2 Congruences 64

4.2.1 Définitions 64
4.3 Axiomatisation de la congruence forte 70
4.3.1 Caractérisation de la congruence forte pour les processus simples . 70
4.3.2 L'ajout de l'opérateur de restriction 77
4.3.3 L'ajout du parallélisme 77
4.4 Conclusion 79

5 Caractérisations induites par des tests 81
5.1 Les préordres induits par des tests 83
5.2 Caractérisation pour le CBS sans passage de valeurs 84
5.2.1 Présentation du CBS sans passage de valeurs 84
5.2.2 Une caractérisation de "may testing" basée sur des traces 86
5.2.3 Une caractérisation de "must testing" basée sur des traces 90
5.3 Caractérisations pour le $b\pi$-calcul 95
5.3.1 Une caractérisation de "may testing" basée sur des traces 95
5.3.2 Une caractérisation de "must testing" basée sur des traces 99
5.4 Conclusion 101

6 Expressivité du $b\pi$-calcul 103
6.1 Des résultats de séparation entre des langages point à point et des langages à diffusion 104
6.1.1 Système symétrique et système électoral 104
6.1.2 Codage uniforme 106
6.1.3 Résultats de séparation 111
6.2 Résultats des minimalités dans le $b\pi$-calcul 116
6.2.1 Non - encodage de $+$ 117
6.2.2 Non - encodage de $\langle x = y \rangle$ 118
6.3 Conclusion 120

7 Conclusion 121

A Preuves rélatives au Chapitre 3 131

B Preuves rélatives aux bisimulations 137

C La correction d'un "FIFO protocole" 141

D Preuves rélatives au congruences 149

Chapitre 1

Introduction

1.1 Construction correcte des systèmes parallèles/distribués

Dû à la complexité croissante des architectures et des logiciels, la programmation des machines devient de plus en plus un veritable art. Les premiers ordinateurs étaient des machines mono - processeurs, et les premières applications étaient relativement simples. De nos jours, les systèmes matériels et logiciels ont beaucoup évolué : les machines multi - processeurs et les réseaux de grand taille (l'Internet étant l'exemple le plus éloquent) ont changé la "philosophie" de l'informatique. Les logiciels ne sont plus des applications "fermées", s'exécutant sur une machine isolée (et dont la sémantique était une fonction dépendant des entrées), mais ils deviennent des "acteurs" qui doivent interagir et collaborer avec d'autres applications s'exécutant de manière distribuée.

Cette complexité ajoute plusieurs ordres de magnitude à la "subtilité" des erreurs: l'explosion de la fusée Ariane (Ariane, 1996), et l'instruction de division erronée du processeur Intel Pentium étant parmi les exemples les plus récents et plus coûteux. En plus des dépenses, de telles erreurs peuvent entraîner des conséquences plus graves dans les programmes gérant des moyens de transport (trains, avions, etc.), ou des centrales nucléaires. L'analyse des programmes perd son caractère intuitif (et sa dimension humaine), et nécessite l'introduction des *méthodes formelles automatisables.* La spécification (des architectures matérielles, des primitives des systèmes d'exploitation, des constructions des langages de programmation) doit être basée sur des principes mathématiques, par l'utilisation d'un langage bien défini syntaxiquement et sémantiquement.

Un modèle de base pour les systèmes de calcul, doit être "controlable", basé sur quelques concepts, et surtout doit bénéficier d'une théorie relativement simple, pour faciliter sa compréhension et son application à la description et à l'analyse formelle des systèmes.

Pour les applications séquentielles, où l'aspect calculatoire est essentiel, les machines de Turing et le λ-calcul sont des modèles "canoniques" : beaucoup d'autres modèles mathématiques classiques, tel que la RAM (machines à accès aléatoire), les fonctions récursives, les structures de contrôle (if then else, goto, while) ont le même pouvoir d'expression. Le λ-calcul peut être considéré comme le noyau de tout langage de programmation séquentiel: les structures de contrôle et les structures de données peuvent y être encodées. Depuis les années '60, le λ-calcul a eu un rôle important dans la conception et l'implantation des langages, et dans le développement de la théorie des types.

Le λ-calcul (comme les autres modèles déjà énumérés) décrit des comportements non-interactifs: pour une entrée donnée, il rend un résultat. Les *systèmes interactifs* (Wegner, 1997) ou *réactifs* (Berry et al., 1993) ont un comportement dépendant de leur environnement et doivent réagir en permanence aux "stimuli" externes: par exemple, les logiciels embarqués pour l'aéronautique, la médecine, les composants électroniques ou les applications dans les télécommunications. Les applications interactives sont souvent distribuées, étant conçues comme un ensemble de plusieurs composants qui sont à leurs tour des applications interactives.

Contrairement aux modèles calculatoires, pour la spécification des systèmes distribués il n'y a pas de modèle canonique. Une des premières théories des systèmes distribués peut être considérée comme étant la théorie des réseaux de Petri ((Petri, 1973), (Reisig, 1985), (Best et Fernández, 1988), (Tiplea et Ene, 1997)). Le modèle est une généralisation de la théorie des automates, où des actions (transitions) peuvent être faites en parallèle, en fonction des diverses conditions (qui permettent la concurrence des actions, ou au contraire, assurent une exclusion mutuelle). Si la concurrence était bien représentée dans les réseaux de Petri, il manquait l'aspect compositionel des applications distribuées: il était difficile de représenter la manière dont deux réseaux différents interagissent entre eux (la compositionalité des réseaux de Petri fait l'objet de plusieurs travaux récents, par exemple (Best et al., 2001)) .

Ces considérations ont conduit au développement des *calculs de processus*, dont les plus connus sont CCS ((Milner, 1980), (Milner, 1989)) et CSP ((Hoare, 1978),(Hoare, 1985)). Depuis les années '80, une multitude d'extensions et de variantes de ces langages ont été proposées. Certains modélisent l'essentiel d'un paradigme de programmation particulier : interactions par mémoire partagée (les modèles de coordination) versus communications par échanges de messages, communications point à point versus multi - synchronisation ou broadcast, systèmes synchrones versus systèmes asynchrones, distribution virtuelle versus distribution explicite. D'autres extensions essaient de prendre en compte certains aspects "quantitatifs" des systèmes réels : les propriétés probabilistes, la notion de temps.

Les aspects qui nous intéressent particulièrement dans cette thèse sont la mobilité et la diffusion.

Les systèmes de processus mobiles sont des systèmes de processus dont la topologie des communications peut varier au cours du temps. La possibilité de "changer" des liens de communications peut être interprétée comme le mouvement d'un processus dans un espace "virtuel", d'où la syntagme "processus mobiles". Dans les langages qui permettent de décrire de tels systèmes, la mobilité est modélisée par un mécanisme de génération dynamique de noms, plus la possibilité d'insérer (et donc de propager) des noms des canaux dans les messages échangés. L'exemple classique d'un calcul algébrique de processus mobiles, est le π-calcul ((Milner et al., 1992), (Milner, 1999)), un modèle qui utilise des primitives de communication point à point synchrones (par rendez-vous). Le π-calcul a beaucoup influencé notre modèle en ce qui concerne la manière de représenter la mobilité.

En ce qui concerne les applications, la diffusion est une caractéristique des systèmes distribuées dont la topologie de communication dépend de l'état des composants à un certain moment pendant l'exécution (Bayerdorffer, 1993). Dans cette classe d'applications, un certain message envoyé par un des composants doit être reçu par tous les participants dont l'état satisfait une certaine propriété.

Prenons comme exemple un algorithme distribué pour la la détection des cycles dans un graphe orienté (graphe construit dynamiquement). Supposons que les arêtes du graphe sont distribués à travers plusieurs objets (acteurs) et que les objets continuent à recevoir des nouvelles arêtes. Un possible algorithme est le suivant: chaque objet o, pour chaque arête (a,b) qu'il posséde, génére une clé unique (un jeton) $k^o_{(a,b)}$ qu'il envoie sous la forme $(k^o_{(a,b)},b)$ "à tous ceux" qui possédent une arête de la forme (b,x) avec x quelconque. De plus, un objet o', pour chaque message (k',x) qu'il reçoit et pour chaque arête de la forme (x,y) qu'il posséde, envoie le message (k',y) "à tous ceux" qui possédent une arête de la forme (y,z) avec z quelconque (les clés sont propagés à travers les chemins du graphe). Un cycle est détecté si un objet o reçoit un message de la forme $(k^o_{(a,b)},a)$ (le jeton a parcouru un cycle). Dans cet exemple, un message de la forme (k',x) doit être reçu "par tous ceux qui possédent une arête de la forme (x,y)", donc une diffusion. De plus, l'ensemble "de tous ceux qui possédent une arête de la forme (x,y)" peut croître dans le temps (le graphe est construit dynamiquement), et donc un message envoyé "à tous ceux" qui satisfont la propriété peut être reçu par des groupes d'objets differents, qui dépendent du moment de l'émission du message. Cet exemple simple sera étendu dans l'Exemple 18 à un algorithme distribué pour la détection des incohérences dans un système transactionnel distribué. L'exemple montre que la diffusion plus la mobilité sont des caractéristiques de certains applications distribuées.

En ce qui concerne les environnements de programmation qui proposent des primitives de groupe, leur nombre ne cesse pas d'accroître: Isis (Birman et al., 1990), Amoeba (Tanenbaum et al., 1990), Totem (Moser et al., 1995), Transis (Amir et al., 1992), Horus (Renesse et al., 1995), Ensemble (Hayden, 1998), PVM (Geist et al., 1994), etc. Dans tous ces exemples, les groupes sont des structures dynamiques: un processus peut joindre ou quitter dynamiquement un groupe (une fois que l'application a connaissance de l'existence du groupe, ou d'une adresse, port ou capacité qui sert d'alias du groupe).

Parmi la "grande famille" des algèbres des processus, il n'existe pas de modèle qui permettrait de modéliser des systèmes mobiles communiquant par diffusion.

La plupart des algèbres de processus utilisent le point à point comme primitive de communication. D'autres langages (CSP, Esterel) utilisent la multi-synchronisation. CBS ((Prasad, 1991), (Prasad, 1995)) utilise comme seule primitive de communication la diffusion (le broadcast). Le broadcast est la primitive de communication fournie par le matériel dans les réseaux locaux (Ethernet). Un autre argument apporté par Prasad pour justifier son calcul, est que la diffusion est le moyen naturel de communication entre les gens : l'action de parler est une diffusion.

Par rapport aux langages point à point, CBS est à l'autre extrême : tout le monde entend tout le monde. Seule exception : on peut définir des filtres qui assurent la confidentialité des messages par rapport à certains sous - groupes de processus. La limitation des filtres tel qu'ils sont présents dans CBS est qu'ils sont statiques (fixés une fois pour tout pour un certain processus). La possibilité d'envoyer une clé à certains participants qui leurs permettrait de comprendre des messages impossibles à déchiffrer (ou à "entendre") auparavant, n'est pas exprimable en CBS.

Ces considérations sont à l'origine de notre motivation pour le développement d'un modèle théorique qui combine à la fois la mobilité et la diffusion.

L'utilisation des calculs de processus pour la spécification des protocoles ou des systèmes distribués est encouragée aussi par le développement d'une riche famille de relations d'équivalences. Les plus utilisés sont:

– les bisimulations ((Park, 1981), (Milner, 1983));

– les équivalences induites par des tests (de Nicola et Hennessy, 1984).

Les bisimulations ont été introduites par Park (Park, 1981), et utilisées dans la théorie des systèmes distribués pour la première fois par Milner (Milner, 1983). Deux processus p et q sont bisimilaires si chaque action de p (q) peut être simulé par q (respectivement p), et les deux processus passent ensuite dans un état où ils restent bisimilaires. En fonction de ce qu'on entend par "action" on peut définir plusieurs notions de bisimulations.

Les équivalences induites par des tests ont été formalisés dans les algèbres de processus par de Nicola et Hennessy dans (de Nicola et Hennessy, 1984). Deux systèmes sont équivalents quand ils satisfont le même ensemble d'observateurs (tests). En fonction de la définition de l'univers des observateurs et de la notion de satisfaction, on peut définir plusieurs équivalences par des tests.

1.2 Contribution et plan de cette thèse

La principale contribution de cette thèse, le $b\pi$-calcul, est l'introduction et le développement d'un modèle simple et expressif qui intègre deux concepts largement utilisés dans les systèmes distribués: la mobilité et la diffusion.

Le Chapitre 2 présente quelques formalismes déjà existants dans la littérature, en insistant sur leur pouvoir croissant d'expression: communications point à point vers communications de groupe (en passant par la multi synchronisation), structures de communications statiques vers structures de communications dynamiques. Une classification des modèles est présentée dans le Tableau 2.9.

Dans le Chapitre 3 nous introduisons le $b\pi$-calcul. La syntaxe (Tableau 3.1) est inspirée de π-calcul. Ce choix nous permet d'avoir une manière simple de modéliser la mobilité, et facilite en même temps la preuve de certains résultats qui sont de nature essentiellement syntaxique. Ensuite nous présentons la sémantique opérationnelle (Tableau 3.5) de notre modèle (définie comme un système transitionnel étiqueté par des actions, sur l'ensemble de processus).

Nous montrons le pouvoir d'expression de notre modèle à travers plusieurs exemples. L'exemple 18 (un algorithme pour la détection des incohérences dans un système transactionnel) prouve que pour certains algorithmes, la diffusion plus la capacité de pouvoir insérer des noms de canaux dans les messages échangés, sont utils pour une description telle que le comportement de chaque participant ne dépend pas de certains parametres du système (par exemple, le nombre des autres "acteurs" existants dans le système).

Dans le Chapitre 4, nous introduisons plusieurs relations d'équivalences qui permettent d'identifier (ou de distinguer) deux processus en fonction de leurs réponses aux interactions avec l'environement.

Les bisimulations barbelées sont des relations d'équivalences qui peuvent être définies dans tout modèle disposant d'une relation de réduction (de réécriture) et muni d'une notion d'observabilité. Deux questions qui apparaissent normalement sont "quel doit être le prédicat d'observabilité?"et "quel relation de réduction doit - on prendre?". Pour la première question, dans un modèle à diffusion, les émissions sont observables, mais pas les réceptions (Hennessy et Rathke, 1995). Pour la deuxième question deux choix possibles induissent deux relations incomparables. Nous considerons leurs clôtures à certaines classes de contextes et nous montrons que les équivalences ainsi obtenues, coincident avec les bisimulations étiquetées pour les processus d'image finie.

Ensuite, nous considérons comme application la spécification d'un protocole de FIFO broadcast dans un réseau asynchrone de processus (il n'y a aucune garantie en ce qui concerne l'ordre des réceptions des messages). Par l'exhibition d'une relation de bisimulation appropriée on prouve la correction du protocole.

Nous terminons le chapitre par une étude des congruences induites par les relations de bisimulations, et par une axiomatisation complete des congruences fortes pour les processus finis.

Dans le Chapitre 5, nous montrons que les bisimulations (largement utilisées pour les systèmes distribués interactifs), sont trop restrictives dans certains cas pour les systèmes à diffusion. Alors nous adaptons les relations induites par des tests: le "must testing" et le "may testing" (de Nicola et Hennessy, 1984) pour les modèles à diffusion.

Ensuite nous présentons des caractérisations basées sur des traces pour le "must testing" et le "may testing", pour une variante sans passage de valeur de CBS et pour le $b\pi$-calcul. Ces caractérisations sont indépendants des observateurs. A notre connaissance, de telles études n'avaient pas été menés auparavant dans les modèles à diffusion.

Dans le Chapitre 6, nous nous intéressons tout d'abord aux relations entre les communications point à point et les primitives de diffusion. Utilisant l'existence (ou la non-existence) des solutions pour l'élection d'un leader dans un système distribué, nous montrons qu'il n'est pas possible d'implanter (sous certains conditions) des langages à diffusion (comme CBS ou $b\pi$ -calcul) dans des langages basés sur des communications point à point (comme CCS ou π-calcul).

Ensuite, nous analysons les opérateurs de choix et de test tels qu'ils sont dans le $b\pi$-calcul (étant différents des opérateurs homonymes du π-calcul où il existe seulement le choix "préfixé" et le test simple "if then"). Utilisant une relation préservée par tous les autres opérateurs du langage à l'exception de l'opérateur de choix, nous montrons l'impossibilité d'implanter le choix générale à l'aide des autres constructions du $b\pi$-calcul plus le choix "prefixé". Par une technique similaire, nous montrons l'impossibilité d'implanter le test "if then else" a l'aide des autres constructions du $b\pi$-calcul plus le test "if then ".

Nous terminons cette thèse par les conclusions de ce travail et quelques continuations possibles.

Chapitre 2

Une hiérarchie de modèles pour les systèmes distribués

Le processus de conception ou d'analyse d'un système distribué commence par la spécification architecturale des principaux composants (acteurs, processus, tâches, objets, etc.) qui coopèrent pour accomplir une tâche particulière. Cette étape est en général itérative: durant la conception on raffine de plus en plus les modules du système jusqu'au niveau approprié pour l'implantation; durant l'analyse, au contraire, on regroupe les entités, tout en essayant de prouver la correction de l'opération de composition, jusqu'au niveau d'abstraction nécessaire. La source de difficultés la plus importante réside dans la spécification des interactions ou communications entre les diverses entités du système. Le programmeur doit structurer correctement l'ordre des événements (communications) dans le déroulement de l'algorithme ou du protocole afin d'obtenir le comportement souhaité.

Dans l'exécution d'une application distribuée on peut identifier divers schémas d'interaction; tous ont en commun le fait que des informations circulent d'un ensemble d'objets source vers un ensemble d'objets destination dont la composition est dépendante de l'état global du système. Comme ces schémas peuvent varier d'une façon très significative, le choix des mécanismes pour la spécification des communications est une tâche importante.

Sous-jacent aux mécanismes de communication de tout système distribué se trouve un système de nommage qui associe des ensembles d'objets aux noms. Spécifier une communication revient à spécifier ses participants; donc, disposer de mécanismes plus expressifs pour indiquer (nommer) des ensembles d'objets permet des schémas de communications plus variés et plus complexes, et conduit donc à des spécifications plus simples. Pour qu'un objet A puisse communiquer avec un objet B, il faut qu'il existe un nom qui soit connu par A et qui soit lié à B. Un nom peut correspondre directement à un objet, être l'identificateur d'un processus, ou l'adresse IP d'une machine. En pratique, un nom peut être associé à un canal, un port, une variable partagée ou une méthode d'un objet. Indépendemment d'un mécanisme, toute communication ne peut s'effectuer sans une source et une destination, par l'intermédiaire d'un nom.

Un système de nommage est donc caractérisé par plusieurs éléments fonctionnels:

- *la connaissance* des noms : l'ensemble des noms connus par un objet est déterminé par un mécanisme de génération des noms et par un mécanisme de propagation des noms;
- *la correspondance* des noms est déterminée par un mécanisme d'association nom - ensemble d'objets, qui dépend généralement du contexte.

Certaines caractéristiques du système sont influencées par les propriétés du système de nommage :

- *l'abstraction* : les primitives de communications classiques sont en général point à point et souvent entre des objets nommés directement. Permettre aux objets d'être appelés indirectement (par des alias), et permettre aux noms d'être liés aux ensembles arbitraires d'objets (plutôt qu'à un seul objet à la fois) augmente le niveau d'abstraction;
- *la mobilité* : pour pouvoir décrire les applications composées d'un nombre de processus (objets) variant au cours du temps et communiquant à travers un réseau reconfigurable, le système de nommage doit permettre aux objets de générer ou d'apprendre des nouveaux noms à l'exécution.

Le but de cette thèse étant d'introduire et de développer un nouveau modèle pour des systèmes distribués mobiles, nous décrivons d'abord quelques formalismes existants dans la littérature, en insistant sur leur pouvoir d'expression : communications point à point vers communications de groupe (en passant par multi-synchronisations), structures de communications statiques vers structures de communications dynamiques.

2.1 Modèles statiques pour les systèmes concurrents

2.1.1 Calculus of Concurrent Systems (CCS)

L'algèbre de processus CCS ((Milner, 1980), (Milner, 1989)) est un des premiers modèles algébriques proposés pour les systèmes concurrents. CCS a été proposé pour fournir une abstraction mathématique aux applications communicant d'une façon synchrone binaire par des messages point à point (par rendez-vous). Il est proposé avec une sémantique opérationnelle, et les relations qui peuvent être établies entre des processus décrits en *CCS* sont généralement de nature comportementale (des bisimulations).

CCS fournit un ensemble réduit d'opérateurs qui permettent de construire des définitions de systèmes à partir des sous-systèmes. Les éléments de base pour les descriptions sont les *actions*. Intuitivement, les actions représentent des étapes d'exécution atomiques, qui peuvent être soit des séquences internes d'exécution, soit des interactions possibles avec l'environnement extérieur.

Soit Ch un ensemble dénombrable de *canaux*, qui ne contient pas le symbole τ. Les actions en *CCS* (dénotées par π) sont soit des actions internes, dénotées par τ, soit des émissions ($\bar{a}$ représente une émission sur le canal ou le port a) soit des réceptions (a représente une réception sur le canal ou le port a).

La syntaxe des processus *Proc* est donnée par la grammaire présentée dans le Tableau 2.1.

$$Proc \ni p \ ::= \quad nil \mid \pi.p \mid \nu Lp \mid p[\![f]\!] \mid p_1 + p_2 \mid p_1 \parallel p_2 \mid X \mid rec\ X.p$$

TAB. 2.1 – *Processus en CCS*

où π est une action $\pi ::= \tau \mid a \mid \bar{a}$, avec $a \in Ch$ et $L \subseteq Ch$ est un ensemble fini de canaux.

- nil est le processus qui a fini son exécution (il fournit la base des définitions structurelles récursives des processus plus complexes);
- $\pi.p$ est le processus (construit par préfixage d'un processus p), qui fait d'abord l'action π et se comporte ensuite comme p;
- dans νLp tous les canaux de L sont internes, ils sont donc cachés à l'environnement;
- $p[\![f]\!]$ permet de renommer les actions du processus p (f doit satisfaire les conditions suivantes: $f(\tau) = \tau$ et $f(\bar{a}) = \overline{f(a)}$);
- $p_1 + p_2$ représente le choix non-déterministe; ce processus construit à partir de p_1 et p_2, se comporte soit comme p_1 soit comme p_2, en fonction des interactions offertes par l'environnement;
- $p_1 \parallel p_2$ est la composition parallèle de p_1 et p_2;
- $recX.p$ est une définition récursive permettant de décrire des systèmes avec des comportements infinis (X étant un identificateur de processus).

La sémantique opérationnelle de *CCS* est présentée dans le Tableau 2.2 (nous avons omis les versions symétriques des règles (6), (7) et (8)). $p[q/X]$ représente la substitution du processus q à la variable X.

(1) $\dfrac{}{\tau.p \xrightarrow{\tau} p}$	(2) $\dfrac{}{a.p \xrightarrow{a} p}$	(3) $\dfrac{}{\bar{a}.p \xrightarrow{\bar{a}} p}$
(4) $\dfrac{p \xrightarrow{\pi} p' \wedge \pi \notin L \cup \bar{L}}{\nu L p \xrightarrow{\pi} \nu L p'}$	(5) $\dfrac{p \xrightarrow{\pi} p'}{p[\![f]\!] \xrightarrow{f(\pi)} p'[\![f]\!]}$	(6) $\dfrac{p_1 \xrightarrow{\pi} p' \vee p_2 \xrightarrow{\pi} p'}{p_1 + p_2 \xrightarrow{\pi} p'}$
(7) $\dfrac{p_1 \xrightarrow{\bar{a}} p_1' \wedge p_2 \xrightarrow{a} p_2'}{p_1 \| p_2 \xrightarrow{\tau} p_1' \| p_2'}$	(8) $\dfrac{p_1 \xrightarrow{\pi} p_1'}{p_1 \| p_2 \xrightarrow{\pi} p_1' \| p_2}$	(9) $\dfrac{p[(rec\ X.p)/X] \xrightarrow{\pi} p'}{(rec\ X.p) \xrightarrow{\pi} p'}$

TAB. 2.2 – *Sémantique opérationnelle de CCS*

Pour établir l'équivalence en CCS, entre une spécification et une implantation, on utilise des bisimulations.

Définition 1 *La* **bisimulation forte** *(notée $\sim$) est la plus grande (au sens de l'inclusion) relation symétrique S sur l'ensemble Proc des processus telle que $(p,q) \in S$ implique pour chaque action π (telle que $p \xrightarrow{\pi} p'$, qu'il existe q' tel que $q \xrightarrow{\pi} q'$ et $(p',q') \in S$.*

Si on prend en compte le fait que τ est une action interne, on peut alors définir une variante faible de la bisimulation (notée $\approx$) qui ne fait pas de différence entre deux processus qui ont le même comportement pour un observateur extérieur, mais qui peuvent faire un nombre différent de transitions internes.

Par exemple on peut montrer que les deux processus suivants sont faiblement équivalents, $Spec \approx Tamp2$, où $Spec$ est la spécification d'un tampon à deux places, et $Tamp2$ est une implantation qui utilise pour cela deux tampons à une place.

$Spec \stackrel{def}{=} recX.\{in.[recY.(in.\overline{out}.Y + \overline{out}.X)] + \overline{out}.X\}$

$Tamp1 \stackrel{def}{=} recX.(in.\overline{out}.X)$

$Tamp2 \stackrel{def}{=} \nu\{c\}(Tamp1[\![f_1]\!] \parallel Tamp2[\![f_2]\!])$

$$f_1(a) \stackrel{def}{=} \begin{cases} c & \text{if } a = out, \\ a & \text{autrement.} \end{cases} \tag{2.1}$$

$$f_2(a) \stackrel{def}{=} \begin{cases} c & \text{if } a = in, \\ a & \text{autrement.} \end{cases} \tag{2.2}$$

En conclusion, CCS permet de spécifier et d'analyser des systèmes distribués statiques , qui communiquent par rendez-vous. En CCS il n'est pas possible de décrire une communication qui implique plus de deux participants. De plus, les noms (canaux, ports, etc.) connus et utilisés par un objet pour communiquer restent les mêmes pendant toute l'exécution (ce qui empêche CCS de pouvoir décrire des systèmes mobiles, c.à.d. des systèmes qui ont une topologie de communication reconfigurable). $SCCS$ (Milner, 1983) est une extension de CCS qui permet des interactions concernant un nombre quelconque de participants. En $SCCS$ les actions sont les éléments d'un groupe commutatif et la composition parallèle est remplacée par un opérateur produit qui exige que tous les composants du systèmes s'exécutent d'une manière fortement synchrone. Si $SCCS$ permet d'exprimer indirectement des systèmes de communication à diffusion, il subsiste l'impossibilité de décrire des systèmes mobiles.

2.1.2 Communicating Sequential Processes (CSP)

CSP, inventé par Hoare ((Hoare, 1978),(Hoare, 1985)), est le premier (en ordre chronologique) langage formel pour la spécification de processus pouvant communiquer entre eux et avec leur environnement. Comme en CCS, les processus peuvent participer à un ensemble d'événements Σ fixés à priori. Contrairement à CCS, en CSP le modèle d'interaction est la multi-synchronisation. Plusieurs processus peuvent participer au même événement en même temps. Une autre différence est le traitement du choix: il y a deux constructions qui permettent de distinguer entre le non-déterminisme interne et le choix externe. CSP utilise une sémantique dénotationnelle pour donner une signification à un processus.

La syntaxe des processus CSP (avec les notations de (Roscoe, 1998)) est donnée par la grammaire présentée dans le Tableau 2.3.

$$\begin{array}{lll} p & ::= & SKIP \mid STOP \mid a \rightarrow p \mid \nu L p \mid p[\![f]\!] \mid p_1 \langle b \rangle p_2 \\ & & p_1 \sqcap p_2 \mid p_1 \Box p_2 \mid p_1 \parallel_A p_2 \mid X \mid rec\ X.p \end{array}$$

TAB. 2.3 – *Processus en CSP*

où $a \in \Sigma$ est un événement, $A, L \subseteq \Sigma$ sont des ensembles finis d'évènements et b est une expression booléenne.

- $SKIP$ est le processus qui finit correctement son exécution, tandis que $STOP$ est le processus bloqué (les deux processus ne peuvent pas communiquer avec l'environnement et permettent de distinguer entre la terminaison correcte d'un programme et l'interblocage); ils fournissent la base des définitions structurelles récursives des processus plus complexes.
- $a.p$ est le processus qui fait d'abord l'action a et se comporte ensuite comme p (en CSP il est possible de simuler de manière indirecte les entrées/sorties);
- dans $\nu L p$ toutes les actions de L sont internes, et sont cachées à l'environnement;
- $p[\![f]\!]$ permet de renommer les actions du processus p;

- $p_1 \langle b \rangle p_2$ teste la condition b, et en fonction du résultat se comporte soit comme p_1, soit comme p_2;
- $p_1 \sqcap p_2$ et $p_1 \Box p_2$ permettent de distinguer entre le non-déterminisme interne et le choix externe; $p_1 \sqcap p_2$ se comporte soit comme p_1, soit comme p_2 (le choix étant interne), tandis que pour $p_1 \Box p_2$ le choix est fait par l'environnement (en fonction des événements qu' il propose en introduction aux deux processus);
- $p_1 \parallel_A p_2$ est la composition parallèle de p_1 et p_2; p_1 et p_2 doivent se synchroniser sur tous les événements présents dans A;
- $recX.p$ est une définition récursive permettant de décrire des systèmes avec des comportements infinis.

Même s'il existe plusieurs sémantiques pour CSP (opérationnelle, algébrique (axiomatique)), la sémantique de base pour le calcul reste la sémantique dénotationnelle. La sémantique classique de CSP associe à tout processus p, une paire ($echecs(p)$,$divergences(p)$). $echecs(p)$ est un ensemble de paires de la forme (s, X) où s est une trace valide (une séquence ordonnée d'actions) du processus p , et X est un ensemble d'événements que p peut refuser après l'exécution de la trace s. $divergences(p)$ réunit les traces qui peuvent conduire p dans un état de divergence (où il peut exécuter un nombre infini d'actions internes τ).

CSP permet d'exprimer des interactions entre un nombre quelconque de participants, mais il ne fait pas de distinction sémantique entre les entrées et les sorties (il y a seulement une distinction syntaxique). Comme CCS et $SCCS$, il ne permet pas de décrire des systèmes mobiles.

CCS et CSP ont inspiré le développement de $LOTOS$ (Bolognesi et Brinksma, 1987), une notation pour la spécification des protocoles qui est devenu un standard ISO.

2.1.3 I/O automata

I/O automata (proposé par N. Lynch et M. Tuttle dans (Lynch et Tuttle, 1989)) est un formalisme utilisé pour la description et l'analyse des algorithmes distribués. Un I/O automata est caractérisé par son interface (qui détermine les actions qu'il peut faire) et par une relation de transition (qui établit l'ensemble des états dans lesquels l'automate peut passer à la suite des actions). Les actions sont de trois types: les sorties, les actions internes et les entrées. Les automates peuvent être composés en parallèle en identifiant leurs actions externes, à condition que leurs interfaces soient compatibles (les ensembles des sorties et des actions internes des composants doivent être distincts). Les méthodes de preuve souvent utilisées sont les techniques basées sur les invariants (pour prouver qu'une propriété est satisfaite par tous les états accessibles) et les simulations (pour montrer qu'un automate est l'implantation d'un autre automate). Pour faciliter l'automatisation des techniques de preuve, les I/O automata ont reçu une caractérisation algébrique dans (de Nicola et Segala, 1995) et (Vaandrager, 1991).

La diffusion est l'interaction de base dans les I/O automata (simulée par une sortie d'un automate qui est une action présente comme entrée dans plusieurs automates). Les I/O automates classiques sont appropriés seulement pour l'analyse des systèmes distribués statiques. Dans la proposition originale, (Lynch et Tuttle, 1989), la mobilité peut être simulée indirectement et d'une façon très limitée (par exemple, des transactions créées dynamiquement sont modélisées à l'aide des actions déjà existantes et "réveillées" par d'autre actions).

Pour résoudre cette limitation, très récemment, P. Attie et N. Lynch (Attie et Lynch, 2001) ont introduit les I/O automata dynamiques comme un modèle formel pour les systèmes mobiles. Les I/O automata classiques sont étendus avec des actions spéciales (*create, destroy, signature change*) pour modéliser des systèmes dynamiques, où de nouveaux automates ou de nouvelles références à des automates déjà existants peuvent être créés à l'exécution.

2.1.4 Calculus of Broadcasting Systems (CBS)

CBS (inventé par S. Prasad (Prasad, 1991), (Prasad, 1995)) est un calcul des processus qui a pour objectif de donner un modèle théorique aux architectures matérielles à base d'un médium à diffusion, et plus précisément aux réseaux locaux. La diffusion est une opération audible par tout composant du système, mais le message est lu seulement par ceux censés le recevoir. Le mode d'interaction de base en CBS est donc la diffusion immédiate sans tampon; tout message a exactement un émetteur et zéro ou plusieurs récepteurs. Notre présentation de CBS est inspirée de (Hennessy et Rathke, 1995).

Soit $Valexp$ un ensemble d'expressions et $Boolexp$ un ensemble d'expressions booleennes. On ne donne pas une signification plus précise à $Valexp$, mais on suppose seulement qu'il contient un symbole spécial τ et un ensemble de valeurs Val. Les processus sont définis par la grammaire donnée dans le Tableau 2.4 (l'ensemble des processus sera noté par $\mathcal{P}_b$).

$$\mathcal{P}_b \ni p \ ::= \quad nil \mid \pi.p \mid \langle b \rangle p_1, p_2 \mid p_1 + p_2 \mid p[\![f,g]\!] \mid p_1 \parallel p_2 \mid X \mid rec\, X.p$$

TAB. 2.4 – *Processus en CBS*

où $\pi ::= x \in S? \mid \bar{e}$, avec x variable, $S \subseteq Valexp \setminus \{\tau\}$ et $e \in Valexp$.

- nil, $p_1 \parallel p_2$, $p_1 + p_2$ et $recX.p$ ont la même signification qu'en CCS;
- $\bar{e}.p$ est le processus qui envoie (dit) l'expression e et se comporte ensuite comme p;
- $x \in S?.p$ est le processus qui entend une expression v appartenant à S, et ensuite se comporte comme $p[v/x]$; si $v \notin S$, alors le procesus $x \in S?.p$ ignore la communication, et reste inchangé;
- dans $p[\![f,g]\!]$, f et g sont utilisés pour filtrer les messages entre un processus et l'environement (on peut ainsi simuler la restriction et le renommage au sens de CCS). Ils permettent à certains messages d'être cachés pour l'environnement. (f et g doivent satisfaire la condition suivante : $f(\tau) = g(\tau) = \tau$, qui confirme l'intuition que le bruit ne peut pas être traduit dans une valeur intelligible).

Pour pouvoir présenter la sémantique opérationnelle de CBS, nous avons besoin (comme dans (Prasad, 1995) et (Hennessy et Rathke, 1995)), d'une relation "ignore" $\longrightarrow \subseteq \mathcal{P}_b \times Valexp$ telle que si $(p,v) \in \longrightarrow$ on note $p \xrightarrow{v:}$ et on entend "p ne peut pas entendre l'expression v" (voir Tableau 2.5.). Intuitivement un processus ignore une valeur qu'il n'est pas censé entendre.

(1) $\dfrac{}{nil \xrightarrow{\alpha:}}$	(2) $\dfrac{}{\bar{e}.p \xrightarrow{v:}}$	(3) $\dfrac{v \notin S}{x \in S?.p \xrightarrow{v:}}$
(4) $\dfrac{b \equiv true \wedge p \xrightarrow{v:}}{\langle b \rangle p,q \xrightarrow{v:}}$	(5) $\dfrac{b \equiv false \wedge q \xrightarrow{v:}}{\langle b \rangle p,q \xrightarrow{v:}}$	(6) $\dfrac{p \xrightarrow{g(v):}}{p[\![f,g]\!] \xrightarrow{v:}}$
(7) $\dfrac{p_1 \xrightarrow{v:} \wedge p_2 \xrightarrow{v:}}{p_1 + p_2 \xrightarrow{v:}}$	(8) $\dfrac{p_1 \xrightarrow{v:} \wedge\ p_2 \xrightarrow{v:}}{p_1 \parallel p_2 \xrightarrow{v:}}$	(9) $\dfrac{p[(rec\ X.p)/X] \xrightarrow{v:}}{rec\ X.p \xrightarrow{v:}}$

TAB. 2.5 – *La relation "ignore"*

La sémantique opérationnelle de CBS est présentée dans le Tableau 2.6. (nous avons omis les versions symétriques de (9), (10) et (11)).

Comme en CCS, pour établir l'équivalence entre une spécification et une implantation, on utilise les bisimulations. Mais dans un calcul à diffusion, l'émission ne dépend plus de l'existence d'un auditeur et donc elle n'est plus bloquante. En conséquence, contrairement à CCS, un observateur ne peut plus s'apercevoir directement si un processus reçoit ou ignore une valeur.

Définition 2 *La* **bisimulation forte** *(notée $\sim$) est la plus grande (au sens de l'inclusion) relation symétrique S sur l'ensemble $\mathcal{P}_b$ des processus tel que $\forall (p,q) \in S$ les propriétés suivantes sont*

(1) $\dfrac{}{\bar{v}.p \xrightarrow{\bar{v}} p}$	(2) $\dfrac{v \in S}{x \in S?.p \xrightarrow{v?} p[v/x]}$	(3) $\dfrac{b \equiv true \wedge p \xrightarrow{\alpha} p'}{\langle b \rangle p,q \xrightarrow{\alpha} p'}$
(4) $\dfrac{b \equiv false \wedge q \xrightarrow{\alpha} q'}{\langle b \rangle p,q \xrightarrow{\alpha} q'}$	(5) $\dfrac{p \xrightarrow{g(v)?} p'}{p[\![f,g]\!] \xrightarrow{v?} p'}$	(6) $\dfrac{p \xrightarrow{\bar{v}} p'}{p[\![f,g]\!] \xrightarrow{\overline{f(v)}} p'}$
(7) $\dfrac{p_1 \xrightarrow{\alpha} p' \vee p_2 \xrightarrow{\alpha} p'}{p_1 + p_2 \xrightarrow{\alpha} p'}$	(8) $\dfrac{p_1 \xrightarrow{v?} p_1' \wedge p_2 \xrightarrow{v?} p_2'}{p_1 \| p_2 \xrightarrow{v?} p_1' \| p_2'}$	(9) $\dfrac{p_1 \xrightarrow{\bar{v}} p_1' \wedge p_2 \xrightarrow{v?} p_2'}{p_1 \| p_2 \xrightarrow{\bar{v}} p_1' \| p_2'}$
(10) $\dfrac{p_1 \xrightarrow{\bar{v}} p_1' \wedge p_2 \xrightarrow{v:}}{p_1 \| p_2 \xrightarrow{\bar{v}} p_1' \| p_2}$	(11) $\dfrac{p_1 \xrightarrow{v?} p_1' \wedge p_2 \xrightarrow{v:}}{p_1 \| p_2 \xrightarrow{v?} p_1' \| p_2}$	(12) $\dfrac{p[(rec\ X.p)/X] \xrightarrow{\alpha} p'}{(rec\ X.p) \xrightarrow{\alpha} p'}$

TAB. 2.6 – *Sémantique opérationnelle de CBS*

satisfaites:

- *pour chaque émission $\bar{v}$ tel que $p \xrightarrow{\bar{v}} p'$, il existe q' tel que $q \xrightarrow{\bar{v}} q'$ et $(p',q') \in S$;*
- *pour chaque réception $v?$ tel que $p \xrightarrow{v?} p'$, soit il existe q' tel que $q \xrightarrow{v?} q'$ et $(p',q') \in S$, soit $q \xrightarrow{v:}$ et $(p',q) \in S$.*

La bisimulation faible $\approx$ est obtenue de $\sim$, d'une façon similaire à celle de CCS.

En CBS, la communication se fait par diffusion sans tampon. Comme dans les I/O automata et dans CCS, il y a une distinction claire entre les émissions et les réceptions. La topologie des communications est fixée au début une fois pour toutes, et il n'est pas possible de décrire des systèmes mobiles.

2.2 Modèles dynamiques pour les systèmes concurrents

2.2.1 Systèmes d'acteurs

L'"ancêtre"de tous les modèles pour la concurrence est probablement le système d'acteurs (inventé par C. Hewitt (Hewitt, 1977), et développé par G. Agha (Agha, 1986), (Agha et al., 1993)). Un système d'acteurs est une collection d'acteurs qui s'exécutent en parallèle. Les acteurs sont des objets actifs qui communiquent d'une façon asynchrone par messages stockés dans des boites aux lettres. Le comportement d'un acteur spécifie la réponse induite par la réception d'un message :

- *envoyer* des messages à d'autres acteurs;
- *créer* d'autres acteurs avec des comportement spécifiés à la naissance;
- *changer* en un nouvel acteur qui commencera par recevoir le message suivant.

La mobilité du système est donnée par la capacité des acteurs à créer pendant l'exécution de nouveaux acteurs, et par la possibilité d'insérer des noms (des adresses) des acteurs dans les messages échangés. Les systèmes d'acteurs ont été proposés comme un modèle pour des *systèmes distribués ouverts*: l'addition de nouveaux composants, le remplacement d'un composant existant ou le changement dans les interconnexions entre les composantes ne doit pas empiéter sur le fonctionnement correct du système.

Pour faciliter l'application des techniques de preuves algébriques dans le monde orienté objet, A. Dalmonte, M. Gaspari et G. Zavattaro ((Dalmonte et al., 1995),(Gaspari et Zavattaro, 1999)) ont proposé une algèbre de processus inspiré des systèmes d'acteurs. Étant définie par l'intermédiaire d'un système transitionnel étiqueté, leur sémantique opérationnelle permet d'utiliser les équivalences standards des algèbres de processus (en particulier celles développées pour le π-calcul asynchrone (Honda et Tokoro, 1991) et (Amadio et al., 1998)).

Le système des acteurs est le premier formalisme pour les applications distribuées où la mobilité occupe un rôle important. La primitive de communication est l'échange de messages asynchrone point à point.

2.2.2 Linda

Linda (Carriero et Gelernter, 1989) est un langage de coordination. Le modèle de programmation offert par Linda est construit à partir d'une mémoire associative partagée : l'espace des *tuples*. L'espace des tuples est un multi-ensemble (il peut y avoir plusieurs copies du même tuple en même temps). Un tuple est un message composé de plusieurs champs de données (de divers types). Les processus qui s'exécutent en parallèle communiquent à travers l'espace des tuples. Ils disposent de six primitives atomiques pour manipuler l'espace partagé :

– $out(t)$ ajoute un élément t à l'espace des tuples;
– $eval(t)$ ajoute un élément "actif" t à l'espace des tuples qui peut contenir des champs fonctions; une fois que le tuple a fini d'évaluer ces fonctions, il redevient un tuple ordinaire;
– $in(t)$ recherche un tuple puis le retire de l'espace des tuples. Le tuple donné en paramètre spécifie les éléments de recherche; cette recherche par un "pattern matching" retrouve tous les tuples qui correspondent. Si il y en a plusieurs, l'un d'entre eux est choisi de manière non déterministe, est extrait de l'espace, et les champs non spécifiés de l'appel à in prennent alors les valeurs de ceux du tuple. La primitive est bloquante : s'il n'existe aucun tuple qui répond aux critères de sélection, le processus appelant est mis en attente jusqu'à ce que l'espace contienne un tuple qui

corresponde;

– $rd(t)$ est similaire à in, mais le tuple n'est pas supprimé de l'espace des tuples.
– $inp(t)$ et $rdp(t)$ ont le même role que in et rd respectivement, mais ne sont pas bloquants.

La mobilité des systèmes construits en Linda est induite par la possibilité d'insérer des noms (des adresses, des valeurs) spécifiques à certains processus dans les tuples (et qui ensuite peuvent être utilisés comme critères de sélection pour d'autres opérations de lecture par d'autres processus).

Comme pour les autres modèles, des algèbres de processus inspirés par Linda ont été proposés ((Busi et al., 1998), (Ciancarini et al., 1995)), mais elles concernent un sous-ensemble de Linda qui ne contient pas de mobilité.

2.2.3 π - calcul

Le π-calcul ((Milner et al., 1992), (Milner, 1999)) est un calcul de processus qui permet de décrire d'une manière directe et naturelle des systèmes mobiles. Non seulement des agents (ou des objets) peuvent être créés dynamiquement, et les composants du système peuvent être arbitrairement liés, mais en plus, les messages peuvent changer dynamiquement la topologie de communication.

Nous présentons la variante monadique du calcul. Les noms de canaux (notés ici par Ch_p) sont les seules valeurs utilisées dans ce langage. Ils servent en même temps comme support de communication et comme valeur transmise.

Les processus sont définis par la grammaire présentée dans le Tableau 2.7.

$p,p_1,p_2 ::=$	$nil \mid pre \mid \nu xp \mid \langle x=y\rangle p \mid p_1 \parallel p_2 \mid A\langle\tilde{x}\rangle \mid (rec\ A\langle\tilde{x}\rangle.p)\langle\tilde{y}\rangle$
$pre,pre_1,pre_2 ::=$	$\alpha.p \mid pre_1 + pre_2$

TAB. 2.7 – *Processus dans le π-calcul*

où α appartient à l'ensemble des actions ($\alpha ::= x(y) \mid \bar{x}y \mid \tau$ avec $x,y \in Ch_p$).

– les préfixes représentent des actions qu'un processus peut faire: $x(y)$ est *la réception* du nom y sur le canal x, $\bar{x}y$ est *l'émission* du nom y sur le canal x, et τ représente un pas interne;
– nil est le processus qui a fini son exécution (il fournit la base des définitions structurelles récursives des processus plus complexes);
– $\alpha.p$ est le processus qui fait d'abord l'action α et se comporte ensuite comme p;

- $\nu x p$ est la création d'un nouveau nom de canal x, différent de tous les noms déjà définis (et dont la portée initiale est le processus p);
- $\langle x = y \rangle p$ se comporte comme p si x et y sont identiques, et comme nil autrement;
- $p_1 + p_2$ représente le non-déterminisme, il se comporte soit comme p_1 soit comme p_2, en fonction des interactions offertes par l'environnement;
- $p_1 \parallel p_2$ est la composition parallèle de p_1 et p_2;
- $(rec A\langle \tilde{x} \rangle.p)\langle \tilde{y} \rangle$ est une définition récursive permettant de décrire des systèmes avec des comportements infinis ($\tilde{x}$ et $\tilde{y}$ doivent respecter l'arité de l'identificateur A).

$\nu x p$ et $y(x).p$, lient les apparitions du x dans p. Sinon , x est *libre* dans p. Pour un processus p, $bn(p)$ et $fn(p)$ représentent respectivement l'ensemble des noms liés, l'ensemble des noms libres dans p. L'ensemble des noms de p est noté par $n(p)$. *L'alpha-conversion* (le rennommage des noms liés) est défini d'une manière clasique (voir par exemple le Tableau 3.3).

Définition 3 **Les actions**, *sont définies par la grammaire suivante:*

$$\alpha \stackrel{def}{=} a(x) \mid \bar{a}x \mid \nu x\bar{a}x \mid \tau$$

où $a,x \in Ch_p$*: une action est soit une réception, soit une émission (qui peut être accompagnée par l'exportation d'un nom) soit un pas interne. Les ensembles des noms liés et libres pour les actions sont définis de la même façon que pour les processus.*

La sémantique opérationnelle (décrite comme un système transitionnel étiqueté par des actions) est présentée dans le Tableau 2.8 (nous identifions les processus alpha - équivalents et nous avons omis les versions symétriques de (7), (8) (9) et (10)).

(1) $\dfrac{}{\tau.p \xrightarrow{\tau} p}$	(2) $\dfrac{}{a(x).p \xrightarrow{a(z)} p[z/x]}$	(3) $\dfrac{}{\bar{a}x.p \xrightarrow{\bar{a}x} p}$
(4) $\dfrac{p \xrightarrow{\bar{a}x} p'}{\nu x p \xrightarrow{\nu x\bar{a}x} p'}$	(5) $\dfrac{p \xrightarrow{\alpha} p' \wedge x \notin n(\alpha)}{\nu x p \xrightarrow{\alpha} \nu x p'}$	(6) $\dfrac{p \xrightarrow{\alpha} p'}{\langle x{=}x \rangle p \xrightarrow{\alpha} p'}$
(7) $\dfrac{p_1 \xrightarrow{\alpha} p'}{p_1 + p_2 \xrightarrow{\alpha} p'}$	(8) $\dfrac{p \xrightarrow{\alpha} p' \wedge\ bn(\alpha) \notin fn(q)}{p \Vert q \xrightarrow{\alpha} p' \Vert q}$	(9) $\dfrac{p \xrightarrow{\bar{a}x} p' \wedge q \xrightarrow{a(x)} q'}{p \Vert q \xrightarrow{\tau} p' \Vert q'}$
(10) $\dfrac{p \xrightarrow{\nu x\bar{a}x} p' \wedge q \xrightarrow{a(x)} q'}{p \Vert q \xrightarrow{\tau} \nu x(p' \Vert q')}$	(11) $\dfrac{p[(rec\ A\langle \tilde{x} \rangle.p)/A, \tilde{y}/\tilde{x}] \xrightarrow{\alpha} p'}{(rec\ A\langle \tilde{x} \rangle.p)\langle \tilde{y} \rangle \xrightarrow{\alpha} p'}$	

TAB. 2.8 – *La sémantique opérationnelle du* π*-calcul*

Le π-calcul étant inspiré de CCS, il hérite aussi des nombreuses relations d'équivalence entre processus définies en CCS, et en particulier des bisimulations. Nous donnerons plus de détails dans les chapitres suivants (chapitres 3 et 4), par des comparaisons avec les mêmes notions correspondant à notre modèle.

En conclusion, en π-calcul, les processus communiquent de manière synchrone, point à point, à travers des canaux (qui peuvent être des canaux privés). La topologie de communication est dynamique. Le calcul est muni d'une sémantique mathématique bien fondée, tout en restant à la fois simple et expressif. Il permet de coder le λ-calcul et les structures de données (les entiers, les booleens, etc.) et il a été utilisé pour prouver des propriétés de correction des protocoles (par exemple le protocole PLMN pour le GSM (Orava et Parrow, 1992)). De nombreuses variantes de ce langage ont été proposées et étudiées: le π-calcul d'ordre supérieur (Sangiorgi, 1992), le π-calcul asynchrone ((Boudol, 1992), (Honda et Tokoro, 1991)), le join calculus (Fournet et Gonthier, 1996), le π-calcul distribué (Hennessy et Riely, 1998), le fusion calculus (Victor, 1998).

2.2.4 Higher Calculus of Broadcasting Systems ($HOBS$)

$HOBS$ (très récemment proposé par K. Ostrovský, K.V.S. Prasad et W. Taha dans (Ostrovský et al., 2001)) est un calcul des processus qui a pour objectif de donner un modèle théorique aux réseaux locaux (de type Ethernet). Il est inspiré de CBS et du λ-calcul (qu'il contient comme sous-calcul).

Les communications se font par l'intermédiaire de l'éther : un message émis par un processus, est entendu instantanément par tous les processus qui font partie du même "sous-réseau"; les processus doivent consommer tous les messages qu'ils entendent; seulement un message peut être envoyé à un certain moment.

HOBS est différent de CBS principalement dans les aspects suivants :

- $HOBS$ est d'ordre supérieur (les messages envoyés sont des processus), contrairement à CBS où les messages étaient des valeurs d'un langage support;
- $HOBS$ permet de modéliser les échanges asynchrones de messages (par un opérateur spécial $\lhd$).

La mobilité en HOBS dérive de la possibilité de pouvoir transmettre des processus et donc des copies qui peuvent s'exécuter ailleurs dans le système.

2.3 Pourquoi un autre modèle pour la mobilité?

Sans avoir la prétention d'être exhaustif, nous avons passé en revue les principaux formalismes pour la concurrence, en insistant surtout sur les modèles algébriques. Une classification des modèles est présentée dans le Tableau 2.9.

Modèle	**mobilité**	**mode d'intéraction**
CCS	non	point à point (rendez-vous)
CSP	non	multi - synchronisation
I/O automata	non	broadcast limité
CBS	non	broadcast
système d'acteurs	oui	point à point (asynchrone)
Linda	oui	memoire partagée
π-calcul	oui	point à point (rendez-vous)
$HOBS$	oui	broadcast
dynamic I/O automata	oui	broadcast limité
$b\pi$ - calcul	oui	broadcast

TAB. 2.9 – *Classification des modèles*

Les modèles les plus anciens pour les systèmes distribués sont statiques (avec l'exception notable des systèmes des acteurs). Ainsi, les modèles étaient moins complexes et permettaient de traiter certains problèmes dans un cadre plus simple. Pourtant, la mobilité est classique dans les systèmes d'exploitation : par exemple une ressource qui sert à un seul utilisateur à la fois, mais dont l'identité peut changer avec le temps; dans l'équilibrage des charges, des tâches ou des processus peuvent être échangés entre des processeurs différents. Dans les langages parallèles, une procédure peut être appelée par plusieurs processus, et il faut garantir que chaque instance activée de la procédure renvoie le bon résultat au bon processus appelant. Avec les développements récents des technologies logicielles ou matérielles, les applications des systèmes distribués mobiles deviennent de plus en plus nombreuses: les applications utilisant Internet, les systèmes de cartes à puce, la téléphonie mobile. Parmi les formalismes présentés qui prennent en compte la mobilité, le π-calcul et les systèmes d'acteurs considèrent comme primitive de base l'échange point à point. La diffusion, au mieux, peut être exprimée d'une manière indirecte. Et pourtant, la diffusion est un moyen naturel de communication: l'action de parler est une diffusion; les communications radio ou les transmissions pour des architectures à base de bus de données sont des diffusions. De plus en plus de bibliothèques de primitives de communication ou des systèmes distribués fournissent des primitives de groupe: Isis (Birman et al., 1990), Amoeba (Tanenbaum et al., 1990), Totem (Moser et al., 1995), Transis (Amir et al., 1992), Horus (Renesse et al., 1995), Ensemble (Hayden, 1998), PVM (Geist et al., 1994), etc. Dans tous ces exemples, les groupes sont des structures dynamiques: un processus peut joindre ou quitter dynamiquement un groupe (une fois que l'application a connaissance de l'existence du groupe, ou d'une adresse, port ou capacité qui sert d'alias du groupe).

Toutes ces raisons fournissent la motivation essentielle de notre travail pour le développement d'un modèle théorique complet qui combine à la fois mobilité et diffusion. Parmi les modèles présentés qui permettent de décrire les systèmes mobiles, les seuls formalismes qui considèrent la diffusion comme moyen de communications sont les I/O automata dynamiques et $HOBS$. Les deux modèles sont parus indépendemment et après la proposition du $b\pi$-calcul (Ene et Muntean, 1999).

Les automates I/O dynamiques (Attie et Lynch, 2001) sont une extension des automates I/O. Avec l'ajout de la dynamicité, le modèle devient assez complexe. Les restrictions du modèle (qui exigent qu'un seul automate ait le droit de faire une action comme action de sortie - ce qui revient au fait qu'un seul processus a le droit de "parler" dans un groupe) font que les automates I/O dynamiques sont peu appropriés pour la spécification et l'analyse des systèmes distribués mobiles qui communiquent par diffusion.

$HOBS$ (Ostrovský et al., 2001) est l'extension de CBS à un calcul de processus d 'ordre supérieur. Contrairement à tous les autre formalismes présentés (hors CBS), il ne considère pas la notion d'adresse (port, canal, action, etc...) comme étant une notion primitive. Contenant le λ-calcul, $HOBS$ peut encoder les structures de données, en particulier les booléens et les entiers. Ensuite, en prenant l'énumération des entiers de Gödel, il est possible de coder le π-calcul. Mais cet encodage est très complexe, et sa correction reste à prouver. $HOBS$, en dépit de son intérêt théorique, semble peu approprié pour la spécification et l'analyse des algorithmes ou protocoles utilisant la diffusion.

Pour nous, la notion d'adresse (ou nom) est essentielle dans la spécification d'un système. Le parallélisme et le non-déterminisme sont des caractéristiques implicites de tout système distribué. Pour modéliser les processus qui peuvent naître dynamiquement à l'exécution, ou les appels multiples d'une même procédure, un opérateur de génération de noms semble irremplaçable. De plus, propager les noms d'un objet à l'autre pendant l'exécution implique la nécessité de pouvoir insérer des noms (des adresses) dans les messages échangés. Toutes ces raisons, et la volonté de garder le modèle assez simple, nous conduisent à considérer un calcul de processus qui syntaxiquement hérite du modèle de Milner pour le π-calcul, mais qui sémantiquement prend comme primitive de base la diffusion. Ce choix facilite en même temps la preuve de certains résultats qui sont de nature essentiellement syntaxique (et qui peuvent être repris de résultats similaires dans le π-calcul).

Chapitre 3

Le $b\pi$-calcul

Dans ce chapitre nous introduisons le $b\pi$-calcul: nous présentons d'abord la syntaxe, suivi par la sémantique opérationnelle du modèle. Ensuite nous illustrons le langage par plusieurs exemples.

3.1 Syntaxe et Sémantique

Le $b\pi$-calcul est un calcul de processus mobiles qui communiquent seulement par diffusion.

Il est inspiré à la fois du CBS (Prasad, 1995) en considérant la diffusion comme seule primitive de communication, et du π-calcul (Milner et al., 1992), en gardant la même syntaxe et surtout la même manière d'exprimer la mobilité. Il diffère de CBS par le fait que les communications utilisent des canaux (ou ports) et les valeurs transmises sont aussi des canaux.

Les noms de canaux (qui sont des éléments d'un ensemble dénombrable Ch_b) sont les seules valeurs utilisées dans ce langage. Comme dans le π-calcul, ils servent en même temps comme support de communication et comme valeur transmise. Nous supposons aussi l'existence d'un ensemble $\mathcal{A}$ d'identificateurs de processus, chaque identificateur A ayant une arité $ar(A)$ donnée par la fonction $ar : \mathcal{A} \longrightarrow \mathbb{N}$. L'arité d'un identificateur A représente le nombre de paramètres (pour garder les notations simples, nous allons omettre les types des canaux).

Les processus sont définis par la grammaire présentée dans le Tableau 3.1. L 'ensemble des processus est noté par $\mathcal{P}_b$.

– les préfixes représentent des actions qu'un processus peut faire: $x(\tilde{y})$ est *la réception* des noms $\tilde{y}$ sur le canal x, $\bar{x}\tilde{y}$ est *l'émission* des noms $\tilde{y}$ sur le canal x, et τ représente

$$\mathcal{P}_b \ni p ::= \quad nil \mid \alpha.p \mid \nu x p \mid \langle x = y\rangle p,q \mid p_1 + p_2 \mid p_1 \parallel p_2 \mid A\langle\tilde{x}\rangle \mid (rec\ A\langle\tilde{x}\rangle.p)\langle\tilde{y}\rangle$$

où α appartient à l'ensemble des actions $\alpha ::= x(\tilde{y}) \mid \bar{x}\tilde{y} \mid \tau$ avec $\tilde{x},\tilde{y} \subseteq Ch_b$, $x, \in Ch_b$.

TAB. 3.1 – *Processus dans le $b\pi$-calcul*

une transition interne;

- nil est le processus qui a fini son exécution;
- $\alpha.p$ est le processus qui fait d'abord l'action α et se comporte ensuite comme p;
- νxp est la création d'un nouveau nom de canal x, différent de tous les noms déjà définis (et dont la portée initiale est le processus p);
- $\langle x = y\rangle p,q$ se comporte comme p si x et y sont identiques, et comme q autrement;
- $p_1 + p_2$ représente le non-déterminisme, il se comporte soit comme p_1 soit comme p_2, en fonction des interactions offertes par l'environnement (le non-déterminisme peut être "interne", quand p_1 et p_2 ont parmi les actions "immédiates" des actions communes);
- $p_1 \parallel p_2$ est la composition parallèle de p_1 et p_2;
- $(recA\langle\tilde{x}\rangle.p)\langle\tilde{y}\rangle$ est une définition récursive permettant de décrire des systèmes avec des comportements infinis ($\tilde{x}$ et $\tilde{y}$ doivent respecter l'arité de l'identificateur A, et $\tilde{x}$ doit contenir tous les noms qui apparaissent libres dans p). Dans le reste du manuscrit, nous considérons que dans toute expression récursive $rec\ A\langle\tilde{x}\rangle.p$, A apparaît sous la portée d'un préfixe dans p.

Si $\tilde{x} = (x_1, \ldots, x_n)$, nous notons $\nu x_1 \nu x_2 \ldots \nu x_n p$ par $\nu\tilde{x}p$ et par abus de notation, $\{\tilde{x}\} = \{x_1, \ldots, x_n\}$, $\tilde{x} \cup \{x_{n+1}\} = (x_1, \ldots, x_n, x_{n+1})$ si $x_{n+1} \notin \{\tilde{x}\}$ et

$$\tilde{x} \setminus y = \begin{cases} (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots x_n) & \text{si } y = x_i \text{ pour un certain } i \in \{1, \ldots, n\}, \\ \tilde{x} & \text{si } y \notin \{x_1, \ldots, x_n\} \end{cases} \tag{3.1}$$

(on suppose que dans $\tilde{x} \setminus y$, y apparaît au plus une fois parmi les noms x_i de $\tilde{x}$).

Quand on établit la correction d'un système parallèle distribué, on doit généralement garantir, que le système fonctionne "bien" dans n'importe quel environnement. Cela se traduit dans notre calcul, par prouver qu'un processus p satisfait une certaine propriété, indépendemment du contexte où le processus est placé. Un *contexte* est un terme contenant un seul "trou", tel que placer un nouveau processus dans le trou définit un terme valide. L'ensemble des contextes Con est définie formellement dans le Tableau 3.2.

Nous allons maintenant définir les notions de noms libres et liés.

$$C ::= \quad \bullet \mid \alpha.C \mid \nu x C \mid \langle x=y\rangle C{,}p \mid \langle x=y\rangle p{,}C \mid C+p \mid$$
$$p+C \mid C \parallel p \mid p \parallel C \mid (rec\ A\langle\tilde{x}\rangle.C)\langle\tilde{y}\rangle$$

où $p \in \mathcal{P}_b$ et $\alpha ::= x(\tilde{y}) \mid \bar{x}\tilde{y} \mid \tau$ avec $x \in Ch_b, \tilde{y} \subseteq Ch_b$.

TAB. 3.2 – *L'ensemble des contextes $\mathcal{C}on$*

(1.)	*si* $y \notin fn(p)$ *alors* $\nu x p \equiv_\alpha \nu y(p[y/x])$
(2.)	*si* $y \notin fn(p)$ *alors* $a(\tilde{x}).p \equiv_\alpha a(\tilde{y}).(p[\tilde{y}/\tilde{x}])$
(3.)	$p \equiv_\alpha p$
(4.)	si $p \equiv_\alpha q$ alors $q \equiv_\alpha p$
(5.)	si $p \equiv_\alpha q$ et $q \equiv_\alpha r$ alors $p \equiv_\alpha r$
(6.)	si $p \equiv_\alpha q$ et $C \in \mathcal{C}on$ alors $C[p] \equiv_\alpha C[q]$

TAB. 3.3 – *L'alpha-conversion*

$\nu\tilde{x}p$ et $y(\tilde{x}).p$, lient les apparitions des noms appartenant à $\tilde{x}$ dans p. Sinon , x est *libre* dans p. $bn(p)$ et $fn(p)$ représentent l'ensemble des noms liés, respectivement l'ensemble des noms libres dans p. L'ensemble des noms du processus p est noté par $n(p)$, et par définition $n(p) = fn(p) \cup bn(p)$).

Pour une fonction $f : Ch_b \longrightarrow Ch_b$, on va noter $prdom(f) \stackrel{not}{=} \{a \mid f(a) \neq a\}$, $prcod(f) \stackrel{not}{=} \{f(a) \mid a \in prdom(f)\}$ et $n(f) = prdom(f) \cup prcod(f)$.

Une substitution σ est une fonction $\sigma : Ch_b \longrightarrow Ch_b$ qui est egale à l'identité à l'exception d'un ensemble fini de noms (canaux), $|prdom(\sigma)| < \infty$.

Si σ est une substitution telle que $prdom(\sigma) = \{x_1, \ldots, x_n\}$, $\sigma(x_i) = z_i$, $\tilde{x} = (x_1, \ldots, x_n)$ et $\tilde{z} = (z_1, \ldots, z_n)$, alors on va utiliser aussi la notation $\sigma = [\tilde{z}/\tilde{x}]$.

Si $\tilde{x} = (x_1, \ldots, x_n)$, $\tilde{z} = (z_1, \ldots, z_n)$ et $\sigma = [\tilde{z}/\tilde{x}]$, alors $p\sigma = p[\tilde{z}/\tilde{x}]$ représente le terme p dans lequel pour tout $i \in \{1, \ldots, n\}$, les occurrences libres de x_i ont été remplacées simultanément par z_i (opération précédée par un renommage des noms liés dans p qui coincident avec un z_i, pour un certain $i \in \{1, \ldots, n\}$, renommage qui utilisera seulement des noms fraiches par rapport à $n(p)$ et $n(\sigma)$).

L'alpha-conversion (la relation de congruence entre termes induite par le renommage de noms liés) est définie dans le Tableau 3.3.

Nous introduisons la notion d'action, qui permet d'étiqueter les transitions dans la sémantique opérationnelle du $b\pi$-calcul.

Définition 4 Les actions, *(notées par α,β,...) sont définies par la grammaire suivante:*

$$\alpha ::= a\langle\tilde{x}\rangle \mid \nu\tilde{y}\bar{a}\tilde{x} \mid \tau \mid a:$$

où $a,x \in Ch_b$, $\tilde{x},\tilde{y} \subseteq Ch_b$. Une action est soit une réception, soit une émission (qui peut être accompagnée par l'exportation de certains noms), soit un pas interne, soit une action "ignorer". Dans $a\langle\tilde{x}\rangle$ et $\nu\tilde{y}\bar{a}\tilde{x}$, a est le sujet de la communication et $\tilde{x}$ est son objet (notés respectivement pour une action α par $sub(\alpha)$ et $obj(\alpha)$). Par extension, $n(\alpha)$ ($fn(\alpha)$, $bn(\alpha)$) représentent l'ensemble des noms (respectivement l'ensemble des noms libres, et l'ensemble des noms liés) utilisés dans l'action α ($fn(\tau) = \emptyset$, $fn(a\langle\tilde{x}\rangle) = \{a\} \cup \tilde{x}$, $fn(\nu\tilde{y}\bar{a}\tilde{x}) = \{a\} \cup \tilde{x} \setminus \tilde{y}$, $fn(a:) = \{a\}$, $bn(\tau) = \emptyset$, $bn(a\langle\tilde{x}\rangle) = \emptyset$, $bn(\nu\tilde{y}\bar{a}\tilde{x}) = \tilde{y}$, $bn(a:) = \emptyset$, $n(\alpha) = fn(\alpha) \cup bn(\alpha)$).

On utilisera souvent la notation $\bar{a}\tilde{x}$ pour $\nu\tilde{y}\bar{a}\tilde{x}$ quand l'ensemble de noms exportés $\tilde{y}$ est vide, $\tilde{y} = (\,)$.

Etant donnée une substitution σ, la notation $\alpha\sigma$ sera utilisée aussi pour une action α avec la même signification que pour les processus. Par contre pour les noms ou les tuples de noms x, $\tilde{x}$ on va utiliser les notations pré-fixées $\sigma(x)$ ou $\sigma(\tilde{x})$.

Nous notons $type(\alpha) = output$, si α est une émission (libre ou liée), et $type(\alpha) = input$ si α est une réception.

La sémantique opérationnelle du $b\pi$-calcul est présentée comme un système traditionnel de transitions (étiquetées par des actions) entre les processus. Avant de la préciser, nous définissons (comme pour le CBS, (Prasad, 1995) et (Hennessy et Rathke, 1995)) une relation "ignore" $\longrightarrow \subseteq \mathcal{P}_b \times Ch_b$ ($(p,a) \in \longrightarrow$ sera notée $p \xrightarrow{a:}$) qui peut se traduire par "p ne peut pas entendre sur le canal a" (voir Tableau 3.4).

(1) $\dfrac{}{nil \xrightarrow{a:}}$	(2) $\dfrac{}{\tau.p \xrightarrow{a:}}$	(3) $\dfrac{}{\bar{b}\tilde{y}.p \xrightarrow{a:}}$
(4) $\dfrac{b \neq a}{b(x).p \xrightarrow{a:}}$	(5) $\dfrac{p \xrightarrow{a:} \vee\ x=a}{\nu x p \xrightarrow{a:}}$	(6) $\dfrac{p_1 \xrightarrow{a:} \wedge p_2 \xrightarrow{a:}}{p_1 + p_2 \xrightarrow{a:}}$
(7) $\dfrac{p_1 \xrightarrow{a:}}{\langle x=x\rangle p_1, p_2 \xrightarrow{a:}}$	(8) $\dfrac{x \neq y \wedge\ p_2 \xrightarrow{a:}}{\langle x=y\rangle p_1, p_2 \xrightarrow{a:}}$	
(9) $\dfrac{p_1 \xrightarrow{a:} \wedge\ p_2 \xrightarrow{a:}}{p_1 \| p_2 \xrightarrow{a:}}$	(10) $\dfrac{p[(rec\ X\langle\tilde{x}\rangle.p)/X, \tilde{y}/\tilde{x}] \xrightarrow{a:}}{(rec\ X\langle\tilde{x}\rangle.p)\langle\tilde{y}\rangle \xrightarrow{a:}}$	

TAB. 3.4 – *La relation "ignore"*

Intuitivement, un processus ignore les communications faites sur des canaux qu'il

$$(1)\frac{}{\tau.p\xrightarrow{\tau}p} \qquad (2)\frac{}{a(\tilde{x}).p\xrightarrow{a\langle\tilde{z}\rangle}p[\tilde{z}/\tilde{x}]} \qquad (3)\frac{}{\bar{a}\tilde{x}.p\xrightarrow{\bar{a}\tilde{x}}p}$$

$$(4)\frac{p\xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}}p' \wedge z\in\tilde{x}\backslash\{a,\tilde{y}\} \wedge w\notin fn(\nu zp')}{\nu zp\xrightarrow{\nu w\nu\tilde{y}\bar{a}(\tilde{x}[w/z])}p'[w/z]} \qquad (5)\frac{p\xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}}p'}{\nu ap\xrightarrow{\tau}\nu a\nu\tilde{y}p'} \qquad (6)\frac{p\xrightarrow{\alpha}p' \wedge x\notin n(\alpha)}{\nu xp\xrightarrow{\alpha}\nu xp'}$$

$$(7)\frac{p_1\xrightarrow{\alpha}p' \wedge bn(\alpha)\cap fn(p_2)=\emptyset}{p_1+p_2\xrightarrow{\alpha}p'} \qquad (8)\frac{p_1\xrightarrow{\alpha}p' \wedge bn(\alpha)\cap fn(p_2)=\emptyset}{\langle x=x\rangle p_1,p_2\xrightarrow{\alpha}p'}$$

$$(9)\frac{x\neq y \wedge p_2\xrightarrow{\alpha}p' \wedge bn(\alpha)\cap fn(p_2)=\emptyset}{\langle x=y\rangle p_1,p_2\xrightarrow{\alpha}p'} \qquad (10)\frac{p[(rec\ X\langle\tilde{x}\rangle.p)/X,\tilde{y}/\tilde{x}]\xrightarrow{\alpha}p'}{(rec\ X\langle\tilde{x}\rangle.p)\langle\tilde{y}\rangle\xrightarrow{\alpha}p'} \qquad (11)\frac{p_1\xrightarrow{a\langle\tilde{x}\rangle}p_1' \wedge p_2\xrightarrow{a\langle\tilde{x}\rangle}p_2'}{p_1\|p_2\xrightarrow{a\langle\tilde{x}\rangle}p_1'\|p_2'}$$

$$(12)\frac{p_1\xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}}p_1' \wedge p_2\xrightarrow{a\langle\tilde{x}\rangle}p_2' \wedge \tilde{y}\cap fn(p_2)=\emptyset}{p_1\|p_2\xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}}p_1'\|p_2'} \qquad (13)\frac{p_1\xrightarrow{\alpha}p_1' \wedge bn(\alpha)\cap fn(p_2)=\emptyset \wedge p_2\xrightarrow{sub(\alpha):}}{p_1\|p_2\xrightarrow{\alpha}p_1'\|p_2}$$

TAB. 3.5 – *La sémantique opérationnelle du bπ-calcul*

n'écoute pas.

- nil, $\tau.p$ or $\bar{b}\tilde{y}.p$ ne sont à l'écoute d'aucun canal (règles (1), (2) et (3));
- un processus qui écoute seulement sur le canal b, ignore tout ce qui se passe sur les autres canaux a (règle (4));
- dans la règle (5), la condition $x = a$ exprime la possibilité que le nom a n'apparaîsse pas libre dans p;
- les autres règles ((6) à (10)) suivent la structure du terme.

Pour faciliter la présentation, nous étendons sub, en prenant par convention $sub(\tau) = obj(\tau) = \tau$, et $p \xrightarrow{\tau:}$ pour tout processus p.

Définition 5 Système transitionnel *La sémantique opérationnelle de bπ-calcul est définie comme un système transitionnel étiqueté sur l'ensemble $\mathcal{P}_b$ de processus. $p \xrightarrow{\alpha} p'$ signifie que le processus p est capable de faire l'action α et d'évoluer ensuite dans p'. La sémantique opérationnelle est donnée dans le Tableau 3.5 (nous avons omis les versions symétriques des règles* (7), (12) *et* (13)*).*

Le mode d'interaction de base est la diffusion sans tampon; tout message a exactement un émetteur et zéro ou plusieurs récepteurs. Contrairement au π-calcul, les émissions ne sont pas bloquantes, il n'y a pas besoin d'un processus receveur. Un processus diffuse un message sur un canal, et les autres processus soit le reçoivent, soit l'ignorent en fonction du fait qu'ils étaient ou non à l'écoute sur le canal qui sert de support pour la communication. Un processus qui "écoute" sur un canal a, ne peut ignorer aucune valeur diffusée sur ce canal.

- les règles (1) à (3) ont la même signification que dans le π - calcul: décrivent la première action qu'un processus peut faire.
- dans la règle (4), un nom de canal local envoyé sur un autre canal produit une émission accompagnée d'une extension de portée. La condition $w \notin fn(\nu z p')$ assure qu'un processus ne peut pas envoyer comme nom de canal nouveaun (local), un canal qu'il possedait déjà comme nom libre.
- la règle (5) n'existe pas dans le π-calcul. Si a est un canal local à P, toutes les communications faites sur a sont cachées pour l'environnement extérieur (celà ressemble à la manière dont les actions sont cachées en $LOTOS$); de plus, cette règle établit la portée des noms qui étaient exportés par l'émission faite sur le canal a: la nouvelle portée des noms $\tilde{y}$ sera l'ensemble des procesus qui étaient à l'attente sur le canal a.
- les règles (6) à (10) sont similaires aux règles correspondantes du π-calcul (voir le Tableau 2.8); la condition $bn(\alpha) \cap fn(p_2) = \emptyset$ assure qu'un processus ne peut pas envoyer comme nom de canal local, un canal qu'il possedait déjà comme nom libre.
- (11) et (12) sont spécifiques à la diffusion: le même message peut être reçu par plusieurs processus à la fois. L'extension de la portée d'un nom est plus complexe que dans le π - calcul à cause du fait que plusieurs processus ("auditeurs") peuvent apprendre l'existence d'un canal local dans une seule communication.
- dans (13) un processus qui n'est pas à l'écoute sur un canal a reste inchangé pendant une communication faite sur ce même canal; de plus τ n'est pas audible.

Pour simplifier les expressions des processus et diminuer le nombre de parenthèses, nous considérons entre les opérateurs, les prioritées suivantes:

$$. > \nu > \langle\,\rangle > \| > + > rec.$$

De plus nous omettons le nil à la fin des processus (par exemple $\bar{x}y.z(t)$ sera l'abbréviation de $\bar{x}y.z(t).nil$).

Pour illustrer la sémantique opérationnelle, nous donnons un exemple de dérivation d'une transition d'un proccessus.

Exemple 6 *Soit* $p \stackrel{def}{=} \left[\nu c(\bar{a}c + b(x).\bar{x}c) \parallel a(y).\bar{y}c\right] \parallel (\bar{b}d + a(z).\bar{z}t)$.

p est un processus qui peut faire une émission liée (le "message" envoyé est un nom de canal local) sur le canal a. Nous utilisons les règles du Tableau 3.5 pour déduire la continuation de p après avoir fait cet émission.

En utilisant succesivement les règles (3), (7), (4), (2) et (12) on obtient:

$$\frac{}{\bar{a}c \xrightarrow{\bar{a}c} nil}(3) \tag{3.2}$$

$$\frac{\bar{a}c \xrightarrow{\bar{a}c} nil}{\bar{a}c + b(x).\bar{x}c \xrightarrow{\bar{a}c} nil}(7) \tag{3.3}$$

$$\frac{\bar{a}c + b(x).\bar{x}c \xrightarrow{\bar{a}c} nil \ \wedge \ c \in \{c\} \ \wedge \ f \notin fn(\nu c\ nil)}{\nu c(\bar{a}c + b(x).\bar{x}c) \xrightarrow{\nu f \bar{a} f} nil}(4) \tag{3.4}$$

$$\frac{}{a(y).\bar{y}c \xrightarrow{a\langle f\rangle} \bar{f}c}(2) \tag{3.5}$$

$$\frac{\nu c(\bar{a}c + b(x).\bar{x}c) \xrightarrow{\nu f \bar{a} f} nil \quad \wedge \quad a(y).\bar{y}c \xrightarrow{a\langle f\rangle} \bar{f}c}{\nu c(\bar{a}c + b(x).\bar{x}c) \parallel a(y).\bar{y}c \xrightarrow{\nu f \bar{a} f} nil \parallel \bar{f}c}(12) \tag{3.6}$$

En utilisant successivement les règles (2) et (7) on obtient:

$$\frac{}{a(z).\bar{z}t \xrightarrow{a\langle f\rangle} f(t)}(2) \tag{3.7}$$

$$\frac{a(z).\bar{z}t \xrightarrow{a\langle f\rangle} f(t)}{\bar{b}d + a(z).\bar{z}t \xrightarrow{a\langle f\rangle} \bar{f}t}(7) \tag{3.8}$$

En appliquant la règle (12) aux descendants des équations 3.6 et 3.8 on obtient

$$\frac{\nu c(\bar{a}c + b(x).\bar{x}c) \parallel a(y).\bar{y}c \xrightarrow{\nu f \bar{a} f} nil \parallel \bar{f}c \quad \wedge \quad a(z).\bar{z}t + \bar{b}d \xrightarrow{a\langle f\rangle} \bar{f}t}{p = \left[\nu c(\bar{a}c + b(x).\bar{x}c) \parallel a(y).\bar{y}c\right] \parallel (\bar{b}d + a(z).\bar{z}t) \xrightarrow{\nu f \bar{a} f} (nil \parallel \bar{f}c) \parallel \bar{f}t}(12) \tag{3.9}$$

□ *Exemple* 6

Dans la suite, nous utilisons les notations suivantes:

- $\overset{\epsilon}{\Longrightarrow} \overset{def}{=} (\overset{\tau}{\longrightarrow})^*$,
- $\overset{\alpha}{\Longrightarrow} \overset{def}{=} \overset{\epsilon}{\Longrightarrow}\overset{\alpha}{\longrightarrow}\overset{\epsilon}{\Longrightarrow}$, si $\alpha \neq \tau$,
- $\overset{\oslash}{\longrightarrow} \overset{def}{=} \bigcup\{ \overset{\alpha}{\longrightarrow} | type(\alpha) = output\ ou\ \alpha = \tau\}$,
- $\overset{\oslash}{\Longrightarrow} \overset{def}{=} (\overset{\oslash}{\longrightarrow})^*$.

Comme dans (Hennessy et Rathke, 1995), nous utilisons aussi, $p \xrightarrow{\alpha:} p$ à la place de $p \xrightarrow{a:}$ si $sub(\alpha) = a$, et nous notons $p \xrightarrow{a\langle\tilde{b}\rangle?} p'$ si $p \xrightarrow{a\langle\tilde{b}\rangle} p'$ ou $p \xrightarrow{a\langle\tilde{b}\rangle:} p'$.

Nous utilisons $\bar{a}(\tilde{x})$ comme notation alternative de l'action $\nu\tilde{x}\bar{a}\tilde{x}$.

Les lemmes suivants établissent la manière dont évolue l'ensemble de noms libres d'un processus au cours de son exécution. Ces lemme seront utils dans les chapitres suivants.

Lemme 7 *Si $p \equiv_\alpha q$, alors $fn(p) = fn(q)$.*

Preuve

Par induction sur l'inférence de $p \equiv_\alpha q$.

□ *Lemme* 7

Lemme 8

1. *Si $p \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} p'$, alors $fn(p') \subseteq fn(p) \cup \{\tilde{y}\}$ et $(\tilde{x} \setminus \tilde{y}) \subseteq fn(p)$.*
2. *Si $p \xrightarrow{a\langle\tilde{z}\rangle} p'$, alors $fn(p') \subseteq fn(p) \cup \{\tilde{z}\}$.*
3. *Si $p \xrightarrow{\tau} p'$, alors $fn(p') \subseteq fn(p)$.*

Preuve

La preuve est faite par induction sur la dérivation de la transition $p \xrightarrow{\alpha} p'$, simultanément pour les trois assertions (voir l'annexe A).

□ *Lemme* 8

Corollaire 9 *Si $p \stackrel{\epsilon}{\Longrightarrow} p'$, alors $fn(p') \subseteq fn(p)$.*

Preuve

Le corollaire est une conséquence directe par induction sur le nombre de transitions dans $p \stackrel{\epsilon}{\Longrightarrow} p'$.

□ *Corollaire* 9

Le lemme suivant suit directement des règles du Tableau 3.5.

Lemme 10 *Si $p \xrightarrow{\alpha} q$, alors $bn(\alpha) \cap fn(p) = \emptyset$.*

Preuve

Par induction sur l' inférence de $p \xrightarrow{\alpha} q$.

□ *Lemme* 10

Les lemmes suivants présentent les relations qui existent entre les exécutions d'un processus p et du processus obtenu par un renommage des noms $p\sigma$.

Lemme 11 *Si $p \equiv_\alpha q$, alors $p\sigma \equiv_\alpha q\sigma$.*

Preuve

Par induction sur l'inférence de $p \equiv_\alpha q$.

□ *Lemme* 11

Lemme 12 *Si* $p\sigma \equiv_\alpha q'$*, alors* $\exists q$ *tel que* $q' = q\sigma$ *et* $p \equiv_\alpha q$*.*

Preuve

Par induction sur l'inférence de $p\sigma \equiv_\alpha q'$.

$\square$ *Lemme* 12

Lemme 13 *Soit* $p \xrightarrow{\alpha} q$ *une transition de* p*, et* $\beta \equiv_\alpha \alpha$ *une action telle que* $bn(\beta) \cap n(p) = \emptyset$*. Alors* $p \xrightarrow{\beta} q[bn(\beta)/bn(\alpha)]$ *par une dérivation de longueur plus petite ou egale à celle de* $p \xrightarrow{\alpha} q$*.*

Preuve

Par induction sur l'inférence de $p \xrightarrow{\alpha} q$ (voir l'annexe A).

$\square$ *Lemme* 13

Si σ est une substitution et $M \subseteq Ch_b$ un ensmble de noms, on va noter $\sigma \setminus M$, la substitution qui coincide à σ pour tous les noms $x \in M$, et est egale à l'identité sur $Ch_b \setminus M$ ($(\sigma \setminus M)(x) = \sigma(x)$ si $x \in M$ et $(\sigma \setminus M)(x) = x$ si $x \notin M$). Nous appelons ρ comme etant une *extension* de σ si $\rho(x) = \sigma(x)$ pour tout $x \in prdom(\sigma)$.

Lemme 14 *Soit* $p \xrightarrow{\alpha} q$ *une transition de* p*, et* σ *une substitution injective telle que* $(n(\sigma) \cup n(p\sigma)) \cap bn(\alpha) = \emptyset$ *et* $prcod(\sigma \setminus fn(p)) \cap fn(p) = \emptyset$*. Alors* $p\sigma \xrightarrow{\alpha\sigma}\equiv_\alpha q\sigma$ *par une dérivation de longueur plus petite ou egale à celle de* $p \xrightarrow{\alpha} q$*.*

Preuve

Par induction sur l'inférence de $p \xrightarrow{\alpha} q$ (voir l'annexe A).

$\square$ *Lemme* 14

Lemme 15 *Soit* σ *une substitution injective telle que* $prcod(\sigma \setminus fn(p)) \cap fn(p) = \emptyset$*.*

Si $p\sigma \xrightarrow{\alpha} q'$ *et* $(n(\sigma) \cup fn(p)) \cap bn(\alpha) = \emptyset$*, alors ils existent une action* β*, un processus* q *et une substitution* ρ *extension injective de* σ*, tels que* $\alpha = \beta\rho$*,* $q' \equiv_\alpha q\rho$*,* $\rho \setminus fn(p) = \sigma \setminus fn(p)$*,* $(n(\rho) \cup fn(p)) \cap bn(\beta) = \emptyset$ *et* $p \xrightarrow{\beta} q$ *par une dérivation de longueur plus petite ou egale à celle de* $p\sigma \xrightarrow{\alpha} q'$*.*

Preuve

Par induction sur l'inférence de $p\sigma \xrightarrow{\alpha} q'$ (voir l'annexe A).

$\square$ *Lemme* 15

3.2 Exemples

Ces exemples ont pour but d'illustrer l'interet d'un langage formel qui combine la diffusion et la mobilité.

Dans les exemples, pour simplifier la présentation, nous supposons un langage qui contient aussi des structures de données classiques (comme les entiers). Ces structures de données peuvent être codées dans le $b\pi$-calcul, suivant des techniques similaires à celles du π-calcul ((Pierce, 1996), (Milner et al., 1992)). On utilise aussi des opérateurs sur ces structures de données tels que les comparaisons, ou des structures de controls comme le "if then else".

Exemple 16 Un algorithme de tri distribué

Nous présentons un algorithme de tri distribué (inspiré de (Prasad, 1995)) qui sert d'illustration à l'utilité des noms locaux.

Il y a trois types de processus: une instance de *Bottom* et respectivement de *Top* qui mémorisent le plus petit et rspectivement le plus grand élément introduit (par une diffusion sur le canal *in*.). Les processus de la forme $InCell\langle m,n,a,b\rangle$ mémorisent un interval de la forme $(m,n]$.

Pour des raisons de simplicité, nous omettons les canaux *in*, *out*, *done* parmi les paramètres de *Sorter*, *Bottom*, *Top*, ou *InCell*. Nous supposons que tous les entiers triés sont différants.

Un signal sur le canal *done* signifie qu'il n'y a plus d'entiers à trier. De plus, il entraîne une liste d'émissions (une émission par processus "en exécution") sur le canal *out*. *Sorter* est un processus qui attend un entier sur le canal *in* et génère deux processus "fils", *Bottom* et *Top*.

$$Sorter \stackrel{def}{=} done.nil + in(x).(\nu c(Bottom\langle x,c\rangle \parallel Top\langle x,c\rangle))$$

$Bottom\langle n,a\rangle$ mèmorise le plus petit entier n de la liste de entiers déjà "rentrée". Pour chaque nouvel entier envoyé sur le canal *in* stocké dans x, s'il s'agit d'une valeur plus petite que a, il faut réactualiser le plus petit élément (par la génération du "fils" $Bottom\langle x,c\rangle$), et d'un processus "buffer" "$InCell\langle x,n,c,a\rangle$" qui contient l'ancienne valeur n. Le nom local c est utilisé pour "se rappeler" l'ordre $x \leq n$. Si l'entier reçu est plus grand que la valeur déjà possédée n, on ne fait rien.

$$Bottom\langle n,a\rangle \stackrel{def}{=} done.\overline{out}n.\bar{a}.nil + in(x).$$

$$(\text{ if } x \leq n \text{ then } \nu c(Bottom\langle x,c\rangle \parallel InCell\langle x,n,c,a\rangle) \text{ else } Bottom\langle n,a\rangle)$$

Top a le même rôle que *Bottom*, mais pour le plus grand élément.

$$Top\langle n,a\rangle \stackrel{def}{=} a.nil + in(x).$$

$$(\text{ if } n < x \text{ then } \nu c(Top\langle x,c\rangle \parallel InCell\langle n,x,a,c\rangle) \text{ else } Top\langle n,a\rangle)$$

Dans $InCell\langle m,n,a,b\rangle$, n est la valeur de ce "buffer", et m est la valeur précédente (en ordre croissant). a et b jouent le rôle de bornes: on pourra émettre la valeur n sur le canal out, seulement après avoir reçu un signal sur le canal a; ensuite, l'émission d'un signal sur le canal b va "libérer" la valeur suivante (en ordre croissant). $InCell\langle m,n,a,b\rangle$ attend des entiers sur le canal in. S'il reçoit un entier x tel que $m < x$ et $x \leq n$, alors il génère deux fils $InCell\langle m,x,a,c\rangle$ et $InCell\langle x,n,c,b\rangle$, sinon, il ne fait rien.

$$InCell\langle m,n,a,b\rangle \stackrel{def}{=} done.a.\overline{out}n.\bar{b}.nil + in(x).(\text{ if } (m < x \text{ and } x \leq n)$$

$$\text{then } \nu c(InCell\langle m,x,a,c\rangle \parallel InCell\langle x,n,c,b\rangle) \text{ else } InCell\langle m,n,a,b\rangle)$$

Exemple 17 Un algorithme distribué pour la détection des cycles

Nous présentons un algorithme distribué pour la détection des cycles dans un graphe orienté.

Nous rappelons du Chapitre 1 les principaux éléments de l'agorithme.

Supposons que les arêtes du graphe sont distribués à travers plusieurs objets (acteurs) et que les objets continuent à recevoir des nouvelles arêtes. Un possible algorithme est le suivant: chaque objet o, pour chaque arête (a,b) qu'il posséde, génére une clé unique (un jeton) $k^o_{(a,b)}$ qu'il envoie sous la forme $(k^o_{(a,b)},b)$ à "tous ceux" qui possédent une arête de la forme (b,x) avec x quelconque. De plus, un objet o', pour chaque message (k',x) qu'il reçoit et pour chaque arête de la forme (x,y) qu'il posséde, envoie le message (k',y) à "tous ceux" qui possédent une arête de la forme (y,z) avec z quelconque (les clés sont propagés à travers les chemins du graphe). Un cycle est détecté si un objet o reçoit un message de la forme $(k^o_{(a,b)},a)$ (le jeton a parcouru un cycle).

Dans notre implantations de l'algorithme dans le $b\pi$ - calcul, pour chaque arête (a,b) on lance un processus $Edge\_manager\langle out,a,b\rangle$. Le graphe est construit dynamiquement en envoyant des arêtes sur le canal in au processus $Detector\langle in,out\rangle$.

Detector est un processus qui attend de nouvelles arêtes d'un graphe sur le canal in, et génère un nouveau "edge manager" *Edge_manager* pour chaque paire de noms reçue (source et destination de l'arête).

$$Detector\langle in,out\rangle \stackrel{def}{=}$$

$$in(x,y).(Detector\langle in,out\rangle \parallel Edge\_manager\langle out,x,y\rangle)$$

Edge_manager diffuse un jeton personnel u (utilisant le mécanisme de génération de noms). Ensuite, pour chaque jeton reçu w, s'il s'agit de son propre jeton u, alors un cycle a été détecté et un signal est émis sur out, sinon, il propage le jeton w plus loin (à travers les chemins du graphe) (on utilise les canaux a et b, qui représentent une arête (a,b) dans le graphe). Donc, tout jeton reçu sur a, et différent de u est transmis sur b.

$$Edge\_manager\langle out,a,b\rangle \stackrel{def}{=}$$

$$\nu u((recY\langle b,u\rangle.\{\bar{b}u.Y\langle b,u\rangle\})\langle b,u\rangle \parallel (recX\langle out,a,b,u\rangle.\{a(w).$$

$$(\langle u = w\rangle \bar{out}.nil,(\bar{b}w.nil \parallel X\langle out,a,b,u\rangle))\})\langle out,a,b,u\rangle)$$

La diffusion est naturelle dans cet exemple: pour chaque arête (a,b), le jeton envoyé par $Edge\_manager\langle out,a,b\rangle$ doit être reçu par tous les processus $Edge\_manager\langle out,b,x\rangle$ qui gérent des arêtes (b,x) incidentes à b. De plus, le comportement d'un gestionaire d'arête $Edge\_manager\langle out,a,b\rangle$ ne depend pas de la structure de graphe, c.à.d. du nombre des arêtes incidentes à b.

Exemple 18 Détecter des incohérences dans un système transactionnel

Nous étendons l'exemple précédent (la détection des cycles dans un graphe) pour donner une implantation dans le $b\pi$-calcul d'un algorithme distribué pour la détection des incohérences dans une base de données distribuée (notre implantation est inspirée de (Bayerdorffer, 1994)).

Dans une base de données distribuée, il y a plusieurs copies de chaque donnée, les copies étant localisées sur des sites distincts du système. Nous supposons qu'à cause des pannes, la base des données peut devenir partitionée (partitions p_j où $j = 1, \ldots ,n$).

Nous permettons aux transactions de continuer à s'exécuter, mais une fois le réseau rétabli, (simulé dans notre implantation par une diffusion sur le canal "$unif$"), nous devons vérifier l'existence d'incohérences. L'idée est de construire un graphe de précédence qui capture l'ordre temporel partiel qui existe entre les transactions. La base de données est cohérente si et seulement si le graphe de précédence ne contient pas de cycles. Les transactions sont les sommets du graphe. Une arête $< t,p > \rightarrow < t_1,p_1 >$ indique que la transaction t apparaît avant la transaction t_1 (p,p_1 sont les partitions où les transactions sont exécutées).

Une telle arête existe si et seulement si une des conditions suivantes est remplie:

1. t lit une donnée i qui est écrite plus tard par t_1 et $p = p_1$;

2. t écrit une donnée i qui est lue ou écrite plus tard par t_1 et $p = p_1$;
3. t lit une donnée i qui est écrite par t_1 et $p \neq p_1$.

Supposons qu'il y a k données, chaque donnée j ayant $n(j)$ copies, et $p_{j_l} \in \{p_1, \ldots, p_{n(j)}\}$ est la partition correspondant à la copie l de la donnée j.

La base des données peut être simulée par:

$$Database \stackrel{def}{=}$$

$$\prod_{j=1,k} [\prod_{l=1,n(j)} Item\langle j_1, j_2, p_{j_l}, unif, Val\rangle]$$

où j_1 et j_2 sont des canaux correspondant à la donnée j.

Un *gestionnaire de données* attend des transactions; pour chaque nouvelle transaction, il lance un nouveau *gestionnaire de transactions*, et sert l'utilisateur qui avait fait la requête. Une transaction pour une donnée i, est simulée par une émission sur le canal i_1, et elle contient un identificateur de transaction t_1, le type (lire ou écrire), la partition affectée, le canal de retour, et une valeur (qui a une signification seulement pour les écritures).

$Item\langle i_1,i_2,p,unif,Val,w\rangle \stackrel{def}{=}$
$unif(p_1).Item\langle i_1,i_2,p_1,unif,Val\rangle +$
$i_1(t_1,type,p_1,req,V).\langle p_1 = p\rangle$
$\quad \{\langle type = w\rangle$
$\quad\quad [Item\langle i_1,i_2,p,unif,V,w\rangle \parallel Tr\_Man_w\langle i_1,i_2,p,unif,t_1\rangle],$
$\quad\quad [Item\langle i_1,i_2,p,unif,Val,w\rangle \parallel Tr\_Man_r\langle i_1,i_2,p,unif,t_1\rangle \parallel \overline{req}Val + req(V)]\},$
$\quad Item\langle i_1,i_2,p,unif,Val,w\rangle$

Un gestionnaire de transactions, lance un nouveau gestionnaire d'arête pour chaque nouvelle transaction qui affecte la donnée située dans la même partition (cas 1. ou 2.).

$Tr\_Man_w\langle i_1,i_2,p,unif,t\rangle \stackrel{def}{=}$
$i_1(t_1,type,p_1,req,V).\{\langle p_1 = p\rangle$
$\quad [Tr\_Man_w\langle i_1,i_2,p,unif,t\rangle \parallel unif(p).Edge\_manager\langle error,t,t_1\rangle],$
$\quad Tr\_Man_w\langle i_1,i_2,p,unif,t\rangle\} +$
$unif(p_1).STr\_Man_w\langle i_2,p,t,w\rangle$

$Tr\_Man_r\langle i_1,i_2,p,unif,t\rangle \stackrel{def}{=}$
$i_1(t_1,type,p_1,req,V).\{\langle p_1 = p\rangle$
$\quad [Tr\_Man_r\langle i_1,i_2,p,unif,t\rangle \parallel \langle type = w\rangle unif(p).Edge\_manager\langle error,t,t_1\rangle,nil],$
$\quad Tr\_Man_r\langle i_1,i_2,p,unif,t\rangle\} +$
$unif(p_1).STr\_Man_r\langle i_2,p,t,r\rangle$

Le test $\langle type = w\rangle unif(p).Edge\_manager\langle error,t,t_1\rangle,nil$ signifie que entre deux lectures, il n'y a pas d'ordre de précedence.

Une fois que le réseau est rétabli, les gestionnaires des transactions changent leurs comportements, essayent de détecter des arêtes du type 3. De plus, si il y a deux transactions qui écrivent la même donnée, dans deux partitions différentes, alors une erreur est détectée immediatement.

$STr\_Man_w\langle i_2,p,t\rangle \stackrel{def}{=}$
$i_2(t_1,type,p_1).\{\langle p_1 = p\rangle$
$\quad STr\_Man_w\langle i_2,p,t\rangle,$
$\quad \langle type = w\rangle\overline{error},[STr\_Man_w\langle i_2,p,t\rangle \parallel Edge\_manager\langle error,t_1,t\rangle]\}+$
$\bar{i}_2[t,w,p].STr\_Man_w\langle i_2,p,t\rangle$

$STr\_Man_r\langle i_2,p,t\rangle \stackrel{def}{=}$
$i_2(t_1,type,p_1).\{\langle p_1 = p\rangle$
$\quad STr\_Man_r\langle i_2,p,t\rangle,$
$\quad STr\_Man_r\langle i_2,p,t\rangle \parallel \langle type = r\rangle nil,Edge\_manager\langle error,t,t_1\rangle]\}+$
$\bar{i}_2[t,r,p].STr\_Man_r\langle i_2,p,t\rangle$

Notons que cet exemple utilise tout le pouvoir d'expression de notre calcul. La même donnée peut être répliquée (pour des raisons de tolérance aux pannes et d'efficacité); une transaction t peut affecter plusieurs données ; donc pour cette classe d'applications, la diffusion est une primitive tout à fait naturelle de communication. En même temps, la capacité d'envoyer et de recevoir des noms des canaux sur les canaux est utilisée par le gestionnaire de données pour générer des nouveaux gestionnaires de transactions et par les gestionnaires des transactions pour générer des nouveaux gestionnaires d'arête correspondants à l'identificateur reçu.

Exemple 19 Sémantique des primitives de communication de groupe

Le $b\pi$-calcul fournit un cadre de spécification et d'analyse des systèmes qui interagissent par un mécanisme de diffusion (broadcast ou multicast) combiné avec la mobilité des processus (noms, adresses). Nous prenons ici comme exemple des programmes qui utilisent des primitives de communication comme dans les bibliothèques PVM (Geist et al., 1994). PVM est un environnement de programmation et d'exécution qui permet à un réseau de machines hétérogène d'être utilisé comme une seule machine parallèle (la *machine virtuelle*). Ainsi, des problèmes de très grande taille peuvent être résolus par l'utilisation de la puissance réunie de plusieurs ordinateurs. PVM fournit des fonctions pour lancer automatiquement des tâches sur la machine virtuelle; PVM permet aussi aux tâches de communiquer (par des communications point à point ou par des communications de groupe) et de se synchroniser entre elles. L'intérêt du $b\pi$ - calcul est la simulation simple des primitives de communication de groupe, ce qui est difficile à exprimer dans une algèbre de processus basée sur des communications point à point (voir par exemple (Ene et Muntean, 1999) ou le Chapitre 6). En plus, même dans CBS (Prasad, 1995), la manière dont on peut décrire les primitives associées aux groupes dynamiques (les tâches peuvent à tout moment rejoindre - "joingroup" - ou quitter - "leavegroup" - un groupe, une tâche peut appartenir à plusieurs groupes en même temps) est peu évidente.

Nous présentons seulement quelques primitives de communication spécifiques aux applications distribuées (pour l'interprétation des caractéristiques impératives des langages dans les algèbres de processus voir par exemple (Rockl et Sangiorgi, 1999)):

$$\begin{array}{rcl} I & ::= & send(a,m) \mid bcast(g,m) \mid x = receive() \mid x = newgroup() \\ & & joingroup(g) \mid leavegroup(g) \mid x = spawn(Q) \\ P & ::= & STOP \mid I;P \end{array}$$

Un processus (ou une tâche) est une séquence d'instructions (actions). Une action est soit une émission d'un message m (adressé à un autre processus a ou groupe de processus g), soit une réception (le message s'il existe est reçu d'un tampon propre) d'un message (qui ensuite est récupéré dans la variable x), soit une création d'un nouveau groupe g, soit une adhésion à un groupe g, soit un départ d'un groupe g soit le lancement d'un nouveau fils Q. Une fois qu'un processus est devenu membre d'un groupe g, il reçoit tous les messages envoyés à ce groupe. Les communications sont asynchrones : les émissions ne sont pas bloquantes (les messages étant gardés dans les tampons des destinataires, tampons qui sont supposés être infinis). Pour des raisons de simplicité, nous supposons qu'il n'y a aucune garantie sur l'ordre dans lequel les messages sont reçus.

Dans notre implantation de PVM dans le $b\pi$-calcul, $\{P\}_a$ est la traduction d'un processus P d'adresse (pid) a:

$$\{P\}_a \stackrel{def}{=} \nu r_a \nu k_a (Pool\langle a,r_a,k_a\rangle \parallel [\![P]\!]_{r_a,\emptyset})$$

Dans $[\![P]\!]_{r,M}$, P est l'état actuel du processus de pid a (état "complété" par un nombre de processus tampons $Pool\langle\ldots\rangle$), r est le canal par lequel le processus récupère les éventuels messages recus, et $M = \{(g_1,k_{g_1}),\ldots,(g_n,k_{g_n})\}$ est un ensemble de paires, où $\{g_1,\ldots,g_n\}$ représente l'ensemble des groupes dont le processus est membre. $Pool\langle a,r,k\rangle$ est un "processus tampon" qui "reçoit" les messages envoyés à l'adresse a:

$$Pool\langle a,r,k\rangle \stackrel{def}{=} k + a(x).(Pool\langle a,r,k\rangle \parallel Cell\langle r,x\rangle)$$

Pour chaque message m reçu, il génère un processus "ressource" $Cell\langle r,m\rangle$. $Cell\langle r,x\rangle$ est un procesus "ressource" qui sert à stocker temporellement la valeur x. $Cell\langle r,x\rangle$ attend un canal de retour c et essaie d'envoyer sa propre valeur (il est possible que le processus "principal" P ait recu dans les tampons plusieurs messages non-consommés, et donc il peut y avoir plusieurs "resources" en exécution, toutes étant en attente sur le même canal r). La première "ressource" qui réussit à envoyer sa propre valeur au processus principal, finit son exécution, tandis que les autres "ressources" retournent dans un état d'attente de requêtes.

$$Cell\langle r,x\rangle \stackrel{def}{=} r(c).(\bar{c}x + c(y).Cell\langle r,x\rangle)$$

Chaque réception est faite par l'envoi d'une requête sur le canal r.

$$[\![x = receive(); P]\!]_{r,M} \stackrel{def}{=} \nu t(\bar{r}t \parallel t(x).[\![P]\!]_{r,M})$$

Les traductions des primitives "send"et "bcast" sont similaires, mais pour un canal a qui apparait comme un argument de la primitive "send", il y a exactement un receveur, tandis que pour un canal g qui apparait comme argument dans la primitive "bcast", il peut y avoir un ou plusieurs receveurs (en fonction du nombre de membres du groupe g).

$$[\![send(a,m); P]\!]_{r,M} \stackrel{def}{=} \bar{a}m.[\![P]\!]_{r,M}$$

$$[\![bcast(g,m); P]\!]_{r,M} \stackrel{def}{=} \bar{g}m.[\![P]\!]_{r,M}$$

L'adhésion d'un processus à un groupe g est traduite par la création d'un tampon correspondant au groupe g, et par l'actualisation de l'ensemble M.

$$[\![joingroup(g); P]\!]_{r,M} \stackrel{def}{=} \nu t \nu k_g(\bar{t}k_g.Pool\langle g,r,k_g\rangle \parallel t(k_g).[\![P]\!]_{r,M\cup\{(g,k_g)\}})$$

L'opération de création d'un nouveau groupe $g = newgroup()$, se traduit par la création d'un nouveau nom de groupe, et ensuite est identique à $joingroup(g)$.

$$[\![g = newgroup(); P]\!]_{r,M} \stackrel{def}{=} \nu t \nu g \nu k_g(\bar{t}g.\bar{t}k_g.Pool\langle g,r,k_g\rangle \parallel t(g).t(k_g).[\![P]\!]_{r,M\cup\{(g,k_g)\}})$$

Un processus quitte un groupe g en "détruisant" le tampon local correspondant au groupe g.

$$[\![leavegroup(g); P]\!]_{r,M\cup\{(g,k_g)\}} \stackrel{def}{=} \nu t(\overline{k_g}.\bar{t}.nil \parallel t.[\![P]\!]_{r,M})$$

L'opération de création d'un nouveau fils est classique. A la "naissance", le fils ne fait partie d'aucun groupe.

$$[\![x = spawn(Q); P]\!]_{r,M} \stackrel{def}{=} \nu a \nu t(\{Q\}_a \parallel \bar{t}a \parallel t(x).[\![P]\!]_{r,M})$$

Quand un processus finit son exécution, il libère tous les ressources.

$$[\![STOP]\!]_{r,\{(g_1,k_{g_1}),\ldots,(g_n,k_{g_n})\}} \stackrel{def}{=} \overline{k_{g_1}}.\ldots.\overline{k_{g_n}}.\overline{r}.nil)$$

Exemple 20 L'implantation d'une RAM dans le $b\pi$-calcul

Dans cet exemple, nous présentons une implantation d'une *Random Access Machine (RAM)* dans le $b\pi$-calcul (la description est inspirée de (Busi et al., 1998)).

La RAM est un modèle classique de la calculabilité séquentielle; une RAM est composée d'un ensemble fini de registres (qui peuvent contenir des entiers) et d'un programme qui est une séquence finie d'instructions numérotées. Pour faire un calcul, les entrées sont fournies dans les registres $r_1, r_2, \ldots, r_n$; les autres registres sont éventuellement utilisés pendant l'exécution et ils sont supposés contenir la valeur 0 au début du calcul. L'exécution du programme commence avec la première instruction, et continue ensuite séquentiellement par l'exécution de l'instruction suivante, ou par l'instruction rencontrée dans un saut conditionnel. L'exécution se termine quand le programme exécute la dernière instruction dans la séquence (qui n'est pas un saut) ou un saut à un nombre plus grand que l'étiquette de la dernière instruction. Il est bien connu que les deux instructions suivantes sont suffisantes pour obtenir la puissance de calcul de la Machine du Turing:

- $Inc(r_j)$: incrémenter par 1 le contenu du registre r_j;
- $Test(r_j, k)$: si le contenu du r_j est un entier positif, alors décrémenter le contenu du r_j par 1 et aller à l'instruction suivante, sinon sauter à l'instruction qui a le numéro d'ordre k.

Nous modèlisons ainsi les instructions:

- $[\![\ i:\ Inc(r_j)\ ]\!] \stackrel{def}{=} I\langle p_i, p_{i+1}, inc_j\rangle$;
- $[\![\ i:\ Test(r_j, k)\ ]\!] \stackrel{def}{=} T\langle p_i, p_{i+1}, p_k, test_j, zero_j, dec_j\rangle$.

où

$$I\langle p_i, p_{i+1}, inc_j\rangle \stackrel{def}{=} p_i.\overline{inc_j}.\overline{p_{i+1}}.I\langle p_i, p_{i+1}, inc_j\rangle$$

$$T\langle p_i, p_{i+1}, p_k, test_j, zero_j, dec_j\rangle \stackrel{def}{=} p_i.\overline{test_j}.(zero_j.\overline{p_k}.T\langle p_i, p_{i+1}, p_k, test_j, zero_j, dec_j\rangle$$

$$+dec_j.\overline{p_{i+1}}.T\langle p_i, p_{i+1}, p_k, test_j, zero_j, dec_j\rangle)$$

Le registre r_j, qui initialement contient la valeur zéro, est modèlisé par le processus $Z\langle test_j, zero_j, inc_j, dec_j\rangle$:

$$Z\langle test_j, zero_j, inc_j, dec_j\rangle \stackrel{def}{=} test_j.\overline{zero_j}.Z\langle test_j, zero_j, inc_j, dec_j\rangle$$

$$+inc_j.\nu a(O\langle a,test_j,zero_j,inc_j,dec_j\rangle \parallel a.Z\langle test_j,zero_j,inc_j,dec_j\rangle)$$

$$O\langle a,test_j,zero_j,inc_j,dec_j\rangle \stackrel{def}{=}$$

$$test_j.\overline{dec_j}.\bar{a} + inc_j.\nu b(E\langle b,test_j,zero_j,inc_j,dec_j\rangle \parallel b.O\langle a,test_j,zero_j,inc_j,dec_j\rangle)$$

$$E\langle b,test_j,zero_j,inc_j,dec_j\rangle \stackrel{def}{=}$$

$$test_j.\overline{dec_j}.\bar{b} + inc_j.\nu c(O\langle c,test_j,zero_j,inc_j,dec_j\rangle \parallel c.E\langle b,test_j,zero_j,inc_j,dec_j\rangle)$$

Maintenant, si nous considérons le programme $I_1,I_2,\ldots,I_m$ qui a comme entrées $n_1,n_2,\ldots,n_p$ et qui utilise les registres $r_1,r_2,\ldots,r_l$, nous le modélisons par

$$Z\langle test_1,zero_1,inc_1,dec_1\rangle \parallel \ldots \parallel Z\langle test_l,zero_l,inc_l,dec_l\rangle \parallel [\![\ I_1\ ]\!] \parallel \ldots \parallel [\![\ I_m\ ]\!] \parallel B$$

où

$$B \stackrel{def}{=} \underbrace{\overline{inc_1}\ldots\overline{inc_1}}_{n_1\text{fois}}\ldots\underbrace{\overline{inc_p}\ldots\overline{inc_p}}_{n_p\text{fois}}.\overline{p_1}$$

3.3 Conclusion

Dans ce chapitre nous avons introduit le $b\pi$-calcul. Les concepts de diffusion (broadcast) et de mobilité ont été combinés dans un calcul simple et expressif. La syntaxe (Tableau 3.1) est inspirée du π-calcul. La sémantique (Tableau 3.5) est claire et intuitive; les règles (5), (6) et (7) éliminent l'"apparente" incompatibilité qui existerait entre la diffusion et la génération (et propagation) des noms privés.

Dans la Section 3.2, nous montrons le pouvoir d'expression de notre modèle à travers plusieurs exemples.

L'exemple 18 (un algorithme pour la détection des incohérences dans un système transactionel) décrit un algorithme qui utilise toutes les constructions de notre langage. La capacité de pouvoir insérer des noms de canaux dans les messages échangés, plus la diffusion sont irremplacables pour une description qui soit indépendente du nombre de données et de copies existentes dans le système.

L'exemple 19 (sémantique des primitives de communication de groupe) montre que notre modèle fournit un cadre de spécification et d'analyse des systèmes qui échangent des messages par des primitives de communication comme dans les bibliothèques PVM (Geist et al., 1994). Même si le langage est fortement synchrone, l'exemple montre que notre calcul permet la spécification des systèmes distribués asynchrones.

Chapitre 4

Bisimulations

Dans ce chapitre nous définissons plusieurs relations d'équivalence qui permettront d'identifier (ou de distinguer) deux processus en fonction de leurs réponses aux interactions avec l'environnement. Les bisimulations ont été utilisées avec succès dans les algèbres de processus pour comparer des systèmes réactifs relativement à leur capacité opérationnelle de se simuler mutuellement.

Les bisimulations sont des relations co-inductives, étant définies comme des plus grands points fixes d'une application monotone (contrairement aux relations construites inductivement comme des plus petits points fixes).

Les bisimulations ont été définies par Park (Park, 1981), et introduites dans la théorie des systèmes distribués par Milner (Milner, 1983). Deux processus p et q sont bisimilaires si chaque action de p (respectivement q) peut être simulée par q (respectivement p), et les deux processus passent ensuite dans un état où ils restent bisimilaires. En fonction de ce qu'on entend par "action" on peut définir plusieurs notions de bisimulations.

Dans le reste de la thèse, nous utilisons le terme processus pour désigner un processus fermé (qui ne contient pas d'identificateur libre), et nous allons spécifier le cas contraire explicitement quand nécessaire.

4.1 Équivalences

Dans cette section nous introduisons trois classes de bisimulations. On peut appeler les deux premières types de bisimulations, des bisimulations barbelées, étant définies selon le modèle donné dans (Sangiorgi, 1992): on part d'une relation de réduction (de réécriture) et d'un predicat d'observabilité. On obtient ainsi deux relations incomparables: les bisimulations barbelées et les ϕ - bisimulations. Comme les deux classes de relations sont assez faibles, on va considerer leurs clôtures aux contextes statiques. On montre que les relations ainsi obtenues coincident avec les bisimulations étiquetées pour les processus d'image finie.

4.1.1 Les équivalences barbelées

Les bisimulations barbelées ont été introduites pour le π-calcul par Sangiorgi et Milner (dans (Milner et Sangiorgi, 1992), (Sangiorgi, 1992)). C'est une relation facile à définir dans les calculs munis d'une sémantique opérationnelle (ou dans les systèmes de réécriture): il suffit de définir une notion d'observabilité et ensuite de distinguer deux processus chaque fois qu'ils ne sont pas observables de la même façon, ou chaque fois que leur observabilité n'est pas préservée par la réduction (ou la réécriture). Les bisimulations barbelées ont déjà été utilisées dans des modèles basés sur la diffusion (Hennessy et Rathke, 1995), mais dans un calcul où les communications étaient faites par l'intermédiaire d'un *éther* global plutôt qu'à travers des canaux explicites.

Les bisimulations barbelées décrivent une relation entre des processus qui peuvent faire les mêmes actions visibles, et si un des processus peut progresser silencieusement, l'autre le peut aussi, en évoluant vers un processus qui préserve la relation. Dans un calcul à diffusion, les émissions sont visibles (si nous surveillons sur un canal un processus, nous recevons chaque valeur qu'il envoie sur ce canal), tandis que les réceptions ne le sont pas (comme l'émission n'est pas bloquante, nous ne pouvons pas savoir si le processus observé a reçu ou a ignoré le message qu'on lui a envoyé). Nous notons $p \downarrow_a$ (respectivement $p \Downarrow_a$) si $p \xrightarrow{\alpha} p'$ (et respectivement $p \overset{\alpha}{\Longrightarrow} p'$) pour une émission α de sujet a.

Définition 21 Les bisimulations barbelées

Une relation symétrique S sur l'ensemble $\mathcal{P}_b$ (de processus) est une **bisimulation barbelée faible** *si et seulement si $(p,q) \in S$ implique*

- *si $p \xrightarrow{\tau} p'$, alors $\exists q'$ tel que $q \overset{\epsilon}{\Longrightarrow} q'$ et $(p',q') \in S$,*
- *$\forall a \in Ch_b$, si $p \downarrow_a$ alors $q \Downarrow_a$.*

Deux processus sont bisimilaires au sens de la bisimulation barbelée faible*, noté $p \approx_b q$, si $(p,q) \in S$ pour une bisimulation barbelée faible S.*

La **bisimulation barbelée forte** *$\sim_b$ est définie d'une façon similaire, en remplaçant $\Longrightarrow$ par $\longrightarrow$ et $\Downarrow$ par $\downarrow$.*

Nous prouvons quelques équivalences entre des processus considérés syntaxiquement congruents dans certains présentationes classiques du π-calcul.

Le lemme suivant résume quelques propriétés sur la relation $\sim_b$.

Lemme 22 *$\sim_b$ satisfait les propriétés:*

(a) $p \equiv_\alpha q$ *implique* $p \sim_b q$

(b) $p \parallel nil \sim_b p$

(c) $p \parallel q \sim_b q \parallel p$

(d) $(p \parallel q) \parallel r \sim_b p \parallel (q \parallel r)$

(e) $p + nil \sim_b p$

(f) $p + q \sim_b q + p$

(g) $(p+q)+r \sim_b p+(q+r)$

(h) $\nu x p \sim_b p$ *si* $x \notin fn(p)$

(i) $\nu y \nu x p \sim_b \nu x \nu y p$ *si* $x \neq y$

(j) $(\nu x p) \parallel q \sim_b \nu x(p \parallel q)$ *si* $x \notin fn(q)$

(k) $(\nu x p) + q \sim_b \nu x(p+q)$ *si* $x \notin fn(q)$

(l) $\langle y = z\rangle(\nu x p),q \sim_b \nu x(\langle y = z\rangle p,q)$ *si* $x \notin fn(q) \cup \{y,z\}$

Preuve

Consequence des Lemmes 34 et 39.

□ *Lemme* 22

Le lemme suivant prouve une propriété assez intéressante de la bisimulation barbelée: la bisimulation barbelée est préservée par l'opérateur de parallélisme.

Lemme 23 *$\sim_b$ et $\approx_b$ sont préservés par le parallélisme:*

- $p \sim_b q$ *implique* $\forall r,\ p \parallel r \sim_b q \parallel r$,
- $p \approx_b q$ *implique* $\forall r,\ p \parallel r \approx_b q \parallel r$,

Preuve

Nous prouvons par exemple, seulement le cas faible. Il suffit de prouver que la relation $\mathcal{R}$ où

$\mathcal{R} \stackrel{def}{=} \{(p \parallel r, q \parallel r) \mid p,q,r \in \mathcal{P}_b,\ p \approx_b q\}$

est une bisimulation barbelée faible.

- Soit $p \parallel r \downarrow_a$.

  Alors soit $p \downarrow_a$, soit $r \downarrow_a$ (obtenus à partir des règles (12) ou (13) ou de l'une de leurs symétriques).

  Si $p \downarrow_a$, comme $p \approx_b q$ on obtient $q \Downarrow_a$ et donc par des applications successives de la règle (13), suivi éventuellement par une application de la règle (12), on obtient $(q \parallel r) \Downarrow_a$. Si $r \downarrow_a$, alors évidemment $(q \parallel r) \downarrow_a$.

– Soit $p \parallel r \xrightarrow{\tau} s$ une transition silencieuse de p.

Alors la dernière règle utilisée dans l'inférence de cette transition est la règle (13) ou sa symétrique.

Si la règle appliquée est la règle (13), alors $s \equiv p' \parallel r$ et $p \xrightarrow{\tau} p'$. Comme $p \approx_b q$ on obtient $q \stackrel{\epsilon}{\Longrightarrow} q'$ avec $p' \approx_b q'$. Par des applications successives de la règle (13), on obtient $q \parallel r \stackrel{\epsilon}{\Longrightarrow} q' \parallel r$.

Si la règle appliquée est la symétrique de la règle (13), alors $s \equiv p \parallel r'$ et $r \xrightarrow{\tau} r'$. Par une application de la symétrique de la règle (13) on obtient $q \parallel r \xrightarrow{\tau} q \parallel r'$.

□ *Lemme* 23

Remarque 24 *Contrairement au π-calcul, dans le bπ-calcul il n'est plus vrai que $p \approx_b q$ ($p \sim_b q$), implique $\nu a p \approx_b \nu a q$ ($\nu a p \sim_b \nu a q$) pour tout canal a. Par exemple, il suffit de prendre $p_0 \stackrel{def}{=} \bar{a}b$ et $q_0 \stackrel{def}{=} \bar{a}b.\bar{c}d$ qui sont bisimilaires au sens de la bisimulation barbelée forte, tandis que $\nu a p_0$ et $\nu a q_0$ ne sont pas même pas bisimilaires au sens de la bisimulation barbelée faible.*

Le lemme 23 et la remarque 24 montrent des différences surprenantes entre le $b\pi$-calcul et le π-calcul. Dans notre modèle, la bisimulation barbelée est préservée par le parallélisme, mais elle n'est pas préservée par la restriction. Dans le π-calcul, la bisimulation barbelée est préservée par la restriction mais elle n'est pas préservée par le parallélisme.

Comme la bisimulation barbelée n'est pas conservée par la restriction et ne peut faire aucune distinction entre $\alpha.p$ et $\alpha.q$ pour tous processus p,q, et action $\alpha \neq \tau$, elle est trop faible pour l'utiliser à identifier deux processus. Comme dans le π-calcul, il est nécessaire de la clore par rapport aux divers familles de contextes pour obtenir des relations plus restrictives. Nous allons considérer sa clôture par rapport aux contextes statiques dans cette section, et nous étudierons sa clôture par rapport à tous les contextes dans la section 4.2, section réservé aux congruences. Un *contexte statique*(défini dans le Tableau 4.1), est un contexte qui est construit par l'utilisation d'un "trou", du parallélisme et du "new".

$$C ::= \quad \bullet \ \mid \ \nu x C \ \mid \ C \parallel p \ \mid \ p \parallel C$$

où $p \in \mathcal{P}_b$ et $\alpha ::= x(\tilde{y}) \ \mid \ \bar{x}\tilde{y} \ \mid \ \tau$ avec $x \in Ch_b, \tilde{y} \subseteq Ch_b$.

TAB. 4.1 – *Contextes statiques Con_s*

Définition 25 L'équivalence barbelée

– *Deux processus p et q sont équivalents au sens de* **l'équivalence barbelée faible**, *noté $p \stackrel{e}{\approx}_b q$, si pour tout contexte statique $C[\bullet]$, $C[p] \approx_b C[q]$.*

– **L'équivalence barbelée forte** ($\overset{e}{\sim}_b$) *est obtenue pareillement à partir de la bisimulation barbelée forte sur les processus.*

En utilisant des techniques classiques, on peut prouver que $\approx_b$, $\sim_b$, $\overset{e}{\approx}_b$ et $\overset{e}{\sim}_b$ sont des équivalences (.a.d. ils sont reflexives, symétriques et transitives).

La remarque 24 implique que l'équivalence barbelée ne coïncide plus avec la clôture de la bisimulation barbelée relativement aux observateurs (comme dans le π-calcul).

4.1.2 Équivalences transitionnelles

Dans notre calcul les processus évoluent d'une manière autonome par la relation $\overset{\oslash}{\Longrightarrow}$ est non pas par $\overset{\epsilon}{\Longrightarrow}$. C'est pour cette raison que nous allons définir une autre caractérisation de l'equivalence barbelée, toujours basée sur les observations.

Nous notons $p \Downarrow_a^{\phi}$ si $p \overset{\oslash}{\Longrightarrow}\overset{\alpha}{\longrightarrow} p'$ pour une émission α de sujet a.

Définition 26 La bisimulation transitionnelle

Une relation symétrique S sur l'ensemble $\mathcal{P}_b$ (des processus) est une **bisimulation transitionnelle faible** *si $(p,q) \in S$, implique*

– *si $p \overset{\oslash}{\longrightarrow} p'$, alors $\exists q'$ tel que $q \overset{\oslash}{\Longrightarrow} q'$ et $(p',q') \in S$,*

– *$\forall a \in Ch_b$, si $p \downarrow_a$ alors $q \Downarrow_a^{\phi}$.*

Deux processus p et q sont bisimilaires au sens de la **bisimulation transitionnelle faible**, *noté $p \approx_\phi q$, si $(p,q) \in S$ pour une bisimulation transitionnelle faible S.*

La **bisimulation transitionnelle forte** $\sim_\phi$ *est défini d'une manière similaire, en remplaçant $\overset{\oslash}{\Longrightarrow}$ par $\overset{\oslash}{\longrightarrow}$ et $\Downarrow^{\phi}$ par $\downarrow$.*

Intuitivement, la bisimulation transitionnelle forte (ou faible) identifie les processus qui peuvent diffuser des messages sur les mêmes ensembles de canaux (immédiatement ou après un certain nombre de transitions autonomes), et chaque fois qu'un des processus peut progresser sans impliquer l'environnement, l'autre processus peut à son tour évoluer d'une manière autonome dans un état qui préserve la relation. En effet, dans notre calcul, la bisimulation transitionnelle est plus naturelle que la bisimulation barbelée, ($\overset{\oslash}{\longrightarrow}$ ($\overset{\oslash}{\Longrightarrow}$) étant la relation de réduction et non $\overset{\tau}{\longrightarrow}$ (ou $\overset{\epsilon}{\Longrightarrow}$) comme dans le π-calcul).

Dans ce qui suit, on va utiliser aussi le terme ϕ - bisimulation pour la bisimulation transitionnelle.

Comme pour la bisimulation barbelée, nous avons les équivalences transitionnelles suivantes.

Lemme 27 *$\sim_\phi$ satisfait les propriétés:*

(a) *$p \equiv_\alpha q$ implique $p \sim_\phi q$*

(b) $p \parallel nil \sim_\phi p$

(c) $p \parallel q \sim_\phi q \parallel p$

(d) $(p \parallel q) \parallel r \sim_\phi p \parallel (q \parallel r)$

(e) $p + nil \sim_\phi p$

(f) $p + q \sim_\phi q + p$

(g) $(p + q) + r \sim_\phi p + (q + r)$

(h) $\nu x p \sim_\phi p$ *si* $x \notin fn(p)$

(i) $\nu y \nu x p \sim_\phi \nu x \nu y p$ *si* $x \neq y$

(j) $(\nu x p) \parallel q \sim_\phi \nu x(p \parallel q)$ *si* $x \notin fn(q)$

(k) $(\nu x p) + q \sim_\phi \nu x(p + q)$ *si* $x \notin fn(q)$

(l) $\langle y = z\rangle(\nu x p), q \sim_\phi \nu x(\langle y = z\rangle p, q)$ *si* $x \notin fn(q) \cup \{y, z\}$

Preuve

Consequence des Lemmes 34 et 41.

□ *Lemme* 27

Comme la bisimulation barbelée, la bisimulation transitionnelle est une relation très faible (elle n'est préservée ni par la composition parallèle ni par la restriction).

Remarque 28

1. $\approx_\phi$ $(\sim_\phi)$ *n'est pas préservée par l'opérateur* $\parallel$*;*
2. $\approx_\phi$ $(\sim_\phi)$ *n'est pas préservée par l'opérateur* ν*;*
3. $\sim_b$ $(\approx_b)$ *et* $\sim_\phi$ $(\approx_\phi)$ *ne sont pas comparables.*

Preuve

1. Soit $p_1 \stackrel{def}{=} \bar{b} + \tau.\bar{c}$, $q_1 \stackrel{def}{=} \bar{b} + \bar{b}.\bar{c}$ et $r_1 \stackrel{def}{=} b + \bar{a}$.
   Alors $p_1 \sim_\phi q_1$. Par contre $(p_1 \parallel r_1) \not\approx_\phi (q_1 \parallel r_1)$ car $q_1 \parallel r_1$ ne peut pas simuler la transition $(p_1 \parallel r_1) \xrightarrow{\tau} (\bar{c} \parallel b + \bar{a})$.
2. Soit $p_2 \stackrel{def}{=} \bar{b}a.\bar{a}$, $q_2 \stackrel{def}{=} \bar{b}c.\bar{a}$.
   Alors $p_2 \sim_\phi q_2$. Par contre $\nu a p_2 \not\approx_\phi \nu a q_2$, car $\nu a q_2$ ne peut pas simuler la transition $\nu a p_2 \xrightarrow{\nu a \bar{b} a} \bar{a}$.
3. Comme $\nu a p_2 \sim_b \nu a q_2$ et $\nu a p_2 \not\approx_\phi \nu a q_2$, on obtient
   $$\sim_b \not\subseteq \approx_\phi .$$
   On a $p_1 \not\approx_b q_1$ car q_1 ne peut pas simuler la transition $p_1 \xrightarrow{\tau} \bar{c}$. Comme $p_1 \sim_\phi q_1$ on obtient
   $$\sim_\phi \not\subseteq \approx_b .$$

Pour obtenir l'assertion 3. il suffit d'utiliser les inégalités $\sim_b \subseteq \approx_b$ et $\sim_\phi \subseteq \approx_\phi$.

$\square$ *Remarque* 28

La remarque 28 justifie la définition de la clôture aux contextes statiques de la bisimulation transitionnelle.

Définition 29 L'équivalence transitionnelle

- *Deux processus p et q sont équivalents au sens de* **l'équivalence transitionnelle faible**, *noté* $p \overset{e}{\approx}_\phi q$, *si pour chaque contexte statique* $C[p] \approx_\phi C[q]$;
- **L'équivalence transitionnelle forte** ($\overset{e}{\sim}_\phi$) *est obtenue pareillement à partir de la bisimulation transitionnelle forte sur les processus.*

Par l'utilisation de quelques contextes appropriés, nous prouvons que l'équivalence transitionnelle implique l'équivalence barbelée.

Lemme 30

1) $p \overset{e}{\sim}_\phi q$ *implique* $p \sim_b q$
2) $p \overset{e}{\approx}_\phi q$ *implique* $p \approx_b q$.

Preuve

Nous démontrons seulement le cas faible.

Soit $p \overset{e}{\approx}_\phi q$ et soit $M \overset{def}{=} \{a' \mid a \in fn(p,q)\}$ un ensemble de canaux tel que $M \cap fn(p,q) = \emptyset$ et c un nouveau canal tel que $c \notin M \cup fn(p,q)$. Soit $T \overset{def}{=} \Sigma_{a \in fn(p,q)} a(x).\bar{a}' + \bar{c}$.

On va prouver que la relation

$$\mathcal{R}^T \overset{def}{=} \{(p',q') \mid fn(p',q') \subseteq fn(p,q),\ p' \parallel T \approx_\phi q' \parallel T\}$$

est une bisimulation barbelée faible.

Soit $(p',q') \in \mathcal{R}^T$. Alors $fn(p',q') \subseteq fn(p,q)$, $p' \parallel T \approx_\phi q' \parallel T$.

- Soit $p' \downarrow_d$.
  Il suit que $d \neq c$, et $p' \xrightarrow{\alpha}$ pour une émission α de sujet d. On obtient $p' \parallel T \xrightarrow{\alpha} (p'' \parallel \bar{d}')$. Comme $p' \parallel T \approx_\phi q' \parallel T$, nous obtenons que $q' \parallel T \Longrightarrow r$ où $(p'' \parallel \bar{d}') \approx_\phi r$. Alors $r = (q''' \parallel \bar{d}')$, où $q' \overset{\epsilon}{\Longrightarrow} \xrightarrow{\gamma} q'' \Longrightarrow q'''$, pour une émission γ de sujet d.
- Soit $p' \xrightarrow{\tau} p''$.
  Alors $p' \parallel T \xrightarrow{\tau} p'' \parallel T$. Comme $p' \parallel T \approx_\phi q' \parallel T$, il suit que $q' \parallel T \Longrightarrow r$, où $p'' \parallel T \approx_\phi r$. Si $\phi \neq \epsilon$, il est facile à remarquer que T a du participer dans la transition ($fn(q') \subseteq M$), et que $r \not\Downarrow_c^\phi$, tandis $(p'' \parallel T) \downarrow_c$, contradiction avec $p'' \parallel T \approx_\phi r$. Donc $r = q'' \parallel T$ où $q' \overset{\epsilon}{\Longrightarrow} q''$. Par le corollaire 8 nous obtenons aussi $fn(p'',q'') \subseteq fn(p',q')$.

□ *Lemme* 30

Corolaire 31

1) $p \overset{e}{\sim}_\phi q$ *implique* $p \overset{e}{\sim}_b q$

2) $p \overset{e}{\approx}_\phi q$ *implique* $p \overset{e}{\approx}_b q$.

Preuve

On va démontrer seulement le cas faible.

Soit $p \overset{e}{\approx}_\phi q$ et soit $C_1[\bullet]$ un contexte statique quelconque. Pour tout contexte statique $C[\bullet]$, comme $C[C_1[\bullet]]$ est aussi un contexte statique, par la Définition 29 on obtient que, $C[C_1[p]] \approx_\phi C[C_1[q]]$. On obtient ainsi $C_1[p] \overset{e}{\approx}_\phi C_1[q]$.

Par le Lemme 30, on obtient alors $C_1[p] \approx_b C_1[q]$. Par la Définition 25 on obtient $p \overset{e}{\approx}_b q$.

□ *Corolaire* 31

4.1.3 Les bisimulations étiquetées

Pour prouver que deux processus p et q ne sont pas équivalents au sens de l'équivalence $\overset{e}{\sim}_b$ (et respectivement $\overset{e}{\sim}_\phi$, $\overset{e}{\approx}_\phi$, $\overset{e}{\approx}_b$), il suffit de trouver un contexte statique C, tel que $C[p]$ et $C[q]$ ne sont pas équivalents au sens de la $\sim_b$ (et respectivement que $C[p]$ et $C[q]$ ne sont pas bisimilaires au sens de la $\sim_\phi$ $\approx_\phi$, $\approx_b$). Mais il est plus difficile, avec les définitions données, de prouver l'équivalence barbelée ou la ϕ - équivalence. La preuve exige une quantification sur tous les contextes statiques. Donc il est plus intéressant d'avoir une manière de prouver directement le fait que deux systèmes sont équivalents.

Définition 32 Les bisimulations étiquetées faibles

Une relation symétrique S *sur l'ensemble* $\mathcal{P}_b$ *(de processus) est une* **bisimulation faible** *si* $(p,q) \in S$, *implique*

1) *si* $p \xrightarrow{\tau} p'$, *alors* $\exists q'$ *tel que* $q \overset{\epsilon}{\Longrightarrow} q'$ *et* $(p',q') \in S$,

2) *si* $p \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} p'$ *et* $\tilde{b} \cap fn(p,q) = \emptyset$, *alors* $\exists q'$ *tel que* $q \overset{\nu\tilde{b}\bar{a}\tilde{c}}{\Longrightarrow} q'$ *et* $(p',q') \in S$,

3) *si* $p \xrightarrow{a\langle\tilde{b}\rangle?} p'$ *alors* $\exists q'$ *tel que* $q \overset{\epsilon}{\Longrightarrow} \bullet \xrightarrow{a\langle\tilde{b}\rangle?} \bullet \overset{\epsilon}{\Longrightarrow} q'$ *et* $(p',q') \in S$.

Deux processus p *et* q *sont* **faiblement bisimilaires**, *noté* $p \approx q$, *si* $(p,q) \in S$ *pour une bisimulation faible* S.

La **bisimulation** forte $\sim$ est définie pareillement, en remplaçant $\Longrightarrow$ par $\longrightarrow$ dans les conditions 1) and 2), et par la substitution de la condition 3) par "si $p \xrightarrow{a\langle\tilde{b}\rangle?} p'$ alors $\exists q'$ tel que $q \xrightarrow{a\langle\tilde{b}\rangle?} q'$ et $(p',q') \in S$".

Définition 33 Les bisimulations étiquetées fortes

Une relation symétrique S sur l'ensemble $\mathcal{P}_b$ (de processus) est une **bisimulation forte** *si* $(p,q) \in S$, *implique*

1) *si* $p \xrightarrow{\tau} p'$, *alors* $\exists q'$ *tel que* $q \xrightarrow{\tau} q'$ *et* $(p',q') \in S$,
2) *si* $p \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} p'$ *et* $\tilde{b} \cap fn(p,q) = \emptyset$, *alors* $\exists q'$ *tel que* $q \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} q'$ *et* $(p',q') \in S$,
3) *si* $p \xrightarrow{a\langle\tilde{b}\rangle?} p'$ *alors* $\exists q'$ *tel que* $q \xrightarrow{a\langle\tilde{b}\rangle?} q'$ *et* $(p',q') \in S$.

Deux processus p et q sont **fortement bisimilaires**, *noté* $p \sim q$, *si* $(p,q) \in S$ *pour une bisimulation forte S.*

Par des arguments classiques (Milner, 1989), nous pouvons prouver que $\sim$ et $\approx$ sont des équivalences.

Comme pour la bisimulation barbelée et pour la ϕ - bisimulation, nous avons pour la bisimulation forte les équivalences suivantes.

Lemme 34 $\sim$ *satisfait les propriétés:*

(a) $p \equiv_\alpha q$ *implique* $p \sim q$
(b) $p \parallel nil \sim p$
(c) $p \parallel q \sim q \parallel p$
(d) $(p \parallel q) \parallel r \sim p \parallel (q \parallel r)$
(e) $p + nil \sim p$
(f) $p + q \sim q + p$
(g) $(p+q)+r \sim p+(q+r)$
(h) $\nu x p \sim p$ *si* $x \notin fn(p)$
(i) $\nu y \nu x p \sim \nu x \nu y p$ *si* $x \neq y$
(j) $(\nu x p) \parallel q \sim \nu x(p \parallel q)$ *si* $x \notin fn(q)$
(k) $(\nu x p) + q \sim \nu x(p+q)$ *si* $x \notin fn(q)$
(l) $\langle y = z\rangle(\nu x p),q \sim \nu x(\langle y = z\rangle p,q)$ *si* $x \notin fn(q) \cup \{y,z\}$.

Preuve

Il suffit de prouver que les relations suivantes sont des bisimulations fortes (voir Annexe B).

(a) $\mathcal{S}^1 \stackrel{def}{=} \{(p,q) \mid p,q \in \mathcal{P}_b,\ p \equiv_\alpha q\}$
(b) $\mathcal{S}^2 \stackrel{def}{=} \{(p \parallel nil,p) \mid p, \in \mathcal{P}_b\}$
(c) $\mathcal{S}^3 \stackrel{def}{=} \{(p \parallel q,q \parallel p) \mid p,q \in \mathcal{P}_b\}$
(d) $\mathcal{S}^4 \stackrel{def}{=} \{(p \parallel q) \parallel r,p \parallel (q \parallel r) \mid p,q,r \in \mathcal{P}_b\}$
(e) $\mathcal{S}^5 \stackrel{def}{=} \{(p+nil,p) \mid p \in \mathcal{P}_b\} \bigcup \{(p,p) \mid p \in \mathcal{P}_b\}$

(f) $\mathcal{S}^6 \stackrel{def}{=} \{(p+q,q+p) \mid p,q \in \mathcal{P}_b\} \bigcup \{(p,p) \mid p \in \mathcal{P}_b\}$

(g) $\mathcal{S}^7 \stackrel{def}{=} \{((p+q)+r,p+(q+r)) \mid p,q,r \in \mathcal{P}_b\} \bigcup \{(p,p) \mid p \in \mathcal{P}_b\}$

(h) $\mathcal{S}^8 \stackrel{def}{=} \{(\nu x p,p) \mid p \in \mathcal{P}_b, x \notin fn(p)\}$

(i) $\mathcal{S}^9 \stackrel{def}{=} \{(\nu x_1 \ldots \nu x_n p, \nu x_{\sigma(1)} \ldots \nu x_{\sigma(n)} p) \mid p \in \mathcal{P}_b,\ n \in \mathbb{N},\ \sigma$ permutation de $\{1,\ldots,n\}$, $x_i \neq x_j,$ pour tous $i,j\}\}$

(j) $\mathcal{S}^{10} \stackrel{def}{=} \{((\nu x p) \parallel q, \nu x(p \parallel q)) \mid p \in \mathcal{P}_b, x \notin fn(q)\} \bigcup \{(p,p) \mid p \in \mathcal{P}_b\}$

(k) $\mathcal{S}^{11} \stackrel{def}{=} \{((\nu x p) + q, \nu x(p+q)) \mid p \in \mathcal{P}_b, x \notin fn(q)\} \bigcup \{(p,p) \mid p \in \mathcal{P}_b\}$

(l) $\mathcal{S}^{12} \stackrel{def}{=} \{(\langle y=z\rangle(\nu x p),q,\nu x(\langle y=z\rangle p,q)) \mid p,q \in \mathcal{P}_b, x \notin fn(q) \cup \{y,z\}\} \bigcup \{(p,p) \mid p \in \mathcal{P}_b\}$

$\Box$ *Lemme* 34

Les lemmes 22, 27 et 34 nous permettent d'omettre les parenthèses dans $p \parallel q \parallel r$ ou $p+q+r$ dans certains contextes, et d'omettre "*nil*"dans certains termes.

Comme dans (Milner, 1989), nous utilisons des "bisimulations jusqu'au" pour réduire la dimension des relations dans les preuves. Nous allons utiliser seulement les bisimulations jusqu'au $\sim$.

Définition 35 *Une relation symétrique S est une* **bisimulation faible jusqu'au** $\sim$ *si* $(p,q) \in S$*, implique*

1) si $p \xrightarrow{\tau} p'$, alors $\exists q'$ tel que $q \stackrel{\epsilon}{\Longrightarrow} q'$ et $(p',q') \in \sim S \sim$,

2) si $p \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} p'$ et $\tilde{b} \cap fn(p,q) = \emptyset$, alors $\exists q'$ tel que $q \stackrel{\nu\tilde{b}\bar{a}\tilde{c}}{\Longrightarrow} q'$ et $(p',q') \in \sim S \sim$,

3) si $p \xrightarrow{a\langle\tilde{b}\rangle?} p'$ alors $\exists q'$ tel que $q \stackrel{\epsilon}{\Longrightarrow} \bullet \xrightarrow{a\langle\tilde{b}\rangle?} \bullet \stackrel{\epsilon}{\Longrightarrow} q'$ et $(p',q') \in \sim S \sim$.

La définition de la **bisimulation forte jusqu'au** $\sim$ *suit comme dans la Définition 33.*

Par des arguments standards, nous pouvons prouver le lemme suivant.

Lemme 36 *Si S est une bisimulation faible jusqu'au $\sim$ (bisimulation forte jusqu'au $\sim$) alors $S \subseteq \approx$ ($S \subseteq \sim$).*

Preuve

On va donner la preuve seulement pour le cas faible.

Soit S une bisimulation faible jusqu'au $\sim$. Nous prouvons que

$$\mathcal{S}^u \stackrel{def}{=} \bigcup_{n<\infty} \mathcal{S}_n$$

est une bisimulation faible, où

$$\mathcal{S}_0 \stackrel{def}{=} \{(p,q) \mid p \sim S \sim q\ \}$$

$\mathcal{S}_{n+1} \stackrel{def}{=} \{(p,q) \mid (p_1,q_1) \in \mathcal{S}_n,\ p \equiv_\alpha p_1[\tilde{z}/\tilde{x}],\ q \equiv_\alpha q_1[\tilde{z}/\tilde{x}],\ \tilde{z} \cap fn(p_1,q_1) = \emptyset,\ [\tilde{z}/\tilde{x}]\ \text{injective}\ \}$

La preuve est par induction sur n.

- Soit $(p,q) \in \mathcal{S}_0$. Alors $\exists p_1,q_1$ tel que $p \sim p_1\ S\ q_1 \sim q$.

  1. Pour toute transition $p \xrightarrow{\alpha} p'$ telle que $bn(\alpha) \cap fn(p,q,p_1,q_1) = \emptyset$, on peut montrer que l'on peut remplir le diagramme suivant:

$$\begin{array}{ccccccc} p & \sim & p_1 & & S & & q_1 & \sim & q \\ \big\downarrow \alpha & & \big\downarrow \alpha & & & & \Big\Downarrow \hat{\alpha} & & \Big\Downarrow \hat{\alpha} \\ p' & \sim & p'_1 & \sim & S & \sim & q'_1 & \sim & q' \end{array}$$

     On prouve l'assertion seulement pour le cas $\alpha = \nu\tilde{b}\bar{a}\tilde{c}$. Les autres cas sont similaires.
     Soit $p \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} p'$, avec $\tilde{b} \cap fn(p,q,p_1,q_1) = \emptyset$.
     Alors $p \sim p_1$ implique $p_1 \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} p'_1$.
     Comme $p_1\ S\ q_1$, par la définition 35 on obtient $q_1 \stackrel{\nu\tilde{b}\bar{a}\tilde{c}}{\Longrightarrow} q'_1$ avec $p'_1 \sim S \sim q'_1$.
     Par induction sur la longueur de $\stackrel{\nu\tilde{b}\bar{a}\tilde{c}}{\Longrightarrow}$ et en utilisant le fait que $q_1 \sim q$, on obtient $q \stackrel{\nu\tilde{b}\bar{a}\tilde{c}}{\Longrightarrow} q'$ avec $q' \sim q'_1$.
     Ensuite on utilise la transitivité de $\sim$ pour obtenir $(p_1,q_1) \in \mathcal{S}_0$.
  2. Si $p \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} p'$ et $\tilde{b} \cap fn(p_1,q_1) \neq \emptyset$, alors soit $\tilde{d}$ tel que $\tilde{d} \cap fn(p,q,p_1,q_1) = \emptyset$. Par le Lemme 13, on obtient $p \xrightarrow{\nu\tilde{d}\bar{a}\tilde{c}} p''$, avec $p' = p''[\tilde{b}/\tilde{d}]$.
     Repetant le raisonnement du cas 1., on obtient que $q \stackrel{\nu\tilde{d}\bar{a}\tilde{c}}{\Longrightarrow} q''$ avec $p'' \sim S \sim q''$.
     Par le Lemme 13, on obtient $q \stackrel{\nu\tilde{b}\bar{a}\tilde{c}}{\Longrightarrow} q'$ avec $q' = q''[\tilde{b}/\tilde{d}]$, et donc $(p',q') \in \mathcal{S}_1$.

- Soit $(p,q) \in \mathcal{S}_{n+1}$. Alors $\exists (p_1,q_1) \in \mathcal{S}_n$ et $[\tilde{z}/\tilde{x}]$ une substitution injective tels que $p \equiv_\alpha p_1[\tilde{z}/\tilde{x}]$, $q \equiv_\alpha q_1[\tilde{z}/\tilde{x}]$, et $fn(p_1,q_1) \cap \tilde{z} = \emptyset$.
  Soit $p \xrightarrow{\alpha} r''$ avec $bn(\alpha) \cap (fn(p_1,q_1) \cup \tilde{z}) = \emptyset$.
  Par le Lemme 34 on obtient $p_1[\tilde{z}/\tilde{x}] \xrightarrow{\alpha} r'$ avec $r' \equiv_\alpha r''$.
  Par un raisonnement similaire au Lemme 15, on peut prouver qu'il existe une substitution σ' injective, extension de $[\tilde{z}/\tilde{x}]$, une action β et un processus r tels que $r' \equiv_\alpha r\sigma'$, $\alpha = \beta\sigma'$, $\sigma' \setminus fn(p_1,q_1) = \sigma \setminus fn(p_1,q_1)$, $prcod(\sigma') \cap fn(p_1,q_1,\beta) = \emptyset$ et $p_1 \xrightarrow{\beta} r$.
  Comme $(p_1,q_1) \in \mathcal{S}_n$, par l'hypothèse d'induction on obtient $q_1 \stackrel{\beta}{\Longrightarrow} s$ avec $(r,s) \in \mathcal{S}_m$ pour un certain $m \in \mathbb{N}$.
  Par le Lemme 14, on a $q_1\sigma' \stackrel{\alpha}{\Longrightarrow}\equiv_\alpha s\sigma'$ et comme $q \equiv_\alpha q_1\sigma'$ par le Lemme 34 on obtient $q \stackrel{\alpha}{\Longrightarrow} s'' \equiv_\alpha s\sigma'$

Comme $prcod(\sigma') \cap fn(p_1,q_1,\beta) = \emptyset$, par le Lemme 8 on obtient $prcod(\sigma') \cap fn(r,s) = \emptyset$, et donc $(r'',s'') \in \mathcal{S}_{m+1}$.

La preuve pour le cas bisimulation forte jusqu'au $\sim$ forte est similaire.

□ *Lemme* 36

La bisimulation étiquetée est une congruence par rapport à la restriction et au parallélisme.

Lemme 37 *Si $p \approx q$ ($p \sim q$) alors $\nu ap \approx \nu aq$ ($\nu ap \sim \nu aq$).*

Preuve

Nous allons prouver que la relation

$$\mathcal{S} \stackrel{def}{=} \bigcup_{n=0}^{\infty} \mathcal{S}_n$$

où
$\mathcal{S}_0 \stackrel{def}{=} \{(p,q) \mid p \approx q\}$
$\mathcal{S}_{n+1} \stackrel{def}{=} \{(p,q) \mid \exists a \in Ch_b, \exists (p',q') \in \mathcal{S}_n,\ p = \nu ap',\ q = \nu aq'\}$
est une bisimulation faible.

On prouve par induction sur n, que si $(p,q) \in \mathcal{S}_n$ et $p \xrightarrow{\alpha} p'$ où α satisfait les conditions de la Définition 32, alors il existe $m \in \mathbb{N}$ et $q \stackrel{\hat{\alpha}}{\Longrightarrow} q'$, tel que $(p',q') \in \mathcal{S}_m$.

Le cas $n = 0$ suit directement de la Définition 32.

Soit $(p,q) \in \mathcal{S}_{n+1}$. Donc $p = \nu ap_1$ et $q = \nu aq_1$ avec $(p_1,q_1) \in \mathcal{S}_n$.

Soit $p \xrightarrow{\alpha} p'$ avec $bn(\alpha) \cap fn(p,q) = \emptyset$. On a plusieurs cas, en fonction de la dernière règle utilisée.

- Si la dernière règle utilisée est (4), alors $\alpha = \nu w \nu \tilde{b} \bar{d} \tilde{c}$, $p_1 \xrightarrow{\nu \tilde{b} \bar{d} \tilde{c}} p'_1$, $p' = p'_1[w/a]$, $w \notin fn(\nu ap'_1)$ et $a \in \tilde{c} \setminus (\tilde{b} \cup \{d\})$. Par induction $\exists m$ tel que $q_1 \stackrel{\nu \tilde{b} \bar{d} \tilde{c}}{\Longrightarrow} q'_1$ et $(p',q') \in \mathcal{S}_m$. On obtient par une application de la règle (4) et plusieurs applications de la règle (6), que $q \stackrel{\nu w \nu a \tilde{b} \bar{d} \tilde{c}}{\Longrightarrow} q'_1[w/a]$, avec $(p'_1[w/a],q'_1[w/a]) \in \mathcal{S}_{m+1}$.
- Si la dernière règle utilisée est (5), alors $p_1 \xrightarrow{\nu \tilde{b} \bar{a} \tilde{c}} p'_1$ et $p' = \nu a \nu \tilde{b} p'_1$. Par induction, $\exists m$ tel que $q_1 \stackrel{\nu \tilde{b} \bar{a} \tilde{c}}{\Longrightarrow} q'_1$ et $(p'_1,q'_1) \in \mathcal{S}_m$. On obtient par une application de la règle (5) et plusieurs applications de la règle (6), que $q \stackrel{\tau}{\Longrightarrow} \nu a \nu \tilde{b} q'_1$ avec $(\nu a \nu \tilde{b} p'_1, \nu a \nu \tilde{b} q'_1) \in \mathcal{S}_{m+1+|\tilde{b}|}$.
- Si la dernière règle utilisée est (6), alors $p_1 \xrightarrow{\alpha} p'_1$, $a \notin n(\alpha)$ et $p' = \nu ap'_1$. Par induction $\exists m$ tel que $q_1 \stackrel{\alpha}{\Longrightarrow} q'_1$ et $(p'_1,q'_1) \in \mathcal{S}_m$. On obtient par plusieurs applications de la règle (6), que $q \stackrel{\alpha}{\Longrightarrow} \nu aq'_1$, et $(\nu ap'_1, \nu aq'_1) \in \mathcal{S}_{m+1}$.

Le cas de la "bisimulation forte"est similaire.

□ *Lemma* 37

Lemme 38

1. *Si* $p \sim q$ *alors* $p \parallel r \sim q \parallel r$.
2. *Si* $p \approx q$ *alors* $p \parallel r \approx q \parallel r$.

Preuve Pour le cas "bisimulation faible", nous prouvons que la relation

$$\mathcal{T} \stackrel{def}{=} \{(p \parallel r, q \parallel r) \mid p \approx q \}$$

est une bisimulation faible.

Soit $(p \parallel r, q \parallel r) \in \mathcal{T}$ et $p \parallel r \xrightarrow{\alpha} s$.

On a plusieurs cas.

- $p \parallel r \xrightarrow{\tau} s$. Il y a deux cas possibles:
  - $s = p' \parallel r$ et $p \xrightarrow{\tau} p'$.
    Comme $p \approx q$ alors $\exists q'$ tel que $q \stackrel{\epsilon}{\Longrightarrow} q'$ et $p' \approx q'$. Alors donc $q \parallel r \stackrel{\epsilon}{\Longrightarrow} q' \parallel r$, avec $(p' \parallel r, q' \parallel r) \in \mathcal{T}$.
  - $s = p \parallel r'$ et $r \xrightarrow{\tau} r'$.
    Alors $q \parallel r \xrightarrow{\tau} q \parallel r'$, avec $(p \parallel r', q \parallel r') \in \mathcal{T}$.
- $p \parallel r \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} s$ avec $\tilde{b} \cap fn(p,q,r) = \emptyset$. Il y a deux cas possibles:
  - $s = p' \parallel r'$, $p \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} p'$ et $r \xrightarrow{a\langle\tilde{c}\rangle?} r'$.
    Comme $p \approx q$ alors $q \stackrel{\nu\tilde{b}\bar{a}\tilde{c}}{\Longrightarrow} q'$ et $p' \approx q'$, et donc $q \parallel r \stackrel{\nu\tilde{b}\bar{a}\tilde{c}}{\Longrightarrow} q' \parallel r'$, avec $(p' \parallel r', q' \parallel r') \in \mathcal{T}$.
  - $s = p' \parallel r'$, $p \xrightarrow{a\langle\tilde{c}\rangle?} p'$ et $r \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} r'$.
    Comme $p \approx q$ alors $q \stackrel{\epsilon}{\Longrightarrow} q'' \xrightarrow{a\langle\tilde{c}\rangle?} q''' \stackrel{\epsilon}{\Longrightarrow} q'$ pour certains q'', q''' et $p' \approx q'$, et donc $q \parallel r \stackrel{\nu\tilde{b}\bar{a}\tilde{c}}{\Longrightarrow} q' \parallel r'$, avec $(p' \parallel r', q' \parallel r') \in \mathcal{T}$.
- $p \parallel r \xrightarrow{a\langle\tilde{b}\rangle?} s$.
  $s = p' \parallel r'$, $p \xrightarrow{a\langle\tilde{b}\rangle?} p'$ et $r \xrightarrow{a\langle\tilde{b}\rangle?} r'$.
  Comme $p \approx q$ alors $q \stackrel{\epsilon}{\Longrightarrow} q'' \xrightarrow{a\langle\tilde{b}\rangle?} q''' \stackrel{\epsilon}{\Longrightarrow} q'$ pour certains q'', q''' et $p' \approx q'$, et donc $q \parallel r \stackrel{\epsilon}{\Longrightarrow} q'' \parallel r \xrightarrow{a\langle\tilde{b}\rangle?} q''' \parallel r' \stackrel{\epsilon}{\Longrightarrow} q' \parallel r'$, avec $(p' \parallel r', q' \parallel r') \in \mathcal{T}$.

Le cas de la "bisimulation forte" est similaire.

□ *Lemme* 38

Les lemmes suivantes montrent que la bisimulation étiquetée est une relation assez forte.

Lemme 39

1) $p \sim q$ *implique* $p \sim_b q$.

2) $p \approx q$ *implique* $p \approx_b q$.

Preuve Nous prouvons seulement le deuxième cas. On va montrer que $\approx$ est une bisimulation barbelée faible.

Soit $(p,q) \in \approx$.

- Soit $p \downarrow_c$. Alors $p \xrightarrow{\alpha}$ pour une émission α de sujet c. Comme $p \approx q$, nous obtenons que $q \overset{\alpha}{\Longrightarrow}$ et donc $q \Downarrow_c$.
- Soit $p \xrightarrow{\tau} p'$. Comme $p \approx q$, alors $q \overset{\epsilon}{\Longrightarrow} q'$ avec $(p',q') \in \approx$.

$\square$ *Lemme* 39

Corolaire 40

1) $p \sim q$ *implique* $p \overset{e}{\sim}_b q$.

2) $p \approx q$ *implique* $p \overset{e}{\approx}_b q$.

Preuve

On donne la preuve pour le cas faible.

Soit $p \approx q$.

Soit C un contexte statique quelconque.

Par induction sur la structure de C, et en utilisant les lemmes 37, et 38, nous obtenons $C[p] \approx C[q]$.

Par le lemme 39, nous obtenons $C[p] \approx_b C[q]$. Donc $p \overset{e}{\sim}_b q$.

$\square$ *Corollaire* 40

De la même façon on peut prouver les résultats suivants.

Lemme 41

1) $p \sim q$ *implique* $p \sim_\phi q$.

2) $p \approx q$ *implique* $p \approx_\phi q$.

Corolaire 42

1) $p \sim q$ *implique* $p \overset{e}{\sim}_\phi q$.

2) $p \approx q$ *implique* $p \overset{e}{\approx}_\phi q$.

Nous utilisons des arguments similaires à (Sangiorgi, 1992), pour prouver que la bisimulation étiquetée, la ϕ - équivalence et l'équivalence barbelée coïncident pour une classe importante de processus, plus exactement, les processus qui sont d'image finie.

Un systeme transitionnel étiquetté $(\mathcal{P}, Act, \mapsto)$ est *d'image finie* si pour tout processus p et action α, l'ensemble $\{q \mid p \overset{\alpha}{\mapsto} q\}$ est fini (dans notre cas $\overset{\alpha}{\mapsto}$ peut être la réduction forte ou la réduction faible).

Un processus p est d'image finie si pour toute action α, l'ensemble $\{q \mid p \overset{\alpha}{\mapsto} q\}$ est fini, et pour tout processus q et action α tel que $p \overset{\alpha}{\mapsto} q$, q est d'image finie.

Remarquez que en ce qui concerne la réduction forte, tous les processus sont d'image finie (jusqu'à l'alpha-conversion). Pour des raisons de simplicité des notations, on va donner la preuve pour la version monadique de $b\pi$ -calcul (mais les résultats sont facilement extensibles pour le calcul polyadique).

Lemme 43 *Pour tous les processus d'image finie p et q ,*

1) $p \overset{e}{\sim}_b q$ implique $p \sim q$,

2) $p \overset{e}{\approx}_b q$ implique $p \approx q$.

Preuve

On va donner la preuve seulement pour le cas faible.

Soit $\mathcal{F}$ l'opérateur monotone sur l'ensemble des relations entre les processus $\mathbf{P}(\mathcal{P}_b \times \mathcal{P}_b)$ associé à la définition des bisimulations étiquetées.

Si S est une relation sur l'ensemble $\mathcal{P}_b$ (de processus) alors $(p,q) \in \mathcal{F}(S)$, si les proprietés suivantes sont satisfaites:

1) si $p \xrightarrow{\tau} p'$, alors $\exists q'$ tel que $q \overset{\epsilon}{\Longrightarrow} q'$ et $(p',q') \in S$,

2) si $p \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} p'$ et $\tilde{b} \cap fn(p,q) = \emptyset$, alors $\exists q'$ tel que $q \overset{\nu\tilde{b}\bar{a}\tilde{c}}{\Longrightarrow} q'$ et $(p',q') \in S$,

3) si $p \xrightarrow{a\langle\tilde{b}\rangle?} p'$ alors $\exists q'$ tel que $q \overset{\epsilon}{\Longrightarrow} \bullet \xrightarrow{a\langle\tilde{b}\rangle?} \bullet \overset{\epsilon}{\Longrightarrow} q'$ et $(p',q') \in S$.

Soit $\approx^0 = \mathcal{P}_b \times \mathcal{P}_b$, $\approx^{n+1} = \mathcal{F}(\approx^n)$, et $\approx^\omega = \bigcap_{k<\omega} \approx^k$. Nous appelons $\approx^m$ une m-bisimulation faible. Il est bien connu (Hennessy et Milner, 1985), que pour un systeme transitionel étiquetté d'image finie $\approx^\omega = \approx$.

Il suffit de montrer que $\forall m,\ p \overset{e}{\approx}_b q \Longrightarrow p \approx^m q$.

Pour un ensemble N de paires de noms, on va noter N_i la projection sur le i-ème élément, $n(N) = N_1 \cup N_2$, et on va exiger $N_1 \cap N_2 = \emptyset$. Nous utilisons H^+ (Y^-) comme une abréviation pour $H \cup \{(y_1,y_1')\}$ $(Y \setminus \{(y_1,y_1')\})$. H_1 joue le rôle des noms qui peuvent être connus par p ou q, et Y joue le rôle des noms qui peuvent être appris par p ou q. Pour prouver l'implication, nous construisons une famille de contextes $C^n_{M,H,Y}[\bullet]$ tel que la relation

$$r^m = \Big\{(p,q) \mid \exists n,M,H,Y,\tilde{x} \text{ tel que } n(H) \cap n(Y) = \emptyset,\ |Y| = m,\ fn(p \parallel q) \subseteq H_1 \subseteq M,$$

$$(H_1 \cup Y_1) \subseteq \{\tilde{x}\}, (H_2 \cup Y_2) \cap \{\tilde{x}\} = \emptyset,\ \nu\tilde{x}C^n_{M,H,Y}[p] \approx_b \nu\tilde{x}C^n_{M,H,Y}[q]\Big\}$$

est une m-bisimulation faible.

Soit in, out, new et $\{b_n | n \geq 0\}$ des noms "fraiches", tels que $(\{\tilde{x}\} \cup n(H) \cup n(Y)) \cap (\{b_n | n \geq 0\} \cup \{in,out,new\}) = \emptyset$.

Soit $C^n_{M,H,Y}[\bullet] \stackrel{def}{=} [\bullet] \parallel ASensor_{r,n}\langle M\rangle \parallel GSensor_{s,m}\langle H,Y\rangle$

où $|M| = r$, $|H| = s$, $|Y| = m$,

Le rôle de $GSensor$ est de fournir à p et q tous les communications possibles. Les conditions $fn(p \parallel q) \subseteq H_1$, $fn(p \parallel q) \cap Y_1 = \emptyset$ et $|Y_1| = m$ assurent que $GSensor_{s,m}\langle H,Y\rangle$ peut "satisfaire"toutes les communications possible de p ou q pendant m transitions.

$$GSensor_{s,m} \stackrel{def}{=} (H,Y)$$

$$\sum_{((a,a'),(b,b'))\in H\times H^+} \bar{a}b.(\tau.GSensor_{s+1,m-1}\langle H^+,Y^-\rangle + \tau.W\langle a',b',in\rangle)$$

$$+ \sum_{a\in H_1} a(y).case\langle y,H_1,(R_z)_{z\in H_1},R'_y\rangle, \qquad \text{avec } |H| = s,\ |Y| = m$$

$$W \stackrel{def}{=} (a,b,r)(\bar{a} + \tau.(\bar{b} + \bar{r})),$$

$$R_z \stackrel{def}{=} \tau.GSensor_{s,m-1}\langle H,Y^-\rangle + \tau.W\langle a',z',out\rangle$$

$$R'_y \stackrel{def}{=} \tau.GSensor_{s+1,m-1}\langle H \cup \{(y,y'_1)\},Y^-\rangle + \tau.W\langle a',new,out\rangle$$

$$case\langle y,H,(R_y)_{y\in H},R\rangle \stackrel{def}{=} \begin{cases} R & \text{si } H = \emptyset, \\ \langle y = z\rangle R_z, case\langle y,H',(R_y)_{y\in H'},R\rangle & \text{si } H = H' \cup \{z\}. \end{cases} \tag{4.1}$$

$ASensor$ compte le nombre des actions visibles faits par p, q ou $GSensor_{s,m}$ (d'où la condition $fn(p \parallel q) \subseteq H_1 \subseteq M$).

$$ASensor_{r,n} \stackrel{def}{=} (M)(\overline{b_n} + \sum_{a\in M} a(x).ASensor_{r+1,n+1}\langle M \cup \{x\}\rangle), \qquad \text{avec } |M| = r$$

Nous prouvons que r^m est une m-bisimulation faible.

Soit $(p,q) \in r^m$. Alors $\exists n,M,H,Y,\tilde{x}$ tel que $n(H) \cap n(Y) = \emptyset$, $|Y| = m$, $fn(p \parallel q) \subseteq H_1 \subseteq M$, $(H_1 \cup Y_1) \subseteq \{\tilde{x}\}$, $(H_2 \cup Y_2) \cap \{\tilde{x}\} = \emptyset$, $\nu\tilde{x}C^n_{M,H,Y}[p] \approx_b \nu\tilde{x}C^n_{M,H,Y}[q]$.

[1] Supposons $p \xrightarrow{\tau} p'$.

Alors $\nu\tilde{x}C^n_{M,H,Y}[p] \xrightarrow{\tau} \nu\tilde{x}C^n_{M,H,Y}[p']$. Il faut qu'il existe une transition

$$\nu\tilde{x}C^n_{M,H,Y}[q] \stackrel{\epsilon}{\Longrightarrow} t \approx_b \nu\tilde{x}C^n_{M,H,Y}[p']$$

Comme $\nu\tilde{x}C^n_{M,H,Y}[p'] \downarrow_{b_n}$, on peut conclure que $ASensor$ reste inchangé, et comme $H_1 \subseteq M$, on obtient que $t \equiv \nu\tilde{x}C^n_{M,H,Y}[q']$ où $q \stackrel{\epsilon}{\Longrightarrow} q'$.

[2] Supposons $p \xrightarrow{\bar{a}b} p'$.

Alors

$$\nu\tilde{x}C^n_{M,H,Y}[p] \xrightarrow{\tau} \nu\tilde{x}(p' \parallel ASensor_{r+1,n+1}\langle M \cup \{b\}\rangle \parallel case\langle b,H_1,(R_z)_{z\in H_1},R'_b\rangle)$$

$$= t_1 \xrightarrow{\tau} \nu\tilde{x}(p' \parallel ASensor_{r+1,n+1}\langle M \cup \{b\}\rangle \parallel GSensor_{s,m-1}\langle H,Y^-\rangle)$$

$$= t_2 = \nu\tilde{x}C^{n+1}_{M\cup\{b\},H,Y^-}[p']$$

Il faut qu'il existe des processus u_1, u_2 dérivés de $\nu\tilde{x}C^n_{M,H,Y}[q]$ tels que

$$\nu\tilde{x}C^n_{M,H,Y}[q] \stackrel{\epsilon}{\Longrightarrow} u_1 \approx_b t_1$$

$$u_1 \stackrel{\epsilon}{\Longrightarrow} u_2 \approx_b t_2$$

Premier pas: $\nu\tilde{x}C^n_{M,H,Y}[p] \xrightarrow{\tau} t_1$. La configuration $\nu\tilde{x}C^n_{M,H,Y}[q]$ doit faire au moins un pas, car $\nu\tilde{x}C^n_{M,H,Y}[q] \downarrow_{b_n}$ tandis que $t_1 \not\Downarrow_{b_n}$. Comme $t_1 \downarrow_{b_{n+1}}$, p et $GSensor$ ne peuvent pas faire plus d'un pas visible. On obtient que u_1 doit avoir la forme suivante:

$$\nu\tilde{x'}(q'' \parallel ASensor_{r+1,n+1}\langle M \cup \{b_1\}\rangle \parallel v'')$$

pour certains b_1,q'', v'', $\nu\tilde{x'}$, $q \stackrel{\alpha'}{\Longrightarrow} q''$, $GSensor_{s,m}\langle H,Y\rangle \xrightarrow{\alpha''} v''$, où α'' et α' sont des actions complémentaires. Le fait que $t_1 \Longrightarrow\Downarrow_{\{a',b',out,b_{n+1}\}}$ implique que α' est une émission, et α'' doit être la réception correspondante; en plus, α' ne peut pas être une émission bornée. On obtient alors $\alpha' = \overline{a_1}b_1$, où $\{a,b\} = \{a_1,b_1\}$. Si on suppose $a_1 = b$, $b_1 = a$, on devrait avoir $t_1 \Longrightarrow\Downarrow_{\{b',out,b_{n+1}\}}$, dérivation que u_1 ne peut pas simuler. Pour le deuxième pas, on obtient facilement que u_2 doit avoir la forme $\nu\tilde{x}C^{n+1}_{M\cup\{b\},H,Y^-}[q']$ où $q'' \stackrel{\epsilon}{\Longrightarrow} q'$ et $v'' \xrightarrow{\tau} GSensor_{s,m-1}\langle H,Y^-\rangle$

[3] Supposons $p \xrightarrow{\nu b\bar{a}b} p'$.

Alors

$$\nu\tilde{x}C^n_{M,H,Y}[p] \xrightarrow{\tau} \nu b\nu\tilde{x}(p' \parallel ASensor_{r+1,n+1}\langle M \cup \{b\}\rangle \parallel case\langle b,H_1,(R_z)_{z\in H_1},R'_b\rangle)$$

$$= t_1 \xrightarrow{\tau} \nu b\nu\tilde{x}(p' \parallel ASensor_{r+1,n+1}\langle M \cup \{b\}\rangle \parallel GSensor_{s+1,m-1}\langle H \cup \{(b,y'_1)\},Y^-\rangle)$$

$$= t_2 = \nu b\nu\tilde{x}C^{n+1}_{M\cup\{b\},H\cup\{(b,y'_1)\},Y^-}[p']$$

Il faut qu'il existe des processus u_1, u_2 dérivés de $\nu\tilde{x}C^n_{M,H,Y}[q]$ tels que

$$\nu\tilde{x}C^n_{M,H,Y}[q] \stackrel{\epsilon}{\Longrightarrow} u_1 \approx_b t_1$$

$$u_1 \stackrel{\epsilon}{\Longrightarrow} u_2 \approx_b t_2$$

Le premier pas: $\nu\tilde{x}C^n_{M,H,Y}[p] \xrightarrow{\tau} t_1$. La configuration $\nu\tilde{x}C^n_{M,H,Y}[q]$ doit faire au moins un pas, car $\nu\tilde{x}C^n_{M,H,Y}[q] \downarrow_{b_n}$ et $t_1 \not\Downarrow_{b_n}$. Comme $t_1 \downarrow_{b_{n+1}}$, p et $GSensor$ ne peuvent pas faire plus d'un pas visible. On obtient alors que u_1 doit avoir la forme suivante:

$$\nu\tilde{x'}(q'' \parallel ASensor_{r+1,n+1}\langle M \cup \{b_1\}\rangle \parallel v'')$$

pour certains b_1,q'', v'', $\tilde{x'}$, $q \stackrel{\alpha'}{\Longrightarrow} q''$, $GSensor_{s,m}\langle H,Y\rangle \xrightarrow{\alpha''} v''$, où α'' et α' sont des actions complémentaires. Comme $t_1 \Longrightarrow\Downarrow_{\{a',new,out,b_{n+1}\}}$ on déduit que α' est une émission, et α'' est la réception correspondante; de plus, α' doit être une émission bornée. Alors $\alpha' = \nu b\overline{a_1}b$, où nécessairement $a = a_1$. Pour le deuxième pas, on prouve facilement que u_2 doit avoir la forme $\nu b\nu\tilde{x}C^{n+1}_{M\cup\{b\},H\cup\{(b,y'_1)\},Y^-}[q']$ où $q'' \stackrel{\epsilon}{\Longrightarrow} q'$ et $v'' \xrightarrow{\tau} GSensor_{s+1,m-1}\langle H \cup \{(b,y'_1)\},Y^-\rangle$

[4] Supposons $p \xrightarrow{a\langle\tilde{b}\rangle?} p'$.

Alors

$$\nu\tilde{x}C^n_{M,H,Y}[p] \xrightarrow{\tau}$$

$$\nu\tilde{x}(p' \parallel ASensor_{r+1,n+1}\langle M \cup \{b\}\rangle \parallel (\tau.GSensor_{s+1,m-1}\langle H^+,Y^-\rangle + \tau.W\langle a',b',in\rangle))$$

$$= t_1 \xrightarrow{\tau} \nu\tilde{x}(p' \parallel ASensor_{r+1,n+1}\langle M \cup \{b\}\rangle \parallel GSensor_{s+1,m-1}\langle H^+,Y^-\rangle)$$

$$= t_2 = \nu\tilde{x}C^{n+1}_{M\cup\{b\},H^+,Y^-}[p']$$

Il faut qu'il existe des processus u_1, u_2 dérivés de $\nu\tilde{x}C^n_{M,H,Y}[q]$ tels que

$$\nu\tilde{x}C^n_{M,H,Y}[q] \stackrel{\epsilon}{\Longrightarrow} u_1 \approx_b t_1$$

$$u_1 \stackrel{\epsilon}{\Longrightarrow} u_2 \approx_b t_2$$

Le premier pas: $\nu\tilde{x}C^n_{M,H,Y}[p] \xrightarrow{\tau} t_1$. La configuration $\nu\tilde{x}C^n_{M,H,Y}[q]$ doit faire au moins un pas, car $\nu\tilde{x}C^n_{M,H,Y}[q] \downarrow_{b_n}$ et $t_1 \not\Downarrow_{b_n}$. Comme $t_1 \downarrow_{b_{n+1}}$, p ou $GSensor$ ne peuvent pas faire plus d'un pas visible. On obtient que u_1 doit avoir la forme:

$$\nu\tilde{x'}(q'' \parallel ASensor_{r+1,n+1}\langle M \cup \{b_1\}\rangle \parallel v'')$$

pour certains b_1, q'', v'', $\nu\tilde{x}'$, $q \stackrel{\alpha'}{\Longrightarrow} q''$, $GSensor_{s,m}\langle H,Y\rangle \stackrel{\alpha''}{\longrightarrow} v''$, où α'' et α' sont des actions complémentaires. Comme $t_1 \Longrightarrow\Downarrow_{\{a',b',in,b_{n+1}\}}$ alors α' doit être soit une émission soit un ignore, tandis que α'' est l'émission correspondante. Donc $\alpha' = a_1\langle\tilde{b_1}\rangle?$, $\alpha'' = \overline{a_1}b_1$, où nécessairement $\{a,b\} = \{a_1,b_1\}$. Si on suppose $a_1 = b$, $b_1 = a$, on devrait avoir $t_1 \Longrightarrow\Downarrow_{\{b',in,b_{n+1}\}}$, transition que u_1 ne peut pas simuler. Pour le deuxième pas, on prouve facilement que u_2 doit avoir la forme $\nu\tilde{x}C^{n+1}_{M\cup\{b\},H^+,Y^-}[q']$ avec $q'' \stackrel{\epsilon}{\Longrightarrow} q'$ et

$v'' \stackrel{\tau}{\longrightarrow} GSensor_{s+1,m-1}\langle H^+,Y^-\rangle$

□ *Lemme* 43

Théoreme 44 *Pour tous les processus d'image finie p et q,*

1) $p \stackrel{e}{\sim}_b q$ *si et seulement si* $p \stackrel{e}{\sim}_\phi q$ *si et seulement si* $p \sim q$

2) $p \stackrel{e}{\approx}_b q$ *si et seulement si* $p \stackrel{e}{\approx}_\phi q$ *si et seulement si* $p \approx q$.

Preuve

L'assertion du théorème est une conséquence immédiate des Corollaires 31 et 42 et du Lemme 43.

□ *Theoreme* 44

4.1.4 Application

Nous allons utiliser les bisimulations étiquetées pour prouver la correction d'un protocole FIFO. Un protocole FIFO (Hadzilacos et Toueg, 1994) assure la réception (par tous les membres d'un groupe) en bon ordre des messages provenant d'une même source (en supposant qu'on dispose des primitives de diffusion asynchrones et qui ne garantissent pas l'ordre des réceptions). Disposer de primitives de diffusion qui garantissent l'ordre de réception des messages peut avoir de nombreuses applications; par exemple, dans un système de réservation distribué, le message d'annulation d'une réservation ne doit pas parvenir à un site qui n'avait pas reçu auparavant une demande de réservation. Dans cet exemple, nous nous contentons d'un protocole simple: il ne garantit pas que tous les messages arrivent à destination, mais tout simplement que deux messages diffusés par une même source, sont reçus dans le bon ordre (l'ordre de leurs émissions) par tous les autres membres du groupe qui faisaient déjà parti du groupe au moment de l'émission des deux messages.

Nous utilisons l'abreviation

$\langle x = a_1,a_2\rangle p_1,p_2,p_3 \stackrel{def}{=} \langle x = a_1\rangle\ p_1, (\langle x = a_2\rangle p_2,p_3)$.

Le processus "*Server*" est le gestionnaire du groupe. Il attend des requêtes sur le canal *req* qui sert d'identificateur pour le groupe (l'adresse du groupe). L'argument *replay* d'une requête représente un alias d'un client qui demande d'adhérer au groupe. Chaque client, dès qu'il joint le groupe, reçoit un identificateur unique *id* qui représente en quelque sorte son "numéro d'ordre" dans le groupe.

$$Server\langle req,net,s_0\rangle \stackrel{def}{=} req(replay).(Server\langle req,net,s_0\rangle \parallel \nu out\, \nu in\, \nu id\, \overline{replay}[out,in,id].Fifo\langle out,in,id,net,s_0\rangle)$$

Pour chaque membre du groupe, le "système" génère (dès l'adhésion du nouveau client), deux interfaces. $Fifo_{out}\langle out,id,net,s_0\rangle$ sert pour les messages "injectés" dans le réseau par le client. $Fifo_{in}\langle in,net,loc\rangle$ est l'interface utilisée par le client pour recevoir les messages diffusés par les autres membres du groupe.

$$Fifo\langle out,in,id,net,s_0\rangle \stackrel{def}{=} Fifo_{out}\langle out,id,net,s_0\rangle \parallel \nu loc\,(Fifo_{in}\langle in,net,loc\rangle \parallel ASend\langle loc,s_0\rangle)$$

Pour chaque émission du client, le "système" ajoute au message "une en-tête" qui contient plusieurs informations: ("*id*" - l'identité de l'"émetteur, "s_m" - l'identificateur du message antérieur émis par le même client, "$s_{m'}$" - un identificateur unique qui est l'identité du paquet actuel), et ensuite il "injecte" le message "dans le réseau" (simulé par une émission sur le canal *net* qui peut être vu comme "un bus virtuel" sur lequel les messages deviennent accessibles à tous les membres du groupe).

$$Fifo_{out}\langle out,id,net,s_m\rangle \stackrel{def}{=} out(m').\nu s_{m'}\ (Fifo_{out}\langle out,id,net,s_{m'}\rangle \parallel \overline{net}[id,m',s_{m'},s_m])$$

Pour chaque paquet reçu (sur le bus "*net*"), l'interface récepteur d'un client génère "une cellule" qui tient compte des informations contenues dans l'en-tête du paquet:

$$Fifo_{in}\langle in,net,l\rangle \stackrel{def}{=} net(id,m',s_{m'},s_m).(Fifo_{in}\langle in,net,l\rangle \parallel Cell\langle id,m',s_{m'},s_m,in,l\rangle)$$

L'asynchronisme du réseau est simulé au niveau des récepteurs (le message n'est pas reçu directement par le client).

L'accès du client aux messages reçus est bloqué jusqu'au moment où le message précédent a été consommé par le client.

$$Cell\langle id,m',s_{m'},s_m,in,l\rangle \stackrel{def}{=} l(y).\langle y=s_m\rangle Data\langle id,s_{m'},in,l\rangle,Cell\langle id,m',s_{m'},s_m,in,l\rangle$$
$$Data\langle id,m',s_{m'},in,l\rangle \stackrel{def}{=} \overline{in}[id,m'].ASend\langle l,s_{m'}\rangle$$
$$ASend\langle l,s\rangle \stackrel{def}{=} \bar{l}s.ASend\langle l,s\rangle$$

Pour prouver la correction du protocole (le fait que deux messages consecutives envoyés par un client ne peuvent pas être reçus par un autre client dans l'ordre inverse), on va utiliser trois clients de test: un "émetteur" et deux "récepteurs".

Le client "Cl_0" (d'identité id_0), diffuse au groupe (à travers son interface "émettrice") les deux messages m_1 et m_2. La condition $out_0 \notin fn(Q)$ plus le fait que la restriction νout_0 a la portée initiale $Cl_0\langle out_0,id_0,m_1,m_2\rangle \parallel Fifo\langle out_0,in_0,id_0,net,s_0\rangle$, garantissent que les premiers messages envoyés par Cl_0 sont m_1 et m_2 dans cette ordre. La condition $net \notin fn(P,Q)$

exprime le fait que le client Cl_0 ne peut pas interagir "directement" avec les autres clients, mais seulement à travers son interface $Fifo\langle out_0,in_0,id_0,net,s_0\rangle$. A par ces conditions, il n'y a aucune autre restriction pour le comportement de Cl_0 (P et Q sont des procesus quelconques tel que $out_0 \notin fn(Q)$ et $net \notin fn(P,Q)$).

$$Cl_0\langle out_0,id_0,m_1,m_2\rangle \stackrel{def}{=} \overline{out_0}m_1.\overline{out_0}m_2.P \parallel Q \qquad out_0 \notin fn(Q), \quad net \notin fn(P,Q)$$

Le client "Cl_1" teste tous les messages reçus. Par contre il finit son exécution soit par un message de succès s'il reçoit le message m_1 provenant du client Cl_0, soit par un message d'erreur s'il reçoit d'abord le message m_2 provenant du client Cl_0.

$$\begin{aligned} Cl_1\langle in_1,id_1,id_0,m_1,m_2,ok,err\rangle \stackrel{def}{=}\ & in_1(i,m).\langle i = id_0\rangle \\ & (\langle m = m_2,m_1\rangle\overline{err},\overline{ok},Cl_1\langle in_1,id_1,id_0,m_1,m_2,ok,err\rangle), \\ & Cl_1\langle in_1,id_1,id_0,m_1,m_2,ok,err\rangle \end{aligned}$$

Le client "Cl_2" teste tous les messages reçus. Il finit son exécution par un message de succès quand il reçoit le message m_1 provenant du client Cl_0 (il "suppose" qu'il ne peut pas recevoir comme premier message de Cl_0 un autre message que m_1).

$$\begin{aligned} Cl_2\langle in_2,id_2,id_0,m_1,m_2,ok,err\rangle \stackrel{def}{=}\ & in_2(i,m).\langle i = id_0\rangle \\ & (\langle m = m_1\rangle\overline{ok},Cl_2\langle in_2,id_2,id_0,m_1,m_2,ok,err\rangle), \\ & Cl_2\langle in_2,id_2,id_0,m_1,m_2,ok,err\rangle \end{aligned}$$

Pour prouver la correction du protocole, on va utiliser deux configurations de test différentes: la première utilise l'émetteur et le premier client, la deuxième utilisant l'émetteur et le deuxième client.

$$\begin{aligned} A_1 \stackrel{def}{=}\ & \nu net\ \nu s_0\{Server\langle req,net,s_0\rangle \parallel \\ & \qquad \nu out_0\ \nu in_0\ \big(Cl_0\langle out_0,id_0,m_1,m_2\rangle \parallel Fifo\langle out_0,in_0,id_0,net,s_0\rangle\big) \\ & \qquad \parallel\ \ \nu out_1\ \nu in_1\ \ \big(Cl_1\langle in_1,id_1,id_0,m_1,m_2,ok,err\rangle \parallel Fifo\langle out_1,in_1,id_1,net,s_0\rangle\big)\} \end{aligned}$$

$$\begin{aligned} A_2 \stackrel{def}{=}\ & \nu net\ \nu s_0\{Server\langle req,net,s_0\rangle \parallel \\ & \qquad \nu out_0\ \nu in_0\ \big(Cl_0\langle out_0,in_0,m_1,m_2\rangle \parallel Fifo\langle out_0,in_0,id_0,net,s_0\rangle\big) \\ & \qquad \parallel\ \ \nu out_2\ \nu in_2\ \ \big(Cl_2\langle in_2,id_2,id_0,m_1,m_2,ok,err\rangle \parallel Fifo\langle out_2,in_2,id_2,net,s_0\rangle\big)\} \end{aligned}$$

Le fait que $A_1 \approx A_2$ (preuve donnée in extenso dans l'annexe C), montre la correction du protocole: deux messages diffusés par un membre du groupe sont reçus dans le même ordre par tous les autres clients qui étaient déjà membres du groupe (remarquez que l'on ne garantit pas que tous les messages diffusés dans le groupe sont reçus par tous les autres membres, mais seulement le fait que l'ordre des messages n'est pas inversé). Cette preuve est très générale: les deux configurations ont le même comportement indépendemment de l'environnement (les autres clients du groupe qui peuvent recevoir les messages et insérer eux aussi des paquets dans le réseau).

Il semble difficile de prouver (en gardant le même niveau d'abstraction et de généralité) ce genre de protocoles dans un modèle qui ne dispose que de communication point à point (comme le π-calcul) alors que l'application utilise naturellement comme primitive de communication la diffusion.

4.2 Congruences

4.2.1 Définitions

Dans cette section nous considérons la congruence induite par la bisimulation barbelée forte. Notre but sera de définir une relation R sur l'ensemble $\mathcal{P}_b$ de processus tel que si $p \; R \; q$ alors p et q sont observables sur le même ensemble de canaux, l'observabilité est préservée par la réduction, et de plus, quand ils sont placés dans un contexte arbitraire C, p et q ne peuvent pas être distingués ($C[p] \; R \; C[q]$).

Définition 45 Congruence barbelée forte
Deux processus p et q sont congruents dans le sens de la **congruence barbelée forte**, *noté* $p \overset{c}{\sim}_b q$, *si pour tout contexte* C, $C[p] \sim_b C[q]$.

Évidemment, $\overset{c}{\sim}_b$ est par définition une congruence. Mais comme pour l'équivalence barbelée, cette définition est plus appropriée pour prouver la non-congruence que la congruence des processus. Donc, nous nous sommes intéressés à la recherche d'une caractérisation de cette congruence qui soit basée sur les transitions étiquetées.

Malheureusement, $\sim$ n'est pas la caractérisation recherchée, affirmation certifiée par la remarque suivante.

Remarque 46

i) $\sim$ *n'est pas préservée par le choix. Nous avons* $a \sim b$, *mais* $a + \bar{c} \not\sim b + \bar{c}$.

ii) $\sim$ *n'est pas préservée par la substitution. Soit* $p \overset{def}{=} \bar{x}.y.\bar{c} + y.(\bar{x} \parallel \bar{c})$ *et* $q \overset{def}{=} \bar{x} \parallel y.\bar{c}$. *Alors* $p \sim q$, *mais* $p[x/y] \not\sim q[x/y]$.

iii) $\sim$ *n'est pas préservée par l'opérateur de préfix. C'est une conséquence directe du point précèdent.*

Contrairement à la bisimulation barbelée dans le π-calcul, $\sim$ n'est pas préservée par le choix. Nous utilisons des idées de (Hennessy et Rathke, 1995) et (Parrow et Sangiorgi, 1995) pour obtenir une relation de congruence qui ne nécessite pas une clôture aux contextes.

Définition 47 *Soit $\sim_+$ une relation symétrique qui satisfait les conditions suivantes:*

1) *si $p \xrightarrow{\tau} p'$, alors $\exists q'$ tel que $q \xrightarrow{\tau} q'$ et $p' \sim q'$,*
2) *si $p \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} p'$ et $\tilde{b} \cap fn(p,q) = \emptyset$, alors $\exists q'$ tel que $q \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} q'$ et $p' \sim q'$,*
3) *si $p \xrightarrow{a\langle\tilde{b}\rangle} p'$ alors $\exists q'$ tel que $q \xrightarrow{a\langle\tilde{b}\rangle} q'$ et $p' \sim q'$.*

Soit $\sim_c$ défini par $p \sim_c q$ si $p\sigma \sim_+ q\sigma$ pour toutes les substitutions σ.

Remarque 48

- $\sim_c \subseteq \sim_+ \subseteq \sim$.
- *les inclusions sont strictes.*
- *$\sim_+$ est préservée par $+$, ν et $\|$.*

Les inclusions suivent directement des définitions. Pour prouver que la première inclusion est stricte, il suffit de prendre $p \stackrel{def}{=} \bar{x}.y.\bar{c} + y.(\bar{x} \parallel \bar{c})$ et $q \stackrel{def}{=} \bar{x} \parallel y.\bar{c}$. Alors $p \sim_+ q$, mais $p \not\sim_c q$.

Pour prouver que la deuxième inclusion est stricte, il suffit de prendre $p \stackrel{def}{=} a$ et $q \stackrel{def}{=} b$. Alors $p \sim q$, mais $p \not\sim_+ q$.

Le fait que $\sim_+$ est préservée par ν et $\|$ peut être prouvé par une analyse par cas, de la même façon que pour $\sim$ dans les Lemmes 37 et 38. Le fait que $\sim_+$ est préservée par $+$ est prouvé par une analyse par cas dans l'annexe D.

Lemme 49 *$\sim_c$ est préservée par le préfixe, la restriction, la somme, le conditionnel, et le parallélisme.*

Preuve

- $p \sim_c q \Longrightarrow \alpha.p \sim_c \alpha.q$. pour tout préfixe α.
  - $\alpha = \tau$.
    Soit σ une substitution. Alors $p \sim_c q \Longrightarrow p\sigma \sim_+ q\sigma \Longrightarrow p\sigma \sim q\sigma \Longrightarrow \tau.(p\sigma) \sim_+ \tau.(q\sigma) \Longrightarrow (\tau.p)\sigma \sim_+ (\tau.q)\sigma$.
    Donc $\tau.p \sim_c \tau.q$.
  - $\alpha = \bar{a}b$.
    Soit σ une substitution. Alors $p \sim_c q \Longrightarrow p\sigma \sim_+ q\sigma \Longrightarrow p\sigma \sim q\sigma \Longrightarrow (\bar{a}\tilde{b})\sigma.(p\sigma) \sim_+ (\bar{a}\tilde{b})\sigma.(q\sigma) \Longrightarrow (\bar{a}\tilde{b}.p)\sigma \sim_+ (\bar{a}\tilde{b}.q)\sigma$.
    On obtient $\bar{a}\tilde{b}.p \sim_c \bar{a}\tilde{b}.q$.
  - $\alpha = a(\tilde{b})$.
    Soit σ une substitution. Alors $p \sim_c q \Longrightarrow p\sigma \sim_+ q\sigma \Longrightarrow p\sigma \sim q\sigma \Longrightarrow (a(\tilde{b}).p)\sigma \sim_+ (a(\tilde{b}).q)\sigma$.

Donc $a(\tilde{b}).p \ \sim_c \ a(\tilde{b}).q$.

– $p \sim_c q \Longrightarrow \nu ap \sim_c \nu aq$ pour tout canal $a \in Ch_b$.
Soit σ une substitution. On peut supposer $a \notin (prdom(\sigma) \cup prcod(\sigma))$. Alors $p \sim_c q \Longrightarrow p\sigma \ \sim_+ \ q\sigma \Longrightarrow$ (par la Remarque 48) $(\nu ap)\sigma \ \sim_+ \ (\nu aq)\sigma$.
Donc $\nu ap \ \sim_c \ \nu aq$.

– $p \sim_c q \Longrightarrow p + r \sim_c q + r$.
Soit σ une substitution. Alors $p \sim_c q \Longrightarrow p\sigma \ \sim_+ \ q\sigma \Longrightarrow$ (par la Remarque 48) $p\sigma + r\sigma \ \sim_+ \ q\sigma + r\sigma \Longrightarrow (p+r)\sigma \ \sim_+ \ (q+r)\sigma$.
On obtient $p + r \ \sim_c \ q + r$.

– $p \sim_c q \Longrightarrow \langle x = y\rangle p,r \sim_c \langle x = y\rangle q,r \wedge \langle x = y\rangle r,p \sim_c \langle x = y\rangle r,q$.
Soit σ une substitution. On peut prouver les implications par une analyse par cas (en fonction de la relation entre $\sigma(x)$ et $\sigma(y)$), en tenant compte du fait que $\sim_+$ est réflexive.

– $p \sim_c q \Longrightarrow p \parallel r \sim_c q \parallel r$ pour tout processus $p \in \mathcal{P}_b$.
Soit σ une substitution. Alors $p \sim_c q \Longrightarrow p\sigma \ \sim_+ \ q\sigma \Longrightarrow$ (par la Remarque 48) $p\sigma \parallel r\sigma \ \sim_+ \ q\sigma \parallel r\sigma \Longrightarrow (p \parallel r)\sigma \ \sim_+ \ (q \parallel r)\sigma$
On obtient $p \parallel r \ \sim_c \ q \parallel r$.

□ *Lemme* 49

Pour un processus E qui contient un identificateur libre X (qui n'est pas sous la portée d'un "rec"), et un processus p, nous notons par $E(p)$ le processus obtenu de E en remplacent $X\langle\tilde{y}\rangle$ par $p\langle\tilde{y}/\tilde{x}\rangle$ où $\tilde{x}$ représente les noms libres dans p. Par exemple, si $p \stackrel{def}{=} (x_1,x_2)(\bar{x_1}.x_2 \parallel \bar{x_2})$ et $E \stackrel{def}{=} \bar{a}b.X\langle a,b\rangle + \nu c\bar{a}c.X\langle c,b\rangle$, alors $E(p) = \bar{a}b.(\bar{a}.b \parallel \bar{b}) + \nu c\bar{a}c.(\bar{c}.b \parallel \bar{b})$.

Définition 50 *Soit E et F deux processus qui contiennent un identificateur libre X. Alors $E \sim F$ (respectivement $E \sim_+ F$, $E \sim_c F$) si $E(p) \sim F(p)$ (respectivement $E(p) \sim_+ F(p)$, $E(p) \sim_c F(p)$) pour tout processus p.*

Pour le cas des processus, nous allons utiliser les bisimulations jusqu'au $\sim_+$.

Définition 51 *Une relation symétrique S est une* **bisimulation jusqu'au** $\sim_+$ *si pour chaque $(p,q) \in S$, on a*

1) si $p \xrightarrow{\tau} p'$, alors $\exists q'$ tel que $q \xrightarrow{\tau} q'$ et $(p',q') \in \sim S \sim$,
2) si $p \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} p'$ et $\tilde{b} \cap fn(p,q) = \emptyset$, alors $\exists q'$ tel que $q \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} q'$ et $(p',q') \in \sim S \sim$,
3) si $p \xrightarrow{a\langle\tilde{b}\rangle} p'$ alors $\exists q'$ tel que $q \xrightarrow{a\langle\tilde{b}\rangle} q'$ et $(p',q') \in \sim S \sim$.

A noter que dans la définition 51 nous exigeons $(p',q') \in \sim S \sim$ et non pas $(p',q') \in \sim_+ S \sim_+$.

Le lemme suivant permet de reduire la taille des relations utilisées pour montrer que deux processus sont bisimilaires au sens de $\sim_+$.

Lemme 52 *Si S est une bisimulation jusqu'au $\sim_+$ alors $S \subseteq \sim_+$.*

Preuve

Soit S une une bisimulation jusqu'au $\sim_+$. Alors S satisfait les conditions de la Définition 35, et donc $S \subseteq\sim$.

Supposons maintenant que $(p,q) \in S$ et soit $p \xrightarrow{\alpha} p'$, où α satisfait les conditions de la Définition 47. Alors par la Définition 51, on obtient $q \xrightarrow{\alpha} q'$ et $(p',q') \in \sim S \sim$.

Comme $S \subseteq\sim$ et $\sim$ est tranzitive, on obtient $(p',q') \in \sim$.

$\square$ *Lemme* 52

Lemme 53 *Soit E et F deux processus qui contiennent un identificateur libre X. Si $E \sim_c F$, alors $(rec\ X\langle\tilde{x}\rangle.E)\langle\tilde{x}\rangle \sim_c (rec\ X\langle\tilde{x}\rangle.F)\langle\tilde{x}\rangle$.*

Soit $p \stackrel{def}{=} (rec\ X\langle\tilde{x}\rangle.E)\langle\tilde{x}\rangle$, $q \stackrel{def}{=} (rec\ X\langle\tilde{x}\rangle.F)\langle\tilde{x}\rangle$.

Nous montrons que la relation

$$\mathcal{C} \stackrel{def}{=} \{(G(p),G(q)) \mid G \text{ contient seulement l'identificateur } X\ \}$$

est une bisimulation jusqu'au $\sim_+$. Par le Lemme 52 on obtient $\mathcal{C} \subseteq \sim_+$. En choisissant $G \equiv X\langle\tilde{z}\rangle$, nous obtenons que $p\langle\tilde{z}\rangle \sim_+ q\langle\tilde{z}\rangle$ pour tout $\tilde{z}$, et donc $p \sim_c q$, qui valide l'assertion du lemme.

Soit $G(p) \xrightarrow{\alpha} p'$ $(*)$.

Nous allons prouver l'existence d'une transition correspondante $G(q) \xrightarrow{\alpha} q'$ qui satisfait les conditions de la Définition 51 par induction structurelle sur l' inférence de la transition $(*)$. Nous présentons seulement quelques cas.

- $G(p) \xrightarrow{\alpha} p'$ par l'application de la règle (10).
  Alors $G = X\langle\tilde{y}\rangle$, et par une déduction plus courte, $E(p)[\tilde{y}/\tilde{x}] \equiv E[\tilde{y}/\tilde{x}](p) \xrightarrow{\alpha} p'$.
  Par induction $E[\tilde{y}/\tilde{x}](q) \xrightarrow{\alpha} q''$ avec $p' \sim \mathcal{C} \sim q''$. Comme $E \sim_c F$, on obtient
  $E[\tilde{y}/\tilde{x}](q) \sim_+ F[\tilde{y}/\tilde{x}](q)$; donc $F[\tilde{y}/\tilde{x}](q) \xrightarrow{\alpha} q' \sim q''$. Par l'application de la règle (10)
  $G(q) = q\langle\tilde{y}\rangle \xrightarrow{\alpha} q'$. La transitivité de $\sim$ implique $p' \sim \mathcal{C} \sim q'$.
- $G(p) \xrightarrow{\alpha} p'$ par l'application d'une règle qui concerne le parallélisme (11), (12), (13) ou leurs symétriques.
  Alors $G = G_1 \parallel G_2$, et par une déduction plus courte, $G_i(p) \xrightarrow{\alpha_i} p'_i$ pour certains α_i appropriés. Par induction $G_i(q) \xrightarrow{\alpha_i} q'_i$ où $p'_i \sim \mathcal{C} \sim q'_i$. Donc $p'_i \sim H_i(p)\ \mathcal{C}\ H_i(q) \sim q'_i$, et par le Lemme 38 nous obtenons que
  $(p'_1 \parallel p'_2) \sim (H_1(p) \parallel H_2(p))\ \mathcal{C}\ (H_1(q) \parallel H_2(q)) \sim (q'_1 \parallel q'_2)$.

□ *Lemme* 53

Corolaire 54 *$\sim_c$ est préservée par la récursion.*

Preuve

Soit $p \sim_c q$ et soit $D[\bullet] \stackrel{def}{=} (recX\langle\tilde{x}\rangle.C[\bullet])\langle\tilde{y}\rangle$ un contexte tel que $D[p]$ et $D[q]$ sont bien formés ($fn(p \parallel q) \subseteq \tilde{x}$).

Par le Lemme 49 on obtient $C[p](r) \sim_c C[q](r)$ pour tout processus r, et donc $C[p] \sim_c C[q]$ au sens de la Définition 50. En utilisant le Lemme 53, nous obtenons $(recX\langle\tilde{x}\rangle.C[p])\langle\tilde{y}\rangle \sim_c (recX\langle\tilde{x}\rangle.C[q])\langle\tilde{y}\rangle$ et donc $D[p] \sim_c D[q]$

□ *Corollaire* 54

Par le Lemme 49 et le Corollaire 54, nous obtenons que $\sim_c$ est une congruence.

On a donc:

Théoreme 55 *$\sim_c$ est une congruence.*

En utilisant certains contextes (bien choisis), nous pouvons prouver que $\sim_c$ et $\stackrel{c}{\sim}_b$ coïncident.

Théoreme 56 $\sim_c = \stackrel{c}{\sim}_b$.

Preuve

- $\sim_c \subseteq \stackrel{c}{\sim}_b$

  Soit C un contexte arbitraire et soit $p \sim_c q$. Par le Théoreme 55, $C[p] \sim_c C[q]$. Par la Remarque 48, on obtient $C[p] \sim C[q]$ et par le Lemme 39 $C[p] \sim_b C[q]$.

  On obtient $p \stackrel{c}{\sim}_b q$.
- $\stackrel{c}{\sim}_b \subseteq \sim_c$

  D'abord nous remarquons que dans le cas de la bisimulation forte, tout processus est d'image finie. Donc le premier point 1) du Théorème 44 peut être lu

  "pour tous les processus p et q:

  $$p \stackrel{e}{\sim}_b q \quad \text{ssi} \quad p \stackrel{e}{\sim}_\phi q \quad \text{ssi} \quad p \stackrel{e}{\sim} q." \tag{4.2}$$

  Soit $p \stackrel{c}{\sim}_b q$ et soit $C_1[\bullet] \stackrel{def}{=} u(x_1).u(x_2)\ldots u(x_n).([\bullet] + \sum_{i=1}^{n} x_i(x).\bar{v})$, où $fn(p \parallel q) = \{x_1, x_2, \ldots, x_n\}$, et $fn(p \parallel q) \cap \{u, v\} = \emptyset$. Nous prouvons que

  $$C_1[p] \sim C_1[q] \Longrightarrow p \sim_c q. \tag{4.3}$$

Soit $\sigma \stackrel{def}{=} [y_i/x_i]$ une substitution (dans la Définition 47 il suffit de se limiter seulement aux substitutions égales à l'identité partout en dehors de $fn(p \parallel q)$).

Considérons la dérivation

$$C_1[p] \xrightarrow{u(y_1)} R_1 \xrightarrow{u(y_2)} R_2 \ldots \xrightarrow{u(y_n)} R_n = p[y_i/x_i] + \sum_{i=1}^{n} y_i(x).\bar{v}$$

Comme $C_1[p]{\sim}C_1[q]$, il faut qu'il existe une dérivation

$$C_1[p] \xrightarrow{u(y_1)} S_1 \xrightarrow{u(y_2)} S_2 \ldots \xrightarrow{u(y_n)} S_n = q[y_i/x_i] + \sum_{i=1}^{n} y_i(x).\bar{v},$$

tel que

$$p[y_i/x_i] + \sum_{i=1}^{n} y_i(x).\bar{v} \sim q[y_i/x_i] + \sum_{i=1}^{n} y_i(x).\bar{v} \tag{4.4}$$

Par une analyse par cas, il est facile de prouver que l'équivalence 4.4 implique $p[y_i/x_i]{\sim_+}q[y_i/x_i]$ (chaque réception de $p[y_i/x_i)$ doit être simulée par une réception de $q[y_i/x_i)$ et réciproquement).

Comme σ était une substitution arbitraire, on obtient l'implication 4.3.

Par l'utilisation des assertions 4.2 et 4.3 nous avons les implications suivantes:

$$(p \ \stackrel{c}{\sim}_b \ q) \implies (C_1[p] \ \stackrel{c}{\sim}_b \ C_1[q]) \implies (C_1[p] \ \stackrel{e}{\sim}_b \ C_1[q])$$

$$\implies (C_1[p] \sim C_1[q]) \implies (p \sim_c q)$$

$\square$ *Theoreme* 56

Si nous notons par $\stackrel{c}{\sim}_\phi$ la congruence induite par $\sim_\phi$, alors il est facile de prouver que $\sim_c = \stackrel{c}{\sim}_b = \stackrel{c}{\sim}_\phi$.

De la meme facon que pour les congruences fortes, on obtient des résultats similaires pour les congruences au sens faible.

Définition 57 Congruence barbelée faible

Deux processus p et q sont congruents dans le sens de la **congruence barbelée faible**, *noté* $p \stackrel{c}{\approx}_b q$, *si pour tout contexte* C, $C[p] \approx_b C[q]$.

Définition 58 *Soit* $\approx_+$ *une relation symétrique qui satisfait les conditions suivantes:*

1) *si* $p \xrightarrow{\tau} p'$, *alors* $\exists q'$ *tel que* $q \stackrel{\tau}{\Longrightarrow} q'$ *et* $p' \approx q'$,

2) *si* $p \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} p'$ *et* $\tilde{b} \cap fn(p,q) = \emptyset$, *alors* $\exists q'$ *tel que* $q \stackrel{\nu\tilde{b}\bar{a}\tilde{c}}{\Longrightarrow} q'$ *et* $p' \approx q'$,

3) *si* $p \xrightarrow{a\langle\tilde{b}\rangle} p'$ *alors* $\exists q'$ *tel que* $q \xrightarrow{a\langle\tilde{b}\rangle} q'$ *ou* $q \stackrel{\tau}{\Longrightarrow}\xrightarrow{a\langle\tilde{b}\rangle;} q'$ *et* $p' \approx q'$,

4) si $p \xrightarrow{a:}$, alors $q \xRightarrow{a:}$.

Soit $\approx_c$ défini par $p \approx_c q$ si $p\sigma \approx_+ q\sigma$ pour toutes les substitutions σ.

Théoreme 59 *$\approx_c$ est une congruence.*

Théoreme 60 *Pour tous les processus d'image finie p et q*

$p \approx_c q \quad ssi \quad p \overset{c}{\approx}_b q.$

4.3 Axiomatisation de la congruence forte

Dans cette section nous donnons une axiomatisation complète de la congruence pour les processus finis (qui ne contiennent pas de récursion). Notre axiomatisation est dérivée de l'axiomatisation donnée pour le π-calcul dans (Parrow et Sangiorgi, 1995). En plus, nous devons prendre en compte que la congruence forte $\sim_c$ n'est pas directement obtenue à partir de la bisimulation forte $\sim$ par une cloture aux substitutions (comme il était le cas pour le π-calcul), mais à partir d'une relation strictement plus forte, $\sim_+$. Le gap entre $\sim$ et $\sim_+$ est rempli par le nouvel axiome (H) (qui n'est pas vrai pour la congruence forte dans le π-calcul), et qui correspond à l'axiome $P - Noisy$ donnée pour CBS dans (Hennessy et Rathke, 1995). Pour garder la syntaxe simple, nous utilisons la version monadique de notre calcul.

4.3.1 Caractérisation de la congruence forte pour les processus simples

Dans cette sous-section nous limitons notre attention aux processus définis par

$$p \ ::= \quad nil \mid \alpha.p \mid p_1 + p_2 \mid \phi p_1, p_2$$

où α appartient à l'ensemble des préfixes $\alpha ::= \tau \mid x(y) \mid \bar{x}y$, $x,y \in Ch_b$.

De la même façon que (Parrow et Sangiorgi, 1995), nous utilisons la forme plus générale $\phi p_1, p_2$ où

$$\phi \ ::= \quad \langle x = y \rangle \mid \neg\phi \mid \phi_1 \wedge \phi_2$$

avec $x,y \in Ch_b$ ($\phi_1 \wedge \phi_2 p_1, p_2$ étant une notation pour $\phi_1(\phi_2 p_1, p_2), p_2$)

et nous utilisons l'abréviation ϕp pour $\phi\ p, nil$ et $\langle x \neq y \rangle p, q$ pour $\neg\langle x = y \rangle p, q$. Nous notons par $In(p)$ l'ensemble des canaux sur lequels p est à l'écoute (l'ensemble des noms a tels que $p \xrightarrow{a\langle x \rangle} p'$ pour un processus quelconque p').

Le système d'axiomes $\mathcal{A}$ pour la congruence forte $\sim_c$ est donné dans la Table 4.2.

(A)	si $p \equiv_\alpha q$, alors $p = q$
(IP)	si $p = q$ alors $\alpha.p = \alpha.q$
(IC)	si $p = q$ alors $\phi p = \phi q$
(IS)	si $p = q$ alors $p + r = q + r$
(H)	si $x \notin fn(p)$ et $\forall b \in In(p)\big(\phi \Rightarrow \langle a \neq b\rangle\big)$ alors $\alpha.p = \alpha.(p + \phi a(x).p)$
$(S1)$	$p + nil = p$
$(S2)$	$p + p = p$
$(S3)$	$p + q = q + p$
$(S4)$	$(p + q) + r = p + (q + r)$
$(C3)$	si $\phi \iff \psi$ alors $\phi p = \psi p$
$(C4)$	$False\ p = False\ q$
$(C5)$	$\phi p,p = p$
$(C6)$	$\phi p,q = \neg\phi q,p$
$(CC1)$	$\phi(\psi p) = [\phi \wedge \psi]p$
$(SC1)$	$\phi(p_1 + p_2),(q_1 + q_2) = \phi p_1,q_1 + \phi p_2,q_2$
$(CP1)$	si $bn(\alpha) \cap n(\phi) = \emptyset$ alors $\phi(\alpha.p) = \phi(\alpha.\phi p)$
$(CP2)$	$\langle x = y\rangle\alpha.p = \langle x = y\rangle(\alpha\{x/y\}).p$
(SP)	$a(x).p + a(x).q = a(x).p + a(x).q + a(x).(\langle x = y\rangle p,q)$

TAB. 4.2 – *Le système d'axiomes $\mathcal{A}$ pour la congruence forte.*

Nous écrivons $\mathcal{A} \vdash p = q$ si $p = q$ peut être prouvé par l'utilisation des règles de la Table 4.2. Le théorème suivant est facile à prouver:

Théoreme 61 (correction de $\mathcal{A}$ pour $\sim_c$)

Si $\mathcal{A} \vdash p = q$ alors $p \sim_c q$.

Preuve

Voir l'annexe D.

□ *Theoreme* 61

Comme dans (Parrow et Sangiorgi, 1995), on peut prouver que pour tout processus, il existe un processus équivalent (dans le système d'axiomes $\mathcal{A}$) qui se trouve dans une "forme normale"; de plus, on va montrer que si deux processus en forme normale sont congruents, alors les deux processus peuvent être prouvés égaux dans notre système d'axiomes.

Définition 62 *(Parrow et Sangiorgi, 1995) Soit V un ensemble de noms; une condition ϕ est* complète *sur V si pour une certaine relation d'équivalence $\mathcal{R}$ sur V, on a $\phi \Rightarrow \langle x = y\rangle$ ssi $x\mathcal{R}y$, et $\phi \Rightarrow \langle x \neq y\rangle$ ssi $\neg(x\mathcal{R}y)$.*

Définition 63 (forme normale préfixe) *Soit V un ensemble de noms. p est en forme normale préfixe sur V (hnf), s'il a la forme* $\Sigma_{i\in I}\phi_i\alpha_i.\phi_i p_i$*, où pour tous les i,*

1 $bn(\alpha_i) \notin V$;

2 ϕ_i *est complète sur V.*

Lemme 64 *Pour tout processus p, et pour tout ensemble fini de noms V avec* $fn(p) \subseteq V$*, il existe un processus h de profondeur plus petite ou égale à celle de p tel que h est en hnf sur V, et* $\mathcal{A} \vdash p = h$.

La preuve est similaire à celle du Lemme 4.8 dans (Parrow et Sangiorgi, 1995).

Définition 65 *Une substitution* σ *concorde avec une condition* ϕ*, et* ϕ *concorde avec* σ*, si pour tout x, y qui apparaissent dans* ϕ*, on a* $\sigma(x) = \sigma(y)$ *ssi* $\phi \Rightarrow [x = y]$.

Lemme 66 *(Parrow et Sangiorgi, 1995) Soit V un ensemble de noms et soit* ϕ *complète sur V.*

1. *Si* σ *et* σ' *sont des substitutions sur V qui concordent avec* ϕ*, alors* $\sigma = \sigma'\rho$ *pour une certaine substitution injective* ρ.
2. *Si* ψ *est une autre condition sur V, soit* $\phi \wedge \psi$ *n'est pas satisfaisable, soit* $\phi \wedge \psi \iff \phi$.
3. *Si* ψ *est une autre condition complète sur V telle que* ϕ *et* ψ *concordent avec la même substitution* σ *alors* $\phi \iff \psi$.

Par une analyse par cas on peut facilement prouver le lemme suivant.

Lemme 67 *Supposons* $p \sim_+ q$ *et que* σ *est injective sur* $fn(p,q)$*. Alors* $p\sigma \sim_+ q\sigma$.

Lemme 68 *Soit* $p \equiv \phi p'$ *et* $q \equiv \phi q'$*, avec* ϕ *complète sur un ensemble* V_1 *de noms. Soit* $V_2 = fn(p,q) - V_1$ *et* σ_1 *une substitution tel que:*

1 $prdom(\sigma_1) \subseteq V_1$ *et* σ_1 *concorde avec* ϕ;

2 $prcod(\sigma_1) \cap V_2 = \emptyset$;

3 *pour tout* σ_2 *avec* $prdom(\sigma_2) \subseteq V_2$*, on a* $p\sigma_1\sigma_2 \sim_+ q\sigma_1\sigma_2$

Alors $p \sim_c q$.

Preuve Similaire à celle de (Parrow et Sangiorgi, 1995) pour la preuve du Lemme 4.5.

□ *Lemme* 68

Par un raisonnement similair à celui utilisé dans (Parrow et Sangiorgi, 1995) (pour la preuve des Théorèmes 4.9 et 4.11), combiné avec l'utilisation de l'axiome (H), nous pouvons prouver que $p \sim_c q$ implique $\mathcal{A} \vdash p = q$.

Théoreme 69 (le système $\mathcal{A}$ est complet pour $\sim_c$) *Si* $p \sim_c q$ *alors* $\mathcal{A} \vdash p = q$.

Preuve

La preuve du Théorème 69 hérite beaucoup des preuves des Théorèmes 4.9 et 4.11 de (Parrow et Sangiorgi, 1995), mais nous devons tenir compte du fait que la congruence forte $\sim_c$ n'est pas obtenue directement de la bisimulation forte $\sim$ par une clôture aux substitutions (comme dans le π-calcul), mais à partir d'une relation plus forte $\sim_+$. Le gap entre $\sim$ et $\sim_+$ est rempli par le nouvel axiome (H).

Par le Lemme 64, il suffit de prouver l'assertion quand p,q sont en forme normale préfixe sur $fn(p,q)$. La preuve se fait par induction sur la somme des profondeurs des p et q.

Soit p_{out} la partie "émission" de p (la somme des composants préfixés par des émissions plus la somme des composants préfixés par des τ dans p), et $p_{\phi,a}$ la somme des composants $\phi_i \alpha_i . p_i$ de p tel que ϕ_i est équivalent à ϕ et tel que le préfixe α_i est le même que $a(x)$ (en tenant compte de l'alpha-conversion et de l'identification des noms impliqués par ϕ_i).

Après avoir utilisé plusieurs fois $(S3)$ et $(S4)$ on peut réécrire p sur la forme

$$\mathcal{A} \vdash p = p_{out} + \sum_{a \in In(p), \phi \text{ complète sur } V} p_{\phi,a}$$

[1.] D'abord nous prouvons que pour chaque composante de p_{out}, il existe une composante de q qui est égale dans le système d'axiomes $\mathcal{A}$.

Soit $\phi\alpha.p'$ une composante de p_{out}, et σ une substitution qui concorde avec ϕ; on peut supposer que σ coïncide avec l'identité sur les noms qui n'apparaissent pas libres dans p ou q.

On a $p\sigma \xrightarrow{\alpha\sigma} p'\sigma$. Comme $p \sim_c q$, nous obtenons que $p\sigma \sim_+ q\sigma$. Soit $\psi\beta.q'$ la composante de q utilisée pour simuler la transition de $p\sigma$. Par l'alpha-conversion on peut supposer que les noms bornés (s'ils en existent) dans α et β sont les mêmes. Par la définition de $\sim_+$ nous avons

– σ concorde avec ψ
– $\alpha\sigma = \beta\sigma$
– $p'\sigma \sim q'\sigma$.

Comme ϕ et ψ sont complètes sur $fn(p,q)$ et ils concordent avec σ, par le Lemme 66, nous obtenons $\phi \iff \psi$ et par (C3) $\mathcal{A} \vdash \psi\beta.q' = \phi\beta.q'$. Si α et β différent dans les noms a et b ($\alpha\{a/b\} = \beta\{a/b\}$), comme $\alpha\sigma = \beta\sigma$, on obtient $\sigma(a) = \sigma(b)$. Comme ϕ concorde avec σ, nous obtenons $\psi \Rightarrow \langle a = b \rangle$ et donc

$$\phi\beta.q' = \phi(\langle a = b\rangle \beta.q') = \phi(\langle a = b\rangle (\beta\{a/b\}).q') = \phi(\langle a = b\rangle (\alpha\{a/b\}).q') = \phi\alpha.q'$$

On ne peut pas s'attendre à prouver directement $p' \sim_+ q'$, car certaines réceptions d'un des processus peuvent être simulées par un "ignore" de la part de l'autre processus. Mais nous allons saturer p' et q' de tel sorte que chacun d'entre eux ne pourra plus ignorer les réceptions de l'autre.

Soit $p' = \phi p''$, $q' = \phi q''$ et soit

$$s' = (\phi p'' + \sum\{\phi a(x).\phi p'' \mid a \in In(q''), \forall b \in In(p'')\ \phi \Rightarrow \langle a \neq b \rangle\})$$

avec $x \notin fn(p'')$, et

$$t' = (\phi q'' + \sum\{\phi a(x).\phi q'' \mid a \in In(p''), \forall b \in In(q'')\ \phi \Rightarrow \langle a \neq b \rangle\})$$

avec $x \notin fn(q'')$.

Par l'utilisation répetée de l'axiome (H) et à la fin de l'axiome (IC), nous pouvons prouver $\mathcal{A} \vdash \phi\alpha.p' = \phi\alpha.s'$ et $\mathcal{A} \vdash \phi\alpha.t' = \phi\alpha.q'$. Nous ne pouvons pas utiliser l'induction pour deduire $\mathcal{A} \vdash \phi\alpha.s' = \phi\alpha.t'$ à partir de $s' \sim_c t'$ car la somme des profondeurs de s' et t' est la même que pour p et q, donc nous allons utiliser le processus intermédiaire $p' + q'$.

Nous montrons que $s' \sim_c (p' + q')$ et $(p' + q') \sim_c t'$.

On peut prouver que $s'\sigma \sim_+ (p' + q')\sigma$ en utilisant le fait que $(\phi p'')\sigma \sim (\phi q'')\sigma$ et que chaque émission de $(\phi p'')\sigma$, qui était simulée par un "ignore", est maintenant simulée par une composante $(\phi a(x).\phi q'')\sigma$. De la même façon nous montrons que $(p' + q')\sigma \sim_+ t'\sigma$. Alors on obtient que $s' \sim_c (p' + q')$ et $(p' + q') \sim_c t'$ par le Lemme 68 (en prenant $V_1 = fn(p,q)$ et $\sigma_1 = \sigma$).

Par l'hypothèse d'induction, nous obtenons $\mathcal{A} \vdash s' = p' + q'$ et $\mathcal{A} \vdash p' + q' = t'$.

En conséquence, $\mathcal{A} \vdash s' = t'$ et en utilisant ensuite (IP) et (IC) nous obtenons $\mathcal{A} \vdash \phi\alpha.s' = \phi\alpha.t'$, et donc $\mathcal{A} \vdash \phi\alpha.p' = \psi\beta.q'$.

[2.] Soit $p_{\phi,a} = \sum_{i=1}^{n} \phi a(x).p_i$ et $q_{\phi,a} = \sum_{j=1}^{m} \phi a(x).q_i$

Pour chaque $i \in [1,n]$ nous allons exhiber les processus u_i et r_i tel que

$\mathcal{A} \vdash \phi a(x).p_i = \phi a(x).u_i$, $\mathcal{A} \vdash q_{\phi,a} = q_{\phi,a} + \phi a(x).r_i$, et $\mathcal{A} \vdash u_i = r_i$.

Soit $V = \{y_1, ..., y_k\}$ l'ensemble des noms libres dans $p_{\phi,a}$ et $q_{\phi,a}$ et soit x un nom tel que $x \notin fn(p_{\phi,a}, q_{\phi,a})$. Soit σ une substitution qui concorde avec ϕ (on peut supposer que $\forall z \notin \{y_1, \ldots, y_k\}\ \sigma(z) = z$).

$p\sigma \sim_+ q\sigma$ implique $p_{\phi,a}\sigma \sim_+ q_{\phi,a}\sigma$. De $p_{\phi,a}\sigma \stackrel{a\langle x\rangle\sigma}{\longrightarrow} p_i\sigma$, par la définition de $\sim_+$, pour chaque $y \in fn(p_{\phi,a}, q_{\phi,a}) \cup \{x\}$, il existe $J(i,y)$ tel que $q_{\phi,a}\sigma \stackrel{a\langle x\rangle\sigma}{\longrightarrow} q_{J(i,y)}\sigma$ et $p_i\sigma\{y/x\} \sim q_{J(i,y)}\sigma\{y/x\}$.

Si M est un ensemble de noms, nous utilisons la notation $[x \notin M]$ pour $[\wedge_{z \in M} \langle x \neq z \rangle]$. Soit $p_i = \phi p'_i$, $q_j = \phi q'_j$ et soit

$$u_i = (\phi p'_i + \sum_{z \in \{y_1, \ldots, y_k\}} \sum_{d \in A_z} [x = z] \wedge \phi d(y).\phi p'_i$$

$$+ \sum_{d \in A_x} [x \notin \{y_1, \ldots, y_k\}] \wedge \phi d(y).\phi p'_i)$$

avec $y \notin fn(p'_i)$, $A_z = \{b \mid b \in In(q'_{J(i,z)}), \forall a \in In(p'_i), [x = z] \wedge \phi \Longrightarrow a \neq b\}$,

$A_x = \{b \mid b \in In(q'_{J(i,x)}), \forall a \in In(p'_i), [x \notin \{y_1, \ldots, y_k\}] \wedge \phi \Longrightarrow a \neq b\}$, et soit

$$t_j = (\phi q'_j + \sum_{z \in \{y_1, \ldots, y_k\}} \sum_{d \in B_z} [x = z] \wedge \phi d(y).\phi q'_j$$

$$+ \sum_{d \in B_x} [x \notin \{y_1, \ldots, y_k\}] \wedge \phi d(y).\phi q'_j)$$

avec $y \notin fn(q'_j)$, $B_z = \{b \mid b \in In(p'_i), \forall a \in In(q'_j), [x = z] \wedge \phi \Longrightarrow a \neq b\}$,

et $B_x = \{b \mid b \in In(p'_i), \forall a \in In(q'_j), [x \notin \{y_1, \ldots, y_k\}] \wedge \phi \Longrightarrow a \neq b\}$.

Soit $s_{i,0} = t_{J(i,x)}$, $s_{i,l} = \langle x = y_l \rangle t_{J(i,y_l)}, s_{i,l-1}$, et $r_i \equiv s_{i,k}$.

Par l'utilisation répétée de l'axiome (H) et ensuite de l'axiome (IC) , nous pouvons prouver que

$$\mathcal{A} \vdash \phi a(x).p_i = \phi a(x).u_i \tag{4.5}$$

et $\mathcal{A} \vdash \phi a(x).q_j = \phi a(x).t_j$.

On prouve maintenant que $\mathcal{A} \vdash q_{\phi,a} = q_{\phi,a} + \phi a(x).s_{i,l}$ par induction sur l.

De $\mathcal{A} \vdash \phi a(x).q_j = \phi a(x).t_j$ et $\mathcal{A} \vdash q_{\phi,a} = q_{\phi,a} + \phi a(x).q_j$ (comme $\phi a(x).q_j$ est une composante de $q_{\phi,a}$), nous obtenons que $\mathcal{A} \vdash q_{\phi,a} = q_{\phi,a} + \phi a(x).t_j$ Si $l = 0$, $\mathcal{A} \vdash q_{\phi,a} = q_{\phi,a} + \phi a(x).t_{J(i,x)}$ suit immédiatement des égalités prouvées antérieurement. Par induction nous supposons

$$\mathcal{A} \vdash q_{\phi,a} = q_{\phi,a} + \phi a(x).s_{i,l-1}$$

Comme $\mathcal{A} \vdash q_{\phi,a} = q_{\phi,a} + \phi a(x).t_{J(i,y_l)}$ nous obtenons que $\mathcal{A} \vdash q_{\phi,a} = q_{\phi,a} + \phi a(x).s_{i,l-1} + \phi a(x).t_{J(i,y_l)}$, et en utilisant (SP) nous obtenons que $\mathcal{A} \vdash q_{\phi,a} = q_{\phi,a} + \phi a(x).s_{i,l-1} + \phi a(x).t_{J(i,y_l)} + a(x).\langle x = y_l \rangle t_{J(i,y_l)}, s_{i,l-1}$ et ensuite $\mathcal{A} \vdash q_{\phi,a} = q_{\phi,a} + \phi a(x).\langle x = y_l \rangle t_{J(i,y_l)}, s_{i,l-1}$. En faisant $l = k$ on obtient

$$\mathcal{A} \vdash q_{\phi,a} = q_{\phi,a} + \phi a(x).r_i. \tag{4.6}$$

On prouve maintenant que $\mathcal{A} \vdash u_i = r_i$.

On ne peut pas appliquer directement l'hypothèse d'induction sur u_i et t_j car la somme des profondeurs des u_i et t_j est la même que pour p et q. Donc nous allons passer par les processus "intermédiaires" $p_i + q_j$ avec $j \in \{1, \ldots, m\}$. A partir de $p_i\sigma\{y/x\} \sim$

$q_{J(i,y)}\sigma\{y/x\}$, par une analyse par cas, nous obtenons

$$u_i\sigma\{y/x\} \sim_+ (p_i + q_{J(i,y)})\sigma\{y/x\} \sim_+ t_{J(i,y)}\sigma\{y/x\} \tag{4.7}$$

(les réceptions antérieurement simulées par des "ignore", sont maintenant simulées par des réceptions “inoffensives” $\phi b(y).\phi p'_i$ ou $\phi b(y).\phi q'_{J(i,y)}$).

La condition ϕ ne mentionne pas x, et il se peut qu'elle ne soit pas complète sur $fn(p_i,q_{J(i,y)}) = fn(p_{\phi,a},q_{\phi,a}) \cup \{x\}$; donc on ne peut pas utiliser directement le Lemme 68; mais comme dans (Parrow et Sangiorgi, 1995) nous pouvons la compléter en ajoutant un conditionnel qui concorde avec $\{y/x\}$. De l'equation 4.7 nous obtenons $\langle x = y\rangle u_i\sigma\{y/x\} \sim_+ \langle x = y\rangle(p_i + q_{J(i,y)})\sigma\{y/x\} \sim_+ \langle x = y\rangle t_{J(i,y)}\sigma\{y/x\}$ pour $y \in V$ et $\langle x \neq y\rangle u_i\sigma\{y/x\} \sim_+ \langle x \neq y\rangle(p_i + q_{J(i,y)})\sigma\{y/x\} \sim_+ \langle x \neq y\rangle t_{J(i,y)}\sigma\{y/x\}$ pour $y \notin V$.

Nous appliquons le Lemme 68 , où $V_1 = V \cup \{x\}$, $\sigma_1 = \sigma\{y/x\}$ et $V_2 = \emptyset$ dans l'énoncé du lemme, et on obtient

$$\langle x = y\rangle u_i\sigma\{y/x\} \sim_c \langle x = y\rangle(p_i + q_{J(i,y)})\sigma\{y/x\} \sim_c \langle x = y\rangle t_{J(i,y)}\sigma\{y/x\} \text{ pour } y \in V \tag{4.8}$$

et

$$\langle x \neq y\rangle u_i\sigma\{y/x\} \sim_c \langle x \neq y\rangle(p_i + q_{J(i,y)})\sigma\{y/x\} \sim_c \langle x \neq y\rangle t_{J(i,y)}\sigma\{y/x\} \text{ pour } y \notin V \tag{4.9}$$

et en utilisant l'hypothèse d'induction, nous obtenons

$$\mathcal{A} \vdash \langle x = y\rangle u_i = \langle x = y\rangle(p_i + q_{J(i,y)}) = \langle x = y\rangle t_{J(i,y)} \text{ pour } y \in \{y_1, \ldots, y_k\}$$

et

$$\mathcal{A} \vdash [x \notin V]u_i = [x \notin V](p_i + q_{J(i,y)}) = [x \notin V]t_{J(i,y)} \text{ pour } y \notin \{y_1, \ldots, y_k\}.$$

De la même façon que dans la preuve du Théorème 4.11 dans (Parrow et Sangiorgi, 1995), en utilisant 4.8 4.9 nous pouvons prouver $\mathcal{A} \vdash u_i = r_i$ et ensuite

$$\mathcal{A} \vdash \phi a(x).u_i = \phi a(x).r_i. \tag{4.10}$$

Dés 4.5, 4.6 et 4.10 nous obtenons que pour tout $i \in [1,n]$ $\mathcal{A} \vdash q_{\phi,a} = q_{\phi,a} + \phi a(x).p_i$, et ensuite $\mathcal{A} \vdash q_{\phi,a} = q_{\phi,a} + p_{\phi,a}$. Par un argument symétrique, nous obtenons $\mathcal{A} \vdash p_{\phi,a} = p_{\phi,a} + q_{\phi,a}$ et donc $\mathcal{A} \vdash q_{\phi,a} = p_{\phi,a}$

$\square$ *Theoreme* 69

De plus, nos axiomes sont indépendants (affirmation que découle du fait que tous les autres axiomes à part (H) sont prouvés indépendants dans (Parrow et Sangiorgi, 1995),

(IR) si $p = q$ alors $\nu xp = \nu xq$
(R) $\nu x\, nil = nil$
(RR) $\nu x \nu yp = \nu y \nu xp$
(RS) $\nu x(p+q) = \nu xp + \nu xq$
$(RP1)$ si $x \notin n(\alpha)$ alors $\nu x\alpha.p = \alpha.\nu xp$
$(RP2)$ $\nu x\, \bar{x}y.p = \tau.\nu xp$
$(RP3)$ $\nu x\, x(y).p = nil$
$(RC1)$ si $x \neq y$ alors $\nu x\langle x = y\rangle p = nil$
$(RC2)$ si $x \neq y,z$ alors $\nu x\langle z = y\rangle p = \langle z = y\rangle \nu xp$

TAB. 4.3 – *Les axiomes pour la restriction.*

et que (H) ne peut pas être prouvé à partir des autres axiomes).

4.3.2 L'ajout de l'opérateur de restriction

A la grammaire donnée dans la section antérieure, nous ajoutons l'opérateur de restriction:

$$p ::= \ldots \mid \nu xp$$

Les axiomes nécessaires pour traiter la restriction sont donnés dans la Table 4.3.

Le seul axiome qui est nouveau (et qui n'est pas vrai dans le π-calcul) est $(RP2)$. La correction de tous les axiomes est facile à prouver. Pour prouver que le système d'axiomes est complet, nous utilisons les axiomes de la Table 4.3 pour "pousser" la restriction à l'intérieur d'un terme jusqu'au moment où, soit il disparaît, soit il génère une émission bornée. La définition de la forme normale change légèrement: $\Sigma_{i\in I}\phi_i\alpha_i.\phi'_i p_i$, où pour tout i,

1 $bn(\alpha_i) \notin V$,

2 ϕ_i est complète sur V,

3 $\phi_i = \phi'_i$ si α_i est τ, une réception ou une émission libre,

4 $\phi'_i = \phi_i \wedge (\wedge_{z\in V}\langle x \neq z\rangle)$ si α_i est une émission bornée de la forme $\nu x\bar{a}x$.

La preuve du fait que les axiomes des Table 4.2 et 4.3 forment un système complet suit comme dans le Théorème 69.

4.3.3 L'ajout du parallélisme

A la grammaire donnée dans la section antérieure, nous ajoutons l'opérateur de parallélisme:

$$p ::= \ldots \mid p_1 \parallel p_2$$

Supposons

$$p \equiv \sum_{i_1 \in M_1} \phi_{i_1} \overline{x_{i_1}}[v].p_{i_1} + \sum_{i_2 \in M_2} \phi_{i_2} x_{i_2}(v).p_{i_2} + \sum_{i_3 \in M_3} \phi_{i_3} \tau.p_{i_3}$$

et

$$q \equiv \sum_{j_1 \in N_1} \phi_{j_1} \overline{x_{j_1}}[v].q_{j_1} + \sum_{j_2 \in N_2} \phi_{j_2} x_{j_2}(v).q_{j_2} + \sum_{j_3 \in N_3} \phi_{j_3} \tau.q_{j_3}$$

où $[v]$ est soit v (une émission libre) soit (v) (une émission bornée).
Soit $S = \{x_i \mid i \in M_2\}$ et $T = \{x_i \mid i \in N_2\}$. Alors:

$$\begin{aligned}
p \parallel q = \quad & \Sigma_{(i_2,j_2)}[\phi_{i_2} \wedge \phi_{j_2} \wedge \langle x_{i_2} = x_{j_2}\rangle] x_{i_2}(v).(p_{i_2} \parallel q_{j_2}) + \\
& \Sigma_{i_2}[\phi_{i_2} \wedge [x_{i_2} \notin T]] x_{i_2}(v).(p_{i_2} \parallel q) + \\
& \Sigma_{j_2}[\phi_{j_2} \wedge [x_{j_2} \notin S]] x_{j_2}(v).(p \parallel q_{j_2}) + \\
& \Sigma_{(i_1,j_2)}[\phi_{i_1} \wedge \phi_{j_2} \wedge \langle x_{i_1} = x_{j_2}\rangle] \overline{x_{i_1}}[v].(p_{i_1} \parallel q_{j_2}) + \\
& \Sigma_{(i_2,j_1)}[\phi_{i_2} \wedge \phi_{j_1} \wedge \langle x_{i_2} = x_{j_1}\rangle] \overline{x_{i_2}}[v].(p_{i_2} \parallel q_{j_1}) + \\
& \Sigma_{i_1}[\phi_{i_1} \wedge [x_{i_1} \notin T]] \overline{x_{i_1}}[v].(p_{i_1} \parallel q) + \Sigma_{j_1}[\phi_{j_1} \wedge [x_{j_1} \notin S]] \overline{x_{j_1}}[v].(p \parallel q_{j_1}) + \\
& \Sigma_{i_3} \phi_{i_3} \tau.(p_{i_3} \parallel q) + \Sigma_{j_3} \phi_{j_3} \tau.(p \parallel q_{j_3})
\end{aligned}$$

TAB. 4.4 – *L'axiome d'expansion.*

Les axiomes nécessaires pour traiter le parallélisme sont l'axiome $(P1)$ $\quad p \parallel nil = p$ plus l'axiome d'expansion donnée dans la Table 4.4.

Dans la Table 4.4, la première composante correspond à la situation où les deux processus font une réception. La deuxième et la troisième composante correspondent à la situation où un des processus fait une reception, et l'autre un "ignore". La quatrième et la cinquième composante correspondent à la situation où un des processus fait une émission, et l'autre une réception. La sixième et la septième composante correspondent à la situation où un des processus fait une émission, et l'autre un "ignore". Enfin, la huitième et la neuvième composante correspondent à la situation où un des processus fait un pas silencieux τ. Pour prouver que le système d'axiomes donné dans les tables 4.2, 4.3 et 4.4 est complet, il suffit d'éliminer l'opérateur $\parallel$ par l'utilisation de l'axiome $(P1)$ et de l'axiome d'expansion.

4.4 Conclusion

Dans ce chapitre nous avons introduit plusieurs relations d'équivalence qui permettent d'identifier (ou de distinguer) deux processus en fonction de leurs réponses aux interactions avec l'environnement.

Les bisimulations barbelées sont des relations d'équivalence qui peuvent être définies dans tout modèle disposant d'une relation de réduction (de réécriture) et muni d'une notion d'observabilité. Cette définition générale permet d'établir des liaisons entre des processus définis dans des modèles différents (par exemple la correction du codage du π-calcul d'ordre superieur dans le π-calcul de premier ordre). Ces considérations nous ont conduit à introduire et étudier les bisimulations barbelées dans notre modèle.

Deux questions qui apparaissent normalement sont "quel doit être le prédicat d'observabilité?"et "quelle relation de réduction doit - on prendre?". Pour la première question, dans un modèle à diffusion, les émissions sont observables, mais pas les réceptions (Prasad, 1993). Pour la deuxième question on peut choisir comme relation de réduction soit l'evolution silencieuse $\xrightarrow{\tau}$ soit l'evolution "autonome" $\xrightarrow{\emptyset}$. Alors on obtient deux relations incomparables (remarque 28): la bisimulation barbelée et la ϕ - bisimulation. Les deux relations sont relativement faibles ce qui nous conduit (comme dans le π -calcul) à étudier leur clôture aux contextes statiques (les équivalences barbelés et les ϕ - équivalences).

Le fait que les deux relations (les équivalences barbelées et les ϕ - équivalences) soient plus appropriées à prouver la non - équivalence plutot que l'équivalence de deux systèmes (par l'exhibition d'un processus observateur qui peut les distinguer), nous a conduit à introduire les bisimulations étiquetées. Par des techniques classiques (Sangiorgi, 1992) nous prouvons que pour les processus d'image finie, les trois classes de relations coïncident.

Ensuite, nous considérons comme application la spécification d'un protocole de FIFO broadcast dans un réseau asynchrone de processus (il n'y a aucune garantie en ce qui concerne l'ordre de réceptions des messages). Par l'exhibition d'une relation de bisimulation appropriée on prouve la correction du protocole.

Nous terminons le chapitre par un étude des congruences induites par les relations de bisimulations, et par une axiomatisation complète des congruences fortes pour les processus finis.

Chapitre 5

Caractérisations induites par des tests

Dans le chapitre précédent, on a étudié les équivalences (congruences) induites par des caractérisations co-inductives. Les bisimulations sont appropriées pour les systèmes distribués réactifs, qui ont souvent des comportements infinis, et dont la sémantique est donnée par les actions qu'ils peuvent faire comme réponse à des "stimuli" externes. De plus, les bisimulations semblent bien adaptées pour les systèmes point à point, où l'exécution du système étudié est totalement contrôlée par l'environnement (ou l'observateur). Dans les calculs à diffusion, seulement les attentes des messages d'un processus sont contrôlées par l'environnement, tandis que les émissions sont contrôlées par le système lui même. Par exemple, les processus $p \stackrel{def}{=} \bar{a}.(\bar{b} + \bar{c})$ et $q \stackrel{def}{=} \bar{a}.\bar{b} + \bar{a}.\bar{c}$ sont distingués par les bisimulations. Dans un modèle point à point, un observateur "intelligent" peut les distinguer: il fournit d'abord l'action a, et ensuite, en fonction de l'évolution du processus q il peut choisir de fournir b ou c et donc de mettre en évidence une différence entre les deux. Dans un modèle à diffusion, ces deux processus sont impossible à différencier, car les actions de p et q ne dépendent plus de l'observateur (on peut essayer éventuellement d'observer l'exécution d'un système, et de lui envoyer des messages; le système lui même, sera forcé d'accepter un message seulement quand il ne pourra plus évoluer de façon autonome). Cet exemple montre que les bisimulations sont trop restrictives dans certains cas. Il est donc intéressant de pouvoir établir d'autres relations (plus faibles) entre les processus.

Dans ce chapitre nous étudions les pre-ordres induits par des tests (de Nicola et Hennessy, 1984). Deux systèmes sont équivalents quand ils satisfont le même ensemble d'observateurs (tests). En fonction de la définition de l'univers des observateurs et de la notion de satisfaction, on peut définir plusieurs équivalences par des tests.

Soit S un système défini sur un ensemble d'actions A; un observateur O pour le système S est un processus défini sur l'ensemble d'actions $A \cup \{\bar{\omega}\}$ où $\bar{\omega}$ est une action nouvelle utilisée par l'observateur pour rapporter le succès de son observation. Pour définir la satisfaction d'un observateur on considère le processus obtenu par la composition parallèle du système et de l'observateur (toutes les exécutions communes possibles). Pour un observateur, un système *doit* le satisfaire (le test d'obligation - "must testing", noté $\ll_{must}$) si chaque exécution *complète* (qui ne peut plus être prolongée) de l'ensemble passe par un état dans le quel l'observateur peut signaler le succès (peut faire une émission $\bar{\omega}$); un système *peut* le satisfaire (le test d'acceptation - "may testing", noté $\ll_{may}$), s'il y a au moins une exécution qui passe par un état dans lequel l'observateur peut signaler le succès (peut faire une émission $\bar{\omega}$).

La notion d'équivalence induite par des tests a été formalisée dans (de Nicola et Hennessy, 1984) pour les systèmes point à point (dans le cadre de CCS). Le préordre induit par cette équivalence peut être divisé en deux préordres de processus. Le premier définit la capacité d'un processus à réagir positivement à un test (acceptations). Le deuxième définit l'incapacité à ne pas répondre positivement au test (obligations). L'équivalence entre les processus est la conjonction des équivalences induites par ces deux préordres.

Les préordres ont un rôle important dans les méthodologies de description et de vérification des systèmes: ils permettent d'établir la conformité d'une implantation par rapport à une spécification. Pour une spécification $Spec$, l'ensemble des comportements qu'une implantation doit avoir est modelisé par l'ensemble d'observateurs que $Spec$ *doit* satisfaire tandis que l'ensemble de comportements erronés est modelisé par l'ensemble des observateurs que $Spec$ ne peut pas satisfaire.

Une implantation $Impl$ du système est correcte par rapport à une spécification $Spec$, si elle satisfait les relations suivantes: $Spec \ll_{must} Impl$ et $Impl \ll_{may} Spec$.

La vérification pour les relations d'équivalence induites par les tests considère les comportements de tous les observateurs possibles, ce qui n'est pas pratique. On a donc besoin d'une relation entre la définition abstraite et le point de vue opérationnel, c.à.d. une caractérisation des préordres par des traces.

Dans ce chapitre nous étudions les équivalences entre les processus induites par les traces dans le cas des calculs à diffusion. Ce chapitre présente des résultats originaux sur le sujet. Pour ce genre d'équivalences, des travaux ont été faits seulement dans le cas des modèles point à point.

Pour les modèles synchrones point à point, la caractérisation des préordres pour le "may testing" et pour le "must testing" est donnée dans (de Nicola et Hennessy, 1984), où Hennessy et De Nicola présentent des systèmes complets de preuves.

Pour les systèmes mobiles point à point (π-calcul ou variantes), la caractérisation est présenté dans (M.Boreale et Nicola, 1995) et (Hennessy, 1991). Ces caractérisations sont basées sur les inclusions des ensembles de traces (pour le "may testing"), ou sur les soidisant "ensembles d'acceptations" (pour le "must testing").

Ces résultats ont été récemment étendus pour les modèles asynchrones.

Des caractérisations pour le "may testing" et le "must testing" ont été données pour une variante asynchrone de CCS munie des opérateurs de choix interne et externe appelée $TACCS$ (Castellani et Hennessy, 1998). Les caractérisations sont similaires aux préordres qui correspondent à la version synchrone du langage. Les auteurs donnent aussi une caractérisation équationnelle du "must testing" pour la partie finitaire de $TACCS$ (sans recursion).

Dans (M.Boreale et al., 1999), les auteurs présentent une caractérisation du "may testing" pour une version asynchrone de CCS appelée $ACCS$ (un calcul plus restrictif par rapport à $TACCS$ - le choix externe pouvant être appliqué seulement pour les processus préfixés). Cette caractérisation est ensuite généralisée au π-calcul asynchrone. Les auteurs présentent aussi une caractérisation du "must testing" pour le calcul $ACCS$.

5.1 Les préordres induits par des tests

Dans ce chapitre, les définitions des préordres induits par des tests (de Nicola et Hennessy, 1984) sont adaptées aux calculs à diffusion CBS et $b\pi$-calcul.

Définition 70 **Les observateurs** *sont des processus qui peuvent réaliser aussi une émission distinguée* $\bar{\omega}$ *avec* $\omega \notin Ch_b$.

L'action $\bar{\omega}$ peut être interprétée comme l'action qui permet aux observateurs d'annoncer leur succès. Les observateurs interagissent avec le processus testé par des échanges de messages.

Définition 71 *Une* **exécution** *de p est une séquence de transitions* $p = p_0 \xrightarrow{\emptyset} p_1 \xrightarrow{\emptyset} p_2 \xrightarrow{\emptyset} \ldots \xrightarrow{\emptyset} p_k \xrightarrow{\emptyset} \ldots$ *qui est soit infinie, soit p_k est* bloqué *(deadlock). L'exécution est* réussie *s'il existe* $n \geq 0$ *tel que* $p_n \xrightarrow{\bar{\omega}}$.

Dans les calculs où la primitive de communication est le rendez-vous (tel que CCS et π-calcul), dans la définition 71, $\xrightarrow{\emptyset}$ est remplacée par $\xrightarrow{\tau}$; dans notre calcul (comme on avait déjà remarqué dans le Chapitre 4), un système évolue de façon autonome (sans l'aide de l'environnement) par l'intermédiaire de la relation $\xrightarrow{\emptyset}$ et non par la relation $\xrightarrow{\tau}$.

Pour chaque processus p et observateur O, on a $p \ \underline{may} \ O$, si et seulement si il existe une exécution réussie de $p \parallel O$.

Définition 72 **Le préordre** *acceptation* **(may testing)**

Pour les processus p et q, on a $p \ll_{may} q$ si et seulement si pour chaque observateur O, p $\underline{may}$ O implique q $\underline{may}$ O.

Pour chaque processus p et observateur O, on a p $\underline{must}$ O, si et seulement si toutes les exécutions de $p \parallel O$ sont réussies.

Définition 73 Le préordre *obligation* **(must testing)**

Pour les processus p et q, on a $p \ll_{must} q$ si et seulement si pour chaque observateur O, p $\underline{must}$ O implique q $\underline{must}$ O.

5.2 Caractérisation pour le CBS sans passage de valeurs

5.2.1 Présentation du CBS sans passage de valeurs

Dans cette section nous présentons une version de CBS sans passage de valeurs inspirée de (Prasad, 1991). Soit Ch un ensemble dénombrable de *canaux* ou d'*en-têtes*.

Les processus sont définis par la grammaire présentée dans le Tableau 5.1.

$$p \; ::= \quad nil \mid \pi.p \mid \nu x p \mid p_1 + p_2 \mid\mid p_1 \parallel p_2 \mid X \mid rec\ X.p$$

où π appartient à l'ensemble de préfixes $\pi ::= \tau \mid x \mid \bar{x}$, avec $x,y \in Ch$.

TAB. 5.1 – *Processus dans CBS*

- les préfixes représentent des actions de base qu'un processus peut faire: x est *une réception* sur le canal x, $\bar{x}$ est *une émission* sur le canal x, et τ représente une transition interne (changement d'état);
- nil est le processus qui a fini son exécution ;
- $\pi.p$ est le processus qui fait d'abord l'action π et se comporte ensuite comme p;
- $\nu x p$ est la création d'un nouveau nom de canal x, différent de tous les noms déjà définis (et dont la portée initiale est le processus p);
- $p_1 + p_2$ représente le non-déterminisme; il se comporte soit comme p_1 soit comme p_2, en fonction des interactions offertes par l'environnement;
- $p_1 \parallel p_2$ est la composition parallèle de p_1 et p_2;
- X est une variable processus et $recX.p$ est un processus récursif.

L'ensemble de noms liés $bn(p)$, l'ensemble de noms libres $fn(p)$, et l'ensemble de noms du p, $n(p)$ sont définis comme pour le CCS. *L'alpha-conversion* est définie de facon classique.

Les actions, (notées par α,β,...) sont définies par la grammaire suivante:

$$\alpha ::= \bar{a} \mid a \mid \tau \mid a: \mid \tau :$$

où $a \in Ch$. Une action est soit une émission, soit une réception, soit un pas interne τ, soit une action d'ignorer. On note par $\mathcal{A}ct$ l'ensemble d'actions.

La sémantique opérationnelle est présentée comme un système transitionnel étiqueté (par des actions) entre les processus. La relation "ignore" $\longrightarrow \subseteq \mathcal{P}_b \times Ch_b$ notée $p \xrightarrow{\alpha:}$ se traduit par "p ignore l'action α "(voir Tableau 5.2).

- les règles (1), (2), (3) expriment le fait que les processus nil, $\tau.p$ et $\bar{b}.p$ ignorent tous les messages.
- la règles (5) exprime le fait qu'un processus qui écoute seulement sur le canal b (ou qui attend un message d'en-tête b), ignore tout ce qui se passe sur les autres canaux a (ou tous les messages avec des en-têtes différents).
- les autres règles ((6) à (8)) suivent la structure du terme.

$$(1)\frac{}{nil \xrightarrow{a:} nil} \qquad (2)\frac{}{\tau.p \xrightarrow{a:} \tau.p} \qquad (3)\frac{}{\bar{b}.p \xrightarrow{a:} \bar{a}.p}$$

$$(4)\frac{b \neq a}{b.p \xrightarrow{a:} b.a} \qquad (5)\frac{p \xrightarrow{\alpha:} p}{\nu x p \xrightarrow{\alpha:} \nu x p} \qquad (6)\frac{p_1 \xrightarrow{\alpha:} p_1 \wedge p_2 \xrightarrow{\alpha:} p_2}{p_1 + p_2 \xrightarrow{\alpha:} p_1 + p_2}$$

$$(7)\frac{p_1 \xrightarrow{\alpha:} p_1 \ \wedge \ p_2 \xrightarrow{\alpha:} p_2}{p_1 \| p_2 \xrightarrow{\alpha:} p_1 + p_2} \qquad (8)\frac{p[(rec\ X.p)/X] \xrightarrow{\alpha:} p[(rec\ X.p)/X]}{rec\ X.p \xrightarrow{\alpha:} rec\ X.p}$$

TAB. 5.2 – *La relation "ignore"*

De plus, comme dans le $b\pi$-calcul on va supposer que $p \xrightarrow{\tau:} p$ pour tout processus p et que $sub(\tau) = \tau$.

Le raisonnement $p \xrightarrow{\alpha} p'$ signifie que le processus p est capable de faire l'action α et d'évoluer ensuite dans p'. La sémantique opérationnelle est donnée dans le Tableau 5.3 (les versions symétriques de (8), (9) sont omises).

La communication entre les processus est la diffusion sans tampon. Un processus diffuse un message sur un canal a (ou équivalent, un message avec l'en-tête a) , et les autres processus le reçoivent ou l'ignorent en fonction du fait qu'ils étaient ou pas en

$$(1)\frac{}{\tau.p\xrightarrow{\tau}p} \qquad (2)\frac{}{a.p\xrightarrow{a}p} \qquad (3)\frac{}{\bar{a}.p\xrightarrow{\bar{a}}p}$$

$$(4)\frac{p\xrightarrow{\bar{a}}p'}{\nu ap\xrightarrow{\tau}\nu ap'} \qquad (5)\frac{p\xrightarrow{\alpha}p' \wedge x\notin n(\alpha)}{\nu xp\xrightarrow{\alpha}\nu xp'} \qquad (6)\frac{p_1\xrightarrow{\alpha}p' \vee p_2\xrightarrow{\alpha}p'}{p_1+p_2\xrightarrow{\alpha}p'}$$

$$(7)\frac{p_1\xrightarrow{a}p_1' \wedge p_2\xrightarrow{a}p_2'}{p_1\|p_2\xrightarrow{a}p_1'\|p_2'} \qquad (8)\frac{p_1\xrightarrow{\bar{a}}p_1' \wedge p_2\xrightarrow{a}p_2'}{p_1\|p_2\xrightarrow{\bar{a}}p_1'\|p_2'} \qquad (9)\frac{p_1\xrightarrow{\alpha}p_1' \wedge p_2\xrightarrow{\alpha:}p_2}{p_1\|p_2\xrightarrow{\alpha}p_1'\|p_2}$$

$$(10)\frac{p[(rec\ X.p)/X]\xrightarrow{\alpha}p'}{(rec\ X.p)\xrightarrow{\alpha}p'}$$

TAB. 5.3 – *La sémantique opérationnelle du CBS*

écoute sur le canal a (respectivement, en attente d'un message d'en-tête a) . Un processus qui "écoute"sur un canal a, ne peut ignorer aucune valeur diffusée sur ce canal.

En plus des notations déjà introduites, on va utiliser $p \not\xrightarrow{\emptyset}$ ssi $\nexists p'$ tel que $p \xrightarrow{\emptyset} p'$.

5.2.2 Une caractérisation de "may testing" basée sur des traces

Une trace d'un processus est une séquence d'actions que le processus peut faire. Un observateur n'est pas sensible aux changements d'état interne des processus. Donc, les apparitions de l'action τ dans les traces d'un processus ne sont pas représentatives. Pour la même trace, toutes les traces qui diffèrent seulement par quelques apparitions de τ, satisfont les mêmes observateurs. C'est pour cette raison que nous allons considérer un ensemble restreint d'actions $\mathcal{R}Act \stackrel{def}{=} \mathcal{A}ct \setminus \{\tau,\tau :\}$. Alors, l'ensemble des traces Tr, est défini par la grammaire: $t ::= \epsilon \mid \alpha.t$, où $\alpha \in \mathcal{R}Act$; ϵ représente la trace vide, et dans $\alpha.t$, le préfixe α dénote la première action faite par un processus, et le suffixe t est la trace que le processus peut faire ensuite. Souvent on va omettre le ϵ à la fin des traces ($a_1.\ldots.a_n.\epsilon$ sera écrit $a_1.\ldots.a_n$).

La longueur d'une trace $t \in Tr$ est notée par $|t|$. L'ensemble des traces d'un processus p, noté par $tr(p)$ est: $tr(p) \stackrel{def}{=} \{w \in \mathcal{R}Act^*|\exists p'$ tel que $p \stackrel{w}{\Longrightarrow} p'\}$.

Pour une trace $s = a_1.\ldots.a_n \in \mathcal{R}Act^*$, l'ensemble des traces préfixes $pre(s)$ est défini par $pre(s) \stackrel{def}{=} \{\epsilon,a_1,a_1.a_2,\ldots,a_1.\ldots.a_n\}$.

On cherche une caractérisation du préordre "may testing" par des traces. Si t est une trace d'un observateur O qui contient l'action $\bar{\omega}$, alors les "traces complémentaires"de t sont les traces que le processus testé p doit être capable de faire pour qu'il satisfasse l'observateur O.

Pour une trace t, $Comp(t)$ est l'ensemble des traces défini par: $Comp : Tr \longrightarrow 2^{Tr}$

$$Comp(t) = \begin{cases} \{\epsilon\} & \text{si } t = \epsilon, \\ \{\bar{a}.s_1 \mid s_1 \in Comp(s)\} & \text{si } t = a:.s \text{ ou } t = a.s,\ a \in Ch, \\ \{a:.s_1,\ a.s_1 \mid s_1 \in Comp(s)\} & \text{si } t = \bar{a}.s,\ a \in Ch. \end{cases} \tag{5.1}$$

$Comp$ sera étendu aux ensembles de traces: si $M \subseteq Tr$, alors $Comp(M) = \bigcup_{s \in M} Comp(s)$.

On va utiliser la notion de complémentarité aussi pour les actions: pour tout $a \in Ch_b$, $\overline{a:} = \bar{a}$, et $\bar{\bar{a}} = a$. Il est facile de remarquer que si $s' \in Comp(s)$ et $\alpha \in \mathcal{R}Act$, alors $\bar{\alpha}.s' \in Comp(\alpha.s)$.

La remarque suivante peut être prouvée par induction sur $|s_2|$.

Remarque 74

– *$s_1 \in Comp(s_2)$ implique $|s_1| = |s_2|$.*

– *La relation Comp est symétrique: $\forall s_1, s_2 \in Tr$ on a ($s_1 \in Comp(s_2)$ si et seulement si $s_2 \in Comp(s_1)$).*

Intuitivement, si $s_i \in tr(p_i)$, alors la "composition"des traces s_1 et s_2 fournit une exécution de l'ensemble $p_1 \parallel p_2$.

Lemme 75

– *Soit $p_1, p_2 \in \mathcal{P}_b$ tel que $p_1 \parallel p_2 \stackrel{\emptyset}{\Longrightarrow} r$. Alors, il existe des processus q_1, q_2 et des traces $s_i \in tr(p_i), i = 1{,}2$, tel que $r = q_1 \parallel q_2$, $p_i \stackrel{s_i}{\Longrightarrow} q_i$, $i = 1{,}2$ et $s_1 \in Comp(s_2)$.*

– *Réciproquement, si $s_i \in tr(p_i), i = 1{,}2$, tel que $r = q_1 \parallel q_2$, $p_i \stackrel{s_i}{\Longrightarrow} q_i$, $i = 1{,}2$ et $s_1 \in Comp(s_2)$, alors $p_1 \parallel p_2 \stackrel{\emptyset}{\Longrightarrow} r$.*

Preuve

On démontre la première partie par induction sur la longueur de la dérivation $p_1 \parallel p_2 \stackrel{\emptyset}{\Longrightarrow} r$ en utilisant le fait que les règles appliquées pour obtenir la dérivation sont (8) ou (9) (ou leurs symétriques) de la Table 5.3

- la longueur de la dérivation est 0, alors il existe les processus $q_1 = p_1$ et $q_2 = p_2$, et les traces $\epsilon \in tr(p_i)$, $i = 1{,}2$, tel que $r = p_1 \parallel p_2$. Évidemment $\epsilon \in Comp(\epsilon)$.

- la propriété est vraie pour tous les $p_1, p_2, r \in \mathcal{P}_b$ tels que $p_1 \parallel p_2 \stackrel{\emptyset}{\Longrightarrow} r$, par une dérivation de longueur au plus n. Supposons que la longueur de la dérivation est $n+1$. Alors elle a la forme: $p_1 \parallel p_2 \stackrel{\alpha}{\longrightarrow} r_1 \stackrel{\emptyset}{\Longrightarrow} r$, où α est τ ou $\bar{a}$, avec $a \in Ch$. Pour le premier pas $p_1 \parallel p_2 \stackrel{\alpha}{\longrightarrow} r_1$ on a une transition obtenue soit par la règle (8) soit par la règle (9) soit par une de leurs symétriques.

- si $\alpha = \tau$ la transition est obtenue par la règle (9) et on a $p_1 \stackrel{\tau}{\longrightarrow} p_1'$, $p_2 \stackrel{\tau:}{\longrightarrow} p_2$ et $p_1 \parallel p_2 \stackrel{\tau}{\longrightarrow} p_1' \parallel p_2$. Alors $r_1 = p_1' \parallel p_2'$, et $p_1' \parallel p_2' \stackrel{\emptyset}{\Longrightarrow} r$ par une dérivation de longueur au plus n, où $p'_2 = p_2$. Par l'hypothèse inductive, il existe les processus q_1, q_2 et les traces $s_i \in tr(p_i), i = 1,2$, tel que $r = q_1 \parallel q_2$, $p'_i \stackrel{s_i}{\Longrightarrow} q_i$, $i = 1,2$ et $s_1 \in Comp(s_2)$. Comme $p_1 \stackrel{\tau}{\longrightarrow} p'_1$, on a que $p_1 \stackrel{s_1}{\Longrightarrow} q_1$ et $p_2 \stackrel{s_2}{\Longrightarrow} q_2$.

- si α est $\bar{a}$ et la transition est obtenue par la règle (8), alors on a $p_1 \stackrel{\bar{a}}{\longrightarrow} p'_1$, $p_2 \stackrel{a}{\longrightarrow} p_2$ et $p_1 \parallel p_2 \stackrel{\bar{a}}{\longrightarrow} p'_1 \parallel p_2$. Alors $r_1 = p'_1 \parallel p'_2$, et $p'_1 \parallel p'_2 \stackrel{\oslash}{\Longrightarrow} r$ par une dérivation de longueur au plus n, où $p'_2 = p_2$. Par l'hypothèse inductive, il existe les processus q_1, q_2 et les traces $s'_i \in tr(p'_i), i = 1,2$, tel que $r = q_1 \parallel q_2$, $p'_i \stackrel{s'_i}{\Longrightarrow} q_i$, $i = 1,2$ et $s'_1 \in Comp(s'_2)$. Comme $p_1 \stackrel{\tau}{\longrightarrow} p'_1$, on a que $p_1 \stackrel{s_1}{\Longrightarrow} q_1$, où $s_1 = \bar{a}.s'_1$. De même $p_2 \stackrel{s_2}{\longrightarrow} p'_2$, où $s_2 = a.s'_2$. Il reste a montrer que $s_1 \in Comp(s_2)$. Conformément à la définition, $Comp(s_2) = Comp(a.s'_2) = \{\bar{a}.s \mid s \in Comp(s'_2)\}$. Mais, par l'hypothèse inductive on a que $s'_1 \in Comp(s'_2)$, c.a.d. $s_1 = \bar{a}.s'_1 \in Comp(s_2)$.

Les autres cas sont similaires.

On démontre la deuxième partie du lemme par induction sur $|s_2|$.

- $s_2 = \epsilon$.

$s_1 \in Comp(s_2) = \{\epsilon\}$ implique $s_1 = \epsilon$, alors $p_i = q_i, i = 1,2$, et comme $r = q_1 \parallel q_2$ on obtient $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$.

$s_2 = \alpha.s'_2$, oú $\alpha \in \{\bar{a}, a\}$.

- $s_2 = a.s'_2$.

$s_1 \in Comp(s_2) = \{\bar{a}.s \mid s \in Comp(s'_2)\}$, alors $s_1 = \bar{a}.s'_1, s'_1 \in Comp(s'_2)$. Par l'hypothèse, $p_i \stackrel{s_i}{\Longrightarrow} q_i$, $i = 1,2$, c.a.d. $p_1 \stackrel{\bar{a}}{\Longrightarrow} p'_1 \stackrel{s'_1}{\Longrightarrow} q_1$ et $p_2 \stackrel{a}{\Longrightarrow} p'_2 \stackrel{s'_2}{\Longrightarrow} q_2$. On a aussi $r = q_1 \parallel q_2$. Utilisant l'hypothèse inductive on a $p'_1 \parallel p'_2 \stackrel{\oslash}{\Longrightarrow} r$. En appliquant une fois la règle (8) et zéro ou plusieurs fois la règle (9) (en fonction du nombre des τ contenus dans $p_1 \stackrel{\bar{a}}{\Longrightarrow} p'_1$ et $p_2 \stackrel{a}{\Longrightarrow} p'_2$), on obtient $p_1 \parallel p_2 \stackrel{\bar{a}}{\Longrightarrow} p'_1 \parallel p'_2 \stackrel{\oslash}{\Longrightarrow} r$, et donc $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$.

- $s_2 = \bar{a}.s'_2$.

$s_1 \in Comp(s_2) = \{a.s,\ a:.s \mid s \in Comp(s'_2)\}$, alors $s_1 = a.s'_1$ ou $s_1 = a:.s'_1$, tel que $s'_1 \in Comp(s'_2)$. Par l'hypothèse, $p_i \stackrel{s_i}{\Longrightarrow} q_i$, $i = 1,2$, c.a.d. $p_1 \stackrel{a}{\Longrightarrow} p'_1 \stackrel{s'_1}{\Longrightarrow} q_1$ ou $p_1 \stackrel{a:}{\Longrightarrow} p'_1 \stackrel{s'_1}{\Longrightarrow} q_1$ et $p_2 \stackrel{\bar{a}}{\Longrightarrow} p'_2 \stackrel{s'_2}{\Longrightarrow} q_2$. On a aussi $r = q_1 \parallel q_2$. Utilisant l'hypothèse inductive on a $p'_1 \parallel p'_2 \stackrel{\oslash}{\Longrightarrow} r$. En appliquant la symétrique de la règle (8) ou de la règle (9), et zéro ou plusieurs fois la règle (9) (en fonction du nombre des τ contenus dans $p_1 \stackrel{\bar{a}}{\Longrightarrow} p'_1$ et $p_2 \stackrel{a}{\Longrightarrow} p'_2$),on obtient $p_1 \parallel p_2 \stackrel{\bar{a}}{\Longrightarrow} p'_1 \parallel p'_2 \stackrel{\oslash}{\Longrightarrow} r$, et donc $p_1 \parallel p_2 \stackrel{\oslash}{\Longrightarrow} r$.

Le cas $s_2 = a:.s'_2$ est similaire au cas $s_2 = a.s'_2$..

$\Box$ *Lemme* 75

La relation *Comp* a la propriété:

Lemme 76 *Pour tous les traces s_0, s_1 et s_2, si $s_0 \in Comp(Comp(s_1))$ et $s_2 \in Comp(s_1)$, alors $s_2 \in Comp(s_0)$.*

Preuve La preuve est par induction sur $|s_1|$.

- $s_1 = \epsilon$

$Comp(\epsilon) = \{\epsilon\}$.

$Comp(Comp(\epsilon)) = \{\epsilon\}$.

$s_0 \in Comp(Comp(s_1))$ implique $s_0 = \epsilon$.

$s_2 \in Comp(s_1)$ implique $s_2 = \epsilon$.

Alors $s_2 \in Comp(s_0)$.

- $s_1 = a.s_1'$

$Comp(a.s_1') = \{\bar{a}.s_2' \mid s_2' \in Comp(s_1')\}$.

$Comp(Comp(a.s_1')) = Comp(\{\bar{a}.s_2' \mid s_2' \in Comp(s_1')\}) = \{a:.s_0', a.s_0' \mid s_0' \in Comp(Comp(s_1'))\}$.

$s_0 \in Comp(Comp(a.s_1'))$ implique $s_0 = a:.s_0'$ ou $s_0 = a.s_0'$ avec $s_0' \in Comp(Comp(s_1'))\}$.

$s_2 \in Comp(s_1)$ implique $s_2 = \bar{a}.s_2'$ avec $s_2' \in Comp(s_1')\}$.

Alors on a $s_2' \in Comp(s_1')$, $s_0' \in Comp(Comp(s_1'))$, et par l'hypothèse inductive, $s_2' \in Comp(s_0')$.

Donc $s_2 = \bar{a}.s_2' \in Comp(s_0)$.

Les autres cas sont analogues.

□ *Lemme* 76

La caractérisation du préordre $\ll_{may}$ par des ensembles de traces:

Théoreme 77 *Pour tous les processus p et q,*

$$p \ll_{may} q \quad ssi \quad tr(p) \subseteq Comp(Comp(tr(q))).$$

D'abord on va définir un ensemble spécial d'observateurs. Si $M \subseteq Ch_b$ est un ensemble fini de canaux, on va noter par

$$in(M) \stackrel{not}{=} \begin{cases} nil & \text{si } M = \emptyset, \\ a.nil + in(M \setminus \{a\}) & \text{autrement, avec } a \in M \text{ quelconque.} \end{cases} \tag{5.2}$$

Soit une trace t et $M \subseteq Ch_b$ un ensemble de canaux. $o_M(t)$ est l'observateur défini par

$$o_M(t) = \begin{cases} \bar{\omega}.nil & \text{si } t = \epsilon, \\ \bar{a}.o_M(s) + in(M) & \text{si } t = a:.s \text{ ou } t = a.s,\ a \in Ch, \\ a.o_M(s) + in(M) & \text{si } t = \bar{a}.s,\ a \in Ch. \end{cases} \tag{5.3}$$

Preuve

$'' \Longrightarrow ''$ Soit $s \in tr(p)$ et soit $M = fn(p,q)$.

Alors, par la construction de $o_M(s)$, il existe $s_2 \in tr(o_M(s))$ tel que $p \stackrel{s}{\Longrightarrow} p'$, $o_M(s) \stackrel{s_2}{\Longrightarrow} \bar{\omega}.nil$, et $s \in Comp(s_2)$.

$p \ll_{may} q$ implique $q \ \underline{may}\ o_M(s)$, c.a.d. $q \parallel o_M(s) \stackrel{\oslash}{\Longrightarrow} q' \parallel \bar{\omega}.nil$.

Le Lemme 75, assure l'existence de $s_0'' \in tr(q)$ et de $s_2'' \in tr(o_M(s))$ tel que $s_2'' \in Comp(s_0'')$ et $o_M(s) \stackrel{s_2''}{\Longrightarrow} \bar{\omega}.nil$.

Par construction, $s_2 \in pre(s_2'')$ et il existe un préfixe $s_0' \in pre(s_0'')$ tel que $s_2 \in Comp(s_0')$ (l'expression $in(M)$ dans la définition de $o_M(s)$ assure que le processus observé ne fait pas des émissions "inattendues").

$s \in Comp(s_2)$ et $s_2 \in Comp(s_0')$ impliquent $s \in Comp(Comp(s_0'))$.

Comme $s_0'' \in tr(q)$ et $s_0' \in pre(s_0'')$ on obtient $s_0' \in tr(q)$.

Alors: $s \in Comp(Comp(s_0')) \subseteq Comp(Comp(tr(q)))$.

$'' \Longleftarrow ''$ Soit $p \ \underline{may}\ o$.

Alors il existe s_1,s_2 avec $s_1 \in Comp(s_2)$ tels que $p \stackrel{s_1}{\Longrightarrow} p'$, $o \stackrel{s_2}{\Longrightarrow} o'$ et $o' \stackrel{\bar{\omega}}{\longrightarrow}$.

$tr(p) \subseteq Comp(Comp(tr(q)))$ implique qu'il existe $s_0 \in tr(q)$ tel que $s_1 \in Comp(Comp(s_0))$, d'où par la symétricité de $Comp$ on obtient $s_0 \in Comp(Comp(s_1))$.

On a $s_0 \in Comp(Comp(s_1))$ et $s_1 \in Comp(s_2)$, et par la Lemme 76, on obtient que $s_0 \in Comp(s_2)$ (en tenant compte de la symtricité de $Comp$).

De $o \stackrel{s_2}{\Longrightarrow} o'$, $s_0 \in Comp(s_2)$, et $s_0 \in tr(q)$, en utilisant la Lemme 75 on a qu'il existe q' tels que $q \parallel o \stackrel{\oslash}{\Longrightarrow} q' \parallel o'$.

Comme $o' \stackrel{\bar{\omega}}{\longrightarrow}$, on obtient $q \ \underline{may}\ o$.

□ *Theoreme 77*

5.2.3 Une caractérisation de "must testing" basée sur des traces

La caractérisation du "may testing" est intuitive et ressemble à celle donnée pour le CCS ((de Nicola et Hennessy, 1984)); la différence est que pour avoir $p \ll_{may} q$ on ne demande plus que l'ensemble $tr(p)$ des traces de p soit contenu dans $tr(q)$, mais que chaque trace s de p soit simulée par une trace $s' \in Comp(Comp(s))$ équivalente dans le sens de la visibilité (rappelons qu'on peut pas savoir directement quand le processus observé reçoit ou ignore une émission).

Au contraire, la caractérisation de "must testing" basée sur des traces est moins évidente, et surtout, elle est complètement différente par rapport aux calculs point à point (CCS ou autres versions). Le fait que les émissions soient autonomes change beaucoup les résultats pour le CBS.

On va dire que p "diverge" (notée $p \Uparrow^{\oslash}$) s'il existe une exécution infinie de $p = p_0 \stackrel{\oslash}{\longrightarrow} p_1 \stackrel{\oslash}{\longrightarrow} \ldots p_k \stackrel{\oslash}{\longrightarrow} \ldots$ tel que $p_k \stackrel{\bar{\omega}}{\not\longrightarrow}$ pour tout $k \in \mathbb{N}$. Au contraire, p "converge" (notée $p \Downarrow^{\oslash}$) ssi $\neg p \Uparrow^{\oslash}$.

On va étendre le prédicat de convergence aux traces comme dans (de Nicola et Hennessy, 1984): pour toute trace $s \in \mathcal{R}Act$, $p \Downarrow_s^{\oslash}$ ssi

- $p \Downarrow_{\epsilon}^{\oslash}$ si $p \Downarrow^{\oslash}$;
- $p \Downarrow_{\alpha.s}^{\oslash}$ si $p \Downarrow^{\oslash}$ et $p \stackrel{\alpha}{\Longrightarrow} p'$ implique $p' \Downarrow_s^{\oslash}$.

On peut prouver que $p \Downarrow_s^{\oslash}$ ssi pour tout préfixe $s_1 \in pre(s)$ on a $p \Downarrow_{s_1}^{\oslash}$.

On va noter $Conv(p,s)$ ssi $\forall s' \in Comp(Comp(s))$, $p \Downarrow_{s'}^{\oslash}$.

On va noter par $\mathcal{S}tab(p,s)$ l'ensemble des processus p', accessibles de p par une trace s' "équivalente" de s et qui ensuite ne peuvent plus évoluer de façon autonome.

$$\mathcal{S}tab(p,s) \stackrel{def}{=} \{p' \mid p \stackrel{s'}{\Longrightarrow} p', s' \in Comp(Comp(s)), p \stackrel{\oslash}{\not\longrightarrow}\}.$$

Alors la caractérisation par traces du "must testing" est donnée par la relation $\leq$.

Définition 78 (caractérisation alternative de $\ll_{must}$)

$p \leq q$ ssi $\forall s \in \mathcal{R}Act^*$, $Conv(p,s)$ implique

- *$Conv(q,s)$;*
- *$\mathcal{S}tab(q,s) \neq \emptyset$ implique $\mathcal{S}tab(p,s) \neq \emptyset$.*

Avant de prouver le théorème principal de cette section, nous avons besoin de deux lemmes auxiliaires.

Si $X =\subseteq Ch_b$ est un ensemble fini de canaux, soit $in'(X)$ le processus défini par

$$in'(X) \stackrel{not}{=} \begin{cases} nil & \text{si } X = \emptyset, \\ a.\bar{\omega} + in'(X \setminus \{a\}) & \text{autrement, avec } a \in X \text{ quelconque.} \end{cases} \tag{5.4}$$

Lemme 79 *Si $p \ll_{must} q$, alors $\forall s \in \mathcal{R}Act^*$, $Conv(p,s)$ implique*

1. *$Conv(q,s)$;*
2. *$Comp(Comp(s)) \cap tr(q) \neq \emptyset$ implique $Comp(Comp(s)) \cap tr(p) \neq \emptyset$.*

Preuve

Soit $p \ll_{must} q$.

1. Supposons qu'il existe $s \in \mathcal{R}Act^*$ tel que $Conv(p,s)$ et $\neg Conv(q,s)$.

   $\neg Conv(q,s)$ implique qu'il existe un prefixe s_1 de s, une trace $t_1 \in Comp(Comp(s_1))$ et un processus q_1 tel que $q \stackrel{t_1}{\Longrightarrow} q_1$ et $q_1 \Uparrow^{\oslash}$.

   Si $X \in Ch_b$ est un ensemble fini de canaux, soit $o_1(t,X)$ l'observateur défini comme suit:

$$o_1(t,X) = \begin{cases} \tau.\bar{\omega} & \text{si } t = \epsilon, \\ \tau.\bar{\omega} + \bar{a}.o_1(s,X) + in'(X) & \text{si } t = a:.s \text{ ou } t = a.s,\ a \in Ch_b, \\ \tau.\bar{\omega} + a.o_1(s,X) + in'(X) & \text{si } t = \bar{a}.s,\ a \in Ch_b. \end{cases} \tag{5.5}$$

Par construction (et l'utilisation du Lemme 75) on a $q \parallel o_1(s_1,fn(p,q)) \overset{\oslash}{\Longrightarrow} q_1 \parallel \tau.\bar{\omega}$ et comme $q_1 \Uparrow^{\oslash}$ on obtient $q\ \underline{m\not{u}st}\ o_1(s_1,fn(p,q))$.
Comme $Conv(p,s)$ et $s_1 \in pre(s)$ on obtient $Conv(p,s_1)$ et donc $p\ \underline{must}\ o_1(s_1,fn(p,q))$, contradiction avec $p \ll_{must} q$.
$in'(X)$ assure que tout "écartement" de la trace s se solde par une exécution réussie.

2. Supposons qu'il existe $s \in \mathcal{R}Act^*$ tel que $Conv(p,s)$, $Comp(Comp(s)) \cap tr(q) \neq \emptyset$ et $Comp(Comp(s)) \cap tr(p) = \emptyset$.
Si $X \in Ch_b$ est un ensmble fini de canaux, soit $o_2(t,X)$ l'observateur défini comme suit:

$$o_2(t,X) = \begin{cases} nil & \text{si } t = \epsilon, \\ \tau.\bar{\omega} + \bar{a}.o_2(s,X) + in'(X) & \text{si } t = a:.s \text{ ou } t = a.s,\ a \in Ch_b, \\ \tau.\bar{\omega} + a.o_2(s,X) + in'(X) & \text{si } t = \bar{a}.s,\ a \in Ch_b. \end{cases} \tag{5.6}$$

Par construction (et l'utilisation du Lemme 75) on a $q \parallel o_2(s,fn(p,q)) \overset{\oslash}{\Longrightarrow} q_1 \parallel nil$ (sans émettre sur le canal ω) et on obtient $q\ \underline{m\not{u}st}\ o_2(s,fn(p,q))$.
Comme $Conv(p,s)$ et $Comp(Comp(s)) \cap tr(p) = \emptyset$ on obtient $p\ \underline{must}\ o_2(s,fn(p,q))$, contradiction avec $p \ll_{must} q$.
$in'(X)$ assure que tout "écartement" de la trace s se solde par une exécution réussie.

□ *Lemme* 79

Lemme 80 *Si $p \leq q$, et $Conv(p,s)$ alors $Comp(Comp(s)) \cap tr(q) \neq \emptyset$ implique $Comp(Comp(s)) \cap tr(p) \neq \emptyset$.*

Preuve

Supposons qu'il existe $s \in \mathcal{R}Act^*$ tel que $Conv(p,s)$, $Comp(Comp(s)) \cap tr(q) \neq \emptyset$.

Comme $p \leq q$, et $Conv(p,s)$, par la définition 78 on obtient $Conv(q,s)$.

$Comp(Comp(s)) \cap tr(q) \neq \emptyset$ implique qu'il existe q_1 et $s_1 \in Comp(Comp(s)) \cap tr(q)$ tel que $q \overset{s_1}{\Longrightarrow} q_1$.

$Conv(q,s)$ implique $q_1 \Downarrow^{\oslash}$.

$q \overset{s_1}{\Longrightarrow} q_1$ et $q_1 \Downarrow^{\oslash}$ impliquent qu'il existe une extension t_1 de la trace s_1 (obtenu de s_1 en ajoutant une séquence finie maximale des actions autonomes que q_1 peut faire) telle que $t_1 \in tr(q)$, $Conv(q,t_1)$, et $\mathcal{S}tab(q,t_1) \neq \emptyset$.

Comme $Conv(p,s)$ et $s_1 \in pre(t_1) \cap Comp(Comp(s))$ et t_1 est obtenu de s_1 en ajoutant à la fin seulement des actions autonomes, on obtient $Conv(p,t_1)$.

Par la Définition 78, $p \leq q$, et $Conv(p,t_1)$ et $\mathcal{S}tab(q,t_1) \neq \emptyset$ implique $\mathcal{S}tab(p,t_1) \neq \emptyset$, et donc $tr(p) \cap Comp(Comp(t_1)) \neq \emptyset$, ce qui implique $Comp(Comp(s)) \cap tr(p) \neq \emptyset$.

$\square$ *Lemme* 80

On peut maintenant prouver l'équivalence des $\ll_{must}$ et $\leq$.

Théoreme 81 *Pour tous les processus p et q,*

$$p \ll_{must} q \quad ssi \quad p \leq q.$$

Preuve

$'' \Longrightarrow ''$

On va prouver que $p \not\leq q$ implique $p \not\ll_{must} q$. Supposons $p \not\leq q$ et $p \ll_{must} q$.
$p \not\leq q$ implique qu'il existe $s \in \mathcal{R}Act^*$, tel que $Conv(p,s)$ et ($\neg Conv(q,s)$ ou ($\mathcal{S}tab(q,s) \neq \emptyset$ et $\mathcal{S}tab(p,s) = \emptyset$)).

- Supposons $Conv(p,s)$ et $\neg Conv(q,s)$.
  Comme $p \ll_{must} q$ et $Conv(p,s)$, par le lemme 79 on obtient $Conv(q,s)$, contradiction.
- Supposons $Conv(p,s)$, $Conv(q,s)$, $\mathcal{S}tab(q,s) \neq \emptyset$ et $\mathcal{S}tab(p,s) = \emptyset$.
  Si $X \in Ch_b$ est un ensmble fini de canaux, soit $o_3(t,X)$ l'observateur défini comme suit:

$$o_3(t,X) = \begin{cases} in'(X) & \text{si } t = \epsilon, \\ \tau.\bar{\omega} + \bar{a}.o_3(s,X) + in'(X) & \text{si } t = a:.s \text{ ou } t = a.s,\ a \in Ch_b, \\ \tau.\bar{\omega} + a.o_3(s,X) + in'(X) & \text{si } t = \bar{a}.s,\ a \in Ch_b. \end{cases} \tag{5.7}$$

$\mathcal{S}tab(q,s) \neq \emptyset$ implique qu'il existe q_1 et $s_1 \in Comp(Comp(s)) \cap tr(q)$ tel que $q \stackrel{s_1}{\Longrightarrow} q_1$ et $q_1 \stackrel{\emptyset}{\not\longrightarrow}$.
Par construction (et l'utilisation du Lemme 75) on a $q \parallel o_3(s,fn(p,q)) \stackrel{\emptyset}{\Longrightarrow} q_1 \parallel in'(X)$ (sans émettre sur le canal ω) et comme $q_1 \stackrel{\emptyset}{\not\longrightarrow}$ on obtient $q\ \underline{m\not{u}st}\ o_3(s,fn(p,q))$.
Comme $\mathcal{S}tab(p,s) = \emptyset$, on peut prouver $p\ \underline{must}\ o_3(s,fn(p,q))$, et donc $p \not\ll_{must} q$.

$'' \Longleftarrow ''$

On va prouver que $p \not\ll_{must} q$ implique $p \not\leq q$. Supposons $p \not\ll_{must} q$ et $p \leq q$.
$p \not\ll_{must} q$ implique qu'il existe o tel que $p\ \underline{must}\ o$ et $q\ \underline{m\not{u}st}\ o$.

Comme $q \ \underline{m\not{u}st}\ o$, un des trois cas suivants doit être satisfait.

- $q \parallel o = q_0 \parallel o_0 \xrightarrow{\oslash} q_1 \parallel o_1 \xrightarrow{\oslash} \ldots \xrightarrow{\oslash} q_n \parallel o_n$ et $q_n \parallel o_n \stackrel{\oslash}{\not\longrightarrow}$, et pour tout $i = 0,n$, $o_i \stackrel{\bar{\omega}}{\not\longrightarrow}$.
  Si on considère les contributions de q et o à cette exécution, par le lemme 75 on obtient qu'il existe $s \in \mathcal{R}Act^*$ et $t \in Comp(s)$ tel que $q \stackrel{s}{\Longrightarrow} q_n$ et $o \stackrel{t}{\Longrightarrow} o_n$.
  De plus, comme $q_n \parallel o_n \stackrel{\oslash}{\not\longrightarrow}$ on obtient $o_n \stackrel{\oslash}{\not\longrightarrow}$ et $q_n \stackrel{\oslash}{\not\longrightarrow}$ et donc $\mathcal{S}tab(q,s) \neq \emptyset$.
  $p\ \underline{must}\ o$, $o \stackrel{t}{\Longrightarrow} o_n$, pour tout $i = 0,n$, $o_i \stackrel{\bar{\omega}}{\not\longrightarrow}$ et $t \in Comp(s)$ impliquent $Conv(p,s)$.
  Par la définition 78, $\mathcal{S}tab(q,s) \neq \emptyset$, $p \leq q$, et $Conv(p,s)$ impliquent $\mathcal{S}tab(p,s) \neq \emptyset$, et donc il existe p_n et $u \in Comp(Comp(s))$ tel que $p \stackrel{u}{\Longrightarrow} p_n$ et $p_n \stackrel{\oslash}{\not\longrightarrow}$.
  De $u \in Comp(Comp(s))$ et $t \in Comp(s)$, par le lemme 76 on obtient $u \in Comp(t)$, et en utilisant le lemme 75 on peut construire l'exécution
  $p \parallel o = p_0 \parallel o_0 \xrightarrow{\oslash} p_1 \parallel o_1 \xrightarrow{\oslash} \ldots \xrightarrow{\oslash} p_n \parallel o_n$ et $p_n \parallel o_n \stackrel{\oslash}{\not\longrightarrow}$, donc $p\ \underline{m\not{u}st}\ o$, contradiction.
- $q \parallel o = q_0 \parallel o_0 \xrightarrow{\oslash} q_1 \parallel o_1 \xrightarrow{\oslash} \ldots \xrightarrow{\oslash} q_n \parallel o_n$, pour tout $i = 0,n$, $o_i \stackrel{\bar{\omega}}{\not\longrightarrow}$ et
  $$(q_n \Uparrow^{\oslash} \text{ ou } o_n \Uparrow^{\oslash}). \tag{5.8}$$
  Si on considère les contributions de q et o à cette exécution, par le lemme 75 on obtient qu'il existe $s \in \mathcal{R}Act^*$ et $t \in Comp(s)$ tel que $q \stackrel{s}{\Longrightarrow} q_n$ et $o \stackrel{t}{\Longrightarrow} o_n$.
  $p\ \underline{must}\ o$, $o \stackrel{t}{\Longrightarrow} o_n$, pour tout $i = 0,n$, $o_i \stackrel{\bar{\omega}}{\not\longrightarrow}$ et $t \in Comp(s)$ impliquent $Conv(p,s)$.
  Par la définition 78, $p \leq q$, et $Conv(p,s)$ impliquent $Conv(q,s)$, et donc
  $$q_n \Downarrow^{\oslash} . \tag{5.9}$$
  $q \stackrel{s}{\Longrightarrow} q_n$ implique $s \in tr(q)$, et comme $p \leq q$, et $Conv(p,s)$, par la lemme 80 on obtient qu'il existe p_n et $u \in Comp(Comp(s))$ tel que $p \stackrel{u}{\Longrightarrow} p_n$.
  De $u \in Comp(Comp(s))$ et $t \in Comp(s)$, par le lemme 76 on obtient $u \in Comp(t)$, et en utilisant le lemme 75 on peut construire l'exécution
  $p \parallel o = p_0 \parallel o_0 \xrightarrow{\oslash} p_1 \parallel o_1 \xrightarrow{\oslash} \ldots \xrightarrow{\oslash} p_n \parallel o_n$, tel que pour tout $i = 0,n$, $o_i \stackrel{\bar{\omega}}{\not\longrightarrow}$.
  Comme $p\ \underline{must}\ o$ on obtient $o_n \Downarrow^{\oslash}$, contradiction avec 5.8 et 5.9.
- $q \parallel o = q_0 \parallel o_0 \xrightarrow{\oslash} q_1 \parallel o_1 \xrightarrow{\oslash} \ldots \xrightarrow{\oslash} q_n \parallel o_n \xrightarrow{\oslash} \ldots$, et pour tout $i \in \mathbb{N}$, $o_i \stackrel{\bar{\omega}}{\not\longrightarrow}$.
  Si on considère les contributions de q et o à cette exécution, par le lemme 75 on obtient qu'il existe les "traces infinies" s et t telles que pour tout $k \in \mathbb{N}$, $s(k) \in \mathcal{R}Act^*$, $t(k) \in Comp(s(k))$, $q \stackrel{s(k)}{\Longrightarrow} q_k$ et $o \stackrel{t(k)}{\Longrightarrow} o_k$.
  Comme $p\ \underline{must}\ o$, par le lemme 75 on obtient qu'il existe $n \in \mathbb{N}$ tel que
  $$tr(p) \cap Comp(Comp(s(n))) = \emptyset. \tag{5.10}$$

(1) $\dfrac{p \xrightarrow{\alpha} p'}{p \xrightarrow{\alpha}_1 p'}$	(2) $\dfrac{p \xrightarrow{a\langle x\rangle} p' \wedge x \notin fn(p)}{p \xrightarrow{a(x)}_1 p'}$	(3) $\dfrac{p \xrightarrow{a:} \wedge\ x \notin fn(p)}{p \xrightarrow{a(x):}_1 p}$

TAB. 5.4 – *La relation* $\xrightarrow{\alpha}_1$

De plus, comme $o \overset{t(n)}{\Longrightarrow}$, $t(n) \in Comp(s(n))$ et $i \in \mathbb{N}$, $o_i \not\xrightarrow{\bar{\omega}}$ on déduit $Conv(p,s(n))$ (sinon on pourrait construire une exécution "divergente" de $p \parallel o$).
$q \overset{s(n)}{\Longrightarrow} q_n$ implique $tr(q) \cap Comp(Comp(s(n))) \neq \emptyset$. Comme de plus $p \leq q$, et $Conv(p,s(n))$, par le lemme 80 on obtient $tr(p) \cap Comp(Comp(s(n))) \neq \emptyset$, contradiction avec 5.10.

$\Box$ *Theoreme* 81

5.3 Caractérisations pour le $b\pi$-calcul

Dans cette section, on va étendre au $b\pi$-calcul monadique, les techniques présentées dans la section précédente. En général, les notations restent inchangées, et on va préciser explicitement les éventuelles différences.

5.3.1 Une caractérisation de "may testing" basée sur des traces

Soit $\xrightarrow{\alpha}_1$ la relation définie dans la Table 5.4.

On rappelle que $\bar{a}(x)$ est une notation pour $\nu x \bar{a}x$. L'ensemble des traces Tr, est défini par la grammaire: $t ::= \epsilon \mid \alpha.t$, où $\alpha \in \mathcal{E}Act = \{\bar{a}x, \bar{a}(x), a\langle x\rangle, a(x), a:, a(x): | a,x \in Ch_b\}$; ϵ représente la trace vide, et dans $\alpha.t$, le préfixe α dénote la première action faite par un processus, et le suffixe t est la trace que le processus peut faire ensuite.

La longueur d'une trace $t \in Tr$ est notée par $|t|$. L'ensemble des traces d'un processus p, noté par $tr(p)$ est: $tr(p) \overset{def}{=} \{w \in \mathcal{E}Act^* | \exists p' \text{ tel que } p \overset{w}{\Longrightarrow}_1 p'\}$.

$a(x)$, $a(x):$ et $\bar{a}(x)$ lient les apparitions de x dans les traces.

On va appeler une trace "fraîche" par rapport à un contexte, si tous ses noms liés sont différents des autres noms présents dans le contexte.

Le lemme suivant admet une preuve similaire au Lemme 13 ou au resultat similaire de π-calcul (voir par exemple la Proposition 2.7 dans (Hennessy, 1991)).

Lemme 82 *Si $p \equiv_\alpha q$ et $p \overset{s}{\Longrightarrow}_1 p'$, pour tout s' fraîche tel que $s \equiv_\alpha s'$, il existe q' tel que $q \overset{s'}{\Longrightarrow}_1 q'$, $p' \equiv_\alpha q'[bn(s)/bn(s')]$ et $q' \equiv_\alpha p'[bn(s')/bn(s)]$.*

Pour une trace t, l'ensemble des traces complémentaires $Comp(t)$ est l'ensemble des traces défini par: $Comp : Tr \longrightarrow 2^{Tr}$

$$Comp(t) = \begin{cases} \{\epsilon\} & \text{si } t = \epsilon, \\ \{\bar{a}x.s_1 \mid s_1 \in Comp(s)\} & \text{si } t = a : .s \text{ ou } t = a\langle x\rangle.s,\ a,x \in Ch, \\ \{\bar{a}(x).s_1 \mid s_1 \in Comp(s)\} & \text{si } t = a(x) : .s \text{ ou } t = a(x).s,\ a,x \in Ch, \\ \{a : .s_1,\ a\langle x\rangle.s_1 \mid s_1 \in Comp(s)\} & \text{si } t = \bar{a}x.s,\ a \in Ch, \\ \{a(x) : .s_1,\ a(x).s_1 \mid s_1 \in Comp(s)\} & \text{si } t = \bar{a}(x).s,\ a \in Ch. \end{cases} \tag{5.11}$$

La remarque suivante peut être prouvée par induction sur $|s_2|$.

Remarque 83

- *$s_1 \in Comp(s_2)$ implique $|s_1| = |s_2|$.*
- *La relation Comp est symétrique: $\forall s_1,s_2 \in Tr$ on a ($s_1 \in Comp(s_2)$ si et seulement si $s_2 \in Comp(s_1)$).*

Intuitivement, si $s_i \in tr(p_i)$, alors la "composition"des traces s_1 et s_2 fournit une exécution de l'ensemble $p_1 \parallel p_2$.

Lemme 84

- *Soit $p_1,p_2 \in \mathcal{P}_b$ tel que $p_1 \parallel p_2 \overset{\oslash}{\Longrightarrow} r$. Alors, il existe les processus q_1, q_2 et les traces $s_i \in tr(p_i), i = 1,2$, tel que $r = q_1 \parallel q_2$, $p_i \overset{s_i}{\Longrightarrow}_1 q_i$, $i = 1,2$ et $s_1 \in Comp(s_2)$.*
- *Réciproquement, si $s_i \in tr(p_i), i = 1,2$, tel que $r = q_1 \parallel q_2$, $p_i \overset{s_i}{\Longrightarrow}_1 q_i$, $i = 1,2$ et $s_1 \in Comp(s_2)$, alors $p_1 \parallel p_2 \overset{\oslash}{\Longrightarrow} r$.*

Preuve

On démontre la première partie par induction sur la longueur de la dérivation $p_1 \parallel p_2 \overset{\oslash}{\Longrightarrow} r$ en utilisant le fait que les règles appliquées pour obtenir la dérivation sont (12) ou (13) (ou leurs symétriques) de la Table 3.5

- la longueur de la dérivation est 0, alors il y a les processus $q_1 = p_1$ et $q_2 = p_2$, et les traces $\epsilon \in tr(p_i), i = 1,2$), tel que $r = p_1 \parallel p_1$. Évidemment $\epsilon \in Comp(\epsilon)$.

- la propriété est vraie pour tous les $p_1,p_2,r \in \mathcal{P}_b$ tels que $p_1 \parallel p_2 \overset{\oslash}{\Longrightarrow} r$, par une dérivation de longueur au plus n. Supposons que la longueur de la dérivation est $n+1$. Alors elle a la forme: $p_1 \parallel p_2 \overset{\alpha}{\longrightarrow} r_1 \overset{\oslash}{\Longrightarrow} r$, où α est τ ou $\bar{a}x$ ou $\bar{a}(x)$, avec $a,x \in Ch$. Pour le premier pas $p_1 \parallel p_2 \overset{\alpha}{\longrightarrow} r_1$ on a une transition obtenue soit par la règle (12) soit par la règle (13) soit par une de leurs symétriques.

- si α est τ la transition est obtenue par la règle (14) et on a $p_1 \xrightarrow{\tau} p_1'$, $p_2 \xrightarrow{\tau:} p_2$ et $p_1 \parallel p_2 \xrightarrow{\tau} p_1' \parallel p_2$. Alors $r_1 = p_1' \parallel p_2'$, et $p_1' \parallel p_2' \xRightarrow{\oslash} r$ par une dérivation de longueur au plus n, où $p_2' = p_2$. Par l'hypothèse inductive, il existe les processus q_1, q_2 et les traces $s_i \in tr(p_i'), i = 1{,}2$, tel que $r = q_1 \parallel q_2$, $p_i' \xRightarrow{s_i}_1 q_i$, $i = 1{,}2$ et $s_1 \in Comp(s_2)$. Comme $p_1 \xrightarrow{\tau} p_1'$, on a que $p_1 \xRightarrow{s_1} q_1$ et $p_2 \xRightarrow{s_2} q_2$.

- si α est $\bar{a}x$ la transition $p_1 \parallel p_2 \xrightarrow{\bar{a}x} p_1' \parallel p_2$ est obtenue par la règle (12) ou (13). On a $p_1 \xrightarrow{\bar{a}x} p_1'$ et soit $p_2 \xrightarrow{a\langle x\rangle} p_2$, soit $p_2 \xrightarrow{a:} p_2$.

Alors $r_1 = p_1' \parallel p_2'$, et $p_1' \parallel p_2' \xRightarrow{\oslash} r$ par une dérivation de longueur au plus n, où $p_2' = p_2$. Par l'hypothèse inductive, il existe les processus q_1, q_2 et les traces $s_i' \in tr(p_i'), i = 1{,}2$, tel que $r = q_1 \parallel q_2$, $p_i' \xRightarrow{s_i}_1 q_i$, $i = 1{,}2$ et $s_1' \in Comp(s_2')$. Comme $p_1 \xrightarrow{\bar{a}x} p_1'$, on a que $p_1 \xRightarrow{s_1}_1 q_1$, où $s_1 = \bar{a}x.s_1'$. De même $p_2 \xRightarrow{s_2}_1 p_2'$, où $s_2 = a : .s_2'$ ou $s_2 = a\langle x\rangle.s_2'$. Il reste a montrer que $s_1 \in Comp(s_2)$. Conformément à la définition, $Comp(s_2) = Comp(a : .s_2') = Comp(a\langle x\rangle.s_2') = \{\bar{a}x.s \mid s \in Comp(s_2')\}$. Mais, par l'hypothèse inductive on a que $s_1' \in Comp(s_2')$, c.a.d. $s_1 = \bar{a}\langle x\rangle.s_1' \in Comp(s_2)$.

- si α est $\bar{a}(x)$ la transition $p_1 \parallel p_2 \xrightarrow{\bar{a}(x)} p_1' \parallel p_2$ est obtenue par la règle (12) ou (13). On a $p_1 \xrightarrow{\bar{a}(x)} p_1'$, $x \notin fn(q)$ et soit $p_2 \xrightarrow{a\langle x\rangle} p_2$, soit $p_2 \xrightarrow{a:} p_2$.

En appliquant les règles de la Table 5.4 on obtient $p_2 \xrightarrow{a(x)}_1 p_2$, ou soit $p_2 \xrightarrow{a(x):} p_2$.

Alors $r_1 = p_1' \parallel p_2'$, et $p_1' \parallel p_2' \xRightarrow{\oslash} r$ par une dérivation de longueur au plus n, où $p_2' = p_2$. Par l'hypothèse inductive, il existe les processus q_1, q_2 et les traces $s_i' \in tr(p_i'), i = 1{,}2$, tel que $r = q_1 \parallel q_2$, $p_i' \xRightarrow{s_i}_1 q_i$, $i = 1{,}2$ et $s_1' \in Comp(s_2')$. Comme $p_1 \xrightarrow{\bar{a}(x)}_1 p_1'$, on a que $p_1 \xRightarrow{s_1}_1 q_1$, où $s_1 = \bar{a}(x).s_1'$. De même $p_2 \xRightarrow{s_2}_1 p_2'$, où $s_2 = a(x) : .s_2'$ ou $s_2 = a(x).s_2'$. Il reste à montrer que $s_1 \in Comp(s_2)$. Conformément à la définition, $Comp(s_2) = Comp(a(x) : .s_2') = Comp(a(x).s_2') = \{\bar{a}(x).s \mid s \in Comp(s_2')\}$. Mais, par l'hypothèse inductive on a que $s_1' \in Comp(s_2')$, c.a.d. $s_1 = \bar{a}(x).s_1' \in Comp(s_2)$.

Les autres cas sont similaires.

On démontre la deuxième partie du lemme par induction sur $|s_2|$.

- $s_2 = \epsilon$.

$s_1 \in Comp(s_2) = \{\epsilon\}$ implique $s_1 = \epsilon$, alors $p_i = q_i, i = (1{,}2)$, et comme $r = q_1 \parallel q_2$ on obtient $p_1 \parallel p_2 \xRightarrow{\oslash} r$.

$s_2 = \alpha.s_2'$, oú $\alpha \in \{\bar{a}x, \bar{a}(x), a\langle x\rangle, a(x), a :, a(x) :\}$.

- $s_2 = a\langle x\rangle.s_2'$.

$s_1 \in Comp(s_2) = \{\bar{a}x.s \mid s \in Comp(s_2')\}$, alors $s_1 = \bar{a}x.s_1', s_1' \in Comp(s_2')$. Par l'hypothèse, $p_i \xRightarrow{s_i}_1 q_i$, $i = 1{,}2$, c.a.d. $p_1 \xrightarrow{\bar{a}x}_1 p_1' \xRightarrow{s_1'}_1 q_1$ et $p_2 \xRightarrow{a\langle x\rangle}_1 p_2' \xRightarrow{s_2'} q_2$. On a aussi $r = q_1 \parallel q_2$.

$p_1 \xrightarrow{\bar{a}x}_1 p_1'$ et $p_2 \xRightarrow{a\langle x\rangle}_1 p_2'$ implique $p_1 \xrightarrow{\bar{a}x} p_1'$ et $p_2 \xRightarrow{a\langle x\rangle} p_2'$

Utilisant l'hypothèse inductive on a $p_1' \parallel p_2' \xRightarrow{\oslash} r$. En appliquant la règle (12), $p_1 \parallel p_2 \xrightarrow{\bar{a}x} p_1' \parallel p_2' \xRightarrow{\oslash} r$, alors $p_1 \parallel p_2 \xRightarrow{\oslash} r$

- $s_2 = \bar{a}(x).s_2'$.

$s_1 \in Comp(s_2) = \{a(x).s, a(x):.s \mid s \in Comp(s_2')\}$, alors $s_1 = a(x).s_1'$ ou $s_1 = a(x):.s_1'$, tel que $s_1' \in Comp(s_2')$.

Par l'hypothèse, $p_i \overset{s_i}{\Longrightarrow}_1 q_i$, $i = 1{,}2$, c.a.d. $p_1 \overset{a(x)}{\longrightarrow}_1 p_1' \overset{s_1'}{\Longrightarrow}_1 q_1$ ou $p_1 \overset{a(x):}{\longrightarrow}_1 p_1' \overset{s_1'}{\Longrightarrow}_1 q_1$ et $p_2 \overset{\bar{a}(x)}{\Longrightarrow}_1 p_2' \overset{s_2'}{\Longrightarrow}_1 q_2$. On a aussi $r = q_1 \parallel q_2$.

$p_1 \overset{a(x)}{\longrightarrow}_1 p_1'$ (respectivement $p_1 \overset{a(x):}{\longrightarrow}_1 p_1'$, $p_2 \overset{\bar{a}(x)}{\Longrightarrow}_1 p_2'$) implique $p_1 \overset{ax}{\longrightarrow} p_1'$ (respectivement $p_1 \overset{a:}{\longrightarrow} p_1'$, $p_2 \overset{\bar{a}(x)}{\Longrightarrow} p_2'$) et $x \notin fn(p_1)$.

Utilisant l'hypothèse inductive on a $p_1' \parallel p_2' \overset{\emptyset}{\Longrightarrow} r$. En appliquant la symétrique de la règle (12) ou de la règle (13), $p_1 \parallel p_2 \overset{\bar{a}}{\longrightarrow} p_1' \parallel p_2' \overset{\emptyset}{\Longrightarrow} r$, et donc $p_1 \parallel p_2 \overset{\emptyset}{\Longrightarrow} r$.

Les autres cas sont similaires.

□ *Lemme* 84

Les lemmes 82 et 84 nous permettent de nous restreindre seulement aux traces fraîches (pour toute exécution réussie (où non-reussie) de $p \parallel o$, il existe une exécution réussie (où non-reussie) telle que les contributions s et t de p et respectivement o sont des traces fraîches). Pour la suite, toutes les traces sont considérés fraîches.

Le lemme suivant admet une preuve similaire à celle donnée pour CBS .

Lemme 85 *Pour tous les traces s_0, s_1 et s_2, si $s_0 \in Comp(Comp(s_1))$ et $s_2 \in Comp(s_1)$, alors $s_2 \in Comp(s_0)$.*

La caractérisation du préordre $\ll_{may}$ par des ensembles de traces (rappelons que dans $tr(p)$ les traces sont considérées fraîches par rapport à $fn(p,q)$):

Théoreme 86 *Pour tous les processus p et q,*

$$p \ll_{may} q \quad ssi \quad tr(p) \subseteq Comp(Comp(tr(q))).$$

Preuve

D'abord on va définir un ensemble spécial d'observateurs. Si $M =\subseteq Ch_b$ est un ensemble fini de canaux, on va noter par

$$in(M) \overset{not}{=} \begin{cases} nil & \text{si } M = \emptyset, \\ a.nil + in(M \setminus \{a\}) & \text{autrement, avec } a \in M \text{ quelconque.} \end{cases} \tag{5.12}$$

$$\langle x \notin M \rangle p \overset{not}{=} \begin{cases} p & \text{si } M = \emptyset, \\ \langle x = a \rangle nil, \langle x \notin M \setminus \{a\} \rangle p & \text{autrement, avec } a \in M \text{ quelconque.} \end{cases} \tag{5.13}$$

Soit une trace t et $M \subseteq Ch_b$ un ensemble de canaux. $o_M(t)$ est l'observateur défini par

$$o_M(t) = \begin{cases} \bar{\omega}.nil + in(M) & \text{si } t = \epsilon, \\ \bar{a}x.o_{M\cup\{x\}}(s) + in(M) & \text{si } t = a : .s \text{ ou } t = ax.s,\ a,x \in Ch, \\ \bar{a}(x).o_{M\cup\{x\}}(s) + in(M) & \text{si } t = a(x) : .s \text{ ou } t = a(x).s,\ a,x \in Ch, \\ a(y).\langle x = y\rangle o_M(s) + in(M) & \text{si } t = \bar{a}x.s,\ a,x \in Ch, \\ a(y).\langle y \notin M\rangle o_{M\cup\{y\}}(s) + in(M) & \text{si } t = \bar{a}(x).s,\ a,x \in Ch. \end{cases} \quad (5.14)$$

Ensuite la preuve suit comme dans la preuve du Théorème 77. Le fait d'avoir dans le $b\pi$-calcul le "if then else" dans sa forme complète (et non seulement le "if then" comme dans le π-calcul) permet de différencier une émission liée d'une émission libre.

$\square$ *Theoreme* 86

5.3.2 Une caractérisation de "must testing" basée sur des traces

La caractérisation alternative "must testing" pour le $b\pi$-calcul est une extension naturelle de celle donnée pour le CBS. Dans cette section on va se contenter de donner seulement les définitions utilisées, le résultat principal, et de présenter dans les preuves, seulement les différences par rapport au CBS.

Les définitions de $p \Uparrow^{\oslash}$ et$p \Downarrow^{\oslash}_s$ restent les mêmes. Dans la définition de $Conv(p,s)$ on prend en compte aussi l'alpha conversion. $Conv(p,s)$ ssi $p \Downarrow^{\oslash}_{s'}$, $\forall s'$ tel que $s' \equiv_\alpha s''$ et $s'' \in Comp(Comp(s))$.

Le lemme suivant admet une justification similaire à celle donnée dans (Hennessy, 1991).

Lemme 87 *Si $bn(s) \cap fn(p) = \emptyset$ alors $Conv(p,s)$ ssi $p \Downarrow^{\oslash}_{s'}$, $\forall s'$ tel que $s' \in Comp(Comp(s))$.*

Les lemmes 82, 84 et 87 nous permettent de nous restreindre seulement aux traces fraîches. Pour la suite, toutes les traces sont considérés fraîches.

On va noter par $\mathcal{S}tab(p,s)$ l'ensemble des processus p', accessibles de p par une trace s' "équivalente" de s et qui ensuite ne peuvent plus évoluer de façon autonome.

$$\mathcal{S}tab(p,s) \overset{def}{=} \{p' \mid p \overset{s'}{\Longrightarrow} p', s' \equiv_\alpha s'', s'' \in Comp(Comp(s)), p \overset{\oslash}{\not\longrightarrow}\}.$$

Alors la caractérisation par traces du "must testing" est donnée par la relation $\leq$.

Définition 88 (caractérisation alternative de $\ll_{must}$)

$p \leq q$ *ssi* $\forall s \in \mathcal{E}Act^*$, *Conv*$(p,s)$ *implique*

– *Conv*(q,s);

– $\mathcal{S}tab(q,s) \neq \emptyset$ *implique* $\mathcal{S}tab(p,s) \neq \emptyset$.

Comme dans la Section 5.2.3, on peut prouver le théorème suivant.

Théoreme 89 *Pour tous les processus p et q,*

$$p \ll_{must} q \quad ssi \quad p \leq q.$$

La preuve suit le même chemin, les observateurs utilisés étant présentés ci-dessous.

Si $X \in Ch_b$ est un ensemble fini de canaux, soit $o_1(t,X)$ $o_2(t,X)$ $o_3(t,X)$ les observateurs définis comme suit:

$$o_1(t,X) = \begin{cases} \tau.\bar{\omega} & \text{si } t = \epsilon, \\ \tau.\bar{\omega} + \bar{a}x.o_1(s,X) + in'(X) & \text{si } t = a : .s \text{ ou } t = a\langle x\rangle.s,\ a,x \in Ch_b, \\ \tau.\bar{\omega} + \bar{a}(x).o_1(s,X \cup \{x\}) + in'(X) & \text{si } t = a(x) : .s \text{ ou } t = a(x).s,\ a,x \in Ch_b, \\ \tau.\bar{\omega} + a(y).\langle x = y\rangle o_1(s,X),\bar{\omega} + in'(X) & \text{si } t = \bar{a}x.s,\ a,x \in Ch_b, \\ \tau.\bar{\omega} + a(y).\langle y \notin X\rangle o_1(s,X \cup \{y\}) + in'(X) & \text{si } t = \bar{a}(x).s,\ a,x \in Ch_b. \end{cases} \tag{5.15}$$

$$o_2(t,X) = \begin{cases} nil & \text{si } t = \epsilon, \\ \tau.\bar{\omega} + \bar{a}x.o_2(s,X) + in'(X) & \text{si } t = a : .s \text{ ou } t = a\langle x\rangle.s,\ a \in Ch_b, \\ \tau.\bar{\omega} + \bar{a}(x).o_2(s,X \cup \{x\}) + in'(X) & \text{si } t = a(x) : .s \text{ ou } t = a(x).s,\ a,x \in Ch_b, \\ \tau.\bar{\omega} + a(y).\langle x = y\rangle o_2(s,X),\bar{\omega} + in'(X) & \text{si } t = \bar{a}x.s,\ a,x \in Ch_b, \\ \tau.\bar{\omega} + a(y).\langle y \notin X\rangle o_2(s,X \cup \{y\}) + in'(X) & \text{si } t = \bar{a}(x).s,\ a,x \in Ch_b. \end{cases} \tag{5.16}$$

$$o_3(t,X) = \begin{cases} in'(X) & \text{si } t = \epsilon, \\ \tau.\bar{\omega} + \bar{a}x.o_3(s,X) + in'(X) & \text{si } t = a : .s \text{ ou } t = a\langle x\rangle.s,\ a \in Ch_b, \\ \tau.\bar{\omega} + \bar{a}(x).o_3(s,X \cup \{x\}) + in'(X) & \text{si } t = a(x) : .s \text{ ou } t = a(x).s,\ a \in Ch_b, \\ \tau.\bar{\omega} + a(y).\langle x = y\rangle o_3(s,X),\bar{\omega} + in'(X) & \text{si } t = \bar{a}x.s,\ a,x \in Ch_b, \\ \tau.\bar{\omega} + a(y).\langle y \notin X\rangle o_3(s,X \cup \{y\}) + in'(X) & \text{si } t = \bar{a}(x).s,\ a,x \in Ch_b. \end{cases} \tag{5.17}$$

où, cette fois

$$\langle x \notin M \rangle p \stackrel{not}{=} \begin{cases} p & \text{si } M = \emptyset, \\ \langle x = a \rangle \bar{\omega}, \langle x \notin M \setminus \{a\} \rangle p & \text{autrement, avec } a \in M \text{ quelconque.} \end{cases} \tag{5.18}$$

si $M \subseteq Ch_b$ est un ensemble fini de canaux.

5.4 Conclusion

Dans ce chapitre on a présenté des caractérisations basées sur des traces pour les relations induites par des tests: le "must testing" et le "may testing", pour une variante sans passage de valeur de CBS et pour le $b\pi$-calcul. Ces caractérisations sont indépendantes des observateurs. A notre connaissance, de telles études n'avaient pas été menées auparavant dans les modèles à diffusion. Comme possible continuation de l'étude, il reste à trouver une caractérisation équationnelle pour $\ll_{must}$ et $\ll_{may}$ qui soit complète pour la partie finie de CBS ou du $b\pi$-calcul.

Chapitre 6

Expressivité du $b\pi$-calcul

De nos jours, l'étude des divers langages à travers leurs traductions dans d'autres langages est devenue presque "un sport". L'intérêt est d'une part de pouvoir tester l'expressivité d'un (nouveau) langage par rapport à un langage où un calcul de base déjà existent (et d'habitude bien étudié); d'autre part (par l'expression d'un constructeur à l'aide seulement des autre constructions du langage), il est intéressant de pouvoir exhiber pour un langage un "noyau" qui préserve tout le pouvoir du langage, mais qui est plus simple à manipuler dans des preuves formelles, ou plus facile à implanter.

Dans la littérature on peut trouver beaucoup d'exemples de traductions dans le λ -calcul (typé ou non typé). En ce qui concerne les algèbres de processus, on peut classifier les divers encodages en deux grandes classes:

- les encodages d'un langage plus riche dans un sous langage:
  - le codage du $HO\pi$ -calcul (l'extension d'ordre supérieur du π -calcul) dans le π-calcul (Sangiorgi, 1992);
  - la traduction du π_{1l}-calcul (une extension “distribuée”du π_1-calcul) dans le π_1-calcul (un sous-calcul du π-calcul asynchrone) (Amadio, 2000);
  - l'implantation des communications synchrones par des communications asynchrones dans le π_{-}-calcul (le calcul sans choix "+")(Boudol, 1992);
  - la traduction du π -calcul polyadique dans le π-calcul monadique ((Milner, 1993), (Quaglia, 1996));
  - la traduction du π_a -calcul (le fragment asynchrone du π-calcul) dans le sous -langage du π_a -calcul qui ne contient pas l'opérateur de choix ((Nestmann, 1996));
- les encodages de langages divers dans des langages complètement différents:
  - le codage des λ -calculs dans les algèbres de processus ((Odersky, 1995b),

(Niehren, 1996), (Sangiorgi, 1998));
– les traductions entre CSP et CCS (Millington, 1987);
– le codage de π-calcul dans CCS (Banach et van Breugel, 1998);
– l'implantation des structures de données et des constructions séquentielles dans le π-calcul ((Odersky, 1995a), (Turner, 1996), (Rockl et Sangiorgi, 1999));
– la traduction des langages orientés objet dans le π-calcul ((Walker, 1995), (Pierce et Turner, 1995)).

Alternativement aux encodages, il est parfois intéressant de prouver des résultats de séparation (par exemple (Bouge, 1988), (Palamidessi, 1997), (Prasad, 1987), (Prasad, 1989)) qui montrent l'intérêt d'étudier certains langages (ou extensions), ou la nécessité de garder certains constructeurs comme étant basiques.

Dans ce chapitre, dans un premier temps nous nous intéressons aux relations entre les communications point à point et les primitives de diffusion. Utilisant l'existence (ou la non-existence) des solutions pour l'élection d'un leader dans un système distribué, on va montrer qu'il n'est pas possible d'implanter (sous certaines conditions) des langages basés sur des communications point à point (comme CCS ou π-calcul) dans des langages à diffusion (comme CBS ou $b\pi$ -calcul).

Ensuite, nous analysons l'intérêt d'avoir un opérateur de choix dans le $b\pi$ - calcul. Utilisant une relation préservée par tous les opérateurs du langage à l'exception de $+$, on va prouver l'impossibilité d'implanter le choix à l'aide des autres constructions du $b\pi$-calcul.

6.1 Des résultats de séparation entre des langages point à point et des langages à diffusion

6.1.1 Système symétrique et système électoral

Dans cette sous-section, nous présentons les notions de système distribué symétrique et de système électoral suivant les définitions données par Palamidessi dans (Palamidessi, 1997). Les définitions sont les mêmes dans le π-calcul et dans le $b\pi$-calcul.

Un *cluster* est un système de processus qui s'exécutent en parallèlle $P = P_1 \parallel P_2 \parallel \ldots \parallel P_n$. Une *exécution* C du cluster est une séquence de transitions (éventuellement ω-infinie)[1]:

1. Comme dans (Palamidessi, 1997), dans le soucis de garder les notations simples, nous supposons que chaque νx est poussé vers "l'extérieur" par l'utilisation répétée des règles de congruence structurelle. En plus, nous ne représentons pas explicitement les "ν" qui sont à l'extérieur (avec l'exception du Lemme 101), et nous supposons que le cluster ne fera pas d'actions visibles sur les noms bornés.

$$
\begin{aligned}
P_1 \parallel P_2 \parallel \ldots \parallel P_n &\xrightarrow{\alpha_1} P_1^1 \parallel P_2^1 \parallel \ldots \parallel P_n^1 \\
&\xrightarrow{\alpha_2} P_1^2 \parallel P_2^2 \parallel \ldots \parallel P_n^2 \\
&\vdots \\
&\xrightarrow{\alpha_m} P_1^m \parallel P_2^m \parallel \ldots \parallel P_n^m \\
&(\xrightarrow{\alpha_{m+1}} \ldots \quad)
\end{aligned}
$$

Si $\tilde{\alpha} = \alpha_1.\alpha_2.\ldots.\alpha_m$, nous allons aussi représenter l'exécution C par $C : P \stackrel{\tilde{\alpha}}{\Longrightarrow} P^m$ (ou par $C : P \stackrel{\tilde{\alpha}}{\Longrightarrow} \ldots$ si C est infinie).

C' *étend* C si $C : P \stackrel{\tilde{\alpha}}{\Longrightarrow} P^m$, et s'il existe $C'' : P^m \stackrel{\tilde{\alpha}}{\Longrightarrow} P^{m+m'}$ ou $C'' : P^m \stackrel{\tilde{\alpha}}{\Longrightarrow} \ldots$ tel que $C' = CC''$, où les deux apparitions de P^m ont été identifiées. La *projection* de C sur P_i (C, P présentées ci-dessus), notée $Proj(C,i)$, est définie comme étant la "contribution"de P_i à l'exécution C.

Comme dans (Palamidessi, 1997), pour la définition d'un système électoral, nous supposons que Ch_p et Ch_b contiennent les entiers $\mathbb{N}$, qui seront utiles pour représenter l'identité des processus dans un réseau. Notre définition est moins restrictive que celle utilisé dans (Palamidessi, 1997).

Soit

$$\mathcal{C}(P) \stackrel{def}{=} \{C|C \text{ est une exécution de } P\}$$

l'ensemble de toutes les exécutions de P , et soit

$$\mathcal{C}_1(P) \stackrel{def}{=} \{C|C \text{ est une exécution de } P, \exists k \in \{1,\ldots,n\}, C \text{ contient } \bar{k}\}$$

$$\mathcal{C}_2(P) \stackrel{def}{=} \{C|C \text{ est une exécution de } P, \exists l,k \in \{1,\ldots,n\}, l \neq k, C \text{ contient } \bar{k},\bar{l}\}.$$

Définition 90 (Système électoral)

- *Un cluster $P = P_1 \parallel P_2 \parallel \ldots \parallel P_n$ est un* **système électoral (fort)** *si $\mathcal{C}_2(P) = \emptyset$ et pour toute exécution $C \in \mathcal{C}(P)$, il existe une extension C' de C tel que $C' \in \mathcal{C}_1(P)$.*
- *Un cluster $P = P_1 \parallel P_2 \parallel \ldots \parallel P_n$ est un* **système électoral faible** *si $\mathcal{C}_2(P) = \emptyset$ et $\mathcal{C}_1(P) \neq \emptyset$.*

Remarquons que pour tout système électoral fort P, une exécution infinie C doit contenir déjà l'émission exigée ($C \in \mathcal{C}_1(P)$), car elle ne peut plus être étendue; de plus, pour tout système électoral (faible ou fort) P, il existe toujours une exécution finie C tel que $C \in \mathcal{C}_1(P)$.

Pour pouvoir définir un système symétrique, on introduit d'abord les notions d'hyper-graphe, et hyper-graphe associé à un cluster.

Un hyper-graphe généralise le concept de graphe permettant à une arête de lier plusieurs sommets.

Définition 91 (Hyper-graphe)
Un hyper-graphe est un triplé $H = (N,X,t)$ où N, X sont des ensembles finis de noeuds *et respectivement* d'arêtes, *et t est la fonction qui associe à une arête $x \in X$ l'ensemble de noeuds connectés par x.*

Étant donné un hyper-graphe $H = (N,X,t)$, un *automorphisme* sur H est une paire $\sigma = (\sigma_N,\sigma_X)$ tel que $\sigma_N : N \longrightarrow N$ et $\sigma_X : X \longrightarrow X$ sont des permutations qui préservent les arêtes: si $t(x) = \{n_1,\dots,n_k\}$, alors $t(\sigma_X(x)) = \{\sigma_N(n_1),\dots,\sigma_N(n_k)\}$

Étant donné un hyper-graphe H et σ un automorphisme sur H, *l'orbite* de $n \in N$ généré par σ est l'ensemble de noeuds $O(n) = \{n,\sigma(n),\dots,\sigma^{h-1}(n)\}$, tel que σ^h coïncide avec l'identité.

Intuitivement, on va associer un hyper-graphe à un cluster de la façon suivante: les processus P_i seront les noeuds, et les noms libres seront les arêtes (chaque nom - arête sera partagé par les noeuds - processus qui le contiennent).

Définition 92 (Hyper-graphe associé à un cluster)
Étant donné un cluster $P = P_1 \parallel P_2 \parallel \dots \parallel P_n$, l'hyper-graphe associé à P, est $H(P) = (N,X,t)$ où $N = \{1,\dots,n\}$, $X = fn(P)$ et pour chaque arête $x \in X$, $t(x) = \{i \mid x \in fn(P_i)\}$.

Un cluster P est symétrique par rapport à un automorphisme σ sur $H(P)$ si et seulement si pour chaque i, le processus associé à un noeud $\sigma(i)$ est identique (jusqu'au alpha-conversion) au processus obtenu par σ-renommage du processus associé au noeud i.

Définition 93 (Système symétrique)
Soit $P = P_1 \parallel P_2 \parallel \dots \parallel P_n$ un cluster, et σ un automorphisme sur l'hyper-graphe $H(P) = (N,X,t)$. Alors P est symétrique par rapport à σ si et seulement si pour tout $i \in N$, $P_{\sigma(i)} \equiv \sigma(P_i)$; P est symétrique s'il est symétrique par rapport à tous les automorphismes sur $H(P)$.

6.1.2 Codage uniforme

Dans cette section nous présentons la définition du codage uniforme, et nous prouvons quelques résultats associés.

Quand on implante un terme d'un calcul dans un autre, on cherche en général une translation qui ne dépende pas du contexte, c.a.d. que le codage de t_1 dans $C'[t_1]$ et dans $C''[t_1]$ soit le même, indifféremment des contextes C' et C''. Donc on exige généralement que le codage soit compositionnel. En ce qui concerne les systèmes concurrents, une bonne propriété de l'implantation est qu'elle préserve la distribution:

$$[\![P \parallel Q]\!] = [\![P]\!] \parallel [\![Q]\!] \tag{6.1}$$

De plus, il semble "raisonnable" d'exiger que le codage “se comporte bien” par rapport aux renommages (substitutions):

$$[\![\sigma(P)]\!] = \sigma([\![P]\!]) \tag{6.2}$$

En dehors des conditions 6.1 et 6.2, les propriétés qui sont en général requises d'un codage peuvent être classifiées dans deux catégories (Nestmann, 1996):

- *abstraction totale*: le codage doit préserver (dans les deux sens) certaines relations d'équivalence, $S_1 \sim_s S_2$ si et seulement si $[\![S_1]\!] \sim_c [\![S_2]\!]$ où $\sim_s$ et $\sim_c$ sont des équivalences du langage source, et respectivement du langage cible.
- *correspondance opérationnelle*: le codage doit assurer une certaine correspondance entre les exécutions d'un terme S dans le langage source et les exécutions de l'implantation du terme $[\![S]\!]$ dans le langage cible.

Nous appelons codage *uniforme* un codage qui satisfait (6.1) et (6.2), et qui en plus implante des émissions dans le calcul source par des émissions correspondantes dans le calcul cible.

Définition 94 (Codage uniforme)

Un codage $[\![\ ]\!] : \mathcal{P}_x \longrightarrow \mathcal{P}_y$ (où $(x,y) \in \{(p,b),(b,p)\}$) est appelé uniforme, s'il satisfait les conditions suivantes (où $\tilde{\alpha}$, $\tilde{\beta}$ et $\tilde{\gamma}$ dénotent des séquences finies d'actions):

1. $[\![P \parallel Q]\!] = [\![P]\!] \parallel [\![Q]\!]$;
2. $[\![\sigma(P)]\!] = \sigma([\![P]\!])$, *pour toute substitution* σ;
3. $(\exists\tilde{\alpha} : P \stackrel{\tilde{\alpha}}{\Longrightarrow} P'$ *si et seulement si* $\exists\tilde{\beta} : [\![P]\!] \stackrel{\tilde{\beta}}{\Longrightarrow} [\![P']\!])$, *où pour tout* $a \in fn(P)$, $(\exists\alpha \in \tilde{\alpha}$ *tel que* $sub(\alpha) = a$ *et* $type(\alpha) = output$ *si et seulement si* $\exists\beta \in \tilde{\beta}$ *tel que* $sub(\beta) = a$ *et* $type(\beta) = output)$;
4. *si* $[\![P]\!] \stackrel{\tilde{\beta}}{\Longrightarrow} Q$ *alors* $\exists\tilde{\gamma} : [\![P]\!] \stackrel{\tilde{\beta}}{\Longrightarrow} Q \stackrel{\tilde{\gamma}}{\Longrightarrow} [\![P']\!]$;
5. *si* $C : [\![P]\!] \stackrel{\tilde{\beta}}{\Longrightarrow} \ldots$ *est une exécution infinie, alors il existe un processus* Q *et les exécutions non vides* $C_1 : [\![P]\!] \stackrel{\tilde{\beta_1}}{\Longrightarrow} [\![Q]\!]$ *et* $C_2 : [\![Q]\!] \stackrel{\tilde{\beta_2}}{\Longrightarrow} \ldots$ *tel que* $C = C_1C_2$.

Remarquons que les conditions 4. et 5. assurent qu'un codage uniforme n'introduit pas de divergence: une exécution infinie de $[\![P]\!]$ est induite par une exécution infinie de P.

Nous montrons maintenant que les conditions de la Définition 94 sont suffisamment fortes pour assurer que tout système électoral fort (faible) du $b\pi$-calcul est implanté dans un système électoral fort (faible) du π-calcul.

Lemme 95 *Tout codage uniforme implante un système électoral (faible)P du $b\pi$-calcul dans un système électoral (faible) $[\![P]\!]$ du π-calcul.*

Preuve

– Soit $P = P_1 \parallel P_2 \parallel \ldots \parallel P_n$ un système électoral fort.

Nous avons

$$R \stackrel{def}{=} [\![P_1 \parallel P_2 \parallel \ldots \parallel P_n]\!] = [\![P_1]\!] \parallel [\![P_2]\!] \parallel \ldots \parallel [\![P_n]\!] = R_1 \parallel R_2 \parallel \ldots \parallel R_n$$

où $R_i \stackrel{def}{=} [\![P_i]\!]$.

Nous allons prouver que $R \stackrel{def}{=} [\![P]\!]$ est un système électoral fort. Nous devons prouver que $\mathcal{C}_2([\![P]\!]) = \emptyset$ et que pour toute exécution $D \in \mathcal{C}([\![P]\!])$ il existe une extension D' de D tel que $D' \in \mathcal{C}_1([\![P]\!])$.

Supposons $\mathcal{C}_2([\![P]\!]) \neq \emptyset$.

Soit

$$D : R = R_1 \parallel R_2 \parallel \ldots \parallel R_n \stackrel{\tilde{\beta}}{\Longrightarrow} Q$$

une exécution de $\mathcal{C}_2([\![P]\!])$ (si $\mathcal{C}_2([\![P]\!]) \neq \emptyset$, on peut toujours obtenir une exécution finie de $\mathcal{C}_2([\![P]\!])$).

Alors $\exists l,k \in \{1,\ldots,n\}$ tel que $\bar{k},\bar{l} \in \tilde{\beta}$ et $l \neq k$.

Par la Condition 4. de la Définition 94 nous obtenons qu'il existe E, extension de D,

$$E : R = [\![P]\!] \stackrel{\tilde{\beta}}{\Longrightarrow} Q \stackrel{\tilde{\gamma}}{\Longrightarrow} R' = [\![P']\!].$$

Par la Condition 3. de la Définition 94, nous obtenons l'exécution correspondante

$$C : P \stackrel{\tilde{\alpha}}{\Longrightarrow} P'$$

tel que $\bar{k},\bar{l} \in \tilde{\alpha}$ et donc $C \in \mathcal{C}_2(P)$, contradiction avec le fait que P est un système électoral fort.

Soit $D \in \mathcal{C}([\![P]\!])$,

$$D : R = R_1 \parallel R_2 \parallel \ldots \parallel R_n \stackrel{\tilde{\beta}}{\Longrightarrow} Q$$

une exécution finie de $[\![P]\!]$.

Par la Condition 4. de la Définition 94 nous obtenons qu'il existe E, extension de D,

$$E : R = [\![P]\!] \stackrel{\tilde{\beta}}{\Longrightarrow} Q \stackrel{\tilde{\gamma}}{\Longrightarrow} R' = [\![P']\!].$$

Par la Condition 3. de la Définition 94, nous obtenons l'exécution correspondante

$$C : P \stackrel{\tilde{\alpha}}{\Longrightarrow} P'.$$

Comme P est un système électoral fort, il existe une extension C' de C

$$C : P \stackrel{\tilde{\alpha}}{\Longrightarrow} P' \stackrel{\tilde{\alpha}'}{\Longrightarrow} P''$$

et il existe $k \in \{1, \ldots, n\}$ (le "leader") tel que C' contient $\bar{k}$. Par la Condition 3. de la Définition 94 nous obtenons l'exécution

$$D' : R = [\![P]\!] \stackrel{\tilde{\beta}}{\Longrightarrow} Q \stackrel{\tilde{\gamma}}{\Longrightarrow} [\![P']\!] \stackrel{\tilde{\beta}'}{\Longrightarrow} [\![P'']\!]$$

tel que $\bar{k} \in \tilde{\beta}\tilde{\gamma}\tilde{\beta}'$ et donc $D' \in \mathcal{C}_1([\![P]\!])$.

Soit

$$D : R = R_1 \parallel R_2 \parallel \ldots \parallel R_n \stackrel{\tilde{\beta}}{\Longrightarrow} \ldots$$

une exécution infinie de $\mathcal{C}([\![P]\!])$. Supposons que $D \notin \mathcal{C}_1([\![P]\!])$. On va construire une exécution

$$C : P = P_1 \parallel P_2 \parallel \ldots \parallel P_n \stackrel{\tilde{\alpha}}{\Longrightarrow} \ldots$$

tel que $C \notin \mathcal{C}_1(P)$. Plus exactement, on va construire (par induction sur m) une séquence d'exécutions non vides

$$C^m : P^{m-1} \stackrel{\tilde{\alpha_m}}{\Longrightarrow} P^m \quad \text{et} \quad D^m : [\![P^{m-1}]\!] \stackrel{\tilde{\beta_m}}{\Longrightarrow} [\![P^m]\!]$$

tel que D étend $D^1 \ldots D^m$ et C^m et D^m satisfont la Condition 3. de la Définition 94. (6.3)

Soit $C^1 = P \stackrel{\tilde{\epsilon}}{\Longrightarrow} P$ l'exécution vide. Évidemment

$$D^1 : [\![P]\!] \stackrel{\tilde{\epsilon}}{\Longrightarrow} [\![P]\!]$$

satisfait les conditions pour 6.3.

Supposons $C^m : P^{m-1} \stackrel{\tilde{\alpha}_m}{\Longrightarrow} P^m$ construite dans les hypothèses 6.3.

Alors D étend

$$D^1 \dots D^m : [\![P]\!] \stackrel{\tilde{\beta}_m}{\Longrightarrow} [\![P^m]\!]$$

et C^m et D^m satisfont la Condition 3. de la Définition 94. Comme D étend

$$D^1 \dots D^m : [\![P]\!] \stackrel{\tilde{\beta}_1 \dots \tilde{\beta}_m}{\Longrightarrow} [\![P^m]\!]$$

il existe

$$E^m : [\![P^m]\!] \stackrel{\tilde{\gamma}_m}{\Longrightarrow} \dots \quad \text{tel que} \quad D = D^1 \dots D^m E^m.$$

Par la Condition 5. de la Définition 94, il existe P^{m+1} et les exécutions non vides

$$D^{m+1} : [\![P^m]\!] \stackrel{\tilde{\beta}_{m+1}}{\Longrightarrow} [\![P^{m+1}]\!] \quad \text{et} \quad E^{m+1} : [\![P^{m+1}]\!] \stackrel{\tilde{\gamma}_{m+1}}{\Longrightarrow} \dots$$

tels que $D = D^1 \dots D^{m+1} E^{m+1}$.

Par la Condition 3. de la Définition 94, nous obtenons l'exécution correspondante $C^{m+1} : P^m \stackrel{\tilde{\alpha}_{m+1}}{\Longrightarrow} P^{m+1}$ et C^{m+1} et D^{m+1} satisfont la Condition 3. de la Définition 94.

Si $B^m : P \stackrel{\tilde{\alpha}_1 \dots \tilde{\alpha}_m}{\Longrightarrow} P^m$, par les conditions de 6.3, et comme $D \notin \mathcal{C}_1([\![P]\!])$ par l'utilisation de la Condition 3. dans la Définition 94, on obtient que $B^m \notin \mathcal{C}_1(P)$ pour tout $m \in \mathbb{N}$. Comme par construction B^{m+1} étend strictement B^m, on obtient une exécution infinie $B \notin \mathcal{C}_1(P)$, contradiction avec le fait que P est un système électoral fort.

– Soit $P = P_1 \parallel P_2 \parallel \dots \parallel P_n$ un système électoral faible.

Nous avons

$$R \stackrel{def}{=} [\![P_1 \parallel P_2 \parallel \dots \parallel P_n]\!] = [\![P_1]\!] \parallel [\![P_2]\!] \parallel \dots \parallel [\![P_n]\!] = R_1 \parallel R_2 \parallel \dots \parallel R_n$$

où $R_i \stackrel{def}{=} [\![P_i]\!]$.

Nous allons prouver que $R \stackrel{def}{=} [\![P]\!]$ est un système électoral faible. Nous devons prouver que $\mathcal{C}_2([\![P]\!]) = \emptyset$ et $\mathcal{C}_1([\![P]\!]) \neq \emptyset$.

Le fait que $\mathcal{C}_2([\![P]\!]) \neq \emptyset$ est prouvé comme dans le cas précédent.

Comme P est un système électoral faible, il existe une exécution

$$C : P \stackrel{\tilde{\alpha}}{\Longrightarrow} P'$$

telle que $C \in \mathcal{C}_1(P)$ ($\exists k \in \{1, \dots, n\}$ tel que $\bar{k} \in \tilde{\alpha}$).

Par l'utilisation de la condition 3. de la Définition 94, nous obtenons l'exécution correspondante

$$D : R = [\![P]\!] \stackrel{\tilde{\beta}}{\Longrightarrow} [\![P']\!]$$

tel que $\bar{k} \in \tilde{\beta}$, et donc $\mathcal{C}_1([\![P]\!]) \neq \emptyset$.

□ *Lemme* 95

Remarquons que pour montrer que tout codage uniforme implante un système électoral faible P du langage source dans un un système électoral faible $[\![P]\!]$ du langage cible on n'utilise pas la condition 5. de la Définition 94.

6.1.3 Résultats de séparation

Dans cette section nous présentons des résultats de séparation entre des langages point à point et des langages à diffusion.

Lemme 96 *Soit* $P_i = \bar{a}i \parallel a(x).\bar{x}$.

Pour tout $n \geq 2$, $P(n) = \prod_{i=1}^{n} P_i$ *est un système électoral faible.*

Preuve

Soit

$$C : P(n) = P_1 \parallel P_2 \parallel \ldots \parallel P_n \xrightarrow{\alpha} P_1^1 \parallel P_2^1 \parallel \ldots \parallel P_n^1 = P^1(n)$$

un pas de $P(n)$. Alors il existe $k \in \{1, \ldots, n\}$ tel que $\alpha = \bar{a}k$ or il existe b tel que $\alpha = ab$. Dans le premier cas, nous pouvons étendre C à C':

$$\begin{aligned} C' : P(n) = P_1 \parallel \ldots \parallel P_k \parallel \ldots \parallel P_n &\xrightarrow{\bar{a}k} \bar{a}1 \parallel \bar{k} \parallel \ldots \parallel nil \parallel \bar{k} \parallel \ldots \parallel \bar{a}n \parallel \bar{k} \\ &\xrightarrow{\bar{k}} \bar{a}1 \parallel nil \parallel \ldots \parallel nil \parallel \bar{k} \parallel \ldots \parallel \bar{a}n \parallel \bar{k} \\ &\vdots \\ &\xrightarrow{\bar{k}} \bar{a}1 \parallel nil \parallel \ldots \parallel nil \parallel nil \parallel \ldots \parallel \bar{a}n \parallel nil \end{aligned}$$

On a $C' \in \mathcal{C}_1(P)$ et on peut prouver que toute extension de C ne peut pas se trouver dans $\mathcal{C}_2(P)$.

Si $\alpha = ab$ et $k \in \{1, \ldots, n\}$, nous procédons de la même façon, si non, toute extension C' de l'exécution C, satisfait $C' \notin \mathcal{C}_1(P)$ et comme $\mathcal{C}_1(P) \subseteq \mathcal{C}_2(P)$ on obtient $\mathcal{C}_1(P) \neq \emptyset$ et $\mathcal{C}_2(P) = \emptyset$.

□ *Lemme* 96

Utilisant la Définition 94, et les Lemmes 95 et 96, nous prouvons le principal résultat de cette section.

Théoreme 97 *Il n'existe pas de codage uniforme du* $b\pi$*-calcul dans le* π*-calcul.*

Preuve

Supposons qu'il existe un codage uniforme $[\![\bullet]\!]$ du $b\pi$-calcul dans le π-calcul.

Notons $R_i \stackrel{def}{=} [\![P_i]\!]$, and $R(n) \stackrel{def}{=} R_1 \parallel R_2 \parallel \ldots \parallel R_n$, où P_i sont les mêmes que dans le Lemme 96. Nous avons

$$R(n) = R_1 \parallel R_2 \parallel \ldots \parallel R_n = [\![P_1]\!] \parallel [\![P_2]\!] \parallel \ldots \parallel [\![P_n]\!] =$$

$$[\![P_1 \parallel P_2 \parallel \ldots \parallel P_n]\!] = [\![P(n)]\!]$$

Comme $\forall n \geq 2, P(n)$ est un système électoral, par le Lemme 95, nous obtenons que $\forall n \geq 2, R(n)$ est un système électoral faible.

Soit $m, m' \geq 2$, et soit la substitution $\sigma : \mathbb{N} \longrightarrow \mathbb{N}, \sigma(i) = i + m'$, $\forall i \in \{1, \ldots, m\}$ et l'identité ailleurs.

Nous avons

$$R(m+m') = R_1 \parallel R_2 \parallel \ldots \parallel R_{m'} \parallel R_{m'+1} \parallel \ldots \parallel R_{m'+m} =$$

$$R_1 \parallel R_2 \parallel \ldots \parallel R_{m'} \parallel [\![P_{m'+1}]\!] \parallel \ldots \parallel [\![P_{m'+m}]\!] =$$

$$R_1 \parallel R_2 \parallel \ldots \parallel R_{m'} \parallel [\![\sigma(P_1)]\!] \parallel \ldots \parallel [\![\sigma(P_m)]\!] =$$

$$R_1 \parallel R_2 \parallel \ldots \parallel R_{m'} \parallel \sigma([\![P_1]\!]) \parallel \ldots \parallel \sigma([\![P_m]\!]) =$$

$$R_1 \parallel R_2 \parallel \ldots \parallel R_{m'} \parallel \sigma(R_1) \parallel \ldots \parallel \sigma(R_m) =$$

$$R(m') \parallel \sigma(R(m))$$

Comme $R(m')$ est un système électoral faible, il existe une exécution $C_1 : R(m') \stackrel{\tilde{\alpha}}{\Longrightarrow} R^p(m')$ et un $k \in \{1, \ldots, m'\}$, tel que C_1 contient une émission de la forme $\bar{k}$. De la même façon, comme $R(m)$ est un système électoral faible, alors il existe une exécution C_2 : $R(m) \stackrel{\tilde{\beta}}{\Longrightarrow} R^q(m)$ et un $k' \in \{1, \ldots, m\}$, tel que C_2 contient une émission de la forme $\bar{k'}$. Donc il existe une exécution $C_3 : \sigma(R(m)) \stackrel{\sigma(\tilde{\beta})}{\Longrightarrow} \sigma(R^q(m))$ de $\sigma(R(m))$ et un $k' \in \{1, \ldots, m\}$, tel que pour un $i \in \{m'+1, \ldots, m'+m\}$ C_3 contient une émission de la forme $\overline{\sigma(k')}$.

Donc nous avons l'exécution suivante

$$C_4 : R(m+m') = R(m') \parallel \sigma(R(m)) \stackrel{\tilde{\alpha}}{\Longrightarrow} R^p(m') \parallel \sigma(R(m))$$

$$\stackrel{\sigma(\tilde{\beta})}{\Longrightarrow} R^p(m') \parallel \sigma(R^q(m))$$

tel que pour un $i \in \{1, \ldots, m'\}$ la projection $Proj(C_4, i)$ contient une émission de la forme $\bar{k}$ où $k \in \{1, \ldots, m'\}$, et pour un $j \in \{m'+1, \ldots, m'+m\}$ la projection $Proj(C_4, j)$ contient une émission de la forme $\overline{\sigma(k')}$. Comme $\sigma(k') \in \{m'+1, \ldots, m'+m\}$, nous avons que $k \neq \sigma(k')$, et donc $C_4 \in \mathcal{C}_2(R(m+m'))$, c.a.d. $\mathcal{C}_2(R(m+m')) \neq \emptyset$, donc $R(m+m')$ n'est pas un système électoral faible, contradiction.

□ *Theoreme* 97

Le Théoreme 97 correspond au résultat de separation entre le π-calcul synchrone et le π-calcul asynchrone prouvé dans (Palamidessi, 1997). Palamidessi utilise pour la preuve de ce résultat le fait que la sémantique du π-calcul asynchrone possède une certaine confluence. Alors, il existe des systèmes électorals symmetriques forts dans le π-calcul synchrone, mais pas dans le π-calcul asynchrone; si dans un système électoral symmetrique fort P, un processus fait une premiere émission, tous les autres processus peuvent faire aussi leur émission correspondante pour passer toujours dans un système symmetrique, générant de cette facon une exécution infinie $C \notin \mathcal{C}_1(P)$.

Pour prouver le Théoreme 97, nous avons exploité une autre différence entre les deux calculs: tandis que le π-calcul possède une sémantique d'entrelacement (si $P \stackrel{\tilde{\alpha}}{\Longrightarrow} P'$ et $Q \stackrel{\tilde{\beta}}{\Longrightarrow} Q'$, alors $P \parallel Q \stackrel{\tilde{\alpha}}{\Longrightarrow} P' \parallel Q \stackrel{\tilde{\beta}}{\Longrightarrow} P' \parallel Q'$), cette dérivation n'est plus nécessairement vraie dans le $b\pi$ - calcul ($P \stackrel{\tilde{\alpha}}{\Longrightarrow} P'$ n'impliquent pas $P \parallel Q \stackrel{\tilde{\alpha}}{\Longrightarrow} P' \parallel Q$).

Si nous appelons $b\pi_a$-calcul la variante asynchrone du $b\pi$-calcul (obtenue à partir du $b\pi$-calcul en interdisant l'émission comme préfixe -la variante asynchrone de π-calcul étant obtenue similairement à partir du π-calcul), nous obtenons le résultat suivant:

Corolaire 98 *Il n'existe pas de codage uniforme de $b\pi_a$-calcul dans le π-calcul.*

Par des arguments similaires nous pouvons prouver la proposition suivante.

Proposition 99 *Il n'existe pas de codage uniforme de CBS avec passage de valeurs dans le π-calcul.*

Preuve

Soit $P_i = i! \;\&\; x?x!$. Par des arguments similaires au Lemme 96 nous pouvons prouver que pour tout $n \geq 2$, $P(n) = \prod_{i=1}^{n} P_i$ est un système électoral faible et par des arguments similaires au Théorème 97, nous obtenons le résultat.

□ *Proposition* 99

En π-calcul (comme dans le $b\pi$-calcul), l'opérateur $\parallel$ est associatif par rapport aux congruences induites par les bisimulations ou par les préordres induits par les tests (dans de nombreuses présentations de π - calcul, l'opérateur $\parallel$ est considéré "directement" associatif par une relation de congruence structurelle syntaxique). Donc il est normal de demander que tout encodage $[\![\bullet]\!]$ ne fasse pas de distinction entre $(P_1 \parallel P_2) \parallel P_3$ et $P_1 \parallel (P_2 \parallel P_3)$. Alors, en utilisant les mêmes preuves que pour le Théoreme 96 et pour la Proposition 99, nous pouvons prouver le résultat suivant:

Théoreme 100 *Il n'existe pas de codage $[\![\bullet]\!]$ du $b\pi$-calcul dans le π-calcul, ou de CBS avec passage de valeurs dans le π-calcul, qui satisfait les conditions suivantes :*

1. $[\![P \parallel Q]\!] = \nu\tilde{a}([\![P]\!] \parallel [\![Q]\!])$;
2. $[\![(P_1 \parallel P_2) \parallel P_3]\!] = [\![P_1 \parallel (P_2 \parallel P_3)]\!]$;
3. $[\![\sigma(P)]\!] = \sigma([\![P]\!])$, *pour toute substitution* σ;
4. $(\exists\tilde{\alpha} : P \stackrel{\tilde{\alpha}}{\Longrightarrow} P'$ *si et seulement si* $\exists\tilde{\beta} : [\![P]\!] \stackrel{\tilde{\beta}}{\Longrightarrow} [\![P']\!])$, *où pour tout* $a \in fn(P)$, $(\exists\alpha \in \tilde{\alpha}$ *tel que* $sub(\alpha) = a$ *et* $type(\alpha) = output$ *si et seulement si* $\exists\beta \in \tilde{\beta}$ *tel que* $sub(\beta) = a$ *et* $type(\beta) = output)$;
5. *si* $[\![P]\!] \stackrel{\tilde{\beta}}{\Longrightarrow} Q$ *alors* $\exists\tilde{\gamma} : [\![P]\!] \stackrel{\tilde{\beta}}{\Longrightarrow} Q \stackrel{\tilde{\gamma}}{\Longrightarrow} [\![P']\!]$;

où $\tilde{a}$ est une séquence de noms et $\tilde{\alpha}$, $\tilde{\beta}$ et $\tilde{\gamma}$ dénotent des séquences finies d'actions.

Par des techniques similaires à celles utilisées par Palamidessi dans (Palamidessi, 1997) pour prouver des résultats de séparation entre des calculs synchrones et asynchrones, nous prouvons qu'il n'existe pas de codage uniforme de CBS sans passage de valeurs dans CCS.

Lemme 101 *Soit $P = P_1 \parallel P_2 \parallel \ldots \parallel P_n$ un cluster dans CCS (qui ne contient pas de ν à l'extérieur), et soit un automorphisme σ sur l'hyper-graphe associé $H(P)$ tel que $\sigma(k) = k+1 \pmod{n}$. Supposons que l'hyper-graphe associé $H(P)$ est symétrique par rapport à σ. Alors P n'est pas un système électoral fort.*

Preuve

Supposons que P est un système électoral fort. Nous allons construire une séquence strictement croissante d'exécutions $C_0, C_1, \ldots, C_m, \ldots$, tel que pour tout j, C_{j+1} étend strictement C_j, et $C_j : P \stackrel{\tilde{\alpha}_j}{\Longrightarrow} P^j$ ne contient pas de émission de la forme $\bar{k}$ avec $k \in \{1, \ldots, n\}$ et P^j est toujours symétrique par rapport à σ. On va obtenir ainsi une contradiction, car l'exécution infinie qui est la limite de cette séquence ne pourra pas être étendue à une exécution de $\mathcal{C}_1(P)$.

La preuve est par induction sur j.

Soit C_0 l'exécution vide.

Étant donnée $C_j : P \stackrel{\tilde{\alpha}_j}{\Longrightarrow} P^j$, nous construisons $C_{j+1} : P \stackrel{\tilde{\alpha}_{j+1}}{\Longrightarrow} P^{j+1}$ comme il suit.

Comme P est un système électoral fort, il doit être possible d'étendre C_j à une exécution C qui contient au moins une émission de la forme $\bar{k}$ avec $k \in \{1, \ldots, n\}$. La première action ne peut pas être une émission de cette forme; sinon, soit P_i^j la composante qui fait cette action. Par symétrie, $P_{\sigma(i)}^j \equiv P_{i+1}^j$ peut faire comme action suivante l'émission $\sigma(\bar{k}) = k \bar{+} 1$ et on obtient une exécution de $\mathcal{C}_2(P)$, contradiction avec le fait que P est un système électoral fort.

Donc $\exists\gamma$ tel que $P^j \xrightarrow{\gamma}$. Dans ce pas, il y a une ou deux composantes qui ont participé.

– Si P_i^j est la seule composante qui participe à l'action, alors soit

$$P_i^j \xrightarrow{\gamma} P_i^{j+1}$$

le pas fait par P_i^j.

Par symétrie, nous avons

$$P^j_{\sigma(i)} \xrightarrow{\sigma(\gamma)} P^{j+1}_{\sigma(i)}$$

$$P^j_{\sigma^2(i)} \xrightarrow{\sigma^2(\gamma)} P^{j+1}_{\sigma^2(i)}$$

$$\vdots$$

$$P^j_{\sigma^{n_1}(i)} \xrightarrow{\sigma^{n-1}(\gamma)} P^{j+1}_{\sigma^{n-1}(i)}$$

En composant tous ces pas, on obtient l'exécution

$$P^j \overset{\tilde{\gamma}}{\Longrightarrow} P^{j+1}$$

où $\tilde{\gamma} = \gamma\sigma(\gamma)\ldots\sigma^{n-1}(\gamma)$ et $P^{j+1} = P_i^{j+1} \parallel P^{j+1}_{\sigma(i)} \parallel \ldots P^{j+1}_{\sigma^{n-1}(i)}$. Donc P^{j+1} est toujours symétrique.

– Si P_i^j et P_k^j sont les deux composantes qui participent à l'action, alors forcément $\gamma = \tau$ et

$$P_i^j \xrightarrow{\gamma_1} P_i^{j+1}$$

et

$$P_k^j \xrightarrow{\gamma_2} P_k^{j+1}$$

où γ_1 et γ_2 sont une émission et respectivement une réception qui se font sur le même canal. Ce canal ne peut pas être sur la portée d'une restriction (voir les conditions de l'énoncé et le fait que dans CCS les portées des noms restent statiques, et donc on n'aura jamais de restriction de nom au niveau le plus extérieur), et on obtient

$$P_i^j \xrightarrow{\gamma_1} P_i^{j+1}$$

par une action qui est sur un canal libre et différent de tout $k \in \{1, \ldots, n\}$. Donc on peut répéter le raisonnement du cas précédent.

□ *Lemme* 101

Proposition 102 *Il n'existe pas de codage uniforme de CBS sans passage de valeurs dans le CCS.*

Preuve

Supposons qu'il existe un codage uniforme $[\![\bullet]\!]$ de CBS sans passage de valeurs dans le CCS.

Soit $P_i = a_i! \parallel \sum_{j=1}^{n} a_j?j!$. Alors $P(n) = \prod_{i=1}^{n} P_i$ est un système électoral fort. De plus $P(n)$ est symétrique par rapport à $\sigma(k) = \sigma(k) + 1 \pmod{n}$. Comme un codage uniforme projette un système électoral fort de CBS dans un système électoral fort de CCS, et de plus il préserve tous les symétries (par la Condition 2. de la Définition 94), on obtient que $[\![P(n)]\!]$ est un système électoral fort qui satisfait toutes les conditions du Lemme 101, contradiction.

□ *Proposition* 102

6.2 Résultats des minimalités dans le $b\pi$-calcul

Parmi les constructeurs du $b\pi$ - calcul, tous peuvent être justifiés du point du vue pratique:

- l'opérateur préfixe $\alpha.p$ représente la sequentialité, et illustre la façon dont deux systèmes distribués interagissent: par échange de messages;
- l'opérateur $\parallel$ exprime le fait que deux programmes sont concurrents et s'exécutent en même temps;
- le ν est utile pour modéliser les processus qui peuvent naître dynamiquement à l'exécution, ou les appels multiples d'une même procédure;
- l'opérateur $+$ représente le choix, et donne la possibilité à un système de réagir différemment en fonction du canal sur lequel le premier événement arrive;
- l'opérateur conditionel $\langle x = y \rangle p,q$ donne la possibilité à un systeme de réagir d'une façon différente en fonction de l'information qu'il avait reçu auparavant;
- l'opérateur *rec* est essentiel pour pouvoir représenter des systemes infinis.

Par rapport à π-calcul, l'opérateur de choix et l'opérateur conditionnel sont différents : en π-calcul, le choix est limité au processus prefixés; le conditionnel est limité au "matching" : $\langle x = y \rangle p$ qui se comporte comme p si $x = y$ et comme nil autrement. Nous avons choisi de garder le choix général dans notre langage, pour son utilité dans les axiomatisations. En ce qui concerne le conditionnel, sa presence dans le π - calcul est justifié par le fait que le π - calcul posséde déjà un certain pouvoir de tester l'égalité des noms. Suivant le même raisonnement, le conditionnel dans le $b\pi$ - calcul est d'une part très util pour les axiomatisations (et la preuve de certains résultats), mais surtout, notre langage posséde déjà un certain pouvoir de tester l'égalité ou l'inégalité des noms. En effet, si $x = y$, alors $\nu u(\bar{x}.\bar{u} \parallel y.p + u.q) \xrightarrow{\bar{x}} \nu up \approx_c p$, et si $x \neq y$, alors $\nu u(\bar{x}.\bar{u} \parallel y.p + u.q) \xrightarrow{\bar{x}} \nu u(\bar{u} \parallel y.p + u.q) \xrightarrow{\tau} \nu uq \approx_c q$, où $u \notin fn(p,q)$.

Dans cette section on va montrer que ces deux opérateurs, tel qu'ils sont présents dans le $b\pi$-calcul, ne peuvent pas être exprimés avec les autres constructeurs du langage tout en préservant la congruence barbelée faible.

6.2.1 Non - encodage de $+$

Soit $\mathcal{P}_b^1$ le sous - ensemble de $\mathcal{P}_b$ défini dans le Table 6.1. $\mathcal{P}_b^1$ représente le sous - ensemble des processus tels que le choix est utilisé seulement pour les processus "prefix"qui se trouvent sous la forme $\alpha.p$.

$$p,p_1,p_2 ::= \quad nil \mid pre \mid \nu xp \mid \langle x=y\rangle p_1,p_2 \mid p_1 \parallel p_2 \mid A\langle\tilde{x}\rangle \mid (rec\ A\langle\tilde{x}\rangle.p)\langle\tilde{y}\rangle$$

$$pre,pre_1,pre_2 ::= \quad \alpha.p \mid pre_1 + pre_2$$

où $\alpha \in \{\tau,x(y),\bar{x}y\}$.

TAB. 6.1 – *L'ensemble de processus* $\mathcal{P}_b^1$

Soit $\mathcal{E}_1$ l'ensemble de termes $E(\bullet_1,\bullet_2)$ générés par la grammaire de la Table 6.2.

$$E ::= \quad \bullet_1 \mid \bullet_2 \mid nil \mid \Sigma_{i\in I}\alpha_i.E_i \mid \nu xE \mid \langle x=y\rangle E,E \mid$$

$$E \parallel E \mid A\langle\tilde{x}\rangle \mid (rec\ A\langle\tilde{x}\rangle.E)\langle\tilde{y}\rangle$$

où I est un ensemble fini d'index, et pour tout $i \in I$, $\alpha_i \in \{\tau,x(y),\bar{x}y\}$.

TAB. 6.2 – *L'ensemble de termes* $\mathcal{E}_1$

On va montrer qu'il n'existe pas un terme $E_+(\bullet_1,\bullet_2) \in \mathcal{E}_1$ tel que pour tous $p,q \in \mathcal{P}_b$, $p + q \overset{c}{\approx}_b E_+(p,q)$. La technique utilisée est inspirée de (Prasad, 1989). On va construire une relation $\Leftrightarrow_1$ qui n'est pas préservée par l'opérateur de choix, $+$ et tel que $\overset{c}{\approx}_b \subseteq \Leftrightarrow_1 \subseteq \approx_b$.

Soit $\mathcal{C}on_1$ la famille de contextes générés par la grammaire de la Table 6.3. Alors nous définissons $p \Leftrightarrow_1 q$ ssi $C[p] \approx_b C[q]$ pour tout contexte $C[\bullet] \in \mathcal{C}on_1$.

Remarque 103

$C ::=$	$\bullet \mid \alpha.C + \Sigma_{i\in I}\alpha_i.p_i \mid \nu x C \mid \langle x = y\rangle C,p \mid \langle x = y\rangle p,C \mid$
	$C \parallel p \mid p \parallel C \mid (rec\ A\langle\tilde{x}\rangle.C)\langle\tilde{y}\rangle$
où I est un ensemble fini d'index, et pour tout $i \in I$, $\alpha_i \in \{\tau, x(y), \bar{x}y\}$, et $p \in \mathcal{P}_b^1$.	

TAB. 6.3 – *Contextes $\mathcal{C}on_1$*

1. *L'inclusion $\overset{c}{\approx}_b \subseteq \Leftrightarrow_1$ suit directement de $\mathcal{C}on_1 \subseteq \mathcal{C}on$.*
2. *Pour tous processus $p,q,r \in \mathcal{P}_b^1$, et expression $E_+(\bullet_1,\bullet_2) \in \mathcal{E}_1$, $p \Leftrightarrow_1 q$ implique $E_+(p,r) \Leftrightarrow_1 E_+(q,r)$. La preuve est faite par induction sur le nombre d'apparitions de $\bullet_1$ dans $E_+(\bullet_1,\bullet_2)$, et utilise l'observation que pour une seule apparition de $\bullet_1$ dans $E_+(\bullet_1,\bullet_2)$, $E_+(\bullet,r) \in \mathcal{C}on_1$ pour tout processus $r \in \mathcal{P}_b^1$.*

Théoreme 104 *Il n'existe pas une expression $E_+(\bullet_1,\bullet_2) \in \mathcal{E}_1$ tel que pour tous $p,q \in \mathcal{P}_b^1$, $p + q \overset{c}{\approx}_b E_+(p,q)$.*

Preuve

Supposons par contradiction qu'il existe une expression $E_+(\bullet_1,\bullet_2) \in \mathcal{E}_1$ tel que pour tous $p,q \in \mathcal{P}_b^1$, $p + q \overset{c}{\approx}_b E_+(p,q)$.

Soit $p \overset{def}{=} \bar{a}$, $q \overset{def}{=} \tau.\bar{a}$, et $r \overset{def}{=} \bar{b}$. On peut prouver que $p \Leftrightarrow_1 q$. On obtient alors les équivalences suivantes (en tenant compte de la remarque 103):

$$p + r \overset{c}{\approx}_b E_+(p,r) \implies p + r \Leftrightarrow_1 E_+(p,r)$$

$$q + r \overset{c}{\approx}_b E_+(q,r) \implies q + r \Leftrightarrow_1 E_+(q,r)$$

$$p \Leftrightarrow_1 q \implies E_+(p,r) \Leftrightarrow_1 E_+(q,r)$$

Par la transitivité de $\Leftrightarrow_1$ on obtient $p + r \Leftrightarrow_1 q + r$ et donc $p + r \approx_b q + r$. Mais $q + r \xrightarrow{\tau} \bar{a}$ transition que $p + r$ ne peut pas simuler.

$\square$ *Theoreme* 104

6.2.2 Non - encodage de $\langle x = y\rangle$

Soit $\mathcal{P}_b^2$ le sous - ensemble de $\mathcal{P}_b$ défini dans le Table 6.4. $\mathcal{P}_b^2$ représente le sous - ensemble des processus tels que l'opérateur de test est utilisé seulement dans sa forme "if then" comme pour le π -calcul.

Soit $\mathcal{E}_2$ l'ensemble de termes $E(\bullet_1,\bullet_2)$ générés par la grammaire de la Table 6.5.

$p,p_1,p_2 ::=$	$nil \mid \alpha.p \mid p_1 + p_2 \mid \nu xp \mid \langle x = y\rangle p \mid p_1 \parallel p_2 \mid$
	$A\langle\tilde{x}\rangle \mid (rec\ A\langle\tilde{x}\rangle.p)\langle\tilde{y}\rangle$
où $i \in \{1,2\}$, $\alpha \in \{\tau,x(y),\bar{x}y\}$.	

TAB. 6.4 – *L'ensemble de processus $\mathcal{P}_b^2$*

$E ::=$	$\bullet_1 \mid \bullet_2 \mid nil \mid \alpha.E \mid \nu xE \mid \langle x = y\rangle E \mid E + E \mid E \parallel E \mid$
	$A\langle\tilde{x}\rangle \mid (rec\ A\langle\tilde{x}\rangle.E)\langle\tilde{y}\rangle$
où $\alpha \in \{\tau,x(y),\bar{x}y\}$.	

TAB. 6.5 – *L'ensemble de termes $\mathcal{E}_2$*

Soit $x,y \in Ch_b, x \neq y$ deux canaux. On va montrer qu'il n'existe pas un terme $E_{if}(\bullet_1,\bullet_2) \in \mathcal{E}_1$ tel que pour tous $p,q \in \mathcal{P}_b^2$, $\langle x = y\rangle p,q \overset{c}{\approx}_b E_{if}(p,q)$.

Par une induction sur la longueur de la dérivation de $p \xrightarrow{\alpha} p'$ (comme dans le Lemme 14) on peut prouver le résultat suivant:

Lemme 105 *Soit $p \in \mathcal{P}_b^2$ et $p \xrightarrow{\alpha} p'$. Si σ est une substitution tel que $bn(\alpha) \cap (prdom(\sigma) \cup prcod(\sigma)) = \emptyset$, alors $p\sigma \xrightarrow{\alpha\sigma} p''$ pour un p'' tel que $p'' \equiv_\alpha p'\sigma$.*

Une conséquence (qui suit par induction sur le nombre de pas dans $p \overset{\alpha}{\Longrightarrow} p'$) est:

Corollaire 106 *Soit $p \in \mathcal{P}_b^2$, $p \overset{\alpha}{\Longrightarrow} p'$. Si σ est une substitution tel que $bn(\alpha) \cap (prdom(\sigma) \cup prcod(\sigma)) = \emptyset$, alors $p\sigma \overset{\alpha\sigma}{\Longrightarrow} p''$ pour un p'' tel que $p'' \equiv_\alpha p'\sigma$.*

Théoreme 107 *Il n'existe pas une expression $E_{if}(\bullet_1,\bullet_2) \in \mathcal{E}_2$ tel que pour tous $p,q \in \mathcal{P}_b^2$, $\langle x = y\rangle p,q \overset{c}{\approx}_b E_{if}(p,q)$.*

Preuve

Supposons par contradiction qu'il existe une expression $E_{if}(\bullet_1,\bullet_2) \in \mathcal{E}_2$ tel que pour tous $p,q \in \mathcal{P}_b^2$, $\langle x = y\rangle p,q \overset{c}{\approx}_b E_{if}(p,q)$. Il est facile à prouver que $E_{if}(p,q) \in \mathcal{P}_b^2$.

Soit $p \overset{def}{=} \bar{a}$, et $q \overset{def}{=} \bar{b}$ où $\{a,b\} \cap \{x,y\} = \emptyset$. Soit $\sigma_1 = [y/x]$. Comme $\langle x = y\rangle p,q \overset{c}{\approx}_b E_{if}(p,q)$ on obtient $E_{if}(p,q) \Downarrow_b$ et $E_{if}(p,q) \not\Downarrow_a$, et par le Lemme 106 on a

$$(E_{if}(p,q))\sigma_1 \Downarrow_b \tag{6.4}$$

De plus, $\langle x=y\rangle p,q \overset{c}{\approx}_b E_{if}(p,q)$ implique $(\langle x=y\rangle p,q)\sigma_1 \overset{c}{\approx}_b (E_{if}(p,q))\sigma_1$, donc $(E_{if}(p,q))\sigma_1 \Downarrow_a$ et $(E_{if}(p,q))\sigma_1 \not\Downarrow_b$, contradiction avec 6.4.

$\square$ *Theoreme* 107

6.3 Conclusion

Dans ce chapitre, dans un premier temps, nous nous sommes intéressés aux relations entre les communications point à point et les primitives de diffusion.

Cette question a déjà été posée dans (Holmer, 1993). Holmer présente un encodage de CBS (sans passage de valeurs) dans $SCCS$ et il conjecture qu'il n'est pas possible de donner une traduction compositionnelle de CBS dans CCS: "CCS is $<<$ *too asynchronous* $>>$ to interpret CBS in".

Nous donnons une réponse partielle à cette conjecture. Utilisant l'existence (ou la non-existence) des solutions pour l'élection d'un leader dans un système distribué, nous montrons qu'il n'est pas possible d'implanter (sous certaines conditions) des langages à diffusion (comme CBS ou $b\pi$ -calcul) dans des langages basés sur des communications point à point (comme CCS ou π-calcul). Si on enlève la condition qui éxige que le codage préserve la distribution, il est possible (Prasad, 2001) de donner une translation de CBS vers CCS qui préserve la sémantique operationnelle.

Ensuite, nous analysons les opérateurs de choix et de test tels qu'ils sont dans le $b\pi$ - calcul (étant différents des opérateurs homonymes du π - calcul où il existe seulement le choix "prefixé" et le test simple "if then"). Utilisant une relation préservée par tous les autres opérateurs du langage à l'exception de l'opérateur de choix, nous montrons l'impossibilité d'implanter le choix général à l'aide des autres constructions du $b\pi$-calcul plus le choix "prefixé". Par une technique similaire, nous montrons l'impossibilité d'implanter le test "if then else" à l'aide des autres constructions du $b\pi$-calcul plus le test "if then ".

Chapitre 7

Conclusion

Dans cette thèse, nous nous sommes intéressés à la modélisation et à la spécification des systèmes mobiles communiquant par diffusion.

Avant d'introduire notre modèle, nous avons présenté une hiérarchie (non - exhaustive) des principaux formalismes pour la concurrence, en insistant surtout sur les modèles algébriques. Nous avons apporté des arguments pour la nécessité d'un modèle simple, fondé sur une théorie bien établie, qui permettrait de modéliser des applications qui utilisent des primitives de groupe fournies par la plupart des bibliothèques de communication. Dans de nombreux exemples les groupes sont des structures dynamiques: un processus peut joindre ou quitter dynamiquement un groupe (une fois que l'application a connaissance de l'existence du groupe, ou d'une adresse, port ou capacité qui sert d'alias du groupe).

Dans le Chapitre 3 nous présentons la syntaxe et la sémantique opérationnelle du $b\pi$-calcul. Ensuite, nous montrons le pouvoir d'expression de notre modèle à travers plusieurs exemples. Ces exemples montrent que pour certains algorithmes, notre langage permet une description telle que le comportement de chaque participant est indépendant du nombre des autres participants dans le système.

Dans le Chapitre 4, nous introduisons plusieurs relations d'équivalences qui permettent d'identifier (ou de distinguer) deux processus en fonction de leurs réponses aux interactions avec l'environnement.

Par des techniques classiques, nous montrons que les équivalences ainsi obtenues, coïncident avec les bisimulations étiquetées pour une large classe de processus (les processus d'image finie).

Ensuite, nous considérons comme application la spécification d'un protocole de FIFO broadcast dans un réseau asynchrone de processus. Par l'exhibition d'une relation de bisimulation appropriée on prouve la correction du protocole.

Nous terminons le chapitre par un étude des congruences induites par les relations de bisimulations, et par une axiomatisation complète des congruences fortes pour les processus finis.

Dans le Chapitre 5, nous montrons que les bisimulations (largement utilisées pour les systèmes distribués interactifs), sont trop restrictives dans certains cas pour les systèmes à diffusion. Alors nous adaptons les relations induites par des tests: le "must testing" et le "may testing" (de Nicola et Hennessy, 1984) pour les modèles à diffusion.

Ensuite nous présentons des caractérisations basées sur des traces pour le "must testing" et le "may testing", pour une variante sans passage de valeur de CBS et pour le $b\pi$-calcul.

Dans le Chapitre 6, nous présentons l'inexistence de certains encodages entre les langages à diffusion (comme CBS ou $b\pi$ -calcul) et les langages basés sur des communications point à point (comme CCS ou π-calcul). Ces résultats apportent un argument supplémentaire pour l'introduction du $b\pi$ -calcul.

Plusieurs continuations de ce travail peuvent être envisagées.

En ce qui concerne les (pre -) congruences induites par le "must testing" et le "may testing" , il reste à développer une axiomatisation qui soit complète pour les processus finis.

Les relations entre les langages à diffusion et les langages basés sur des communications point à point restent encore peu étudiées ((Jensen, 1994), (Prasad, 2001)). Trouver des encodages qui préservent des bonnes propriétés entre le $b\pi$ -calcul et le π-calcul reste une direction intéressante de travail.

Un aspect intéressant dans $b\pi$ - calcul est l'éxtension dynamique de la portée d'un nom dans un calcul à diffusion. Une continuation de cet étude sera de trouver des typages pérmetant de limiter une "trop large" diffusion d'un nom privé. Un possible point de départ pourrait être le typage présenté dans (Cardelli et al., 2000) pour un π-calcul avec des groupes.

Nous souhaiterions appliquer les techniques développées pour le $b\pi$ -calcul à d'autres protocoles où la mobilité et la diffusion jouent un rôle important. Pour la preuve de certains protocoles de broadcast, (Hadzilacos et Toueg, 1994) il sera probablement nécessaire de prendre en compte la notion de temps, ou de rendre l'asynchronisme plus explicite dans notre modèle. Une manière de faire cela (Berger et Honda, 2000), est d'introduire un opérateur qui marque le passage du temps. Ensuite, il faudra adapter les résultats développés dans cette thèse pour le nouveau langage.

Bibliographie

AGHA, G. (1986). *Actors: A Model of Concurrent Computation*. MIT Press.

AGHA, G., MASON, I., SMITH, S., et TALCOTT, C. (1993). « A foundation for actor computation Journal of Functional Programming ».

AMADIO, R. M. (2000). « On modelling mobility ». *Theoretical Computer Science*, 240(1):147–176.

AMADIO, R. M., CASTELLANI, I., et SANGIORGI, D. (1998). « On Bisimulations for the Asynchronous π-Calculus ». *Theoretical Computer Science*, 195(2):291–324. An extended abstract appeared in *Proceedings of CONCUR '96*, LNCS 1119: 147–162.

AMIR, Y., DOLEV, D., et KRAMER, S. (1992). « Transis: A Communication Sub-System for High Availability ». Dans *22d Int. Symp. on Fault-tolerant Computing*.

ARIANE (1996). « Inquiry Board Traces Ariane 5 Failure to Overflow Error ». *SIAM News, Society for Industrial and Applied Mathematics*, 29(8).

ATTIE, P. et LYNCH, N. (2001). « A formal model for dynamic computation ». Rapport Technique, MIT Lab. for Computer Science. To appear.

BANACH, R. et van BREUGEL, F. (1998). « Mobility and Modularity: Expressing Pi-Calculus in CCS. ». Rapport de recherche.

BAYERDORFFER, B. (1993). « *Associative Broadcast and the Communication Semantics of Naming in Concurrent Systems* ». PhD thesis, University of Texas at Austin.

BAYERDORFFER, B. (1994). « Distributed Programming with Associative Broadcast ». Dans *Proceedings of the 27th Annual Hawaii International Conference on System Sciences. Volume 2: Software Technology (HICSS94-2), Wailea, HW, USA*, pages 353–362.

BERGER, M. et HONDA, K. (2000). « The Two-Phase Commitment Protocol in an Extended pi-Calculus ». pages 105–130.

BERRY, G., RAMESH, S., et SHYAMASUNDAR, R. K. (1993). « Communicating Reactive Processes ». Dans *Proc. 20th ACM Conf. on Principles of Programming Languages (POPL), Charleston, Virginia*.

BEST, E., DEVILLER, R., et KOUTNY, M. (2001). *Petri Net Algebra*. Springer EATCS Monographs on Theoretical Computer Science.

BEST, E. et FERNÁNDEZ, C. (1988). *Nonsequential Processes: a Petri Net View*. Springer EATCS Monographs on Theoretical Computer Science.

BIRMAN, K. P., COOPER, R., JOSPEH, T. A., KANE, K. P., SCHMUCK, F., et WOOD, M. (1990). « ISIS-a distributed programming environment ». Rapport Technique, Ithaca, NY.

BOLOGNESI, T. et BRINKSMA, E. (1987). « Introduction to the ISO Specication Language LOTOS ». *Computer Networks and ISDN Systems*, 14(1):25–59.

BOUDOL, G. (1992). « Asynchrony and the π-calculus (Note) ». Rapport de Recherche 1702, INRIA Sophia-Antipolis.

BOUGE, L. (1988). « On the Existence of Symmetric Algorithms to Find Leaders in Networks of Communicating Sequential Processes. ». *Acta Informatica*, 25(2):179–201.

BUSI, N., GORRIERI, R., et ZAVATTARO:, G. (1998). « A Process Algebraic View of Linda Coordination Primitives ». *Theoretical Computer Science*, 192(2):167–199.

CARDELLI, L., GHELLI, G., et GORDON, A. D. (2000). « Secrecy and Group Creation ». Dans PALAMIDESSI, C., éditeur, *CONCUR 2000: Concurrency Theory (11th International Conference, University Park, PA, USA)*, volume 1877 de *LNCS*, pages 365–379. Springer.

CARRIERO, N. et GELERNTER, D. (1989). « Linda in context ». *Communications of the ACM*, 32(4):444–458.

CASTELLANI, I. et HENNESSY, M. (1998). « Testing Theories for Asynchronous Processes ». Dans *Lecture Notes in Computer Science*, volume 1530, pages 90–108. Springer Verlag.

CIANCARINI, P., GORRIERI, R., et ZAVATTARO, G. (1995). « Generative Communication in Process Algebra ». Rapport Technique UBLCS-95-16, University of Bologna.

DALMONTE, A., GASPARI, M., et GORRIERI, R. (1995). « An algebra of actors ». Rapport Technique, Universit'a di Bologna, Italy.

de NICOLA, R. et HENNESSY, M. (1984). « Testing Equivalences for Processes ». *Theoretical Computer Science*, 34:83–133.

de NICOLA, R. et SEGALA, R. (1995). « A process algebraic view of input/output automata ». *Theoretical Computer Science*, 138:391–424.

ENE, C. et MUNTEAN, T. (1999). « Expressiveness of Point-to-Point versus Broadcast Communications ». Dans *Fundamentals of Computation Theory, 12th International Symposium, Lecture Notes in Computer Science*, volume 1684. Springer Verlag.

ENE, C. et MUNTEAN, T. (2001). « A Broadcast-based Calculus for Communicating Systems ». Dans *In 6th International Workshop on Formal Methods for Parallel Programming: Theory and Applications, San Francisco*.

FOURNET, C. et GONTHIER, G. (1996). « The reflexive CHAM, and the join calculus ». Dans *23rd Annual Symposium on Principles of Programming Languages (POPL) (St. Petersburg Beach, Florida)*, pages 372–385. ACM.

GASPARI, M. et ZAVATTARO, G. (1999). « An Algebra of Actors ». Dans *Proc. 3nd IFIP Conf. on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, pages 3–18. Kluwer Academic Publishers.

GEIST, A., BEGUELIN, A., DONGARRA, J., MANCHEK, R., JIANG, W., et SUNDERAM, V. (1994). *PVM: A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press.

HADZILACOS, V. et TOUEG, S. (1994). « A modular approach to fault-tolerant broadcasts and related problems ». Rapport Technique TR94-1425, Cornell University, Department of Computer Science, Ithaca, New York, USA.

HAYDEN, M. (1998). « The Ensemble System ». Rapport Technique TR98-1662, Cornell University.

HENNESSY, M. (1991). « A model for the π-calculus ». Rapport Technique 8/91, School of Cognitive and Computer Science, University of Sussex.

HENNESSY, M. et MILNER, R. (1985). « Algebraic Laws for Nondeterminism and Concurrency ». *Journal of the Association for Computing Machinery*, 32(1):137–161.

HENNESSY, M. et RATHKE, J. (1995). « Bisimulations for a Calculus of Broadcasting Systems ». Dans *CONCUR 95, Lecture Notes in Computer Science*, volume 962. Springer Verlag.

HENNESSY, M. et RIELY, J. (1998). « Resources acess control in systems of mobile agents ». Rapport Technique 2/98, School of Cognitive and Computer Science, University of Sussex.

HEWITT, C. (1977). « Viewing control structures as patterns of message passing ». *Journal of Artificial Inteligence*, 8(3):323–364.

HOARE, C. (1978). « Communicating Sequential Processes ». *Communications of ACM*, 21(8):666–677.

HOARE, C. (1985). *Communicating Sequential Processes*. Prentice-Hall.

HOLMER, U. (1993). « Interpreting Broadcast Communication in SCCS ». Dans *Proceedings of CONCUR '93*, volume 715 de *Lecture Notes in Computer Science*. Springer-Verlag.

HONDA, K. et TOKORO, M. (1991). « An Object Calculus for Asynchronous Communication ». Dans AMERICA, P., éditeur, *European Conference on Object-Oriented Programming (ECOOP'91)*, volume 512 de *LNCS*, pages 133–147. Springer.

JENSEN, C. T. (1994). « Interpreting broadcast communications in CCS with priority choice ». Dans *In Proceedings of 6th Nordic Workshop on Programming Theory, Aarhus University*. BRICS Report Series.

LYNCH, N. et TUTTLE, M. (1989). « An Introduction to Input/Output automata ». Rapport Technique, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands. Also, Technical Memo MIT/LCS/TM-373, Laboratory for Computer Science, Massachusetts Institute ofTechnology.

M.BOREALE et NICOLA, R. D. (1995). « Testing Equivalence for Mobile Systems ». *Information and Computation*, 120:279–303.

M.BOREALE, NICOLA, R. D., et PUGLIESE, R. (1999). « Trace and Testing Equivalence on Asynchronous Processes ». Dans *Lecture Notes in Computer Science*, volume 1578, pages 165–179. Springer Verlag.

MILLINGTON, M. (1987). « Theories of translation correctness for concurrent programming languages ». Rapport Technique CST-46-87, University of Edinburgh, Department of Computer Science. also published as ECS-LFCS-87-39: Thesis.

MILNER, R. (1980). « A Calculus of Communicating Systems. ». Dans *Lecture Notes in Computer Science. Springer-Verlag*.

MILNER, R. (1983). « Calculi for Synchrony and Asynchrony ». *Theoretical Computer Science*, 25:267–310.

MILNER, R. (1989). *Communication and concurrency*. Prentice-Hall.

MILNER, R. (1993). « The Polyadic π-Calculus: A Tutorial ». Dans BAUER, F. L., BRAUER, W., et SCHWICHTENBERG, H., éditeurs, *Logic and Algebra of Specification, Proceedings of International NATO Summer School (Marktoberdorf, Germany, 1991)*, volume 94 de *Series F*. NATO ASI, Springer. Available as Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991.

MILNER, R. (1999). *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press.

MILNER, R., PARROW, J., et WALKER, D. (1992). « A Calculus of Mobile Processes, Part I/II ». *Journal of Information and Computation*, 100:1–77.

MILNER, R. et SANGIORGI, D. (1992). « Barbed Bisimulation ». Dans *Proc. of 19-the International Colloquium on Automata, Languages and Programming (ICALP '92), Lecture Notes in Computer Science*, volume 623. Springer Verlag.

MOSER, L. E., MELLIAR-SMITH, P. M., AGARWAL, D. A., BUDHIA, R. K., LINGLEY-PAPADOPOULOS, C. A., et ARCHAMBAULT, T. P. (1995). « The Totem System ». Dans *In Proceedings of the 25th International Symposium on Fault Tolerant Computing, Pasedena, CA*.

NESTMANN, U. (1996). « *On Determinacy and Nondeterminacy in Concurrent Programming* ». PhD thesis, Technische Fakultät, Universität Erlangen. Arbeitsbericht IMMD-29(14).

NIEHREN, J. (1996). « Functional Computation as Concurrent Computation ». Dans

23rd Annual Symposium on Principles of Programming Languages (POPL) (St. Petersburg Beach, Florida), pages 333–343. ACM.

ODERSKY, M. (1995a). « Applying π: Towards a Basis for Concurrent Imperative Programming ». Dans REDDY, U. S., éditeur, *Second ACM SIGPLAN Workshop on State in Programming Languages*, pages 95–108. ACM/SIGPLAN, University of Illinois at Urbana-Champaign.

ODERSKY, M. (1995b). « Polarized Name Passing ». Dans THIAGARAJAN, P. S., éditeur, *15th Conference on Foundations of Software Technology and Theoretical Computer Science (Bangalore, India, December 18–20, 1995)*, volume 1026 de *LNCS*, pages 324–337. Springer.

ORAVA, F. et PARROW, J. (1992). « An Algebraic Verification of a Mobile Network ». *Journal of Formal Aspects of Computing*, 4:497–543.

OSTROVSKÝ, K., PRASAD, K. V. S., et TAHA, W. (2001). « Towards a primitive higher order calculus of broadcasting systems ». Rapport Technique, Dept. of Computing Science, Chalmers University of Tech.

PALAMIDESSI, C. (1997). « Comparing the Expressive Power of the Synchronous and the Asynchronous π-calculus ». Dans *24th Annual Symposium on Principles of Programming Languages (POPL) (Paris, France)*, pages 256–265. ACM.

PARK, D. (1981). « Concurrency on Automata and infinite sequences ». Dans *Conf. on Theoretical Computer Science, (P. Deussen ed.)*, volume 104 de *LNCS*. Springer Verlag,.

PARROW, J. et SANGIORGI, D. (1995). « Algebraic Theories for Name-Passing Calculi ». *Information and Computation*, 120(2):174–197.

PETRI, C. A. (1973). « Concepts of Net Theory. ». Dans *Mathematical Foundations of Computer Science: Proc. of Symposium and Summer School, High Tatras, Sep. 3–8, 1973*, pages 137–146. Math. Inst. of the Slovak Acad. of Sciences.

PIERCE, B. C. (1996). Foundational Calculi for Programming Languages. Dans TUCKER, A. B., éditeur, *Handbook of Computer Science and Engineering*, Chapitre 139. CRC Press.

PIERCE, B. C. et TURNER, D. N. (1995). « Concurrent Objects in a Process Calculus ». Dans ITO, T. et YONEZAWA, A., éditeurs, *Theory and Practice of Parallel Programming (TPPP, Sendai, Japan, 1994)*, volume 907 de *LNCS*, pages 187–215. Springer.

PRASAD, K. V. S. (1987). « *Combinators and Bisimulation Proofs for Restartable Systems* ». PhD thesis, University of Edinburgh.

PRASAD, K. V. S. (1989). « On the Non-Derivability of Operators in CCS ». Rapport Technique 55, Dept. of Computing Science, Chalmers University of Tech.

PRASAD, K. V. S. (1991). « A Calculus of Broadcasting Systems ». Dans *In TAPSOFT'91, Volume 1: CAAP, Lecture Notes in Computer Science*, volume 493. Springer Verlag.

PRASAD, K. V. S. (1993). « A Calculus of Value Broadcasts ». Dans *In PARLE'93, Lecture Notes in Computer Science*, volume 694. Springer Verlag.

PRASAD, K. V. S. (1995). « A Calculus of Broadcasting Systems ». *Science of Computer Programming*, 25.

PRASAD, K. V. S. (2001). « Broadcast Calculus Interpreted in CCS upto Bisimulation ». Dans *8th International Workshop on Expressiveness in Concurrency*.

QUAGLIA, P. (1996). « *The π-Calculus with Explicit Substitutions* ». PhD thesis, Dipartimento di Informatica, Università di Pisa-Genova-Udine. Available as PhD number TD-09/96.

REISIG, W. (1985). *Petri Nets, An Introduction*. Springer Verlag, Berlin.

RENESSE, R. V., BIRMAN, K. P., GLADE, B. B., GUO, K., HAYDEN, M., HICKEY, T., MALKI, D., VAYSBURD, A., et VOGELS, W. (1995). « Horus:A Flexible Group Communications System ». Rapport Technique Technical Report TR95-1500, Cornell University.

ROCKL, C. et SANGIORGI, D. (1999). « A pi-calculus Semantics of Concurrent Idealised ALGOL ». Dans *In Proceedings of Fossacs'99, Lecture Notes in Computer Science*, volume 1578. Springer Verlag.

ROSCOE, A. (1998). *The Theory and Practice of Concurrency*. Prentice Hall.

SANGIORGI, D. (1992). « *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigm* ». PhD thesis, University of Edinburgh.

SANGIORGI, D. (1998). « Interpreting Functions as pi-Calculus Processes: a Tutorial ». Rapport de Recherche RR-3470, INRIA Sophia-Antipolis.

TANENBAUM, A. S., van RENESSE, R., van STAVEREN, H., SHARP, G. J., MULLENDER, S. J., JANSEN, J., et van ROSSUM, G. (1990). « Experiences with the Amoeba distributed operating system ». *Communications of the ACM*, 33(12):46–63.

TIPLEA, F. L. et ENE, C. (1997). « Hierarchies of Petri Net Languages and a Super-Normal Form ». *Journal of Automata, Languages and Combinatorics*, 2(3).

TURNER, D. N. (1996). « *The Polymorphic Pi-Calculus: Theory and Implementation* ». PhD thesis, LFCS, University of Edinburgh. CST-126-96 (also published as ECS-LFCS-96-345).

VAANDRAGER, F. (1991). « On the Relationship between Process Algebra and Input/Output Automata ». Dans *LICS: IEEE Symposium on Logic in Computer Science*.

VICTOR, B. (1998). « *Expressiveness and Symmetry in Mobile Processes* ». PhD thesis, Department of Computer Systems, Uppsala University, Sweden. Available as report DoCS 98/98.

WALKER, D. (1995). « Objects in the π-calculus ». *Journal of Information and Computation*, 116(2):253–271.

WEGNER, P. (1997). « Why Interaction Is More Powerful Than Algorithms ». *Communications of the ACM*, 40(5):80–91.

Annexe A

Preuves rélatives au Chapitre 3

Lemme 8

1. Si $p \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} p'$, alors $fn(p') \subseteq fn(p) \cup \{\tilde{y}\}$ et $(\tilde{x} \setminus \tilde{y}) \subseteq fn(p)$.
2. Si $p \xrightarrow{a\langle\tilde{z}\rangle} p'$, alors $fn(p') \subseteq fn(p) \cup \{\tilde{z}\}$.
3. Si $p \xrightarrow{\tau} p'$, alors $fn(p') \subseteq fn(p)$.

Preuve

La preuve est faite par induction sur la dérivation de la transition $p \xrightarrow{\alpha} p'$, simultanément pour les trois assertions.

c1) $p \xrightarrow{\alpha} p'$ est obtenu par la règle (1).
Alors $p = \tau.p'$ et $fn(p') = fn(\tau.p') = fn(p)$

c2) $p \xrightarrow{\alpha} p'$ est obtenu par la règle (2).
Donc $p = a(\tilde{x}).p''$, $p' = p''[\tilde{z}/\tilde{x}]$ et $\alpha = a\langle\tilde{z}\rangle$.
Alors $fn(p) = \{a\} \cup fn(p'') \setminus \{\tilde{x}\}$, $fn(p') \subseteq fn(p'') \setminus \{\tilde{x}\} \cup \{\tilde{z}\} \subseteq (fn(p'') \setminus \{\tilde{x}\}) \cup \{a,\tilde{z}\} = fn(p) \cup \{\tilde{z}\}$.

c3) $p \xrightarrow{\alpha} p'$ est obtenu par la règle (3).
Alors $p = \bar{a}\tilde{x}.p'$ et $\alpha = \bar{a}\tilde{x}$.
Donc $\tilde{x} \subseteq fn(\bar{a}\tilde{x}.p') = fn(p)$ et $(fn(p') \subseteq fn(p') \cup \{a,\tilde{x}\} = fn(\bar{a}\tilde{x}.p') = fn(p) \subseteq fn(p) \cup \{\}$.

c4) $p \xrightarrow{\alpha} p'$ est obtenu par la règle (4).
Alors $\exists q,q'$ tel que $p = \nu z q$, $p' = q'[w/z]$, $\alpha = \nu w \nu\tilde{y}\bar{a}(\tilde{x}[w/z])$, $z \in \tilde{x} \setminus \{a,\tilde{y}\}$, $w \notin fn(\nu z q')$ et $q \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} q'$ par une dérivation plus courte.
Utilisant l'hypothèse d'induction, on obtient $\tilde{x} \setminus \tilde{y} \subseteq fn(q)$. Comme $fn(p) = fn(q) \setminus \{z\}$ on obtient $\tilde{x} \setminus \{z,\tilde{y}\} \subseteq fn(p)$, d'où $\tilde{x}[w/z] \setminus \{\tilde{y},w\} \subseteq fn(p)$.
Utilisant de nouveau l'hypothèse d'induction on obtient $fn(q') \subseteq fn(q) \cup \{\tilde{y}\}$.

On a $fn(p') = (fn(q') \setminus \{z\}) \cup \{w\} \subseteq (fn(q) \cup \{\tilde{y}\} \setminus \{z\}) \cup \{w\}$.
Comme $z \in \tilde{x} \setminus \{a,\tilde{y}\}$, on a $(fn(q) \cup \{\tilde{y}\} \setminus \{z\}) \cup \{w\} = fn(q) \setminus \{z\} \cup \{\tilde{y}\} \cup \{w\} = fn(p) \cup \{\tilde{y}\} \cup \{w\}$ et donc $fn(p') \subseteq fn(p) \cup \{w,\tilde{y}\}$.

c5) $p \xrightarrow{\alpha} p'$ est obtenu par la règle (5).
Alors $\alpha = \tau$, $p = \nu ar$, $p' = \nu a \nu \tilde{y} r'$ et $r \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} r'$.
Comme $r \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} r'$ est obtenu par une déduction plus courte, utilisant l'hypothèse d'induction on obtient $fn(r') \subseteq fn(r) \cup \tilde{y}$. En utilisant que $fn(p) = fn(r) \setminus \{a\}$, $fn(p') = fn(r') \setminus \{a,\tilde{y}\}$, nous obtenons $fn(p') \subseteq (fn(r) \cup \tilde{y}) \setminus \{a,\tilde{y}\} = fn(r) \setminus \{a\} = fn(p)$.

c6) $p \xrightarrow{\alpha} p'$ est obtenu par la règle (6).
Alors $\exists q$ tel que $p = \nu xq$, $p' = \nu xq'$, $q \xrightarrow{\alpha} q'$ et $x \notin n(\alpha)$.
Donc $fn(p) = fn(q) \setminus \{x\}$, $fn(p') = fn(q') \setminus \{x\}$, et ensuite il suffit d'utiliser l'hypothèse d'induction pour la transition $q \xrightarrow{\alpha} q'$.

c7) $p \xrightarrow{\alpha} p'$ est obtenu par la règle (7).
Alors $p = p_1 + p_2$, et (obtenu par une dérivation plus courte) $p_1 \xrightarrow{\alpha} p'$.
Comme $fn(p_1) \subseteq fn(p)$, il suffit ensuite d'utiliser l'hypothèse d'induction pour la transition $p_1 \xrightarrow{\alpha} p'$.
Si la règle appliquée est la symetrique de la règle (7), la preuve est similaire.

c8) $p \xrightarrow{\alpha} p'$ est obtenu par une des règles (8), (9) ou (10). Similaire au cas $c7$).

c9) $p \xrightarrow{\alpha} p'$ est obtenu par la règle (11).
Alors $p = p_1 \parallel p_2$, $p' = p'_1 \parallel p'_2$, $\alpha = a\langle\tilde{x}\rangle$, et (obtenus par une dérivation plus courte) $p_1 \xrightarrow{a\langle\tilde{x}\rangle} p'_1$ et $p_2 \xrightarrow{\bar{a}\langle\tilde{x}\rangle} p'_2$.
Par hypothèse d'induction, $fn(p'_i) \subseteq fn(p_i) \cup \{\tilde{x}\}$ et on obtient $fn(p') = fn(p'_1) \cup fn(p'_2) \subseteq (fn(p_1) \cup \{\tilde{x}\}) \cup (fn(p_2) \cup \{\tilde{x}\})$ d'où $fn(p') \subseteq fn(p) \cup \{\tilde{x}\}$.

c10) $p \xrightarrow{\alpha} p'$ est obtenu par la règle (12).
Alors $p = p_1 \parallel p_2$, $p' = p'_1 \parallel p'_2$, $\alpha = \nu\tilde{y}\bar{a}\tilde{x}$, $\tilde{y} \cap fn(p_2) = \emptyset$, et (obtenus par une dérivation plus courte) $p_1 \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} p'_1$ et $p_2 \xrightarrow{a\langle\tilde{x}\rangle} p'_2$.
Par hypothèse d'induction, $(\tilde{x} \setminus \tilde{y}) \subseteq fn(p_1)$, $fn(p'_1) \subseteq fn(p_1) \cup \{\tilde{y}\}$, $fn(p'_2) \subseteq fn(p_2) \cup \{\tilde{x}\}$ et on obtient $fn(p') = fn(p'_1) \cup fn(p'_2) \subseteq (fn(p_1) \cup \{\tilde{y}\}) \cup (fn(p_2) \cup \{\tilde{x}\})$.
On a $(\tilde{x} \setminus \tilde{y}) \subseteq fn(p_1) \subseteq fn(p)$, et on obtient $fn(p') \subseteq fn(p) \cup \{\tilde{y}\}$.
Si la règle appliquée est la symetrique de la règle (12), la preuve est similaire.

c11) $p \xrightarrow{\alpha} p'$ est obtenu par la règle (13) . Similaire aux cas $c10$).

$\square$ *Lemme* 8

Lemme 13

Soit $p \xrightarrow{\alpha} q$ une transition de p, et $\beta \equiv_\alpha \alpha$ une action telle que $bn(\beta) \cap n(p) = \emptyset$. Alors $p \xrightarrow{\beta} q[bn(\beta)/bn(\alpha)]$ par une dérivation de longueur plus petite ou egale à celle de $p \xrightarrow{\alpha} q$.

Preuve

Par induction sur l'inférence de $p \xrightarrow{\alpha} q$.

c1) $p \xrightarrow{\alpha} q$ est obtenu par la règle (1).
Alors $p = \tau.q$ et $\alpha = \tau$.
Evidemment $\beta = \alpha = \tau$ et $p \xrightarrow{\beta} q = q[bn(\beta)/bn(\alpha)]$.

c2) $p \xrightarrow{\alpha} q$ est obtenu par la règle (2).
Donc $p = a(\tilde{x}).r$, $q = r[\tilde{z}/\tilde{x}]$ et $\alpha = a\langle\tilde{z}\rangle$.
Evidemment $\beta = \alpha = a\langle\tilde{z}\rangle$ et $p \xrightarrow{\beta} q = q[bn(\beta)/bn(\alpha)]$.

c3) $p \xrightarrow{\alpha} q$ est obtenu par la règle (3).
Alors $p = \bar{a}\tilde{x}.q$ et $\alpha = \bar{a}\tilde{x}$.
Evidemment $\beta = \alpha = \bar{a}\tilde{x}$ et $p \xrightarrow{\beta} q = q[bn(\beta)/bn(\alpha)]$.

c4) $p \xrightarrow{\alpha} q$ est obtenu par la règle (4).
Alors $\exists r,s$ tel que $p = \nu zr$, $q = s[w/z]$, $w \notin fn(\nu zs)$, $z \in \tilde{x} \setminus \{a,\tilde{y}\}$, $\alpha = \nu w\nu\tilde{y}\bar{a}(\tilde{x}[w/z])$, et $r \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} s$ par une dérivation plus courte.
Alors $\beta = \nu w_1\nu\tilde{y_1}\bar{a}(\tilde{x}[(w_1,\tilde{y_1})/(z,\tilde{y})])$, $\{w_1,\tilde{y_1}\} \cap n(p) = \emptyset$ et $\{z,w_1\} \cap \{\tilde{y_1},\tilde{y}\} = \emptyset$ $\quad (*)$.
Par l'hypothèse d'induction, on a $r \xrightarrow{\nu\tilde{y_1}\bar{a}(\tilde{x}[\tilde{y_1}/\tilde{y}])} s[\tilde{y_1}/\tilde{y}]$ (par une déduction plus petite ou egale à celle de $r \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} s$).
Comme $p = \nu zr$ et utilisant les conditions $(*)$ et la règle (4) on obtient $p \xrightarrow{\nu w_1\nu\tilde{y_1}\bar{a}(\tilde{x}[(w_1,\tilde{y_1})/(z,\tilde{y})])} s[(w_1,\tilde{y_1})/(z,\tilde{y})] = q[(w_1,\tilde{y_1})/(w,\tilde{y})] = q[bn(\beta)/bn(\alpha)]$.

c5) $p \xrightarrow{\alpha} q$ est obtenu par la règle (5).
Evidemment $\beta = \alpha = \tau$ et $p \xrightarrow{\beta} q = q[bn(\beta)/bn(\alpha)]$.

c6) $p \xrightarrow{\alpha} q$ est obtenu par la règle (6).
Alors $\exists r,s$ tels que $p = \nu xr$, $q = \nu xs$, $r \xrightarrow{\alpha} s$ et $x \notin n(\alpha)$.
Comme $x \in n(p)$ et $bn(\beta) \cap n(p) = \emptyset$ on obtient $x \notin bn(\beta)$. Comme $x \notin n(\alpha)$ et $fn(\alpha) = fn(\beta)$ on a aussi $x \notin fn(\beta)$ et donc $x \notin n(\beta)$.
Par l'hypothèse d'induction on obtient $r \xrightarrow{\beta} s[bn(\beta)/bn(\alpha)]$ (par une déduction plus petite ou egale à celle de $r \xrightarrow{\alpha} s$), et comme $x \notin n(\beta)$, par la la règle (6) on obtient la transition $p = \nu xr \xrightarrow{\beta} \nu x(s[bn(\beta)/bn(\alpha)]) = (\nu xs)[bn(\beta)/bn(\alpha)]$.

c7) $p \xrightarrow{\alpha} q$ est obtenu par la règle (7).
Alors $p = p_1 + p_2$, et (obtenu par une dérivation plus courte) $p_1 \xrightarrow{\alpha} q$.
Par l'hypothèse d'induction pour la transition on a $p_1 \xrightarrow{\beta} q[bn(\beta)/bn(\alpha)]$, et par la règle (7) on obtient $p \xrightarrow{\beta} q[bn(\beta)/bn(\alpha)]$.

c8) $p \xrightarrow{\alpha} q$ est obtenu par une des règles (8), (9) ou (10), (11), (12) ou (13). Similaire au cas $c7$).

□ *Lemme* 13

Lemme 14

Soit $p \xrightarrow{\alpha} q$ une transition de p, et σ une substitution injective telle que $(n(\sigma) \cup n(p\sigma)) \cap bn(\alpha) = \emptyset$ et $prcod(\sigma \setminus fn(p)) \cap fn(p) = \emptyset$. Alors $p\sigma \xrightarrow{\alpha\sigma}\equiv_\alpha q\sigma$ par une dérivation de longueur plus petite ou egale à celle de $p \xrightarrow{\alpha} q$.

Preuve

Par induction sur l'inférence de $p \xrightarrow{\alpha} q$.

c1) $p \xrightarrow{\alpha} q$ est obtenu par la règle (1).
Alors $p = \tau.q$ et $\alpha = \tau$.
On a $p\sigma = \tau.q\sigma$, d'où par la règle (1) on obtient $p\sigma \xrightarrow{\alpha\sigma} q\sigma$.

c2) $p \xrightarrow{\alpha} q$ est obtenu par la règle (2).
Donc $p = a(\tilde{x}).r$, $q = r[\tilde{z}/\tilde{x}]$ et $\alpha = a\langle\tilde{z}\rangle$.
Pour un $\tilde{x_1} \cap n(\sigma) = \emptyset$, $p\sigma = \sigma(a)(\tilde{x_1}).((r[\tilde{x_1}/\tilde{x}])\sigma)$, $q\sigma \equiv_\alpha (r[\tilde{z}/\tilde{x}])\sigma$, $\alpha\sigma = \sigma(a)\langle\sigma(\tilde{z})\rangle$.
On obtient $p\sigma \xrightarrow{\alpha\sigma} ((r[\tilde{x_1}/\tilde{x}])\sigma)[\sigma(\tilde{z})/\tilde{x_1}]$.
Comme $\tilde{x_1} \cap n(\sigma) = \emptyset$, on obtient $((r[\tilde{x_1}/\tilde{x}])\sigma)[\sigma(\tilde{z})/\tilde{x_1}] = (r[\tilde{z}/\tilde{x}])\sigma \equiv_\alpha q\sigma$ et donc $p\sigma \xrightarrow{\alpha\sigma}\equiv_\alpha q\sigma$.

c3) $p \xrightarrow{\alpha} q$ est obtenu par la règle (3).
Alors $p = \bar{a}\tilde{x}.q$ et $\alpha = \bar{a}\tilde{x}$.
On a $p\sigma = \overline{\sigma(a)}\sigma(\tilde{x}).(q\sigma)$ et par la règle (3) on obtient $p\sigma \xrightarrow{\alpha\sigma} q\sigma$.

c4) $p \xrightarrow{\alpha} q$ est obtenu par la règle (4).
Alors $\exists r,s$ tel que $p = \nu z r$, $q = s[w/z]$, $w \notin fn(\nu z s)$, $z \in \tilde{x} \setminus \{a,\tilde{y}\}$, $\alpha = \nu w \nu\tilde{y}\bar{a}(\tilde{x}[w/z])$, et $r \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} s$ par une dérivation plus courte.
De plus $\{w,\tilde{y}\} \cap n(\sigma) = \emptyset$ et $prcod(\sigma \setminus fn(p)) \cap fn(p) = \emptyset$.
$p\sigma = (\nu z r)\sigma = \nu z_1(r[z_1/z]\sigma)$ pour un $z_1 \notin n(\sigma)$.
Comme $\{w,z_1\} \cap n(\sigma) = \emptyset$ on a $(\sigma(\tilde{x}[z_1/z]))[w/z_1] = (\sigma(\tilde{x}[w/z]))$ et $(s[z_1/z]\sigma)[w/z_1] = s[w/z]\sigma$.
De plus $w \notin fn(\nu z s)$ et $w \notin n(\sigma)$ impliquent $w \notin fn(\sigma(fn(\nu z s)))$ et donc $w \notin fn(\nu z_1(s[z_1/z]\sigma))$
Comme $r \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} s$ par une dérivation plus courte, on obtient par l'hypothèse d'induction, $r[z_1/z]\sigma \xrightarrow{\nu\tilde{y}\overline{\sigma(a)}(\sigma(\tilde{x}[z_1/z]))}\equiv_\alpha s[z_1/z]\sigma$ (par une dérivation plus courte ou egale).
Par la règle (4), on obtient $p\sigma \xrightarrow{\nu w \nu\tilde{y}\overline{\sigma(a)}(\sigma(\tilde{x}[w/z]))}\equiv_\alpha s[w/z]\sigma = q\sigma$.

c5) $p \xrightarrow{\alpha} q$ est obtenu par la règle (5).
Alors $\exists r,s$ tels que $\alpha = \tau$, $p = \nu a r$, $q = \nu a \nu\tilde{y}s$ et $r \xrightarrow{\nu\tilde{y}\bar{a}\tilde{x}} s$.
Alors pour $a_1,\tilde{y_1}$ tels que $\{a_1,\tilde{y_1}\} \cap (n(\sigma) \cup n(p) \cup fn(q))) = \{a,a_1\} \cap \{\tilde{y},\tilde{y_1}\} = \emptyset$, on a

$$p\sigma = \nu a_1(r[a_1/a]\sigma), q\sigma \equiv_\alpha \nu a_1 \nu\tilde{y_1}(s[(a_1,\tilde{y_1})/(a,\tilde{y})]\sigma). \tag{A.1}$$

Comme $r \xrightarrow{\nu\tilde{y}\tilde{a}\tilde{x}} s$ est obtenu par une déduction plus courte, utilisant l'hypothèse d'induction on obtient $r[a_1/a] \xrightarrow{\nu\tilde{y}\tilde{a_1}\tilde{x}} \equiv_\alpha s[a_1/a]$ (par une dérivation plus courte ou egale). Par le Lemme 13, on obtient $(r[a_1/a]) \xrightarrow{\nu\tilde{y_1}\tilde{a_1}(\tilde{x}[\tilde{y_1}/\tilde{y}])} \equiv_\alpha (s[a_1/a])[\tilde{y_1}/\tilde{y}]$ (par une dérivation plus courte ou egale).

Par l'hypothèse d'induction on obtient

$$(r[a_1/a])\sigma \xrightarrow{(\nu\tilde{y_1}\tilde{a_1}(\tilde{x}[\tilde{y_1}/\tilde{y}]))\sigma} \equiv_\alpha (s[a_1/a])[\tilde{y_1}/\tilde{y}]\sigma \tag{A.2}$$

(par une dérivation plus courte ou egale).

Dès assertions A.1 et A.2 on obtient par la règle (5) $p\sigma \xrightarrow{\tau} \equiv_\alpha q\sigma$.

c6) $p \xrightarrow{\alpha} q$ est obtenu par la règle (6).

Alors $\exists r,s$ tels que $p = \nu xr$, $q = \nu xs$, $r \xrightarrow{\alpha} s$ et $x \notin n(\alpha)$.

Alors pour $x_1 \notin n(\sigma) \cup n(p,\alpha)$, $p\sigma = \nu x_1((r[x_1/x])\sigma)$, $q\sigma \equiv \nu x_1((s[x_1/x])\sigma)$.

$x_1 \notin n(\alpha)$ implique $\alpha[x_1/x] = \alpha$ et $x_1 \notin n(\alpha[x_1/x])$. Comme $x_1 \in n(\sigma)$ on obtient $x_1 \notin n(\alpha\sigma)$.

Par l'hypothèse d'induction on obtient la transition $r[x_1/x]\sigma \xrightarrow{\alpha\sigma} \equiv_\alpha s[x_1/x]\sigma$ (par une dérivation plus courte ou egale), et par la règle (6), $p\sigma \xrightarrow{\alpha\sigma} \equiv_\alpha q\sigma$.

c7) $p \xrightarrow{\alpha} q$ est obtenu par la règle (7).

Alors $p = p_1 + p_2$, et (obtenu par une dérivation plus courte) $p_1 \xrightarrow{\alpha} q$.

Alors $p\sigma = p_1\sigma + p_2\sigma$, et par l'hypothèse d'induction pour la transition on a $p_1\sigma \xrightarrow{\alpha\sigma} \equiv_\alpha q\sigma$.

Il suffit ensuite d'appliquer la règle (7).

c8) $p \xrightarrow{\alpha} q$ est obtenu par une des règles (8), (9) ou (10), (11), (12) ou (13). Similaire au cas *c*8), en tenant compte pour les règles (8) et (9) du fait que σ est injective.

$\square$ *Lemme* 14

Lemme 15

Soit σ une substitution injective telle que $prcod(\sigma \setminus fn(p)) \cap fn(p) = \emptyset$.

Si $p\sigma \xrightarrow{\alpha} q'$ et $(n(\sigma) \cup fn(p)) \cap bn(\alpha) = \emptyset$, alors ils existent une action β, un processus q et une substitution ρ extension injective de σ, tels que $\alpha = \beta\rho$, $q' \equiv_\alpha q\rho$, $\rho \setminus fn(p) = \sigma \setminus fn(p)$, $(n(\rho) \cup fn(p)) \cap bn(\beta) = \emptyset$ et $p \xrightarrow{\beta} q$ par une dérivation de longueur plus petite ou egale à celle de $p\sigma \xrightarrow{\alpha} q'$.

Preuve

Par induction sur l'inférence de $p\sigma \xrightarrow{\alpha} q'$.

c1) $p\sigma \xrightarrow{\alpha} q'$ est obtenu par la règle (1).

Alors $p\sigma = \tau.q'$ et donc il existe q tel que $p = \tau.q$ et $q' = q\sigma$. Il suffit ensuite de prendre $\rho = \sigma$.

c2) $p\sigma \xrightarrow{\alpha} q'$ est obtenu par la règle (2).
Donc $p\sigma = b(\tilde{x}).r'$, $q' = r'[\tilde{z}/\tilde{x}]$ et $\alpha = b\langle\tilde{z}\rangle$.
Alors $\tilde{x} \cap n(\sigma) = \emptyset$ et pour certains $\tilde{x_1}$, a et q_1 on a $p = a(\tilde{x_1}).q_1$, $\sigma(a) = b$ et $r' \equiv_\alpha (q_1[\tilde{x}/\tilde{x_1}])\sigma$.
C'est ici que nous construisons l'extension de σ. Il se peut qu'il existe des noms dans $\tilde{z}$ qui n'appartiennent pas a $prcod(\sigma)$. Soit $\tilde{z'} = \tilde{z} \setminus prcod(\sigma)$ et $\tilde{z''} = prcod(\sigma) \cap \tilde{z}$. Soit $\tilde{u}$ un tuple de noms "fraiches" de même arité que $\tilde{z'}$ (tel que $\tilde{u} \cap (n(p) \cup n(\sigma) \cup n(\alpha)) = \emptyset$) et soit $\rho = \sigma \cup [\tilde{z'}/\tilde{u}]$, et $\beta = a\langle\tilde{v}\rangle$ où dans $\tilde{v}$, chaque v_i est soit u_{i_j} (si $z_i \in \tilde{z'}$ et $z_i = z'_{i_j}$), soit $\sigma^{-1}(z''_{i_k})$ (si $z_i \in \tilde{z''}$ et $z_i = z''_{i_k}$). Alors ρ est injective ($\tilde{z'} \cap prcod(\sigma) = \emptyset$) et par construction $\alpha = \beta\sigma$. Soit $q = q_1[\tilde{v}/\tilde{x_1}]$. Alors $p \xrightarrow{\beta} q$, $q' \equiv_\alpha q\rho$, et $\sigma \setminus fn(P) = \rho \setminus fn(P)$.

c3) $p \xrightarrow{\alpha} q$ est obtenu par une des règles (3) à (13). Similaire et plus simple (en tenant compte pour les règles (8) et (9) du fait que σ est injective).

$\square$ *Lemme* 15

Annexe B

Preuves rélatives aux bisimulations

Lemme 34 $\sim$ satisfait les propriétés:

(a) $p \equiv_\alpha q$ implique $p \sim q$

(b) $p \parallel nil \sim p$

(c) $p \parallel q \sim q \parallel p$

(d) $(p \parallel q) \parallel r \sim p \parallel (q \parallel r)$

(e) $p + nil \sim p$

(f) $p + q \sim q + p$

(g) $(p+q)+r \sim p+(q+r)$

(h) $\nu x p \sim p$ si $x \notin fn(p)$

(i) $\nu y \nu x p \sim \nu x \nu y p$ si $x \neq y$

(j) $(\nu x p) \parallel q \sim \nu x(p \parallel q)$ si $x \notin fn(q)$

(k) $(\nu x p) + q \sim \nu x(p+q)$ si $x \notin fn(q)$

(l) $\langle y = z\rangle(\nu x p),q \sim \nu x(\langle y = z\rangle p,q)$ si $x \notin fn(q) \cup \{y,z\}$.

Preuve

a) $p \equiv_\alpha q$ implique $p \sim q$
La relation S^1 où
$\mathcal{S}^1 \stackrel{def}{=} \{(p,q) \mid p,q \in \mathcal{P}_b,\ p \equiv_\alpha q\}$
est une bisimulation forte. Le résultat suit de l′ observation que $p \equiv_\alpha q$ et $p \stackrel{\gamma}{\longrightarrow} p'$ implique $q \stackrel{\gamma}{\longrightarrow} q'$ pour un certain q' tel que $p' \equiv_\alpha q'$ (par induction sur la dérivation de $p \stackrel{\gamma}{\longrightarrow} p'$, utilisant les Lemmes 13 et 14).

b) $p \parallel nil \sim p$
La relation
$\mathcal{S}^2 \stackrel{def}{=} \{(p \parallel nil,p) \mid p, \in \mathcal{P}_b\}$
est une bisimulation forte.

Soit $(p,q) \in \mathcal{S}^2$. Alors $p = q \parallel nil$.

Si $p \xrightarrow{\alpha} p'$, alors la derniere regle utilisée dans l' inférence de cette transition est la règle (13). Donc $p' = q' \parallel nil$ et $q \xrightarrow{\alpha} q'$. Alors $(q' \parallel nil, q') \in \mathcal{S}^2$.

Si $q \xrightarrow{\alpha} q'$, par l' application de la règle (13) on obtient $p \xrightarrow{\alpha} q' \parallel nil$ avec $(q' \parallel nil, q') \in \mathcal{S}^2$.

c) $p \parallel q \sim q \parallel p$

La relation

$\mathcal{S}^3 \overset{def}{=} \{(p \parallel q, q \parallel p) \mid p,q \in \mathcal{P}_b\}$

est une bisimulation forte.

Si $p \parallel q \xrightarrow{\alpha} r$, alors la derniere regle utilisée dans l' inférence de cette transition est l' une des règles (11), (12) ou (13) et $r = p' \parallel q'$. Par l' application de la règle (11), ou l' une des symetriques des règles (12) ou (13) on obtient $q \parallel p \xrightarrow{\alpha} q' \parallel p'$.

d) $(p \parallel q) \parallel r \sim p \parallel (q \parallel r)$

La relation

$\mathcal{S}^4 \overset{def}{=} \{(p \parallel q) \parallel r, p \parallel (q \parallel r) \mid p,q,r \in \mathcal{P}_b\}$

est une bisimulation forte. La preuve est similaire au cas précedent.

e) $p + nil \sim p$

La relation

$\mathcal{S}^5 \overset{def}{=} \{(p + nil, p) \mid p \in \mathcal{P}_b\} \bigcup \{(p,p) \mid p \in \mathcal{P}_b\}$

est une bisimulation forte.

Soit $(p,q) \in \mathcal{S}^5$. Alors $p = q + nil$ ou $p = q$.

Le cas $p = q$ est evident.

Soit $p = q + nil$.

Si $q + nil \xrightarrow{\alpha} r$, alors la derniere regle utilisée dans l'inférence de cette transition est la règle (7), et $r = q'$, où $q \xrightarrow{\alpha} q'$. Evidement $(q', q') \in \mathcal{S}^5$.

Si $q \xrightarrow{\alpha} r$, alors par l'application de la symetrique de la règle (7), on obtient $q + nil \xrightarrow{\alpha} r$ et $(r,r) \in \mathcal{S}^5$.

f) $p + q \sim q + p$

g) $(p + q) + r \sim p + (q + r)$.

Pour les cas f) et g), la preuve que $\mathcal{S}^6$ et $\mathcal{S}^7$ sont des bisimulations fortes est similaire au cas e).

$\mathcal{S}^6 \overset{def}{=} \{(p + q, q + p) \mid p,q \in \mathcal{P}_b\} \bigcup \{(p,p) \mid p \in \mathcal{P}_b\}$

$\mathcal{S}^7 \overset{def}{=} \{((p + q) + r, p + (q + r)) \mid p,q,r \in \mathcal{P}_b\} \bigcup \{(p,p) \mid p \in \mathcal{P}_b\}$

(h) $\nu x p \sim p$ si $x \notin fn(p)$

La relation

$\mathcal{S}^8 \stackrel{def}{=} \{(\nu x p, p) \mid p \in \mathcal{P}_b, x \notin fn(p)\}$
est une bisimulation forte.
Soit $(p,q) \in \mathcal{S}^8$. Alors $p = \nu x q$ avec $x \notin fn(q)$.
Si $\nu x q \xrightarrow{\alpha} r$, alors la derniere regle utilisée dans l' inférence de cette transition est la règle (6), et $r = \nu x q'$, où $q \xrightarrow{\alpha} q'$ et $x \notin fn(q')$. Evidement $(\nu x q', q') \in \mathcal{S}^8$.
Si $q \xrightarrow{\alpha} r$, alors par l'application de la règle (6), on obtient $\nu x q \xrightarrow{\alpha} \nu x r$ et $(\nu x r, r) \in \mathcal{S}^8$ (sachant que $x \notin fn(r)$).

(i) $\nu y \nu x p \sim \nu x \nu y p$ si $x \neq y$
La relation
$\mathcal{S}^9 \stackrel{def}{=} \{(\nu x_1 \ldots \nu x_n p, \nu x_{\sigma(1)} \ldots \nu x_{\sigma(n)} p) \mid n \in \mathbb{N}, \sigma$ permutation de $\{1, \ldots, n\}$, pour tous $i,j,\ x_i \neq x_j, p \in \mathcal{P}_b\}\}$
est une bisimulation forte.
Soit $(\nu x_1 \ldots \nu x_n p, \nu x_{\sigma(1)} \ldots \nu x_{\sigma(n)} p) \in \mathcal{S}^9$ et soit $\nu x_1 \ldots \nu x_n p \xrightarrow{\alpha} q'$ une transition.
Par induction sur n, on peut prouver que cette transition est dérivée d' une transition de p, $p \xrightarrow{\beta} p'$. On va faire une analyse par cas en fonction de β.

- $\beta = \tau$.
  Alors par n applications de la règle (6), on obtient $\nu x_1 \ldots \nu x_n p \xrightarrow{\tau} \nu x_1 \ldots \nu x_n p'$ et $\nu x_{\sigma(1)} \ldots \nu x_{\sigma(n)} p \xrightarrow{\tau} \nu x_{\sigma(1)} \ldots \nu x_{\sigma(n)} p'$.
- $\beta = a\langle \tilde{b} \rangle$.
  Alors $a \notin \{x_1, \ldots, x_n\}$, et par n applications de la règle (6), on obtient $\nu x_1 \ldots \nu x_n p \xrightarrow{a\langle \tilde{b} \rangle} \nu x_1 \ldots \nu x_n p'$ et $\nu x_{\sigma(1)} \ldots \nu x_{\sigma(n)} p \xrightarrow{a\langle \tilde{b} \rangle} \nu x_{\sigma(1)} \ldots \nu x_{\sigma(n)} p'$.
- $\beta = \nu \tilde{b} \bar{a} \tilde{c}$, et $\tilde{c} \setminus \tilde{b} \cap \{x_1, \ldots, x_n\} = \{x_{i_1}, \ldots, x_{i_m}\}$ On a de nouveau deux cas.
  - $a \notin \{x_1, \ldots, x_n\}$
    Alors par $n-m$ applications de la règle (6) et par m applications de la règle (4), on obtient $\nu x_1 \ldots \nu x_n p \xrightarrow{\nu x_{i_1} \ldots \nu x_{i_m} \nu \tilde{b} \bar{a} \tilde{c}} \nu y_1 \ldots \nu y_{n-m} p'$ et $\nu x_{\sigma(1)} \ldots \nu x_{\sigma(n)} p \xrightarrow{\nu x_{i_1} \ldots \nu x_{i_m} \nu \tilde{b} \bar{a} \tilde{c}} \nu y_{\rho(1)} \ldots \nu y_{\rho(n-m)} p'$ pour une certaine permutation ρ de $\{1, \ldots, n-m\}$ et $\{y_1 \ldots y_{n-m}\} = \{y_{\rho(1)} \ldots y_{\rho(n-m)}\} = \{x_1, \ldots, x_n\} \setminus \{x_{i_1}, \ldots, x_{i_m}\}$.
  - $a \in \{x_1, \ldots, x_n\}$
    Alors par plusieurs applications des règles (4) et (6) et une application de la règle (5), on obtient $\nu x_1 \ldots \nu x_n p \xrightarrow{\tau} \nu y_1 \ldots \nu y_{n+k} p'$ et $\nu x_{\sigma(1)} \ldots \nu x_{\sigma(n)} p \xrightarrow{\tau} \nu y_{\rho(1)} \ldots \nu y_{\rho(n+k)} p'$ pour une certaine permutation ρ de $\{1, \ldots, n+k\}$, $|\tilde{b}| = k$ et $\{y_1 \ldots y_{n+k}\} = \{y_{\rho(1)} \ldots y_{\rho(n+k)}\} = \{x_1, \ldots, x_n\} \cup \tilde{b}$.

(j) $(\nu xp) \parallel q \sim \nu x(p \parallel q)$ si $x \notin fn(q)$

La relation

$\mathcal{S}^{10} \stackrel{def}{=} \{((\nu xp) \parallel q, \nu x(p \parallel q)) \mid p \in \mathcal{P}_b, x \notin fn(q)\} \bigcup \{(p,p) \mid p \in \mathcal{P}_b\}$ *

est une bisimulation forte.

Soit $((\nu xp) \parallel q, \nu x(p \parallel q)) \in \mathcal{S}^{10}$, avec $x \notin fn(q)$.

Si $(\nu xp) \parallel q \xrightarrow{\alpha} r$, alors la derniere regle utilisée dans l' inférence de cette transition est l' une des règles (11), (12), (13) ou une des symtriques des règles (12) ou (13).

Si $\alpha = \nu\tilde{b}\bar{a}\tilde{c}$ et $x \in \tilde{b}$, alors pour certains p' et q' on a $(\nu xp) \parallel q \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} p' \parallel q'$ et $\nu x(p \parallel q) \xrightarrow{\nu\tilde{b}\bar{a}\tilde{c}} p' \parallel q'$ par l' application de la règle (4).

Dans tous les autres cas, on obtient $(\nu xp) \parallel q \xrightarrow{\alpha} (\nu yp') \parallel q'$ et $\nu x(p \parallel q) \xrightarrow{\alpha} \nu y(p' \parallel q')$ pour certains y, p' et q'.

(k) $(\nu xp) + q \sim \nu x(p + q)$ si $x \notin fn(q)$

(l) $\langle y = z\rangle(\nu xp), q \sim_b \nu x(\langle y = z\rangle p, q)$ si $x \notin fn(q) \cup \{y,z\}$

Pour les cas *k*) et *l*), la preuve que $\mathcal{S}^{11}$ et $\mathcal{S}^{12}$ sont des bisimulations fortes est similaire au cas *e*).

$\mathcal{S}^{11} \stackrel{def}{=} \{((\nu xp) + q, \nu x(p + q)) \mid p \in \mathcal{P}_b, x \notin fn(q)\} \bigcup \{(p,p) \mid p \in \mathcal{P}_b\}$

$\mathcal{S}^{12} \stackrel{def}{=} \{(\langle y = z\rangle(\nu xp), q, \nu x(\langle y = z\rangle p, q)) \mid p, q \in \mathcal{P}_b, x \notin fn(q) \cup \{y,z\}\} \bigcup \{(p,p) \mid p \in \mathcal{P}_b\}$

□ *Lemme* 34

Annexe C

La correction d'un "FIFO protocole"

Nous utilisons l'abreviation
$\langle x = a_1, a_2\rangle p_1, p_2, p_3 \stackrel{def}{=} \langle x = a_1\rangle\ p_1, (\langle x = a_2\rangle p_2, p_3)$.
Nous rappelons les processus utilisé pour la description du protocole.
$Server\langle req, net, s_0\rangle \stackrel{def}{=}$
$\quad req(replay).(Server\langle req, net, s_0\rangle \parallel \nu out\ \nu in\ \nu id\ \overline{replay}[out, in, id].Fifo\langle out, in, id, net, s_0\rangle)$
$Fifo\langle out, in, id, net, s_0\rangle \stackrel{def}{=} Fifo_{out}\langle out, id, net, s_0\rangle \parallel \nu loc\,(Fifo_{in}\langle in, net, loc\rangle \parallel ASend\langle loc, s_0\rangle)$
$Fifo_{out}\langle out, id, net, s_m\rangle \stackrel{def}{=} out(m').\nu s_{m'}\ (Fifo_{out}\langle out, id, net, s_{m'}\rangle \parallel \overline{net}[id, m', s_{m'}, s_m])$
$Fifo_{in}\langle in, net, l\rangle \stackrel{def}{=} net(id, m', s_{m'}, s_m).(Fifo_{in}\langle in, net, l\rangle \parallel Cell\langle id, m', s_{m'}, s_m, in, l\rangle)$
$Cell\langle id, m', s_{m'}, s_m, in, l\rangle \stackrel{def}{=} l(y).\langle y = s_m\rangle Data\langle id, s_{m'}, in, l\rangle, Cell\langle id, m', s_{m'}, s_m, in, l\rangle$
$Data\langle id, m', s_{m'}, in, l\rangle \stackrel{def}{=} \overline{in}[id, m'].ASend\langle l, s_{m'}\rangle$
$ASend\langle l, s\rangle \stackrel{def}{=} \bar{l}s.ASend\langle l, s\rangle$
$Cl_0\langle out_0, id_0, m_1, m_2\rangle \stackrel{def}{=} \overline{out_0}m_1.\overline{out_0}m_2.P \parallel Q \qquad m_2 \notin fn(Q),\quad net \notin fn(P, Q)$

$$\begin{aligned} Cl_1\langle in_1, id_1, id_0, m_1, m_2, ok, err\rangle \stackrel{def}{=}\ & in_1(i, m).\langle i = id_0\rangle \\ & (\langle m = m_2, m_1\rangle \overline{err}, \overline{ok}, Cl_1\langle in_1, id_1, id_0, m_1, m_2, ok, err\rangle), \\ & Cl_1\langle in_1, id_1, id_0, m_1, m_2, ok, err\rangle \end{aligned}$$

$$\begin{aligned} Cl_2\langle in_2, id_2, id_0, m_1, m_2, ok, err\rangle \stackrel{def}{=}\ & in_2(i, m).\langle i = id_0\rangle \\ & (\langle m = m_1\rangle \overline{ok}, Cl_2\langle in_2, id_2, id_0, m_1, m_2, ok, err\rangle), \\ & Cl_2\langle in_2, id_2, id_0, m_1, m_2, ok, err\rangle. \end{aligned}$$

Nous devons prouver $A_1 \approx A_2$ où

$$\begin{aligned} A_i \stackrel{def}{=} \nu net\ \nu s_0\{ & Server\langle req, net, s_0\rangle \parallel \\ & \big(\nu out_0\ \nu in_0\ (Cl_0\langle out_0, id_0, m_1, m_2\rangle \parallel Fifo\langle out_0, in_0, id_0, net, s_0\rangle) \\ & \parallel\ \ \nu out_i\ \nu in_i\ (Cl_i\langle in_i, id_i, id_0, m_1, m_2, ok, err\rangle \parallel Fifo\langle out_i, in_i, id_i, net, s_0\rangle)\big)\}. \end{aligned}$$

Soit

$$\begin{aligned} Test_1\langle in_1, id_1, id_0, m_1, m_2, ok, err, i, m\rangle \stackrel{def}{=}\ & \langle i = id_0\rangle \\ & (\langle m = m_2, m_1\rangle \overline{err}, \overline{ok}, Cl_1\langle in_1, id_1, id_0, m_1, m_2, ok, err\rangle), \end{aligned}$$

$$Cl_1\langle in_1,id_1,id_0,m_1,m_2,ok,err\rangle$$

$$Test_2\langle in_2,id_2,id_0,m_1,m_2,ok,err,i,m\rangle \stackrel{def}{=} \langle i = id_0\rangle$$
$$(\langle m = m_1\rangle\overline{ok},Cl_2\langle in_2,id_2,id_0,m_1,m_2,ok,err\rangle),$$
$$Cl_2\langle in_2,id_2,id_0,m_1,m_2,ok,err\rangle.$$

On a donc pour $i = 1,2$:
$Cl_i\langle in_i,id_i,id_0,m_1,m_2,ok,err\rangle \stackrel{def}{=} in_i(i,m).Test_i\langle in_i,id_i,id_0,m_1,m_2,ok,err,i,m\rangle.$

Par le Lemme 36, il suffit de montrer que A_1 et A_2 sont bisimilaires jusqu' au $\sim$.

Soit *Cond* la condition décrit ci-dessous:

$$\begin{aligned}
Cond \stackrel{def}{=} & \big(\forall j \in \{p+1,\ldots,p+q\}.\ id_j \neq id_0\big)\wedge \\
& \big(\forall (j,l) \in \{p+1,\ldots,p+q\} \times \{1,\ldots,t_i+u_i+v_j\}.\ m_{j,l} \neq net\big)\wedge \\
& \big(\forall l \in \{1,\ldots,t_0\}.\ m_{0,l} \neq net\big)\wedge \\
& net \notin fn(P,Q)\wedge \qquad out_0 \notin fn(Q)\wedge \\
& \big(\forall (i,l) \in \{1,2\} \times \{t_i+u_i+1,\ldots,t_i+u_i+v_i\}.\ id_{i,l} \neq id_0\big)\wedge \\
& \big(\forall (i,l) \in \{1,2\} \times \{t_i+1,\ldots,t_i+u_i\}.\ id_{i,l} = id_0 \Rightarrow \\
& s_{i_1,l} \notin \{s_{i,l} | l \in \{t_i+u_i+v_i+1,\ldots t_i+u_i+v_i+w_i\}\} \\
& \cup \{s_{i_2,l} | l \in \{t_i+u_i+1,\ldots t_i+u_i+v_i\}\}\big)\wedge \\
& \big(\forall l \in \{1,\ldots,t_0\}.\ s_{0_1,l} \notin \\
& \{s_{i,l} | l \in \{t_i+u_i+v_i+1,\ldots t_i+u_i+v_i+w_i\}\}\cup \\
& \{s_{i_2,l} | l \in \{t_i+u_i+1,\ldots t_i+u_i+v_i\}\}\big)\wedge \\
& s_* \notin \{s_{i,l} | l \in \{t_i+u_i+v_i+1,\ldots t_i+u_i+v_i+w_i\}\} \\
& \cup \{s_{i_2,l} | l \in \{t_i+u_i+1,\ldots t_i+u_i+v_i\}\}\big).
\end{aligned}$$

et soit $C_{k,i}$ ($k \in \{1,\ldots,10\}$ les processus suivants:

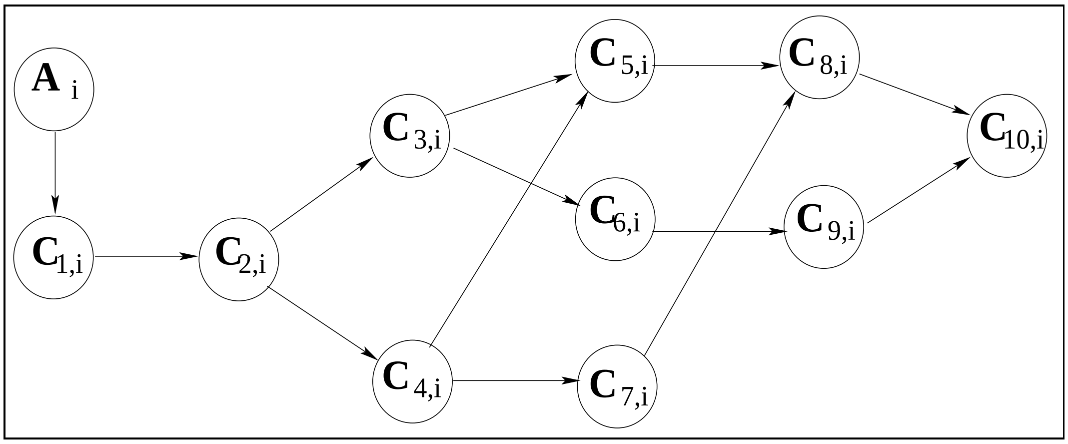

FIG. C.1 – **Les états accessibles de** A_i

$$
\begin{aligned}
C_{1,i}=\nu net\ \nu s_0\ \nu s_1\ \ldots \nu s_n\ \{&Server\langle req,net,s_0\rangle \parallel \\
&\prod_{j=3}^{p} \nu out_j\ \nu in_j\ \nu id_j\ \big(\overline{replay_j}[out_j,in_j,id_j].Fifo\langle out_j,in_j,id_j,net,s_0\rangle\big) \parallel \\
&\prod_{j=p+1}^{p+q} \nu s_{j_2,1}\ \ldots \nu s_{j_2,t_j}\ \{\big(Fifo_{out}\langle out_j,id_j,net,s_{j_2,t_j}\rangle \parallel \prod_{l=1}^{t_j} \overline{net}[id_j,m_{j,l},s_{j_2,l},s_{j_1,l}]\big) \parallel \\
&\qquad \nu loc\ \big(Fifo_{in}\langle in_j,net,loc\rangle \parallel \\
&\qquad\quad \prod_{l=t_j+1}^{t_j+u_j} Cell\langle id_{j,l},m_{j,l},s_{j_2,l},s_{j_1,l},in_j,loc\rangle \parallel \\
&\qquad\quad \prod_{l=t_j+u_j+1}^{t_j+u_j+v_j} Data\langle id_{j,l},m_{j,l},s_{j_2,l},in_j,loc\rangle \parallel \\
&\qquad\quad \prod_{l=t_j+u_j+v_j+1}^{t_j+u_j+v_j+w_j} ASend\langle loc,s_{j,l}\rangle \parallel \\
&\qquad\quad ASend\langle loc,s_0\rangle\big)\} \parallel \\
&\quad \nu out_0\ \nu in_0\ [Fifo_{out}\langle out_0,id_0,net,s_0\rangle \parallel \\
&\qquad \nu s_{0_2,1}\ \ldots \nu s_{0_2,t_0} \prod_{l=1}^{t_0} \overline{net}[id_0,m_{0,l},s_{0_2,l},s_{0_1,l}] \parallel \\
&\qquad\quad \nu loc\ (\overline{out_0}m_1.\overline{out_0}m_2.P \parallel Q \parallel Fifo_{in}\langle in_0,net,loc\rangle \parallel \\
ASend\langle loc,s_0\rangle)] \\
&\quad \nu in_i\ [Test_i\langle in_i,id_i,id_0,m_1,m_2,ok,err,i,m\rangle \parallel \nu loc\ (Fifo_{in}\langle in_i,net,loc\rangle \parallel \\
&\qquad\qquad \prod_{l=t_i+1}^{t_i+u_i} Cell\langle id_{i,l},m_{i,l},s_{i_2,l},s_{i_1,l},in_i,loc\rangle \parallel \\
&\qquad\qquad \prod_{l=t_i+u_i+1}^{t_i+u_i+v_i} Data\langle id_{i,l},m_{i,l},s_{i_2,l},in_i,loc\rangle\big) \parallel \\
&\qquad\qquad \prod_{l=t_i+u_i+v_i+1}^{t_i+u_i+v_i+w_i} ASend\langle loc,s_{i,l}\rangle \parallel \\
&\qquad\qquad ASend\langle loc,s_0\rangle]\big)\}.
\end{aligned}
$$

$$
\begin{aligned}
C_{2,i}=\nu net\ \nu s_0\ \nu s_1\ \ldots \nu s_n\ \{&Server\langle req,net,s_0\rangle \parallel \\
&\prod_{j=3}^{p} \nu out_j\ \nu in_j\ \nu id_j\ \big(\overline{replay_j}[out_j,in_j,id_j].Fifo\langle out_j,in_j,id_j,net,s_0\rangle\big) \parallel \\
&\prod_{j=p+1}^{p+q} \nu s_{j_2,1}\ \ldots \nu s_{j_2,t_j}\ \{\big(Fifo_{out}\langle out_j,id_j,net,s_{j_2,t_j}\rangle \parallel \prod_{l=1}^{t_j} \overline{net}[id_j,m_{j,l},s_{j_2,l},s_{j_1,l}]\big) \parallel \\
&\qquad \nu loc\ \big(Fifo_{in}\langle in_j,net,loc\rangle \parallel \\
&\qquad\quad \prod_{l=t_j+1}^{t_j+u_j} Cell\langle id_{j,l},m_{j,l},s_{j_2,l},s_{j_1,l},in_j,loc\rangle \parallel \\
&\qquad\quad \prod_{l=t_j+u_j+1}^{t_j+u_j+v_j} Data\langle id_{j,l},m_{j,l},s_{j_2,l},in_j,loc\rangle \parallel \\
&\qquad\quad \prod_{l=t_j+u_j+v_j+1}^{t_j+u_j+v_j+w_j} ASend\langle loc,s_{j,l}\rangle \parallel \\
&\qquad\quad ASend\langle loc,s_0\rangle\big)\} \parallel \\
&\nu s_{m_1}\ \big(\overline{net}[id_0,m_1,s_{m_1},s_0] \parallel \\
&\quad \nu out_0\ \nu in_0\ [Fifo_{out}\langle out_0,id_0,net,s_{m_1}\rangle \parallel \\
&\qquad \nu s_{0_2,1}\ \ldots \nu s_{0_2,t_0} \prod_{l=1}^{t_0} \overline{net}[id_0,m_{0,l},s_{0_2,l},s_{0_1,l}] \parallel \\
&\qquad \nu loc\ (\overline{out_0}m_2.P \parallel Q \parallel Fifo_{in}\langle in_0,net,loc\rangle \parallel ASend\langle loc,s_0\rangle)]\big) \\
&\quad \nu in_i\ [Test_i\langle in_i,id_i,id_0,m_1,m_2,ok,err,i,m\rangle \parallel \nu loc\ (Fifo_{in}\langle in_i,net,loc\rangle \parallel \\
&\qquad\qquad \prod_{l=t_i+1}^{t_i+u_i} Cell\langle id_{i,l},m_{i,l},s_{i_2,l},s_{i_1,l},in_i,loc\rangle \parallel \\
&\qquad\qquad \prod_{l=t_i+u_i+1}^{t_i+u_i+v_i} Data\langle id_{i,l},m_{i,l},s_{i_2,l},in_i,loc\rangle\big) \parallel \\
&\qquad\qquad \prod_{l=t_i+u_i+v_i+1}^{t_i+u_i+v_i+w_i} ASend\langle loc,s_{i,l}\rangle \parallel \\
&\qquad\qquad ASend\langle loc,s_0\rangle]\big)\}.
\end{aligned}
$$

$$
\begin{aligned}
C_{3,i} = \nu net\ \nu s_0\ \nu s_1\ \ldots \nu s_n\ \{ & Server\langle req,net,s_0\rangle \parallel \\
& \textstyle\prod_{j=3}^{p} \nu out_j\ \nu in_j\ \nu id_j\ \big(\overline{replay_j}[out_j,in_j,id_j].Fifo\langle out_j,in_j,id_j,net,s_0\rangle\big) \parallel \\
& \textstyle\prod_{j=p+1}^{p+q} \nu s_{j_2,1}\ \ldots \nu s_{j_2,t_j}\ \{\big(Fifo_{out}\langle out_j,id_j,net,s_{j_2,t_j}\rangle \parallel \prod_{l=1}^{t_j} \overline{net}[id_j,m_{j,l},s_{j_2,l},s_{j_1,l}]\big) \parallel \\
& \quad \nu loc\ \big(Fifo_{in}\langle in_j,net,loc\rangle \parallel \\
& \quad\quad \textstyle\prod_{l=t_j+1}^{t_j+u_j} Cell\langle id_{j,l},m_{j,l},s_{j_2,l},s_{j_1,l},in_j,loc\rangle \parallel \\
& \quad\quad \textstyle\prod_{l=t_j+u_j+1}^{t_j+u_j+v_j} Data\langle id_{j,l},m_{j,l},s_{j_2,l},in_j,loc\rangle \parallel \\
& \quad\quad \textstyle\prod_{l=t_j+u_j+v_j+1}^{t_j+u_j+v_j+w_j} ASend\langle loc,s_{j,l}\rangle \parallel \\
& \quad\quad ASend\langle loc,s_0\rangle\big)\} \parallel \\
& \nu s_{m_1}\ \nu s_{m_2}\ \big(\overline{net}[id_0,m_1,s_{m_1},s_0] \parallel \overline{net}[id_0,m_2,s_{m_2},s_{m_1}] \parallel \\
& \quad \nu out_0\ \nu in_0\ [Fifo_{out}\langle out_0,id_0,net,s_*\rangle \parallel \\
& \quad\quad \nu s_{0_2,1}\ \ldots \nu s_{0_2,t_0} \textstyle\prod_{l=1}^{t_0} \overline{net}[id_0,m_{0,l},s_{0_2,l},s_{0_1,l}] \parallel \\
& \quad\quad \nu loc\ (P \parallel Fifo_{in}\langle in_0,net,loc\rangle \parallel ASend\langle loc,s_0\rangle)]\big) \\
& \quad \nu in_i\ [Test_i\langle in_i,id_i,id_0,m_1,m_2,ok,err,i,m\rangle \parallel \nu loc\ (Fifo_{in}\langle in_i,net,loc\rangle \parallel \\
& \quad\quad \textstyle\prod_{l=t_i+1}^{t_i+u_i} Cell\langle id_{i,l},m_{i,l},s_{i_2,l},s_{i_1,l},in_i,loc\rangle \parallel \\
& \quad\quad \textstyle\prod_{l=t_i+u_i+1}^{t_i+u_i+v_i} Data\langle id_{i,l},m_{i,l},s_{i_2,l},in_i,loc\rangle\big) \parallel \\
& \quad\quad \textstyle\prod_{l=t_i+u_i+v_i+1}^{t_i+u_i+v_i+w_i} ASend\langle loc,s_{i,l}\rangle \parallel \\
& \quad\quad ASend\langle loc,s_0\rangle]\big)\}.
\end{aligned}
$$

$$
\begin{aligned}
C_{4,i} = \nu net\ \nu s_0\ \nu s_1\ \ldots \nu s_n\ \nu s_{m_1}\ \{ & Server\langle req,net,s_0\rangle \parallel \\
& \textstyle\prod_{j=3}^{p} \nu out_j\ \nu in_j\ \nu id_j\ \big(\overline{replay_j}[out_j,in_j,id_j].Fifo\langle out_j,in_j,id_j,net,s_0\rangle\big) \parallel \\
& \textstyle\prod_{j=p+1}^{p+q} \nu s_{j_2,1}\ \ldots \nu s_{j_2,t_j}\ \{\big(Fifo_{out}\langle out_j,id_j,net,s_{j_2,t_j}\rangle \parallel \prod_{l=1}^{t_j} \overline{net}[id_j,m_{j,l},s_{j_2,l},s_{j_1,l}]\big) \parallel \\
& \quad \nu loc\ \big(Fifo_{in}\langle in_j,net,loc\rangle \parallel \\
& \quad\quad \textstyle\prod_{l=t_j+1}^{t_j+u_j} Cell\langle id_{j,l},m_{j,l},s_{j_2,l},s_{j_1,l},in_j,loc\rangle \parallel \\
& \quad\quad \textstyle\prod_{l=t_j+u_j+1}^{t_j+u_j+v_j} Data\langle id_{j,l},m_{j,l},s_{j_2,l},in_j,loc\rangle \parallel \\
& \quad\quad \textstyle\prod_{l=t_j+u_j+v_j+1}^{t_j+u_j+v_j+w_j} ASend\langle loc,s_{j,l}\rangle \parallel \\
& \quad\quad ASend\langle loc,s_0\rangle\big)\} \parallel \\
& \quad \nu out_0\ \nu in_0\ [Fifo_{out}\langle out_0,id_0,net,s_{m_1}\rangle \parallel \\
& \quad\quad \nu s_{0_2,1}\ \ldots \nu s_{0_2,t_0} \textstyle\prod_{l=1}^{t_0} \overline{net}[id_0,m_{0,l},s_{0_2,l},s_{0_1,l}] \parallel \\
& \quad\quad \nu loc\ (\overline{out_0}m_2.P \parallel Q \parallel Fifo_{in}\langle in_0,net,loc\rangle \parallel ASend\langle loc,s_0\rangle)] \\
& \nu in_i\ [Test_i\langle in_i,id_i,id_0,m_1,m_2,ok,err,i,m\rangle \parallel \nu loc\ (Fifo_{in}\langle in_i,net,loc\rangle \parallel \\
& \quad\quad \textstyle\prod_{l=t_i+1}^{t_i+u_i} Cell\langle id_{i,l},m_{i,l},s_{i_2,l},s_{i_1,l},in_i,loc\rangle \parallel \\
& \quad\quad Cell\langle id_0,m_1,s_{m_1},s_0,in_i,loc\rangle \parallel \\
& \quad\quad \textstyle\prod_{l=t_i+u_i+1}^{t_i+u_i+v_i} Data\langle id_{i,l},m_{i,l},s_{i_2,l},in_i,loc\rangle\big) \parallel \\
& \quad\quad \textstyle\prod_{l=t_i+u_i+v_i+1}^{t_i+u_i+v_i+w_i} ASend\langle loc,s_{i,l}\rangle \parallel \\
& \quad\quad ASend\langle loc,s_0\rangle]\}.
\end{aligned}
$$

$$
\begin{aligned}
C_{5,i} = \nu net\ \nu s_0\ \nu s_1\ \ldots \nu s_n\ \nu s_{m_1}\ \{ & Server\langle req,net,s_0\rangle \parallel \\
& \textstyle\prod_{j=3}^{p} \nu out_j\ \nu in_j\ \nu id_j\ \big(\overline{replay_j}[out_j,in_j,id_j].Fifo\langle out_j,in_j,id_j,net,s_0\rangle\big) \parallel \\
& \textstyle\prod_{j=p+1}^{p+q} \nu s_{j_2,1}\ \ldots \nu s_{j_2,t_j}\ \{\big(Fifo_{out}\langle out_j,id_j,net,s_{j_2,t_j}\rangle \parallel \prod_{l=1}^{t_j} \overline{net}[id_j,m_{j,l},s_{j_2,l},s_{j_1,l}]\big) \parallel \\
& \qquad \nu loc\ \big(Fifo_{in}\langle in_j,net,loc\rangle \parallel \\
& \qquad\quad \textstyle\prod_{l=t_j+1}^{t_j+u_j} Cell\langle id_{j,l},m_{j,l},s_{j_2,l},s_{j_1,l},in_j,loc\rangle \parallel \\
& \qquad\quad \textstyle\prod_{l=t_j+u_j+1}^{t_j+u_j+v_j} Data\langle id_{j,l},m_{j,l},s_{j_2,l},in_j,loc\rangle \parallel \\
& \qquad\quad \textstyle\prod_{l=t_j+u_j+v_j+1}^{t_j+u_j+v_j+w_j} ASend\langle loc,s_{j,l}\rangle \parallel \\
& \qquad\quad ASend\langle loc,s_0\rangle\big)\} \parallel \\
& \nu s_{m_2}\ \big(\overline{net}[id_0,m_2,s_{m_2},s_{m_1}] \parallel \\
& \quad \nu out_0\ \nu in_0\ [Fifo_{out}\langle out_0,id_0,net,s_*\rangle \parallel \\
& \qquad \nu s_{0_2,1}\ \ldots \nu s_{0_2,t_0} \textstyle\prod_{l=1}^{t_0} \overline{net}[id_0,m_{0,l},s_{0_2,l},s_{0_1,l}] \parallel \\
& \qquad \nu loc\ (P \parallel Fifo_{in}\langle in_0,net,loc\rangle \parallel ASend\langle loc,s_0\rangle)]\big) \\
& \nu in_i\ [Test_i\langle in_i,id_i,id_0,m_1,m_2,ok,err,i,m\rangle \parallel \nu loc\ (Fifo_{in}\langle in_i,net,loc\rangle \parallel \\
& \qquad \textstyle\prod_{l=t_i+1}^{t_i+u_i} Cell\langle id_{i,l},m_{i,l},s_{i_2,l},s_{i_1,l},in_i,loc\rangle \parallel \\
& \qquad Cell\langle id_0,m_1,s_{m_1},s_0,in_i,loc\rangle \parallel \\
& \qquad \textstyle\prod_{l=t_i+u_i+1}^{t_i+u_i+v_i} Data\langle id_{i,l},m_{i,l},s_{i_2,l},in_i,loc\rangle\big) \parallel \\
& \qquad \textstyle\prod_{l=t_i+u_i+v_i+1}^{t_i+u_i+v_i+w_i} ASend\langle loc,s_{i,l}\rangle \parallel \\
& \qquad ASend\langle loc,s_0\rangle]\}.
\end{aligned}
$$

$$
\begin{aligned}
C_{6,i} = \nu net\ \nu s_0\ \nu s_1\ \ldots \nu s_n\ \nu s_{m_1}\ \nu s_{m_2}\ \{ & Server\langle req,net,s_0\rangle \parallel \\
& \textstyle\prod_{j=3}^{p} \nu out_j\ \nu in_j\ \nu id_j\ \big(\overline{replay_j}[out_j,in_j,id_j].Fifo\langle out_j,in_j,id_j,net,s_0\rangle\big) \parallel \\
& \textstyle\prod_{j=p+1}^{p+q} \nu s_{j_2,1}\ \ldots \nu s_{j_2,t_j}\ \{\big(Fifo_{out}\langle out_j,id_j,net,s_{j_2,t_j}\rangle \parallel \prod_{l=1}^{t_j} \overline{net}[id_j,m_{j,l},s_{j_2,l},s_{j_1,l}]\big) \parallel \\
& \qquad \nu loc\ \big(Fifo_{in}\langle in_j,net,loc\rangle \parallel \\
& \qquad\quad \textstyle\prod_{l=t_j+1}^{t_j+u_j} Cell\langle id_{j,l},m_{j,l},s_{j_2,l},s_{j_1,l},in_j,loc\rangle \parallel \\
& \qquad\quad \textstyle\prod_{l=t_j+u_j+1}^{t_j+u_j+v_j} Data\langle id_{j,l},m_{j,l},s_{j_2,l},in_j,loc\rangle \parallel \\
& \qquad\quad \textstyle\prod_{l=t_j+u_j+v_j+1}^{t_j+u_j+v_j+w_j} ASend\langle loc,s_{j,l}\rangle \parallel \\
& \qquad\quad ASend\langle loc,s_0\rangle\big)\} \parallel \\
& \big(\overline{net}[id_0,m_1,s_{m_1},s_0] \parallel \\
& \quad \nu out_0\ \nu in_0\ [Fifo_{out}\langle out_0,id_0,net,s_*\rangle \parallel \\
& \qquad \nu s_{0_2,1}\ \ldots \nu s_{0_2,t_0} \textstyle\prod_{l=1}^{t_0} \overline{net}[id_0,m_{0,l},s_{0_2,l},s_{0_1,l}] \parallel \\
& \qquad \nu loc\ (P \parallel Fifo_{in}\langle in_0,net,loc\rangle \parallel ASend\langle loc,s_0\rangle)]\big) \\
& \nu in_i\ [Test_i\langle in_i,id_i,id_0,m_1,m_2,ok,err,i,m\rangle \parallel \nu loc\ (Fifo_{in}\langle in_i,net,loc\rangle \parallel \\
& \qquad \textstyle\prod_{l=t_i+1}^{t_i+u_i} Cell\langle id_{i,l},m_{i,l},s_{i_2,l},s_{i_1,l},in_i,loc\rangle \parallel \\
& \qquad Cell\langle id_0,m_2,s_{m_2},s_{m_1},in_i,loc\rangle \parallel \\
& \qquad \textstyle\prod_{l=t_i+u_i+1}^{t_i+u_i+v_i} Data\langle id_{i,l},m_{i,l},s_{i_2,l},in_i,loc\rangle\big) \parallel \\
& \qquad \textstyle\prod_{l=t_i+u_i+v_i+1}^{t_i+u_i+v_i+w_i} ASend\langle loc,s_{i,l}\rangle \parallel \\
& \qquad ASend\langle loc,s_0\rangle]\}.
\end{aligned}
$$

$$
\begin{array}{l}
C_{7,i} = \nu net\ \nu s_0\ \nu s_1\ \ldots \nu s_n\ \nu s_{m_1}\ \{Server\langle req,net,s_0\rangle \parallel \\
\quad \prod_{j=3}^{p} \nu out_j\ \nu in_j\ \nu id_j\ \big(\overline{replay_j}[out_j,in_j,id_j].Fifo\langle out_j,in_j,id_j,net,s_0\rangle\big) \parallel \\
\quad \prod_{j=p+1}^{p+q} \nu s_{j_2,1}\ \ldots \nu s_{j_2,t_j}\ \{\big(Fifo_{out}\langle out_j,id_j,net,s_{j_2,t_j}\rangle \parallel \prod_{l=1}^{t_j} \overline{net}[id_j,m_{j,l},s_{j_2,l},s_{j_1,l}]\big) \parallel \\
\qquad \nu loc\ \big(Fifo_{in}\langle in_j,net,loc\rangle \parallel \\
\qquad\quad \prod_{l=t_j+1}^{t_j+u_j} Cell\langle id_{j,l},m_{j,l},s_{j_2,l},s_{j_1,l},in_j,loc\rangle \parallel \\
\qquad\quad \prod_{l=t_j+u_j+1}^{t_j+u_j+v_j} Data\langle id_{j,l},m_{j,l},s_{j_2,l},in_j,loc\rangle \parallel \\
\qquad\quad \prod_{l=t_j+u_j+v_j+1}^{t_j+u_j+v_j+w_j} ASend\langle loc,s_{j,l}\rangle \parallel \\
\qquad\quad ASend\langle loc,s_0\rangle\big)\} \parallel \\
\qquad \nu out_0\ \nu in_0\ [Fifo_{out}\langle out_0,id_0,net,s_{m_1}\rangle \parallel \\
\qquad\quad \nu s_{0_2,1}\ \ldots \nu s_{0_2,t_0} \prod_{l=1}^{t_0} \overline{net}[id_0,m_{0,l},s_{0_2,l},s_{0_1,l}] \parallel \\
\qquad\quad \nu loc\ (\overline{out_0}m_2.P \parallel Q \parallel Fifo_{in}\langle in_0,net,loc\rangle \parallel ASend\langle loc,s_0\rangle)] \\
\quad \nu in_i\ [Test_i\langle in_i,id_i,id_0,m_1,m_2,ok,err,i,m\rangle \parallel \nu loc\ (Fifo_{in}\langle in_i,net,loc\rangle \parallel \\
\qquad\quad \prod_{l=t_i+1}^{t_i+u_i} Cell\langle id_{i,l},m_{i,l},s_{i_2,l},s_{i_1,l},in_i,loc\rangle \parallel \\
\qquad\quad \prod_{l=t_i+u_i+1}^{t_i+u_i+v_i} Data\langle id_{i,l},m_{i,l},s_{i_2,l},in_i,loc\rangle\big) \parallel \\
\qquad\quad Data\langle id_0,m_1,s_{m_1},in_i,loc\rangle \parallel \\
\qquad\quad \prod_{l=t_i+u_i+v_i+1}^{t_i+u_i+v_i+w_i} ASend\langle loc,s_{i,l}\rangle \parallel \\
\qquad\quad ASend\langle loc,s_0\rangle]\}.
\end{array}
$$

$$
\begin{array}{l}
C_{8,i} = \nu net\ \nu s_0\ \nu s_1\ \ldots \nu s_n\ \nu s_{m_1}\ \{Server\langle req,net,s_0\rangle \parallel \\
\quad \prod_{j=3}^{p} \nu out_j\ \nu in_j\ \nu id_j\ \big(\overline{replay_j}[out_j,in_j,id_j].Fifo\langle out_j,in_j,id_j,net,s_0\rangle\big) \parallel \\
\quad \prod_{j=p+1}^{p+q} \nu s_{j_2,1}\ \ldots \nu s_{j_2,t_j}\ \{\big(Fifo_{out}\langle out_j,id_j,net,s_{j_2,t_j}\rangle \parallel \prod_{l=1}^{t_j} \overline{net}[id_j,m_{j,l},s_{j_2,l},s_{j_1,l}]\big) \parallel \\
\qquad \nu loc\ \big(Fifo_{in}\langle in_j,net,loc\rangle \parallel \\
\qquad\quad \prod_{l=t_j+1}^{t_j+u_j} Cell\langle id_{j,l},m_{j,l},s_{j_2,l},s_{j_1,l},in_j,loc\rangle \parallel \\
\qquad\quad \prod_{l=t_j+u_j+1}^{t_j+u_j+v_j} Data\langle id_{j,l},m_{j,l},s_{j_2,l},in_j,loc\rangle \parallel \\
\qquad\quad \prod_{l=t_j+u_j+v_j+1}^{t_j+u_j+v_j+w_j} ASend\langle loc,s_{j,l}\rangle \parallel \\
\qquad\quad ASend\langle loc,s_0\rangle\big)\} \parallel \\
\quad \nu s_{m_2}\ \big(\overline{net}[id_0,m_2,s_{m_2},s_{m_1}] \parallel \\
\qquad \nu out_0\ \nu in_0\ [Fifo_{out}\langle out_0,id_0,net,s_*\rangle \parallel \\
\qquad\quad \nu s_{0_2,1}\ \ldots \nu s_{0_2,t_0} \prod_{l=1}^{t_0} \overline{net}[id_0,m_{0,l},s_{0_2,l},s_{0_1,l}] \parallel \\
\qquad\quad \nu loc\ (P \parallel Fifo_{in}\langle in_0,net,loc\rangle \parallel ASend\langle loc,s_0\rangle)]\big) \\
\quad \nu in_i\ [Test_i\langle in_i,id_i,id_0,m_1,m_2,ok,err,i,m\rangle \parallel \nu loc\ (Fifo_{in}\langle in_i,net,loc\rangle \parallel \\
\qquad\quad \prod_{l=t_i+1}^{t_i+u_i} Cell\langle id_{i,l},m_{i,l},s_{i_2,l},s_{i_1,l},in_i,loc\rangle \parallel \\
\qquad\quad \prod_{l=t_i+u_i+1}^{t_i+u_i+v_i} Data\langle id_{i,l},m_{i,l},s_{i_2,l},in_i,loc\rangle\big) \parallel \\
\qquad\quad Data\langle id_0,m_1,s_{m_1},in_i,loc\rangle \parallel \\
\qquad\quad \prod_{l=t_i+u_i+v_i+1}^{t_i+u_i+v_i+w_i} ASend\langle loc,s_{i,l}\rangle \parallel \\
\qquad\quad ASend\langle loc,s_0\rangle]\}.
\end{array}
$$

$$C_{9,i} = \nu net\ \nu s_0\ \nu s_1\ \ldots \nu s_n\ \nu s_{m_1}\ \nu s_{m_2}\ \{Server\langle req,net,s_0\rangle \parallel$$
$$\prod_{j=3}^{p} \nu out_j\ \nu in_j\ \nu id_j\ \big(\overline{replay_j}[out_j,in_j,id_j].Fifo\langle out_j,in_j,id_j,net,s_0\rangle\big) \parallel$$
$$\prod_{j=p+1}^{p+q} \nu s_{j_2,1}\ \ldots \nu s_{j_2,t_j}\ \{\big(Fifo_{out}\langle out_j,id_j,net,s_{j_2,t_j}\rangle \parallel \prod_{l=1}^{t_j} \overline{net}[id_j,m_{j,l},s_{j_2,l},s_{j_1,l}]\big) \parallel$$
$$\nu loc\ \big(Fifo_{in}\langle in_j,net,loc\rangle \parallel$$
$$\prod_{l=t_j+1}^{t_j+u_j} Cell\langle id_{j,l},m_{j,l},s_{j_2,l},s_{j_1,l},in_j,loc\rangle \parallel$$
$$\prod_{l=t_j+u_j+1}^{t_j+u_j+v_j} Data\langle id_{j,l},m_{j,l},s_{j_2,l},in_j,loc\rangle \parallel$$
$$\prod_{l=t_j+u_j+v_j+1}^{t_j+u_j+v_j+w_j} ASend\langle loc,s_{j,l}\rangle \parallel$$
$$ASend\langle loc,s_0\rangle\big)\} \parallel$$
$$\nu out_0\ \nu in_0\ [Fifo_{out}\langle out_0,id_0,net,s_*\rangle \parallel$$
$$\nu s_{0_2,1}\ \ldots \nu s_{0_2,t_0} \prod_{l=1}^{t_0} \overline{net}[id_0,m_{0,l},s_{0_2,l},s_{0_1,l}] \parallel$$
$$\nu loc\ (P \parallel Fifo_{in}\langle in_0,net,loc\rangle \parallel ASend\langle loc,s_0\rangle)]$$
$$\nu in_i\ [Test_i\langle in_i,id_i,id_0,m_1,m_2,ok,err,i,m\rangle \parallel \nu loc\ (Fifo_{in}\langle in_i,net,loc\rangle \parallel$$
$$\prod_{l=t_i+1}^{t_i+u_i} Cell\langle id_{i,l},m_{i,l},s_{i_2,l},s_{i_1,l},in_i,loc\rangle \parallel$$
$$Cell\langle id_0,m_1,s_{m_1},s_0,in_i,loc\rangle \parallel$$
$$Cell\langle id_0,m_2,s_{m_2},s_{m_1},in_i,loc\rangle \parallel$$
$$\prod_{l=t_i+u_i+1}^{t_i+u_i+v_i} Data\langle id_{i,l},m_{i,l},s_{i_2,l},in_i,loc\rangle\big) \parallel$$
$$\prod_{l=t_i+u_i+v_i+1}^{t_i+u_i+v_i+w_i} ASend\langle loc,s_{i,l}\rangle \parallel$$
$$ASend\langle loc,s_0\rangle]\}.$$

$$C_{10,i} = \nu net\ \nu s_0\ \nu s_1\ \ldots \nu s_n\ \nu s_{m_1}\ \nu s_{m_2}\ \{Server\langle req,net,s_0\rangle \parallel$$
$$\prod_{j=3}^{p} \nu out_j\ \nu in_j\ \nu id_j\ \big(\overline{replay_j}[out_j,in_j,id_j].Fifo\langle out_j,in_j,id_j,net,s_0\rangle\big) \parallel$$
$$\prod_{j=p+1}^{p+q} \nu s_{j_2,1}\ \ldots \nu s_{j_2,t_j}\ \{\big(Fifo_{out}\langle out_j,id_j,net,s_{j_2,t_j}\rangle \parallel \prod_{l=1}^{t_j} \overline{net}[id_j,m_{j,l},s_{j_2,l},s_{j_1,l}]\big) \parallel$$
$$\nu loc\ \big(Fifo_{in}\langle in_j,net,loc\rangle \parallel$$
$$\prod_{l=t_j+1}^{t_j+u_j} Cell\langle id_{j,l},m_{j,l},s_{j_2,l},s_{j_1,l},in_j,loc\rangle \parallel$$
$$\prod_{l=t_j+u_j+1}^{t_j+u_j+v_j} Data\langle id_{j,l},m_{j,l},s_{j_2,l},in_j,loc\rangle \parallel$$
$$\prod_{l=t_j+u_j+v_j+1}^{t_j+u_j+v_j+w_j} ASend\langle loc,s_{j,l}\rangle \parallel$$
$$ASend\langle loc,s_0\rangle\big)\} \parallel$$
$$\nu out_0\ \nu in_0\ [Fifo_{out}\langle out_0,id_0,net,s_*\rangle \parallel$$
$$\nu s_{0_2,1}\ \ldots \nu s_{0_2,t_0} \prod_{l=1}^{t_0} \overline{net}[id_0,m_{0,l},s_{0_2,l},s_{0_1,l}] \parallel$$
$$\nu loc\ (P \parallel Fifo_{in}\langle in_0,net,loc\rangle \parallel ASend\langle loc,s_0\rangle)]$$
$$\nu in_i\ [Test_i\langle in_i,id_i,id_0,m_1,m_2,ok,err,i,m\rangle \parallel \nu loc\ (Fifo_{in}\langle in_i,net,loc\rangle \parallel$$
$$\prod_{l=t_i+1}^{t_i+u_i} Cell\langle id_{i,l},m_{i,l},s_{i_2,l},s_{i_1,l},in_i,loc\rangle \parallel$$
$$Cell\langle id_0,m_2,s_{m_2},s_{m_1},in_i,loc\rangle \parallel$$
$$\prod_{l=t_i+u_i+1}^{t_i+u_i+v_i} Data\langle id_{i,l},m_{i,l},s_{i_2,l},in_i,loc\rangle\big) \parallel$$
$$Data\langle id_0,m_1,s_{m_1},in_i,loc\rangle \parallel$$
$$\prod_{l=t_i+u_i+v_i+1}^{t_i+u_i+v_i+w_i} ASend\langle loc,s_{i,l}\rangle \parallel$$
$$ASend\langle loc,s_0\rangle]\}.$$

Dans $Cond$, la condition $\big(\forall j \in \{p+1,\dots,p+q\}.\ id_j \neq id_0\big)$ exprime le fait que seulement le client Cl_0 peut envoyer des messages "signés" avec id_0. Les conditions $\big(\forall (j,l) \in \{p+1,\dots,p+q\} \times \{1,\dots,t_i+u_i+v_j\}.\ m_{j,l} \neq net\big)$ et $\big(\forall l \in \{1,\dots,t_0\}.\ m_{0,l} \neq net\big)$ assurent que net ne peut pas etre employé comme message, et donc garantissent l' invariance de la condition $net \notin fn(P,Q)$.La condition $net \notin fn(P,Q)$ assure que le client Cl_0 peut interagir avec les autres clients seulement à travers son interface $Fifo$ et ne peut pas inserer directement des messages dans le réseau (ou recevoir directement des messages depuis le réseau). La condition $out_0 \notin fn(Q)$ assure que le premier message envoyé par Cl_0 est le message m_1. Les autres conditions de $Cond$ assurent que le premier message de source id_0 qui peut etre delivré au clients Cl_1 ou Cl_2, est forcement le message m_1.

Pour $k \in \{1,2\}$, $C_{k,i}$ représentent les états accesibles de A_i, (à partir du moment où m_1 et m_2 sont envoyés par Cl_0, et jusqu' à la récéption par Cl_1 et Cl_2 du message m_1).

Dans $C_{k,i}$, $\prod_{j=3}^{p} \nu out_j\ \nu in_j\ \nu id_j\ \big(\overline{replay_j}[out_j,in_j,id_j].Fifo\langle out_j,in_j,id_j,net,s_0\rangle\big)$, représente les acquittements pour les clients qui sont en train d' adherer au groupe, tandis que

$$\begin{aligned}\prod_{j=p+1}^{p+q} \nu s_{j_2,1} \dots \nu s_{j_2,t_j} \{ \big(& Fifo_{out}\langle out_j,id_j,net,s_{j_2,t_j}\rangle \parallel \prod_{l=1}^{t_j} \overline{net}[id_j,m_{j,l},s_{j_2,l},s_{j_1,l}]\big) \parallel \\ & \nu loc\ \big(Fifo_{in}\langle in_j,net,loc\rangle \parallel \\ & \quad \prod_{l=t_j+1}^{t_j+u_j} Cell\langle id_{j,l},m_{j,l},s_{j_2,l},s_{j_1,l},in_j,loc\rangle \parallel \\ & \quad \prod_{l=t_j+u_j+1}^{t_j+u_j+v_j} Data\langle id_{j,l},m_{j,l},s_{j_2,l},in_j,loc\rangle \parallel \\ & \quad \prod_{l=t_j+u_j+v_j+1}^{t_j+u_j+v_j+w_j} ASend\langle loc,s_{j,l}\rangle \parallel \\ & \quad ASend\langle loc,s_0\rangle\big)\}\end{aligned}$$

représente "la partie réseau" des clients (autres que Cl_0, Cl_1 et Cl_2) qui font partie du groupe. $\prod_{l=1}^{t_j} \overline{net}[id_j,m_{j,l},s_{j_2,l},s_{j_1,l}]$ sont les messages que le client d' identité id_j a diffusé dans le réseau. $\prod_{l=t_j+1}^{t_j+u_j} Cell\langle id_{j,l},m_{j,l},s_{j_2,l},s_{j_1,l},in_j,loc\rangle$ sont les messages récus par "le tampon" du client d' identité id_j, mais qui pour des raisons d' ordonencement ne peuvent pas etre délivrés au client. $\prod_{l=t_j+u_j+1}^{t_j+u_j+v_j} Data\langle id_{j,l},m_{j,l},s_{j_2,l},in_j,loc\rangle$ sont les messages qui n' ont pas été encore consommés par le client d' identité id_j (mais qui peuvent etre délivrés au client sans aucune condition).

Soit $\mathcal{S} \stackrel{def}{=} \{ \big(T_1,T_2\big) \mid \exists C_{k,i}$tel que $T_1 = C_{k,1}$, $T_2 = C_{k,2}$, $k \in \{1,\dots,10\}$, et $C_{k,i}$ satisfait $Cond\}$ $\bigcup \{(p,p)\ \mid\ p \in \mathcal{P}_b\}$.

Alors on peut prouver que $\mathcal{S}$ est une bisimulation jusqu' au $\sim$, et que $(A_1,A_2) \in \mathcal{S}$.

Annexe D

Preuves rélatives au congruences

Remarque 48 $\sim_+$ est préservée par $+$.

Preuve

Soit $p \sim_+ q$. On prouve $(p+r) \sim_+ (q+r)$.

Si $p + r \xrightarrow{\alpha} s$, alors cette transition est inférée par la règle (7) ou sa symétrique, à partir de $p \xrightarrow{\alpha} s$ ou $r \xrightarrow{\alpha} s$.

Si $p \xrightarrow{\alpha} s$, comme $p \sim_+ q$, par l' utilisation de la Définition 47, on obtient $q \xrightarrow{\alpha} t$ avec $s \sim t$. Par l' utilisation de la règle (7) ou sa symétrique, on obtient $q + r \xrightarrow{\alpha} t$.

Si $r \xrightarrow{\alpha} s$, par l' utilisation de la règle (8), on obtient $q + r \xrightarrow{\alpha} s$, et evidemment $s \sim s$.

Le cas $q + r \xrightarrow{\alpha} s$ est similaire.

□ *Remarque* 48

Théoreme 61 (correction de $\mathcal{A}$ pour $\sim_c$)

Si $\mathcal{A} \vdash p = q$ alors $p \sim_c q$.

Preuve

Il suffit de montrer la correction des règles données dans le Tableau 4.2. L' asertion suit ensuite par induction sur la longueur de l' inférence de $\mathcal{A} \vdash p = q$.

Nous justifions seulement la correction des axiomes (H) et (SP). Les autres cas sont plus simples.

– (H) si $x \notin fn(p)$ et $\forall b \in In(p)\big(\phi \Rightarrow \langle a \neq b\rangle\big)$ alors $\alpha.p = \alpha.(p + \phi a(x).p)$

Soit σ une substitution. On peut supposer que $x \notin (prdom(\sigma) \cup prcod(\sigma)$.

Nous devons prouver $(\alpha.p)\sigma{\sim_+}(\alpha.(p + \phi a(x).p))\sigma$. Par la Définition 47, il suffit de prouver $p\sigma{\sim}(p + \phi a(x).p)\sigma$, c.a.d. $p\sigma{\sim}p\sigma + (\phi\sigma)\sigma(a)(x).p\sigma$.

Si σ ne concorde pas avec ϕ (Définition 65), alors $\phi\sigma \Longleftrightarrow \mathit{False}$, et donc $(p\sigma + (\phi\sigma)\sigma(a)(x).p\sigma) \xrightarrow{\alpha} q$ si et seulement si $p\sigma \xrightarrow{\alpha} q$.

Supposons que σ concorde avec ϕ. Si $p\sigma \xrightarrow{\alpha} q$, évidement $(p\sigma + (\phi\sigma)\sigma(a)(x).p\sigma) \xrightarrow{\alpha} q$.

Suppsons $(p\sigma + (\phi\sigma)\sigma(a)(x).p\sigma) \xrightarrow{\alpha} q$.

Alors la derniere règle utilisée est la règle (7) ou sa symétrique, et soit $p\sigma \xrightarrow{\alpha} q$, soit $(\phi\sigma)\sigma(a)(x).p\sigma \xrightarrow{\alpha} q$. Le premier cas est trivial. Supposons $(\phi\sigma)\sigma(a)(x).p\sigma \xrightarrow{\alpha} q$.

Alors, pour un certain $c \in Ch_b$, $\alpha = \sigma(a)\langle c\rangle$ et comme $x \notin fn(p)$, on obtient $q = p\sigma[c/x] = p\sigma$.

Soit $b \in In(p)$. Alors $\phi \Rightarrow \langle a \neq b\rangle$. Comme σ concorde avec ϕ, par la Définition 65, on obtient $\sigma(b) \neq \sigma(a)$. Utilisant le fait que $In(p\sigma) = \{\sigma(b)|b \in In(p)\}$, on obtient $\sigma(a) \notin In(p\sigma)$, et donc $p\sigma \xrightarrow{\sigma(a):} p\sigma$.

– (SP) $\quad a(x).p + a(x).q = a(x).p + a(x).q + a(x).(\langle x = y\rangle p,q)$

Soit σ une substitution. On peut supposer que $x \notin (prdom(\sigma) \cup prcod(\sigma)$.

Nous devons prouver $(a(x).p + a(x).q)\sigma {\sim_+} (a(x).p + a(x).q + a(x).(\langle x = y\rangle p,q))\sigma$.

Si $(a(x).p + a(x).q)\sigma \xrightarrow{\alpha} q$, évidement $(a(x).p + a(x).q + a(x).(\langle x = y\rangle p,q))\sigma \xrightarrow{\alpha} q$.

Supposons $(a(x).p + a(x).q + a(x).(\langle x = y\rangle p,q))\sigma \xrightarrow{\alpha} q$.

Alors la derniere règle utilisée est la règle (7) ou sa symétrique, et soit $(a(x).p + a(x).q)\sigma \xrightarrow{\alpha} q$, soit $(a(x).(\langle x = y\rangle p,q))\sigma \xrightarrow{\alpha} q$. Le premier cas est trivial. Supposons $(a(x).(\langle x = y\rangle p,q))\sigma \xrightarrow{\alpha} q$. Le seul cas intéressant est $\alpha = \sigma(a)\langle c\rangle$ pour un certain $c \in Ch_b$. Donc $q = \langle c = \sigma(y)\rangle(p\sigma)[c/x],(q\sigma)[c/x]$.

Si $\sigma(y) = c$, alors par l' utilisation des règles (2) et (7), on obtient $(a(x).p + a(x).q)\sigma \xrightarrow{\sigma(a)\langle c\rangle} (p\sigma)[c/x]$, et il est facile à prouver que $\langle c = \sigma(y)\rangle(p\sigma)[c/x],(q\sigma)[c/x] \sim (p\sigma)[c/x]$.

Si $\sigma(y) \neq c$, alors par l' utilisation des règles (2) et (7), on obtient $(a(x).p + a(x).q)\sigma \xrightarrow{\sigma(a)\langle c\rangle} (q\sigma)[c/x]$, et il est facile à prouver que $\langle c = \sigma(y)\rangle(p\sigma)[c/x],(q\sigma)[c/x] \sim (q\sigma)[c/x]$.

$\square$ *Theoreme* 61

Résumé

Les systèmes de processus mobiles sont des systèmes dont la topologie de communication entre les processus peut varier au cours du temps. Dans les langages qui permettent de décrire de tels systèmes, la mobilité est modélisée par un mécanisme de génération dynamique de noms de canaux, plus la possibilité d'insérer (et donc de propager) des noms des canaux dans les messages échangés.

La diffusion est une caractéristique souvent requise des systèmes distribués dont la topologie de communication dépend de l'état des composants à un certain moment pendant l'exécution. Dans cette classe d'applications, un message envoyé par un des composants doit être reçu par tous les participants dont l'état satisfait une certaine propriété P. Le fait que l'ensemble des participants dont l'état satisfait P peut changer au cours du temps implique la réconfigurabilité de la topologie de communication.

Pour raisonner sur de tesl systèmes, on utilise souvent des algèbres des processus. Parmi les algèbres des processus déjà proposées dans la littérature, il n'existe pas de modèle pour les systèmes mobiles communiquant par diffusion. La principale contribution de cette thèse est l'introduction et le développement d'un modèle formel original, le $b\pi$-calcul, intégrant d'une façon cohérente la mobilité et la diffusion.

Après un état de l'art sur quelques modèles formels connus pour les systèmes distribuées (présentés suivant leur pouvoir croissant d'expression: communications point à point vers communications de groupe, structures de communication statiques vers structures de communication dynamiques), nous présentons la syntaxe et la sémantique opérationnelle du $b\pi$-calcul. Nous illustrons le pouvoir d'expression de notre modèle à travers plusieurs exemples.

Ensuite, nous introduisons plusieurs relations d'équivalences qui permettent d'identifier (ou de distinguer) deux processus en fonction de leurs réponses aux interactions avec l'environnement. Nous commençons par les bisimulations: nous présentons trois classes d'équivalences induites par des bisimulations barbelées ou étiquetées, et nous prouvons que les trois classes coïncident pour les processus d'image finie. Nous terminons l'étude des bisimulations par l'analyse des congruences induites, et par une axiomatisation complète des congruences fortes pour les processus finis. Pour les pré-ordres induits par des tests, nous obtenons deux classes de relations: le "must testing" et le "may testing". Nous présentons pour ces relations des caractérisations basées sur des traces.

Enfin, nous nous intéressons aux relations entre les communications point à point et la diffusion. Nous montrons qu'il n'est pas possible d'implanter (sous certaines conditions) des langages à diffusion par des communications point à point.

Mots clés : π - calcul, $b\pi$ - calcul, modèle formel, systèmes distribués, mobilité, diffusion, algèbre de processus, bisimulations, traces, pouvoir d'expression.

Abstract

Systems composed of mobile processes are systems where the communication topology between processes can change in time. In languages which allow to describe such systems, mobility is modeled by a dynamic generation mechanism of channel names combined with the possibility to insert (and hence to propagate) the channels names as part of exchanged messages.

Broadcasting is a characteristic which is often required in distributed systems where the topology of communications depends on the states of components at a certain moment during execution. For such systems, a message sent by one component must be received by all participants whose state satisfies a certain property P. As set of participants whose state satisfies P can change in time, implies the reconfigurability of communication topology.

Process algebras are a good framework to reason about distributed systems. Among process algebras proposed in the literature, there is no model dealing with broadcasting mobile processes. The main contribution of this thesis, is the introduction and development of an original formal model, the $b\pi$-calculus, integrating mobility and broadcast in a coherent manner.

After a survey on some well-known forma models for distributed systems (presented in ascending order of expressiveness: point to point communications versus group communications, static structure of communications versus dynamic structure of communications), we present the syntax and semantics of $b\pi$-calculus. We illustrate the expressiveness of our model by some examples.

Then, we introduce several equivalence relations which allow to identify (or distinguish) two processes depending on their answers to interactions with the environment. We begin by bisimulations: we get three classes of equivalences relations induced by barbed or labeled bisimulations. We prove that all three classes coincide for image finite processes and we give a complete axiomatisation for the induced congruence relations for finite processes. For the preorders induced by tests, we get two classes of relations: the "must testing" and the "may testing". For both relations, we present traces based characterizations.

Finally, we deal with the relation between point to point communications and broadcast. We prove that it is not possible (under some conditions) to implement broadcast based languages by point to point communications.

Keywords: π - calculus, $b\pi$ - calculus, formal model, distributed systems, mobility, broadcast, process algebra, bisimulations, traces, expressiveness.