Axiomatisations and Types for Probabilistic and Mobile Processes

Yuxin Deng

Collège doctoral

THESE

pour obtenir le grade de

Docteur de l'Ecole des Mines de Paris

Spécialité: Informatique Temps réel, Robotique et Automatique

présentée et soutenue publiquement

par

Yuxin DENG

le 22 juillet 2005

Axiomatisations et types pour des processus probabilistes et mobiles

Directeur de thèse : Davide SANGIORGI

Jury

Mme. Delia KESNER	Présidente
M. Matthew HENNESSY	Rapporteur
M. Roberto SEGALA	Rapporteur
M. Roberto DI COSMO	Examinateur
M. Davide SANGIORGI	Examinateur

Résumé

Cette thèse se concentre sur des bases théoriques utiles pour l'analyse d'algorithmes et de protocoles pour des systèmes répartis modernes. Deux caractéristiques importantes des modèles pour ces systèmes sont les probabilités et la mobilité typée : des probabilités peuvent être utilisées pour quantifier des comportements incertains ou imprévisibles, et des types peuvent être utilisés pour garantir des comportements sûrs dans des systèmes mobiles. Dans cette thèse nous développons des techniques algébriques et des techniques basées sur les types pour l'étude comportementale des processus probabilistes et mobiles.

Dans la première partie de la thèse nous étudions la théorie algébrique d'un calcul de processus qui combine les comportements non-déterministe et probabiliste dans le modèle des automates probabilistes proposés par Segala et Lynch. Nous considérons diverses équivalences comportementales fortes et faibles, et nous fournissons des axiomatisations complètes pour des processus à états finis, limitées à la récursion gardée dans le cas des équivalences faibles.

Dans la deuxième partie de la thèse nous étudions la théorie algébrique du π-calcul en présence des types de capacités, qui sont très utiles dans les calculs de processus mobiles. Les types de capacités distinguent la capacité de lire sur un canal, la capacité d'écrire sur un canal, et la capacité de lire et d'écrire à la fois. Ils introduisent également une relation de sous-typage naturelle et puissante. Nous considérons deux variantes de la bisimilarité typée, dans leurs versions retardées et anticipées. Pour les deux variantes, nous donnons des axiomatisations complètes pour les termes fermés. Pour une des deux variantes, nous fournissons une axiomatisation complète pour tous les termes finis.

Dans la dernière partie de la thèse nous développons des techniques basées sur les types pour vérifier la propriété de terminaison de certains processus mobiles. Nous fournissons quatre systèmes de types pour garantir cette propriété. Les systèmes de types sont obtenus par des améliorations successives des types du π-calcul simplement typé. Les preuves de terminaison utilisent des techniques employées dans les systèmes de réécriture. Ces systèmes de types peuvent être utilisés pour raisonner sur le comportement de terminaison de quelques exemples non triviaux : les codages des fonctions récursives primitives, le protocole pour coder le choix séparé en terme de composition parallèle, une table de symboles implementée comme une chaîne dynamique de cellules.

Ces résultats établissent des bases pour une future étude de modèles plus avancés qui peuvent combiner des probabilités avec des types. Ils soulignent également la robustesse des techniques algébriques et de celles basées sur les types pour le raisonnement comportemental.

Abstract

The focus of this thesis are the theoretical foundations for reasoning about algorithms and protocols for modern distributed systems. Two important features of models for these systems are probability and typed mobility: probabilities can be used to quantify unreliable or unpredictable behaviour and types can be used to guarantee secure behaviour in systems with a mobile structure. In this thesis we develop algebraic and type-based techniques for behavioural reasoning on probabilistic and mobile processes.

In the first part of the thesis we study the algebraic theory of a process calculus which combines both nondeterministic and probabilistic behaviour in the style of Segala and Lynch's probabilistic automata. We consider various strong and weak behavioural equivalences, and we provide complete axiomatisations for finite-state processes, restricted to guarded recursion in the case of the weak equivalences.

In the second part of the thesis we investigate the algebraic theory of the π-calculus under the effect of capability types, which are one of the most useful forms of types in mobile process calculi. Capability types allow one to distinguish between the capability to read from a channel, to write to a channel, and to both read and write. They also give rise to a natural and powerful subtyping relation. We consider two variants of typed bisimilarity, both in their late and in their early version. For both of them, we give complete axiomatisations on the closed finite terms. For one of the two variants, we provide a complete axiomatisation for the open finite terms.

In the last part of the thesis we develop a type-based technique for verifying the termination property of some mobile processes. We provide four type systems to guarantee this property. The type systems are obtained by successive refinements of the types of the simply typed π-calculus. The termination proofs take advantage of techniques from term rewriting systems. These type systems can be used for reasoning about the terminating behaviour of some non-trivial examples: the encodings of primitive recursive functions, the protocol for encoding separate choice in terms of parallel composition, a symbol table implemented as a dynamic chain of cells.

These results lay out the foundations for further study of more advanced models which may combine probabilities with types. They also highlight the robustness of the algebraic and type-based techniques for behavioural reasoning.

To my parents

Acknowledgements

I would like to express my gratitude to Davide Sangiorgi, my supervisor, for his inspiration, guidance, and encouragement. He was always willing to discuss the problems that I encountered in my research and my life. From him I have received invaluable help and advice.

I am very indebted to Catuscia Palamidessi. Her intelligence and enthusiasm had a substantial influence on my research interests in the later period of my Ph.D. study. I have learned much from her about the ways of doing research and the style of presenting it.

I owe a lot to Pierre-Louis Curien for having received me in his laboratory PPS and for having made insightful comments on each piece of work that I have done. Without his generosity and unconditional support, this thesis would not have been possible.

I am also very grateful to Yuxi Fu, my Master's thesis supervisor, for having introduced me to the field of process algebra.

PPS has offered creative and pleasant working atmosphere. I would like to thank all the past and current members for their friendship and interesting discussions. In particular, I thank Samuel Hym, Vincent Balat, and Fabien Tarissan for their help of correcting the resume in French.

I have the pleasure of having stayed three months in the MIMOSA project of INRIA Sophia-Antipolis. Many thanks must go to Gérard Boudol and Ilaria Castellani for having provided the friendly environment.

I appreciate the stimulating discussions with the members of COMETE and PARSIFAL projects of INRIA Futurs. I thank particularly my colleagues Jun Pang and Tom Chothia for the nice collaboration that we had.

I would also like to thank all my friends. They have made my time in Paris both fruitful and enjoyable.

My special gratitude goes to my family, for their unfailing support.

The EU project PROFUNDIS has funded this research. The Department of Computer Science in University of Bologna has sponsored me for two productive trips to Bologna.

Main Notations

Below are the important notations used in this thesis, with the section number of their first appearance.

Metavariables		
$u, v, ...$	names	2.1
ℓ	labels	2.1
α, β	actions	2.2.2
$X, Y, ...$	process variables	2.1
$E, F, ...$	process expressions	2.1
$P, Q, ...$	π-calculus processes	2.2.2
ι	sorts	2.2.3
p, q, r	probabilities	3.1
η, θ	discrete probability distributions	3.1
$\mathcal{R}, \mathcal{S}$	relations	3.3
Γ, Δ	type environments	2.2.5
S, T	types	2.2.5
Miscellaneous symbols		
`bool`	boolean type	2.2.5
`Nat`	natural number type	2.2.5
$\sharp T$	channel type	2.2.5
$\sharp^n T$	channel type with level	5.2
$\mathtt{i}T,\ \mathtt{o}S,\ \mathtt{b}\langle T, S\rangle$	capability types	4.1.1
$fpv(E)$	free process variables	2.1
$\{\widetilde{F}/\widetilde{X}\}$	substitution of expressions	2.1
$\{\widetilde{v}/\widetilde{u}\}$	substitution of names	2.2.2
$fn(\cdot)$	free names of specified entities	2.2.2
$bn(\cdot)$	bound names of specified entities	2.2.2
$subj(\alpha)$	subject of action	2.2.2
$obj(\alpha)$	object of action	2.2.2
$\Delta \sharp P$	configuration	4.1.2
Process constructions		

$\mathsf{0}$	inaction	2.1
$\ell.E$	prefix	2.1
$E + F$	nondeterministic choice	2.1
$E \mid F$	parallel composition	2.1
$\nu a E$	restriction	2.1
$\mu_X E$	recursion	2.1
$\sum_{i\in 1..m} E_i$	indexed nondeterministic choice	3.2
$\bigoplus_{i\in 1..n} p_i \ell_i.E_i$	probabilistic choice	3.2
$u(x).P$	input prefix	2.2.2
$\bar{u}v.P$	output prefix	2.2.2
$!u(x).P$	replicated input	2.2.2
$\texttt{if}\ w\ \texttt{then}\ P\ \texttt{else}\ Q$	if-then-else	5.3.1
φPQ	condition	4.1.1

Transitions

$\xrightarrow{\ell}$	labelled transition	2.1
$\rightarrow$	strong probabilistic transition	3.2
$\rightarrow_c$	strong combined transition	3.2
$\Rightarrow$	weak probabilistic transition	3.2
$\stackrel{\epsilon}{\Rightarrow}_c$	weak combined transition	3.2
$\Rightarrow_c$	normal weak combined transition	3.2

Equivalences

$\equiv_{\mathcal{R}}$	equivalences of distributions	3.3.1
$\sim$	strong bisimilarity	3.3.2
$\sim_c$	strong probabilistic bisimilarity	3.3.2
$\approx$	weak probabilistic bisimilarity	3.3.2
$\simeq$	observational equivalence	3.3.2
$\eqsim$	divergency-sensitive equivalence	3.3.2
$\mathrel{-\!\!\!=}$	typed bisimilarity	4.1.3
$\mathrel{-\!\!\!=}^e$	typed early bisimilarity	4.1.3
$\mathrel{\Rightarrow\!\!\!=}$	a variant typed bisimilarity	4.4.1

Contents

Résumé en français

L'informatique vise à expliquer d'une manière rigoureuse comment les systèmes informatiques se comportent. Actuellement la notion de système informatique inclut non seulement des *systèmes séquentiels*, comme des programmes simples dans des ordinateurs isolés, mais également des *systèmes parallèles*, comme des réseaux informatiques, et même des protéines en biologie et des particules en physique. Les modèles mathématiques classiques (par exemple le λ-*calcul* [Bar84]), malgré leur succès pour décrire des systèmes séquentiels, demeurent insuffisants pour raisonner sur des systèmes parallèles.

Dans les années 80 les *calculs de processus* (parfois appelés *algèbres de processus*), notamment CCS [Mil89a], CSP [Hoa85] et ACP [BK84, BW90], ont été proposés pour décrire et analyser des systèmes parallèles. Tous ont été conçus autour de l'idée centrale d'*interaction* ou de *communication* entre processus. Dans ces formalismes, un système complexe est construit à partir de ses sous-composants, par un petit ensemble d'opérateurs primitifs comme le *préfixe*, le *choix non-déterministe*, la *restriction*, la *composition parallèle* et la *récursion*. La limitation de ces algèbres traditionnelles est qu'elles ne peuvent pas être utilisées pour décrire efficacement des *systèmes mobiles*, c'est-à-dire des systèmes dont la topologie des liaisons change dynamiquement. Sur la base de CCS, Milner, Parrow et Walker ont inventé le π-*calcul* [MPW92], qui réalise la mobilité par un mécanisme où un nom reçu sur un canal peut être lui-même utilisé comme un nom de canal en émission ou en réception. Le π-calcul est un formalisme très expressif. Il permet d'encoder des structures de données [Mil91], le λ-calcul [Mil92] et les communications d'ordre supérieur (lorsque des processus sont transmis à la place des noms) [San93]. En outre, il peut être utilisé comme un outil de raisonnement sur des *langages orientés objet* [Wal95].

Comme aucune théorie n'atteindra tous les objectifs, un grand nombre de variantes et d'extensions des calculs de processus classiques sont parues dans la littérature. Grossièrement ils peuvent être regroupés en trois catégories en fonction des intentions des concepteurs.

- Pour mieux capturer quelques caractéristiques spécifiques des systèmes parallèles comme les communications asynchrones, les communications d'ordre supérieur, les localités et les migrations. On peut faire une longue liste d'exemples de calculs faits dans ce but : le π-calcul asynchrone [HT91, Bou92], le πI-calcul [San96a], le Lπ-calcul [Mer00], le calcul Fusion [PV98], le χ-calcul [Fu99], le calcul Join [Fou98], CHOCS [Tho95], HOπ [San93], Dπ [HR02b], Klaim [DFP98], le calcul des Ambients Mobiles [CG00] et ses variantes, pour en citer juste quelques uns.

- Pour équiper les processus mobiles de types, de sorte que les processus interagissent entre eux d'une manière plus sûre et plus efficace. Par exemple, un certain nombre de systèmes de types ont été conçus pour le π-calcul ; ils sont utilisés dans diverses applications comme la détection statique des erreurs dans les programmes parallèles [Mil91], les optimisations de compilateur [KPT99], le contrôle d'accès de ressources [PS96, HR02b]. En plus, ils garantissent d'autres propriétés de sécurité comme l'exécution sans blocage [Kob98], la non-intervention [HY05] et la terminaison [YBH04, DS04a].

- Pour soutenir le raisonnement sur les comportements probabilistes qui existent, par exemple, dans les systèmes aléatoires, répartis et résistants aux pannes. L'approche générale que l'on adopte est d'étendre avec des probabilités les modèles et les techniques existants qui ont déjà été couronnés de succès dans les cadres non-probabilistes. La caractéristique commune des calculs de processus probabilistes est l'existence de l'opérateur de *choix probabiliste* ; voir par exemple des extensions probabilistes de CCS [GJS90, HJ90, Tof94, YL92], CSP probabiliste [Low91], ACP probabiliste [And99] et le π-calcul asynchrone probabiliste [HP04].

Dans cette thèse nous illustrerons les calculs de processus des deuxième et troisième catégories en détail.

Afin d'étudier un langage de programmation ou un calcul de processus, on doit fournir une signification cohérente à chaque programme ou processus de ce langage. Cette signification est la *sémantique* du langage ou du calcul. La sémantique est utile pour vérifier ou montrer que les programmes se comportent comme prévu. D'une manière générale, il y a trois approches principales pour donner des sémantiques à un langage de programmation. L'approche *dénotationnelle* cherche une fonction d'évaluation qui associe à un programme sa signification mathématique. Cette approche réussit à modéliser beaucoup de langages séquentiels ; un programme est interprété comme une fonction du domaine des valeurs d'entrée vers le domaine des valeurs de sortie. Cependant, jusqu'ici l'interprétation dénotationnelle des programmes parallèles n'est pas aussi satisfaisante que le traitement dénotationnel des programmes séquentiels.

L'approche *opérationnelle* s'avère très utile pour donner des sémantiques aux systèmes parallèles. Le comportement d'un processus est indiqué par sa *sémantique opérationnelle structurelle* [Plo81], décrite par un ensemble de règles de transitions étiquetées inductivement définies sur la structure des termes. De cette façon chaque processus correspond à un *graphe de transitions étiquetées*. La limitation de la sémantique opérationnelle est qu'elle est trop concrète, car un graphe de transitions peut contenir beaucoup d'états qui devraient intuitivement être confondus. On a alors proposé beaucoup d'équivalences pour comparer les différents graphes de transitions.

L'approche *axiomatique* vise à comprendre un langage par quelques axiomes et règles d'inférence. Son importance est motivée, entre autres, par les deux raisons suivantes.

- Les systèmes corrects, même s'ils ne sont pas complets, peuvent être utiles pour la manipulation des termes par un humain ou par des machines. En exploitant ces systèmes, un certain nombre de problèmes pratiques de vérification peuvent être abordés.

- Les systèmes complets aident à comprendre la nature des équivalences. Par exemple, la différence entre deux équivalences peut être caractérisée par quelques axiomes, en partic-

ulier si en ajoutant ces axiomes à un système complet pour une équivalence on obtient un système complet pour l'autre équivalence. Une autre méthode de comparaison est de fixer une équivalence et de changer les expressions. Parfois on étend le système complet d'un sous-langage au langage entier, en ajoutant quelques axiomes supplémentaires. Comme nous le verrons plus tard, les deux phénomènes se produisent aux chapitres 3 et 4.

Dans les calculs de processus, un sujet important et toujours actif est d'explorer la connexion entre les sémantiques opérationnelles et axiomatiques. Milner [Mil78] a été le premier à préconiser le développement d'une algèbre des comportements qui obéit à un certain nombre d'axiomes exprimés par des équations. Dans [Mil80] un lien direct est établi pour la première fois entre une théorie algébrique et une équivalence comportementale basée sur une sémantique opérationnelle. Depuis, un grand nombre de travaux portent sur les théories algébriques de processus, pour différentes équivalences comportementales et dans divers calculs de processus. Cependant, on ne voit pas beaucoup d'attention prêtée aux calculs de processus probabilistes et typés, bien qu'ils s'avèrent être très utiles dans l'analyse des systèmes répartis modernes.

Objectifs

Cette thèse se concentre sur des bases théoriques utiles pour l'analyse d'algorithmes et de protocoles pour des systèmes répartis modernes. Nous pensons que ce genre de raisonnement est important parce que si un système est établi sans analyse rigoureuse de toutes les interactions possibles entre ses composants, alors son comportement est souvent incorrect. En est témoin la découverte récente des défauts de sécurité dans les protocoles de transmission sans fil comme IEEE 802,11 et Bluetooth [BGW01, LL03].

Dans les systèmes répartis il est intéressant de considérer des modèles qui incluent des probabilités. Une raison est qu'on espère que ces systèmes fournissent des services fiables en dépit de l'occurrence de divers échecs. Les processus probabilistes peuvent être utilisés pour décrire des systèmes résistants aux pannes. Par exemple, l'information probabiliste peut être utilisée pour indiquer le taux de perte des messages par les canaux de transmission défectueux. En plus, les modèles probabilistes peuvent être utilisés pour casser la symétrie dans des problèmes de coordinations distribuées (par exemple, le problème des philosophes, le problème d'élection de chef, et le problème de consensus), pour prévoir le comportement de systèmes basés sur le calcul des caractéristiques d'exécution, et pour représenter et mesurer d'autres formes d'incertitude.

Un modèle pour les systèmes répartis devrait également inclure la caractéristique de mobilité. Les systèmes physiques tendent à avoir une structure fixe. Mais la plupart des systèmes dans le monde de l'information ne sont pas physiques car leurs liens peuvent être symboliques ou virtuels. Par exemple, quand on clique sur un lien hypertexte dans une page web, un lien symbolique est créé entre la machine et le serveur web à distance. Un exemple de lien virtuel est une connexion radio, comme les liens entre les téléphones mobiles et un réseau de stations de base. Cette connexion radio, avec des liens transitoires, a une structure mobile.

Avec la mobilité, les types s'avèrent être essentiels. Par exemple, la théorie du π-calcul non typé est souvent insuffisante pour prouver des propriétés comportementales sur les processus. La raison

est que quand on utilise le π-calcul pour décrire un système, on suit normalement une méthode qui détermine comment utiliser des noms. Mais cette méthode n'est pas explicite dans les processus et elle ne peut donc pas jouer un rôle dans les preuves. Des types peuvent être utilisés pour rendre une telle méthode explicite (cf. Partie IV de [SW01]). En outre, les types sont utiles pour exprimer le contrôle de l'intervention, du droit d'accès, du déclassement robuste, de la composition sûre des composants, et de la limite des consommations de ressources (par exemple, des allocations de temps ou de mémoire).

Il y a une motivation pratique pour considérer les probabilités et la mobilité en même temps. Comment un système de téléphone mobile peut-il s'exécuter de façon satisfaisante si le concepteur ne considère jamais le comportement probable des utilisateurs ? Un certain nombre de modèles probabilistes ont été présentés en tant que variantes des chaînes de Markov, mais pour la mobilité ils sont peu développés.

Dans la littérature, les probabilités et la mobilité typée sont souvent étudiées séparément. Des techniques opérationnelles ont été développées, mais très peu d'efforts ont été faits sur des techniques algébriques. Cependant, elles sont très utiles en informatique. Par exemple, dans le modèle relationnel pour les bases de données [Cod70], les lois algébriques ont servi à l'optimisation de demande [RG02]. Dans les calculs de processus, des équations algébriques peuvent être considérées en tant que règles de réécriture pour la manipulation automatisée de termes [vdP01].

Dans cette thèse nous étudions des techniques algébriques en considérant l'impact de la mobilité, des probabilités et des types sur les théories algébriques des calculs de processus. Puisque chaque caractéristique présente de nouveaux problèmes non triviaux, il est difficile de développer d'emblée des techniques algébriques pour des modèles basés sur la mobilité typée *et* les probabilités. Par conséquent, il vaut mieux les étudier d'abord séparément. Dans le chapitre 3 nous considérons donc des axiomatisations pour un calcul probabiliste sans mobilité, et dans le chapitre 4 nous fournissons des axiomatisations pour un calcul de processus mobile typé sans probabilités. Les types que nous utilisons sont les *types de capacité* [PS96], qui distinguent la capacité de lire sur un canal, la capacité d'écrire sur un canal, et la capacité de lire et d'écrire à la fois. Ce genre de types sont utiles et fondamentaux pour les calculs de processus. Ils ont été utilisés pour garantir l'échange de données cohérentes sur des canaux, et pour contrôler des droits d'accès aux canaux. Des variantes des types de capacités sont maintenant présentes dans presque tous les calculs de processus. Parfois, elles deviennent une partie de la syntaxe, par exemple dans le Lπ-calcul et le calcul Join, seules les capacités d'écrire peuvent être transmises.

Dans les calculs de processus mobiles, les types peuvent être utilisés comme une technique de vérification pour analyser diverses propriétés des programmes concurrents, comme l'exécution sans blocage [Kob98], l'exécution sans attente active [Kob00], et le flux d'information [HVY00, HR02a]. Dans le chapitre 5 nous développons une telle technique pour le problème de terminaison, qui est une propriété importante que beaucoup d'algorithmes et protocoles dans les systèmes répartis doivent garantir. Dans le cas des systèmes répartis symétriques, les algorithmes probabilistes sont souvent plus efficaces que les algorithmes déterministes, au prix que certaines propriétés se produiront avec la probabilité 1 mais pas nécessairement avec certitude. Pour tous les buts pratiques, cependant, cette différence est insignifiante. Par conséquent, il est intéressant de parler de la terminaison probabiliste

aussi. Cependant, puisque la terminaison est elle-même un problème non trivial, nous considérons des types sans probabilité.

Pour récapituler, dans cette thèse nous développons des techniques algébriques et des techniques basées sur les types pour raisonner sur les processus avec probabilités et mobilité typée. Nous considérons ces deux caractéristiques séparément, à la fois dans le cas des axiomatisations et celui de la terminaison, mais nous croyons que notre travail contribue à établir des bases pour étudier des modèles plus avancés qui peuvent combiner les probabilités avec la mobilité typée.

Avant de discuter dans les sections suivantes des motivations pour chaque sujet de la thèse, nous devons présenter une certaine terminologie. Nous utilisons le concept général *axiomatisations* pour désigner à la fois des systèmes d'axiomes et des systèmes de preuves. Pour une équivalence sur un ensemble de termes, un *système d'axiomes* se compose de quelques axiomes équationnels et des règles du raisonnement équationnel (c'est-à-dire, les règles de réfléxivité, de symétrie, de transitivité, et les règles de congruence qui permettent de remplacer n'importe quel sous-terme d'un processus par un terme équivalent). Un *système de preuves* a, en plus des axiomes et de certaines règles du raisonnement équationnel, d'autres règles d'inférence. Généralement un système d'axiomes est préférable à un système de preuves, parce que, par exemple, les techniques générales de la réécriture de termes peuvent alors être applicables. Cependant, quand le calcul de processus en question inclut des caractéristiques non triviales comme la récursion ou les types, parfois il est difficile d'obtenir un système d'axiomes qui est complet parce que nous devons utiliser d'autres règles d'inférence, c'est-à-dire, ce que nous obtenons est réellement un système de preuves. Dans ce cas nous appelons aussi ce système une axiomatisation, comme on l'a fait dans la littérature [Mil89b, Par01]. Pour une axiomatisation, la *complétude* signifie que si deux processus montrent un comportement semblable, c'est-à-dire, leurs graphes de transition sont équivalents, alors on peut prouver qu'ils sont égaux dans un système d'axiomes ou un système de preuves; la *correction* signifie l'inverse.

Axiomatisations pour les processus probabilistes

La dernière décennie a été témoin de l'intérêt croissant dans le domaine des méthodes formelles pour la spécification et l'analyse des systèmes probabilistes [Seg95, BH97, AB01, PLS00, Sto02, CS02]. Dans [vGSS95] van Glabbeek *et al.* ont classifié les modèles probabilistes dans trois catégories : les modèles *réactifs*, les modèles *génératifs* et les modèles *stratifiés*. Dans les modèles réactifs, une probabilité est associée à chaque transition étiquetée, et pour chaque état la somme des probabilités de ses transitions avec la même étiquette est 1. Les modèles génératifs diffèrent des modèles réactifs parce que pour chaque état la somme des probabilités de toutes les transitions sortantes est 1. Les modèles stratifiés ont plus de structure et pour chaque état soit il y a exactement une transition étiquetée sortante soit il y a seulement des transitions non étiquetées et la somme de leurs probabilités est 1.

Dans [Seg95] Segala a indiqué que ni les modèles réactifs ni les modèles génératifs ni les modèles stratifiés ne capturent le vrai non-déterminisme, une notion essentielle pour modéliser la liberté d'ordonnancement, la liberté d'implémentation, l'environnement externe et l'information incomplète. Il a donc présenté une sorte de modèles, les *automates probabilistes* (PA), où les probabilités et le

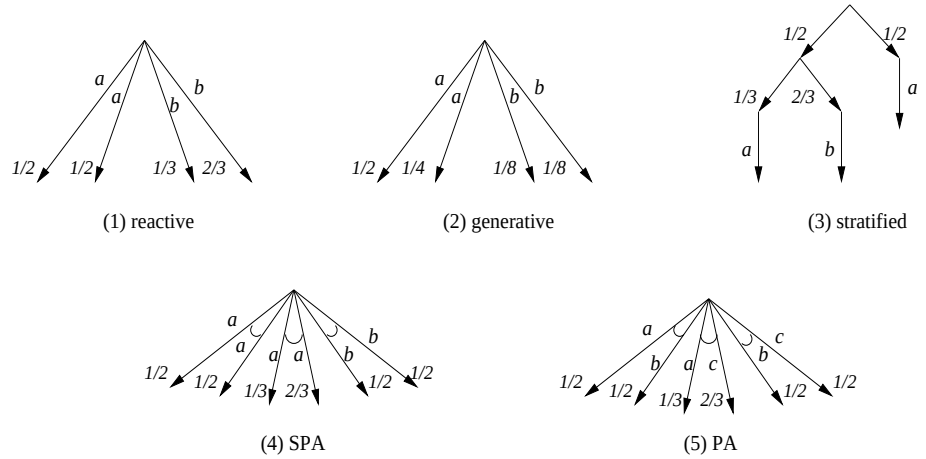

Figure 1: Modèles probabilistes

non-déterminisme sont tous deux pris en considération. Le choix probabiliste est exprimé par la notion de *transition*, qui, dans les PA, mène à une distribution probabiliste sur des paires (action, état) et des impasses (c'est-à-dire, des états qui n'ont pas de transitions sortantes). Le choix non-déterministe, par contre, est exprimé par la possibilité de choisir différentes transitions. Segala a proposé également une version simplifiée de PA appelée *automates probabilistes simples* (SPA), qui sont comme les automates ordinaires mais sont tels qu'une transition étiquetée mène à une distribution probabiliste sur un ensemble d'états au lieu d'un seul état.

La figure 1 donne un exemple des modèles probabilistes discutés ci-dessus. Dans les modèles où les probabilités et le non-déterminisme sont présents à la fois, comme ceux des diagrammes (4) et (5), une transition est représentée comme un paquet de flèches qui sont liées par un petit arc. [SdV04] fournit une comparaison détaillée entre les différents modèles, et montre dans un certain sens que les PA subsument tous les autres modèles ci-dessus sauf les modèles stratifiés.

Nous étudierons au chapitre 3 les systèmes d'axiomes pour un calcul de processus basé sur les PA, dans le sens où la sémantique opérationnelle de chaque expression du langage est un automate[1]. Les systèmes d'axiomes sont très importants car au niveau théorique, ils aident à comprendre le calcul et à établir ses bases, et au niveau pratique, ils peuvent être utilisés comme un outil intéressant de spécification et de vérification des systèmes. Notre calcul est essentiellement une version probabiliste du calcul employé par Milner pour exprimer les comportements d'états finis [Mil84, Mil89b].

Nous considérerons deux équivalences fortes, une équivalence faible qui est commune dans la littérature, ainsi qu'une notion d'équivalence faible ayant l'avantage d'être sensible à la divergence. Pour les expressions sans récursion nous fournissons des axiomatisations complètes des quatre équivalences. Pour les équivalences fortes nous donnons également des axiomatisations complètes pour toutes les expressions, alors que pour les équivalences faibles nous obtenons ce résultat seulement pour les expressions gardées.

La raison pour laquelle nous sommes intéressés par l'étude d'un modèle qui exprime le com-

[1]sauf le cas du blocage qui est traité légèrement différemment : en suivant la tradition des calculs de processus, dans notre cas le blocage est un état, mais dans les PA il est un des composants possibles d'une transition.

portement non-déterministe et probabiliste, et d'une équivalence sensible à la divergence, est qu'un des buts à long terme de cette ligne de recherche est de développer une théorie qui nous permettra de raisonner sur des algorithmes probabilistes utilisés dans des systèmes répartis. Dans ce domaine il est important d'assurer qu'un algorithme fonctionne sous n'importe quel ordonnanceur, et sous d'autres facteurs inconnus ou incontrôlables. Le composant non-déterministe de notre calcul nous permet de traiter toutes ces conditions d'une manière uniforme et élégante. En outre, dans beaucoup d'applications des systèmes répartis il est important d'assurer l'exécution sans attente active, et donc nous aurons besoin d'une sémantique qui n'ignore pas la divergence.

Nous finissons cette section par une discussion au sujet de certains travaux voisins dans cette direction de recherche. Dans [Mil84] et [Mil89b] Milner a donné des axiomatisations complètes pour la bisimilarité forte et l'équivalence observationnelle, respectivement, dans le cadre d'un noyau de *CCS* [Mil89a]. Ces deux articles nous servent de point de départ : dans plusieurs preuves de complétude qui comportent la récursion nous adoptons deux théorèmes de Milner : le *théorème de caractérisation équationnelle* et le *théorème de solution unique*. Dans les section 3.4.1 et 3.5.2 nous étendons [Mil84] et [Mil89b] (pour les expressions gardées) respectivement, dans le cadre de l'algèbre de processus probabiliste.

Dans [SS00] Stark et Smolka ont donné une version probabiliste des résultats de [Mil84]. Nous étendons donc les résultats de [SS00] parce que nous considérons également le non-déterminisme. Quand le choix non-déterministe est ajouté, la technique de Stark et Smolka pour prouver la correction des axiomes n'est plus utilisable (voir la discussion à l'annexe A.2.) La même remarque s'applique également à [AÉI02] qui suit l'approche de [SS00] mais utilise quelques axiomes d'algèbre d'itération pour caractériser la récursion. En revanche, notre version probabiliste de la technique "bisimulation up to" [Mil89a] marche bien avec la technique ordinaire de l'induction sur les transitions.

Dans [BS01] Bandini et Segala ont donné les axiomatisations des équivalences comportementales fortes et faibles pour les calculs de processus correspondant aux SPA et à une version de SPA pourvue d'une sémantique alternative. Puisque leur calcul de processus avec la sémantique non-alternative correspond aux SPA, nos résultats de la section 3.6 peuvent être considérés comme une extension de leurs travaux aux PA.

Pour l'algèbre de processus probabiliste de style ACP, plusieurs systèmes complets d'axiomes sont apparus dans la littérature. Cependant, dans chacun de ces systèmes soit la bisimilarité faible n'est pas étudiée [BBS95, And99], soit le choix non-déterministe est supprimé [BBS95, AB01].

Axiomatisations pour les processus mobiles typés

La théorie du π-calcul a été profondément étudiée [Mil99, SW01], et deux thèmes majeurs y sont la théorie algébrique et les systèmes de types. La majeure partie de la théorie algébrique a été développée sur le calcul non typé ; les résultats incluent les axiomatisations qui sont corrects et complets sur les processus finis pour les équivalences comportementales principales : les bisimilarités retardées et anticipées, les congruences retardées et anticipées [PS95, Lin94, Lin03], la bisimilarité ouverte [San96b], l'équivalence de test [BD95]. Une grande partie de la recherche sur les types

s'est concentrée sur leurs effets comportementaux. Par exemple, on a proposé des variantes des équivalences comportementales standards afin de tenir compte des types [PS96, SW01].

Nous étudierons au chapitre 4 l'impact des types sur la théorie algébrique du π-calcul. Plus précisément, nous étudions des axiomatisations du π-calcul typé. Bien que quelques lois algébriques pour les calculs typés de processus mobiles aient été considérées dans la littérature [SW01], nous n'avons vu aucune axiomatisation.

Le système de types que nous considérons a des types de capacités (parfois appelés les types I/O) [PS96, HR02b]. Ces types nous permettent de distinguer, par exemple, la capacité d'utiliser un canal pour lire des noms de la capacité d'utiliser le canal pour écrire des noms. Un type montre la capacité d'un canal et, en plus, les capacités des canaux portés par ce canal. Par exemple, le type $a : \mathtt{iob}T$ (pour une expression appropriée de types) indique que le canal a peut être utilisé seulement pour lire des noms ; et n'importe quel canal lu sur a peut être utilisé seulement pour écrire des canaux qui ont la capacité d'écrire et de lire des noms de type T. Alors, le processus $a(x).\bar{x}b.b(y).\bar{b}y$ est bien typé dans l'environnement de typage $a : \mathtt{iob}T, b : \mathtt{b}T$. Rappelons que $\bar{a}b.P$ désigne un processus qui veut écrire le nom b sur le canal a, puis continuer son exécution P ; $a(x).P$ désigne un processus qui veut lire un nom sur le canal a, puis reprendre son exécution P, où les occurrences libres de x ont été remplacées par le nom que l'on a lu.

Dans les calculs pour la mobilité, les types de capacités sont devenus les types les plus utiles, et dont les effets comportementaux sont les plus connus. Les capacités sont utiles pour protéger des ressources ; par exemple, dans un modèle de client/serveur, elles peuvent être utilisées pour empêcher un client de saisir le canal d'accès au serveur en lecture et de voler des messages au serveur ; d'une façon similaire, elles peuvent être utilisées dans la programmation répartie pour exprimer des contraintes de sécurité [HR02b]. Les capacités introduisent la relation de *sous-typage* : les capacités d'écrire sont contravariantes, tandis que les capacités de lire sont covariantes. Par exemple, nous montrons une relation de sous-typage à la figure 2, où une flèche indique la relation de sous-typage. Il y a trois formes de types pour les noms de canaux : $\mathtt{i}T$, $\mathtt{o}S$ et $\mathtt{b}\langle T, S\rangle$, elles donnent aux noms les capacités de lire des valeurs du type T, d'écrire des valeurs du type S, ou de faire les deux. Nous notons $\mathtt{b}T$ comme l'abréviation de $\mathtt{b}\langle T, T\rangle$. La profondeur de l'imbrication des capacités est 1 pour tous les types dans le diagramme (a), et 2 pour tous les types dans le diagramme (b) (les définitions formelles des types et de la relation de sous-typage seront données à la section 4.1.1). Le sous-typage est utile en particulier quand le π-calcul est exploité pour la programmation orientée objet, ou pour donner une sémantique aux langages orientés objet.

Pour voir pourquoi l'addition des types de capacités a des conséquences sémantiques, considérons

$$P \stackrel{\text{def}}{=} \nu c\, \overline{b}c.a(y).(\overline{y} \mid c) \qquad Q \stackrel{\text{def}}{=} \nu c\, \overline{b}c.a(y).(\overline{y}.c + c.\overline{y}).$$

Ces processus ne sont pas comportementalement équivalents en π-calcul non typé. Par exemple, si le canal lu sur a est c, alors P peut se terminer après 2 interactions avec l'observateur externe. En revanche, Q se termine toujours après 4 interactions avec l'observateur. Cependant, si nous imposons la condition que seulement la capacité de lire des canaux peut être transmise sur b, alors P et Q montrent le même comportement dans n'importe quel contexte bien typé. Par exemple, puisque l'observateur reçoit seulement la capacité de lire des noms sur c, il ne peut pas écrire c

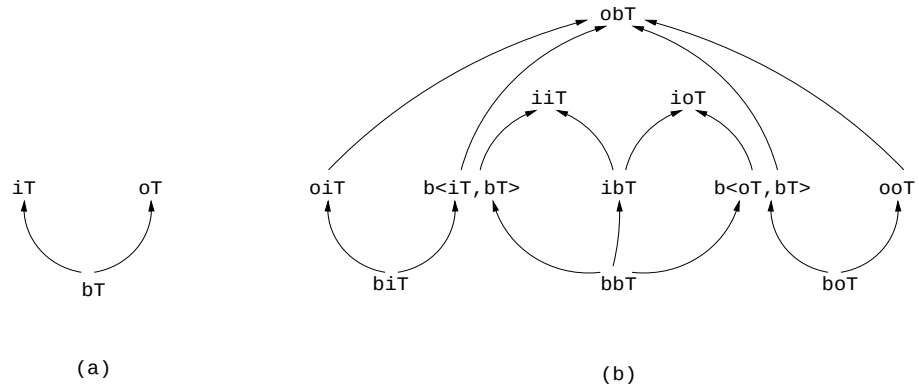

Figure 2: Un exemple de la relation de sous-typage, où $T = \texttt{unit}$

sur a : les canaux écrits sur a exigent au moins la capacité d'écrire (cf. l'occurrence de $\overline{y}$). Par conséquent, dans le cas typé, les processus sont comparés par un observateur avec certaines capacités (c'est-à-dire, types sur des canaux). Si l'on dénote ces capacités par Δ, alors la bisimilarité typée entre P et Q est écrite $P \mathrel{\dot{=}}_{\Delta} Q$.

En π-calcul non typé, les systèmes de transitions étiquetés (LTS pour *labelled transition systems*) sont définis sur des processus ; la transition $P \xrightarrow{\alpha} P'$ signifie que P peut accomplir l'action α et puis devenir P'. En π-calcul typé, les informations sur les capacités de l'observateur sont pertinentes parce que l'observateur ne peut interroger des processus que par des interactions pour lesquelles il a toutes les capacités nécessaires. Par conséquent les systèmes de transitions étiquetés typés (TLTS pour *typed labelled transition systems*) sont définis sur des configurations, et une configuration $\Delta \sharp P$ se compose d'un processus P et des capacités Δ (parfois nous appelons l'observateur Δ l'environnement externe). Maintenant une transition $\Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P'$ signifie que P devient P' après avoir accompli une action α permise par l'environnement Δ, qui se transforme en Δ' par ailleurs.

Une version de types de capacités a été présentée dans [PS96]. Et depuis on a proposé un certain nombre de variantes et d'extensions. Nous suivons le système de Hennessy et Riely [HR02b], dans lequel, au contraire du système dans [PS96] : (i) il existe deux opérations partielles sur les types (*meet* et *join*) ; (ii) la règle de typage pour la construction *comparaison* (la construction utilisée pour tester l'égalité entre deux noms) est très libérale, parce qu'elle peut être appliquée aux canaux de n'importe quel type (dans [PS96] deux canaux peuvent être comparés s'il possèdent la capacité de lire et la capacité d'écrire à la fois). Tandis que (i) simplifie seulement certains détails techniques, (ii) semble essentiel. En effet, l'importance de la comparaison pour la théorie algébrique du π-calcul est bien connue (c'est la raison principale de l'existence de la comparaison dans le calcul non typé).

La bisimilarité typée et l'utilisation des configurations pour définir la bisimilarité typée ont été présentées dans [BS98]. Nous suivons une de ses variantes proposée par Hennessy et Rathke [HR04], parce qu'elle emploie le système de types de [HR02b] et inclut la construction de comparaison.

Deux résultats importants que nous avons obtenus sont un système de preuve et un système d'axiomes pour la bisimilarité typée ($\dot{=}$). Le système de preuve a une preuve de correction simple mais il marche seulement pour les termes fermés. Le système d'axiomes traite tous les termes finis.

Notre bisimilarité $\simeq$ est une variante de celle de [HR04]. Pour la bisimilarité typée de [HR04] nous fournissons un système de preuve pour les termes fermés, et une axiomatisation indirecte pour tous les termes parce qu'elle exploite le système de $\simeq$. Nous n'avons pas pu donner une axiomatisation directe qui ne dépend pas du système de $\simeq$: les difficultés principales sont discutées à la section 4.4.1. Tous les résultats sont donnés pour les versions retardées et anticipées des bisimilarités.

Les systèmes d'axiomes et les systèmes de preuves sont obtenus en modifiant certaines règles des systèmes pour le π-calcul non typé, et en ajoutant quelques nouvelles lois. Les preuves de correction et de complétude, bien que nous suivions le schéma général des preuves du calcul non typé, différent beaucoup dans les détails. Un exemple de ceci est le traitement des canaux frais dans les actions de lecture et la fermeture par les substitutions injectives que nous commentons ci-dessous.

Dans le π-calcul non typé, l'assertion suivante est vraie :

$$\textit{Si } P \simeq Q \textit{ et } \sigma \textit{ est injective sur } fn(P,Q), \textit{ alors } P\sigma \simeq Q\sigma.$$

Par conséquent, il est suffisant de considérer tous les canaux libres dans P, Q et *un* seul canal frais en comparant les actions de lecture qu'accomplissent P et Q dans le jeu de bisimulation. Ce résultat est crucial dans la théorie algébrique du calcul non typé. Par exemple, dans le système de preuve pour la bisimilarité (version retardée) la règle d'inférence pour le préfixe de lecture est la suivante :

$$\begin{array}{l}\textit{Si } P\{b/x\} = Q\{b/x\} \textit{ pour tout } b \in fn(P,Q,c), \textit{ où } c \textit{ est un canal frais,}\\ \textit{alors } a(x).P = a(x).Q.\end{array}$$

Pour la bisimilarité typée la situation est différente. Prenons les processus

$$P \stackrel{\text{def}}{=} a(x : \mathtt{ob}T).\bar{x}c.\bar{c} \qquad Q \stackrel{\text{def}}{=} a(x : \mathtt{ob}T).\bar{x}c$$

et comparons-les contre un observateur Δ. Considérons ce qui se passe quand la variable x est remplacée par un canal frais b, dont le type dans Δ est S. Par la contrainte imposée par le typage, S doit être un sous-type de $\mathtt{ob}T$ (cf. Figure 2 (b)). Nous remarquons que les différents choix pour S donnent des résultats différents. Par exemple, si S est $\mathtt{ob}T$ lui-même, l'observateur n'a aucune capacité de lire sur b, il ne peut donc pas communiquer avec P et Q sur b. C'est-à-dire, du point de vue de l'observateur le préfixe écriture $\overline{b}c$ n'est pas observable et les deux processus sont considérés comme équivalents. De même si S est $\mathtt{bo}T$ alors le préfixe écriture $\overline{c}$ n'est pas observable. Cependant, si S est $\mathtt{bb}T$ alors $\bar{b}c.\bar{c}$ n'est pas équivalent à $\bar{b}c$, puisque toutes les écritures deviennent observables. Cet exemple illustre les difficultés essentielles pour la formulation des systèmes de preuves pour les bisimilarités typées :

1. La présence de sous-typage dans les substitutions change le type original d'une variable en un de ses sous-types.

2. Le choix de ces sous-types joue sur l'équivalence comportementale.

3. Les différents sous-types peuvent être incompatibles (ils n'ont aucun sous-type commun) entre eux (par exemple, $\mathtt{bo}T$ et $\mathtt{bb}T$ dans l'exemple ci-dessus ; ils sont tous les deux sous-types de $\mathtt{ob}T$).

Une conséquence de (2) et de (3), par exemple, est qu'il n'y a pas un "meilleur sous-type", qui est un type unique avec la propriété que l'équivalence sous ce type implique l'équivalence sous n'importe quels autres types.

Un autre exemple des modifications apportées par des types dans la théorie algébrique est la règle de congruence pour les préfixes : nous devons distinguer le cas dans lequel le sujet du préfixe est un canal, du cas dans lequel le sujet est une variable. C'est une différence plutôt subtile et technique ; elle est discutée à la Section 4.3.

Terminaison de processus mobiles par la typabilité

Un terme termine si toutes ses séquences de réduction sont de longueur finie. Dans les langages de programmation, la terminaison signifie que tous les calculs dans un programme finiront par s'arrêter. En informatique la terminaison a été intensivement étudiée dans les systèmes de réécriture [DM79, DH95] et le λ-calcul [Gan80, Bou03] (où la normalisation forte est un synonyme souvent utilisé). La terminaison a été également discutée dans les calculs de processus, notamment le π-calcul.

En effet, la terminaison est intéressante dans la concurrence. Par exemple, si nous interrogeons un processus, nous aimerions savoir qu'une réponse sera finalement produite (la terminaison toute seule ne garantit pas ceci, mais elle serait l'ingrédient principal dans une preuve). D'une façon similaire, quand nous chargeons une applet nous voudrions savoir que l'applet ne s'exécutera pas infiniment sur notre machine, qui plus est en absorbant toutes les ressources informatiques (une attaque du type "refus de service"). En général, si la vie d'un processus est infinie, nous voudrions savoir que le processus ne demeure pas vivant simplement en raison de l'activité interne infinie, et que le processus acceptera finalement des interactions avec l'environnement.

Deux langages de processus qui terminent ont été proposés dans [YBH04] et [San05]. Dans les deux cas, les preuves de la terminaison se servent des relations logiques, une technique bien connue pour les langages fonctionnels. Les langages de processus ainsi obtenus sont plutôt "fonctionnels", parce que les structures permises sont semblables à celles dérivées en encodant des fonctions comme processus. En particulier, les langages sont très restrictifs sur les lectures imbriquées (c'est-à-dire, la possibilité d'avoir des lectures sur des noms libres suivant d'autres lectures), et les lectures récursives (c'est-à-dire, les réplications comme $!a(x).P$ dans lequel le corps P peut appeler récursivement la garde a de la réplication). On interdit entièrement de tels motifs dans [YBH04] ; on permet des lectures imbriquées dans [San05] mais sous une forme très restreinte. Par exemple, le processus

$$a(x).!b.\bar{x}.\mathsf{0} \mid \bar{a}c.\mathsf{0} \tag{1}$$

(parfois le $\mathsf{0}$ à la fin est omis) n'est légal ni pour [YBH04] ni pour [San05]. Les restrictions dans [YBH04, San05] éliminent également des processus fonctionnels qui sont utiles, par exemple

$$F \stackrel{\text{def}}{=} !a(n,b).\ \texttt{if}\ n = 1\ \texttt{then}\ \bar{b}\langle 1\rangle\ \texttt{else}\ \nu c(\bar{a}\langle n-1, c\rangle \mid c(m).\bar{b}\langle m * n\rangle) \tag{2}$$

qui représente la fonction factorielle.

Pour garantir la terminaison des processus mobiles nous proposons plusieurs systèmes de types pour le π-calcul. Nous commençons par un système simple de types, qui ajoute une information de niveau aux types du π-calcul simplement typé. L'information de niveau nous aide à construire une mesure qui diminue le long de chaque chemin de réduction d'un processus bien typé. Par conséquent le fait que cette mesure soit bien fondée implique la terminaison des processus. Comme le système de types n'est pas très expressif, nous l'étendons en relachant quelques contraintes sur les lectures imbriquées et les lectures récursives, pour obtenir trois systèmes étendus de types. L'utilité de ces systèmes de types est montrée par trois exemples non triviaux : (1) il s'avère que toutes les fonctions récursives primitives peuvent être encodées comme des processus qui terminent ; (2) la méthode qui consiste à encoder les choix séparé en termes de composition parallèle, proposée dans [Nes00, SW01], n'introduit pas de divergence ; (3) chaque demande à la table de symboles (implémentée comme une chaîne dynamique de cellules), proposée dans [Jon93, San99], reçoit toujours une réponse en temps fini.

De façon générale, pour chaque système de types qui garantit la terminaison des processus nous choisissons une mesure qui diminue après certains pas de réduction. Pour comparer deux mesures, nous exploitons des ordres *lexicographiques* et des ordres *multi-ensemble*, des techniques bien connues dans les systèmes de réécriture [DM79, DJ90]. Pour le système simple de types, la mesure est seulement un vecteur qui compte, pour chaque niveau, le nombre d'écritures (qui ne sont pas gardées par des lectures répliquées) sur les canaux dont les types ont ce niveau. Pour les systèmes étendus de types, les idées sont semblables, mais les mesures deviennent plus sophistiquées puisque nous leur permettons de diminuer après un certain nombre (inconnu et variable mais fini) de réductions, avec quelques commutativités de réductions et des manipulations de processus.

Plan de la thèse

Nous introduisons au chapitre 2 quelques notions de base sur les calculs de processus comme CCS et le π-calcul. Nous prêtons une large attention aux types des canaux ; nous rappelons les notions de sortes, de types simples de canaux, et de sous-typage progressivement. Le matériel présenté dans ce chapitre sert à préparer le développement technique des chapitres suivants.

Au chapitre 3 nous présentons un calcul de processus probabiliste qui inclut les choix non-déterministe et probabiliste, en plus de la récursion. Nous donnons sa sémantique par les automates probabilistes proposés par Segala et Lynch. Nous présentons deux équivalences fortes et deux équivalences faibles. Nous montrons quelques propriétés des équivalences, en utilisant une version probabiliste de la technique de preuve dite "bisimulation up to". Pour les équivalences fortes nous donnons des axiomatisations complètes pour toutes les expressions, mais pour les équivalences faibles nous réalisons ce résultat seulement pour des expressions gardées. Nous conjecturons que dans le cas général de la récursion non-gardée les équivalences faibles sont indécidables. Dans les preuves de complétude, nos schémas de preuve sont inspirés par [Mil84, Mil89b, SS00], mais les détails sont plus compliqués à cause de la présence des dimensions probabiliste et non-déterministe. En effet, il s'avère que, pour obtenir une axiomatisation complète de l'équivalence d'observation, l'éxtension probabiliste des trois lois concernant τ de Milner [Mil89a] ne serait pas suffisante, et que nous

avons besoin d'une nouvelle règle. Enfin, pour les expressions sans récursion nous fournissons des axiomatisations complètes des quatre équivalences, avec des preuves de complétude bien simples.

Au chapitre 4 nous étudions la théorie algébrique d'un π-calcul de processus typés finis. Le système de types utilise des types de capacités. Premièrement nous considérons un sous-langage sans parallélisme. Ce petit langage montre déjà les obstacles principaux pour les axiomatisations. En suivant [HR04] nous donnons la sémantique opérationnelle du langage par un système de transitions étiquetées typées, sur lequel nous définissons la bisimulation typée (retardée). Deuxièmement nous construisons un système complet de preuve pour les termes fermés. Ensuite nous présentons une axiomatisation complète pour les termes ouverts. Le schéma de la preuve de complétude est semblable à celui du π-calcul non typé [PS95]. Les détails, cependant, sont tout à fait différents, en raison de la relation de sous-typage du système de types. Troisièmement nous rappelons la bisimilarité typée proposée dans [HR04], et fournissons un système de preuve pour les termes fermés, avec une axiomatisation indirecte pour tous les termes. Quatrièmement nous prouvons que la différence entre la bisimilarité retardée et la bisimilarité anticipée peut être capturée par un axiome. Finalement nous admettons la composition parallèle. Son effet sur les axiomatisations est d'ajouter une loi d'expansion pour éliminer toutes les occurrences de l'opérateur.

Au chapitre 5 nous considérons plusieurs systèmes de types tels que les processus bien typés dans chaque système terminent. D'abord, nous présentons un système simple de types, qui ajoute de l'information de niveau aux types du π-calcul simplement typé. Puis nous donnons trois améliorations de ce système, en vue notamment de traiter les lectures imbriquées et les lectures récursives. Pour tous les systèmes de types (sauf le deuxième, qui peut capturer toutes les fonctions récursives et primitives) nous présentons également des bornes supérieures du nombre de pas de normalisation. Ces bornes dépendent des structures des processus et des types des noms dans les processus. Nous montrons l'utilité des systèmes de types sur trois exemples non triviaux : les codages des fonctions récursives et primitives, la méthode pour coder le choix séparé par la composition parallèle [Nes00, SW01], une table de symboles implémentée par une chaîne dynamique de cellules [Jon93, San99].

Au chapitre 6 nous récapitulons les résultats de cette thèse et discutons quelques directions pour les travaux futurs.

Provenance du matériel

Cette thèse est partiellement basée sur des écrits publiés. La présentation d'un calcul de processus probabiliste et les axiomatisations de plusieurs équivalences comportementales probabilistes sont déjà parues dans [DP05] ; l'étude du π-calcul typé et les axiomatisations des bisimilarités typées ont été rapportées dans [DS04b, DS05]; le développement des systèmes de types pour assurer la propriété de terminaison de π-processus a été présenté dans [DS04a].

Chapter 1

Introduction

1.1 Background

Computer science aims to explain in a rigorous way how computational systems behave. Nowadays the notion of computational systems includes not only *sequential systems*, such as single programs in free-standing computers, but also *concurrent systems*, such as computer networks, and even proteins in biology and particles in physics. Some classical mathematical models (e.g. the λ-calculus [Bar84]), in spite of their success for describing sequential systems, turn out to be insufficient for reasoning about concurrent systems.

In the 1980's *process calculi* (sometimes called *process algebras*), notably CCS [Mil89a], CSP [Hoa85] and ACP [BK84, BW90], were proposed for describing and analyzing concurrent systems. All of them were designed around the central idea of *interaction* or *communication* between processes. In these formalisms, complex systems are built from simple subcomponents structurally, by a small set of primitive operators such as *prefix, nondeterministic choice, restriction, parallel composition* and *recursion*. The limitation of these traditional process algebras is that they are not able to effectively specify *mobile systems*, i.e., systems with a dynamically changing communication topology. On the basis of CCS, Milner, Parrow and Walker developed the π-calculus [MPW92], which achieves mobility by a powerful name-passing mechanism. The π-calculus is a very expressive formalism. It allows to encode data structures [Mil91], the λ-calculus [Mil92] and higher-order communications [San93]. Furthermore, it can be used for reasoning about object-oriented languages [Wal95].

As no single theory will serve all purposes, a great many variants and extensions of the classical process calculi have appeared in the literature. In the case of process calculi for distributed systems, there are three strands of work that have been developed and shown to be extremely important.

- The first strand is concerned with tuning the syntactic constructions of terms in order to better capture some specific features of concurrent systems such as asynchronous communications, higher-order communications, localities and migrations. In this respect one can make a long list: the asynchronous π-calculus [HT91, Bou92], the πI-calculus [San96a], the Lπ-calculus [Mer00], the Fusion calculus [PV98], the χ-calculus [Fu99], the Join calculus [Fou98], CHOCS

[Tho95], HOπ [San93], Dπ [HR02b], Klaim [DFP98], the Ambient calculus [CG00] and its variants, just to name a few.

- The second strand consists in equipping untyped process calculi with types so that processes interact in a safer and more efficient way. For example, a number of type systems are designed for the π-calculus; they are used in various applications such as static detection of errors in concurrent programs [Mil91], compiler optimizations [KPT99], resource access control [PS96, HR02b], guaranteeing other security properties such as deadlock-freedom [Kob98], noninterference [HY05] and termination [YBH04, DS04a].

- The third strand deals with probabilistic process calculi that support reasoning about probabilistic behaviour, as exhibited for instance in randomized, distributed and fault-tolerant systems. The typical approach is based on extending with probabilities existing models and techniques that have already proved successful in the nonprobabilistic settings. The usual feature of probabilistic process calculi is the existence of a *probabilistic choice* operator, see for example probabilistic extensions of CCS [GJS90, HJ90, Tof94, YL92], probabilistic CSP [Low91], probabilistic ACP [And99] and probabilistic asynchronous π-calculus [HP04].

Briefly speaking, this thesis includes our contributions in the second and third strands.

In order to study a programming language or a process calculus, one needs to assign a consistent meaning to each program or process under consideration. This meaning is the *semantics* of the language or calculus. Semantics is useful to verify or prove that programs behave as intended. Generally speaking, there are three major approaches for giving semantics to a programming language. The *denotational* approach seeks a valuation function which maps a program to its mathematical meaning. This approach has been very successful in modelling many sequential languages; programs are interpreted as functions from the domain of input values to the domain of output values. However, so far denotational interpretation of concurrent programs has not been as satisfactory as the denotational treatment of sequential programs.

The *operational* approach is shown to be quite useful for giving semantics of concurrent systems. The behaviour of a process is specified by its *structural operational semantics* [Plo81], described via a set of labelled transition rules inductively defined on the structure of a term. In this way each process corresponds to a labelled *transition graph*. The shortcoming of operational semantics is that it is too concrete, as a transition graph may contain many states which should be intuitively identified. Thus a lot of equivalences have been proposed and different transition graphs are compared modulo some equivalence relations.

The *axiomatic* approach aims at understanding a language through a few axioms and inference rules. Its importance is motivated by, among others, the following two reasons.

- Sound systems, even if they are not complete, may be useful for human or machine manipulation of terms. By exploiting these systems, a number of practical verification problems can be addressed.

- Complete systems help gaining insight into the nature of the operators and the equivalences involved. For example, the difference between two equivalences can be characterised by a

few axioms, particularly if adding these axioms to a complete system for one equivalence gives a complete system for the other equivalence. Another way of comparison is to fix a notion of equivalence and vary the expressions. Sometimes one lifts a complete system from a sublanguage to the whole language, by adding some extra axioms. Comparisons of both kinds are carried out in Chapter 3 and Chapter 4.

To explore the connection between operational and axiomatic semantics has always been an important and active subject in process calculi. Milner [Mil78] was the first person to advocate the development of an algebra of behaviours which are subject to a number of axioms expressed as equations. In [Mil80] a direct link is made for the first time between an algebraic theory and a behavioural equivalence based on an operational semantics. Since then there has been a large amount of work on algebraic theories of processes, for various behavioural equivalences in a wide range of process calculi. However, no much attention was paid to probabilistic and typed process calculi, though they turn out to be very useful in the analysis of modern distributed systems.

1.2 Objectives

This thesis focuses on the theoretical foundations of reasoning about algorithms and protocols for modern distributed systems. We believe that this kind of reasoning is important because, as happens too often, if a system is built without rigorous analysis of all the possible interactions between its components, then its behaviour is frequently incorrect. One witness is the recent discovery of security flaws in the IEEE 802.11 and the Bluetooth wireless communication protocols [BGW01, LL03].

For distributed systems it is interesting to consider models which encompass probabilities. One reason is that these systems are expected to provide reliable services despite the occurrence of various types of failure. Probabilistic processes can be used to describe fault-tolerant systems. For example, probabilistic information can be used for specifying the rate at which faulty communication channels drop messages and for verifying message-delivery properties of the corresponding system. In addition, probabilistic modelling can be used to break symmetry in distributed coordination problems (e.g. dining philosophers' problem, leader election problem, and consensus problem), to predict system behaviour based on the calculation of performance characteristics, and to represent and quantify other forms of uncertainty.

A model for distributed systems should also include the feature of mobility. Physical systems tend to have a fixed structure. But most systems in the information world are not physical; their links may be symbolic or virtual. For example, when one clicks on a hypertext in a web page, he induces a symbolic link between his machine and the remote web server. These symbolic links can be created or destroyed on the fly. An example of a virtual link is a radio connection, like the linkage between mobile phones that are roaming around and a network of base stations. Systems like these, with transient links, have a mobile structure.

With mobility, types turn out to be essential. For example, the theory of the untyped π-calculus is often insufficient to prove "expected" behavioural properties of processes. The reason is that when one uses the π-calculus to describe a system, one normally follows a discipline that controls how names may be used; but this discipline is not explicit in π-terms, and therefore it cannot play a role

in proofs. Types can be used to make such discipline explicit (cf. Part IV of [SW01]). Furthermore, types are useful for expressing control of interference, access rights, robust declassification, secure composition of components, as well as bounds on resource consumptions (e.g. time or memory allocations).

In fact, there is a strong practical motivation for considering both probability and mobility. How can a mobile phone system perform to satisfaction if the designer never considers the probable behaviour of users? A number of probabilistic models have been introduced which are variants of Markov chains, but for mobility they are at an early stage.

In the literature, probability and typed mobility are usually studied separately. Corresponding operational techniques have been developed. But very little has been done on algebraic techniques. However, algebraic techniques are very useful in computer science. For example, in the relational model for database [Cod70], algebraic laws have served as a basis for query optimisation and queries could be efficiently implemented through indexing and join techniques [RG02]. In process calculi, algebraic equations may be considered as rewriting rules for automated term manipulation [vdP01].

In this thesis we investigate algebraic techniques by considering the impact of probability and type mobility on the algebraic theories of process calculi. As each feature introduces new and non-trivial problems, to develop algebraic techniques for models that have both probability *and* typed mobility would be very complex. Therefore it is better to study them first in isolation. Due to this reason, in Chapter 3 we consider axiomatisations for a probabilistic calculus without mobility, and in Chapter 4 we provide axiomatisations for a typed mobile process calculus without probability. The types that we shall use are *capability types* [PS96], which distinguish between input capability, output capability, both input and output capability. This kind of types are one of the most useful and basic form of types in process calculi. They have been used to ensure type-consistent data exchange on communication channels, and to control access rights to channels and locations. Variants of capability types are now present in almost all experimental process calculi such as Klaim [DFP98], Spi [Aba99], and the Ambients Calculus [LS00]. Sometimes, they even become part of the syntax, e.g. in the Join calculus and the Lπ-calculus only output capabilities can be transmitted.

In mobile process calculi, types themselves can be used as a verification technique to analyse various properties of concurrent programs, such as deadlock [Kob98], livelock [Kob00], and information flow [HVY00, HR02a]. In Chapter 5 we develop one such technique for the problem of termination, which is an important property that many algorithms and protocols in distributed systems need to guarantee. In the case of symmetric distributed systems, probabilistic algorithms are usually more efficient than their deterministic counterparts, at the (insignificant) price that certain properties will happen with probability one but not necessarily with certainty (e.g., when tossing a fair coin, a "head" will eventually occur with probability one, but not with certainty). For all practical purposes, however, this difference is meaningless. Therefore, it is interesting to talk about probabilistic termination as well. However, since termination is itself a non-trivial problem, we consider types in isolation, without probability.

To summarise, in this thesis we develop algebraic and type-based techniques for reasoning about processes that feature probability and typed mobility. We consider the two features separately, both in the case of axiomatisations and in the case of termination, but we believe that our work

contributes building the basis for studying more advanced models which may combine probability with typed mobility.

Before proceeding to discuss in the following sections the motivations for each research topic of the thesis, we need to introduce some terminology. We use the general concept *axiomatisations* to mean both axiom systems and proof systems. For an equivalence on a set of terms, an *axiom system* consists of some equational axioms and the rules of equational reasoning (that is, rules on reflexivity, symmetry, transitivity, and congruence rules that make it possible to replace any subterm of a process by an equivalent term). A *proof system* has, in addition to axioms and rules of equational reasoning, other inference rules. Usually an axiom system is preferable to a proof system, because for example general techniques from term rewriting may then be applicable. However, when the process calculus in question includes non-trivial features such as recursion or types, sometimes it is hard to get a complete axiom system because we have to use other inference rules, i.e., what we obtain is actually a proof system. In that case we still call that system an axiomatisation, as in literature [Mil89b, Par01]. For an axiomatisation, *completeness* means that if two processes exhibit similar behaviour, i.e., their transition graphs are equivalent, then they are provably equal in the axiom system or the proof system; *soundness* means the converse.

1.3 Axiomatisations for Probabilistic Processes

The last decade has witnessed increasing interest in the area of formal methods for the specification and analysis of probabilistic systems [Seg95, BH97, AB01, PLS00, Sto02, CS02]. In [vGSS95] van Glabbeek *et al.* classified probabilistic models into *reactive, generative* and *stratified*. In reactive models, each labelled transition is associated with a probability, and for each state the sum of the probabilities with the same label is 1. Generative models differ from reactive ones in that for each state the sum of the probabilities of all the outgoing transitions is 1. Stratified models have more structure and for each state either there is exactly one outgoing labelled transition or there are only unlabelled transitions and the sum of their probabilities is 1.

In [Seg95] Segala pointed out that neither reactive nor generative nor stratified models capture real nondeterminism, an essential notion for modeling scheduling freedom, implementation freedom, the external environment and incomplete information. He then introduced a model, the *probabilistic automata* (PA), where both probability and nondeterminism are taken into account. Probabilistic choice is expressed by the notion of *transition*, which, in PA, leads to a probabilistic distribution over pairs (action, state) and deadlock. Nondeterministic choice, on the other hand, is expressed by the possibility of choosing different transitions. Segala proposed also a simplified version of PA called *simple probabilistic automata* (SPA), which are like ordinary automata except that a labelled transition leads to a probabilistic distribution over a set of states instead of a single state.

Figure 1.1 exemplifies the probabilistic models discussed above. In models where both probability and nondeterminism are present, like those of diagrams (4) and (5), a transition is usually represented as a bundle of arrows linked by a small arc. [SdV04] provides a detailed comparison between the various models, and argues that PA subsume all other models above except for the stratified ones.

We shall investigate in Chapter 3 axiom systems for a process calculus based on PA, in the sense

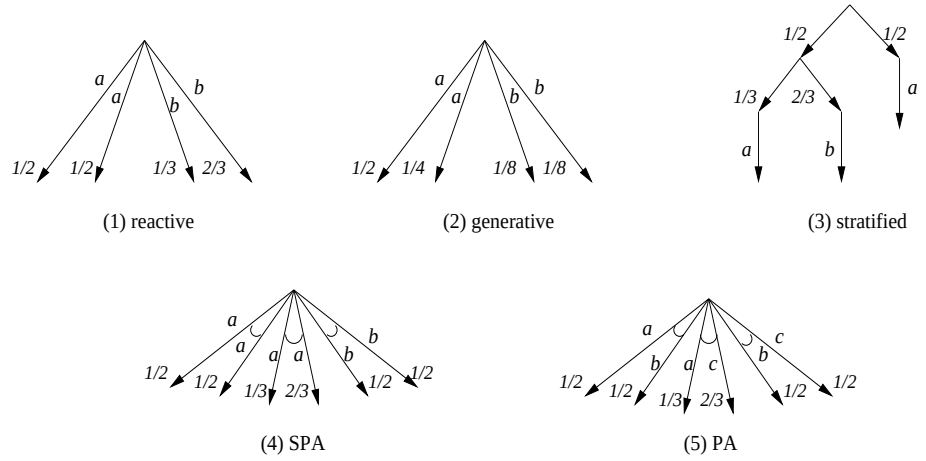

Figure 1.1: Probabilistic models

that the operational semantics of each expression of the language is a probabilistic automaton[1]. Axiom systems are important both at the theoretical level, as they help gaining insight into the calculus and establishing its foundations, and at the practical level, as tools for systems specification and verification. Our calculus is basically a probabilistic version of the calculus used by Milner to express finite-state behaviours [Mil84, Mil89b].

We shall consider two strong equivalences, one weak equivalence common in the literature, plus one novel notion of weak equivalence having the advantage of being sensitive to divergency. For recursion-free expressions we provide complete axiomatisations of all the four equivalences. For the strong equivalences we also give complete axiomatisations for all expressions, while for the weak equivalences we achieve this result only for guarded expressions.

The reason why we are interested in studying a model which expresses both nondeterministic and probabilistic behaviour, and an equivalence sensitive to divergency, is that one of the long-term goals of this line of research is to develop a theory which will allow us to reason about probabilistic algorithms used in distributed computing. In that domain it is important to ensure that an algorithm will work under any scheduler, and under other unknown or uncontrollable factors. The nondeterministic component of the calculus allows coping with these conditions in a uniform and elegant way. Furthermore, in many distributed computing applications it is important to ensure livelock-freedom (progress), and therefore we will need a semantics which does not simply ignore divergencies.

We end this section with a discussion about some related work in this research direction. In [Mil84] and [Mil89b] Milner gave complete axiomatisations for strong bisimulation and observational equivalence, respectively, for a core *CCS* [Mil89a]. These two papers serve as our starting point: in several completeness proofs that involve recursion we adopt Milner's *equational characterisation theorem* and *unique solution theorem*. In Section 3.4.1 and Section 3.5.2 we extend [Mil84] and [Mil89b] (for guarded expressions) respectively, to the setting of probabilistic process algebra.

[1]Except for the case of deadlock, which is treated slightly differently: following the tradition of process calculi, in our case deadlock is a state, while in PA it is one of the possible components of a transition.

In [SS00] Stark and Smolka gave a probabilistic version of the results of [Mil84] by replacing nondeterministic choice with probabilistic choice. So we extend the results of [SS00] in that we consider also nondeterminism. Note that when nondeterministic choice is added, Stark and Smolka's technique of proving soundness of axioms is no longer usable. (See the discussion at the beginning of Appendix A.2.) The same remark applies also to [AÉI02] which follows the approach of [SS00] but uses some axioms from iteration algebra to characterise recursion. In contrast, our probabilistic version of "bisimulation up to" techniques [Mil89a] work well when combined with the usual transition induction.

In [BS01] Bandini and Segala axiomatized both strong and weak behavioural equivalences for process calculi corresponding to SPA and to an alternated-model version of SPA. As their process calculus with non-alternating semantics corresponds to SPA, our results in Section 3.6 can be regarded as an extension of that work to PA.

For probabilistic process algebra of ACP-style, several complete axiom systems have appeared in the literature. However, in each of the systems either weak bisimulation is not investigated [BBS95, And99] or nondeterministic choice is prohibited [BBS95, AB01].

1.4 Axiomatisations for Typed Mobile Processes

The theory of the π-calculus has been studied in depth [Mil99, SW01]. Relevant parts of it are the algebraic theory and the type systems. Most of the algebraic theory has been developed on the untyped calculus; the results include axiomatisations that are sound and complete on finite processes for the main behavioural equivalences: late and early bisimilarity, late and early congruence [PS95, Lin94, Lin03], open bisimilarity [San96b], testing equivalence [BD95]. But at the same time, much of the research on types has focused on their behavioural effects. For instance, modifications of the standard behavioural equivalences have been proposed so as to take types into account [PS96, SW01].

We shall study in Chapter 4 the impact of types on the algebraic theory of the π-calculus. Precisely, we study axiomatisations of the typed π-calculus. Although algebraic laws for typed calculi of mobile processes have been considered in the literature [SW01], we are not aware of any axiomatisation.

The type system that we consider has *capability types* (sometimes called I/O types) [PS96, HR02b]. These types allow us to distinguish, for instance, the capability of using a channel in input from that of using the channel in output. A capability type shows the capability of a channel and, recursively, of the channels carried by that channel. For instance, a type $a : \mathtt{iob}T$ (for an appropriate type expression T) says that channel a can be used only in input; moreover, any channel received at a may only be used in output — to send channels which can be used both in input and in output. Thus, process $a(x).\bar{x}b.b(y).\bar{b}y.\mathtt{0}$ (sometimes the trailing $\mathtt{0}$ is omitted) is well-typed under the type assignment $a : \mathtt{iob}T, b : \mathtt{b}T$. We recall that $\bar{a}b.P$ is the output at a of channel b with continuation P; $a(x).P$ is an input at a with x a placeholder for channels received in the input whose continuation is P.

On calculi for mobility, capability types have emerged as one of the most useful forms of types,

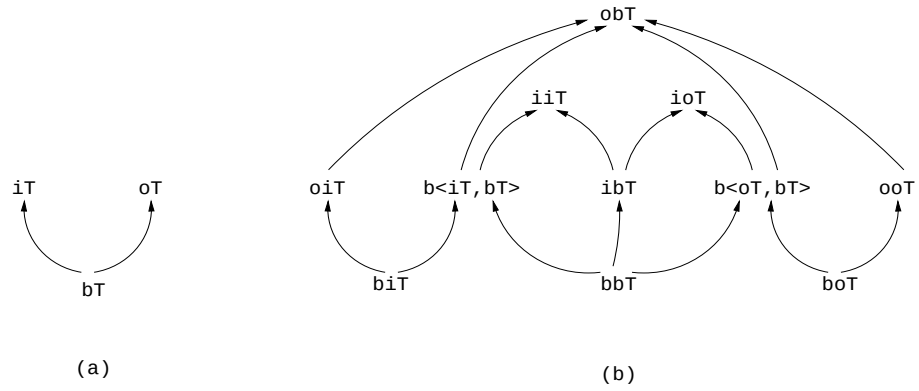

Figure 1.2: An example of subtyping relation, with $T = \texttt{unit}$

and one whose behavioural effects are most prominent. Capabilities are useful for protecting resources; for instance, in a client-server model, they can be used for preventing clients from using the access channel to the server in input and stealing messages to the server; similarly they can be used in distributed programming for expressing security constraints [HR02b]. Capabilities give rise to *subtyping*: the output capability is contravariant, whereas the input capability is covariant. As an example, we show a subtyping relation in Figure 1.2, where an arrow from one type to another means that the source of the arrow is a subtype of the target. There are three forms of types for channel names: $\texttt{i}T, \texttt{o}S$ and $\texttt{b}\langle T, S\rangle$; they correspond to the capability to receive values of type T, send values of type S, or to do both. We use $\texttt{b}T$ as an abbreviation of $\texttt{b}\langle T, T\rangle$. The depth of nesting of capabilities is 1 for all types in diagram (a), and 2 for all types in diagram (b). (The formal definitions of types and subtyping relation will be given in Section 4.1.1.) Subtyping is useful when the π-calculus is used for object-oriented programming, or for giving semantics to object-oriented languages.

To see why the addition of capability types has semantic consequences, consider

$$P \stackrel{\text{def}}{=} \nu c\, \overline{b}c.a(y).(\overline{y} \mid c) \qquad Q \stackrel{\text{def}}{=} \nu c\, \overline{b}c.a(y).(\overline{y}.c + c.\overline{y}).$$

These processes are not behaviourally equivalent in the untyped π-calculus. For instance, if the channel received at a is c, then P can terminate after 2 interactions with the external observer. By contrast, Q always terminates after 4 interactions with the observer. However, if we require that only the input capability of channels may be communicated at b, then P and Q are indistinguishable in any (well-typed) context. For instance, since the observer only receives the input capability on c, it cannot resend c along a: channels sent at a require at least the output capability (cf. the occurrence of $\overline{y}$). Therefore, in the typed setting, processes are compared w.r.t. an observer with certain capabilities (i.e., types on channels). Denoting with Δ these capabilities, then typed bisimilarity between P and Q is written $P \mathrel{\dot{=}}_{\Delta} Q$.

In the untyped π-calculus, labelled transition systems (LTS) are defined on processes; the transition $P \xrightarrow{\alpha} P'$ means that P can perform action α and then become P'. In the typed π-calculus, the information about the observer capabilities is relevant because the observer can only test processes on interactions for which the observer has all needed capabilities. Hence typed labelled transition

systems (TLTS) are defined on configurations, and a configuration $\Delta\sharp P$ is composed of a process P and the observer capabilities Δ (we sometimes call Δ the external environment). A transition $\Delta\sharp P \xrightarrow{\alpha} \Delta'\sharp P'$ now means that P can evolve into P' after performing an action α allowed by the environment Δ, which in turn evolves into Δ'.

Capability types have been introduced in [PS96]. A number of variants and extensions have then been proposed. We follow Hennessy and Riely's system [HR02b], in which, in contrast with the system in [PS96]: (i) there are partial meet and join operations on types; (ii) the typing rule for the *matching* construct (the construct used for testing equality between channels) is very liberal, in that it can be applied to channels of arbitrary types (in [PS96] only channels that possess both the input and the output capability can be compared). While (i) only simplifies certain technical details, (ii) seems essential. Indeed, the importance of matching for the algebraic theory of the π-calculus is well-known (it is the main reason for the existence of matching in the untyped calculus).

Typed bisimilarity and the use of configurations for defining typed bisimilarity have been introduced in [BS98]. We follow a variant of them put forward by Hennessy and Rathke [HR04], because it uses the type system of [HR02b] and includes the matching construct.

Two important results that we have obtained are a proof system and an axiom system for typed bisimilarity ($\eqsim$). The proof system has a simple correctness proof but only works on the closed terms. The axiom system is for all finite processes. The bisimilarity $\eqsim$ is a variant of the one in [HR04]. For the typed bisimilarity in [HR04] we provide a proof system for the closed terms, and an indirect axiomatisation of all terms that exploits the system of $\eqsim$. We have not been able to give a direct axiomatisation: the main difficulties are discussed in Section 4.4.1. All results are given for both the late and the early versions of the bisimilarities.

The axiomatisations are obtained by modifying some of the rules of the systems for the untyped π-calculus, and by adding a few new laws. While the proofs of soundness and completeness follow the general schema of the proofs of the untyped calculus, they have quite different details. An example of this is the treatment of fresh channels in input actions and the closure under injective substitutions, that we comment on below.

In the untyped π-calculus, the following holds:

$$\textit{If } P \eqsim Q \textit{ and } \sigma \textit{ is injective on } fn(P,Q), \textit{ then } P\sigma \eqsim Q\sigma.$$

Hence it is sufficient to consider all free channels in P, Q and *one* fresh channel when comparing the input actions of P and Q in the bisimulation game. This result is crucial in the algebraic theory of untyped calculi. For instance, in the proof system for (late) bisimilarity the inference rule for input is:

> *If* $P\{b/x\} = Q\{b/x\}$ *for all* $b \in fn(P,Q,c)$, *where* c *is a fresh channel,*
> *then* $a(x).P = a(x).Q$.

For typed bisimilarity the situation is different. Take the processes

$$P \stackrel{\text{def}}{=} a(x : \mathtt{ob}T).\bar{x}c.\bar{c} \qquad Q \stackrel{\text{def}}{=} a(x : \mathtt{ob}T).\bar{x}c$$

and compare them w.r.t. an observer with capabilities Δ. Consider what happens when the variable x is replaced by a fresh channel b, whose type in Δ is S. By the constraint imposed by types, S

must be a subtype of the type $\mathtt{ob}T$ for x (see Figure 1.2 (b)). Now, different choices for S will give different results. For instance, if S is $\mathtt{ob}T$ itself, then the observer has no input capability on b, thus cannot communicate with P and Q at b. That is, from the observer's point of view the output $\overline{b}c$ is not observable and the two processes evolve to equivalent ones. Similarly if S is $\mathtt{bo}T$ then the output $\overline{c}$ is not observable. However, if S is $\mathtt{bb}T$ then $\overline{b}c.\overline{c}$ is not equivalent to $\overline{b}c$, since all outputs become observable. This example illustrates the essential difficulties in formulating proof systems for typed bisimilarities:

1. Subtyping appears in substitutions and changes the original type of a variable into one of its subtypes.

2. The choice of this subtype is relevant for behavioural equivalence.

3. Different subtypes may be incompatible (have no common subtype) with one another (for instance, $\mathtt{bo}T$ and $\mathtt{bb}T$ in the example above; they are both subtypes of $\mathtt{ob}T$).

A consequence of the last two clauses, for instance, is that there is not a "best subtype", that is a single type with the property that equivalence under this type implies equivalence under any other types.

Another example of the consequences brought by types in the algebraic theory is the congruence rule for prefixes: we have to distinguish the cases in which the subject of the prefix is a channel from the case in which the subject is a variable. This is a rather subtle and technical difference, that is discussed in Section 4.3.

1.5 Termination of Mobile Processes by Typability

A term terminates if all its reduction sequences are of finite length. As far as programming languages are concerned, termination means that computation in programs will eventually stop. In computer science termination has been extensively investigated in term rewriting systems [DM79, DH95] and λ-calculi [Gan80, Bou03] (where strong normalization is a synonym more commonly used). Termination has also been discussed in process calculi, notably the π-calculus.

Indeed, termination is interesting in concurrency. For instance, if we interrogate a process, we may want to know that an answer is eventually produced (termination alone does not guarantee this, but termination would be the main ingredient in a proof). Similarly, when we load an applet we would like to know that the applet will not run for ever on our machine, possibly absorbing all the computing resources (a "denial of service" attack). In general, if the lifetime of a process can be infinite, we may want to know that the process does not remain alive simply because of non-terminating internal activity, and that, therefore, the process will eventually accept interactions with the environment.

Languages of terminating processes are proposed in [YBH04] and [San05]. In both cases, the proofs of termination make use of logical relations, a well-known technique from functional languages. The languages of terminating processes so obtained are however rather "functional", in that the structures allowed are similar to those derived when encoding functions as processes. In particular,

the languages are very restrictive on nested inputs (that is, the possibility of having free inputs underneath other inputs), and recursive inputs (that is, replications $!a(x).P$ in which the body P can recursively call the guard a of the replication). Such patterns are entirely forbidden in [YBH04]; nested inputs are allowed in [San05] but in a very restricted form. For example, the process

$$a(x).!b.\bar{x}.\mathsf{0} \mid \bar{a}c.\mathsf{0} \tag{1.1}$$

is legal neither for [YBH04] nor for [San05]. The restrictions in [YBH04, San05] actually rule out also useful functional processes, for instance

$$F \stackrel{\text{def}}{=} !a(n,b).\ \texttt{if}\ n = 1\ \texttt{then}\ \bar{b}\langle 1\rangle\ \texttt{else}\ \nu c(\bar{a}\langle n-1, c\rangle \mid c(m).\bar{b}\langle m * n\rangle) \tag{1.2}$$

which represents the factorial function.

To guarantee the termination property of mobile processes we propose several type systems (which are quite different from the type systems discussed in Section 1.4) for the π-calculus. We start from a core type system, which adds level information to the types of the simply typed π-calculus. The level information helps us to construct a measure which decreases along with each reduction path of a well-typed process. Therefore the well-foundedness of the measure implies the desired termination property of processes. As the core type system is not very expressive, we extend it by relaxing some constraints on nested inputs and recursive inputs, thus we obtain three extended type systems. The usefulness of these type systems are shown by some non-trivial examples. For instance, it turns out that all primitive recursive functions can be encoded as terminating processes; the protocol of encoding separate choice in terms of parallel composition proposed in [Nes00, SW01] does not introduce divergency; each request to the symbol table (implemented as a dynamic chain of cells) given in [Jon93, San99] is always answered within finite amount of time.

Roughly, for each type system to prove termination we choose a measure which decreases after finite steps of reduction. To compare two measures, we exploit *lexicographic* and *multiset orderings*, well-known techniques in term rewriting systems [DM79, DJ90]. For the core type system, the measure is just a vector recording, for each level, the number of outputs (unguarded by replicated inputs) at channels with that level in the type. For the extended type systems, the ideas are similar, but the measures become more sophisticated since we allow them to decrease after some finite (unknown and variable) number of reductions, up to some commutativities of reductions and process manipulations.

1.6 Outline of the Thesis

The material presented in Chapter 2 is meant to prepare the technical development in the rest of the thesis. We introduce some basic notions about process calculi, with CCS and the π-calculus as our templates. We then focus on channel types; we review sorts, simple channel types and subtyping progressively.

In Chapter 3 we introduce a probabilistic process calculus which includes both nondeterministic and probabilistic choice, as well as recursion. We give its semantics in terms of Segala and Lynch's probabilistic automata. We introduce two strong equivalences and two weak equivalences. We show

some properties of the equivalences, using a probabilistic version of "bisimulation up to" proof techniques. For the strong equivalences we give complete axiomatisations for all expressions, while for the weak equivalences we achieve this result only for guarded expressions. We conjecture that in the general case of unguarded recursion the "natural" weak equivalences are undecidable. In the completeness proofs, our proof schemas are inspired by [Mil84, Mil89b, SS00], but the details are more involved due to the presence of both probabilistic and nondeterministic dimensions. Indeed, it turns out that, to give a complete axiomatisation of observational equivalence, the simple probabilistic extension of Milner's three τ-laws [Mil89a] would not be sufficient, thus we need a new rule. At last, for recursion-free expressions we provide axiomatisations of all the four equivalences, whose completeness proofs are very simple.

In Chapter 4 we study the algebraic theory of a finite π-calculus with capability types. Firstly we consider a sublanguage without parallelism. This small language already shows the major obstacles for axiomatisations. Following [HR04] we give the operational semantics of the language in terms of a typed labelled transition system, from which we define typed (late) bisimulation. Secondly we set up a complete proof system for closed terms. Then we present a complete axiom system for open terms. The schema of the completeness proof is similar to that for the untyped π-calculus [PS95]. The details, however, are quite different, due to the rich subtyping relation of the type system. Thirdly we recall the typed bisimilarity proposed in [HR04], and provide a proof system for closed terms, together with an indirect axiomatisation for all terms. Fourthly we show that the difference between late and early bisimilarity can be captured by one axiom. Lastly we admit parallel composition. Its effect on the axiomatisations is to add an expansion law to eliminate all occurrences of the operator.

In Chapter 5 we consider several type systems such that well-typed processes under each system are ensured to terminate. First, we present a core type system, which adds level information to the types of the simply typed π-calculus. Then we give three refinements of the core system. Nested inputs and recursive inputs are the main patterns we focus on. For all the type systems (except for the second one, which can capture primitive recursive functions) we also present upper bounds to the number of steps well-typed processes take to terminate. Such bounds depend on the structure of the processes and on the types of the names in the processes. We show the usefulness of the type systems on some non-trivial examples: the encodings of primitive recursive functions, the protocol for encoding separate choice in terms of parallel composition from [Nes00, SW01], a symbol table implemented as a dynamic chain of cells from [Jon93, San99].

In Chapter 6 we summarise the achievements of this thesis and discuss some directions for potential future work.

Provenance of the material

This thesis is partially based on published material. The presentation of a probabilistic process calculus and the axiomatisations of several probabilistic behavioural equivalences appeared in [DP05]; the study of the typed π-calculus and the axiomatisation of typed bisimilarity were presented in [DS04b, DS05]; the type systems for ensuring the termination property of π-processes were proposed in [DS04a].

Chapter 2

Preliminaries

This chapter introduces some basic notions about process calculi. They are going to be lifted to richer settings in the following chapters by accommodating probabilities and more advanced types. The presentation is based on CCS and the π-calculus, and partly guided by two textbooks [Mil99, SW01].

2.1 A Calculus of Communicating Systems

We presuppose an infinite set of *process variables*, $Var = \{X, Y, ...\}$, and an infinite set of *names*, $\mathcal{N} = \{u, v, ...\}$. We use the set of *conames*, $\overline{\mathcal{N}} = \{\bar{u} \mid u \in \mathcal{N}\}$. Given a special name τ, we let ℓ range over the set of *labels*, $\mathcal{L} = \mathcal{N} \cup \overline{\mathcal{N}} \cup \{\tau\}$. A label represents an indivisible action that a communicating system performs, such as reading a datum, or sending a datum. The class of *process expressions* $\mathcal{E}_{\mathsf{ccs}}$ is given by the following grammar:

$$E, F ::= \mathsf{0} \;\big|\; \ell.E \;\big|\; E + F \;\big|\; E \mid F \;\big|\; \nu u E \;\big|\; X \;\big|\; \mu_X E$$

The expression $\mathsf{0}$ represents *inaction*. The *prefix* $\ell.E$ describes the behaviour of first performing an action labelled ℓ then behaving like E. The *sum* or *nondeterministic choice* $E + F$ behaves either like E or F nondeterministically. The *parallel composition* $E \mid F$ allows each of its components to behave independently, but also to synchronize with each other by a handshake on a complementary name. The *restriction* $\nu u E$ restricts the scope of u to E. The *recursion* $\mu_X E$ provides infinite behaviour by unfolding itself to be $E\{\mu_X E/X\}$. Operator precedence is (1) prefix, restriction, recursion, (2) parallel composition, and (3) nondeterministic choice.

Note that in CCS [Mil89a] the operators differ a little. The restriction $\nu u E$ is written $E\backslash u$. There is also a *renaming* operator $E[v_1/u_1, ..., v_n/u_n]$, which is not present here; its job is largely done by syntactic substitution of names. We shall write $E\{\widetilde{v}/\widetilde{u}\}$ for syntactic substitution of names $\widetilde{v}$ for names $\widetilde{u}$.

We use $fpv(E)$ for the set of free process variables (i.e., not bound by any μ_X) in E. As usual we identify expressions which differ only by a change of bound process variables. We shall write $E\{F_1, ..., F_n/X_1, ..., X_n\}$ or $E\{\widetilde{F}/\widetilde{X}\}$ for the result of simultaneously substituting F_i for each occurrence of X_i in E $(1 \leq i \leq n)$, renaming bound variables if necessary.

$$\mathsf{act}\ \frac{}{\ell.E \xrightarrow{\ell} E} \qquad \mathsf{sum1}\ \frac{E \xrightarrow{\ell} E'}{E + F \xrightarrow{\ell} E'}$$

$$\mathsf{par1}\ \frac{E \xrightarrow{\ell} E'}{E \mid F \xrightarrow{\ell} E' \mid F} \qquad \mathsf{com1}\ \frac{E \xrightarrow{u} E' \quad F \xrightarrow{\bar{u}} F'}{E \mid F \xrightarrow{\tau} E' \mid F'}$$

$$\mathsf{res}\ \frac{E \xrightarrow{\ell} E'}{\nu u E \xrightarrow{\ell} \nu u E'} \quad \text{for } u \neq \ell \qquad \mathsf{rec}\ \frac{E\{\mu_X E/X\} \xrightarrow{\ell} E'}{\mu_X E \xrightarrow{\ell} E'}$$

Table 2.1: The transition rules for $\mathcal{E}_{\mathsf{ccs}}$

For operational semantics, we use a labelled transition system

$$(\mathcal{E}_{\mathsf{ccs}}, \mathcal{L}, \{\xrightarrow{\ell} \subseteq \mathcal{E}_{\mathsf{ccs}} \times \mathcal{E}_{\mathsf{ccs}} \mid \ell \in \mathcal{L}\})$$

with $\mathcal{E}_{\mathsf{ccs}}$ as the set of states and $\mathcal{L}$ as transition labels. The transition relation is defined as the smallest relation generated by the rules in Table 2.1. The symmetric rules of sum1, par1 and com1 are omitted. As can be seen from the rule com1, for a communication between two processes to take place, one of them must offer an atomic action u, the other its complementary action $\bar{u}$. The communication results in a τ-action, meaning that the communication serves as synchronisation and the result is invisible. On the other hand, in some literature on the analysis of distributed systems, parallel composition is defined as in CSP [Hoa85], where a communication between two processes occurs if both of them offer the same action u, and the result is still a u-action.

2.2 The π-calculus

We first give the motivation and introduce the untyped π-calculus. Then we focus on channel types; we review sorts, simple channel types and subtyping progressively.

2.2.1 From CCS to the π-calculus

A significant limitation of CCS, as argued in [Mil99], is that it is not able to naturally specify communicating systems with dynamically changing connectivity. For example, let us consider the system composed of three components P, Q and R as displayed in Figure 2.1(1). Initially P and R are connected by the link a, while P and Q are connected by b. In the configuration of Figure 2.1(2), P and Q have evolved into P' and Q' respectively and the link to R has moved from P to Q. Since CCS gives us no way of creating new links among existing components, we are not able to specify the system in (1) as a CCS expression that can evolve into (2). However, this kind of evolution occurs often in many real systems. For instance, we may imagine R as a critical section that are accessed by P and Q successively. A natural way of dealing with link mobility like this is to give actions more structures so that links can be passed around in communicating systems. This is the method adopted by the π-calculus.

(1) (2)

Figure 2.1: Link mobility

2.2.2 The Untyped π-calculus

Let the set $\mathcal{N}$ of names be defined as in Section 2.1. The set $\mathcal{P}_\pi$ of processes is defined by the following syntax:

$$P, Q ::= 0 \mid u(x).P \mid \bar{u}v.P \mid P \mid Q \mid P + Q \mid \nu u P \mid !u(x).P$$

The *input prefix* $u(x).P$ can receive any name via u and continue as P with the received name substituted for x. The *output prefix* $\bar{u}v.P$ can send v via u and continue as P. The *replicated input* $!u(x).P$ can be thought of as an infinite composition $u(x).P \mid u(x).P \mid \cdots$, and it can encode recursive definitions [Mil91]. For example, take the simple CCS expression $E \stackrel{\text{def}}{=} \mu_X(u.(X \mid X))$, which has the infinite behaviour:

$$E \xrightarrow{u} E \mid E \xrightarrow{u} E \mid E \mid E \xrightarrow{u} \ldots$$

The same effect can be derived by using a replicated input:

$$\begin{array}{ll} & \nu v(\bar{v} \mid !v.u.(\bar{v} \mid \bar{v})) \\ \xrightarrow{\tau}\xrightarrow{u} & \nu v(\bar{v} \mid \bar{v} \mid !v.u.(\bar{v} \mid \bar{v})) \\ \xrightarrow{\tau}\xrightarrow{u} & \nu v(\bar{v} \mid \bar{v} \mid \bar{v} \mid !v.u.(\bar{v} \mid \bar{v})) \\ \xrightarrow{\tau}\xrightarrow{u} & \ldots \end{array}$$

All other operators (inaction, sum, restriction, and parallel composition) keep their meaning as in Section 2.1.

The π-calculus has two name-binding operators. In the processes $u(v).P$ and $\nu v P$ the occurrences of v in P are considered *bound* with scope P. An occurrence of a name in a process is *free* if it is not bound. We write $bn(P)$ (resp. $fn(P)$) for the set of names that have a bound (resp. free) occurrence in P. Changing a bound name into a fresh name is called *alpha-conversion*, and we identify processes up to alpha-conversion.

A substitution $\{v/u\}$ is a function on names that maps u to v and acts as identity on other names. Hence the postfix operator $P\{v/u\}$ is defined as the result of replacing all free occurrences of u in P by v, possibly applying alpha-conversion to avoid name capture by introducing unintended bound occurrences of names.

Convention: When considering a collection of processes and substitutions, we assume that each bound name of the processes is chosen to be unique, i.e., different from other names of the processes and the substitutions.

kind	α	$subj(\alpha)$	$obj(\alpha)$	$fn(\alpha)$	$bn(\alpha)$
input	uv	u	v	$\{u, v\}$	$\emptyset$
free output	$\bar{u}v$	u	v	$\{u, v\}$	$\emptyset$
bound output	$\bar{u}(v)$	u	v	$\{u\}$	$\{v\}$
internal action	τ	-	-	$\emptyset$	$\emptyset$

Table 2.2: Terminology and notation for actions

The early style [MPW92] of operational semantics for processes in $\mathcal{P}_\pi$ is specified via a labelled transition system

$$(\mathcal{P}_\pi, Act, \{\xrightarrow{\alpha} \subseteq \mathcal{P}_\pi \times \mathcal{P}_\pi \;\big|\; \alpha \in Act\})$$

where Act stands for the set of *actions*, of which there are four kinds.

1. The *internal* action τ. As in CCS, $P \xrightarrow{\tau} Q$ means that P can evolve into Q without any interaction with the environment. Internal actions arise from internal communication within a process.

2. An *input* action uv. The transition $P \xrightarrow{uv} Q$ means that P can receive v along u before evolving into Q. This departs from CCS because an input action contains the actual received value. Input actions arise from input prefixes.

3. A *free output* action $\bar{u}v$. The transition $P \xrightarrow{\bar{u}v} Q$ implies that P can emit the free name v along name u. Free output actions arise from output prefixes.

4. A *bound output* action $\bar{u}(v)$. Intuitively, $P \xrightarrow{\bar{u}(v)} Q$ means that P can emit the private name v (i.e. v is bound in P) along u before evolving into Q. Bound output actions arise from free output actions which carry names out of their scope, as in the process $\nu v(\bar{u}v.Q)$ for example.

Table 2.2 displays each kind of action, its *subject*, its *object*, its set of *free names*, and its set of *bound names*. We let $n(\alpha) \stackrel{\text{def}}{=} fn(\alpha) \cup bn(\alpha)$ denote the set of names occurring in α.

The transition relation $\xrightarrow{\alpha}$ is defined by the rules in Table 2.3. The symmetric rules of sum1, par1, com1 and close1 are omitted. Some of the rules deserve to be explained. We see from the rule in that $u(x).P$ can receive *any* name via u, and when a name is received it is substituted for the placeholder x in P. The rule open expresses extrusion of the scope of the name v, which can be seen in the rule close1. A process capable of performing a bound output $\bar{u}(v)$ can interact with a process that can receive v via u and in which v is not free. The interaction is represented by a τ-transition, and in the derivative the two components are within the scope of a restriction νv. We may say that the scope of v is opened via open while closed again via close1. The scope of the restricted name is extended to include the process that receives it. The side condition in the rule par1 is necessary because it prevents free names in Q from being incorrectly identified as bound names in P'. The rule rep captures the idea that $!u(x).P$ can spawn infinitely many copies of $u(x).P$ and each copy can perform an input action as in the rule in.

Sometimes we use the notation $\stackrel{\alpha}{\Longrightarrow}$ which is an abbreviation for $(\xrightarrow{\tau})^* \xrightarrow{\alpha} (\xrightarrow{\tau})^*$, where $(\xrightarrow{\tau})^*$ is the reflexive and transitive closure of $\xrightarrow{\tau}$.

$$\textsf{in}\ \frac{}{u(x).P \xrightarrow{uv} P\{v/x\}} \qquad \textsf{out}\ \frac{}{\bar{u}v.P \xrightarrow{\bar{u}v} P}$$

$$\textsf{sum1}\ \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \textsf{par1}\ \frac{P \xrightarrow{\alpha} P' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\textsf{com1}\ \frac{P \xrightarrow{\bar{u}v} P' \quad Q \xrightarrow{uv} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \textsf{close1}\ \frac{P \xrightarrow{\bar{u}(v)} P' \quad Q \xrightarrow{uv} Q' \quad v \notin fn(Q)}{P \mid Q \xrightarrow{\tau} \nu v(P' \mid Q')}$$

$$\textsf{res}\ \frac{P \xrightarrow{\alpha} P' \quad u \notin n(\alpha)}{\nu u P \xrightarrow{\alpha} \nu u P'} \qquad \textsf{open}\ \frac{P \xrightarrow{\bar{u}v} P' \quad v \neq u}{\nu v P \xrightarrow{\bar{u}(v)} P'}$$

$$\textsf{rep}\ \frac{}{!u(x).P \xrightarrow{uv} !u(x).P \mid P\{v/x\}}$$

Table 2.3: The transition rules for $\mathcal{P}_\pi$

The capacity to change the connectivity of a network of processes is the crucial difference between the π-calculus and CCS. Let us consider an example based on Figure 2.1. Suppose two processes P, Q need to use some resource R in a critical section. Initially only process P has access to the resource, represented by a communication link a. After an interaction with Q along other link b this access is transferred to Q. This kind of behaviour can be described in the π-calculus as follows: process P that sends a along b is $\bar{b}a.P'$ (suppose a does not appear in P'); process Q that receives some link along b and then uses it to send data c is $b(x).\bar{x}c.Q''$. The interaction between P and Q is formulated as:

$$\bar{b}a.P' \mid b(x).\bar{x}c.Q'' \xrightarrow{\tau} P' \mid \bar{a}c.Q''.$$

After the interaction, the connection between P and R disappears while a new connection between Q' and R is built, where Q' is the process $\bar{a}c.Q''$.

The π-calculus presented above is *monadic* in that a message consists of exactly one name. Sometime we want to send messages consisting of more than one name. So it is useful to allow *polyadic* inputs and outputs: $u(x_1, ..., x_n).P$ and $\bar{u}\langle v_1, ..., v_n\rangle.Q$. Accordingly we can extend the transition rules in Table 2.3 to allow for polyadic communication:

$$u(\widetilde{x}).P \mid \bar{u}\langle\widetilde{v}\rangle.Q \xrightarrow{\tau} P\{\widetilde{v}/\widetilde{x}\} \mid Q$$

where $\widetilde{x}$ and $\widetilde{v}$ have the same length. After the extension we obtain the polyadic π-calculus [Mil91].

2.2.3 Sorts and Sorting

To regulate the use of names, Milner introduced the notion *sorting* [Mil91], which is essential to avoid disagreement in the arities of tuples carried by a given name in the polyadic π-calculus. Assume a basic collection Σ of *sorts*. To every name u is assigned a sort ι, and we write $u : \iota$. A *sort list* over Σ is a finite sequence $\widetilde{\iota} = \iota_1, ..., \iota_n$ of sorts. Σ^* is the set of all sort lists over Σ. We write $\widetilde{u} : \widetilde{\iota}$ if $u_i : \iota_i$ for all i with $1 \leq i \leq n$. A *sorting* over Σ is a partial function

$$f : \Sigma \mapsto \Sigma^*$$

Figure 2.2: A printer example

and we say that a process respects f if, for every subterm of the form $u(\widetilde{v}).P$ or $\bar{u}\langle\widetilde{v}\rangle.Q$,

$$\text{if } u : \iota \text{ then } \widetilde{v} : f(\iota).$$

For example, for the process F in (1.2), let us choose $\Sigma = \{S_a, S_{bc}, \mathtt{Nat}\}$ with

$$a : S_a,\ b : S_{bc},\ c : S_{bc},\ m : \mathtt{Nat},\ n : \mathtt{Nat}.$$

Then a sorting f respected by F is such that

$$f : \begin{cases} S_a & \mapsto & \mathtt{Nat}, S_{bc} \\ S_{bc} & \mapsto & \mathtt{Nat}. \end{cases}$$

2.2.4 A Simple Example

Before proceeding to the formal presentation of type systems for the π-calculus, we informally explain the usefulness of types, capability types in particular, by a simple example from [PS96]. Imagine the common situation in which two processes must cooperate in the use of a shared resource such as a printer. The printer provides a request channel u on which the client processes send their data for printing. If one client process has the form $Q_1 \stackrel{\text{def}}{=} \bar{u}v_1.\bar{u}v_2.\mathtt{0}$, then we expect that executing the program $\nu u(P \mid Q_1 \mid Q_2)$ should result in the print jobs represented by v_1 and v_2 eventually being received and processed, in that order, by the printer process P (see Figure 2.2(1), where an arrow from one process to another means that some data are transmitted from the source of the arrow to the target). However, this is not necessarily the case: a misbehaving implementation of Q_2 can disrupt the protocol expected by P and Q_1 simply by reading print requests from u and throwing them away: $Q_2 \stackrel{\text{def}}{=} !u(v).\mathtt{0}$ (see Figure 2.2(2)). We can prevent this kind of bad behaviour by distinguishing three kinds of access to a channel – the capability to write values, the capability to read values, and the capability to do both – and requiring each process to use its channels with some prescribed capabilities. Here, for instance, the client processes should only be allowed to write to u. The printer, on the other hand, should only read from u. When we impose this constraint, process Q_2 will be ruled out because it attempts to read from u.

2.2.5 The Simply Typed π-calculus

To begin with, we introduce some terminology and notation concerning types. An *assignment* of a type T to a name u is of the form $u : T$. A *type environment* is a finite set of assignments of types

T	$::=$	$V \mid L$	types
V	$::=$	$L \mid \texttt{bool} \mid \texttt{Nat}$	value types
L	$::=$	$\sharp V$	channel types
Γ	$::=$	$\emptyset \mid \Gamma, x : T$	type environments
w	$::=$	$x \mid true, false \mid 0, 1, 2, \cdots$	values
P, Q	$::=$	$\mathsf{0} \mid u(x : V).P \mid \bar{u}w.P \mid P \mid Q \mid P + Q \mid (\nu a : L)P \mid !u(x : V).P$	processes

$$\textsf{T-in}\ \frac{\Gamma \vdash u : \sharp V \quad \Gamma, x : V \vdash P}{\Gamma \vdash u(x : V).P} \qquad \textsf{T-out}\ \frac{\Gamma \vdash u : \sharp V \quad \Gamma \vdash w : V \quad \Gamma \vdash P}{\Gamma \vdash \bar{u}w.P} \qquad \textsf{T-nil}\ \frac{}{\Gamma \vdash \mathsf{0}}$$

$$\textsf{T-par}\ \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q} \qquad \textsf{T-sum}\ \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P + Q} \qquad \textsf{T-res}\ \frac{\Gamma, a : L \vdash P}{\Gamma \vdash (\nu a : L)P}$$

$$\textsf{T-rep}\ \frac{\Gamma \vdash u(x : V).P}{\Gamma \vdash !u(x : V).P}$$

Table 2.4: Processes, types and typing rules of the simply typed π-calculus

to names, where the names in the assignments are all different. We use Γ, Δ to range over type environments. Sometimes we regard a type environment Γ as a partial function from names to types. Thus we write $\Gamma(u)$ for the type assigned to u by Γ, and say that the names of the assignments in Γ are the names on which Γ is defined. We write $dom(\Gamma)$ for the set of names of the assignments in Γ. When $dom(\Gamma) \cap dom(\Delta) = \emptyset$, we write Γ, Δ for the union of Γ and Δ.

A process type judgment $\Gamma \vdash P$ asserts that process P is well typed under the type environment Γ, and a value type judgment $\Gamma \vdash w : V$ that value w has type V under the type assumptions in Γ. We say P is well typed under Γ if the judgment $\Gamma \vdash P$ can be derived by using the typing rules of a given type system.

A *channel* is a name that may be used to engage in communications. The *values* are the objects that can be exchanged along channels. The *channel types* are the types that can be ascribed to channels. The *value types* are the types that can be ascribed to values. In the π-calculus, channel types can be used as value types. In other words, we allow channels to be transmitted as values, and hence allow mobility.

Since our purpose in this section is to introduce the type system of the simply typed π-calculus rather than to propose a pragmatic notation for programming, we adopt an explicitly typed presentation in which every bound name is annotated with a type. The syntax of types and processes as well as the typing rules are shown in Table 2.4. The syntactic distinction between value types and channel types is made by the use of V to range over value types and L over channel types (the letter C is reserved for other use later). However, in typing and operational rules, unless important for the sense we will use only the letters S, T, which stand for arbitrary types. We observe that in the simply typed π-calculus there is only one channel type constructor $\sharp V$. A type assignment $u : \sharp V$ means that u can be used as a channel to carry values of type V. Value types include channel types

and *basic types*, thus both channels and basic values are allowed to be communicated. In the above table, we only display the typing rules for processes. The typing rules for values are the usual ones. For example, we may have the following rules:

$$\frac{}{\Gamma, x : T \vdash x : T} \qquad \frac{}{\Gamma \vdash true : \texttt{bool}} \qquad \frac{}{\Gamma \vdash 0 : \texttt{Nat}} \qquad \ldots$$

For simplicity we only consider two basic types: `bool`, for boolean values, and `Nat`, for natural numbers. Values of basic types are said to be of first-order because, unlike channels, they cannot carry other values. We also assume some basic operations on first-order values. For example, we may use addition $(n+m)$, subtraction $(n-m)$, multiplication $(n*m)$ for `Nat` expressions. To avoid being too specific, we do not give a rigid syntax and typing rules for first-order expressions. We just assume a separate mechanism for evaluating expressions of type `Nat`.

The inert process `0` is well typed under any type environment. The parallel composition and the sum of two processes are well typed if each is well typed in isolation. A process $(\nu a : L)P$ is well typed if P observes the constraints imposed both by the type environment and by the declared type L of the new name a. Note that here L is a channel type. In an input $u(x : V).P$ the subject u should have a channel type, which is compatible with the type of x, moreover, the body P is well typed under the extension of Γ with the type of x. The case for $!u(x : V).P$ is similar. An output $\bar{u}w.P$ is well typed if u has a channel type compatible with that of w, and P itself is well typed.

The transition rules for typed processes are similar to those of the untyped processes (Table 2.3). We just need to annotate bound names with their types. For example, the rule `open` would take this form:

$$\frac{P \xrightarrow{(\nu \widetilde{v}:\widetilde{V})\bar{u}w} P' \quad a \in fn(w) \setminus \{\widetilde{v}, u\}}{(\nu a : L)P \xrightarrow{(\nu \widetilde{v}:\widetilde{V}, a:L)\bar{u}w} P'}$$

Given the operational semantics for typed processes, we can prove the *subject reduction* property, which represents the fact that type judgments are invariant under computation. In particular, if $\Gamma \vdash P$ and $P \xrightarrow{\tau} P'$ then it holds that $\Gamma \vdash P'$.

2.2.6 Subtyping

Subtyping is a preorder on types. If S is a subtype of T then all operations available on values of type T are also available on values of type S; therefore an expression of type S can always replace an expression of type T. The possibility of having operations that work on all subtypes of a given type is a major advantage of subtyping in a programming language.

We shall write *subtype judgments* in the form $S <: T$, which asserts that S is a subtype of T (equally T is a supertype of S). A type construct is *covariant* in its i-th argument if the construct preserves the direction of subtyping in that argument. Dually, a type construct is *contravariant* in its i-th argument if the construct inverts the direction of subtyping in that argument. A type construct is *invariant* in its i-th argument if it is both covariant and contravariant in that argument.

We now refine channel types by distinguishing between the capabilities of using a channel in input or in outputs. For this we introduce the types $\texttt{i}V$ and $\texttt{o}V$, with the intended meanings: $\texttt{i}V$

$$\textsf{S-ref}\ \frac{}{T <: T} \qquad \textsf{S-tra}\ \frac{T <: T' \quad T' <: T''}{T <: T''} \qquad \textsf{S-bi}\ \frac{}{\sharp T <: \mathtt{i}T}$$

$$\textsf{S-bo}\ \frac{}{\sharp T <: \mathtt{o}T} \qquad \textsf{S-ii}\ \frac{T <: T'}{\mathtt{i}T <: \mathtt{i}T'} \qquad \textsf{S-oo}\ \frac{T <: T'}{\mathtt{o}T' <: \mathtt{o}T}$$

$$\textsf{S-bb}\ \frac{T <: T' \quad T' <: T}{\sharp T <: \sharp T'}$$

$$\textsf{T-ins}\ \frac{\Gamma \vdash u : \mathtt{i}V \quad \Gamma, x : V \vdash P}{\Gamma \vdash u(x : V).P} \qquad \textsf{T-outs}\ \frac{\Gamma \vdash u : \mathtt{o}V \quad \Gamma \vdash w : V \quad \Gamma \vdash P}{\Gamma \vdash \bar{u}w.P}$$

$$\textsf{subsum}\ \frac{\Gamma \vdash u : T \quad T <: T'}{\Gamma \vdash u : T'}$$

(rules T-ins and T-outs replace T-in and T-out respectively)

Table 2.5: Additional rules on subtyping

is the type of a channel that can be used only in input and that carries values of type V; similar for $\mathtt{o}V$ w.r.t. output. By extending the simply typed π-calculus with the two capability types, we obtain the *simply typed π-calculus with subtyping*. For this, we redefine channel types as

$$L ::= \sharp V \mid \mathtt{i}V \mid \mathtt{o}V \qquad \text{channel types}$$

and use the additional rules reported in Table 2.5.

We briefly comment on the subtyping rules. The rules S-ref and S-tra show that $<:$ is a preorder. The axioms S-bi and S-bo show that a name of all capabilities can be used in places where only the input or only the output capability is required. Rule S-ii says that `i` is a covariant construct, while S-oo says that `o` is a contravariant construct. Finally S-bb shows that $\sharp$ is invariant.

The typing rules T-ins and T-outs are similar to the rules T-in and T-out, except that now the subject of a prefix is checked to have the appropriate input or output capability. The old rules are derivable from the new ones.

Chapter 3

Axiomatisations for Probabilistic Processes

In this chapter we study a process calculus which combines both nondeterministic and probabilistic behaviour in the style of Segala and Lynch's probabilistic automata. We consider various strong and weak behavioural equivalences, and we provide complete axiomatisations for finite-state processes, restricted to guarded recursion in the case of the weak equivalences. We conjecture that in the general case of unguarded recursion the "natural" weak equivalences are undecidable.

The contents of this chapter are organized as follows. First we briefly recall some basic concepts and definitions about probabilistic distributions. In Section 3.2 we introduce a probabilistic process calculus, with its syntax and operational semantics. In Section 3.3 we define the four behavioural equivalences we are interested in, and we extend the "bisimulation up to" techniques of [Mil89a] to the probabilistic setting. These techniques are extensively used for the proofs of soundness of some axioms, especially in the case of the weak equivalences. In Sections 3.4 and 3.5 we give complete axiomatisations for the strong equivalences and for the weak equivalences respectively, restricted to guarded expressions in the second case. Section 3.6 gives complete axiomatisations for the four equivalences in the case of the finite fragment of the language. The interest of this section is that we use different and much simpler proof techniques. At last we conclude with some discussions about the conjecture mentioned above.

3.1 Probabilistic Distributions

Let M be a set. A function $\eta : M \mapsto [0,1]$ is called a *discrete probability distribution*, or *distribution* for short, on M if the *support* of η, defined as $spt(\eta) = \{x \in M \mid \eta(x) > 0\}$, is finite or countably infinite and $\sum_{x \in M} \eta(x) = 1$. If η is a distribution with finite support and $N \subseteq spt(\eta)$ we use the set $\{(s_i : \eta(s_i))\}_{s_i \in N}$ to enumerate the probability associated with each element of N. To manipulate

the set we introduce the operator $\uplus$ defined as follows.

$$\{(s_i : p_i)\}_{i\in I} \uplus \{(s : p)\} \stackrel{\text{def}}{=} \begin{cases} \{(s_i : p_i)\}_{i\in I\setminus j} \cup \{s_j : (p_j + p)\} & \text{if } s = s_j \text{ for some } j \in I \\ \{(s_i : p_i)\}_{i\in I} \cup \{(s : p)\} & \text{otherwise.} \end{cases}$$

$$\{(s_i : p_i)\}_{i\in I} \uplus \{(t_j : p_j)\}_{j\in 1..n} \stackrel{\text{def}}{=} (\{(s_i : p_i)\}_{i\in I} \uplus \{(t_1 : p_1)\}) \uplus \{(t_j : p_j)\}_{j\in 2..n}$$

Given some distributions $\eta_1, ..., \eta_n$ on S and some real numbers $r_1, ..., r_n \in [0, 1]$ with $\sum_{i\in 1..n} r_i = 1$, we define the *convex combination* $r_1\eta_1 + ... + r_n\eta_n$ of $\eta_1, ..., \eta_n$ to be the distribution η such that $\eta(s) = \sum_{i\in 1..n} r_i\eta_i(s)$, for each $s \in S$.

Lemma 3.1 *If η is a convex combination of $\eta_1, ..., \eta_n$ and each η_i $(i \leq n)$ is a convex combination of some distributions $\theta_1, ..., \theta_m$ on S, then η is also a convex combination of $\theta_1, ..., \theta_m$.*

Proof: Suppose that $\eta = r_1\eta_1 + ... + r_n\eta_n$ with $\sum_{i\in 1..n} r_i = 1$, and that $\eta_i = p_{i1}\theta_1 + ... + p_{im}\theta_m$ with $\sum_{j\in 1..m} p_{ij} = 1$, for all $i \leq n$. For each $s \in S$, we have that

$$\eta(s) = \sum_{i\in 1..n} r_i\eta_i(s) = \sum_{i\in 1..n} r_i \sum_{j\in 1..m} p_{ij}\theta_j(s) = \sum_{j\in 1..m} \sum_{i\in 1..n} r_i p_{ij}\theta_j(s).$$

So η is the convex combination $(\sum_{i\in 1..n} r_i p_{i1})\theta_1 + ... + (\sum_{i\in 1..n} r_i p_{im})\theta_m$. Indeed it can be checked that $\sum_{j\in 1..m} \sum_{i\in 1..n} r_i p_{ij} = 1$. □

3.2 A Probabilistic Process Calculus

The set *Var* of process variables and the set $\mathcal{L}$ of labels are defined as in Section 2.1. We let ξ range over the set $\mathit{Var} \cup \mathcal{L}$. The class of expressions $\mathcal{E}$ is defined by the following syntax:

$$E, F ::= \bigoplus_{i\in 1..n} p_i\ell_i.E_i \;\Big|\; \sum_{i\in 1..m} E_i \;\Big|\; X \;\Big|\; \mu_X E$$

Here $\bigoplus_{i\in 1..n} p_i\ell_i.E_i$ stands for a *probabilistic choice* operator, where the p_i's represent positive probabilities, i.e., they satisfy $p_i \in (0, 1]$ and $\sum_{i\in 1..n} p_i = 1$. When $n = 0$ we abbreviate the probabilistic choice as $\mathsf{0}$; when $n = 1$ we abbreviate it as $\ell_1.E_1$. Sometimes we are interested in certain branches of the probabilistic choice; in this case we write $\bigoplus_{i\in 1..n} p_i\ell_i.E_i$ as $p_1\ell_1.E_1 \oplus \cdots \oplus p_n\ell_n.E_n$ or $(\bigoplus_{i\in 1..(n-1)} p_i\ell_i.E_i) \oplus p_n\ell_n.E_n$ where $\bigoplus_{i\in 1..(n-1)} p_i\ell_i.E_i$ abbreviates (with a slight abuse of notation) $p_1\ell_1.E_1 \oplus \cdots \oplus p_{n-1}\ell_{n-1}.E_{n-1}$. The second construction $\sum_{i\in 1..m} E_i$ stands for *indexed nondeterministic choice*, and occasionally we may write it as $E_1 + ... + E_m$.

Definition 3.2 *The variable X is* weakly guarded *(resp.* guarded*) in E if every free occurrence of X in E occurs within some subexpression $\ell.F$ (resp. $\ell.F$ but $\ell \neq \tau$), otherwise X is* weakly unguarded *(resp.* unguarded*) in E.*

The operational semantics of an expression E is defined as a probabilistic automaton whose states are the expressions reachable from E and the transition relation is defined by the axioms and

var	$X \to \vartheta(X)$	psum	$\bigoplus_{i\in 1..n} p_i\ell_i.E_i \to \biguplus_{i\in 1..n}\{(\ell_i, E_i : p_i)\}$
rec	$\dfrac{E\{\mu_X E/X\} \to \eta}{\mu_X E \to \eta}$	nsum	$\dfrac{E_j \to \eta}{\Sigma_{i\in 1..m} E_i \to \eta}$ for some $j \in 1..m$

Table 3.1: Strong transitions

inference rules in Table 3.1, where $E \to \eta$ describes a transition that leaves from E and leads to a distribution η over $(Var \cup \mathcal{L}) \times \mathcal{E}$. We shall use $\vartheta(X)$ for the special distribution $\{(X, 0 : 1)\}$. It is evident that $E \to \vartheta(X)$ iff X is weakly unguarded in E.

The behaviour of each expression can be visualized by a transition graph. For instance, the expression $(\frac{1}{2}a \oplus \frac{1}{2}b) + (\frac{1}{3}a \oplus \frac{2}{3}c) + (\frac{1}{2}b \oplus \frac{1}{2}c)$ exhibits the behaviour drawn in diagram (5) of Figure 1.1.

As in [BS01], we define the notion of *combined transition* as follows: $E \to_c \eta$ if there exists a collection $\{\eta_i, r_i\}_{i\in 1..n}$ of distributions and probabilities such that $\sum_{i\in 1..n} r_i = 1$, $\eta = r_1\eta_1 + ... + r_n\eta_n$ and $E \to \eta_i$, for each $i \in 1..n$.

Lemma 3.3 *If $\eta = r_1\eta_1 + ... + r_n\eta_n$ and $E \to_c \eta_i$ for each $i \leq n$, then $E \to_c \eta$.*

Proof: Suppose that for each $i \leq n$, η_i is a convex combination of $\eta_{i1}, ..., \eta_{im_i}$, with $E \to \eta_{ij}$ for $j \leq m_i$. Let

$$\bigcup_{i\in 1..n} \{\eta_{i1}, ..., \eta_{im_i}\} = \{\theta_1, ..., \theta_m\}$$

Clearly each η_i $(i \leq n)$ is also a convex combination of $\theta_1, ..., \theta_m$. It follows from Lemma 3.1 that η is a convex combination of $\theta_1, ..., \theta_m$. Note that $E \to \theta_j$ for each $j \leq m$. Therefore we have the result that $E \to_c \eta$. □

We now introduce the notion of weak transitions, which generalizes the notion of *finitary weak transitions* in SPA [Sto02] to the setting of PA. First we discuss the intuition behind it. Given an expression E, if we unfold its transition graph, we get a finitely branching tree. By cutting away all but one alternative in case of several nondeterministic candidates, we are left with a subtree with only probabilistic branches. A weak transition of E is a finite subtree of this kind, called *weak transition tree*, such that in any path from the root to a leaf there is at most one visible action. For example, let E be the expression $\mu_X(\frac{1}{2}a \oplus \frac{1}{2}\tau.X)$. It is represented by the transition graph displayed in Diagram (1) of Figure 3.1. After one unfolding, we get Diagram (2) which represents the weak transition $E \Rightarrow \eta$, where $\eta = \{(a, 0 : \frac{3}{4}), (\tau, E : \frac{1}{4})\}$.

Formally, weak transitions are defined by the rules in Table 3.2. Rule wea1 says that a weak transition tree starts from a bundle of labelled arrows derived from a strong transition. The meaning of Rule wea2 is as follows. Given two expressions E, F and their weak transition trees $tr(E), tr(F)$, if F is a leaf of $tr(E)$ and there is no visible action in $tr(F)$, then we can extend $tr(E)$ with $tr(F)$ at node F. If F_j is a leaf of $tr(F)$ then the probability of reaching F_j from E is pq_j, where p and q_j are the probabilities of reaching F from E, and F_j from F, respectively. Rule wea3 is similar to Rule

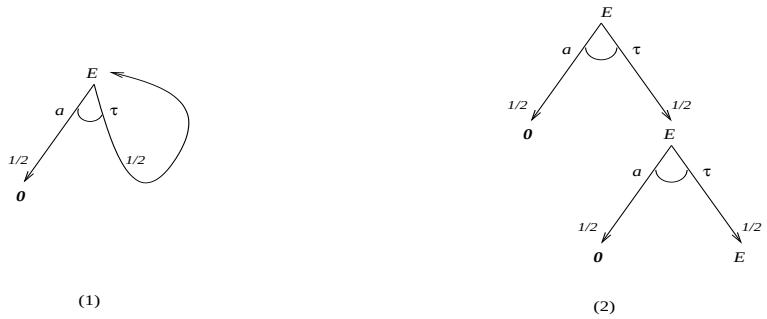

Figure 3.1: A weak transition

$$\textsf{wea1}\ \frac{E \to \eta}{E \Rightarrow \eta}$$

$$\textsf{wea2}\ \frac{E \Rightarrow \{(\ell_i, E_i : p_i)\}_i \uplus \{(\ell, F : p)\} \quad F \Rightarrow \{(\tau, F_j : q_j)\}_j}{E \Rightarrow \{(\ell_i, E_i : p_i)\}_i \uplus \{(\ell, F_j : pq_j)\}_j}$$

$$\textsf{wea3}\ \frac{E \Rightarrow \{(\ell_i, E_i : p_i)\}_i \uplus \{(\tau, F : p)\} \quad F \Rightarrow \{(h_j, F_j : q_j)\}_j}{E \Rightarrow \{(\ell_i, E_i : p_i)\}_i \uplus \{(h_j, F_j : pq_j)\}_j}$$

$$\textsf{wea4}\ \frac{E \Rightarrow \{(\tau, E_i : p_i)\}_i \quad \forall i, E_i \Rightarrow \vartheta(X)}{E \Rightarrow \vartheta(X)}$$

Table 3.2: Weak transitions

wea2, with the difference that we can have visible actions in $tr(F)$, but not in the path from E to F. Rule wea4 allows to construct weak transitions to unguarded variables. Note that if $E \Rightarrow \vartheta(X)$ then X is unguarded in E.

For any expression E, we use $\delta(E)$ for the unique distribution $\{(\tau, E : 1)\}$, called the *virtual distribution* of E. We define a *weak combined transition*: $E \stackrel{\epsilon}{\Rightarrow}_c \eta$ if there exists a collection $\{\eta_i, r_i\}_{i \in 1..n}$ of distributions and probabilities such that $\sum_{i \in 1..n} r_i = 1$, $\eta = r_1\eta_1 + ... + r_n\eta_n$ and for each $i \in 1..n$, either $E \Rightarrow \eta_i$ or η_i is $\delta(E)$. We write $E \Rightarrow_c \eta$ if every component of η is derived from a weak transition, namely, $E \Rightarrow \eta_i$ for all $i \leq n$. Note in particular that for any expression E we can derive a virtual distribution by $E \stackrel{\epsilon}{\Rightarrow}_c \delta(E)$, but $E \not\Rightarrow_c \delta(E)$.

Lemma 3.4 *1. If $E \stackrel{\epsilon}{\Rightarrow}_c \eta$ then $\tau.E \Rightarrow_c \eta$;*

2. If $E \stackrel{\epsilon}{\Rightarrow}_c \vartheta(X)$ then $E \Rightarrow \vartheta(X)$.

Proof: The first clause is easy to show. Let us consider the second one. If $\vartheta(X)$ is a convex combination of $\eta_1, .., \eta_n$ and $E \Rightarrow \eta_i$ for all $i \in 1..n$, then each η_i must assign probability 1 to $(X, \mathsf{0})$, thus $\eta_i = \vartheta(X)$. □

Lemma 3.5 *1. If $\eta = r_1\eta_1 + ... + r_n\eta_n$ and $E \stackrel{\epsilon}{\Rightarrow}_c \eta_i$ for each $i \leq n$, then $E \stackrel{\epsilon}{\Rightarrow}_c \eta$.*

2. If $\eta = r_1\eta_1 + ... + r_n\eta_n$ *and* $E \Rightarrow_c \eta_i$ *for each* $i \leq n$*, then* $E \Rightarrow_c \eta$*.*

Proof: Similar to the proof of Lemma 3.3. □

3.3 Behavioural Equivalences

In this section we define four behavioural equivalences, namely, strong bisimulation, strong probabilistic bisimulation, divergency-sensitive equivalence and observational equivalence. We also introduce a probabilistic version of "bisimulation up to" techniques to show some interesting properties of the behavioural equivalences.

To define behavioural equivalences in probabilistic process calculi, it is customary to consider equivalence of distributions with respect to equivalence relations on processes.

3.3.1 Equivalence of Distributions

If η is a distribution on $M_1 \times M_2$, $s \in M_1$ and $N \subseteq M_2$, we write $\eta(s, N)$ for $\sum_{t\in N} \eta(s, t)$. We lift an equivalence relation on $\mathcal{E}$ to a relation between distributions over $(Var \cup \mathcal{L}) \times \mathcal{E}$ in the following way.

Definition 3.6 *Given two distributions* η_1 *and* η_2 *over* $(Var \cup \mathcal{L}) \times \mathcal{E}$*, we say that they are equivalent w.r.t. an equivalence relation* $\mathcal{R}$ *on* $\mathcal{E}$*, written* $\eta_1 \equiv_{\mathcal{R}} \eta_2$*, if*

$$\forall N \in \mathcal{E}/\mathcal{R}, \forall \xi \in Var \cup \mathcal{L},\ \eta_1(\xi, N) = \eta_2(\xi, N).$$

Lemma 3.7 *Given three distributions* η_1, η_2, η_3 *and an equivalence relation* $\mathcal{R}$*, if* $\eta_1 \equiv_{\mathcal{R}} \eta_2$ *and* $\eta_2 \equiv_{\mathcal{R}} \eta_3$ *then* $\eta_1 \equiv_{\mathcal{R}} \eta_3$*.*

Proof: Straightforward by definition. □

The above lemma says that $\equiv_{\mathcal{R}}$ is transitive. It follows immediately that $\equiv_{\mathcal{R}}$ is an equivalence relation. Next we report two fundamental lemmas that underpin many other results in the subsequent sections.

Lemma 3.8 *If* $\eta_1 \equiv_{\mathcal{R}_1} \eta_2$ *and* $\mathcal{R}_1 \subseteq \mathcal{R}_2$ *then* $\eta_1 \equiv_{\mathcal{R}_2} \eta_2$*.*

Proof: Let $N \in \mathcal{E}/\mathcal{R}_2$. Since $\mathcal{R}_1$ is contained in $\mathcal{R}_2$, we know that N is the disjoint union of a family of sets $\{N_i\}_{i\in I}$ such that $N_i \in \mathcal{E}/\mathcal{R}_1$ for each $i \in I$. It follows from $\eta_1 \equiv_{\mathcal{R}_1} \eta_2$ that

$$\forall i \leq n, \forall \xi \in Var \cup \mathcal{L},\ \eta_1(\xi, N_i) = \eta_2(\xi, N_i).$$

Therefore we have

$$\eta_1(\xi, N) = \sum_{i\in I} \eta_1(\xi, N_i) = \sum_{i\in I} \eta_2(\xi, N_i) = \eta_2(\xi, N).$$

□

Lemma 3.9 *Let* $\eta = r_1\eta_1 + ... + r_n\eta_n$ *and* $\eta' = r_1\eta'_1 + ... + r_n\eta'_n$ *with* $\sum_{i\in 1..n} r_i = 1$*. If* $\eta_i \equiv_{\mathcal{R}} \eta'_i$ *for each* $i \leq n$*, then* $\eta \equiv_{\mathcal{R}} \eta'$*.*

Proof: For any $N \in \mathcal{E}/\mathcal{R}$ and $\xi \in Var \cup \mathcal{L}$, we have

$$\eta(\xi, N) = \sum_{i \in 1..n} r_i \eta_i(\xi, N) = \sum_{i \in 1..n} r_i \eta'_i(\xi, N) = \eta'(\xi, N).$$

Therefore $\eta \equiv_{\mathcal{R}} \eta'$ by definition. □

3.3.2 Behavioural Equivalences

Strong bisimulation is defined by requiring equivalence of distributions at every step. Because of the way equivalence of distributions is defined, we need to restrict to bisimulations which are equivalence relations.

Definition 3.10 *An equivalence relation $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{E}$ is a* strong bisimulation *if $E\ \mathcal{R}\ F$ implies:*

- *whenever $E \to \eta_1$, there exists η_2 such that $F \to \eta_2$ and $\eta_1 \equiv_{\mathcal{R}} \eta_2$.*

Two expressions E, F are strong bisimilar*, written $E \sim F$, if there exists a strong bisimulation $\mathcal{R}$ s.t. $E\ \mathcal{R}\ F$.*

If we allow a strong transition to be matched by a strong combined transition, then we get a relation slightly coarser than strong bisimulation.

Definition 3.11 *An equivalence relation $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{E}$ is a* strong probabilistic bisimulation *if $E\ \mathcal{R}\ F$ implies:*

- *whenever $E \to \eta_1$, there exists η_2 such that $F \to_c \eta_2$ and $\eta_1 \equiv_{\mathcal{R}} \eta_2$.*

We write $E \sim_c F$, if there exists a strong probabilistic bisimulation $\mathcal{R}$ s.t. $E\ \mathcal{R}\ F$.

To show that $\sim_c$ is an equivalence relation, we need the following lemma, which can be used to prove the transitivity of $\sim_c$.

Lemma 3.12 *If $E \sim_c F$ then whenever $E \to_c \eta$, there exists η' such that $F \to_c \eta'$ and $\eta \equiv_{\sim_c} \eta'$.*

Proof: Suppose that $\eta = r_1\eta_1 + ... + r_n\eta_n$ and $E \to \eta_i$ for $i \leq n$. Since $E \sim_c F$, there exists η'_i for each $i \leq n$ such that $F \to_c \eta'_i$ and $\eta_i \equiv_{\sim_c} \eta'_i$. Now let $\eta' = r_1\eta'_1 + ... + r_n\eta'_n$. By Lemma 3.3 we know that $F \to_c \eta'$. By Lemma 3.9 it holds that $\eta \equiv_{\sim_c} \eta'$. □

We now consider the case of the weak bisimulation. The definition of weak bisimulation for PA is not at all straightforward. In fact, the "natural" weak version of Definition 3.10 would be the following one.

Definition (Tentative). *An equivalence relation $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{E}$ is a* weak bisimulation *if $E\ \mathcal{R}\ F$ implies:*

- *whenever $E \to \eta_1$, then either $\eta_1 \equiv_{\mathcal{R}} \delta(F)$ or there exists some η_2 such that $F \Rightarrow \eta_2$ and $\eta_1 \equiv_{\mathcal{R}} \eta_2$.*

E and F are weak bisimilar*, written $E \asymp F$, whenever there exists a weak bisimulation $\mathcal{R}$ s.t. $E\ \mathcal{R}\ F$.*

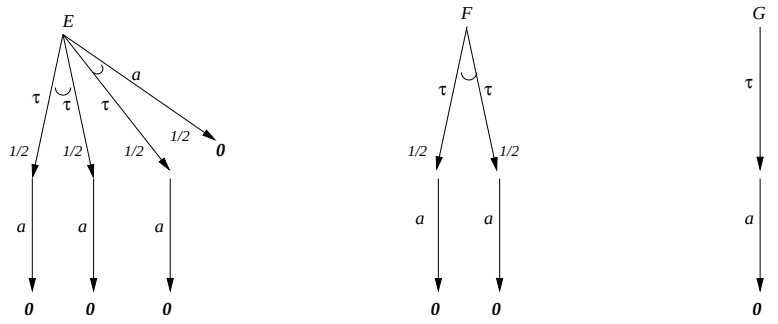

Figure 3.2: Transition graphs of E, F and G

Unfortunately the above definition is incorrect because it defines a relation which is not transitive. That is, there exist E, F and G with $E \asymp F$ and $F \asymp G$ but $E \not\asymp G$. For example, consider the following expressions (their transition graphs are displayed in Figure 3.2) and relations:

$$\begin{array}{rcl} E & \stackrel{\text{def}}{=} & (\frac{1}{2}\tau.a \oplus \frac{1}{2}\tau.a) + (\frac{1}{2}\tau.a \oplus \frac{1}{2}a) \\ F & \stackrel{\text{def}}{=} & \frac{1}{2}\tau.a \oplus \frac{1}{2}\tau.a \\ G & \stackrel{\text{def}}{=} & \tau.a \end{array}$$

$$\begin{array}{rcl} \mathcal{R}_1 & \stackrel{\text{def}}{=} & \{(E,F),(F,E),(E,E),(F,F),(a,a),(\mathsf{0},\mathsf{0})\} \\ \mathcal{R}_2 & \stackrel{\text{def}}{=} & \{(F,G),(G,F),(F,F),(G,G),(a,a),(\mathsf{0},\mathsf{0})\} \end{array}$$

It can be checked that $\mathcal{R}_1$ and $\mathcal{R}_2$ are weak bisimulations according to the tentative definition. However we have $E \not\asymp G$. To see this, consider the transition $E \to \eta$, where $\eta = \{(\tau, a : \frac{1}{2}), (a, \mathsf{0} : \frac{1}{2})\}$. From G there are only two possible weak transitions $G \Rightarrow \eta_1$ and $G \Rightarrow \eta_2$ with $\eta_1 = \{(\tau, a : 1)\}$ and $\eta_2 = \{(a, \mathsf{0} : 1)\}$. Now, among the three distributions $\delta(G)$, η_1 and η_2, none is equivalent to η. Therefore, E and G are not bisimilar. Nevertheless, if we consider the weak combined transition: $G \Rightarrow_c \eta'$ where $\eta' = \frac{1}{2}\eta_1 + \frac{1}{2}\eta_2$, we observe that $\eta \equiv \eta'$.

The above example suggests that for a "good" definition of weak bisimulation it is necessary to use combined transitions. So we cannot give a weak variant of Definition 3.10, but only of Definition 3.11, called weak probabilistic bisimulation.

Definition 3.13 *An equivalence relation $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{E}$ is a* weak probabilistic bisimulation *if $E\ \mathcal{R}\ F$ implies:*

- *whenever $E \to \eta_1$, there exists η_2 such that $F \stackrel{\epsilon}{\Rightarrow}_c \eta_2$ and $\eta_1 \equiv_{\mathcal{R}} \eta_2$.*

We write $E \approx F$ whenever there exists a weak probabilistic bisimulation $\mathcal{R}$ s.t. $E\ \mathcal{R}\ F$.

The following lemma is indispensable to show the transitivity of $\approx$.

Lemma 3.14 *Let $\mathcal{R}$ be a weak probabilistic bisimulation. If $E\ \mathcal{R}\ F$ then whenever $E \stackrel{\epsilon}{\Rightarrow}_c \eta$, there exists η' such that $F \stackrel{\epsilon}{\Rightarrow}_c \eta'$ and $\eta \equiv_{\mathcal{R}} \eta'$.*

Proof: See Appendix A.1. □

Lemma 3.15 *Let $\mathcal{R} = \bigcup_i \{\mathcal{R}_i \mid \mathcal{R}_i$ is a weak probabilistic bisimulation$\}$. Then the equivalence closure of $\mathcal{R}$, written $\mathcal{R}^*$, is a probabilistic weak bisimulation.*

Proof: If $E\ \mathcal{R}^*\ F$ then there exist some weak probabilistic bisimulations $\eta_1, ..., \eta_n$ and some expressions $E_0, .., E_n$ such that $E \equiv E_0, E_n \equiv F$, and for all i with $0 \leq i < n$, we have $E_i\ \mathcal{R}_i\ E_{i+1}$. If $E \rightarrow \eta_0$ then there exists η_1 such that $E_1 \stackrel{\epsilon}{\Rightarrow}_c \eta_1$ and $\eta_0 \equiv_{\mathcal{R}_0} \eta_1$. For all i with $1 \leq i < n$, by Lemma 3.14 there exists η_{i+1} such that $E_{i+1} \stackrel{\epsilon}{\Rightarrow}_c \eta_{i+1}$ and $\eta_i \equiv_{\mathcal{R}_i} \eta_{i+1}$. By Lemma 3.8 and the transitivity of $\equiv_{\mathcal{R}^*}$ it holds that $\eta_0 \equiv_{\mathcal{R}^*} \eta_n$. □

Because of the above lemma we can equivalently express $\approx$ as $\mathcal{R}^*$, which is the biggest weak probabilistic bisimulation.

As usual, observational equivalence is defined in terms of weak probabilistic bisimulation.

Definition 3.16 *Two expressions E, F are* observationally equivalent*, written $E \simeq F$, if*

1. *whenever $E \rightarrow \eta_1$, there exists η_2 such that $F \Rightarrow_c \eta_2$ and $\eta_1 \equiv_\approx \eta_2$;*
2. *whenever $F \rightarrow \eta_2$, there exists η_1 such that $E \Rightarrow_c \eta_1$ and $\eta_1 \equiv_\approx \eta_2$.*

The following lemma plays the same role as Lemma 3.14, and the proof of the former is similar to that of the latter. Then it is evident that $\simeq$ is an equivalence relation.

Lemma 3.17 *Suppose $E \simeq F$. If $E \Rightarrow_c \eta$ then there exists η' s.t. $F \Rightarrow_c \eta'$ and $\eta \equiv_\approx \eta'$.*

Often observational equivalence is criticised for being insensitive to divergency. We therefore introduce a variant which does not have this shortcoming.

Definition 3.18 *An equivalence relation $\mathcal{R} \subseteq \mathcal{E} \times \mathcal{E}$ is a* divergency-sensitive equivalence *if $E\ \mathcal{R}\ F$ implies:*

- *whenever $E \rightarrow \eta_1$, there exists η_2 such that $F \Rightarrow_c \eta_2$ and $\eta_1 \equiv_\mathcal{R} \eta_2$.*

We write $E \eqsim F$ whenever there exists a divergency-sensitive equivalence $\mathcal{R}$ s.t. $E\ \mathcal{R}\ F$.

Here the difference from Definition 3.13 is that we use the transition $F \Rightarrow_c \eta_2$ in place of $F \stackrel{\epsilon}{\Rightarrow}_c \eta_2$ to match a strong transition. In other words, F cannot stay idle; it must make some real move.

It is easy to see that $\eqsim$ lies between $\sim_c$ and $\simeq$. For example, we have that $\mu_X(\tau.X + a)$ and $\tau.a$ are related by $\simeq$ but not by $\eqsim$ (this shows also that $\eqsim$ is sensitive to divergency), while $\tau.a$ and $\tau.a + a$ are related by $\eqsim$ but not by $\sim_c$. Further, $\tau.a$ and a are not related by $\eqsim$ because the transition $\tau.a \rightarrow \{\tau, a : 1\}$ cannot be matched up by $a \Rightarrow_c \{a, \mathsf{0} : 1\}$. So $\eqsim$ does not simply detect divergency, it counts internal moves in a certain sense.

One can check that all the relations defined above (except for $\asymp$) are indeed equivalence relations and we have the inclusion ordering: $\sim \subsetneq \sim_c \subsetneq \eqsim \subsetneq \simeq \subsetneq \approx$.

3.3.3 Probabilistic "Bisimulation up to" Techniques

In the classical process algebra, the conventional approach to show $E \sim F$, for some expressions E, F, is to construct a binary relation $\mathcal{R}$ which includes the pair (E, F), and then to check that $\mathcal{R}$

is a bisimulation. This approach can still be used in probabilistic process algebra, but things are more complicated because of the extra requirement that $\mathcal{R}$ must be an equivalence relation. For example we cannot use some standard set-theoretic operators to construct $\mathcal{R}$, because, even if $\mathcal{R}_1$ and $\mathcal{R}_2$ are equivalences, $\mathcal{R}_1\mathcal{R}_2$ and $\mathcal{R}_1 \cup \mathcal{R}_2$ may not be equivalences.

To avoid the restrictive condition and at the same time to reduce the size of the relation $\mathcal{R}$, we introduce the probabilistic version of "bisimulation up to" techniques, whose usefulness will be exhibited in the next section.

In the following definitions, for a binary relation $\mathcal{R}$ we denote the relation $(\mathcal{R} \cup \sim)^*$ by $\mathcal{R}_\sim$. Similar for other notations such as $\mathcal{R}_\approx$ and $\mathcal{R}_\simeq$.

Definition 3.19 *A binary relation $\mathcal{R}$ is a* strong bisimulation up to $\sim$ *if $E\ \mathcal{R}\ F$ implies:*

1. *whenever $E \rightarrow \eta_1$, there exists η_2 such that $F \rightarrow \eta_2$ and $\eta_1 \equiv_{\mathcal{R}_\sim} \eta_2$;*
2. *whenever $F \rightarrow \eta_2$, there exists η_1 such that $E \rightarrow \eta_1$ and $\eta_1 \equiv_{\mathcal{R}_\sim} \eta_2$.*

A strong bisimulation up to $\sim$ is not necessarily an equivalence relation. It is just an ordinary binary relation included in $\sim$, as shown by the next proposition.

Proposition 3.20 *If $\mathcal{R}$ is a strong bisimulation up to $\sim$, then $\mathcal{R}_\sim$ is a strong bisimulation and $\mathcal{R} \subseteq \sim$.*

Proof: If $E\ \mathcal{R}_\sim\ F$ then there exist some expressions $E_0, ..., E_n$ such that $E \equiv E_0, E_n \equiv F$, and for all i with $1 \leq i < n$ we have either $E_i \sim E_{i+1}$ or $E_i\ \mathcal{R}\ E_{i+1}$. Suppose that $E_i \rightarrow \eta_i$. If $E_i\ \mathcal{R}\ E_{i+1}$ then there exists η_{i+1} such that $E_{i+1} \rightarrow \eta_{i+1}$ and $\eta_i \equiv_{\mathcal{R}_\sim} \eta_{i+1}$. If $E_i \sim E_{i+1}$ then there exists η_{i+1} such that $E_{i+1} \rightarrow \eta_{i+1}$ and $\eta_i \equiv_\sim \eta_{i+1}$. Since $\sim \subseteq \mathcal{R}_\sim$, we know from Lemma 3.8 that $\eta_i \equiv_{\mathcal{R}_\sim} \eta_{i+1}$. So in both cases we have matching transitions and $\eta_i \equiv_{\mathcal{R}_\sim} \eta_{i+1}$, which implies $\eta_0 \equiv_{\mathcal{R}_\sim} \eta_n$ by Lemma 3.7. Therefore $\mathcal{R}_\sim$ is a strong bisimulation, i.e., $\mathcal{R}_\sim \subseteq \sim$. Since $\mathcal{R} \subseteq \mathcal{R}_\sim$, it follows that $\mathcal{R} \subseteq \sim$. □

One can also define a strong probabilistic bisimulation up to $\sim_c$ relation and show that it is included in $\sim_c$.

Lemma 3.21 *Let $\mathcal{R}$ be a strong probabilistic bisimulation up $\sim_c$. If $E\ \mathcal{R}\ F$ then whenever $E \rightarrow_c \eta$, there exists η' such that $F \rightarrow_c \eta'$ and $\eta \equiv_{\mathcal{R}_{\sim_c}} \eta'$.*

Proof: Similar to the proof of Lemma 3.12. □

Proposition 3.22 *If $\mathcal{R}$ is a strong probabilistic bisimulation up to $\sim_c$, then $\mathcal{R} \subseteq \sim_c$.*

Proof: Similar to the proof of Proposition 3.20. The only difference is that when matching transitions, we use Lemma 3.21 instead of directly applying the definitions. □

For weak probabilistic bisimulation, the "up to" relations can be defined as well, but we need to be careful.

Definition 3.23 *A binary relation $\mathcal{R}$ is a* weak probabilistic bisimulation up to $\approx$ *if $E\ \mathcal{R}\ F$ implies:*

1. *whenever* $E \Rightarrow \eta_1$, *there exists* η_2 *such that* $F \stackrel{\epsilon}{\Rightarrow}_c \eta_2$ *and* $\eta_1 \equiv_{\mathcal{R}\approx} \eta_2$;

2. *whenever* $F \Rightarrow \eta_2$, *there exists* η_1 *such that* $E \stackrel{\epsilon}{\Rightarrow}_c \eta_1$ *and* $\eta_1 \equiv_{\mathcal{R}\approx} \eta_2$.

In the above definition, we are not able to replace the first double arrow in each clause by a simple arrow. Otherwise, the resulting relation is not included in $\approx$. A counterexample is $\mathcal{R} = \{(\tau.a.\mathsf{0}, \mathsf{0})\}$, as in the nonprobabilistic setting [SM92].

Proposition 3.24 *If* $\mathcal{R}$ *is a weak probabilistic bisimulation up to* $\approx$, *then* $\mathcal{R} \subseteq \approx$.

Proof: Similar to the proof of Proposition 3.22. □

Definition 3.25 *A binary relation* $\mathcal{R}$ *is an* observational equivalence up to $\simeq$ *if* $E\ \mathcal{R}\ F$ *implies:*

1. *whenever* $E \Rightarrow \eta_1$, *there exists* η_2 *such that* $F \Rightarrow_c \eta_2$ *and* $\eta_1 \equiv_{\mathcal{R}\approx} \eta_2$;

2. *whenever* $F \Rightarrow \eta_2$, *there exists* η_1 *such that* $E \Rightarrow_c \eta_1$ *and* $\eta_1 \equiv_{\mathcal{R}\approx} \eta_2$.

As expected, observational equivalence up to $\simeq$ is useful because of the following property.

Proposition 3.26 *If* $\mathcal{R}$ *is an observational equivalence up to* $\simeq$, *then* $\mathcal{R} \subseteq \simeq$.

Proof: Note that if $\mathcal{R}$ is an observational equivalence up to $\simeq$, then it is also a weak probabilistic bisimulation up to $\approx$. So $\mathcal{R}_{\approx} \subseteq \approx$ and it becomes evident that $\mathcal{R} \subseteq \simeq$ by the definition of observational equivalence. □

3.3.4 Some Properties of Strong Bisimilarity

In this section we show some properties of strong bisimilarity, by exploiting the probabilistic "bisimulation up to" techniques introduced in Section 3.3.3 and Milner's transition induction technique [Mil89a].

Proposition 3.27 $\sim$ *and* $\sim_c$ *are congruence relations.*

Proof: This is a special version of the proof of Proposition 3.35, to which we shall give detailed arguments. □

Proposition 3.28 $\mu_X E \sim E\{\mu_X E/X\}$.

Proof: Observe that $\mu_X E \rightarrow \eta$ iff $E\{\mu_X E/X\} \rightarrow \eta$. □

Lemma 3.29 *If* $fpv(E) \subseteq \{\widetilde{X}, Z\}$ *and* $Z \notin fpv(\widetilde{F})$ *then*

$$E\{E'/Z\}\{\widetilde{F}/\widetilde{X}\} \equiv E\{\widetilde{F}/\widetilde{X}\}\{E'\{\widetilde{F}/\widetilde{X}\}/Z\}.$$

Proof: By induction on the structure of E. □

We now extend two results seen in nonprobabilistic process algebra [Mil84]. It should be emphasized that the "strong bisimulation up to" technique plays an important role in the subsequent proofs, because in these two cases it is difficult to directly construct an equivalence relation and prove that it is a strong bisimulation.

Proposition 3.30 $\mu_X(E+X) \sim \mu_X E$.

Proof: We show that the relation

$$\mathcal{R} = \{(F\{\mu_X(E+X)/X\}, F\{\mu_X E/X\} \mid F \in \mathcal{E} \text{ and } fpv(F) \subseteq \{X\}\}$$

is a strong bisimulation up to $\sim$. Below we prove the following two assertions:

1. If $F\{\mu_X(E+X)/X\} \to \eta_1$ then there exists η_2 s.t. $F\{\mu_X E/X\} \to \eta_2$ and $\eta_1 \equiv_{\mathcal{R}_\sim} \eta_2$;
2. If $F\{\mu_X E/X\} \to \eta_2$ then there exists η_1 s.t. $F\{\mu_X(E+X)/X\} \to \eta_1$ and $\eta_1 \equiv_{\mathcal{R}_\sim} \eta_2$.

We consider (1) by induction on the depth of the inference $F\{\mu_X(E+X)/X\} \to \eta_1$. Let us examine two typical cases, among others.

- $F \equiv X$: Then $(E+X)\{\mu_X(E+X)/X\} \to \eta_1$ by a shorter inference. Hence, by induction hypothesis, $(E+X)\{\mu_X E/X\} \to \eta_2$ with $\eta_1 \equiv_{\mathcal{R}_\sim} \eta_2$. Then we have either $\mu_X E \to \eta_2$ or $E\{\mu_X E/X\} \to \eta_2$. From the latter case we can also derive that $\mu_X E \to \eta_2$.
- $F \equiv \mu_Z F'$: Then $F'\{\mu_X(E+X)/X\}\{F\{\mu_X(E+X)/X\}/Z\} \to \eta_1$ by a shorter inference. By Lemma 3.29 we have $F'\{F/Z\}\{\mu_X(E+X)/X\} \to \eta_1$. By induction hypothesis, we have $F'\{F/Z\}\{\mu_X E/X\} \to \eta_2$ s.t. $\eta_1 \equiv_{\mathcal{R}_\sim} \eta_2$. Inversely it is easy to derive that $F\{\mu_X E/X\} \to \eta_2$.

Similarly (2) can be shown by induction on the depth of the inference $F\{\mu_X E/X\} \to \eta_2$. For example, if $F \equiv X$, then $E\{\mu_X E/X\} \to \eta_2$ by a shorter inference. By induction hypothesis, there exists η_1 s.t. $E\{\mu_X(E+X)/X\} \to \eta_1$ and $\eta_1 \equiv_{\mathcal{R}_\sim} \eta_2$. By rule nsum we have $(E+X)\{\mu_X(E+X)/X\} \equiv E\{\mu_X(E+X)/X\} + X\{\mu_X(E+X)/X\} \to \eta_1$. At last by rule rec we infer that $\mu_X(E+X) \to \eta_1$. □

The lemma below states that if X is weakly guarded in E, then different substitutions for X do not affect the first transition of E.

Lemma 3.31 *Suppose $fpv(E) \subseteq \{X\}$ and all free occurrences of X in E are weakly guarded. If $E\{F/X\} \to \eta_1$ with $\eta_1 \equiv \{(\ell_i, E_i : p_i)\}_i$ then E_i takes the form $E_i'\{F/X\}$; Moreover, for any G, $E\{G/X\} \to \eta_2$ with $\eta_2 \equiv \{(\ell_i, E_i'\{G/X\} : p_i)\}_i$ and $\eta_1 \equiv_{\mathcal{R}_\sim} \eta_2$ where*

$$\mathcal{R} = \{(E\{F/X\}, E\{G/X\}) \mid E \in \mathcal{E} \text{ and } fpv(E) \subseteq \{X\}\}.$$

Proof: By transition induction. □

Proposition 3.32 *If $E \sim F\{E/X\}$ and X weakly guarded in F, then $E \sim \mu_X F$.*

Proof: Similar to the proof of Proposition 3.30. Now we take $\mathcal{R}$ as:

$$\mathcal{R} = \{(G\{E/X\}, G\{\mu_X F/X\} \mid G \in \mathcal{E} \text{ and } fpv(G) \subseteq \{X\}\}$$

Let us consider the case that $G \equiv X$. Suppose $E \to \eta_1$. Since $E \sim F\{E/X\}$, there exists η_1' s.t. $F\{E/X\} \to \eta_1'$ and $\eta_1 \equiv_\sim \eta_1'$. By Lemma 3.31 there exists η_2 s.t. $F\{\mu_X F/X\} \to \eta_2$ and $\eta_1' \equiv_{\mathcal{R}_\sim} \eta_2$. By rule rec we have $\mu_X F \to \eta_2$. By Lemma 3.8 and the transitivity of $\equiv_{\mathcal{R}_\sim}$, we have $\eta_1 \equiv_{\mathcal{R}_\sim} \eta_2$. With similar reasoning, one can show that if $\mu_X F \to \eta_2$ there exists η_1 s.t. $E \to \eta_1$ and $\eta_1 \equiv_{\mathcal{R}_\sim} \eta_2$. □

3.3.5 Some Properties of Observational Equivalence

In this section we report some properties of $\eqsim$ and $\simeq$, especially those concerning recursions. As in last section, we heavily rely on the "bisimulation up to" techniques and transition induction.

Proposition 3.33 *1. $E \approx F$ iff $\tau.E \simeq \tau.F$;*

2. If $\tau.E \simeq \tau.E + F$ and $\tau.F \simeq \tau.F + E$ then $\tau.E \simeq \tau.F$.

Proof: The first clause is straightforward. For the second one, it suffices to prove that $E \approx F$. Consider the relation

$$\mathcal{R} = \{(E, F) \mid E, F \in \mathcal{E}, \tau.E \simeq \tau.E + F \text{ and } \tau.F \simeq \tau.F + E\}.$$

We show that $\mathcal{R}$ is a weak probabilistic bisimulation up to $\approx$. Suppose that $E \Rightarrow \eta$. By the condition $E + \tau.F \simeq \tau.F$ and Lemma 3.17, there exists η' s.t. $\tau.F \Rightarrow_c \eta'$ and $\eta \equiv_{\approx} \eta'$. Since $\tau.F \approx F$, by Lemma 3.14 there exists η'' s.t. $F \stackrel{\epsilon}{\Rightarrow}_c \eta''$ and $\eta' \equiv_{\approx} \eta''$. Then it is easy to see that $\eta \equiv_{\mathcal{R}_{\approx}} \eta''$. Similar result holds when E and F exchange their roles. □

Proposition 3.34 *If $E \simeq F$ then $\mu_X E \simeq \mu_X F$.*

Proof: We show that the relation

$$\mathcal{R} = \{(G\{\mu_X E/X\}, G\{\mu_X F/X\}) \mid E, F, G \in \mathcal{E} \text{ and } E \simeq F\}$$

is an observational equivalence up to $\simeq$. To achieve this goal, we need to prove the important property that $\simeq$ is closed under all substitutions. See Appendix A.2 for more details. □

Proposition 3.35 *$\simeq$ is a congruence relation.*

Proof: Given $\widetilde{E} \simeq \widetilde{F}$, we need to show the following three clauses:

1. $\bigoplus_i p_i \ell_i.E_i \simeq \bigoplus_i p_i \ell_i.F_i$
2. $\sum_{i \in 1..n} E_i \simeq \sum_{i \in 1..n} F_i$
3. $\mu_X E_1 \simeq \mu_X F_1$.

Among them, the first two clauses are easy to prove; the third one is shown in Proposition 3.34. □

We use a measure $d_X(E)$ to count the depth of guardedness of the free variable X in expression E.

$$\begin{array}{rcl} d_X(X) & \stackrel{\text{def}}{=} & 0 \\ d_X(Y) & \stackrel{\text{def}}{=} & 0 \\ d_X(a.E) & \stackrel{\text{def}}{=} & d_X(E) + 1 \\ d_X(\tau.E) & \stackrel{\text{def}}{=} & d_X(E) \\ d_X(\bigoplus_i p_i \ell_i.E_i) & \stackrel{\text{def}}{=} & min\{d_X(\ell_i.E_i)\}_i \\ d_X(\sum_i E_i) & \stackrel{\text{def}}{=} & min\{d_X(E_i)\}_i \\ d_X(\mu_Y E) & \stackrel{\text{def}}{=} & d_X(E) \end{array}$$

If $d_X(E) > 0$ then X is guarded in E.

The following Lemma is a counterpart of Lemma 3.31.

Lemma 3.36 *Let $d_X(G) > 1$. If $G\{E/X\} \Rightarrow_c \eta$ then $G\{F/X\} \Rightarrow_c \eta'$ such that $\eta \equiv_{\mathcal{R}^*} \eta'$ where $\mathcal{R} = \{(G\{E/X\}, G\{F/X\}) \mid$ for any $G \in \mathcal{E}\}$.*

Proof: See Appendix A.3. □

Proposition 3.37 *If $E \simeq F\{E/X\}$ and X is guarded in F then $E \simeq \mu_X F$.*

Proof: We show that the relation $\mathcal{R} = \{(G\{E/X\}, G\{\mu_X F/X\}) \mid$ for any $G \in \mathcal{E}\}$ is an observational equivalence up to $\simeq$. That is, we need to show the following assertions:

1. if $G\{E/X\} \Rightarrow \eta$ then there exists η' s.t. $G\{\mu_X F/X\} \Rightarrow_c \eta'$ and $\eta \equiv_{\mathcal{R}_\approx} \eta'$;
2. if $G\{\mu_X F/X\} \Rightarrow \eta'$ then there exists η s.t. $G\{E/X\} \Rightarrow_c \eta$ and $\eta \equiv_{\mathcal{R}_\approx} \eta'$.

We concentrate on the first clause since the second one is similar. The proof follows closely the arguments in proving Proposition 3.34, thus we only consider the case that $G \equiv X$.

We write $G(E)$ for $G\{E/X\}$ and $G^2(E)$ for $G(G(E))$. Since $E \simeq F(E)$, we have $E \simeq F^2(E)$ since $\simeq$ is an congruence relation by Proposition 3.35. If $E \Rightarrow \eta$ then by Lemma 3.17 there exists θ_1 s.t. $F^2(E) \Rightarrow_c \theta_1$ and $\eta \equiv_\approx \theta_1$. Since X is guarded in F, i.e., $d_X(F) > 0$, then it follows that $d_X(F^2(X)) > 1$. By Lemma 3.36, there exists θ_2 s.t. $F^2(\mu_X F) \Rightarrow_c \theta_2$ and $\theta_1 \equiv_{\mathcal{R}^*} \theta_2$. From Proposition 3.28 we have $\mu_X F \sim F^2(\mu_X F)$, thus $\mu_X F \simeq F^2(\mu_X F)$. By Lemma 3.17 there exists η' s.t. $\mu_X F \Rightarrow_c \eta'$ and $\theta_2 \equiv_\approx \eta'$. From Lemma 3.8 and the transitivity of $\equiv_{\mathcal{R}_\approx}$ it follows that $\eta \equiv_{\mathcal{R}_\approx} \eta'$. □

It is not difficult to see that all the propositions proved in this section for $\simeq$, except for Proposition 3.33, are also valid for $\eqsim$. In other words, $\eqsim$ is a substitutive congruence relation.

3.4 Axiomatisations for All Expressions

In this section we provide sound and complete axiomatisations for two strong behavioural equivalences: $\sim$ and $\sim_c$. The class of expressions to be considered is $\mathcal{E}$.

3.4.1 Axiomatizing Strong Bisimilarity

First we present the axiom system $\mathcal{A}_r$, which includes all axioms and rules displayed in Table 3.3. We assume the usual rules for equality (reflexivity, symmetry, transitivity and substitutivity), and the alpha-conversion of bound variables.

The notation $\mathcal{A}_r \vdash E = F$ (and $\mathcal{A}_r \vdash \widetilde{E} = \widetilde{F}$ for a finite sequence of equations) means that the equation $E = F$ is derivable by applying the axioms and rules from $\mathcal{A}_r$. The following theorem shows that $\mathcal{A}_r$ is sound with respect to $\sim$.

Theorem 3.38 (Soundness of $\mathcal{A}_r$) *If $\mathcal{A}_r \vdash E = E'$ then $E \sim E'$.*

S1	$E + 0 = E$
S2	$E + E = E$
S3	$\sum_{i\in I} E_i = \sum_{i\in I} E_{\rho(i)}$ ρ is any permutation on I
S4	$\bigoplus_{i\in I} p_i\ell_i.E_i = \bigoplus_{i\in I} p_{\rho(i)}\ell_{\rho(i)}.E_{\rho(i)}$ ρ is any permutation on I
S5	$(\bigoplus_i p_i\ell_i.E_i) \oplus p\ell.E \oplus q\ell.E = (\bigoplus_i p_i\ell_i.E_i) \oplus (p+q)\ell.E$
R1	$\mu_X E = E\{\mu_X E/X\}$
R2	If $E = F\{E/X\}$, X weakly guarded in F, then $E = \mu_X F$
R3	$\mu_X(E + X) = \mu_X E$

Table 3.3: The axiom system $\mathcal{A}_r$

Proof: The soundness of the recursion axioms **R1-3** is shown in Section 3.3.4; the soundness of **S1-4** is obvious, and **S5** is a consequence of Definition 3.6. □

For the completeness proof, the basic points are: (1) if two expressions are bisimilar then we can construct an equation set in a certain format (standard format) that they both satisfy; (2) if two expressions satisfy the same standard equation set, then they can be proved equal by $\mathcal{A}_r$. This schema is inspired by [Mil84, SS00], but in our case the definition of standard format and the proof itself are more complicated due to the presence of both probabilistic and nondeterministic dimensions.

Definition 3.39 *Let $\widetilde{X} = \{X_1, ..., X_m\}$ and $\widetilde{W} = \{W_1, W_2, ...\}$ be disjoint sets of variables. Let $\widetilde{H} = \{H_1, ..., H_m\}$ be expressions with free variables in $\widetilde{X} \cup \widetilde{W}$. In the equation set $\zeta : \widetilde{X} = \widetilde{H}$, we call $\widetilde{X}$* formal variables *and $\widetilde{W}$* free variables*. We say ζ is* standard *if each H_i takes the form $\sum_j E_{f(i,j)} + \sum_l W_{h(i,l)}$ where $E_{f(i,j)} = \bigoplus_k p_{f(i,j,k)}\ell_{f(i,j,k)}.X_{g(i,j,k)}$. We call ζ* weakly guarded *if there is no H_i s.t. $H_i \to \vartheta(X_i)$. We say that E* provably satisfies *ζ if there are expressions $\widetilde{E} = \{E_1, ..., E_m\}$, with $E_1 \equiv E$ and $fpv(\widetilde{E}) \subseteq \widetilde{W}$, such that $\mathcal{A}_r \vdash \widetilde{E} = \widetilde{H}\{\widetilde{E}/\widetilde{X}\}$.*

We first recall the theorem of unique solution of equations originally appeared in [Mil84]. Adding probabilistic choice does not affect the validity of this theorem.

Theorem 3.40 (Unique solution of equations I) *If ζ is a weakly guarded equation set with free variables in $\widetilde{W}$, then there is an expression E which provably satisfies ζ. Moreover, if F provably satisfies ζ and has free variables in $\widetilde{W}$, then $\mathcal{A}_r \vdash E = F$.*

Proof: Exactly as in [Mil84]. □

Below we give an extension of Milner's equational characterisation theorem by accommodating probabilistic choice.

Theorem 3.41 (Equational characterisation I) *For any expression E, with free variables in $\widetilde{W}$, there exist some expressions $\widetilde{E} = \{E_1, ..., E_m\}$, with $E_1 \equiv E$ and $fpv(\widetilde{E}) \subseteq \widetilde{W}$, satisfying m*

equations

$$\mathcal{A}_r \vdash E_i = \sum_{j\in 1..n(i)} E_{f(i,j)} + \sum_{j\in 1..l(i)} W_{h(i,j)} \qquad (i \leq m)$$

where $E_{f(i,j)} \equiv \bigoplus_{k\in 1..o(i,j)} p_{f(i,j,k)}\ell_{f(i,j,k)}.E_{g(i,j,k)}$.

Proof: By induction on the structure of E, similar to the proof in [Mil84]. □

The following completeness proof is closely analogous to that of [SS00]. It is complicated somewhat by the presence of nondeterministic choice. For example, to construct the formal equations, we need to consider a more refined relation $L_{iji'j'}$ underneath the relation $K_{ii'}$ while in [Mil84, SS00] it is sufficient to just use $K_{ii'}$.

Theorem 3.42 (Completeness of $\mathcal{A}_r$) *If $E \sim E'$ then $\mathcal{A}_r \vdash E = E'$.*

Proof: Let E and E' have free variables in $\widetilde{W}$. By Theorem 3.41 there are provable equations such that $E \equiv E_1$, $E' \equiv E'_1$ and

$$\mathcal{A}_r \vdash E_i = \sum_{j\in 1..n(i)} E_{f(i,j)} + \sum_{j\in 1..l(i)} W_{h(i,j)} \qquad (i \leq m)$$

$$\mathcal{A}_r \vdash E'_{i'} = \sum_{j'\in 1..n'(i')} E'_{f'(i',j')} + \sum_{j'\in 1..l'(i')} W_{h'(i',j')} \qquad (i' \leq m')$$

with

$$E_{f(i,j)} \equiv \bigoplus_{k\in 1..o(i,j)} p_{f(i,j,k)}\ell_{f(i,j,k)}.E_{g(i,j,k)}$$

$$E'_{f'(i',j')} \equiv \bigoplus_{k'\in 1..o'(i',j')} p'_{f'(i',j',k')}\ell'_{f'(i',j',k')}.E'_{g'(i',j',k')}.$$

Let $I = \{\langle i, i'\rangle \mid E_i \sim E'_{i'}\}$. By hypothesis we have $E_1 \sim E'_1$, so $\langle 1, 1\rangle \in I$. Moreover, for each $\langle i, i'\rangle \in I$, the following holds, by the definition of strong bisimilarity:

1. There exists a total surjective relation $K_{ii'}$ between $\{1, ..., n(i)\}$ and $\{1, ..., n'(i')\}$, given by

   $$K_{ii'} = \{\langle j, j'\rangle \mid \langle f(i,j), f'(i',j')\rangle \in I\}.$$

   Furthermore, for each $\langle j, j'\rangle \in K_{ii'}$ there exists a total surjective relation $L_{iji'j'}$ between $\{1, ..., o(i,j)\}$ and $\{1, ..., o'(i',j')\}$, given by

   $$L_{iji'j'} = \{\langle k, k'\rangle \mid \ell_{f(i,j,k)} = \ell'_{f'(i',j',k')} \text{ and } \langle g(i,j,k), g'(i',j',k')\rangle \in I\}.$$

2. $\mathcal{A}_r \vdash \sum_{j\in 1..l(i)} W_{h(i,j)} = \sum_{j'\in 1..l'(i')} W_{h'(i',j')}$.

Now, let $L_{iji'j'}(k)$ denote the image of $k \in \{1, ..., o(i,j)\}$ under $L_{iji'j'}$ and $L^{-1}_{iji'j'}(k')$ the preimage of $k' \in \{1, ..., o'(i',j')\}$ under $L_{iji'j'}$. We write $[k]_{iji'j'}$ for the set $L^{-1}_{iji'j'}(L_{iji'j'}(k))$ and $[k']_{iji'j'}$ for $L_{iji'j'}(L^{-1}_{iji'j'}(k'))$. It follows from the definitions that

1. If $\langle i, i'_1\rangle \in I$, $\langle i, i'_2\rangle \in I$, $\langle j, j'_1\rangle \in K_{ii'_1}$ and $\langle j, j'_2\rangle \in K_{ii'_2}$, then $[k]_{iji'_1j'_1} = [k]_{iji'_2j'_2}$.

2. If $q_1 \in [k]_{iji'j'}$ and $q_2 \in [k]_{iji'j'}$, then $\ell_{f(i,j,q_1)} = \ell_{f(i,j,q_2)}$ and $E_{g(i,j,q_1)} \sim E_{g(i,j,q_2)}$.

Define $\nu_{ijk} = \sum_{q\in[k]_{iji'j'}} p_{f(i,j,q)}$ for any i', j' such that $\langle i,i'\rangle \in I$ and $\langle j,j'\rangle \in K_{ii'}$; define $\nu'_{i'j'k'} = \sum_{q'\in[k']_{iji'j'}} p'_{f'(i',j',q')}$ for any i, j such that $\langle i,i'\rangle \in I$ and $\langle j,j'\rangle \in K_{ii'}$. It is easy to see that whenever $\langle i,i'\rangle \in I$, $\langle j,j'\rangle \in K_{ii'}$ and $\langle k,k'\rangle \in L_{iji'j'}$ then $\nu_{ijk} = \nu'_{i'j'k'}$.

We now consider the formal equations, one for each $\langle i,i'\rangle \in I$:

$$X_{i,i'} = \sum_{\langle j,j'\rangle\in K_{ii'}} H_{f(i,j),f'(i',j')} + \sum_{j\in 1..l(i)} W_{h(i,j)}$$

where

$$H_{f(i,j),f'(i',j')} \equiv \bigoplus_{\langle k,k'\rangle\in L_{iji'j'}} (\frac{p_{f(i,j,k)}p'_{f'(i',j',k')}}{\nu_{ijk}})\ell_{f(i,j,k)}.X_{g(i,j,k),g'(i',j',k')}.$$

These equations are provably satisfied when each $X_{i,i'}$ is instantiated to E_i, since $K_{ii'}$ and $L_{iji'j'}$ are total and the right-hand side differs at most by repeated summands from that of the already proved equation for E_i. Note that each probabilistic branch $p_{f(i,j,k)}\ell_{f(i,j,k)}.E_{g(i,j,k)}$ in E_i becomes the probabilistic summation of several branches like

$$\bigoplus_{q'\in[k']_{iji'j'}} (\frac{p_{f(i,j,k)}p'_{f'(i',j',q')}}{\nu_{ijk}})\ell_{f(i,j,k)}.E_{g(i,j,k)}$$

in $H_{f(i,j),f'(i',j')}\{E_i/X_{i,i'}\}_i$, where $\langle i,i'\rangle \in I$, $\langle j,j'\rangle \in K_{ii'}$ and $\langle k,k'\rangle \in L_{iji'j'}$. But they are provably equal because

$$\begin{array}{rcl}\sum_{q'\in[k']_{iji'j'}}(\frac{p_{f(i,j,k)}p'_{f'(i',j',q')}}{\nu_{ijk}}) & = & \frac{p_{f(i,j,k)}}{\nu_{ijk}}\cdot\sum_{q'\in[k']_{iji'j'}} p'_{f'(i',j',q')} \\ & = & \frac{p_{f(i,j,k)}}{\nu_{ijk}}\cdot\nu'_{i'j'k'} = p_{f(i,j,k)}\end{array}$$

and then the axiom **S5** can be used. Symmetrically, the equations are provably satisfied when each $X_{i,i'}$ is instantiated to $E'_{i'}$; this depends on the surjectivity of $K_{ii'}$ and $J_{iji'j'}$.

Finally, we note that each $X_{i,i'}$ is weakly guarded in the right-hand sides of the formal equations. It follows from Theorem 3.40 that $\vdash E_i = E'_{i'}$ for each $\langle i,i'\rangle \in I$, and hence $\vdash E = E'$. □

3.4.2 Axiomatizing Strong Probabilistic Bisimilarity

The difference between $\sim$ and $\sim_c$ is characterised by the following axiom:

$$\mathbf{C}\quad \sum_{i\in 1..n}\bigoplus_j p_{ij}\ell_{ij}.E_{ij} = \sum_{i\in 1..n}\bigoplus_j p_{ij}\ell_{ij}.E_{ij} + \bigoplus_{i\in 1..n}\bigoplus_j r_i p_{ij}\ell_{ij}.E_{ij}$$

where $\sum_{i\in 1..n} r_i = 1$. It is easy to show that the expressions on the left and right sides are strong probabilistic bisimilar. We denote $\mathcal{A}_r \cup \{\mathbf{C}\}$ by $\mathcal{A}_{rc}$.

Theorem 3.43 (Soundness and completeness of $\mathcal{A}_{rc}$) *$E \sim_c E'$ iff $\mathcal{A}_{rc} \vdash E = E'$.*

Proof: The soundness part follows immediately by the definition of $\rightarrow_c$. Below we focus on the completeness part.

Let E and E' have free variables in $\widetilde{W}$. By Theorem 3.41 there are provable equations such that $E \equiv E_1$, $E' \equiv E'_1$ and

$$\mathcal{A}_{rc} \vdash E_i = A_i \qquad (i \le m)$$

$$\mathcal{A}_{rc} \vdash E'_{i'} = A'_{i'} \qquad (i' \leq m')$$

where $A_i \equiv \sum_{j\in 1..n(i)} E_{f(i,j)} + \sum_{j\in 1..l(i)} W_{h(i,j)}$ and

$$E_{f(i,j)} \equiv \bigoplus_{k\in 1..o(i,j)} p_{f(i,j,k)}\ell_{f(i,j,k)}.E_{g(i,j,k)}$$

Similar for the form of $A'_{i'}$.

Next we shall use axiom **C** to saturate the right hand side of each equation with some summands so as to transform each A_i (resp. $A'_{i'}$) into a provably equal expression B_i (resp. $B'_{i'}$) which satisfies the following property:

(*) *For any* $C_1, C_2 \in \widetilde{B} \cup \widetilde{B'}$ *with* $C_1 \sim_c C_2$, *if* $C_1 \to \eta_1$ *then there exists some* η_2 *s.t.* $C_2 \to \eta_2$ *and* $\eta_1 \equiv_{\sim_c} \eta_2$.

Initially we set $\widetilde{B} = \widetilde{A}$ and $\widetilde{B'} = \widetilde{A'}$. Let $V = \{(C_1, C_2) \mid C_1 \sim_c C_2 \text{ and } C_1, C_2 \in \widetilde{A} \cup \widetilde{A'}\}$. Clearly the set V is finite because there are finitely many expressions in $\widetilde{A} \cup \widetilde{A'}$. Without loss of generality, we take a pair (C_1, C_2) from V such that $C_1 \equiv A'_{i'} \in \widetilde{A'}$ and $C_2 \equiv A_i \in \widetilde{A}$ (we do similar manipulations for other three cases, namely (i) $C_1, C_2 \in \widetilde{A}$; (ii) $C_1, C_2 \in \widetilde{A'}$; (iii) $C_1 \in \widetilde{A}$ and $C_2 \in \widetilde{A'}$). If $A'_{i'} \to \eta'$ then for some η we have $A_i \to_c \eta$ and $\eta \equiv_{\sim_c} \eta'$, by the definition of $\sim_c$. If $A_i \to \eta$ (obviously we are in this case if $\eta = \vartheta(X)$) we do nothing but go on to pick another pair from V to do the analysis. Otherwise η is a convex combination $\eta = r_1\eta_1 + ... + r_n\eta_n$ and $A_i \to \eta_j$ for each $j \leq n$. Hence each η_j must be in the form $\{(\ell_{f(i,j,k)}, E_{g(i,j,k)} : p_{f(i,j,k)})\}_k$ and $E_{f(i,j)}$ is a summand of A_i (so it is also a summand of B_i). By axiom **C** we have

$$\mathcal{A}_{rc} \vdash B_i = B_i + \bigoplus_{j\in 1..n} \bigoplus_k r_j p_{f(i,j,k)}\ell_{f(i,j,k)}.E_{g(i,j,k)}.$$

Now we update B_i to be to the expression on the right hand side of last equation. To this point we have finished the analysis to the pair (C_1, C_2). We need to pick a different pair from V to iterate the above procedure. When all the pairs in V are exhausted, we end up with $\widetilde{B}$ and $\widetilde{B'}$ which are easy to be verified to satisfy property (*). Observe that only axiom **C** is involved when updating B_i, so we have the following results:

$$\mathcal{A}_{rc} \vdash E_i = B_i \qquad (i \leq m)$$

$$\mathcal{A}_{rc} \vdash E'_{i'} = B'_{i'} \qquad (i' \leq m')$$

From now on, by using the above equations as our starting point, the subsequent arguments are like those for Theorem 3.42, so we omit them. □

3.5 Axiomatisations for Guarded Expressions

Now we proceed with the axiomatisations of the two weak behavioural equivalences: $\approx$ and $\simeq$. We are not able to give a complete axiomatisation for the whole set of expressions (and we conjecture that it is not possible, see Section 3.7), so we restrict to the subset of $\mathcal{E}$ consisting of *guarded expressions* only. An expression is guarded if for each of its subexpression of the form $\mu_X F$, the variable X is guarded in F (cf. Definition 3.2).

R2′	If $E = F\{E/X\}$, X guarded in F, then $E = \mu_X F$
T1	$\bigoplus_i p_i\tau.(E_i + X) = X + \bigoplus_i p_i\tau.(E_i + X)$
T2	$(\bigoplus_i p_i\ell_i.E_i) \oplus p\tau.(F + \bigoplus_j q_j h_j.F_j) + (\bigoplus_i p_i\ell_i.E_i) \oplus (\bigoplus_j pq_j h_j.F_j)$ $= (\bigoplus_i p_i\ell_i.E_i) \oplus p\tau.(F + \bigoplus_j q_j h_j.F_j)$
T3	$(\bigoplus_i p_i\ell_i.E_i) \oplus p\ell.(F + \bigoplus_j q_j\tau.F_j) + (\bigoplus_i p_i\ell_i.E_i) \oplus (\bigoplus_j pq_j\ell.F_j)$ $= (\bigoplus_i p_i\ell_i.E_i) \oplus p\ell.(F + \bigoplus_j q_j\tau.F_j)$

Table 3.4: Some laws for the axiom system $\mathcal{A}_{gd}$

3.5.1 Axiomatizing Divergency-Sensitive Equivalence

We first study the axiom system for $\eqsim$. As a starting point, let us consider the system $\mathcal{A}_{rc}$. Clearly, **S1-5** are still valid for $\eqsim$, as well as **R1**. **R3** turns out to be not needed in the restricted language we are considering. As for **R2**, we replace it with its (strongly) guarded version, which we shall denote as **R2′** (see Table 3.4). As in the standard process algebra, we need some τ-laws to abstract from invisible steps. For $\eqsim$ we use the probabilistic τ-laws **T1-3** shown in Table 3.4. Note that **T3** is the probabilistic extension of Milner's third τ-law ([Mil89b] page 231), and **T1** and **T2** together are equivalent, in the nonprobabilistic case, to Milner's second τ-law. However, Milner's first τ-law cannot be derived from **T1-3**, and it is actually unsound for $\eqsim$. Below we let $\mathcal{A}_{gd}$ ={**R2′, T1-3**} $\cup\mathcal{A}_{rc}\backslash${**R2-3**}.

Theorem 3.44 (Soundness of $\mathcal{A}_{gd}$) *If $\mathcal{A}_{gd} \vdash E = E'$ then $E \eqsim E'$.*

Proof: The rule **R2′** can be shown to be sound as Proposition 3.37. The soundness of **T1-3**, and therefore of $\mathcal{A}_{gd}$, is evident. □

For the completeness proof, it is convenient to use the following saturation property, which relates operational semantics to term transformation, and which can be shown by using the probabilistic τ-laws and the axiom **C**.

Lemma 3.45 (Saturation) *1. If $E \Rightarrow \eta$ with $\eta = \{(\ell_i, E_i : p_i)\}_i$, then $\mathcal{A}_{gd} \vdash E = E + \bigoplus_i p_i\ell_i.E_i$;*

2. If $E \Rightarrow_c \eta$ with $\eta = \{(\ell_i, E_i : p_i)\}_i$, then $\mathcal{A}_{gd} \vdash E = E + \bigoplus_i p_i\ell_i.E_i$;

3. If $E \Rightarrow \vartheta(X)$ then $\mathcal{A}_{gd} \vdash E = E + X$.

Proof: The first and third clauses are proved by transition induction on the inference of $E \Rightarrow \eta$; the second clause can be considered as a corollary of the first one. See Appendix A.4 for more details. □

To show the completeness of $\mathcal{A}_{gd}$, we need some notations. Given a standard equation set $\zeta : \widetilde{X} = \widetilde{H}$, which has free variables $\widetilde{W}$, we define the relations $\rightarrow_\zeta \subseteq \widetilde{X} \times \mathcal{P}((Var \cup \mathcal{L}) \times \widetilde{X})$ (the notation $\mathcal{P}(V)$ represents all distributions on V) as $X_i \rightarrow_\zeta \eta$ iff $H_i \rightarrow \eta$. From $\rightarrow_\zeta$ we can define

the weak transition $\Rightarrow_\zeta$ in the same way as in Section 3.2. We write $X_i \rightsquigarrow_\zeta X_k$ iff $X_i \Rightarrow_\zeta \eta$, with $\eta = \{(\ell_j, X_j : p_j)\}_{j\in J}$, $k \in J$ and $\ell_k = \tau$. We shall call ζ *guarded* if there is no X_i s.t. $X_i \rightsquigarrow_\zeta X_i$. We call ζ *saturated* if for all $X \in \widetilde{X}$, $X \Rightarrow_\zeta \eta$ implies $X \rightarrow_\zeta \eta$. The variable W is *guarded* in ζ if it is not the case that $X_1 \rightarrow_\zeta \vartheta(W)$ or $X_1 \rightsquigarrow_\zeta \rightarrow_\zeta \vartheta(W)$.

For guarded expressions, the equational characterisation theorem and the unique solution theorem given in last section can now be refined, as done in [Mil89b].

Theorem 3.46 (Equational characterisation II) *Every guarded expression E with free variables $\widetilde{W}$ provably satisfies a standard guarded equation set ζ with free variables in $\widetilde{W}$. Moreover, if W is guarded in E then W is guarded in ζ.*

Proof: By induction on the structure of E. Consider the case that $E \equiv \bigoplus_{i\in I} p_i\ell_i.E_i$. For each $i \in I$, let X_i be the distinguished variable of the equation set ζ_i for E_i. We can define ζ as $\{X = \bigoplus_{i\in I} p_i\ell_i.X_i\} \cup \bigcup_{i\in I} \zeta_i$, with the new variable X distinguished. All other cases are the same as in [Mil89b]. □

Lemma 3.47 *Let E provably satisfies the standard guarded equation set ζ. Then there is a saturated, standard, and guarded equation set ζ' provably satisfied by E.*

Proof: Let ζ be the equation set $\widetilde{X} = \widetilde{H}$ and $\mathcal{A}_{gd} \vdash \widetilde{E} = \widetilde{H}\{\widetilde{E}/\widetilde{X}\}$. By using Lemma 3.45, we show that if $X_i \Rightarrow \eta$ then $\mathcal{A}_{gd} \vdash E_i = E_i + \bigoplus_j p_j\ell_j.E_j$ when $\eta \equiv \{(\ell_j, X_j : p_j)\}_j$, and $\mathcal{A}_{gd} \vdash E_i = E_i + X$ when $\eta \equiv \vartheta(X)$. Repeat this procedure for all weak transitions of E_i, at last we get $\mathcal{A}_{gd} \vdash E_i = H_i'\{\widetilde{E}/\widetilde{X}\}$. Hence we can take ζ' to be the equation set $\widetilde{X} = \widetilde{H'}$. □

Theorem 3.48 (Unique solution of equations II) *If ζ is a guarded equation set with free variables in $\widetilde{W}$, then there is an expression E which provably satisfies ζ. Moreover, if F provably satisfies ζ and has free variables in $\widetilde{W}$, then $\mathcal{A}_{gd} \vdash E = F$.*

Proof: Nearly the same as the proof of Theorem 3.40, just replacing the recursion rule **R2** with **R2′**. □

The completeness result can be proved in a similar way as Theorem 3.42. The main difference is that here the key role is played by equation sets which are not only in standard format, but also saturated. The transformation of a standard equation set into a saturated one is obtained by using Lemma 3.45.

Theorem 3.49 (Completeness of $\mathcal{A}_{gd}$) *If E and E' are guarded expressions and $E \simeq E'$ then $\mathcal{A}_{gd} \vdash E = E'$.*

Proof: By Theorem 3.46 there are provable equations such that $E \equiv E_1$, $E' \equiv E_1'$ and

$$\mathcal{A}_{rc} \vdash E_i = A_i \qquad (i \le m)$$

$$\mathcal{A}_{rc} \vdash E_{i'}' = A_{i'}' \qquad (i' \le m')$$

For any $C \in \widetilde{A} \cup \widetilde{A'}$, we assume by Lemma 3.47 that C is saturated. Therefore it is easy to show that $C \Rightarrow_c \eta$ implies $C \rightarrow_c \eta$. Let $C' \in \widetilde{A} \cup \widetilde{A'}$. We note the interesting property that if $C \simeq C'$

T4	$\ell.\tau.E = \ell.E$
T5	If $\tau.E = \tau.E + F$ and $\tau.F = \tau.F + E$ then $\tau.E = \tau.F$.

Table 3.5: Two τ-laws for the axiom system $\mathcal{A}_{go}$

and $C \to \eta$ then there exists η' s.t. $C' \to_c \eta'$ and $\eta \equiv_{\approx} \eta'$. Thanks to this property the remaining arguments are quite similar to that in Theorem 3.43, thus are omitted. □

3.5.2 Axiomatizing Observational Equivalence

In this section we focus on the axiomatisation of $\simeq$. In order to obtain completeness, we can follow the same schema as for Theorem 3.42, with the additional machinery required for dealing with observational equivalence, like in [Mil89b]. The crucial point of the proof is to show that, if $E \simeq F$, then we can construct an equation set in standard format which is satisfied by E and F. The construction of the equation is more complicated than in [Mil89b] because of the subtlety introduced by the probabilistic dimension (cf. Theorem 3.53). Indeed, it turns out that the simple probabilistic extension of Milner's three τ-laws would not be sufficient, and we need an additional rule for the completeness proof to go through. We shall further comment on this rule at the end of Section 3.6.

The probabilistic extension of Milner's τ-laws are axioms **T1-4**, where **T1-3** are those introduced in previous section, and **T4**, defined in Table 3.5, takes the same form as Milner's first τ-law [Mil89b]. In the same table **T5** is the additional rule mentioned above. We let $\mathcal{A}_{go} = \mathcal{A}_{gd} \cup \{\textbf{T4-5}\}$.

Theorem 3.50 (Soundness of $\mathcal{A}_{go}$) *If $\mathcal{A}_{go} \vdash E = F$ then $E \simeq F$.*

Proof: Rule **T5** is proved to be sound in Proposition 3.33. The soundness of **T4**, and therefore of $\mathcal{A}_{go}$, is straightforward. □

The rest of the section is devoted to the completeness proof of $\mathcal{A}_{go}$. First we need two basic properties of weak combined transitions.

Lemma 3.51 *1. If $E \stackrel{\epsilon}{\Rightarrow}_c \eta$ then $\tau.E \Rightarrow_c \eta$;*

2. If $E \stackrel{\epsilon}{\Rightarrow}_c \vartheta(X)$ then $E \Rightarrow \vartheta(X)$.

Proof: The first clause is easy to show. Let us consider the second one. If $\vartheta(X)$ is a convex combination of $\eta_1, .., \eta_n$ and $E \Rightarrow \eta_i$ for all $i \in 1..n$, then each η_i must assign probability 1 to $(X, \mathsf{0})$, thus $\eta_i = \vartheta(X)$. □

Lemma 3.52 *If $E \stackrel{\epsilon}{\Rightarrow}_c \eta$ with $\eta = \{(\ell_i, E_i : p_i)\}_i$ then $\mathcal{A}_{gd} \vdash \tau.E = \tau.E + \bigoplus_i p_i \ell_i.E_i$.*

Proof: It follows from Lemma 3.51 and Lemma 3.45. □

The following theorem plays a crucial role in proving the completeness of $\mathcal{A}_{go}$.

Theorem 3.53 *Let E provably satisfy ζ and F provably satisfy ζ', where both ζ and ζ' are standard, guarded equation sets, and let $E \simeq F$. Then there is a standard, guarded equation set ζ'' satisfied by both E and F.*

Proof: Suppose that $\widetilde{X} = \{X_1, ..., X_m\}$, $\widetilde{Y} = \{Y_1, ..., Y_n\}$ and $\widetilde{W} = \{W_1, W_2, ...\}$ are disjoint sets of variables. Let

$$\zeta : \widetilde{X} = \widetilde{H}$$
$$\zeta' : \widetilde{Y} = \widetilde{J}$$

with $fpv(\widetilde{H}) \subseteq \widetilde{X} \cup \widetilde{W}$, $fpv(\widetilde{J}) \subseteq \widetilde{Y} \cup \widetilde{W}$, and that there are expressions $\widetilde{E} = \{E_1, ..., E_m\}$ and $\widetilde{F} = \{F_1, ..., F_n\}$ with $E_1 \equiv E$, $F_1 \equiv F$, and $fpv(\widetilde{E}) \cup fpv(\widetilde{F}) \subseteq \widetilde{W}$, so that

$$\mathcal{A}_{go} \vdash \widetilde{E} = \widetilde{H}\{\widetilde{E}/\widetilde{X}\}$$
$$\mathcal{A}_{go} \vdash \widetilde{F} = \widetilde{J}\{\widetilde{F}/\widetilde{Y}\}.$$

Consider the least equivalence relation $\mathcal{R} \subseteq (\widetilde{X} \cup \widetilde{Y}) \times (\widetilde{X} \cup \widetilde{Y})$ such that

1. whenever $(Z, Z') \in \mathcal{R}$ and $Z \to \eta$, then there exists η' s.t. $Z' \stackrel{\epsilon}{\Rightarrow}_c \eta'$ and $\eta \equiv_{\mathcal{R}} \eta'$;
2. $(X_1, Y_1) \in \mathcal{R}$ and if $X_1 \to \eta$ then there exists η' s.t. $Y_1 \Rightarrow_c \eta'$ and $\eta \equiv_{\mathcal{R}} \eta'$.

Clearly $\mathcal{R}$ is a weak probabilistic bisimulation on the transition system over $\widetilde{X} \cup \widetilde{Y}$, determined by $\rightarrow \stackrel{\text{def}}{=} \rightarrow_\zeta \cup \rightarrow_{\zeta'}$. Now for two given distributions $\eta = \{(\ell_i, X_i : p_i)\}_{i \in I}$, $\eta' = \{(h_j, Y_j : q_j)\}_{j \in J}$, with $\eta \equiv_{\mathcal{R}} \eta'$, we introduce the following notations:

$$\begin{array}{rcll} K_{\eta,\eta'} & = & \{(i,j) \mid i \in I,\ j \in J,\ \ell_i = h_j \text{ and } (X_i, Y_j) \in \mathcal{R}\} & \\ \nu_i & = & \sum\{p_{i'} \mid i' \in I,\ u_{i'} = \ell_i, \text{ and } (X_i, X_{i'}) \in \mathcal{R}\} & \text{for } i \in I \\ \nu_j & = & \sum\{p_{j'} \mid j' \in J,\ v_{j'} = h_j, \text{ and } (Y_j, Y_{j'}) \in \mathcal{R}\} & \text{for } j \in J \end{array}$$

Since $\eta \equiv_{\mathcal{R}} \eta'$ it follows by definition that if $(i,j) \in K_{\eta,\eta'}$, for some η, η', then $\nu_i = \nu_j$. Thus we can define the expression

$$G_{\eta,\eta'} \stackrel{\text{def}}{=} \bigoplus_{(i,j) \in K_{\eta,\eta'}} \frac{p_i q_j}{\nu_i} \ell_i . Z_{ij}$$

which will play the same role as the expression $H_{f(i,j),f'(i',j')}$ in the proof of Theorem 3.42. On the other hand, if $\eta = \eta' = \vartheta(X)$ we simply define the expression $G_{\eta,\eta'} \stackrel{\text{def}}{=} X$.

Based on the above $\mathcal{R}$ we choose a new set of variables $\widetilde{Z}$ such that

$$\widetilde{Z} = \{Z_{ij} \mid X_i \in \widetilde{X},\ Y_j \in \widetilde{Y} \text{ and } (X_i, Y_j) \in \mathcal{R}\}.$$

Furthermore, for each $Z_{ij} \in \widetilde{Z}$ we construct three auxiliary finite sets of expressions, denoted by A_{ij}, B_{ij} and C_{ij}, by the following procedure.

1. Initially the three sets are empty.
2. For each η with $X_i \to \eta$, arbitrarily choose one (and only one — the same principle applies in other cases too) η' (if it exists) satisfying $\eta \equiv_{\mathcal{R}} \eta'$ and $Y_j \Rightarrow_c \eta'$, construct the expression $G_{\eta,\eta'}$ and update A_{ij} to be $A_{ij} \cup \{G_{\eta,\eta'}\}$; Similarly for each η' with $Y_j \to \eta'$, arbitrarily choose one η (if it exists) satisfying $\eta \equiv_{\mathcal{R}} \eta'$ and $X_i \Rightarrow_c \eta$, construct $G_{\eta,\eta'}$ and update A_{ij} to be $A_{ij} \cup \{G_{\eta,\eta'}\}$.

3. For each η with $X_i \to \eta$, arbitrarily choose one η' (if it exists) satisfying $\eta \equiv_{\mathcal{R}} \eta'$, $Y_j \stackrel{\epsilon}{\Rightarrow}_c \eta'$ but not $Y_j \Rightarrow_c \eta'$, construct the expression $G_{\eta,\eta'}$ and update B_{ij} to be $B_{ij} \cup \{G_{\eta,\eta'}\}$.

4. For each η' with $Y_j \to \eta'$, arbitrarily choose one η (if it exists) satisfying $\eta \equiv_{\mathcal{R}} \eta'$, $X_i \stackrel{\epsilon}{\Rightarrow}_c \eta$ but not $X_i \Rightarrow_c \eta$, construct $G_{\eta,\eta'}$ and update C_{ij} to be $C_{ij} \cup \{G_{\eta,\eta'}\}$.

Clearly the three sets constructed in this way are finite. Now we build a new equation set

$$\zeta'' : \widetilde{Z} = \widetilde{L}$$

where ζ''_{11} is the distinguished variable and

$$L_{ij} = \begin{cases} \sum_{G \in A_{ij}} G & \text{if } B_{ij} \cup C_{ij} = \emptyset \\ \tau.(\sum_{G \in A_{ij} \cup B_{ij} \cup C_{ij}} G) & \text{otherwise.} \end{cases}$$

We assert that E provably satisfies the equation set ζ''. To see this, we choose expressions

$$G_{ij} = \begin{cases} E_i & \text{if } B_{ij} \cup C_{ij} = \emptyset \\ \tau.E_i & \text{otherwise} \end{cases}$$

and verify that $\mathcal{A}_{go} \vdash G_{ij} = L_{ij}\{\widetilde{G}/\widetilde{Z}\}$.

In the case that $B_{ij} \cup C_{ij} = \emptyset$, all those summands of $L_{ij}\{\widetilde{G}/\widetilde{Z}\}$ which are not variables are of the forms:

$$\bigoplus_{(i,j) \in K_{\eta,\eta'}} \frac{p_i q_j}{\nu_i} \ell_i.E_i \quad \text{or} \quad \bigoplus_{(i,j) \in K_{\eta,\eta'}} \frac{p_i q_j}{\nu_i} \ell_i.\tau.E_i.$$

By **T4** we can transform the second form into the first one. Then by some arguments similar to those in Theorem 3.42, together with Lemma 3.45, we can show that

$$\mathcal{A}_{go} \vdash L_{ij}\{\widetilde{G}/\widetilde{Z}\} = H_i\{\widetilde{E}/\widetilde{X}\} = E_i.$$

On the other hand, if $B_{ij} \cup C_{ij} \neq \emptyset$, we let $C_{ij} = \{D_1, ..., D_o\}$ ($C_{ij} = \emptyset$ is a special case of the following argument) and $D = \sum_{l \in 1..o} D_l\{\widetilde{G}/\widetilde{Z}\}$. As in last case we can show that

$$\mathcal{A}_{go} \vdash L_{ij}\{\widetilde{G}/\widetilde{Z}\} = \tau.(H_i\{\widetilde{E}/\widetilde{X}\} + D).$$

For any l with $1 \leq l \leq o$, let $D_l\{\widetilde{G}/\widetilde{Z}\} = \bigoplus_k p_k u_k.E_k$. It is easy to see that $E_i \stackrel{\epsilon}{\Rightarrow}_c \eta$ with $\eta = \{(u_k, E_k : p_k)\}_k$. So by Lemma 3.52 it holds that

$$\mathcal{A}_{go} \vdash \tau.E_i = \tau.E_i + D_l\{\widetilde{G}/\widetilde{Z}\}.$$

As a result we can infer

$$\mathcal{A}_{go} \vdash \tau.E_i = \tau.E_i + D = \tau.E_i + (E_i + D).$$

by Lemma 3.45. Similarly,

$$\mathcal{A}_{go} \vdash \tau.(E_i + D) = \tau.(E_i + D) + E_i.$$

Consequently it follows from **T5** that

$$\mathcal{A}_{go} \vdash \tau.E_i = \tau.(E_i + D) = \tau.(H_i\{\widetilde{E}/\widetilde{X}\} + D) = L_{ij}\{\widetilde{G}/\widetilde{Z}\}.$$

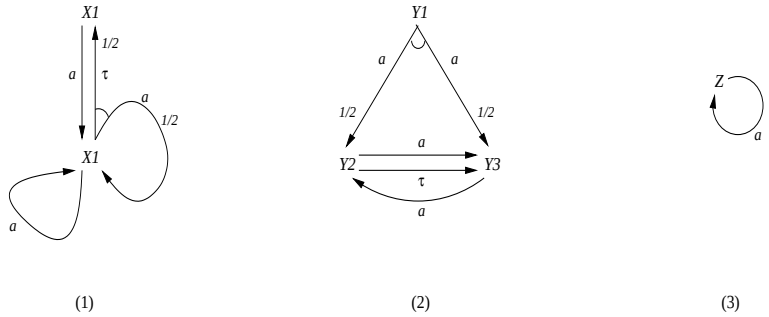

Figure 3.3: Observationally equivalent states X_1, Y_1 and Z

(i,j)	A_{ij}	B_{ij}	C_{ij}
$(1,1)$	$\{\frac{1}{2}a.Z_{22} \oplus \frac{1}{2}a.Z_{23}\}$	$\emptyset$	$\emptyset$
$(1,2)$	$\{a.Z_{23}\}$	$\emptyset$	$\{\tau.Z_{13}\}$
$(1,3)$	$\{a.Z_{22}\}$	$\emptyset$	$\emptyset$
$(2,1)$	$\{\frac{1}{2}a.Z_{22} \oplus \frac{1}{2}a.Z_{23}\}$	$\{\frac{1}{4}a.Z_{22} \oplus \frac{1}{4}a.Z_{23} \oplus \frac{1}{2}\tau.Z_{11}\}$	$\emptyset$
$(2,2)$	$\{a.Z_{23},\ \frac{1}{2}a.Z_{23} \oplus \frac{1}{2}\tau.Z_{13}\}$	$\emptyset$	$\{\tau.Z_{23}\}$
$(2,3)$	$\{a.Z_{22}\}$	$\{\frac{1}{2}a.Z_{22} \oplus \frac{1}{2}\tau.Z_{13}\}$	$\emptyset$

Table 3.6: The construction of sets A_{ij}, B_{ij}, C_{ij}

In the same way we can show that F provably satisfies ζ''. At last ζ'' is guarded because ζ and ζ' are guarded. □

To help understanding the proof of the above theorem, we illustrate the construction of the equation set ζ'' by a simple example. Consider the equation sets ζ and ζ' as follows.

$$\begin{array}{lrcl} \zeta: & X_1 & = & a.X_2 \\ & X_2 & = & a.X_2 + \frac{1}{2}a.X_2 \oplus \frac{1}{2}\tau.X_1 \end{array} \qquad \begin{array}{lrcl} \zeta': & Y_1 & = & \frac{1}{2}a.Y_2 \oplus \frac{1}{2}a.Y_3 \\ & Y_2 & = & a.Y_3 + \tau.Y_3 \\ & Y_3 & = & a.Y_2 \end{array}$$

The two equation sets describes the transition graphs in Figure 3.3 (1) and (2) respectively. Note that if E_1, E_2 provably satisfy ζ, and F_1, F_2, F_3 provably satisfy ζ', then $E_1 \simeq F_1 \simeq \mu_Z(a.Z)$ (cf. Figure 3.3 (3)).

Let $\mathcal{R}$ be the equivalence relation that has a unique equivalence class $\{X_1, X_2, X_3, Y_1, Y_2\}$. It is easy to check that $\mathcal{R}$ is a weak bisimulation on the transition system over $\widetilde{X} \cup \widetilde{Y}$. Now we take new variables $\{Z_{ij} \mid 1 \leq i \leq 2, 1 \leq j \leq 3\}$ and form the sets A_{ij}, B_{ij} and C_{ij} for each variable Z_{ij}, as displayed in Table 3.6, by using the procedure presented in the above proof.

We construct the equation set ζ'', based on all expressions shown in Table 3.6.

$$\begin{array}{lrcl}
\zeta'': & Z_{11} & = & \frac{1}{2}a.Z_{22} \oplus \frac{1}{2}a.Z_{23} \\
& Z_{12} & = & \tau.(a.Z_{23} + \tau.Z_{13}) \\
& Z_{13} & = & a.Z_{22} \\
& Z_{21} & = & \tau.(\frac{1}{2}a.Z_{22} \oplus \frac{1}{2}a.Z_{23} + \frac{1}{4}a.Z_{22} \oplus \frac{1}{4}a.Z_{23} \oplus \frac{1}{2}\tau.Z_{11}) \\
& Z_{22} & = & \tau.(a.Z_{23} + \frac{1}{2}a.Z_{23} \oplus \frac{1}{2}\tau.Z_{13} + \tau.Z_{23}) \\
& Z_{23} & = & \tau.(a.Z_{22} + \frac{1}{2}a.Z_{22} \oplus \frac{1}{2}\tau.Z_{13})
\end{array}$$

We can see that E_1 provably satisfies ζ'' by substituting E_1, $\tau.E_1$, E_1, $\tau.E_2$, $\tau.E_2$, $\tau.E_2$ for $Z_{11}, Z_{12}, Z_{13}, Z_{21}, Z_{22}, Z_{23}$, respectively; similarly F_1 provably satisfies ζ'' by substituting F_1, $\tau.F_2$, F_3, $\tau.F_1$, $\tau.F_2$, $\tau.F_3$ for these variables.

Theorem 3.54 (Completeness of $\mathcal{A}_{go}$) *If E and F are guarded expressions and $E \simeq F$, then $\mathcal{A}_{go} \vdash E = F$.*

Proof: A direct consequence by combining Theorem 3.46, 3.53 and 3.48. □

3.6 Axiomatisations for Finite Expressions

In this section we consider the recursion-free fragment of $\mathcal{E}$, that is the class $\mathcal{E}_f$ of all expressions which do not contain constructs of the form $\mu_X F$. In other words all expressions in $\mathcal{E}_f$ have the form: $\sum_i \bigoplus_j p_{ij} u_{ij}.E_{ij} + \sum_k X_k$.

We define four axiom systems for the four behavioural equivalences studied in this paper. Basically $\mathcal{A}_s, \mathcal{A}_{sc}, \mathcal{A}_{fd}, \mathcal{A}_{fo}$ are obtained from $\mathcal{A}_r$, $\mathcal{A}_{rc}$, $\mathcal{A}_{gd}$, $\mathcal{A}_{go}$ respectively, by cutting away all those axioms and rules that involve recursions.

$$\begin{array}{lcllcl}
\mathcal{A}_s & \stackrel{\text{def}}{=} & \{\textbf{S1-5}\} & \mathcal{A}_{sc} & \stackrel{\text{def}}{=} & \mathcal{A}_s \cup \{\textbf{C}\} \\
\mathcal{A}_{fd} & \stackrel{\text{def}}{=} & \mathcal{A}_{sc} \cup \{\textbf{T1-3}\} & \mathcal{A}_{fo} & \stackrel{\text{def}}{=} & \mathcal{A}_{fd} \cup \{\textbf{T4-5}\}
\end{array}$$

Theorem 3.55 (Soundness and completeness) *For any $E, F \in \mathcal{E}_f$,*

1. *$E \sim F$ iff $\mathcal{A}_s \vdash E = F$;*
2. *$E \sim_c F$ iff $\mathcal{A}_{sc} \vdash E = F$;*
3. *$E \approx F$ iff $\mathcal{A}_{fd} \vdash E = F$;*
4. *$E \simeq F$ iff $\mathcal{A}_{fo} \vdash E = F$.*

The soundness part is obvious. The completeness can be shown by following the lines of previous sections. However, since there is no recursion here, we have a much simpler proof which does not use the equational characterisation theorem and the unique solution theorem. Roughly speaking,

all the clauses are proved by induction on the depth of the expressions. We define the depth of a process, $d(E)$, as follows.

$$\begin{aligned} d(\mathsf{0}) &= 0 \\ d(X) &= 1 \\ d(\textstyle\bigoplus_i p_i\ell_i.E_i) &= 1 + max\{E_i\}_i \\ d(\textstyle\sum_i E_i) &= max\{d(E_i)\}_i \end{aligned}$$

The completeness proof of $\mathcal{A}_{fo}$ is a bit tricky. In the classical process algebra the proof can be carried out directly by using Hennessy Lemma [Mil89a], which says that if $E \approx F$ then either $\tau.E \simeq F$ or $E \simeq F$ or $E \simeq \tau.F$. In the probabilistic case, however, Hennessy Lemma does not hold. For example, let

$$E \stackrel{\text{def}}{=} a \quad \text{and} \quad F \stackrel{\text{def}}{=} a + (\frac{1}{2}\tau.a \oplus \frac{1}{2}a).$$

We can check that: (1) $\tau.E \not\approx F$, (2) $E \not\approx F$, (3) $E \not\approx \tau.F$. In (1) the distribution $\{(\tau, E : 1)\}$ cannot be simulated by any distribution from F. In (2) the distribution $\{(\tau, a : \frac{1}{2}), (a, \mathsf{0} : \frac{1}{2})\}$ cannot be simulated by any distribution from E. In (3) the distribution $\{(\tau, F : 1)\}$ cannot be simulated by any distribution from E.

Fortunately, to prove the completeness of $\mathcal{A}_{fo}$, it is sufficient to use the following weaker property.

Lemma 3.56 (Promotion) *For any $E, F \in \mathcal{E}_f$, if $E \approx F$ then $\mathcal{A}_{fo} \vdash \tau.E = \tau.F$.*

Proof: By induction on $d = d(E) + d(F)$. We consider the nontrivial case that $d > 0$.

If X is a nondeterministic summand of E, then $E \to \vartheta(X)$. Since $E \approx F$ it holds that $F \stackrel{\epsilon}{\Rightarrow}_c \vartheta(X)$. By Lemma 3.51 we have $\tau.F \Rightarrow \vartheta(X)$. It follows from (the recursion-free version of) Lemma 3.45 that $\mathcal{A}_{fd} \vdash \tau.F = \tau.F + X$.

Let $\bigoplus_{i\in I} p_i\ell_i.E_i$ be any summand of E. Then we have $E \to \eta$, with $\eta = \{(\ell_i, E_i : p_i)\}_{i\in I}$. Since $E \approx F$, there exists η', with $\eta' = \{(h_j, F_j : q_j)\}_{j\in J}$ s.t. $F \stackrel{\epsilon}{\Rightarrow}_c \eta'$ and $\eta \equiv_{\approx} \eta'$. For any $k, l \in I$ with $\ell_k = \ell_l$ and $E_k \approx E_l$, it follows from **T4** and induction hypothesis that $\mathcal{A}_{fo} \vdash \ell_k.E_k = \ell_k.\tau.E_k = \ell_l.\tau.E_l = \ell_l.E_l$. By **S5** we can derive that $\mathcal{A}_{fo} \vdash \bigoplus_{i\in I} p_i\ell_i.E_i = \bigoplus_{i'\in I'} p'_{i'}\ell'_{i'}.E'_{i'}$, where the process on the right hand side is "compact", i.e., for any $k', l' \in I'$, if $\ell'_{k'} = \ell'_{l'}$ and $E'_{k'} = E'_{l'}$ then $k' = l'$. Similarly we can derive $\mathcal{A}_{fo} \vdash \bigoplus_{j\in J} q_j h_j.F_j = \bigoplus_{j'\in J'} q'_{j'}h'_{j'}.F'_{j'}$ with the process on the right hand side "compact". From $\eta \equiv_{\approx} \eta'$ and the soundness of $\mathcal{A}_{fd}$, it is easy to prove that $\mathcal{A}_{fo} \vdash \bigoplus_{i'\in I'} p'_{i'}\ell'_{i'}.E'_{i'} = \bigoplus_{j'\in J'} q'_{j'}h'_{j'}.F'_{j'}$ since each probabilistic branch of one process is provably equal to a unique branch of the other process. It follows that $\mathcal{A}_{fo} \vdash \bigoplus_{i\in I} p_i\ell_i.E_i = \bigoplus_{j\in J} q_j h_j.F_j$. By (a recursion-free version of) Lemma 3.52 we infer $\mathcal{A}_{fo} \vdash \tau.F = \tau.F + \bigoplus_{j\in J} q_j h_j.F_j = \tau.F + \bigoplus_{i\in I} p_i\ell_i.E_i$.

In summary $\mathcal{A}_{fo} \vdash \tau.F = \tau.F + E$. Symmetrically $\mathcal{A}_{fo} \vdash \tau.E = \tau.E + F$. Therefore $\mathcal{A}_{fo} \vdash \tau.E = \tau.F$ by **T5**. □

The promotion lemma is inspired by [FY03], where a similar result is proved for a language of mobile processes.

At last, the completeness part of Theorem 3.55 (4) can be proved as Lemma 3.56. Note that for any $k, l \in I$ with $u_k = u_l$ and $E_k \approx E_l$, we derive $\mathcal{A}_{fo} \vdash u_k.E_k = u_l.E_l$ by using **T4** and the promotion lemma instead of using induction hypothesis.

S1	$E + 0 = E$
S2	$E + E = E$
S3	$\sum_{i\in I} E_i = \sum_{i\in I} E_{\rho(i)}$ $\quad \rho$ is any permutation on I
S4	$\bigoplus_{i\in I} p_i\ell_i.E_i = \bigoplus_{i\in I} p_{\rho(i)}\ell_{\rho(i)}.E_{\rho(i)}$ $\quad \rho$ is any permutation on I
S5	$(\bigoplus_i p_i\ell_i.E_i) \oplus p\ell.E \oplus q\ell.E = (\bigoplus_i p_i\ell_i.E_i) \oplus (p+q)\ell.E$
C	$\sum_{i\in 1..n} \bigoplus_j p_{ij}\ell_{ij}.E_{ij} = \sum_{i\in 1..n} \bigoplus_j p_{ij}\ell_{ij}.E_{ij} + \bigoplus_{i\in 1..n} \bigoplus_j r_i p_{ij}\ell_{ij}.E_{ij}$
T1	$\bigoplus_i p_i\tau.(E_i + X) = X + \bigoplus_i p_i\tau.(E_i + X)$
T2	$(\bigoplus_i p_i\ell_i.E_i) \oplus p\tau.(F + \bigoplus_j q_j h_j.F_j) + (\bigoplus_i p_i\ell_i.E_i) \oplus (\bigoplus_j pq_j h_j.F_j)$ $= (\bigoplus_i p_i\ell_i.E_i) \oplus p\tau.(F + \bigoplus_j q_j h_j.F_j)$
T3	$(\bigoplus_i p_i\ell_i.E_i) \oplus p\ell.(F + \bigoplus_j q_j\tau.F_j) + (\bigoplus_i p_i\ell_i.E_i) \oplus (\bigoplus_j pq_j\ell.F_j)$ $= (\bigoplus_i p_i\ell_i.E_i) \oplus p\ell.(F + \bigoplus_j q_j\tau.F_j)$
T4	$\ell.\tau.E = \ell.E$
T5	If $\tau.E = \tau.E + F$ and $\tau.F = \tau.F + E$ then $\tau.E = \tau.F$.
R1	$\mu_X E = E\{\mu_X E/X\}$
R2	If $E = F\{E/X\}$, X weakly guarded in F, then $E = \mu_X F$
R2′	If $E = F\{E/X\}$, X guarded in F, then $E = \mu_X F$
R3	$\mu_X(E + X) = \mu_X E$
	In **C**, there is a side condition $\sum_{i\in 1..n} r_i = 1$.

Table 3.7: All the axioms and rules

It is worth noticing that rule **T5** is necessary to prove Lemma 3.56. Consider the following two expressions: $\tau.a$ and $\tau.(a + (\frac{1}{2}\tau.a \oplus \frac{1}{2}a))$. It is easy to see that they are observational equivalent. However, we cannot prove their equality if rule **T5** is excluded from the system $\mathcal{A}_{fo}$. In fact, by using only the other rules and axioms it is impossible to transform $\tau.(a + (\frac{1}{2}\tau.a \oplus \frac{1}{2}a))$ into an expression without a probabilistic branch $p\tau.a$ occurring in any subexpression, for some p with $0 < p < 1$. So it is not provably equal to $\tau.a$, which has no probabilistic choice.

3.7 Summary

In this chapter we have proposed a probabilistic process calculus which corresponds to Segala and Lynch's probabilistic automata. We have presented strong bisimilarity, strong probabilistic bisimilarity, divergency-sensitive equivalence and observational equivalence. Sound and complete inference systems for the four behavioural equivalences are summarized in Table 3.8.

Note that we have axiomatized divergency-sensitive equivalence and observational equivalence only for guarded expressions. For unguarded expressions whose transition graphs include τ-loops, we conjecture that the two behavioural equivalences are undecidable and therefore not finitely axiomatizable. The reason is the following: in order to decide whether two expressions E and F are

strong equivalences	finite expressions	all expressions
$\sim$	$\mathcal{A}_s$: **S1-5**	$\mathcal{A}_r$: **S1-5,R1-3**
$\sim_c$	$\mathcal{A}_{sc}$: **S1-5,C**	$\mathcal{A}_{rc}$: **S1-5,R1-3,C**

weak equivalences	finite expressions	guarded expressions
$\approx$	$\mathcal{A}_{fd}$: **S1-5,C,T1-3**	$\mathcal{A}_{gd}$: **S1-5,C,T1-3,R1,R2′**
$\simeq$	$\mathcal{A}_{fo}$: **S1-5,C,T1-5**	$\mathcal{A}_{go}$: **S1-5,C,T1-5,R1,R2′**

Table 3.8: All the inference systems

observationally equivalent, one can compute the two sets

$$S_E = \{\eta \mid E \Rightarrow \eta\} \quad \text{and} \quad S_F = \{\eta \mid F \Rightarrow \eta\}$$

and then compare them to see whether each element of S_E is related to some element of S_F and vice versa. For guarded expressions E and F, the sets S_E and S_F are always finite and thus they can be compared in finite time. For unguarded expressions, these sets may be infinite, and so the above method does not apply. Furthermore, these sets can be infinite even when we factorize them with respect to an equivalence relation as required in the definition of weak probabilistic bisimulation. For example, consider the expression $E = \mu_X(\frac{1}{2}a \oplus \frac{1}{2}\tau.X)$. It can be proved that S_E is an infinite set $\{\eta_i \mid i \geq 1\}$, where

$$\eta_i = \{(a, \mathsf{0} : (1 - \frac{1}{2^i})), (\tau, E : \frac{1}{2^i})\}.$$

Furthermore, for each $i, j \geq 1$ with $i \neq j$ we have $\eta_i \not\equiv_{\mathcal{R}} \eta_j$ for any equivalence relation $\mathcal{R}$ which distinguishes E from $\mathsf{0}$. Hence the set S_E modulo $\mathcal{R}$ is infinite.

It should be remarked that the presence of τ-loops in itself does not necessarily cause non-decidability. For instance, the notion of weak probabilistic bisimulation defined in [Seg95, CS02] is decidable for finite-state PA. The reason is that in those works weak transitions are defined in terms of schedulers, and one may get some weak transitions that are not derivable by the (finitary) inference rules used in this paper. For instance, consider the transition graph of the above example. The definition of [Seg95, CS02] allows the underlying probabilistic execution to be infinite as long as that case occurs with probability 0. Hence with that definition one has a weak transition that leads to the distribution $\theta = \{(a, \mathsf{0} : 1)\}$. Thus each η_i becomes a convex combination of θ and $\delta(E)$, i.e. these two distributions are enough to characterise all possible weak transitions. By exploiting this property, Cattani and Segala gave a decision algorithm for weak probabilistic bisimulation in [CS02].

In this chapter we have chosen, instead, to generate weak transitions via (finitary) inference rules, which means that only finite executions can be derived. This approach, which is also known in the literature ([SL94]), has the advantage of being more formal, and in the case of guarded recursion it is equivalent to the one of [Seg95, CS02]. In the case of unguarded recursion, however, we feel that it would be more natural to consider also the "limit" weak transitions of [Seg95, CS02]. The axiomatisation of the corresponding notion of observational equivalence is an open problem.

Chapter 4

Axiomatisations for Typed Mobile Processes

In this chapter we study the impact of types on the algebraic theory of the π-calculus. The type system has capability types, which give rise to a natural and powerful subtyping relation – the main source of challenges and interests of this chapter. We consider two variants of typed bisimilarity, both in their late and in their early version. For both of them, we give complete axiomatisations for the closed finite terms. For one of the two variants, we provide a complete axiomatisation for the open finite terms.

The contents of this chapter are presented in the following order. In Section 4.1 we introduce the syntax, semantics and typed bisimilarity for a version of the π-calculus without parallelism. This small language already shows the major obstacles for axiomatisations and hence makes the presentation of our ideas neater. In Section 4.2 we set up a complete axiomatisation for closed terms. In Section 4.3 we axiomatize the typed bisimilarity for all finite terms. In Section 4.4 we examine other equivalences and relate their axiomatisations to the results obtained in the previous sections. In Section 4.5 we show how the operator of parallel composition is admitted in the language. The effect on the axiomatisations is to add an expansion law to eliminate all occurrences of the operator. Finally we end this chapter with some concluding remarks.

4.1 A Fragment of The Typed π-calculus

In this section we review the π-calculus (without parallelism), capability types, the usual operational semantics, typed labelled transition system as well as typed bisimilarity.

4.1.1 Standard Operational Semantics

We assume an infinite set of channels, ranged over by $a, b, \ldots$, and an infinite set of variables, ranged over by $x, y, \ldots$. We write $*$ for the unit value (we shall use `unit` as the only base type). Channels, variables and $*$ are the *names*, ranged over by $u, v, \ldots$. Below is the syntax of finite processes (also

called terms).

$$\begin{array}{rcl} P,Q & ::= & \mathsf{0} \mid \tau.P \mid u(x:T).P \mid \bar{u}v.P \mid P+Q \mid (\nu a:T)P \mid \varphi PQ \\ \varphi & ::= & [u=v] \mid \neg\varphi \mid \varphi \wedge \psi \end{array}$$

It has the usual constructors of finite monadic π-calculus: inaction, prefix, sum and restriction. The *match* constructor is replaced by a more general *condition*, ranged by φ, ψ etc, and produced by match, *negation* and *conjunction*. *Mismatching* like $[u \neq v]$ abbreviates $\neg[u = v]$. We also use $\vee$, which can be derived from $\wedge$ as usual. Here φPQ is an if-then-else construct on the boolean condition φ. We omit the else branch Q when it is $\mathsf{0}$. We have not included an operator of recursion because our main results in this chapter are about axiomatisations for finite terms. However, all results and definitions in Section 4.1 remain valid when recursion is added.

There is a channel-binding and a variable-binding operator. In $(\nu a : S)P$ the displayed occurrence of channel a is *binding* with *scope* P. In $u(x : T).P$ the occurrence of variable x is binding with scope P. An occurrence of a channel (resp. variable) in a process is *bound* if it lies within the scope of a binding occurrence of the channel (resp. variable). An occurrence of a channel or a variable in a process is *free* if it is not bound. We write $fn(P)$ and $fv(P)$ for the set of free names and the set of free variables, respectively, in P. We use $n(\varphi)$ for all names appearing in φ. When φ has no variables, $[\![\varphi]\!]$ denotes the boolean value of φ.

When $fv(P) \neq \emptyset$, P is an open term. We can make open terms closed by the use of *closing substitutions*, ranged over by $\sigma, \sigma', \sigma_i, \cdots$, which are substitutions mapping variables to channels and acting as identity on channels (thus similar to the concept of ground substitution used in term rewriting systems [Zan03]). In the calculus, the distinction between channels and variables simplifies certain technical details; see for instance the discussion on the rules for substitutivity of prefixes in Section 4.3: the rules are different depending on whether the prefixes use channels or variables. (This is not the case in the untyped case: for instance, [PS95] does not distinguish between variables and channels, but it is quite straightforward to adapt the work to the case where there is such a distinction.)

The standard operational semantics is presented in the late style in Table 4.1. The symmetric rule of sum is omitted. In a transition $P \xrightarrow{\alpha} P'$, the closed term P may become open in P' after performing the action α. As usual there are four forms of actions: τ (interaction), $a(x : T)$ (input), $\bar{a}b$ (free output), $\bar{a}(b : T)$ (bound output). We also use α to range over the set of extended prefixes, which contains the tau, the input prefixes, the output prefixes and the bound output prefixes. The bound output $\bar{u}(a : T).P$ is an abbreviation of $(\nu a : T)\bar{u}a.P$. As in Section 2.2.2 we use $subj(\alpha)$, $bn(\alpha)$ and $n(\alpha)$ to stand for the subject, bound name and names of α. As usual we identify terms up to alpha-conversion.

We recall the capability types, as from [HR04, HR02b]. The subtyping relation $<:$ and the typing rules for processes are displayed in Table 4.2. We write $T :: \mathtt{TYPE}$ to mean that T is a well-defined type. There are three forms of types for channel names: $\mathtt{i}T, \mathtt{o}S$ and $\mathtt{b}\langle T, S\rangle$, they correspond to the ability to receive values of type T, send values of type S, or to do both. For simplicity we often abbreviate $\mathtt{b}\langle T, T\rangle$ to $\mathtt{b}T$ (which is actually the simple channel type $\sharp T$ given in Section 2.2.5). As shown in [HR02b], this extension to the original I/O types (cf. Section 2.2.6) makes it possible to define two partial operators meet ($\sqcap$) and join ($\sqcup$). But the definitions of the two operators are

in $\dfrac{}{a(x:T).P \xrightarrow{a(x:T)} P}$	out $\dfrac{}{\bar{a}b.P \xrightarrow{\bar{a}b} P}$
tau $\dfrac{}{\tau.P \xrightarrow{\tau} P}$	sum $\dfrac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'}$
true $\dfrac{[\![\varphi]\!] = \mathit{True} \quad P \xrightarrow{\alpha} P'}{\varphi\ P\ Q \xrightarrow{\alpha} P'}$	false $\dfrac{[\![\varphi]\!] = \mathit{False} \quad Q \xrightarrow{\alpha} Q'}{\varphi\ P\ Q \xrightarrow{\alpha} Q'}$
open $\dfrac{P \xrightarrow{\bar{a}b} P' \quad a \neq b}{(\nu b:T)P \xrightarrow{\bar{a}(b:T)} P'}$	res $\dfrac{P \xrightarrow{\alpha} P' \quad b \notin n(\alpha)}{(\nu b:T)P \xrightarrow{\alpha} (\nu b:T)P'}$

Table 4.1: Transition rules

rather long, so we do not repeat them and recommend the reader to consult Section 6 of [HR02b].[1] Intuitively, the meet (resp. join) of T and S is the union (resp. intersection) of their capabilities.

Proposition 4.1 *Given types T_1, T_2 and S with $T_1 <: T_2$.*

1. *If $T_i \sqcap S$ are defined, for $i = 1, 2$, then $T_1 \sqcap S <: T_2 \sqcap S$;*
2. *If $T_i \sqcup S$ are defined, for $i = 1, 2$, then $T_1 \sqcup S <: T_2 \sqcup S$;*
3. *$T_1 \sqcap T_2 = T_1$;*
4. *$T_1 \sqcup T_2 = T_2$.*

Proof: Following the definitions of meet and join, the result is straightforward by structural induction on types. □

A type environment Δ is a partial function from channels and variables to types; we write $\Delta_{\mathtt{c}}$ and $\Delta_{\mathtt{v}}$ for the channel and variable parts of Δ, respectively. A type environment is undefined on infinitely many channels and variables (to make sure it can always be extended). We will often view, and talk about, $\Delta_{\mathtt{c}}$ as a set of assignments of the form $a : T$, describing the value of $\Delta_{\mathtt{c}}$ on all the channels on which $\Delta_{\mathtt{c}}$ is defined. Similarly for $\Delta_{\mathtt{v}}$. If $\Delta(u)$ is defined and takes the form $\mathtt{i}T$ or $\mathtt{b}\langle T, S\rangle$, then the predicate $\Delta(u)\downarrow_{\mathtt{i}}$ holds and we write $\Delta(u)_{\mathtt{i}}$ for T, otherwise the predicate $\Delta(u)\not\downarrow_{\mathtt{i}}$ holds, indicating that Δ has no input capability on u. Similarly for $\Delta(u)_{\mathtt{o}}$ and $\Delta(u)\downarrow_{\mathtt{o}}$ (output capability). Notice that $\Delta(u)\downarrow_{\mathtt{i}}$ is covariant and $\Delta(u)\downarrow_{\mathtt{o}}$ is contravariant.

Proposition 4.2 *Suppose that $u, v \in dom(\Delta)$ and $\Delta(u) <: \Delta(v)$.*

1. *If $\Delta(v)\downarrow_{\mathtt{i}}$ then $\Delta(u)_{\mathtt{i}} <: \Delta(v)_{\mathtt{i}}$;*
2. *If $\Delta(v)\downarrow_{\mathtt{o}}$ then $\Delta(v)_{\mathtt{o}} <: \Delta(u)_{\mathtt{o}}$.*

[1] The only modification we have made is as follows. If two channel types T and S have no common capability, then in our setting $T \sqcup S$ is undefined, while in [HR02b] $T \sqcup S$ is defined to be a maximal type, which is a supertype of every channel type.

Types:

$$\frac{}{\mathtt{unit} :: \mathtt{TYPE}} \qquad \frac{T :: \mathtt{TYPE}}{\mathtt{i}T, \mathtt{o}T :: \mathtt{TYPE}} \qquad \frac{T, S :: \mathtt{TYPE} \quad S <: T}{\mathtt{b}\langle T, S\rangle :: \mathtt{TYPE}}$$

Subtyping:

$$\frac{}{T <: T} \qquad \frac{T <: T' \quad T' <: T''}{T <: T''} \qquad \frac{T <: T'}{\mathtt{i}T <: \mathtt{i}T'}$$

$$\frac{T <: T'}{\mathtt{o}T' <: \mathtt{o}T} \qquad \frac{T <: T'}{\mathtt{b}\langle T, S\rangle <: \mathtt{i}T'} \qquad \frac{T <: T'}{\mathtt{b}\langle S, T'\rangle <: \mathtt{o}T}$$

$$\frac{T <: T' \quad S <: S'}{\mathtt{b}\langle T, S'\rangle <: \mathtt{b}\langle T', S\rangle}$$

Typing rules:

$$\frac{\Gamma(u) <: T}{\Gamma \vdash u : T} \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P + Q} \qquad \frac{\Gamma, x : T \vdash P \quad \Gamma \vdash u : \mathtt{i}T}{\Gamma \vdash u(x : T).P}$$

$$\frac{}{\Gamma \vdash \mathtt{0}} \qquad \frac{\Gamma, a : T \vdash P}{\Gamma \vdash (\nu a : T)P} \qquad \frac{\Gamma \vdash u : \mathtt{o}T \quad \Gamma \vdash v : T \quad \Gamma \vdash P}{\Gamma \vdash \bar{u}v.P}$$

$$\frac{\Gamma \vdash P}{\Gamma \vdash \tau.P} \qquad \frac{\Gamma \vdash P \quad \Gamma \vdash Q \quad n(\varphi) \subseteq dom(\Gamma)}{\Gamma \vdash \varphi\ P\ Q}$$

Table 4.2: Types and typing rules

The typing rules for processes are standard except for conditions. We impose no constraint for the types of names appearing in conditions. The reason is discussed in Section 1.4. This mild modification does not affect the proofs of the following two results [PS96, HR02b, HR04].

Lemma 4.3 (Substitution) *If* $\Gamma \vdash a : T$ *and* $\Gamma, x : T \vdash P$, *then* $\Gamma \vdash P\{a/x\}$.

Theorem 4.4 (LTS subject reduction) *Suppose* $\Gamma \vdash P$ *and* $P \xrightarrow{\alpha} P'$.

1. *if* $\alpha = \tau$ *then* $\Gamma \vdash P'$.
2. *if* $\alpha = a(x : T)$ *then* $\Gamma(a) \downarrow_{\mathtt{i}}$ *and* $\Gamma, x : T \vdash P'$.
3. *if* $\alpha = \bar{a}b$ *then* $\Gamma(a) \downarrow_{\mathtt{o}}$, $\Gamma \vdash b : \Gamma(a)_{\mathtt{o}}$ *and* $\Gamma \vdash P'$.
4. *if* $\alpha = \bar{a}(b : T)$ *then* $\Gamma(a) \downarrow_{\mathtt{o}}$, $\Gamma, b : T \vdash b : \Gamma(a)_{\mathtt{o}}$ *and* $\Gamma, b : T \vdash P'$.

4.1.2 Typed Labelled Transition System

Two known TLTSs were presented in [BS98, HR04], both of them were given in early style. We prefer to write a TLTS in late style, so as to define the late version of bisimilarity in a concise way.

$$\textsf{Red}\ \frac{P \xrightarrow{\tau} P'}{\Delta \sharp P \xrightarrow{\tau} \Delta \sharp P'} \qquad \textsf{Out}\ \frac{\Delta(a) \downarrow_{\mathtt{i}}}{\Delta \sharp \bar{a}b.P \xrightarrow{\bar{a}b} \Delta \sqcap b : \Delta(a)_{\mathtt{i}} \sharp P}$$

$$\textsf{In}\ \frac{\Delta(a) \downarrow_{\mathtt{o}}}{\Delta \sharp a(x:T).P \xrightarrow{a(x:T)} \Delta, x : T \sharp P} \qquad \textsf{Open}\ \frac{\Delta \sharp P \xrightarrow{\bar{a}b} \Delta' \sharp P' \quad a \neq b}{\Delta \sharp (\nu b : T)P \xrightarrow{\bar{a}(b:T)} \Delta' \sharp P'}$$

$$\textsf{Res}\ \frac{\Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P' \quad a \notin n(\alpha)}{\Delta \sharp (\nu a : T)P \xrightarrow{\alpha} \Delta' \sharp (\nu a : T)P'} \qquad \textsf{Sum}\ \frac{\Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P'}{\Delta \sharp P + Q \xrightarrow{\alpha} \Delta' \sharp P'}$$

$$\textsf{True}\ \frac{[\![\varphi]\!] = \mathit{True} \quad \Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P'}{\Delta \sharp \varphi PQ \xrightarrow{\alpha} \Delta' \sharp P'} \qquad \textsf{False}\ \frac{[\![\varphi]\!] = \mathit{False} \quad \Delta \sharp Q \xrightarrow{\alpha} \Delta' \sharp Q'}{\Delta \sharp \varphi PQ \xrightarrow{\alpha} \Delta' \sharp Q'}$$

Table 4.3: Typed LTS

First we extend the subtyping relation to type environments, but only considering the types of channels. So $\Gamma <: \Delta$ means that $\Gamma_{\mathtt{v}} = \Delta_{\mathtt{v}}$, $dom(\Delta_{\mathtt{c}}) \subseteq dom(\Gamma_{\mathtt{c}})$ and $\Gamma_{\mathtt{c}}(a) <: \Delta_{\mathtt{c}}(a)$ for all $a \in dom(\Delta_{\mathtt{c}})$.

Definition 4.5 *A configuration is a pair $\Delta \sharp P$ which respects some type environment Γ, i.e., $\Gamma <: \Delta$ and $\Gamma \vdash P$.*

The above definition implies the condition $fv(P) \subseteq dom(\Delta_{\mathtt{v}})$, because we have $fv(P) \subseteq dom(\Gamma_{\mathtt{v}})$ by $\Gamma \vdash P$ and $dom(\Gamma_{\mathtt{v}}) = dom(\Delta_{\mathtt{v}})$ by $\Gamma <: \Delta$. Since alpha-conversion is implicitly used throughout this thesis, we may assume $bn(P) \cap dom(\Delta) = \emptyset$. Here there exists a mild difference from the definitions of configuration given in [BS98, HR04]. We do not require the environment to have knowledge of all the free channels used by P. The less knowledge it grasps, the weaker testing power it owns when observing the behaviour of P. In Table 4.3, we present a transition system built on this definition. In the premise of rule Red, $P \xrightarrow{\tau} P'$ stands for the standard reduction relation of the typed π-calculus, as given in Table 4.1.

Using the partial meet operation, we can extend a type environment Δ to $\Delta \sqcap u : T$, which is just $\Delta, u : T$ if $u \notin dom(\Delta)$, otherwise it differs from Δ at name u because the capability of this name is extended to be $\Delta(u) \sqcap T$ (if $\Delta(u) \sqcap T$ is undefined, then so is $\Delta \sqcap u : T$). In this way we can define $\Delta_1 \sqcap \Delta_2$ as the meet of two environments Δ_1 and Δ_2. In rule Out, the process sends channel b to the environment, so the latter should be dynamically extended with the capability on b thus received. For this, we use the meet operator, and exploit the following property on types:

$$R <: T \text{ and } R <: S \text{ imply } T \sqcap S \text{ defined and } R <: T \sqcap S$$

for any type T, S and R. (This property does not hold for the capability types as in Section 2.2.6.)

The next three fundamental lemmas describe various properties of the TLTS. They underpin many later results. The well-definedness of our TLTS is based on Lemma 4.6. The close relationship between processes and configurations is reflected by their corresponding transitions, as can be seen in Lemma 4.7. Finally Lemma 4.8 says that the more capabilities an environment owns, the more behaviours it can observe on a process.

Lemma 4.6 (TLTS subjection reduction) *If $\Delta\sharp P$ is a configuration which respects Γ and $\Delta\sharp P \xrightarrow{\alpha} \Delta'\sharp P'$, then $\Delta'\sharp P'$ is also a configuration, respecting Γ', where*

1. *if $\alpha = \tau$ then $\Delta' = \Delta$ and $\Gamma' = \Gamma$.*
2. *if $\alpha = a(x:T)$ then $\Delta' = \Delta, x:T$ and $\Gamma' = \Gamma, x:T$.*
3. *if $\alpha = \bar{a}b$ then $\Delta' = \Delta \sqcap b : \Delta(a)_{\mathtt{i}}$ and $\Gamma' = \Gamma$.*
4. *if $\alpha = \bar{a}(b:T)$ then $\Delta' = \Delta, b : \Delta(a)_{\mathtt{i}}$ and $\Gamma' = \Gamma, b:T$.*

Proof: By induction on depth of inference. LTS subject reduction theorem is needed. □

Lemma 4.7 *Suppose that $\Delta\sharp P$ is a configuration.*

1. *$\Delta\sharp P \xrightarrow{\tau} \Delta\sharp P'$ iff $P \xrightarrow{\tau} P'$.*
2. *$\Delta\sharp P \xrightarrow{a(x:T)} \Delta, x:T\sharp P'$ iff $\Delta(a) \downarrow_{\mathtt{o}}$ and $P \xrightarrow{a(x:T)} P'$.*
3. *$\Delta\sharp P \xrightarrow{\bar{a}b} \Delta \sqcap b : \Delta(a)_{\mathtt{i}}\sharp P'$ iff $\Delta(a) \downarrow_{\mathtt{i}}$ and $P \xrightarrow{\bar{a}b} P'$.*
4. *$\Delta\sharp P \xrightarrow{\bar{a}(b:T)} \Delta, b : \Delta(a)_{\mathtt{i}}\sharp P'$ iff $\Delta(a) \downarrow_{\mathtt{i}}$ and $P \xrightarrow{\bar{a}(b:T)} P'$.*

Proof: By induction on depth of inference. □

Lemma 4.8 *Suppose that $\Delta\sharp P \xrightarrow{\alpha} \Delta'\sharp P'$, $\Gamma <: \Delta$ and $\Gamma\sharp P$ is a configuration. Then $\Gamma\sharp P \xrightarrow{\alpha} \Gamma'\sharp P'$ and $\Gamma' <: \Delta'$.*

Proof: Straightforward by using the preceding lemma. □

4.1.3 Typed Bisimilarity

When comparing two typed actions, to require them to be syntactically the same is too restrictive. For example one would not be able to say $(\nu a : T_1)\bar{u}a$ is bisimilar to $(\nu a : T_2)\bar{u}a$ under the environment $\Delta = u : \mathtt{bob}T$, where $T_1 = \mathtt{bo}T, T_2 = \mathtt{bb}T$. Therefore we do not check types in the bisimulation game. We shall write $|\alpha|$ for the action α where its type annotations have been stripped off.

$P \mathrel{\dot{=}}_\Delta Q$ reads "P and Q are bisimilar under type environment Δ". The type environment Δ is used as follows: $\Delta_{\mathtt{c}}$ shows the channels that are known to the external observer testing the processes in the bisimulation game, and the types with which the observer is allowed to use such channels. By contrast, $\Delta_{\mathtt{v}}$ shows the set of variables that may appear free in the processes and the types for these variables show how the observer can instantiate such variables (in closing substitutions). Therefore: the channels of $\Delta_{\mathtt{c}}$ are to be used by the observer, with the types indicated in $\Delta_{\mathtt{c}}$; the variables in $\Delta_{\mathtt{v}}$ are to be used by the processes, but the observer can instantiate them following the types indicated in $\Delta_{\mathtt{v}}$.

A process is *closed* if it does not have free variables; similarly a type environment is closed if it is only defined on channels. Otherwise, processes and type environments are *open*. We first define $\mathrel{\dot{=}}_\Delta$ on the closed terms, then on the open terms. Bisimilarity is given in the late style; we consider the early style in Section 4.4.2.

Definition 4.9 *A family of symmetric binary relations over closed terms, indexed by type environments, and written* $\{\mathcal{R}_\Delta\}_\Delta$*, is a* typed bisimulation *whenever* $P\ \mathcal{R}_\Delta\ Q$ *implies that, for two configurations* $\Delta\sharp P$ *and* $\Delta\sharp Q$*,*

1. *if* $\Delta\sharp P \xrightarrow{\alpha} \Delta'\sharp P'$ *and* α *is not an input action, then for some* Q'*,* $\Delta\sharp Q \xrightarrow{\beta} \Delta'\sharp Q'$*,* $|\alpha| = |\beta|$ *and* $P'\ \mathcal{R}_{\Delta'}\ Q'$*.*
2. *if* $\Delta\sharp P \xrightarrow{a(x:T)} \Delta'\sharp P'$*, then for some* Q'*,* $\Delta\sharp Q \xrightarrow{a(x:S)} \Delta''\sharp Q'$ *and for all* b *with* $\Delta_{\mathsf{c}} \vdash b : \Delta(a)_{\mathsf{o}}$ *it holds that* $P'\{b/x\}\ \mathcal{R}_\Delta\ Q'\{b/x\}$*.*

Two processes P *and* Q *are typed* Δ*-bisimilar, written* $P \mathrel{\dot{=}}_\Delta Q$*, if there exists a typed bisimulation* $\{\mathcal{R}_\Delta\}_\Delta$ *such that* $P\ \mathcal{R}_\Delta\ Q$*.*

The difference w.r.t. typed bisimilarity as in [BS98, HR04] is that, in the input clause, the type environment Δ is not extended. In other words, the knowledge of the external observer does not change through interactions with the process in which the value transmitted is supplied by the observer itself (by contrast, the knowledge does change when the value is supplied by the process; cf. rule Out in Table 4.3). Therefore $\mathrel{\dot{=}}_\Delta$ is optimised for reasoning on finite systems. To deal with infinite systems, it is more suitable to use the alternative equivalence where the environment can be extended. We shall turn to this topic in Section 4.4.1.

Definition 4.10 *Two processes* P *and* Q *are bisimilar under the environment* $\Delta = \Delta_{\mathsf{c}}, \widetilde{x} : \widetilde{T}$*, written* $P \mathrel{\dot{=}}_\Delta Q$*, if* $\Delta\sharp P$*,* $\Delta\sharp Q$ *are configurations and, for all* $\widetilde{b}$ *with* $\Delta_{\mathsf{c}} \vdash \widetilde{b} : \widetilde{T}$*, it holds that* $P\{\widetilde{b}/\widetilde{x}\} \mathrel{\dot{=}}_{\Delta_{\mathsf{c}}} Q\{\widetilde{b}/\widetilde{x}\}$*.*

The intuition behind the above definition is that channels are capabilities while variables are obligations of the environment. The environment is obliged to fill in the variables at the specified types. Once the obligations are determined, they cannot be strengthened or weakened. That's why variables are invariant in the subtyping relation on type environments given before.

Below we report three basic properties of typed bisimilarity.

Lemma 4.11 *If* $P \mathrel{\dot{=}}_\Delta Q$ *and* $\Delta <: \Delta'$*, then* $P \mathrel{\dot{=}}_{\Delta'} Q$*.*

Proof: By Lemma 4.7, 4.8 and the definition of typed bisimilarity. □

The intuition behind this lemma is quite clear. When two processes exhibit similar behaviours under an environment with stronger discriminating power, they are also indistinguishable by a weaker environment. In the presence of distinction between channels and variables, we have the following interesting property for typed bisimilarity.

Lemma 4.12 *If* $P \mathrel{\dot{=}}_{\Delta, x:T} Q$ *and* $S <: T$ *then* $P \mathrel{\dot{=}}_{\Delta, x:S} Q$*.*

Proof: It follows easily from the definition of typed bisimilarity on open terms. □

As we said before in Section 1.4, generally speaking, typed behavioural equivalences are not closed under injective substitutions. Nevertheless, if a substitution only maps channels and variables to other channels and variables of the same types respectively (called *type-preserving substitution*), we do have the property seen in untyped π-calculus, as expressed by the lemma below. (With a slight abuse of notation, here we use σ to stand for type-preserving substitutions.)

Lemma 4.13 *If $P \doteq_\Delta Q$ then $P\sigma \doteq_{\Delta\sigma} Q\sigma$ for σ injective on $fn(P,Q) \cup dom(\Delta)$ and $\Delta\sigma$ is the type environment which maps $\sigma(u)$ to $\Delta(u)$ for all $u \in dom(\Delta)$.*

Proof: Similar to the proof in untyped setting. It follows from the fact that $\Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P'$ implies $\Delta\sigma \sharp P\sigma \xrightarrow{\alpha\sigma} \Delta'\sigma \sharp P'\sigma$, for injective type-preserving substitution σ. □

Since all processes are finite, and we do not use recursive types, in $P \doteq_\Delta Q$, the environment Δ can always be taken to be *finite* (i.e., defined only on a finite number of channels and variables): it is sufficient that Δ has enough names fresh w.r.t. P and Q, for all relevant types. This can be proved with a construction similar to that in Lemma 4.34. In the remainder of the chapter all type environments are assumed to be finite. (If Δ is infinite, our proof systems in Section 4.2 and 4.4.1 remain sound and complete; the axiom system in Section 4.3 is still sound, but its completeness proof relies on the finiteness of Δ.) We should stress, however, that all results and definitions presented up to this section are also valid for non-finite processes (i.e., processes extended with recursion) and for infinite type environment.

4.2 Proof System for the Closed Terms

In this section we present a proof system for the closed terms.

The proof system $\mathcal{P}$ for typed bisimilarity is composed of all inference rules and axioms in Table 4.4. Whenever we write $P =_\Delta Q$ it is intended that both $\Delta \sharp P$ and $\Delta \sharp Q$ are configurations (see Definition 4.5 and the explanations immediately follow the definition), and in this section P, Q are deemed to be closed terms. The rules are divided into six groups, namely those for: substitutivity, sums, looking up the type environment, conditions, restrictions and alpha-conversion. The rules that are new or different w.r.t. those of the untyped π-calculus are marked with an asterisk.

Tin* shows that an input prefix is not observable if the observer has no output capability on the subject of the input. This comes as no surprise because the only means that the observer uses for testing a process is to communicate with it. When no communication happens, he/she simply regards the process being tested as $\mathtt{0}$. **Tout*** is the symmetric rule, for output. **Twea*** gives us weakening for type environments, corresponding to Lemma 4.11. In **Ires***, the side condition $a \notin dom(\Delta)$ is added for the sake of clarity, but formally it is not needed because of the definition of configurations and our convention on bound names. Note that different types T_1, T_2 are used for the processes in the conclusion. We cannot replace **Ires*** with two simpler rules such as

- If $P =_\Delta Q$ then $(\nu a : T)P =_\Delta (\nu a : T)Q$
- $(\nu a : T_1)P =_\Delta (\nu a : T_2)P$,

for equalities like $(\nu b : \mathtt{bi}T)\bar{a}b.b(x : \mathtt{i}T).\mathtt{0} =_{a:\mathtt{iob}T} (\nu b : \mathtt{bo}T)\bar{a}b.b(x : \mathtt{o}T).\mathtt{0}$ could not be derived (due to the constraints given by the well-typedness of processes). Similarly for rule **Iinc***.

Iinc* and **Iout*** are the rules for substitutivity for input and output prefixes. In **Iinc***, the well-definedness of the two configurations $\Delta \sharp a(x : T_1).P$ and $\Delta \sharp a(x : T_2).Q$ implies the condition: $\Delta(a)_\circ <: T_i$ for $i = 1, 2$. In **Iout***, the observer knowledge of the type of b may increase when the

Iinc*	If $P\{b/x\} =_\Delta Q\{b/x\}$ for all b with $\Delta_{\mathsf{c}} \vdash b : \Delta(a)_{\mathsf{o}}$ then $a(x:T_1).P =_\Delta a(x:T_2).Q$.
Iout*	If $P =_{\Delta \sqcap b:\Delta(a)_{\mathsf{i}}} Q$ then $\bar{a}b.P =_\Delta \bar{a}b.Q$
Itau	If $P =_\Delta Q$ then $\tau.P =_\Delta \tau.Q$
Isum	If $P =_\Delta Q$ then $P + R =_\Delta Q + R$
Ires*	If $P =_\Delta Q$ then $(\nu a : T_1)P =_\Delta (\nu a : T_2)Q \quad a \notin dom(\Delta)$
S1	$P + \mathsf{0} =_\Delta P$
S2	$P + P =_\Delta P$
S3	$P + Q =_\Delta Q + P$
S4	$P + (Q + R) =_\Delta (P + Q) + R$
Tin*	If $\Delta(a)\not\downarrow_{\mathsf{o}}$ then $a(x:T).P =_\Delta \mathsf{0}$
Tout*	If $\Delta(a)\not\downarrow_{\mathsf{i}}$ then $\bar{a}u.P =_\Delta \mathsf{0}$
Twea*	If $P =_\Delta Q$ and $\Delta <: \Delta'$ then $P =_{\Delta'} Q$
Ca	$\varphi\ P\ Q =_\Delta P \quad$ if $[\![\varphi]\!] = \mathit{True}$
Cb	$\varphi\ P\ Q =_\Delta Q \quad$ if $[\![\varphi]\!] = \mathit{False}$
R1	$(\nu a : T)\mathsf{0} =_\Delta \mathsf{0}$
R2	$(\nu a : T)\alpha.P =_\Delta \mathsf{0} \quad$ if $subj(\alpha) = a$
R3	$(\nu a : T)(\nu b : S)P =_\Delta (\nu b : S)(\nu a : T)P$
R4	$(\nu a : T)(P + Q) =_\Delta (\nu a : T)P + (\nu a : T)Q$
R5	$(\nu a : T)\alpha.P =_\Delta \alpha.(\nu a : T)P \quad$ if $a \notin n(\alpha)$
A	$P =_\Delta Q \quad$ if P alpha-equivalent to Q

Table 4.4: The proof system $\mathcal{P}$ for the closed terms

processes emit b themselves (for the type under which b is emitted is composed with the possible type of b in Δ).

Compared with the proof system for untyped π-calculus [PS95], **Tin*** and **Tout*** are the main differences.

Theorem 4.14 (Soundness of $\mathcal{P}$) *If $\mathcal{P} \vdash P =_\Delta Q$ then $P \simeq_\Delta Q$.*

Proof: By constructing appropriate bisimulations. □

The completeness proof uses a standard strategy. By using the axioms **S1-4**, **R1-5** and **Ca-b**, we can transform each closed term into a canonical form $\sum_i \alpha_i.P_i$. If P and Q are bisimilar, their canonical forms P' and Q' are provably equal by induction on the depth of $P' + Q'$.

Theorem 4.15 (Completeness of $\mathcal{P}$) *If $P \simeq_\Delta Q$ then $\mathcal{P} \vdash P =_\Delta Q$, where P and Q are closed terms.*

Proof: This proof differs from the completeness proof of untyped π-calculus [MPW92] in one place: instead of showing that each summand of P is provably equivalent to a summand in Q, we only require that each *active summand* of P is matched by an active summand of Q, and vice versa. By active summand, we mean that the prefix can perform actions allowed by the environment Δ. More precisely, if $a_i(x_i : T_i).P_i$ is a summand of P and $\Delta(a_i)\downarrow_{\mathtt{o}}$ then this is an active input prefix. Similarly for output prefixes. Inactive summand is provably equivalent to $\mathtt{0}$ by **Tin*** and **Tout***, thus can be consumed by **S1**. After finite steps of transformation, we have $\mathcal{P} \vdash P =_\Delta \sum_{i=1}^n \alpha_i.P_i$ and $\mathcal{P} \vdash Q =_\Delta \sum_{j=1}^m \beta_j.Q_j$, where all summands in P and Q are active.

Suppose that $\alpha_i = \bar{a}(b : T_1)$. Then $\Delta \sharp P \stackrel{\bar{a}(b:T_1)}{\longrightarrow} \Delta, b : \Delta(a)_{\mathtt{i}} \sharp P_i$. Hence there is some $\beta_j = \bar{a}(b : T_2)$ such that $P_i \simeq_{\Delta, b:\Delta(a)_{\mathtt{i}}} Q_j$. Since the depth of $P_i + Q_j$ is less than the depth of $P + Q$, we can use induction hypothesis to derive $\mathcal{P} \vdash P_i =_{\Delta, b:\Delta(a)_{\mathtt{i}}} Q_j$. By **A** we assume that the bound name $b \notin dom(\Delta)$, so $\Delta, b : \Delta(a)_{\mathtt{i}} = \Delta \sqcap b : \Delta(a)_{\mathtt{i}}$. Therefore we have $\mathcal{P} \vdash \bar{a}b.P_i =_\Delta \bar{a}b.Q_j$ by **Iout***, and furthermore $\mathcal{P} \vdash \bar{a}(b : T_1).P_i =_\Delta \bar{a}(b : T_2).Q_j$ by **Ires***.

Suppose that $\alpha_i = a(x : T_1)$. Then $\Delta \sharp P \stackrel{a(x:T_1)}{\longrightarrow} \Delta' \sharp P_i$. There must exist a $\beta_j = a(x : T_2)$ such that $P_i\{b/x\} \simeq_\Delta Q_j\{b/x\}$, for all b s.t. $\Delta_{\mathtt{c}} \vdash b : \Delta(a)_{\mathtt{o}}$. Now observe that the depth of $P_i\{b/x\} + Q_j\{b/x\}$ is less than the depth of $P + Q$, thus it follows from induction hypothesis that $\mathcal{P} \vdash P_i\{b/x\} =_\Delta Q_j\{b/x\}$. Using **Iinc*** we infer that $\mathcal{P} \vdash a(x : T_1).P_i =_\Delta a(x : T_2).Q_j$.

Other cases can be analyzed similarly. As a result, each active summand of P is provably equal to some active summand of Q. Symmetric arguments also hold. □

4.3 Axioms for Typed Bisimilarity

In this section we give an axiom system for typed bisimilarity and prove its soundness and completeness. This axiomatisation is for all finite terms of the language given in Section 4.1, including both open and closed terms.

4.3.1 The Axiom System

The axiom system $\mathcal{A}$ for typed bisimilarity is presented in Table 4.5. Roughly speaking, it is obtained from $\mathcal{P}$ by adding some axioms for dealing with conditions. In open terms usually the conditions cannot be simply eliminated by **Ca-b**, so we need the axioms **C1-7** and **R6-7** to manipulate them. We use the notation $\varphi \Rightarrow \psi$ to mean that φ logically implies ψ; in **C1** the condition $\varphi \Longleftrightarrow \psi$ means that φ and ψ are logically equivalent. In view of **C3** and **R6**, axiom **R1** is redundant. The rule **Iinc*** of $\mathcal{P}$ now becomes the concise axiom **Iin*** in $\mathcal{A}$. **Tvar*** shows that a variable can only be instantiated with channels that in the type environment have types compatible with that of the variable. **Tpre*** is used to replace names underneath a match. It implies, in the presence of other axioms of $\mathcal{A}$, a more powerful axiom: $[x = a]P =_\Delta [x = a]P\{a/x\}$ if $\Delta(a) <: \Delta(x)$, which substitutes through P. In the untyped setting, **Tpre*** has no side condition. Here we need one to ensure well-typedness of the process resulting from the substitution, since the names in the match can have arbitrary — and possibly unrelated — types.

The following axioms and rules are derivable from **{S1-S4, C1-C6, Tvar*}**. More derived rules are given in Appendix B.1.

C8	$P =_\Delta \varphi P + \neg\varphi P$	**C9**	$\varphi PQ =_\Delta \varphi P + \neg\varphi Q$
C10	$[\varphi \vee \psi]P =_\Delta \varphi P + \psi P$	**C11**	$\varphi(P + Q) =_\Delta \varphi P + \varphi Q$
Cnn1	$[a = b]P =_\Delta \mathsf{0}$ if $a \neq b$	**Tvn1**	$[x = a]P =_\Delta \mathsf{0}$ if $a \notin dom(\Delta)$
Cnn2	$[a \neq b]P =_\Delta P$ if $a \neq b$	**Tvn2**	$[x \neq a]P =_\Delta P$ if $a \notin dom(\Delta)$
Tv1	$P =_{\Delta,x:T} \mathsf{0}$ if there exists no $a \in dom(\Delta)$ s.t. $\Delta(a) <: T$		

Note that in **Iin*** and **Iout***, the free names of the input and output prefixes are channels rather than variables. Below we discuss:

1. the unsoundness of the rules in which (some or all) the channels are replaced by variables;
2. other rules, that are valid for variables;
3. why these other rules are not needed in the axiom system.

Intuitively the reason for (1) is the different usage of channels and variables that appear in a type environment: the information on channels tells us how these channels are to be used by the *external environment*, while the information on variables tells us how these variables are to be instantiated inside the *tested processes*.

To see that **Iin*** is unsound when the subject of the prefix is a variable, take $\Delta_\mathsf{c} \stackrel{\text{def}}{=} a : \mathtt{b}\mathtt{o}T, b : \mathtt{o}T$ and $\Delta \stackrel{\text{def}}{=} \Delta_\mathsf{c}, x : \mathtt{b}\langle \mathtt{o}T, \mathtt{b}T\rangle$. Then we have

$$[y = b]\tau \doteq_{\Delta, y:\Delta(x)_\mathsf{o}} \mathsf{0}$$

because $\Delta(x)_\mathsf{o} = \mathtt{b}T$ and no c in Δ satisfies the condition $\Delta_\mathsf{c} \vdash c : \mathtt{b}T$ and can therefore instantiate y. However,

$$x(y : \mathtt{o}T).[y = b]\tau \neq_\Delta x(y : \mathtt{o}T).\mathsf{0}.$$

To see this, let us look at the possible closing substitutions. In $dom(\Delta_\mathsf{c})$, a is the only channel satisfying $\Delta_\mathsf{c} \vdash a : \Delta(x)$, and so the only substitution we need to consider is $\{a/x\}$. After applying

Iin*	If $P =_{\Delta,x:\Delta(a)_\circ} Q$ then $a(x:T_1).P =_\Delta a(x:T_2).Q$
Icon	If $P =_\Delta Q$ then $\varphi P =_\Delta \varphi Q$
Tvar*	$[x \neq a_1] \cdots [x \neq a_m] P =_\Delta \mathtt{0}$ if $\{b \in dom(\Delta_\mathtt{c}) \mid \Delta(b) <: \Delta(x)\} \subseteq \{a_1, \cdots, a_m\}$
Tpre*	$[x = a]\alpha.P =_\Delta [x = a](\alpha\{a/x\}).P$ if $\Delta(a) <: \Delta(x)$
C1	$\varphi\ P =_\Delta \psi\ P$ if $\varphi \Longleftrightarrow \psi$
C2	$[a = b]P =_\Delta [a = b]Q$ if $a \neq b$
C3	$\varphi\ P\ P =_\Delta P$
C4	$\varphi\ P\ Q =_\Delta \neg\varphi\ Q\ P$
C5	$\varphi(\psi P) =_\Delta [\varphi \wedge \psi]P$
C6	$\varphi\ (P_1 + P_2)\ (Q_1 + Q_2) =_\Delta \varphi\ P_1\ Q_1 + \varphi\ P_2\ Q_2$
C7	$\varphi\ (\alpha.P) =_\Delta \varphi\ (\alpha.\varphi P)$ if $bn(\alpha) \cap n(\varphi) = \emptyset$
R6	$(\nu a:T)[a = u]P =_\Delta \mathtt{0}$ if $a \neq u$
R7	$(\nu a:T)[u = v]P =_\Delta [u = v](\nu a:T)P$ if $a \neq u, v$
$\mathcal{P}\backslash\{$**Iinc*, Ca-b, R1**$\}$	

Table 4.5: The axiom system $\mathcal{A}$

this substitution, the resulting closed terms are not bisimilar:

$$a(y:\mathtt{o}T).[y = b]\tau \neq_\Delta a(y:\mathtt{o}T).\mathtt{0}$$

This holds because the observer can send b along a and, after the communication, y is instantiated to be b, thus validating the condition $y = b$ and liberating the prefix τ. When the subject of the prefix is a variable, the following rule is needed in place of **Iin***:

Iv1 If $P =_{\Delta,y:\Delta(x)_\mathtt{i}} Q$ then $x(y:T_1).P =_\Delta x(y:T_2).Q$

In rule **Iout***, both the subject and object of the output prefix are channels. The rule is also valid when the object is a variable. However, it is not valid if the subject is a variable. As a counterexample, let $\Delta_\mathtt{c} \stackrel{\text{def}}{=} a:\mathtt{i}T, b:\mathtt{bb}T$ and $\Delta \stackrel{\text{def}}{=} \Delta_\mathtt{c}, x:\mathtt{b}\langle\mathtt{i}T, \mathtt{b}T\rangle$. Then we have $a \eqsim_{\Delta\sqcap a:\mathtt{i}T} \mathtt{0}$ but $\bar{x}a.a \not\eqsim_\Delta \bar{x}a.\mathtt{0}$ because, under the substitution $\{b/x\}$, it holds that $\bar{b}a.a \not\eqsim_\Delta \bar{b}a.\mathtt{0}$. When the subject of the prefix is a variable, we need the following rule:

Iv2 If $P =_{\Delta\sqcap v:\Delta(x)_\circ} Q$ then $\bar{x}v.P =_\Delta \bar{x}v.Q$

We show, by means of an example, why rules **Iin*** and **Iout*** are sufficient in the axiom system (rules **Iv1** and **Iv2** are derivable, see Appendix B.1). Consider the equality

$$x(y:\mathtt{ii}T).y \eqsim_\Delta x(y:\mathtt{io}T).\mathtt{0}$$

where $\Delta \stackrel{\text{def}}{=} a:\mathtt{bib}T, b:\mathtt{ib}T, x:\mathtt{bib}T$. First, we infer

$$y =_{\Delta'} \mathtt{0} \quad \text{for } \Delta' = \Delta, y:\mathtt{ib}T \qquad (1)$$

proceeding as follows:

$$\begin{array}{lll} y & =_{\Delta'} & [y=b]y+[y\neq b]y \qquad \text{by } \mathbf{C8} \\ & =_{\Delta'} & [y=b]y \qquad \text{by } \mathbf{Tvar^*} \\ & =_{\Delta'} & [y=b]b \qquad \text{by } \mathbf{Tpre^*} \\ & =_{\Delta'} & [y=b]\mathsf{0} \qquad \text{by } \mathbf{Tin^*} \\ & =_{\Delta'} & \mathsf{0} \qquad \text{by } \mathbf{C3} \end{array}$$

Then we derive $x(y:\mathtt{ii}T).y =_\Delta x(y:\mathtt{io}T).\mathsf{0}$ in a similar way:

$$\begin{array}{lll} & x(y:\mathtt{ii}T).y & \\ =_\Delta & [x=a]x(y:\mathtt{ii}T).y+[x\neq a]x(y:\mathtt{ii}T).y & \text{by } \mathbf{C8} \\ =_\Delta & [x=a]x(y:\mathtt{ii}T).y & \text{by } \mathbf{Tvar^*} \\ =_\Delta & [x=a]a(y:\mathtt{ii}T).y & \text{by } \mathbf{Tpre^*} \\ =_\Delta & [x=a]a(y:\mathtt{io}T).\mathsf{0} & \text{by } \mathbf{(1), Iin^*, Icon} \\ =_\Delta & x(y:\mathtt{io}T).\mathsf{0} & \text{by } \mathbf{Tpre^*, Tvar^*, C8} \end{array}$$

4.3.2 Soundness and Completeness

The soundness of the axioms displayed in Table 4.5, and therefore of $\mathcal{A}$, is easy to be verified.

Theorem 4.16 (Soundness of $\mathcal{A}$) *If $\mathcal{A} \vdash P =_\Delta Q$ then $P \mathrel{\dot{=}}_\Delta Q$.*

The remainder of the section is devoted to proving the completeness of $\mathcal{A}$. The schema of the proof is similar to that for the untyped π-calculus [PS95]. The details, however, are quite different. An example of this is the manipulation of terms underneath input and output prefixes mentioned above. We discuss below another example, related to the issue of invariance of bisimilarity under injective substitutions. In the untyped case, the process $x \mid \overline{a}$ (the operational semantics of parallel composition is standard and will be given in Section 4.5) is equal to $x.\overline{a}+\overline{a}.x+\tau$ when x is instantiated to a, to $x.\overline{a}+\overline{a}.x$ otherwise. This can be expressed by expanding the process by means of conditions: that is, using conditions to make a case analysis on the possible values that the variable may take. Thus, $x \mid \overline{a}$ is expanded to $[x=a](x \mid \overline{a})+[x\neq a](x \mid \overline{a})$. Now, underneath $[x=a]$ we know that x will be a, and therefore $x \mid \overline{a}$ can be rewritten as $x.\overline{a}+\overline{a}.x+\tau$, whereas underneath $[x\neq a]$ we know that x will not be a and therefore $x \mid \overline{a}$ can be rewritten as $x.\overline{a}+\overline{a}.x$. In general, the expansion of a process with a free variable x produces a summand $[x\neq a_1]\cdots[x\neq a_n]P$ where $a_1,\cdots,a_n$ are all channels (different from x) that appear free in P. The mismatch $[x\neq a_1]\cdots[x\neq a_n]$ tells us that x in P will be instantiated to a fresh channel, which is sufficient for all manipulations of P involving x, since bisimulation is invariant under injective substitutions. In the typed calculus, by contrast, knowing that x is fresh may not be sufficient: we may also need the information on the type with which x will be instantiated. This type may be different from the type T of x in the type environment: x could be instantiated to a fresh channel whose type is a *subtype* of T (the behavioural consequences of this type information can be seen in the example at the end of Section 4.4.1). We have therefore adopted a strategy different from that in the proof for untyped calculi: rather than manipulating processes that begin with "complete" sequences of mismatches — as in the untyped case — we try to cancel them, using rule **Tvar***; further, the conditional expansion of a process takes into account also the names that appear in the type environment.

Definition 4.17 *A condition φ is satisfiable if $[\![\varphi\sigma]\!] = True$ for some closing substitution σ. Given a set of names V, a condition φ is complete on V if for some equivalence relation $\mathcal{R}$ on V, called the equivalence relation corresponding to φ, it holds that $\varphi \Rightarrow [u = v]$ iff $u\mathcal{R}v$ and $\varphi \Rightarrow [u \neq v]$ iff $\neg(u\mathcal{R}v)$, for any $u, v \in V$.*

In the untyped setting which does not distinguish channels from variables, like in [PS95], every complete condition is satisfiable, and two substitutions satisfying the same complete condition relate to each other by some injective substitution. In this chapter, however, due to the distinction between variables and channels and the concept of closing substitution, there exist some conditions which are complete but not satisfiable. For instance, $\varphi = [x = a] \wedge [a = b] \wedge [b \neq c]$ is complete on $V = \{x, a, b, c\}$, with the equivalence classes $\{\{x, a, b\}, \{c\}\}$. This condition is not satisfiable because closing substitutions do not map channels to other channels, then $\sigma(a) = a \neq b = \sigma(b)$ for any closing substitution σ, i.e., $[\![\varphi\sigma]\!] = False$. In a typed setting, there are even fewer conditions which are satisfiable. For a given type environment $\Delta = \Delta_{\mathsf{c}}, \widetilde{x} : \widetilde{T}$ we are only interested in closing substitutions of the form (called *legal substitution on* Δ): $\sigma = \{\widetilde{b}/\widetilde{x}\}$ where $\Delta_{\mathsf{c}} \vdash \widetilde{b} : \widetilde{T}$. As to the simple condition $[x_i = a]$, with $x_i, a \in dom(\Delta)$, if $\Delta(a) \not<: T_i$, the substitution $\{a/x_i\}$ is illegal and not considered. So no legal substitution can satisfy $[x_i = a]$, i.e., the condition is not satisfiable.

Lemma 4.18 *If φ is complete on $dom(\Delta)$ and $\emptyset \subset dom(\Delta_{\mathsf{v}}) \subset dom(\Delta)$, there is at most one legal substitution which satisfies φ.*

Proof: Since φ is complete, there is a corresponding equivalence relation $\mathcal{R}$. For φ to be satisfiable by a closing substitution σ on $dom(\Delta)$, each equivalence class of $\mathcal{R}$, say $\{u_1, \cdots, u_n\}$, must meet the following two conditions.

- Not all u_i are variables. Otherwise, for any $a \in dom(\Delta_{\mathsf{c}}), \varphi \Rightarrow [u_i \neq a]$. Then $\varphi\sigma \Rightarrow [\sigma(u_i) \neq a]$ for all $a \in dom(\Delta_{\mathsf{c}})$, contradicting the definition of closing substitution, which maps variables to channels, i.e., $\sigma(u_i) \in dom(\Delta_{\mathsf{c}})$.
- There is no more than one channel in any equivalence class. Otherwise, let a, b be two channels and $\varphi \Rightarrow [a = b]$, then $\varphi\sigma \Rightarrow [a = b]$, i.e., $[\![\varphi\sigma]\!] = False$.

As a result, in each equivalence class there is one and only one channel, possibly with some variables. So the class looks like $\{a, x_1, \cdots, x_{n-1}\}$ where $n \geq 1$. The substitution which satisfies φ must map all the variables in the equivalence class into its unique channel. Moreover, to ensure that φ is satisfied by a legal substitution, there is a third constraint imposed on the equivalence class:

- $\Delta(a) <: \Delta(x_i)$ for all $i \leq n - 1$.

All these conditions determine the uniqueness of the legal substitution, if it exists. □

Lemma 4.19 *If φ and ψ are complete conditions on $dom(\Delta)$ and are satisfied by the same legal substitution on Δ, then $\varphi \Longleftrightarrow \psi$.*

Proof: $\varphi \wedge \psi$ is also satisfiable by the same legal substitution. Then $\varphi \Longleftrightarrow \varphi \wedge \psi \Longleftrightarrow \psi$ because φ and ψ are complete conditions. □

The following lemma shows that in the presence of complete conditions, it is sufficient to test one substitution for typed bisimilarity of open terms.

Lemma 4.20 *Let $P \equiv \varphi P'$ and $Q \equiv \varphi Q'$, with φ complete on $dom(\Delta)$. If σ is a legal substitution on Δ, σ satisfies φ and $P\sigma \eqsim_{\Delta_c} Q\sigma$, then $P \eqsim_{\Delta} Q$.*

Proof: By Lemma 4.18, besides σ there is no other substitution $\rho = \{\widetilde{c}/\widetilde{x}\}$ with $\Delta_c \vdash \widetilde{c} : \widetilde{T}$ which can satisfy φ. In other words, $(\varphi P')\rho \eqsim_{\Delta_c} \mathsf{0} \eqsim_{\Delta_c} (\varphi Q')\rho$. Therefore we have $P \eqsim_{\Delta} Q$ by the definitions of typed bisimilarity. □

As in [PS95], the definition of head normal form exploits complete conditions. Here the difference is that we only consider those conditions which can be satisfied by some legal substitutions, while in [PS95] all complete conditions are involved because all of them are satisfiable.

Definition 4.21 (head normal form) *We say that P is in head normal form w.r.t. Δ if P is of the form*

$$\sum_i \varphi_i \alpha_i.\varphi_i' P_i$$

where for all i,

1. *$bn(\alpha_i) \notin dom(\Delta)$;*
2. *φ_i is complete on $dom(\Delta)$ and satisfiable by some legal substitution on Δ;*
3. *$\varphi_i' = \varphi_i$ if α_i is an input or free action;*
4. *$\varphi_i' = \varphi_i \wedge (\bigwedge_{v \in dom(\Delta)}[a \neq v])$ if $\alpha_i = \bar{u}(a:T)$.*

The proof of completeness is established by induction on the depth, $d(P)$, of a head norm form (hnf) P. Its depth is defined as:

$$\begin{array}{rcl} d(\mathsf{0}) & \stackrel{\text{def}}{=} & 0 \\ d(\sum_{i=1}^n \varphi_i \alpha_i.\varphi_i' P_i) & \stackrel{\text{def}}{=} & 1 + max\{d(P_i) \mid 1 \leq i \leq n\} \end{array}$$

Lemma 4.22 *For each process P and environment Δ, with $fv(P) \subseteq dom(\Delta_v)$, there is some H of no greater depth than P and in hnf w.r.t. Δ, such that $\mathcal{A} \vdash P =_{\Delta} H$.*

Proof: By structural induction on processes. Let $V = dom(\Delta)$. We consider two interesting cases.

The first is when $P \equiv \alpha.P'$. Let x be any variable in V. If for each channel $a \in V$, $\Delta(a) \not<: \Delta(x)$, then we use **Tv1** to derive that $\mathcal{A} \vdash P =_{\Delta} \mathsf{0}$. Otherwise, suppose $V_x = \{a_1, \cdots, a_n\}$ collects all channels in V such that $\Delta(a_i) <: \Delta(x)$. As in the untyped setting [PS95] we can infer that $\mathcal{A} \vdash P =_{\Delta} \sum_{i=1}^m \psi_i \alpha.\psi_i P'$, where each ψ_i is complete on V, but not necessarily satisfiable by some legal substitution on Δ. There are two occasions where ψ_i is not satisfiable.

1. If $\psi_i \Rightarrow [a = b]$ for $a, b \in dom(\Delta_c)$ and $a \neq b$, we use **Cnn1** to get $\mathcal{A} \vdash \psi_i \alpha.\psi_i P' =_{\Delta} \mathsf{0}$.
2. If $\psi_i \Rightarrow [x \neq a_1] \cdots [x \neq a_n]$ we can use **Tvar*** to derive that $\mathcal{A} \vdash \psi_i \alpha.\psi_i P' =_{\Delta} \mathsf{0}$.

So we can remove the summand $\psi_i\alpha.\psi_i P'$ if ψ_i is not satisfiable. All other summands are satisfiable by some legal substitutions because $\psi_i \Rightarrow [x = a_i]$ for one $a_i \in V_x$ and $\psi_i \Rightarrow [x \neq b]$ for any other $b \in dom(\Delta_{\mathsf{c}})$.

The second case is when $P \equiv \psi\ Q\ R$. By induction hypothesis Q and R can be transformed into hnf w.r.t. Δ: $\mathcal{A} \vdash Q =_\Delta \sum_{i=1}^n \psi_i\alpha_i.\psi_i' Q_i$ and $\mathcal{A} \vdash R =_\Delta \sum_{j=1}^m \psi_j\alpha_j.\psi_j' R_j$. Let us examine the general case that $n, m > 0$. By **C9** and **C11**, it is easy to see that

$$\mathcal{A} \vdash P =_\Delta \sum_{i=1}^n [\psi \wedge \psi_i]\alpha_i.\psi_i' Q_i + \sum_{j=1}^m [\neg\psi \wedge \psi_j]\alpha_j.\psi_j' R_j.$$

Clearly ψ can be reduced to a disjunctive normal form $\bigvee_{k=1}^s \bigwedge_{l=1}^t \varphi_{kl}$ where $s, t \geq 1$ and φ_{kl} is a match $[u_{kl} = v_{kl}]$ or mismatch $[u_{kl} \neq v_{kl}]$. Let $Q_i' = \alpha_i.\psi_i' Q_i$. We transform each summand $[\psi \wedge \psi_i]Q_i'$ as follows.

$$\begin{array}{lll} \mathcal{A} \vdash [\psi \wedge \psi_i]Q_i' & =_\Delta & [(\bigvee_{k=1}^s \bigwedge_{l=1}^t \varphi_{kl}) \wedge \psi_i]Q_i' \qquad \text{by } \mathbf{C1} \\ & =_\Delta & [\bigvee_{k=1}^s (\psi_i \wedge \bigwedge_{l=1}^t \varphi_{kl})]Q_i' \qquad \text{by } \mathbf{C1} \\ & =_\Delta & \sum_{k=1}^s [\psi_i \wedge \bigwedge_{l=1}^t \varphi_{kl}]Q_i' \qquad \text{by } \mathbf{C10} \end{array}$$

Now we assert that each summand $[\psi_i \wedge \bigwedge_{l=1}^t \varphi_{kl}]Q_i'$ is provably equal to $\mathsf{0}$ or $\psi_i Q_i'$.

Let $\phi_k = \bigwedge_{l=2}^t \varphi_{kl}$ if $t > 1$, and $\phi_k = \mathit{True}$ if $t = 1$. So by **C1** we have $\mathcal{A} \vdash [\psi_i \wedge \bigwedge_{l=1}^t \varphi_{kl}]Q_i' =_\Delta [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i'$. Here φ_{k1} may be a match or mismatch. We look at match first. Let $\varphi_{k1} = [u_{k1} = v_{k1}]$ for some u_{k1}, v_{k1} s.t. $u_{k1} \neq v_{k1}$.

1. If $u_{k1}, v_{k1} \in V$, then $[\varphi_{k1} \wedge \phi_k \wedge \psi_i]$ is semantically equivalent either to False or to $[\phi_k \wedge \psi_i]$ because ψ_i is complete on V. That is, we can infer $\mathcal{A} \vdash [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i' =_\Delta \mathsf{0}$ or $\mathcal{A} \vdash [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i' =_\Delta [\phi_k \wedge \psi_i]Q_i'$.

2. If $u_{k1}, v_{k1} \notin V$, then u_{k1}, v_{k1} are channels because $fv(P) \subseteq V$. By **Cnn1** we get $\mathcal{A} \vdash [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i' =_\Delta \mathsf{0}$.

3. If $u_{k1} \in V$ and $v_{k1} \notin V$, then v_{k1} is a channel but u_{k1} can be either a channel or a variable.

   (a) u_{k1} is also a channel. We infer $\mathcal{A} \vdash [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i' =_\Delta \mathsf{0}$ by **Cnn1**.

   (b) u_{k1} is a variable, i.e., $u_{k1} \in \widetilde{x}$. We infer $\mathcal{A} \vdash [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i' =_\Delta \mathsf{0}$ by **Tvn1**.

When φ_{k1} is a mismatch $[u_{k1} \neq v_{k1}]$ we apply similar arguments. In Case 1 the result is the same. In the last two cases, using **Cnn2** or **Tvn2** we infer that $\mathcal{A} \vdash [\varphi_{k1} \wedge \phi_k \wedge \psi_i]Q_i' =_\Delta [\phi_k \wedge \psi_i]Q_i'$. Since there are only t components in $\bigwedge_{l=1}^t \varphi_{kl}$, we can repeat this inference for at most t times and eventually get either $\mathcal{A} \vdash [\psi_i \wedge \bigwedge_{l=1}^t \varphi_{kl}]Q_i' =_\Delta \mathsf{0}$ or $\mathcal{A} \vdash [\psi_i \wedge \bigwedge_{l=1}^t \varphi_{kl}]Q_i' =_\Delta \psi_i Q_i'$.

Similar result can be got for $[\neg\psi \wedge \psi_j]\alpha_j.\psi_j' R_j$ as well.

In summary we have shown that each summand of P can either be removed or put into the form of the summands of a hnf. □

Theorem 4.23 (Completeness of $\mathcal{A}$) *If $P \mathrel{\widehat{=}}_\Delta Q$ then $\mathcal{A} \vdash P =_\Delta Q$.*

Proof: Let $\Delta = \Delta_{\mathtt{c}}, \widetilde{x} : \widetilde{T}$. If there is no legal substitution on Δ, i.e., no $\widetilde{a}$ with $\Delta_{\mathtt{c}} \vdash \widetilde{a} : \widetilde{T}$, then by **Tv1** we have that $\mathcal{A} \vdash P =_\Delta \mathtt{0} =_\Delta Q$.

Below we suppose that there exist legal substitutions on Δ. By Lemma 4.22 we assume that P and Q are in hnf w.r.t. Δ. Let

$$\mathcal{A} \vdash P =_\Delta \sum_i \varphi_i \alpha_i.P_i \qquad \text{and} \qquad \mathcal{A} \vdash Q =_\Delta \sum_j \psi_j \beta_j.Q_j.$$

For any summand $\varphi_i\alpha_i.P_i$ of P, let σ_i be a legal substitution on Δ which satisfies φ_i (actually σ_i is the only legal substitution satisfying φ_i, according to Lemma 4.18). So if $\varphi_i \Rightarrow [x = a]$ then $\Delta(a) <: \Delta(x)$ and $x\sigma_i = a$. By using **Tpre*** we can transform the action α_i into $\alpha_i\sigma_i$ which contains no free variable. For example, if $\alpha_i = \bar{x}y$ and $\varphi_i \Rightarrow [x = a] \wedge [y = b]$, then $\varphi_i\bar{x}y.P_i =_\Delta \varphi_i\overline{x\sigma_i}y\sigma_i.P_i \equiv \varphi_i\overline{a}b.P_i$. Furthermore, if the action $\alpha_i\sigma_i$ is disallowed by the environment (e.g., $\alpha_i\sigma_i = \bar{a}b$ and $\Delta(a)\not/_{\mathtt{i}}$, similar for input actions), then by **Tin*** and **Tout*** the summand $\varphi_i\alpha_i.P_i$ is provably equal to $\mathtt{0}$ and thus can be consumed by **S1**. After finite steps of transformation, all remaining summands are active, i.e., can perform some actions allowed by Δ. We do similar transformation for Q.

Now we prove by induction on the depth of $P + Q$ that each active summand of P is provably equal to some active summand of Q. An active summand $\varphi_i\alpha_i.P_i$ of P gives rise to a transition $\Delta_{\mathtt{c}} \sharp P\sigma_i \xrightarrow{\alpha_i\sigma_i} \Delta'_{\mathtt{c}} \sharp P_i\sigma_i$. Since $P \mathrel{\widehat{=}}_\Delta Q$, we have $P\sigma_i \mathrel{\widehat{=}}_{\Delta_{\mathtt{c}}} Q\sigma_i$. So there is a matching transition $\Delta_{\mathtt{c}} \sharp Q\sigma_i \xrightarrow{\beta_j\sigma_i} \Delta''_{\mathtt{c}} \sharp Q_j\sigma_i$ contributed by some active summand $\psi_j\beta_j.Q_j$ of Q, with ψ_j satisfied by σ_i. By Lemma 4.19 we know that $\varphi_i \Longleftrightarrow \psi_j$. From the definition of $\mathrel{\widehat{=}}_{\Delta_{\mathtt{c}}}$ we have:

1. if $\alpha_i\sigma_i = \beta_j\sigma_i = \tau$ then $P_i\sigma_i \mathrel{\widehat{=}}_{\Delta_{\mathtt{c}}} Q_j\sigma_i$;
2. if $\alpha_i\sigma_i = \beta_j\sigma_i = \bar{a}b$, for some channels a, b, then $P_i\sigma_i \mathrel{\widehat{=}}_{\Delta_{\mathtt{c}} \sqcap b:\Delta(a)_{\mathtt{i}}} Q_j\sigma_i$;
3. if $\alpha_i\sigma_i = \bar{a}(b : T_1)$ and $\beta_j\sigma_i = \bar{a}(b : T_2)$ for some channels a, b then $P_i\sigma_i \mathrel{\widehat{=}}_{\Delta_{\mathtt{c}}, b:\Delta(a)_{\mathtt{i}}} Q_j\sigma_i$;
4. if $\alpha_i\sigma_i = a(x : T_1)$ and $\beta_j\sigma_i = a(x : T_2)$, for some a and x, then for all c with $\Delta_{\mathtt{c}} \vdash c : \Delta(a)_{\mathtt{o}}$ it holds that $P_i\sigma_i\{c/x\} \mathrel{\widehat{=}}_{\Delta_{\mathtt{c}}} Q_j\sigma_i\{c/x\}$.

Let us analyze the last two cases in details. In Case 3, σ_i is also a legal substitution on $\Delta, b : \Delta(a)_{\mathtt{i}}$. By Lemma 4.20 one can infer that $P_i \mathrel{\widehat{=}}_{\Delta, b:\Delta(a)_{\mathtt{i}}} Q_j$. By induction hypothesis $\mathcal{A} \vdash P_i =_{\Delta, b:\Delta(a)_{\mathtt{i}}} Q_j$. By **Iout*, Ires*, Icon** and **C1** it can be inferred that $\mathcal{A} \vdash \varphi_i\bar{a}(b : T_1).P_i =_\Delta \psi_j\bar{a}(b : T_2)Q_j$. The required result is got by using **Tpre***.

In Case 4, we have that $P_i\sigma_i\{c/x\} \mathrel{\widehat{=}}_{\Delta_{\mathtt{c}}} Q_j\sigma_i\{c/x\}$ for all c satisfying the condition $\Delta_{\mathtt{c}} \vdash c : \Delta(a)_{\mathtt{o}}$. Note that $P_i = \varphi_i P'_i$ and $Q_j = \psi_j Q'_j$. By Lemma 4.18, any substitution $\rho = \{\widetilde{c}/\widetilde{x}, d/x\}$, with $\Delta_{\mathtt{c}} \vdash \widetilde{c} : \widetilde{T}, d : \Delta(a)_{\mathtt{o}}$, which can satisfy φ_i and ψ_j, must coincide with σ on variables $\widetilde{x}$. That is, $\rho = \sigma\{d/x\}$. Therefore $P_i\rho \mathrel{\widehat{=}}_{\Delta_{\mathtt{c}}} Q_j\rho$. For any other substitution, say ρ', $[\![\varphi_i\rho']\!] = [\![\psi_j\rho']\!] = \mathit{False}$, and so $P_i\rho' \mathrel{\widehat{=}}_{\Delta_{\mathtt{c}}} \mathtt{0} \mathrel{\widehat{=}}_{\Delta_{\mathtt{c}}} Q_j\rho'$. Consequently for all ρ we have $P_i\rho \mathrel{\widehat{=}}_{\Delta_{\mathtt{c}}} Q_j\rho$, i.e., $P_i \mathrel{\widehat{=}}_{\Delta, x:\Delta(a)_{\mathtt{o}}} Q_j$. Now applying induction hypothesis, $\mathcal{A} \vdash P_i =_{\Delta, x:\Delta(a)_{\mathtt{o}}} Q_j$. It follows that $\mathcal{A} \vdash a(x : T_1).P_i =_\Delta a(x : T_2).Q_j$ by **Iin***. Then we can infer $\mathcal{A} \vdash \varphi_i\alpha_i.P_i =_\Delta \psi_j\beta_j.Q_j$ by using **Icon, C1** and **Tpre***, in the listed order. □

4.4 Other Equivalences

In this section we study a variant bisimilarity proposed in [HR04], which allows extension of environments and enjoys a nice contextual property. Proof systems for closed terms are given. An indirect axiomatisation is got by resorting to the system $\mathcal{A}$ of Section 4.3. We also show that the difference between late and early style of typed bisimilarity is characterised by one axiom.

4.4.1 Hennessy and Rathke's Typed Bisimilarity

Proof System for Closed Terms

In the input clause of $\simeq$ (Definition 4.9), the type environment Δ is not extended. By contrast, extensions are allowed in the bisimilarity used in [HR04]. We denote with $\hat{\simeq}_\Delta$ the variant of $\simeq_\Delta$ which allows extension; its definition is obtained from that of $\simeq_\Delta$ by using the following input clause:

- if $\Delta \sharp P \stackrel{a(x:T)}{\longrightarrow} \Delta' \sharp P'$, then for some Q', $\Delta \sharp Q \stackrel{a(x:S)}{\longrightarrow} \Delta'' \sharp Q'$ and $\Delta, \Delta''' \vdash b : \Delta(a)_\circ$ implies $P'\{b/x\}\ \mathcal{R}_{\Delta,\Delta'''}\ Q'\{b/x\}$, for any channel b and closed type environment Δ''' with $dom(\Delta''') \cap (fn(P,Q) \cup dom(\Delta)) = \emptyset$.

Similarly, Δ can be extended in the definition on open terms.

Lemma 4.24 *If $P \hat{\simeq}_\Delta Q$ then $P \simeq_\Delta Q$.*

In $\hat{\simeq}_\Delta$, the environment collects the knowledge of the observer *relative* to the tested processes, in the sense that the environment only tells us what the observer knows of the free channels of the processes. In contrast, in $\simeq_\Delta$, the environment collects the *absolute* knowledge of the observer, including information on channels that at present do not appear in the tested processes, but that might appear later — if the observer decides to send them to the processes. The main advantage of $\hat{\simeq}_\Delta$ is that the environment is allowed to invent an unbounded number of distinct names, so it is more suitable for infinite systems. On the other hand, $\simeq_\Delta$ allows us to express more refined interrogations on the equivalence of processes, for it gives us more flexibility in setting the observer knowledge. Indeed, while $\hat{\simeq}$-equivalences can be expressed using $\simeq$ (Lemma 4.24), the converse is false. For instance, the processes

$$P \stackrel{\text{def}}{=} a(x : \mathtt{bo}T).[x = y]\tau \qquad Q \stackrel{\text{def}}{=} a(x : \mathtt{bo}T).\mathsf{0}$$

are in the relation $\simeq_\Delta$, for $\Delta \stackrel{\text{def}}{=} a : \mathtt{obo}T, b : \mathtt{bb}T, y : \mathtt{ob}T$. However, they are not in a relation $\hat{\simeq}_\Gamma$, for any Γ: the observer can always create a new channel of type $\mathtt{bo}T$, and use it to instantiate both x and y, thus validating the condition $[x = y]$.

In the following lemma we give two properties of $\hat{\simeq}_\Delta$. They are analogous to Lemma 4.11 and 4.13 respectively, and can be proved as their counterparts.

Lemma 4.25 *1. If $P \hat{\simeq}_\Delta Q$ and $\Delta \lessdot \Delta'$, then $P \hat{\simeq}_{\Delta'} Q$.*

2. If $P \hat{\simeq}_\Delta Q$ then $P\sigma \hat{\simeq}_{\Delta\sigma} Q\sigma$ for σ injective on $fn(P,Q) \cup dom(\Delta)$ and $\Delta\sigma$ is the type environment which maps $\sigma(u)$ to $\Delta(u)$ for all $u \in dom(\Delta)$.

An important property which is enjoyed by $\fallingdotseq_\Delta$ but not by $\doteq_\Delta$ is as follows.

Lemma 4.26 *If $P \fallingdotseq_\Delta Q$ and $a \notin fn(P,Q) \cup dom(\Delta)$, then $P \fallingdotseq_{\Delta,a:T} Q$.*

This lemma says that increasing capabilities on irrelevant channels does not raise an observer's discriminating power. The reason is that the observer already has the ability to create new channels, since in the definitions of bisimulations we test all channels with appropriate types for the case of input.

Lemma 4.27 *It holds that $a(x:T_1).P \fallingdotseq_\Delta a(x:T_2).Q$, if the following two conditions are satisfied.*
(i) $P\{b/x\} \fallingdotseq_\Delta Q\{b/x\}$ for all b with $\Delta_{\mathsf{c}} \vdash b : \Delta(a)_{\mathsf{o}}$;
(ii) given $c \notin fn(P,Q) \cup dom(\Delta)$, $P\{c/x\} \fallingdotseq_{\Delta,c:T} Q\{c/x\}$ for all $T \lessdot \Delta(a)_{\mathsf{o}}$.

Proof: The action of the configuration $\Delta\sharp a(x:T_1).P$ can be matched by that of $\Delta\sharp a(x:T_2).Q$. So we only show that $P\{b/x\} \fallingdotseq_{\Delta,\Delta'} Q\{b/x\}$ for any b and Δ' with $dom(\Delta') \cap fn(P,Q) = \emptyset$ and $\Delta,\Delta' \vdash b : \Delta(a)_{\mathsf{o}}$. There are two possibilities:

1. $b \in dom(\Delta)$. When $\Delta' = \emptyset$, the result follows from the hypothesis (i). For other Δ', we get the result indirectly by using Lemma 4.26.

2. $b \notin dom(\Delta)$. We consider the case that $\Delta' = b : T$ with $T \lessdot \Delta(a)_{\mathsf{o}}$. Base on this case, the result for other Δ' with $\Delta' = b : T, \Delta''$ can be inferred from Lemma 4.26. From (ii) we know that $P\{c/x\} \fallingdotseq_{\Delta,c:T} Q\{c/x\}$. Since bisimulation is insensitive to injective type-preserving substitutions by Lemma 4.25 (2), we have $P\{c/x\}\{b/c\} \fallingdotseq_{\Delta,b:T} Q\{c/x\}\{b/c\}$. That is, $P\{b/x\} \fallingdotseq_{\Delta,\Delta'} Q\{b/x\}$, which is the required result.

□

We can derive a proof system for $\fallingdotseq$ with a simple modification of that for $\doteq$ in Section 4.2. Let $\mathcal{P}'$ be the system obtained from $\mathcal{P}$ by replacing rule **Iinc*** with **Iinc′**:

Iinc′ If
- $P\{b/x\} =_\Delta Q\{b/x\}$ for all b with $\Delta_{\mathsf{c}} \vdash b : \Delta(a)_{\mathsf{o}}$, and
- given $c \notin fn(P,Q) \cup dom(\Delta)$, $P\{c/x\} =_{\Delta,c:T} Q\{c/x\}$ for all $T \lessdot \Delta(a)_{\mathsf{o}}$,

then $a(x:T_1).P =_\Delta a(x:T_2).Q$.

The quantification on T in the premises is finite: any type has only finitely many subtypes.

Theorem 4.28 *$\mathcal{P}' \vdash P =_\Delta Q$ iff $P \fallingdotseq_\Delta Q$, where P and Q are closed.*

Proof: According to Lemma 4.27, rule **Iinc′** is sound. The soundness of other rules is easy to show. The completeness proof is similar to that of $\mathcal{P}$ (Theorem 4.15). □

Indirect Axiomatisation

The previous definition of $\fallingdotseq$ involves infinitely many substitutions. Nevertheless we show in the following lemma that there exists an efficient characterisation of the equivalence which employs only finitely many substitutions. This characterisation result relies on the assumption that the set

of subtypes of any type is finite and the environment contains finitely many variables (the terms could even be extended with non-finite operators such as recursion, as long as they contain finitely many free variables). First, we introduce a notation. Let $\widetilde{T} = T_1, \cdots, T_n$. There are only finitely many different types, say $S_1, \ldots, S_m$, each of which is a subtype of some T_i for $i \leq n$. Then we pick n fresh names (which do not appear in Δ, P and Q) $a_{i1}, \cdots, a_{in}$ for each type S_i and extend Δ in the following way.

$$Env(\Delta, \widetilde{T}, P, Q) \stackrel{\text{def}}{=} \Delta \cup \{a_{ik} : S_i \mid 0 < i \leq m, 0 < k \leq n, a_{ik} \notin fn(\Delta, P, Q)\}$$

Lemma 4.29 *Suppose* $\Delta \stackrel{\text{def}}{=} \Delta_{\mathsf{c}}, \widetilde{x} : \widetilde{T}$. *If for each legal substitution* σ *on* $Env(\Delta, \widetilde{T}, P, Q)$ *it holds that* $P\sigma \mathrel{\dot{\approx}}_{Env(\Delta_{\mathsf{c}}, \widetilde{T}, P, Q)} Q\sigma$, *then* $P \mathrel{\dot{\approx}}_{\Delta} Q$.

Proof: Let $\Delta_1 = Env(\Delta_{\mathsf{c}}, \widetilde{T}, P, Q)$, and the length of the tuple $\widetilde{T}$ be n with $n > 0$. We prove a stronger result $P \mathrel{\dot{\approx}}_{\Delta_1, \widetilde{x}:\widetilde{T}} Q$ and then conclude by Lemma 4.25 (1). We shall show that $P\{\widetilde{b}/\widetilde{x}\} \mathrel{\dot{\approx}}_{\Delta_1, \Delta'} Q\{\widetilde{b}/\widetilde{x}\}$ for any $\widetilde{b}$ and closed environment Δ' s.t. $dom(\Delta') \cap fn(P, Q) = \emptyset$ and $\Delta_1, \Delta' \vdash \widetilde{b} : \widetilde{T}$. We proceed by induction on the number of names appearing in $\widetilde{b}$ but not in $dom(\Delta_1)$, which is defined as follows.

$$num(\emptyset) \stackrel{\text{def}}{=} 0$$

$$num(\widetilde{b}) \stackrel{\text{def}}{=} \begin{cases} num(b_1 \cdots b_{n-1}) + 1 & \text{if } b_n \notin dom(\Delta_1) \\ num(b_1 \cdots b_{n-1}) & \text{otherwise} \end{cases}$$

• Base step. Suppose $num(\widetilde{b}) = 0$. When $\Delta' = \emptyset$, the result follows from the hypothesis. For other Δ', the result is got indirectly by using Lemma 4.26.

• Inductive step. Suppose that the result holds for all $\widetilde{b}$ which satisfy the conditions in the hypothesis and $num(\widetilde{b}) \leq k$. Given another $\widetilde{b}$ with $num(\widetilde{b}) = k + 1$. Without loss of generality we assume that there exists a $c \notin dom(\Delta_1)$ and $l \leq n$ such that $b_1 = b_2 = \cdots = b_l = c$ and $b_i \neq c$ for all $i > l$. Then Δ_1, Δ' can be rewritten as $\Delta_2, c : S_i$ for some Δ_2 and S_i s.t. $S_i \leq T_j$ for all $j \leq l$. Choose one name from $\{a_{i1}, \cdots, a_{in}\}$, say a_{ij}, which is different from any names in $b_{l+1}, \cdots, b_n$, and construct a substitution

$$\sigma = \{a_{ij}/x_1, \cdots, a_{ij}/x_l, b_{l+1}/x_{l+1}, \cdots, b_n/x_n\}$$

Obviously $\Delta_2 \vdash a_{ij} : T_1, \cdots, a_{ij} : T_l, b_{l+1} : T_{l+1}, \cdots, b_n : T_n$ and $num(a_{ij}, \cdots, a_{ij}, b_{l+1}, \cdots, b_n) \leq k$. By induction hypothesis $P\sigma \mathrel{\dot{\approx}}_{\Delta_2} Q\sigma$. From Lemma 4.25 (2) we have

$$P\sigma\{c/a_{ij}\} \mathrel{\dot{\approx}}_{\Delta_2\{c/a_{ij}\}} Q\sigma\{c/a_{ij}\}$$

i.e., $P\{\widetilde{b}/\widetilde{x}\} \mathrel{\dot{\approx}}_{\Delta_2\{c/a_{ij}\}} Q\{\widetilde{b}/\widetilde{x}\}$. As $a_{ij} \notin dom(\Delta_2\{c/a_{ij}\})$, by Lemma 4.26 we get $P\{\widetilde{b}/\widetilde{x}\} \mathrel{\dot{\approx}}_{\Delta_3} Q\{\widetilde{b}/\widetilde{x}\}$ for $\Delta_3 = \Delta_2\{c/a_{ij}\}, a_{ij} : S_i = \Delta_1, \Delta'$, which is just the required result. □

Below we establish a property of $\mathrel{\dot{=}}_{\Delta}$, corresponding to Lemma 4.26 for $\mathrel{\dot{\approx}}_{\Delta}$. It allows the extension of Δ in a limited way. The proof employs the concept of *depth* of a process P, written $d(P)$, which we define as follows.

$$\begin{array}{rcl@{\qquad}rcl} d(\mathsf{0}) & \stackrel{\text{def}}{=} & 0 & d(P + Q) & \stackrel{\text{def}}{=} & max\{d(P), d(Q)\} \\ d(\alpha.P) & \stackrel{\text{def}}{=} & 1 + d(P) & d(\varphi PQ) & \stackrel{\text{def}}{=} & max\{d(P), d(Q)\} \\ d((\nu a : S)P) & \stackrel{\text{def}}{=} & d(P) & d(P \mid Q) & \stackrel{\text{def}}{=} & d(P) + d(Q) \end{array}$$

One can verify that if $\Delta \sharp P \xrightarrow{\alpha} \Delta' \sharp P'$ then $d(P) > d(P')$ and $fn(P') \subseteq fn(P) \cup bn(\alpha)$.

Lemma 4.30 *Given two closed terms P and Q, let $\Delta = \Delta_0, c_1 : T, ..., c_n : T$ with $n \geq d(P+Q)$ and $c_i \notin fn(P,Q)$ for all $i \in 1..n$. If $P \simeq_\Delta Q$ then $P \simeq_{\Delta, a:T} Q$ for $a \notin fn(P,Q) \cup dom(\Delta)$.*

Proof: By induction on the depth of $P + Q$. If $d(P+Q) = 0$ then it is obvious that $P \simeq_{\Delta,a:T} 0 \simeq_{\Delta,a:T} Q$. Below we suppose $d(P+Q) > 0$. If $\Delta, a : T \sharp P \xrightarrow{\alpha} \Delta' \sharp P'$ there must exist some Δ'' s.t. $\Delta' = \Delta'', a : T$ because a does not affect the transition. In other words, we have $\Delta \sharp P \xrightarrow{\alpha} \Delta'' \sharp P'$. Since $P \simeq_\Delta Q$, we have a matching transition $\Delta \sharp Q \xrightarrow{\beta} \Delta''' \sharp Q'$, where $\mid \alpha \mid = \mid \beta \mid$. It follows that $\Delta, a : T \sharp Q \xrightarrow{\beta} \Delta''', a : T \sharp Q'$. There are two cases:

1. α is not an input action. In this case $\Delta'' = \Delta'''$ and $P' \simeq_{\Delta''} Q'$. By induction hypothesis we have $P' \simeq_{\Delta'', a:T} Q'$.

2. α is an input action $b(x : S)$. Then for each d with $\Delta \vdash d : \Delta(b)_\circ$ it holds that $P'\{d/x\} \simeq_\Delta Q'\{d/x\}$.

   (a) If $d \in dom(\Delta_0)$ with $\Delta_0 \vdash d : \Delta(b)_\circ$, then $n \geq d(P+Q) > d(P'\{d/x\} + Q'\{d/x\})$ and $c_i \notin fn(P'\{d/x\}, Q'\{d/x\})$ for $i \in 1..n$. By induction hypothesis we have $P'\{d/x\} \simeq_{\Delta,a:T} Q'\{d/x\}$.

   (b) If $c_1 : T, ..., c_n : T \vdash d : \Delta(b)_\circ$, then without loss of generality we may assume that $d = c_1$. It can be checked that $n - 1 \geq d(P+Q) - 1 \geq d(P'\{d/x\} + Q'\{d/x\})$ and $c_i \notin fn(P'\{d/x\}, Q'\{d/x\})$ for $i \in 2..n$. We can now appeal to induction hypothesis and get the result that $P'\{d/x\} \simeq_{\Delta,a:T} Q'\{d/x\}$.

   (c) If $a : T \vdash a : \Delta(b)_\circ$, then $T <: \Delta(b)_\circ$ and thus $\Delta \vdash c_1 : \Delta(b)_\circ$, which implies $P'\{c_1/x\} \simeq_\Delta Q'\{c_1/x\}$. As $\{a/c_1\}$ is an injective type-preserving substitution, we have

   $$P'\{c_1/x\}\{a/c_1\} \simeq_{\Delta\{a/c_1\}} Q'\{c_1/x\}\{a/c_1\}$$

   i.e., $P'\{a/x\} \simeq_{\Delta\{a/c_1\}} Q'\{a/x\}$. Now observe that

   i. $n - 1 \geq d(P+Q) - 1 \geq d(P'\{a/x\} + Q'\{a/x\})$,

   ii. $c_i \notin fn(P'\{a/x\}, Q'\{a/x\})$ for $i \in 2..n$,

   iii. $c_1 \notin fn(P'\{a/x\}, Q'\{a/x\}) \cup dom(\Delta\{a/c_1\})$.

   By induction hypothesis we have $P'\{a/x\} \simeq_{\Delta\{a/c_1\}, c_1:T} Q'\{a/x\}$. Note that $\Delta\{a/c_1\}, c_1 : T = \Delta, a : T$.

In summary, for each d with $\Delta, a : T \vdash d : \Delta(b)_o$, it always holds that $P'\{a/x\} \simeq_{\Delta,a:T} Q'\{a/x\}$, which is the required result.

□

We know from Lemma 4.24 that $\simeq_\Delta$ is weaker than $\eqsim_\Delta$. This gives rise to an interesting question: whether there exists some Δ^* such that under the extended environment Δ, Δ^* we have that $P \simeq_{\Delta,\Delta^*} Q$ iff $P \eqsim_\Delta Q$. We shall give a positive answer to this question, though we did not

succeed in obtaining the counterpart of Theorem 4.23 for $\eqsim$. The encountered problem is discussed at the end of this subsection.

We define the depth, $d(T)$, of a type T, indicating the maximum number of nesting of capabilities in it.

$$d(\mathtt{unit}) = 0 \qquad d(\mathtt{i}T) = d(\mathtt{o}T) = 1 + d(T)$$
$$d(\mathtt{b}\langle T, S\rangle) = 1 + max\{d(T), d(S)\}$$

Let $\Gamma \vdash P$. Each name in P has a type, either recorded in the syntax of P or in Γ. If $T_1, \ldots, T_n$ are all such types, $d(\Gamma, P)$ is $max\{d(T_i) \mid 1 \leq i \leq n\}$. Now, if $\Delta \sharp P_i$ is a configuration, for $i = 1, 2$, then there are type environments Γ_i such that $\Gamma_i \lessdot \Delta$ and $\Gamma_i \vdash P_i$. In this case, we set $d(P_1, P_2, \Gamma_1, \Gamma_2)$ as $max\{d(\Gamma_1, P_1), d(\Gamma_2, P_2)\}$. There are only finitely many different types with depth less than or equal to $d(P_1, P_2, \Gamma_1, \Gamma_2)$, say $S_1, \ldots, S_m$, and $\Delta_{\mathtt{v}}$ is defined on finitely many variables, say $x_1, \ldots, x_k$. We can pick up n fresh (hitherto unused) channels $a_{i1}, \ldots, a_{in}$ for each S_i, where $n = max\{k, d(P_1 + P_2)\}$, and construct a type environment

$$Env(\Delta, P_1, P_2, \Gamma_1, \Gamma_2) = \{a_{ij} : S_{ij} \mid 0 < i \leq m, 0 < j \leq n\}.$$

We say that $P_1 \eqsim_\Delta P_2$ *under* Γ_1, Γ_2 if $\Gamma_i \lessdot \Delta$ and $\Gamma_i \vdash P_i$ $(i = 1, 2)$.

Lemma 4.31 *If* $P_1 \eqsim_\Delta P_2$ *under* Γ_1, Γ_2 *then* $P_1 \ \eqsim_{\Delta, Env(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)} \ P_2$.

Proof: By Lemma 4.26 we have $P_1 \ \eqsim_{\Delta, Env(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)} \ P_2$. Then the result follows from Lemma 4.24. □

In the above lemma, P_1, P_2 can be either closed or open. For the opposite direction, we consider closed terms first.

Lemma 4.32 *If* $\Delta \sharp P_i$ *respects* Γ_i, P_i *is closed, for* $i = 1, 2$, *and* $P_1 \ \eqsim_{\Delta, Env(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)} \ P_2$, *then* $P_1 \eqsim_\Delta P_2$.

Proof: By induction on the depth of $P_1 + P_2$. In the case $d(P_1 + P_2) = 0$, it is immediate that $P_1 \eqsim_\Delta \mathsf{0} \eqsim_\Delta P_2$. Below we suppose $d(P_1 + P_2) > 0$. Let $\Delta^* = Env(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)$. Since $dom(\Delta^*) \cap fn(P_1, P_2) = \emptyset$, all actions of the configuration $\Delta, \Delta^* \sharp P_1$ can be performed by $\Delta \sharp P_1$, and vice versa. Suppose that $\Delta \sharp P_1 \xrightarrow{\alpha} \Delta' \sharp P_1'$. It is easy to see that there is a matching transition $\Delta \sharp P_2 \xrightarrow{\beta} \Delta'' \sharp P_2'$.

1. If α is not an input action, then $\mid \alpha \mid = \mid \beta \mid$, $\Delta' = \Delta''$ and $P_1' \eqsim_{\Delta', \Delta^*} P_2'$. Suppose that $\Delta' \sharp P_i'$ respects Γ_i' for $i = 1, 2$. Clearly $d(P_1', P_2', \Gamma_1', \Gamma_2') \leq d(P_1, P_2, \Gamma_1, \Gamma_2)$ by Lemma 4.6. From Lemma 4.30 we have $P_1' \eqsim_{\Delta_1} P_2'$ where $\Delta_1 = \Delta', \Delta^*, Env(\Delta', P_1', P_2', \Gamma_1', \Gamma_2')$. Now it follows from Lemma 4.11 that $P_1' \ \eqsim_{\Delta', Env(\Delta', P_1', P_2', \Gamma_1', \Gamma_2')} \ P_2'$. By induction hypothesis we get $P_1' \eqsim_{\Delta'} P_2'$.

2. If α is an input action $a(x : T)$, then $P_1'\{b/x\} \ \eqsim_{\Delta, \Delta^*} \ P_2'\{b/x\}$ for all b with $\Delta, \Delta^* \vdash b : \Delta(a)_{\mathtt{o}}$. Note that $\Delta, \Delta^* \supseteq \Delta_1$ for some $\Delta_1 = Env(\Delta, \Delta(a)_{\mathtt{o}}, P_1', P_2')$ by the definition of $Env(\Delta, \widetilde{T}, P_1, P_2)$ given in the beginning of this subsection. So for all c with $\Delta_1 \vdash c : \Delta(a)_{\mathtt{o}}$

we have $P_1'\{c/x\} \;\dot{=}_{\Delta,\Delta^*}\; P_2'\{c/x\}$. It can be checked that $\Delta_1 \sharp P_i'\{c/x\}$ is a configuration respecting $\Gamma_i' \stackrel{\text{def}}{=} \Gamma_i, \Delta^*$ for $i = 1, 2$. As

$$d(\Delta_1, P_1'\{c/x\}, P_2'\{c/x\}, \Gamma_1', \Gamma_2') \leq d(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)$$

we have $P_1'\{c/x\} \dot{=}_{\Delta_2} P_2'\{c/x\}$, where

$$\Delta_2 = \Delta, \Delta^*, Env(\Delta_1, P_1'\{c/x\}, P_2'\{c/x\}, \Gamma_1', \Gamma_2')$$

by Lemma 4.30. It follows from Lemma 4.11 that $P_1'\{c/x\} \dot{=}_{\Delta_3} P_2'\{c/x\}$ where $\Delta_3 = \Delta_1$, $Env(\Delta_1, P_1'\{c/x\}, P_2'\{c/x\}, \Gamma_1', \Gamma_2')$. By induction hypothesis we get $P_1'\{c/x\} \mathrel{\dot{\approx}}_{\Delta_1} P_2'\{c/x\}$. By Lemma 4.29 it follows that $P_1' \mathrel{\dot{\approx}}_{\Delta, x:\Delta(a)_\circ} P_2'$, which is the required result.

□

Lemma 4.33 *If $\Delta \sharp P_i$ respects Γ_i, for $i = 1, 2$, and $P_1 \;\dot{=}_{\Delta, Env(\Delta,P_1,P_2,\Gamma_1,\Gamma_2)}\; P_2$ then $P_1 \mathrel{\dot{\approx}}_\Delta P_2$.*

Proof: Similar to the second case of the proof in Lemma 4.32. Let $\Delta = \Delta_{\mathtt{c}}, \widetilde{x} : \widetilde{T}$ and $\Delta^* = Env(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)$. Then for any legal substitution σ on Δ, Δ^* we have that $P_1\sigma \;\dot{=}_{\Delta_{\mathtt{c}}, \Delta^*}\; P_2\sigma$. We also have $\Delta_{\mathtt{c}}, \Delta^* \supseteq \Delta_1$ for some $\Delta_1 = Env(\Delta_{\mathtt{c}}, \widetilde{T}, P_1, P_2)$. So for all $\rho = \{\widetilde{c}/\widetilde{x}\}$ with $\Delta_1 \vdash \widetilde{c} : \widetilde{T}$ we have $P_1\rho \;\dot{=}_{\Delta_{\mathtt{c}}, \Delta^*}\; P_2\rho$. One can prove that $\Delta_1 \sharp P_i\rho$ is a configuration respecting $\Gamma_i' \stackrel{\text{def}}{=} \Gamma_i, \Delta^*$. Obviously $d(\Delta_1, P_1\rho, P_2\rho, \Gamma_1', \Gamma_2') = d(\Delta, P_1, P_2, \Gamma_1, \Gamma_2)$, so $P_1\rho \;\dot{=}_{\Delta_2}\; P_2\rho$ for some environment $\Delta_2 = \Delta_{\mathtt{c}}, \Delta^*$, $Env(\Delta_1, P_1\rho, P_2\rho, \Gamma_1', \Gamma_2')$. It follows that $P_1\rho \;\dot{=}_{\Delta_1, Env(\Delta_1, P_1\rho, P_2\rho, \Gamma_1', \Gamma_2')}\; P_2\rho$. By Lemma 4.32 we have $P_1\rho \mathrel{\dot{\approx}}_{\Delta_1} P_2\rho$, which implies $P_1 \mathrel{\dot{\approx}}_\Delta P_2$ by Lemma 4.29. □

Combining Lemma 4.31 and 4.33 we have the result below.

Lemma 4.34 *$P_1 \mathrel{\dot{\approx}}_\Delta P_2$ under Γ_1, Γ_2 iff $P_1 \;\dot{=}_{\Delta, Env(\Delta,P_1,P_2,\Gamma_1,\Gamma_2)}\; P_2$.*

As a consequence of this lemma, we obtain the following theorem.

Theorem 4.35 *$P_1 \mathrel{\dot{\approx}}_\Delta P_2$ under Γ_1, Γ_2 iff $\mathcal{A} \vdash P_1 =_{\Delta, Env(\Delta,P_1,P_2,\Gamma_1,\Gamma_2)} P_2$.*

Directly axiomatizing $\mathrel{\dot{\approx}}$ appears far from straightforward due to complications entailed by subtyping. We consider an example. Let $T \stackrel{\text{def}}{=} \mathtt{unit}$ and

$$\begin{array}{rcl} \Delta & \stackrel{\text{def}}{=} & a : \mathtt{obo}T, y : \mathtt{ob}T \\ R & \stackrel{\text{def}}{=} & \tau.((\nu c : \mathtt{b}T)\bar{y}c.\bar{c} + a(x : \mathtt{bo}T).[x = y]\tau) \\ R_1 & \stackrel{\text{def}}{=} & \tau.((\nu c : \mathtt{b}T)\bar{y}c.\mathsf{0} + a(x : \mathtt{bo}T).[x = y]\tau) \\ R_2 & \stackrel{\text{def}}{=} & \tau.((\nu c : \mathtt{b}T)\bar{y}c.\bar{c} + a(x : \mathtt{bo}T).\mathsf{0}). \end{array}$$

It holds that

$$R + R_1 + R_2 \mathrel{\dot{\approx}}_\Delta R_1 + R_2.$$

Here y can be instantiated by channels with subtypes of $\mathtt{ob}T$, which can be seen in Figure 1.2 (b). When y is instantiated by a channel with type $\mathtt{bo}T$, we can simulate R with R_1. For other subtypes of $\mathtt{ob}T$, we can simulate R with R_2. That is, we have two equivalent processes, say P and Q, with a free variable y, and the actions from a summand of P have to be matched by different summands of Q, depending on the types of the channels used to instantiate y. It appears hard to capture this relationship among terms using axioms involving only the standard operators of the π-calculus.

4.4.2 Early Bisimilarity

All bisimilarities considered so far in this chapter are in the late style. As usual, the early versions are obtained by commuting the quantifiers in the input clause of bisimilarity. For example, typed early bisimulation is defined as in Definition 4.9 except for using the following input clause:

- if $\Delta\sharp P \stackrel{a(x:T)}{\longrightarrow} \Delta'\sharp P'$, then for each b with $\Delta_{\mathsf{c}} \vdash b : \Delta(a)_{\mathsf{o}}$ there exists some Q' such that $\Delta\sharp Q \stackrel{a(x:S)}{\longrightarrow} \Delta''\sharp Q'$ and $P'\{b/x\}\ \mathcal{R}_\Delta\ Q'\{b/x\}$.

As in the untyped case, the difference between late and early equivalences is captured by the axiom **SP** [PS95]:

$$\begin{array}{lll} \textbf{SP} & & a(x:T_1).P + a(x:T_2).Q \\ & =_\Delta & a(x:T_1).P + a(x:T_2).Q + a(x:T_3).([x=u]PQ) \end{array}$$

All results in this chapter also hold for the early versions of the equivalences, when rule **SP** is added. For example, by letting the early version of $\simeq$ be $\simeq^e$, $\mathcal{A}_e$ be $\mathcal{A}\cup\{\textbf{SP}\}$ and $\mathcal{P}_e$ be $\mathcal{P}\cup\{\textbf{SP}\}$, we can establish the counterparts of Theorem 4.15 and 4.23.

Theorem 4.36 *1. $P \simeq^e_\Delta Q$ iff $\mathcal{P}_e \vdash P =_\Delta Q$, where P and Q are closed;*

2. $P \simeq^e_\Delta Q$ iff $\mathcal{A}_e \vdash P =_\Delta Q$.

Proof: See Appendix B.2. □

4.5 Adding Parallelism

So far the only π-calculus operator that we have not considered is parallel composition. When it is admitted, Table 4.1 should be extended with the following three transition rules (their symmetric rules are omitted).

$$\mathsf{par}\ \frac{P \stackrel{\alpha}{\longrightarrow} P' \quad bn(\alpha)\cap fn(Q)=\emptyset}{P \mid Q \stackrel{\alpha}{\longrightarrow} P' \mid Q} \qquad \mathsf{com}\ \frac{P \stackrel{\bar{a}b}{\longrightarrow} P' \quad Q \stackrel{a(x:S)}{\longrightarrow} Q'}{P \mid Q \stackrel{\tau}{\longrightarrow} P' \mid Q'\{b/x\}}$$

$$\mathsf{close}\ \frac{P \stackrel{\bar{a}(b:T)}{\longrightarrow} P' \quad Q \stackrel{a(x:S)}{\longrightarrow} Q'}{P \mid Q \stackrel{\tau}{\longrightarrow} (\nu b:T)(P' \mid Q'\{b/x\})}$$

In the typed setting, we incorporate the standard typing rule

$$\frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}$$

into Table 4.2. The TLTS shown in Table 4.3 is now extended with one rule:

$$\mathsf{Par}\ \frac{\Delta \sharp P \stackrel{\alpha}{\longrightarrow} \Delta' \sharp P' \quad bn(\alpha)\cap fn(Q)=\emptyset}{\Delta \sharp P \mid Q \stackrel{\alpha}{\longrightarrow} \Delta' \sharp P' \mid Q}$$

After the above modifications, all definitions and results in Section 4.1 are still valid.

To lift the results in Section 4.2, 4.3 and 4.4 to the full π-calculus, it suffices to enrich Table 4.4 with the two rules in Table 4.6. As in untyped π-calculus, the expansion law **E*** is used to reduce the parallel composition of two terms into the sum of parallel-free terms. In the typed setting we add

Ipar* Assume $\Delta_0 \sharp P$ respects Γ_1, $\Delta_0 \sharp Q$ respects Γ_2, and $\Delta = \Delta_0, Env(\Delta_0, P, Q, \Gamma_1, \Gamma_2)$. If $P =_\Delta Q$ and $\Delta \vdash R$ then $P \mid R =_\Delta Q \mid R$

E* Assume $P \equiv \Sigma_i \varphi_i \alpha_i.P_i$ and $Q \equiv \Sigma_j \psi_j \beta_j.Q_j$ where no α_i (resp. β_j) binds a name free in Q (resp. P). Let $\Delta \sharp P \mid Q$ respect Γ. Then infer:

$$P \mid Q =_\Delta \sum_i \varphi_i \alpha_i.(P_i \mid Q) + \sum_j \psi_j \beta_j.(P \mid Q_j) + \sum_{\alpha_i \ opp \ \beta_j} [\varphi_i \wedge \psi_j \wedge (u_i = v_j)]\tau.R_{ij}$$

where $\alpha_i \ opp \ \beta_j, u_i, v_j$ and R_{ij} are defined as follows:

1. α_i is $\bar{u}_i w$, β_j is $v_j(x:T)$ and $\Gamma(w) <: T$; then R_{ij} is $P_i \mid Q_j\{w/x\}$;
2. α_i is $\bar{u}_i(w:S)$, β_j is $v_j(x:T)$ and $S <: T$; then R_{ij} is $(\nu w:S)(P_i \mid Q_j\{w/x\})$;
3. the converse of (1) or (2).

Table 4.6: Two rules for parallel composition

conditions on types in order to check the typability of the resulting process R_{ij}. Rule **Ipar*** says that if Δ cannot distinguish P from Q, then it cannot distinguish $P \mid R$ from $Q \mid R$ either, provided that: (i) Δ contains enough fresh channels; (ii) R requires no capabilities beyond the knowledge of Δ. Note that we cannot do without the first condition, i.e., the rule cannot be simplified as:

$$\textit{For any } \Delta, \textit{ if } P =_\Delta Q \textit{ and } \Delta \vdash R \textit{ then } P \mid R =_\Delta Q \mid R$$

which is unsound for $\eqsim$ (though it is sound for $\dot{\eqsim}$). The point is that when comparing $P \mid R$ and $Q \mid R$, the observer may first increase his knowledge by interacting with R, then distinguish P from Q by the new knowledge. For example, let $\Delta \stackrel{\text{def}}{=} a : \mathtt{b}T, e : \mathtt{b}T, b : T$ and

$$P \stackrel{\text{def}}{=} a(x:T).[x \neq b]\tau \qquad Q \stackrel{\text{def}}{=} a(x:T).\mathbf{0} \qquad R \stackrel{\text{def}}{=} (\nu c:T)\bar{e}c.$$

It is easy to see that $P \eqsim_\Delta Q$ and $\Delta \vdash R$ but $P \mid R \not\eqsim_\Delta Q \mid R$. After the interaction with R, the environment evolves into $\Delta, c : T$. Later the new channel c may be used to instantiate x, thus validating the condition $x \neq b$ and liberating the prefix τ.

The soundness of **E*** is easy to show. To prove that **Ipar*** is sound, we define a family of relations $\mathcal{R} = \{\mathcal{R}_\Delta\}_\Delta$ where

$$\begin{aligned}\mathcal{R}_\Delta = \{((\nu \widetilde{a} : \widetilde{T_1})(P \mid R), (\nu \widetilde{a} : \widetilde{T_2})(Q \mid R)) \;\big|\; & P \eqsim_{\Delta \sqcap \Delta'} Q,\ \Delta \sqcap \Delta' \vdash R, \\ & \Delta = \Delta_0, Env(\Delta_0, P, Q, \Gamma_1, \Gamma_2),\ \Delta_0 \sqcap \Delta' \sharp P \text{ respects } \Gamma_1, \widetilde{a} : \widetilde{T_1}, \\ & \text{and } \Delta_0 \sqcap \Delta' \sharp Q \text{ respects } \Gamma_2, \widetilde{a} : \widetilde{T_2}, \text{ for some } \Delta_0, \Delta', \Gamma_1, \Gamma_2\}.\end{aligned}$$

Then it can be proved that $\mathcal{R}$ is a typed bisimulation.

In general, if $P \eqsim_\Delta Q$ then the equality $P =_\Delta Q$ can be inferred in two steps:

1. By **E***, **Ipar*** and **Twea*** we infer $P =_\Delta P'$ and $Q =_\Delta Q'$, where both P' and Q' are parallel-free terms.

2. After the above preprocessing job, we infer $P' =_\Delta Q'$ by the proof systems and axiomatisations presented in previous sections.

4.6 Summary

In this chapter we have constructed a proof system and an axiom system for typed bisimilarity ($\eqsim$). For the variant bisimilarity proposed in [HR04], we have provided a proof system for closed terms, and an indirect axiomatisation of all terms that depends on the system of $\eqsim$. Early versions of the systems are obtained by adding one axiom **SP**. All the systems are proved to be sound and complete.

As partial meet and join operators do not exist in the original capability types [PS96], we adopt in this chapter one of their extensions, Hennessy and Rathke's types [HR04]. An alternative path to take is to go in the opposite direction and add some syntactic constraints to capability types, thus only certain shapes of types are legal and partial meet and join operators exist upon the legal types. For instance, in synchronous localised π-calculus there are two forms of legal types: $\mathtt{oo}\cdots\mathtt{o}B$ and $\mathtt{bo}\cdots\mathtt{o}B$ where B is a basic type. It is easy to see that the two operators exist because whenever $T <\!: S$ holds, then either $T \equiv S$ or $T \equiv \mathtt{b}T', S \equiv \mathtt{o}T'$ for some T', which means:

1. if $T <\!: T_1, T_2$ and $T_1 \not\equiv T_2$ then $T_1 \sqcap T_2 = T$;
2. if $T_1, T_2 <\!: T$ and $T_1 \not\equiv T_2$ then $T_1 \sqcup T_2 = T$.

Therefore axiomatisation in synchronous localised π-calculus is a special case of the problem addressed in this chapter.

Chapter 5

Termination of Mobile Processes by Typability

Many modern programming languages are equipped with some notions of typing to statically detect programming errors. In mobile process calculi types are shown to be useful for reasoning about the behaviour of processes. In this chapter we use type-based method to reason about the terminating behaviour of mobile processes.

We give four type systems that ensure termination of well-typed π-calculus processes. The systems are obtained by successive refinements of the types of the simply typed π-calculus. For all (but one of) the type systems we also present upper bounds to the number of steps well-typed processes take to terminate. The termination proofs use techniques from term rewriting systems.

We show the usefulness of the type systems on some non-trivial examples: the encodings of primitive recursive functions, the protocol for encoding separate choice in terms of parallel composition, a symbol table implemented as a dynamic chain of cells.

5.1 Preliminary Notations

To begin with, we introduce some notations about vectors, partial orders and multisets. We write $\mathbf{0}_i$ as the abbreviation of a vector $\langle n_k, n_{k-1}, \cdots, n_1 \rangle$ where $k \geq 1$, $n_i = 1$ and $n_j = 0$ for all $j \neq i$ $(1 \leq i, j \leq k)$, and $\mathbf{0}$ for a vector with all 0 components. The binary operator *sum* can be defined between two vectors. Let $\varphi_1 \stackrel{\text{def}}{=} \langle n_k, n_{k-1}, \cdots, n_1 \rangle$, $\varphi_2 \stackrel{\text{def}}{=} \langle m_l, m_{l-1}, \cdots, m_1 \rangle$ and $k \geq l$. First we extend the length of φ_2 to k by inserting $(k-l)$ zeros to the left of m_l to get an equivalent vector φ_2'. Then we do pointwise addition over two vectors with equal length. We also define an order between two vectors of equal length as follows: $\langle n_k, n_{k-1}, \cdots, n_1 \rangle \prec \langle m_k, m_{k-1}, \cdots, m_1 \rangle$ iff $\exists i \leq k$ with $n_j = m_j$ for $j > i$ and $n_i < m_i$.

Let U be a set and $>$ a strict partial order on U. Following [Bez03], we write a multiset $\mathcal{M}$ *over* U in the form $\mathcal{M} = [x_1, \ldots, x_n]$, where $x_i \in U$ for $1 \leq i \leq n$ (when $n = 0$ we get the empty multiset $[\,]$); we use $(\mathcal{M} \uplus \mathcal{M}')$ for the *union* of $\mathcal{M}$ and $\mathcal{M}'$, and write $>_{mul}$ for the multiset ordering (on multisets over U) induced by $>$. A multiset becomes smaller, in the sense of $>_{mul}$, by replacing one

$$\textsf{T-in}\ \frac{\vdash u : \sharp^n V \quad x : V \quad \vdash P}{\vdash u(x).P} \qquad \textsf{T-out}\ \frac{\vdash u : \sharp^n V \quad \vdash w : V \quad \vdash P}{\vdash \bar{u}w.P} \qquad \textsf{T-nil}\ \frac{}{\vdash \mathsf{0}}$$

$$\textsf{T-par}\ \frac{\vdash P \quad \vdash Q}{\vdash P \mid Q} \qquad \textsf{T-sum}\ \frac{\vdash P \quad \vdash Q}{\vdash P + Q} \qquad \textsf{T-res}\ \frac{a : L \quad \vdash P}{\vdash \nu a P}$$

$$\textsf{T-rep}\ \frac{\vdash u : \sharp^n V \quad x : V \quad \vdash P \quad \forall v \in os(P), lv(v) < n}{\vdash !u(x).P}$$

Table 5.1: The core type system

or more of its elements by any finite number (including zero) of smaller elements. It can indeed be shown that if $>$ is well-founded then so is $>_{mul}$ [Bez03].

In this chapter we make no syntactic difference between channels and variables, both of them are names. We shall restrict our attention to the termination property of closed processes, i.e., processes without free names of `bool` or `Nat` types.

5.2 The Core System: the Simply Typed π-calculus with Levels

Our first type system for termination is obtained by making mild modifications to the types and typing rules of the simply typed π-calculus (cf. Section 2.2.5). We assign a level, which is a natural number, to each channel name and incorporate it into the type of the name. Now the syntax of channel type takes the new form:

L	$::=$	$\sharp^n V$	channel types
n	$::=$	$1, 2, \cdots$	levels

For convenience of presentation, in this chapter we only study type systems *à la Church*, and each name is assigned a type a priori. Hence we do not annotate bound names with types. We write $x : T$ to mean that the name x has type T. A judgment $\vdash P$ says that P is a well-typed process, and $\vdash w : V$ says that w is a well-typed value of type V. Our core type system is displayed in Table 5.1.

The main difference from the simply typed π-calculus lies in the rule T-rep, in which $os(P)$ is a set collecting all names in P which appear as subjects of those outputs that are not underneath any replicated input (we say this kind of outputs are *active*). Specifically, $os(P)$ is defined inductively as follows:

$$\begin{array}{rcl@{\qquad}rcl}
os(\mathsf{0}) & \stackrel{\text{def}}{=} & \emptyset & os(\bar{u}w.P) & \stackrel{\text{def}}{=} & \{u\} \cup os(P) \\
os(!u(x).P) & \stackrel{\text{def}}{=} & \emptyset & os(P \mid Q) & \stackrel{\text{def}}{=} & os(P) \cup os(Q) \\
os(u(x).P) & \stackrel{\text{def}}{=} & os(P) & os(P + Q) & \stackrel{\text{def}}{=} & os(P) \cup os(Q) \\
os(\nu a P) & \stackrel{\text{def}}{=} & os(P) & & &
\end{array}$$

The function $lv(v)$ calculates the *level* of channel v from its type. If $v : \sharp^n V$ then $lv(v) = n$.

The purpose of using levels is to rule out recursive inputs as, for instance, in the process

$$\bar{a} \,|!a.\bar{b} \,|!b.\bar{a} \tag{5.1}$$

where the two replicated processes can call each other thus producing a divergence. Our type system requires that in any replication $!a(x).P$, the level of a is greater than the level of any name that appears as subject of an active output of P. In other words, a process spawned by the resource $!a(x).P$ can only access other resources with a lower level. Process (5.1) is therefore illegal because $!a.\bar{b}$ requires $lv(a) > lv(b)$ while $!b.\bar{a}$ expects $lv(b) > lv(a)$. For the same reason, for the process $P \stackrel{\text{def}}{=} a(x).!x.\bar{c} \,|!c.\bar{b}$ to be well typed it is necessary that names received along channel a have a higher level than $lv(c)$. Therefore $P \mid \bar{a}b$ is illegal, since, due to the right component of P, we have $lv(c) > lv(b)$. As a final example, consider the process

$$\bar{a} \,|!a.(\bar{c} \,|!b.\bar{a}). \tag{5.2}$$

In this process, there is an output at a underneath the replication at a. The output at a, however, is not active in the body $\bar{c} \,|!b.\bar{a}$ of the replication because it is located underneath another replication. Therefore this process is typable by our type system. We call $\mathcal{T}$ this type system and write $\mathcal{T} \vdash P$ to mean that P is a well-typed process under $\mathcal{T}$. The subject reduction theorem of the simply typed π-calculus can be easily adapted to $\mathcal{T}$.

Before proceeding to prove the termination property of well-typed processes, we need some preliminary notations. If name a appears as the subject of some active output in a subterm of P and $lv(a) = i$, then we say a has at least one *output (subject) occurrence* at level i. It does not matter whether a is free or bound in the whole process P. For example, let

$$Q \stackrel{\text{def}}{=} (\nu d : \sharp^1\texttt{Nat})(a(x).b(y).(\bar{x}y \mid \bar{c}d.\bar{c}d.\bar{d}3)).$$

It is easy to see that Q is a well-typed process if the types of a, b and c are $\sharp^3\sharp^1\texttt{Nat}, \sharp^3\texttt{Nat}$ and $\sharp^2\sharp^1\texttt{Nat}$, respectively. In this process x and d have one output occurrence at level 1 respectively, c has two output occurrences at level 2, a and b have zero output occurrence at any level. Thus, the identity of names that appear in output occurrences is not important: what we need is the number of output occurrences of names belonging to the same level, and this for each level. For every well-typed process P, we use n_i to stand for the number of output occurrences at level i; hence n_i is simply calculated by scanning the process expression. Then the *weight*, $wt(P)$, of a process P is the vector $\langle n_k, n_{k-1}, \cdots, n_1\rangle$, with k representing the highest level on which the process has non-zero output occurrence. As to the process Q defined above, it has the weight $wt(Q) = \langle 2, 2\rangle$. Formally we have the following definition of $wt(P)$. It is related to the set $os(P)$ since we only count the levels of names appearing in $os(P)$.

$$\begin{array}{rcl@{\qquad}rcl}
wt(\mathsf{0}) & \stackrel{\text{def}}{=} & \mathbf{0} & wt(\bar{u}w.P) & \stackrel{\text{def}}{=} & wt(P) + \mathbf{0}_{lv(u)} \\
wt(!u(x).P) & \stackrel{\text{def}}{=} & \mathbf{0} & wt(P \mid Q) & \stackrel{\text{def}}{=} & wt(P) + wt(Q) \\
wt(u(x).P) & \stackrel{\text{def}}{=} & wt(P) & wt(P + Q) & \stackrel{\text{def}}{=} & max\{wt(P), wt(Q)\} \\
wt(\nu aP) & \stackrel{\text{def}}{=} & wt(P) & & &
\end{array}$$

The next lemma says that weight is a good measure because it decreases at each reduction step. This property leads naturally to the termination theorem of well-typed processes, by the well-foundedness of weight.

Lemma 5.1 *Suppose* $\mathcal{T} \vdash P, P \xrightarrow{\tau} P'$*, then* $wt(P') \prec wt(P)$.

Proof: By induction on transitions. See Appendix C.1. □

Theorem 5.2 *If* $\mathcal{T} \vdash P$*, then* P *terminates.*

Proof: By induction on the weight of well typed processes.

- Base case: All processes with weight $\mathbf{0}$ are terminating because they have no active output.
- Inductive step: Suppose all processes with weights less than $wt(P)$ are terminating. We show that P is also terminating. Consider the set $I = \{i \mid P \xrightarrow{\tau} P_i\}$. For each $i \in I$ we know that: (i) $\mathcal{T} \vdash P_i$ by the subjection reduction property of $\mathcal{T}$, (ii) $wt(P_i) \prec wt(P)$ by Lemma 5.1. So each such P_i is terminating by induction hypothesis, which ensures that P is terminating.

□

The type system $\mathcal{T}$ provides us with a concise way of handling nested inputs. For example, let $a : \sharp^1\sharp^1\mathtt{Nat}, b : \sharp^2\mathtt{Nat}, c : \sharp^1\mathtt{Nat}$, then process (1.1) is well-typed and therefore terminating. Similarly, process (5.2) is well-typed if the types of a, b and c are $\sharp^2\mathtt{Nat}, \sharp^3\mathtt{Nat}$ and $\sharp^1\mathtt{Nat}$, respectively.

Lemma 5.1 implies that the weight of a process gives us a bound on the time that the process takes to terminate. First we define the size of a process as the whole number of literals in the process expression.

Proposition 5.3 *Let* n *and* k *be the size and the highest level in a well-typed process* P*, respectively. Then* P *terminates in polynomial time* $\mathcal{O}(n^k)$.

Proof: Let $wt(P)$ be $\langle n_k, ..., n_1 \rangle$, thus $\sum_{i=1}^{k} n_i < n$. The worst case is that when an active output of level i is consumed, all (less than n) new active outputs appear at level $i-1$. Hence one output occurrence of level i gives rise to at most $f(i)$ steps of reduction, where

$$f(i) = \begin{cases} 1 & \text{if } i = 1 \\ 1 + n * f(i-1) & \text{if } i > 1. \end{cases}$$

In other words,

$$f(i) = \sum_{j=0}^{i-1} n^j = \frac{n^i - 1}{n - 1}.$$

Since the weight of P is $\langle n_k, ..., n_1 \rangle$, the length of any reduction sequence from P is bounded by $\sum_{i=1}^{k} n_i * f(i)$. As

$$\sum_{i=1}^{k} n_i * f(i) \leq \sum_{i=1}^{k} n_i * f(k) = (\sum_{i=1}^{k} n_i) * f(k) < n * f(k) = \frac{n(n^k - 1)}{n - 1}$$

we know that P terminates in time $\mathcal{O}(n^k)$. □

As a consequence of Proposition 5.3 we are not able to encode the simply typed λ-calculus into the π-calculus with type system $\mathcal{T}$, according to the known result that computing the normal form of a non-trivial λ-term cannot be finished in elementary time [Sta79, Loa98]. However, we shall see in the next section an extension of $\mathcal{T}$ that makes it possible to encode all primitive recursive functions (some of which are not representable in the simply typed λ-calculus).

5.3 Allowing Limited Forms of Recursive Inputs

The previous type system allows nesting of inputs but forbids all forms of recursive inputs (i.e. replications $!a(x).P$ with the body P having active outputs at channel a). In this and the following sections we study how to relax this restriction.

5.3.1 The Type System

Let us consider a simple example. Process P below has a recursive input: underneath the replication at a there are two outputs at a itself. However, the values emitted at a are "smaller" than the value received. This, and the fact that the "smaller than" relation on natural numbers is well-founded, ensures the termination of P. In other words, the termination of P is ensured by the relation among the subjects and objects of the prefixes – rather the subjects alone as it was in the previous system.

$$\begin{array}{rl} P \stackrel{\text{def}}{=} & \bar{a}\langle 10\rangle \mid !a(n).\ \texttt{if}\ n > 0\ \texttt{then}\ (\bar{a}\langle n-1\rangle \mid \bar{a}\langle n-1\rangle) \\ \stackrel{\tau}{\longrightarrow} & \bar{a}\langle 9\rangle \mid \bar{a}\langle 9\rangle \mid !a(n).\ \texttt{if}\ n > 0\ \texttt{then}\ (\bar{a}\langle n-1\rangle \mid \bar{a}\langle n-1\rangle) \end{array}$$

For simplicity, the only well-founded values that we consider are naturals. But the arguments below apply to any data type on whose values a well-founded relation can be defined.

We use function $out(P)$ to extract all active outputs in P. The definition is similar to that of $os(P)$ in Section 5.2. The main difference is that each element of $out(P)$ is a complete output prefix, including both subject and object names. For example, we have $out(!a(x).P) = \emptyset$ and $out(\bar{a}w.P) = \{\bar{a}w\} \cup out(P)$.

In the typing rule, in any replication $!a(x).P$ we compare the active outputs in P with the input $a(x)$ using the relation $\lhd$ below. We have that $\bar{b}w \lhd a(x)$ holds in two cases: (1) b has a lower level than a; (2) b and a have the same level, but the object w of b is provably smaller than the object x of a. For this, we assume a mechanism for evaluating (possibly open) integer expressions that allows us to derive assertions such as $x - i < x$ if $i > 0$. We adopt an eager reduction strategy, thereby the expression in an output is evaluated before the output fires.

Definition 5.4 *Let $u : \sharp^n S$ and $v : \sharp^m T$. We write $\bar{v}w \lhd u(x)$ if one of the two cases holds: (i) $m < n$; (ii) $m = n$, $S = T = \texttt{Nat}$ and $w < x$.*

By substituting the following rule for T-rep in Table 5.1, we get the extended type system $\mathcal{T}'$. The second condition in the definition of $\lhd$ allows us to include some recursive inputs and gives us the difference from $\mathcal{T}$.

$$\textsf{T-rep}\quad \frac{\vdash u : \sharp^n V \quad x : V \quad \vdash P \quad \forall \bar{v}w \in out(P'), \bar{v}w \lhd u(x)}{\vdash !u(x).P}$$

The termination property of $\mathcal{T}'$ can also be proved with a schema similar to the proof in last section. However, the details are more complex because we need to be clear about how the first-order values in which we are interested evolve with the reduction steps. So we use a measure which records, for each output prefix, the value of the object and the level information of the subject. More precisely, the measure is a *compound vector*, which consists of two parts: the *Nat-multiset* and the weight, corresponding to each aspect of information that we wish to record.

To a given process P and level i, with $0 < i \leq k$ and k is the highest level in P, we assign a unique Nat-multiset $\mathcal{M}_{P,i} = [n_1, \cdots, n_l]$, with $n_j \in \mathbb{N} \cup \{\infty\}$ for all $j \leq l$. (Here we consider ∞ as the upper bound of the infinite set $\mathbb{N}$.) Intuitively, this multiset is obtained as follows. For each active output $\bar{b}w$ in P with $lv(b) = i$, there are three possibilities. If w is a constant value ($w \in \mathbb{N}$), then w is recorded in $\mathcal{M}_{P,i}$. If w contains variables of type `Nat`, then a ∞ is recorded in $\mathcal{M}_{P,i}$. Otherwise, w is not of type `Nat` and thus contributes nothing to the Nat-multiset. For instance, suppose $a : \sharp^3\texttt{Nat}, b : \sharp^2\texttt{Nat}, c : \sharp^1\texttt{Nat}$ and $P \stackrel{\text{def}}{=} \bar{a}\langle 1\rangle \mid \bar{a}\langle 1\rangle \mid \bar{b}\langle 2\rangle \mid !a(n).\bar{b}\langle n+1\rangle \mid b(n).\bar{c}\langle n\rangle$, then $\mathcal{T}' \vdash P$ and there are three Nat-multisets: $\mathcal{M}_{P,3} = [1,1]$, $\mathcal{M}_{P,2} = [2]$ and $\mathcal{M}_{P,1} = [\infty]$. Formally, we define $\mathcal{M}_{P,i}$ as follows:

$$
\begin{array}{rcl@{\qquad}rcl}
\mathcal{M}_{\mathtt{0},i} & \stackrel{\text{def}}{=} & [\,] & \mathcal{M}_{\nu aP,i} & \stackrel{\text{def}}{=} & \mathcal{M}_{P,i} \\
\mathcal{M}_{!a(x).P,i} & \stackrel{\text{def}}{=} & [\,] & \mathcal{M}_{P|Q,i} & \stackrel{\text{def}}{=} & \mathcal{M}_{P,i} \uplus \mathcal{M}_{Q,i} \\
\mathcal{M}_{a(x).P,i} & \stackrel{\text{def}}{=} & \mathcal{M}_{P,i} & \mathcal{M}_{P+Q,i} & \stackrel{\text{def}}{=} & \mathcal{M}_{P,i} \uplus \mathcal{M}_{Q,i} \\
\mathcal{M}_{\bar{a}w.P,i} & \stackrel{\text{def}}{=} & \left\{ \begin{array}{ll} \mathcal{M}_{P,i} \uplus [w] & \text{if } a : \sharp^i\texttt{Nat} \text{ and } w \in \mathbb{N} \\ \mathcal{M}_{P,i} \uplus [\infty] & \text{if } a : \sharp^i\texttt{Nat} \text{ and } \mathit{fvn}(w) \neq \emptyset \\ \mathcal{M}_{P,i} & \text{otherwise} \end{array} \right. & & &
\end{array}
$$

where $\mathit{fvn}(w)$ is the set of variables of type `Nat`. We define an operator $\searrow$ to combine a set of Nat-multisets $\{\mathcal{M}_{Q,i} \mid 0 < i \leq k\}$ with the weight of Q (as defined in the previous section), $wt(Q) = \langle n_k, \cdots, n_1\rangle$, so as to get a *compound vector* $t_Q = \langle(\mathcal{M}_{Q,k}; n_k), \cdots, (\mathcal{M}_{Q,1}; n_1)\rangle$. For the above example $wt(P) = \langle 2,1,1\rangle$, so $t_P = \{\mathcal{M}_{P,i} \mid 0 < i \leq k\} \searrow wt(P) = \langle([1,1];2), ([2];1), ([\infty];1)\rangle$. The order $\prec$ and the operator $+$ can be extended to compound vectors.

Definition 5.5 *Suppose $t_P = \langle(s_k), \cdots, (s_1)\rangle$ and $t_Q = \langle(s'_k), \cdots, (s'_1)\rangle$, where $s_i = \mathcal{M}_{P,i}; n_i$ and $s'_i = \mathcal{M}_{Q,i}; n'_i$ for $0 < i \leq k$.*

1. *$s_i \prec s'_i$ if $\mathcal{M}_{P,i} <_{mul} \mathcal{M}_{Q,i} \vee (\mathcal{M}_{P,i} = \mathcal{M}_{Q,i} \wedge n_i < n'_i)$*
2. *$s_i = s'_i$ if $\mathcal{M}_{P,i} = \mathcal{M}'_{Q,i} \wedge n_i = n'_i$*
3. *$s_i + s'_i = \mathcal{M}_{P,i} \uplus \mathcal{M}'_{Q,i}; n_i + n'_i$*
4. *$t_P \prec t_Q$ if $\exists i \leq k, s_j = s'_j$ for $j > i$ and $s_i \prec s'_i$*
5. *$t_P = t_Q$ if $s_i = s'_i$ for all $i \leq k$*
6. *$t_P + t_Q = \langle(s_k + s'_k), \cdots, (s_1 + s'_1)\rangle$*

Using compound vectors as the measure, we can build, with similar proof schemas, the counterparts of Lemma 5.1 and Theorem 5.2.

Lemma 5.6 *If $\mathcal{T}' \vdash P$ and $P \xrightarrow{\tau} P'$ then $t_{P'} \prec t_P$.*

Proof: See Appendix C.2. □

Theorem 5.7 *If $\mathcal{T}' \vdash P$ then P terminates.*

Proof: Followed from Lemma 5.6. □

Note that the measure used here is much more powerful than that in Section 5.2. With weights, we can only prove the termination of processes which always terminate in polynomial time. By using compound vectors, however, as we shall see in Section 5.3.2, we are able to capture the termination property of some processes which terminate in time $\mathcal{O}(f(n))$, where $f(n)$ a is primitive recursive function. For example, we can write a process to encode the *repeated exponentiation*, where $E(0) = 1$, $E(n+1) = 2^{E(n)}$. Once received a number n, the process does internal computation in time $\mathcal{O}(E(n))$ before sending out its result.

5.3.2 Example: Primitive Recursive Functions

For simplicity of presentation, we have concentrated mainly on monadic communication. However, it is easy to extend our calculus and type systems to allow polyadic communications and an if-then-else construct [1] (see Appendix C.3), which are needed in this example. The advantage of $\mathcal{T}'$ over $\mathcal{T}$ lies in the fact that primitive recursive functions can now be captured.

Definition 5.8 (Primitive recursive functions)[Bec80] *The class of primitive recursive functions consists of those functions that can be obtained by repeated application of composition and primitive recursion starting with (1) the successor function, $f(x) = x+1$, (2) the zero function, $f(x) = 0$, (3) the generalized identity functions $f_i^{(n)}(x_1, \cdots, x_n) = x_i$, with the generating rules for composition and primitive recursion being*

1. *Composition* $\quad f(x_1, \cdots, x_n) = g(e_1(x_1, \cdots, x_n), \cdots, e_m(x_1, \cdots, x_n))$
2. *Primitive recursion*

$$\begin{cases} f(0, x_2, \cdots, x_n) = e(x_2, \cdots, x_n) \\ f(x_1 + 1, x_2, \cdots, x_n) = g(x_1, f(x_1, \cdots, x_n), x_2, \cdots, x_n) \end{cases}$$

Proposition 5.9 *All primitive recursive functions can be represented as terminating processes in the π-calculus.*

Proof: A function $f(\widetilde{x})$ can be represented as a process F_a which has replicated input like $!a(\widetilde{x}, y).R$, where name a is called port of F, with type $T_{m,n} = \sharp^m(\widetilde{\mathtt{Nat}}, \sharp^n\mathtt{Nat})$ where $m > n$. After receiving via a some arguments $\widetilde{x}$ and a return channel y, process R does some computation, and finally the result is delivered at y. For the three basic functions, the results are returned immediately. This

[1] For convenience of presention, in the rest of the thesis we shall use an if-then-else construct in place of the nondeterministic choice construct, instead of considering the two constructs simultaneously.

style of encoding follows from Milner's encoding of λ-terms into π-processes [Mil92]. In the similar way can the correctness of the following five encodings be verified.

The encoding of the three basic functions is straightforward.

(1) The zero function $F_a \stackrel{\text{def}}{=} !a(x,y).\bar{y}\langle 0\rangle$.

(2) The successor function $F_a \stackrel{\text{def}}{=} !a(x,y).\bar{y}\langle x+1\rangle$

(3) The identity functions $F_a^{(i,n)} \stackrel{\text{def}}{=} !a(\widetilde{x},y).\bar{y}\langle x_i\rangle$.

By assigning to a the type $T_{2,1}$, the three processes defined above are typable in our core type system $\mathcal{T}$, thus typable in $\mathcal{T}'$.

(4) Composition

Suppose that E_{ia_i} is defined for e_i with the type of a_i being T_{m_i,n_i} for all $1 \leq i \leq m$, and G_c is defined for g with the type of c being $T_{m',n'}$. By induction hypothesis, they are well typed in $\mathcal{T}'$. Then we can define F_a for f as:

$$\begin{array}{rcl} F_a & \stackrel{\text{def}}{=} & !a(\widetilde{x},y).(\nu\widetilde{a}\widetilde{b}c)(E_{1a_1} \mid \bar{a}_1\langle\widetilde{x},b_1\rangle \mid \cdots \mid E_{ma_m} \mid \bar{a}_m\langle\widetilde{x},b_m\rangle \\ & & \qquad \mid b_1(z_1).\cdots.b_m(z_m).\bar{c}\langle\widetilde{z},y\rangle \mid G_c) \end{array}$$

Let $m'' = max\{m_1,\cdots,m_m,m'\}+1$ and give name a the type $T_{m'',n'}$. It can be easily checked that process F_a is typable in $\mathcal{T}'$.

(5) Primitive recursion

Suppose that E_b is defined for e with the type of b being T_{m_1,n_1}, and $G_{a'}$ is defined for g with the type of a' being T_{m_2,n_2}. By induction hypothesis they are well typed in $\mathcal{T}'$. We define F_a as follows.

$$\begin{array}{rcl} F_a & \stackrel{\text{def}}{=} & !a(\widetilde{x},y).\ \texttt{if}\ x_1 = 0\ \texttt{then}\ (\nu b)(E_b \mid \bar{b}\langle x_2,\cdots,x_n,y\rangle) \\ & & \qquad \texttt{else}\ (\nu b')(\bar{a}\langle x_1-1,x_2,\cdots,x_n,b'\rangle \\ & & \qquad\qquad \mid b'(z).(\nu a')(G_{a'} \mid \bar{a'}\langle x_1-1,z,x_2,\cdots,x_n,y\rangle)) \end{array}$$

Let $m = max\{m_1,m_2\}+1$ and give type T_{m,n_2} to a. It is easy to see that F_a is well typed in $\mathcal{T}'$. □

For the process F in (1.2), which represents the factorial function, it is typable if we give name a the type $\sharp^2(\texttt{Nat},\sharp^1\texttt{Nat})$. By contrast, the encoding of functions that are not primitive recursive may not be typable. An example is Ackermann's function.

5.4 Asynchronous Names

In this section we start a new direction for extending our core type system of Section 5.2: we prove termination by exploiting the structure of processes instead of the well-foundedness of first-order values. The goal of the new type systems (in this and in the next section) is to gain more flexibility in handling nested inputs. In the previous type systems, we required that in a replicated process $!a(x).P$, the highest level should be given to a. This condition appears rigid when we meet a process like $!a.b.\bar{a}$ because we do not take advantage of the level of b. This is the motivation for relaxing the requirement. The basic idea is to take into account the sum of the levels of two input subjects

a, b, and compare it with the level of the output subject a. However, this incurs another problem. Observe the following reduction:

$$\begin{array}{rcl} P & \stackrel{\text{def}}{=} & \bar{a} \mid \bar{b} \mid !a.b.\bar{a} \\ & \stackrel{\tau}{\longrightarrow} & \bar{b} \mid b.\bar{a} \mid !a.b.\bar{a} \\ & \stackrel{\tau}{\longrightarrow} & \bar{a} \mid !a.b.\bar{a} \end{array}$$

The weight of P does not decrease after the first step of reduction (we consume a copy of $\bar{a}$ but liberate another one). Only after the second reduction does the weight decrease. Further, P might run in parallel with another process, say Q, that interferes with P and prevents the second reduction from happening. This example illustrates two new problems that we have to consider: the weight of a process may not decrease at every step; because of interferences and interleaving among the activities of concurrent processes, consecutive reductions may not yield "atomic blocks" after which the weight decreases.

In the new type system we allow the measure of a process to decrease after a finite number of steps, rather than at every step, and up to some commutativities of reductions and process manipulations. This difference has a strong consequence in the proofs. For technical reasons related to the proofs, we require certain names to be asynchronous.

5.4.1 Proving Termination with Asynchronous Names

A name a is *asynchronous* if all outputs with subject a are followed by $\mathsf{0}$. That is, if $\bar{a}v.P$ appears in a process then $P = \mathsf{0}$. A convenient way of distinguishing between synchronous and asynchronous names is using Milner's sorts (cf. Section 2.2.3). Thus we assume two sorts of names, ι_a and ι_s, for asynchronous and synchronous names respectively, with the requirement that all names in ι_a are syntactically used as asynchronous names. We assume that all processes are well-sorted in this sense and will not include the requirements related to sorts in our type systems. (We stick to using both asynchronous and synchronous names instead of working on asynchronous π-calculus, because synchronous π-calculus is sometimes useful – see for instance the example in Section 5.5.2 – and it is more expressive [Pal03]. However, all the results in this paper are valid for asynchronous π-calculus as well.)

We make another syntactic modification to the calculus (with an if-then-else construct in place of the nondeterministic choice in Table 2.4) by adding a construct to represent a sequence of inputs underneath a replication:

$$\begin{array}{rcll} \kappa & ::= & u_1(x_1).\cdots.u_n(x_n) & n \geq 1 \text{ and } \forall i < n, u_i : \iota_a \\ P & ::= & \ldots \mid !\kappa.P & \end{array}$$

This addition is not necessary – it only simplifies the presentation. It is partly justified by the usefulness of input sequences in applications. (It also strongly reminds us of the input pattern construct of the Join-calculus [Fou98]). We call κ an input pattern. Note that all but the last input subject in κ are required to be asynchronous. As far as termination is concerned, we believe that the constraint – and therefore the distinction between asynchronous and synchronous names – can be lifted. However, we do not know how to prove Theorem 5.10 without it.

The usual form of replication $!u(x).P$ is now considered as a special case where the input pattern has length 1, i.e., it is composed of just one input prefix. We extend the definition of weight to input patterns by taking account of the levels of input subjects: $wt(u_1(x_1).\cdots.u_n(x_n)) = \mathbf{0}_{k_1} + \cdots + \mathbf{0}_{k_n}$ where $lv(u_i) = k_i$. The typing rule $\mathsf{T\text{-}rep}$ in Table 5.1 is replaced by the following one.

$$\mathsf{T\text{-}rep}\ \frac{\vdash \kappa.P \quad wt(\kappa) \succ wt(P)}{\vdash !\kappa.P}$$

Intuitively, this rule means that we consume more than what we produce. That is, to produce a new process P, we have to consume all the prefixes from $u_1(x_1)$ to $u_n(x_n)$ on the left of P, which leads to the consumption of corresponding outputs at $u_1, \cdots, u_n$. Since the sum of weights of all the outputs is larger than the weight of P, the whole process has a tendency to decrease its weight. Although the idea behind this type system ($\mathcal{T}''$) is simple, the proof of termination is non-trivial because we need to find out whether and when a whole input pattern is consumed and thus the measure decreases. The rest of the subsection is devoted to proving the following theorem.

Theorem 5.10 *If $\mathcal{T}'' \vdash P$ then P terminates.*

Below we briefly explain the structure of the proof and proceed in four steps. Firstly, we decorate processes and transition rules with tags, which indicate the origin of each reduction: whether it is caused by calling a replicated input, a non-replicated input or it comes from an if-then-else structure. This information helps us to locate some points, called *landmarks*, in a reduction path. If a process performs a sequence of reductions that are locally ordered (that is, all and only the input prefixes of a given input pattern are consumed), then the process goes from a landmark to the next one and decreases its weight (Lemma 5.12). (This is not sufficient to guarantee termination, since in general the reductions of several input patterns may interleave and some input patterns may be consumed only partially.) Secondly, by taking advantage of the constraint about asynchronous names, we show a limited form of commutativity of reductions (Lemma 5.13). Thirdly, by commuting consecutive reductions, we adjust a reduction path and establish on it some locally ordered sequences separated by landmarks. Moreover, when an input pattern is not completely consumed, we perform some manipulations on the derivatives of processes and erase some inert subprocesses. Combining all of these with the result of Step 1, we are able to prove the termination property of tagged processes Lemma (5.14). Finally, the termination of untagged processes follows from the operational correspondence between tagged and untagged processes (Lemma 5.11), which concludes our proof of Theorem 5.10.

We begin with introducing the concepts of *atomic tag*, *tag* and *tagged process*. Atomic tags are names from a separate infinite set $\mathcal{N}'$, which is disjoint from the set $\mathcal{N}$ used for constructing untagged processes. We use the function $\rho : \mathcal{N}' \mapsto \mathbb{N}$ to associate every atomic tag with a natural number. Note that we require $\mathcal{N}'$ to be an infinite set so that it can always supply fresh atomic tags as we need. We let $l, l', l_1, \cdots$ range over atomic tags and ϵ stand for a special atomic tag by setting $\rho(\epsilon) = 0$. A tag is a pair (l, n) where l is an atomic tag and n is an integer with $n \leq \rho(l)$. We let $t, t', \cdots$ range over tags and write ϵ as the abbreviation of the special tag $(\epsilon, 0)$. The only difference between tagged processes and untagged ones is that the former gives tags for all non-replicated inputs.

$$P \quad ::= \quad \cdots \mid u^t(x).P$$

$$\textsf{if-t}\ \frac{}{\texttt{if}\ true\ \texttt{then}\ P\ \texttt{else}\ Q \xrightarrow{\epsilon'} P} \qquad \textsf{if-f}\ \frac{}{\texttt{if}\ false\ \texttt{then}\ P\ \texttt{else}\ Q \xrightarrow{\epsilon'} Q}$$

$$\textsf{com1}\ \frac{P \xrightarrow{(\nu\widetilde{a})\bar{u}w} P' \quad Q \xrightarrow{u^t w} Q' \quad \widetilde{a} \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{t} (\nu\widetilde{a})(P' \mid Q')} \qquad \textsf{in}\ \frac{}{u^t(x).P \xrightarrow{u^t w} P\{w/x\}}$$

$$\textsf{rep}\ \frac{\kappa = u_1(x_1).\cdots.u_n(x_n) \quad l \text{ fresh} \quad \rho(l) = n}{!\kappa.P \xrightarrow{u_1^{(l,1)} w} !\kappa.P \mid (u_2^{(l,2)}(x_2).\cdots.u_n^{(l,n)}(x_n).P)\{w/x_1\}}$$

Table 5.2: Transition rules for tagged processes

Note that we do not give tags to input patterns. A tagged process P is *regular* if the only tag that appears in P is the special tag ϵ. On the contrary, if there is a tag t with $t \neq \epsilon$ in P, then P is irregular. We reserve the tag ϵ' for the transition rules if-t and if-f (see Table 5.2). Unlike ϵ, ϵ' only appears in transitions, not in tagged processes. We define the operator $erase(\cdot)$ to erase all tags in a tagged process so as to get an untagged process. Let P be a tagged process. We define $wt(P)$ as $wt(erase(P))$, and we write $\mathcal{T}'' \vdash P$ if $\mathcal{T}'' \vdash erase(P)$. The transition rules for tagged processes are the same as in Table 2.3 except for rules in, com1, rep, if-t and if-f, which are displayed in Table 5.2. In the rule rep, a fresh atomic tag l is introduced to witness the invocation of the replicated input $!\kappa.P$. The result of invoking $!\kappa.P$ is the generation of a new process $(u_2^{(l,2)}(x_2).\cdots.u_n^{(l,n)}(x_n).P)\{w/x_1\}$. The condition $\rho(l) = n$ relates l to κ by requiring the number of input prefixes in κ to be $\rho(l)$. So if an input prefix has tag $(l, \rho(l))$ then it originates from the last input prefix in κ.

Note that substitutions of names do not affect tags. More precisely, we have that $(a^t(x).P)\{c/b\} = (a\{c/b\})^t(x).P\{c/b\}$. From the transition rules it can be seen that tags are never used as values to be transmitted between processes and that there is no substitution for tags.

Tags give us information about the transitions of tagged processes. For example, if P is regular and $P \xrightarrow{t} P'$, then at least we know the following information:

- if $t = \epsilon'$ then an if-then-else structure in P disappears when P evolves into P';
- if $t = \epsilon$ then the reduction results from an internal communication between an active output and a non-replicated input;
- if $t = (l, 1)$ then the reduction results from an internal communication between an active output and a replicated input of the form $!u_1(x_1).\cdots.u_{\rho(l)}(x_{\rho(l)}).Q$; moreover, if $\rho(l) > 1$ then P' has a subprocess $u_2^{(l,2)}(x_2).\cdots.u_{\rho(l)}^{(l,\rho(l))}(x_{\rho(l)}).Q$.

We define the operator $(\cdot)^\circ$, which is complementary to $erase(\cdot)$, to translate untagged processes into regular processes by giving all non-replicated inputs the special tag ϵ.

$$\begin{array}{rcl@{\qquad}rcl}
0^\circ & \stackrel{\text{def}}{=} & 0 & (u(x).P)^\circ & \stackrel{\text{def}}{=} & u^\epsilon(x).P^\circ \\
(\bar{u}w.P)^\circ & \stackrel{\text{def}}{=} & \bar{u}w.P^\circ & (\nu aP)^\circ & \stackrel{\text{def}}{=} & \nu aP^\circ \\
(P \mid Q)^\circ & \stackrel{\text{def}}{=} & P^\circ \mid Q^\circ & (!\kappa.P)^\circ & \stackrel{\text{def}}{=} & !\kappa.P^\circ \\
\multicolumn{6}{c}{(\ \texttt{if}\ w\ \texttt{then}\ P\ \texttt{else}\ Q\ )^\circ \stackrel{\text{def}}{=} \texttt{if}\ w\ \texttt{then}\ P^\circ\ \texttt{else}\ Q^\circ}
\end{array}$$

Note that $erase(P^\circ) = P$ holds but $(erase(P))^\circ = P$ may not be valid. For example $!a.b.\bar{c} \mid \bar{a} \xrightarrow{(l,1)} !a.b.\bar{c} \mid b^{(l,2)}.\bar{c} \equiv P'$, and thus $(erase(P'))^\circ = !a.b.\bar{c} \mid b^{\epsilon}.\bar{c} \neq P'$. However, there exists operational correspondence between tagged and untagged processes since tags do not have semantic meaning and the purpose of using tags is to identify every newly created process from some replicated process. This is precisely what the next lemma shows.

Lemma 5.11 *Let P be a tagged process and Q an untagged one.*

1. *If $P \xrightarrow{t} P'$ then $erase(P) \xrightarrow{\tau} erase(P')$.*
2. *If $Q \xrightarrow{\tau} Q'$ and $erase(P) = Q$, then $P \xrightarrow{t} P'$ and $erase(P') = Q'$ for some t.*

As expressed in Lemma 5.12 and 5.13, (well-typed) tagged processes have some interesting properties such as decrement of weight after some specific steps of reduction and commutativity of reductions.

Lemma 5.12
1. *If $P \xrightarrow{\epsilon} P'$ then $wt(P) \succ wt(P')$.*
2. *If $P \xrightarrow{\epsilon'} P'$ then $wt(P) \succeq wt(P')$*
3. *If $P \xrightarrow{(l,1)} P_1 \xrightarrow{(l,2)} \cdots P_{n-1} \xrightarrow{(l,n)} P'$ and $n = \rho(l) > 0$ then $wt(P) \succ wt(P')$.*

Proof: See Appendix C.4. □

Generally speaking, commutativity of reductions does not hold in the π-calculus. For instance, the process $P = a.b \mid \bar{a} \mid \bar{b}$ has reduction path $P \xrightarrow{\tau_a}\xrightarrow{\tau_b}$ but not $\xrightarrow{\tau_b}\xrightarrow{\tau_a}$, where $\xrightarrow{\tau_c}$ means that an internal communication happens on channel c. As we shall see in the next two lemmas, this property does hold in the presence of certain constraints. We write $P \stackrel{\widetilde{t}}{\Longrightarrow} R$ for $P \xrightarrow{t_1} \cdots \xrightarrow{t_n} R$, where $\widetilde{t} = t_1 \cdots t_n$.

Lemma 5.13
1. *If P is regular and $P \stackrel{\widetilde{t}}{\Longrightarrow} R \xrightarrow{(l,i)} R_1 \xrightarrow{t} R'$, $t \in \{\epsilon, \epsilon'\}$ and $i < \rho(l)$, then there exists R_1' such that $R \xrightarrow{t} R_1' \xrightarrow{(l,i)} R'$.*
2. *If P is regular and $P \stackrel{\widetilde{t}}{\Longrightarrow} R \xrightarrow{(l',j)} R_1 \xrightarrow{(l,i)} R'$, $l \neq l'$, $j < \rho(l')$ and $i \leq \rho(l)$, then there exists some R_1' such that $R \xrightarrow{(l,i)} R_1' \xrightarrow{(l',j)} R'$.*

Proof: See Appendix C.4. □

In the following lemma, we make full use of commutativity and reorganize a reduction path in a way easy of pinpointing landmarks, which witness the decrement of the measure that we choose for the beginning process of the path.

Lemma 5.14 *All the regular tagged processes terminate.*

Proof: We sketch the idea of the proof; more details are given in Appendix C.4.

Let P be a regular tagged process. We show that P terminates by induction on its weight $wt(P)$.
• Base case: All processes with weight $\mathbf{0}$ must be terminating because they have no active outputs.
• Inductive step: Suppose P is non-terminating and thus has an infinite reduction sequence

$$P \equiv P_0 \xrightarrow{t_1} P_1 \xrightarrow{t_2} \cdots \xrightarrow{t_i} P_i \xrightarrow{t_{i+1}} \cdots$$

Now the tag t_1 takes one of the three forms: ϵ', ϵ or (l, i). By doing case analysis we can prove that in every case there always exists some Q such that: (i) Q is also non-terminating; (ii) Q is regular; (iii) $wt(P) \succ wt(Q)$. When Q is found, we get a contradiction since by induction hypothesis all processes with weights less than $wt(P)$ are terminating. So the supposition is false and P should be terminating.

In seeking for this Q, we carefully manipulate the reduction path of P by commuting reductions (Lemmas 5.13) in order to put all tags belonging to the same input pattern in contiguous positions. Then we can use Lemma 5.12 to prove (iii). If an input pattern cannot be completed, which means that its continuation does not contribute to the subsequent reductions of P, we can substitute $\mathsf{0}$ for the continuation. For example, suppose $P \stackrel{\text{def}}{=} \nu a_2(\bar{a}_1 \mid !a_1.a_2.R_1) \mid R_2$ and there is a reduction sequence like:

$$P \xrightarrow{(l,1)} P_1 \xrightarrow{t_2} P_2 \xrightarrow{t_3} \cdots$$

with $P_1 \equiv \nu a_2(a_2^{(l,2)}.R_1 \mid !a_1.a_2.R_1) \mid R_2$. Since $a_2^{(l,2)}.R_1$ is never consumed in the reduction sequence, it contributes nothing to the subsequent reductions starting from P_1. So we can safely take Q to be $\nu a_2(\mathsf{0} \mid !a_1.a_2.R_1) \mid R_2$, and the same transition sequence can still be made, with $\mathsf{0}$ in place of the top level $a_2^{(l,2)}.R_1$ in all derivatives.

Consequently, for each new atomic tag l with $\rho(l) > 0$ created by the derivatives of P, either we have found the complete input pattern corresponding to l, or the input pattern cannot be completed but no l appears in the infinite reduction path starting from Q. As a result, no new tag appears in Q, i.e. (ii) is satisfied. □

Now we are ready to prove Theorem 5.10 by applying the last lemma.

Proof of Theorem 5.10:

By Lemma 5.11 it is easy to prove the following claim:

> Let P be a untagged process and Q be a tagged process such that $erase(Q) = P$, then P is non-terminating iff Q is non-terminating.

Since $erase(P^\circ) = P$, it follows that P° is non-terminating iff P is non-terminating. By the definition of the translation $(\cdot)^\circ$ we know that P° is regular. Therefore Lemma 5.14 applies and P° must be terminating, which in turn implies the termination of P. □

Proposition 5.15 *For a process P well-typed under $\mathcal{T}''$, let n and k be its size and the highest level, respectively. Then P terminates in polynomial time $\mathcal{O}(n^{k+1})$.*

Proof: Let $wt(P)$ be $\langle n_k, ..., n_1 \rangle$. From the proof Lemma 5.14 we know that: (i) commutation of reductions does not change the length of a reduction sequence; (ii) the measure diminishes from one landmark to the next one; (iii) the distance between two neighboring landmarks is less than n. In addition, by similar arguments as in the proof of Proposition 5.3 it can be shown that in each locally ordered reduction path there are at most $\frac{n(n^k-1)}{n-1}$ landmarks. Therefore the whole length of each reduction path is bounded by $\frac{n^2(n^k-1)}{n-1}$. □

$$
\begin{array}{l}
[\Sigma_{i=1}^{n}\bar{a}_i d_i.P_i] \stackrel{\text{def}}{=} \nu s\ (\ \bar{s}\langle true\rangle \\
\qquad\qquad\qquad\qquad \mid\ \Pi_{i=1}^{n}\nu e\ \bar{a}_i\langle d_i, s, e\rangle.e(x).\ \texttt{if}\ x\ \texttt{then}\ [P_i]\ \texttt{else}\ \texttt{0}) \\
[\Sigma_{i=1}^{m} b_i(z).Q_i] \stackrel{\text{def}}{=} \\
\nu t\ (\ \bar{t}\langle true\rangle \\
\quad \mid\ \Pi_{i=1}^{m}\nu g\ (\ \bar{g} \\
\qquad\qquad \mid\ !g.b_i(z,s,e).t(x).\ \texttt{if}\ x\ \texttt{then} \\
\qquad\qquad\qquad\qquad\qquad (\ s(y).\ \texttt{if}\ y\ \texttt{then} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\ \bar{t}\langle false\rangle \mid\ \bar{s}\langle false\rangle \mid\ \bar{e}\langle true\rangle \mid\ [Q_i]) \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \texttt{else} \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\ \bar{t}\langle true\rangle \mid\ \bar{s}\langle false\rangle \mid\ \bar{e}\langle false\rangle \mid\ \bar{g})) \\
\qquad\qquad\qquad\qquad\qquad\ \texttt{else} \\
\qquad\qquad\qquad\qquad\qquad\ \bar{t}\langle false\rangle \mid \bar{b_i}\langle z,s,e\rangle))
\end{array}
$$

where t, s and e are fresh and $\Pi_{i=1}^{n}P_i$ means $P_1 \mid \cdots \mid P_n$.

Table 5.3: The protocol of encoding separate choice

5.4.2 Example: the Protocol of Encoding Separate Choice

Consider the following protocol which is used for encoding separate choice (the summands of the choice are either all inputs or all outputs) by parallel composition [Nes00], [SW01, Section 5.5.4]. One of the main contributions in [Nes00] is the proof that the protocol does not introduce divergence. Here we prove it using typability.

The protocol uses two locks s and t. When one input branch meets a matching output branch, it receives a datum together with lock s and acknowledge channel e. Then the receiver tests t and s sequentially. If t signals failure, because another input branch has been chosen, the receiver is obliged to resend the value just received. Otherwise, it continues to test s. When s also signals success, the receiver enables the acknowledge channel and let the sender proceed. At the same time, both t and s are set to $false$ to prevent other branches from proceeding. If the test of s is negative, because the current output branch has committed to another input branch, the receiver should restart from the beginning and try to catch other send-requests. This backtracking is implemented by recursively triggering a new copy of the input branch.

Usually when a protocol employs a mechanism of backtracking, it has a high probability to give rise to divergence. The protocol in this example is an exception. However, to figure out this fact is non-trivial, one needs to do careful reasoning so as to analyze the possible reduction paths in all different cases. With the aid of type system $\mathcal{T}''$, we reduce the task to a routine type-checking problem. We show that the protocol does not add any infinite loop by proving that the typability of $[P_i]$ and $[Q_i]$ implies that of $[\Sigma_i \bar{a}_i d_i.P_i]$ and $[\Sigma_i b_i(z).Q_i]$. Then we conclude by Theorem 5.10. Here we take the i-th branch of input guarded choice as an example and assume that b_i does not appear in Q_i. Suppose that $[Q_i]$ is typable by $\mathcal{T}''$ and the highest level of names in Q_i is n with $n > 1$. Let us give type $\sharp^1\texttt{bool}$ to t, type $\sharp^{n+1}\texttt{Nat}$ to g and type $\sharp^2(T_z, \sharp^1\texttt{bool}, \sharp^1\texttt{bool})$ to b_i where T_z is the type

of the datum z. Take $g.b_i(z,s,e)$ as the input pattern, noted as κ, and abbreviate its continuation as P. Then $!\kappa.P$ is well typed under $\mathcal{T}''$ because $wt(\kappa) = \langle 1,\cdots,1,0\rangle$ and $wt(P) = \langle 1,\cdots,0,3\rangle$ (the dots stand for a 0-sequence of length $(n-2)$), thus $wt(\kappa) \succ wt(P)$.

5.5 Partial Orders

The purpose of our final type system is to type processes even if they contain replications whose input and output parts have the same weight. Of course not all such processes can be accepted. For instance, $!a.b.(\bar{a} \mid \bar{b})$ should not be accepted, since it does not terminate when running together with $\bar{a} \mid \bar{b}$. However, we might want to accept

$$!g(a,b).a.(\bar{g}\langle a,b\rangle \mid \bar{b}) \tag{5.3}$$

where a and b have the same type. Processes like (5.3) are useful. For instance they often appear in systems composed of several "similar" processes (an example is the chain of cells in Section 5.5.2). In (5.3) the input pattern $g(a,b).a$ and the continuation $\bar{g}\langle a,b\rangle \mid \bar{b}$ have the same weight, which makes rule T-rep of $\mathcal{T}''$ inapplicable. In the new system, termination is proved by incorporating partial orders into certain channel types. For instance, (5.3) will be accepted if the partial order extracted from the type of g shows that b is below a (both b and a being names that are received along g).

5.5.1 The Type System

We present the new type system $\mathcal{T}'''$. The general structure of the associated termination proof goes along the same line as the proof in Section 5.4.1. But now we need a measure which combines lexicographic and multiset orderings.

To begin with, we introduce some preliminary notations. Let $\mathcal{A}$ be a set and $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$ be a partial order on elements of $\mathcal{A}$. The set of names appearing in elements of $\mathcal{R}$ is $n(\mathcal{R}) = \{a \mid a\mathcal{R}b \vee b\mathcal{R}a \text{ for some } b\}$. Let $\widetilde{x}$ be a tuple of names $x_1,\cdots,x_n$, we write the length n of the tuple as $\mid \widetilde{x} \mid$. In the following, we define some operators for partial orders. They will be used for simplifying the presentation of our typing rules in Table 5.4.

Definition 5.16 *Let $\mathcal{R} \subseteq \mathcal{N} \times \mathcal{N}$ and $\mathcal{S} \subseteq \mathtt{Nat} \times \mathtt{Nat}$ be two partial orders and $\widetilde{x}$ is a tuple of names in $\mathcal{N}$. We define two operators $/$ and $*$ to transform one partial order into the other.*

1. $\mathcal{R}/\widetilde{x} \stackrel{\text{def}}{=} \begin{cases} \emptyset & \text{if } n(\mathcal{R}) \cap \widetilde{x} = \emptyset \\ \{(i,j) \mid x_i \mathcal{R} x_j\} & \text{if } n(\mathcal{R}) \subseteq \widetilde{x} \\ \text{undefined} & \text{otherwise} \end{cases}$

2. $\mathcal{S} * \widetilde{x} \stackrel{\text{def}}{=} \{(x_i,x_j) \mid i\mathcal{S}j\}$ if $max\{n(\mathcal{S})\} \leq \mid \widetilde{x} \mid$

As shown by the following lemma, the two operators are complementary to each other to some extent.

Lemma 5.17 1. $(\mathcal{R}/\widetilde{x}) * \widetilde{x} = \mathcal{R}$ *if* $n(\mathcal{R}) \subseteq \widetilde{x}$

2. $(\mathcal{S} * \widetilde{x})/\widetilde{x} = \mathcal{S}$ *if* $max\{n(\mathcal{S})\} \leq |\ \widetilde{x}\ |$

Proof: By the definition of / and $*$ directly. □

Remark: In this paper we use partial order in a very narrow sense. We require each partial order on names to satisfy the following two conditions: (i) mathematically it is a strict partial order (irreflexive, antisymmetric and transitive); (ii) all names in $n(\mathcal{R})$ are of the same type (this type is written $T_{\mathcal{R}}$).

Let $\mathcal{R}$ be a partial order. We extract the sub-partial order defined on $n(\mathcal{R}) \setminus \widetilde{x}$ by $\mathcal{R}\Downarrow_{\widetilde{x}} = \{(a,b) \mid a, b \notin \widetilde{x}$ and $a\mathcal{R}c_1\mathcal{R}\cdots\mathcal{R}c_nb$ for some $\widetilde{c} \subseteq \widetilde{x}$ and $n \geq 0\}$. Given two partial orders $\mathcal{R}_1$, $\mathcal{R}_2$ with $T_{\mathcal{R}_1} = T_{\mathcal{R}_2}$, we let $\mathcal{R}_1 + \mathcal{R}_2$ be $\mathcal{R}_1 \cup \mathcal{R}_2$ if such a union is a partial order. Otherwise, it is undefined.

The operator $os(\cdot)$ of Section 5.2 is now refined to be $mos_{\mathcal{R}}(\cdot)$, which defines a multiset recording all subject occurrences of names in active outputs and with type $T_{\mathcal{R}}$.

$$
\begin{array}{rcl}
mos_{\mathcal{R}}(\mathsf{0}) & \stackrel{\text{def}}{=} & [\,] \\
mos_{\mathcal{R}}(!u(\widetilde{x}).P) & \stackrel{\text{def}}{=} & [\,] \\
mos_{\mathcal{R}}(u(\widetilde{x}).P) & \stackrel{\text{def}}{=} & mos_{\mathcal{R}}(P) \\
mos_{\mathcal{R}}(\nu aP) & \stackrel{\text{def}}{=} & mos_{\mathcal{R}}(P) \\
mos_{\mathcal{R}}(\bar{u}\widetilde{w}.P) & \stackrel{\text{def}}{=} & \begin{cases} [u] \uplus mos_{\mathcal{R}}(P) & \text{if } u : T_{\mathcal{R}} \\ mos_{\mathcal{R}}(P) & \text{otherwise} \end{cases} \\
mos_{\mathcal{R}}(P \mid Q) & \stackrel{\text{def}}{=} & mos_{\mathcal{R}}(P) \uplus mos_{\mathcal{R}}(Q) \\
mos_{\mathcal{R}}(\texttt{ if } b \texttt{ then } P \texttt{ else } Q) & \stackrel{\text{def}}{=} & mos_{\mathcal{R}}(P) \uplus mos_{\mathcal{R}}(Q)
\end{array}
$$

The operator $mos_{\mathcal{R}}(\cdot)$ can be extended to input patterns by defining: $mos_{\mathcal{R}}(\kappa) \stackrel{\text{def}}{=} mos_{\mathcal{R}}(\bar{u}_1\widetilde{x}_1 \mid \cdots \mid \bar{u}_n\widetilde{x}_n)$ if $\kappa = u_1(\widetilde{x}_1).\cdots.u_n(\widetilde{x}_n)$.

Let $\mathcal{R}$ be a partial order and $\mathcal{R}_{mul}$ be the induced multiset ordering on multisets over $n(\mathcal{R})$. The binary relation defined below will act as the second component of our measure, which is a lexicographic ordering with weight of processes as its first component.

Definition 5.18 *Let $\mathcal{R}$ be a partial order on names, Q be a process, P be either an input pattern or a process. It holds that $P\ \widehat{\mathcal{R}}\ Q$ if the following three conditions are satisfied, for some multisets on names $\mathcal{M}_1, \mathcal{M}_2$ and $\mathcal{M}$: (i) $mos_{\mathcal{R}}(P) = \mathcal{M} \uplus \mathcal{M}_1$; (ii) $mos_{\mathcal{R}}(Q) = \mathcal{M} \uplus \mathcal{M}_2$; (iii) $\mathcal{M}_1\ \mathcal{R}_{mul}\ \mathcal{M}_2$.*

Essentially the relation $\widehat{\mathcal{R}}$ is an extension of the multiset ordering $\mathcal{R}_{mul}$. One can easily prove that $\widehat{\mathcal{R}}$ is also well-founded: if $\mathcal{R}$ is finite, then there exists no infinite sequence like $P_0\ \widehat{\mathcal{R}}\ P_1\ \widehat{\mathcal{R}}\ P_2\ \widehat{\mathcal{R}}\cdots$

Now we are well-prepared to present our types and type system. Here we consider polyadic π-calculus and redefine channel type as follows.

$$L ::= \sharp^n_{\mathcal{S}}\widetilde{V} \quad \text{where} \quad \forall i, j \in n(\mathcal{S}),\ \ V_i = V_j$$

where $\mathcal{S} \subseteq \texttt{Nat} \times \texttt{Nat}$ is a partial order on the indexes of $\widetilde{V}$ (that is, if $|\ \widetilde{V}\ | = m$ then $\mathcal{S}$ is a partial order on the set $\{1, ..., m\}$). The condition in the definition says that if i and j are two indexes related by $\mathcal{S}$, then the i-th and j-th components of $\widetilde{V}$ have the same type.

If νaP is a subprocess of Q, we say that the restriction νa is *unguarded* if νaP is not underneath any input or output prefix. More precisely, we define a set $ur(P)$ to collect all unguarded restrictions

in P.

$$\begin{array}{rcl@{\qquad}rcl} ur(\mathsf{0}) & \stackrel{\text{def}}{=} & \emptyset & ur(u(\widetilde{x}).P) & \stackrel{\text{def}}{=} & \emptyset \\ ur(!u(\widetilde{x}).P) & \stackrel{\text{def}}{=} & \emptyset & ur(\bar{u}\widetilde{w}.P) & \stackrel{\text{def}}{=} & \emptyset \\ ur(\nu aP) & \stackrel{\text{def}}{=} & \{a\} \cup ur(P) & ur(P \mid Q) & \stackrel{\text{def}}{=} & ur(P) \cup ur(Q) \end{array}$$

$$ur(\texttt{ if } b \texttt{ then } P \texttt{ else } Q) \stackrel{\text{def}}{=} ur(P) \cup ur(Q)$$

If we pull all unguarded restrictions of Q to the outmost positions, the resulting process $\nu\widetilde{a}Q'$ has the same behavior as Q. In literature this property is often characterized by a sequence of structural rules describing scope extension, see for example [Par01]. Since we assume that bound names are different from free names, the side conditions of those rules are met automatically. We use this property implicitly and often write Q as $\nu\widetilde{a}Q'$ without unguarded restrictions in Q'.

Besides the two sorts ι_a and ι_s introduced in the beginning of Section 5.4.1, now we need another sort ι_r. It requires that

$$\text{if } \alpha.P \text{ is a process with } subj(\alpha) : \iota_r \text{ then } ur(P) = \emptyset.$$

In other words, if a name of sort ι_r appears in the subject position of a prefix (either input or output), then the continuation process has no unguarded restrictions. This technical condition facilitates the presentation of Definition 5.19.

Suppose $\kappa = a_1(\widetilde{x}_1).\cdots.a_n(\widetilde{x}_n)$ and each a_i has type $\sharp_{\mathcal{S}_i}^{m_i}\widetilde{V}$. We extract a partial order from κ by defining $\mathcal{R}_\kappa = \mathcal{S}_1 * \widetilde{x}_1 \cup \cdots \cup \mathcal{S}_n * \widetilde{x}_n$. It is well defined because all the bound names are assumed to be different from each other. For example, if $\kappa = a_1(x_{11}, x_{12}, x_{13}).a_2(x_{21}, x_{22}, x_{23})$, $\mathcal{S}_1 = \{(1,2)\}$ and $\mathcal{S}_2 = \{(2,1)\}$, then we have $\mathcal{R}_\kappa = \{(x_{11}, x_{12}), (x_{22}, x_{21})\}$.

Definition 5.19 *Let $\kappa = u_1(\widetilde{x}_1).\cdots.u_n(\widetilde{x}_n)$. The relation $\kappa \succ\!\!\!\succ P$ holds if one of the following two cases holds: (i) $wt(\kappa) \succ wt(P)$; (ii) $wt(\kappa) = wt(P)$, $\kappa\ \widehat{\mathcal{R}_\kappa}\ P$ and $u_n : \iota_r$.*

The second condition indicates the improvement of $\mathcal{T}'''$ over $\mathcal{T}''$. We allow the input pattern to have the same weight as that of the continuation, as long as there is some partial order to reflect a tendency of decrement.

The typing rules of $\mathcal{T}'''$ are presented in Table 5.4. Now the judgment $\mathcal{R} \vdash P$ means that P is a well-typed process and the free names in P respect the (possibly empty) partial order $\mathcal{R}$. In the premise of rule T-in, if there exists some non-empty partial order relation on $\widetilde{x}$, then it is exactly captured by $\mathcal{R}$, the partial order built upon free names of P. In rule T-out for $\mathcal{R} + \mathcal{S} * \widetilde{v}$ to be well defined, the partial order on $\widetilde{v}$ should not conflict with the partial order exhibited by P. Similarly in rules T-par and T-if the partial orders contributed by P and Q should be compatible and thus can be combined together. As we only consider the partial order on free names of νaP, in rule T-res all pairs concerning a are deleted from $\mathcal{R}$ while the relative partial order relation on other names are kept intact. In rule T-rep the appearance of the replication operator does not affect the existing partial order, but it requires the validity of the condition $\kappa \succ\!\!\!\succ P$, which plays an important role in Lemma 5.21 and gives us the possibility of doing termination proof.

In Definition 5.19 the constraint imposed on u_n is used to prohibit potential extension of partial orders caused by the restriction operator. Let us consider two examples, concerning two different occurrences of restricted names.

$$\textsf{T-in}\ \frac{u : \sharp^n_{\mathcal{S}} \widetilde{V} \quad \widetilde{x} : \widetilde{V} \quad \mathcal{R} \vdash P \quad \mathcal{S} = \mathcal{R}/\widetilde{x}}{\mathcal{R} \Downarrow_{\widetilde{x}} \vdash u(\widetilde{x}).P} \qquad \textsf{T-nil}\ \frac{}{\emptyset \vdash \mathsf{0}}$$

$$\textsf{T-out}\ \frac{u : \sharp^n_{\mathcal{S}} \widetilde{V} \quad \widetilde{w} : \widetilde{V} \quad \mathcal{R} \vdash P}{\mathcal{R} + \mathcal{S} * \widetilde{w} \vdash \bar{u}\widetilde{w}.P} \qquad \textsf{T-par}\ \frac{\mathcal{R}_1 \vdash P \quad \mathcal{R}_2 \vdash Q}{\mathcal{R}_1 + \mathcal{R}_2 \vdash P \mid Q}$$

$$\textsf{T-if}\ \frac{b : \mathtt{bool} \quad \mathcal{R}_1 \vdash P \quad \mathcal{R}_2 \vdash Q}{\mathcal{R}_1 + \mathcal{R}_2 \vdash \ \mathtt{if}\ b\ \mathtt{then}\ P\ \mathtt{else}\ Q} \qquad \textsf{T-res}\ \frac{a : L \quad \mathcal{R} \vdash P}{\mathcal{R} \Downarrow_a \vdash \nu a P}$$

$$\textsf{T-rep}\ \frac{\mathcal{R} \vdash \kappa.P \quad \kappa \ \succ P}{\mathcal{R} \vdash !\kappa.P}$$

Table 5.4: Typing rules of $\mathcal{T}'''$

(i) Underneath an input pattern

$$\begin{array}{rcl} P & \stackrel{\text{def}}{=} & !g(a,b).a.\nu c(\bar{g}\langle b,c\rangle \mid \bar{b}) \mid \bar{g}\langle a,b\rangle \mid \bar{a} \mid \bar{g}\langle a,b\rangle \\ & \stackrel{\tau}{\longrightarrow} & !g(a,b).a.\nu c(\bar{g}\langle b,c\rangle \mid \bar{b}) \mid a.\nu c(\bar{g}\langle b,c\rangle \mid \bar{b}) \mid \bar{a} \mid \bar{g}\langle a,b\rangle \\ & \stackrel{\tau}{\longrightarrow} & !g(a,b).a.\nu c(\bar{g}\langle b,c\rangle \mid \bar{b}) \mid \nu c(\bar{g}\langle b,c\rangle \mid \bar{b}) \mid \bar{g}\langle a,b\rangle \\ & \equiv & \nu d(!g(a,b).a.\nu c(\bar{g}\langle b,c\rangle \mid \bar{b}) \mid \bar{g}\langle b,d\rangle \mid \bar{b} \mid \bar{g}\langle a,b\rangle) \\ & \stackrel{\text{def}}{=} & \nu d P' \end{array}$$

(ii) Outside an input pattern

$$\begin{array}{rcl} Q & \stackrel{\text{def}}{=} & !g(a,b).a.(\bar{g}\langle a,b\rangle \mid \bar{b}) \mid \bar{g}\langle a,b\rangle \mid \bar{a}.\nu c\bar{g}\langle b,c\rangle \\ & \stackrel{\tau}{\longrightarrow} & !g(a,b).a.(\bar{g}\langle a,b\rangle \mid \bar{b}) \mid a.(\bar{g}\langle a,b\rangle \mid \bar{b}) \mid \bar{a}.\nu c\bar{g}\langle b,c\rangle \\ & \stackrel{\tau}{\longrightarrow} & !g(a,b).a.(\bar{g}\langle a,b\rangle \mid \bar{b}) \mid \bar{g}\langle a,b\rangle \mid \bar{b} \mid \nu c\bar{g}\langle b,c\rangle \\ & \equiv & \nu d(!g(a,b).a.(\bar{g}\langle a,b\rangle \mid \bar{b}) \mid \bar{g}\langle a,b\rangle \mid \bar{b} \mid \bar{g}\langle b,d\rangle) \\ & \stackrel{\text{def}}{=} & \nu d Q' \end{array}$$

Let the type of name g be $\sharp^2_{\{(1,2)\}}(\sharp^1_\emptyset V, \sharp^1_\emptyset V)$. Assume $\mathcal{R} = \{(a,b)\}$ and $\mathcal{R}' = \{(a,b),(b,d)\}$. If the condition $a_n : \iota_r$ in Definition 5.19 was lifted, then both P and Q would be well typed: in the first example, it could be derived that $\mathcal{R} \vdash P$ and $\mathcal{R}' \vdash P'$; in the second example, $\mathcal{R} \vdash Q$ and $\mathcal{R}' \vdash Q'$. In both cases the new name d extends the partial order $\mathcal{R}$ to be $\mathcal{R}'$.

However, the process P does not terminate because it can make cyclic reduction and the two steps from P to $\nu P'$ form a cycle. Therefore the structure in (i) is dangerous and should be disallowed. The process Q always terminates in at most 6 steps, but ruling out the structure in (ii) simplifies our proof of Lemma 5.22.

For this type system, we have the following subject reduction property.

Theorem 5.20 (Subject reduction) *Suppose $\mathcal{R} \vdash P$ and $P \stackrel{\alpha}{\longrightarrow} P'$.*

1. *If $\alpha = \tau$ due to a communication then $\mathcal{R} \vdash P'$.*
2. *If $\alpha = \tau$ due to a conditional then $\mathcal{R}' \vdash P'$ with $\mathcal{R} = \mathcal{R}' + \mathcal{R}''$ for some $\mathcal{R}'$ and $\mathcal{R}''$.*

3. *If* $\alpha = a\widetilde{w}$ *then there exists* $n, \mathcal{S}$ *and* $\widetilde{V}$ *such that*
   (a) $a : \sharp^n_{\mathcal{S}}\widetilde{V}$ *and* $\widetilde{w} : \widetilde{V}$
   (b) *if* $\mathcal{S} * \widetilde{w}$ *is a partial order then* $\mathcal{R} + \mathcal{S} * \widetilde{w} \vdash P'$.
4. *If* $\alpha = (\nu\widetilde{b})\bar{a}\widetilde{w}$ *then there exists* $n, \mathcal{S}, \mathcal{R}'$ *and* $\widetilde{V}$ *such that*
   (a) $a : \sharp^n_{\mathcal{S}}\widetilde{V}$ *and* $\widetilde{w} : \widetilde{V}$
   (b) $\mathcal{R}' \vdash P'$
   (c) $\mathcal{R} = (\mathcal{R}' + \mathcal{S} * \widetilde{w})\Downarrow_{\widetilde{b}}$

Proof: See Appendix C.5. Most efforts are made to check the consistency of partial orders in the type environments. □

The following lemma is the counterpart of Lemma 5.12.

Lemma 5.21 *Suppose that* $ur(P) = \emptyset$, $\mathcal{R} \vdash P$, $P \xrightarrow{(l,1)} P_1 \xrightarrow{(l,2)} \cdots P_{n-1} \xrightarrow{(l,n)} P'$ *and* $n = \rho(l) > 0$. *Then one of the following two cases holds.*

1. $wt(P) \succ wt(P')$;
2. $P \;\widehat{\mathcal{R}}\; P'$ *and* $ur(P') = \emptyset$.

Proof: See Appendix C.5. □

With the last lemma we are able to prove Lemma 5.22, whose role in $\mathcal{T}'''$ is the same as that of Lemma 5.14 in $\mathcal{T}''$.

Lemma 5.22 *All the regular tagged processes (well-typed under* $\mathcal{T}'''$*) terminate.*

Proof: Compared with the proof of Lemma 5.14, the main difference is that when we have completed some input patterns and get a reduction sequence like

$$P_0 \stackrel{\widetilde{t_1}}{\Longrightarrow} P_1 \xrightarrow{\epsilon} P_2 \stackrel{\widetilde{t_2}}{\Longrightarrow} \cdots \xrightarrow{\epsilon'} P_{i-1} \stackrel{\widetilde{t_i}}{\Longrightarrow} P_i \cdots$$

it may be possible that $\forall j < i, wt(P_j) = wt(P_{j+1})$. Let $\mathcal{R} \vdash P_0$, we can show by contradiction that the sequence of processes of equal weight is finite, by the well-foundedness of $\mathcal{R}_{mul}$. See Appendix C.5 for more details. □

Finally we have the following termination theorem for $\mathcal{T}'''$, due to the operational correspondence between tagged and untagged process and Lemma 5.22.

Theorem 5.23 *If* $\mathcal{R} \vdash P$ *then* P *terminates. Moreover, let* n *and* k *be its size and the highest level, then* P *terminates in time* $\mathcal{O}(n^{k+3})$.

Proof: The proof of termination is straightforward. Let us look at the time complexity. Clearly the sizes of the two sets $n(\mathcal{R})$ and $mos_{\mathcal{R}}(P)$ are less than n. If there is a sequence $P_0 \;\widehat{\mathcal{R}}\; P_1 \;\widehat{\mathcal{R}}\; \cdots \;\widehat{\mathcal{R}}\; P_m$, then it can be shown that $m < n^2$. By similar arguments as in the proof of Proposition 5.15 it can be shown that in each locally ordered reduction path there are at most $\frac{n(n^k-1)}{n-1}$ landmarks and the distance between two neighbouring landmarks is less than n^3. Therefore the whole length of each reduction path is bounded by $\frac{n^4(n^k-1)}{n-1}$. □

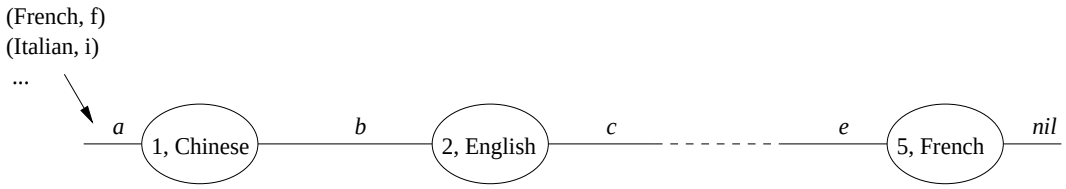

Figure 5.1: An example of symbol table

$$
\begin{array}{rcl}
G & \stackrel{\text{def}}{=} & !g(a,b,n,s).a(t,x). \\
 & & \quad \texttt{if } t = s \texttt{ then} \\
 & & \quad\quad \bar{x}\langle n\rangle.\bar{g}\langle a,b,n,s\rangle \\
 & & \quad \texttt{else if } b = nil \texttt{ then} \\
 & & \quad\quad\quad \bar{x}\langle n+1\rangle.\nu c(\bar{g}\langle c,nil,n+1,t\rangle \mid \bar{g}\langle a,c,n,s\rangle) \\
 & & \quad\quad \texttt{else } \bar{b}\langle t,x\rangle.\bar{g}\langle a,b,n,s\rangle \\
ST_0 & \stackrel{\text{def}}{=} & \nu g(G \mid \bar{g}\langle a,nil,1,s_0\rangle) \\
ST_m & \stackrel{\text{def}}{=} & ST_0 \mid \bar{a}\langle t_1,x_1\rangle \mid \cdots \mid \bar{a}\langle t_m,x_m\rangle
\end{array}
$$

Table 5.5: The implementation of a symbol table

5.5.2 Example: Symbol Table

This example comes from [Jon93, San99]. It is about the implementation of a symbol table as a chain of cells. In Table 5.5 G is a generator for cells; ST_0 is the initial state of the symbol table with only one cell; ST_m is the system in which the symbol table has m pending requests.

Every cell of the chain stores a pair (n, s), where s is a string and n is a key identifying the position of the cell in the chain. A cell is equipped with two channels so as to be connected to its left and right neighbours. The first cell has a public left channel a to communicate with the environment and the last cell has a right channel nil to mark the end of the chain. Once received a query for string t, the table lets the request ripple down the chain until either t is found in a cell, or the end of the chain is reached, which means that t is a new string and thus a new cell is created to store t. In both cases, the key associated to t is returned as a result. See Figure 5.1 for a concrete example, where three cells and two requests are shown; the first cell stores the string "Chinese" and its key "1", while the first request queries the string "French" and an answer will be delivered at channel f. There is parallelism in the system: many requests can be rippling down the chain at the same time.

As to termination, the example is interesting for at least two reasons. (1) The chain exhibits a syntactically challenging form. The replicated process G has a sophisticated structure of recursive inputs: the input pattern has inputs at g and a, while the continuation has a few outputs at g and one output at b, which has the same type as a. (2) Semantically, the chain is a dynamic structure, which can grow to finite but unbounded length, depending on the number of requests it serves.

Moreover, the chain has a high parallelism involving independent threads of activities. The number of steps that the symbol table takes to serve a request depends on the length of the chain, on the number of internal threads in the chain, and on the value of the request.

Suppose $T \stackrel{\text{def}}{=} \sharp^2_\emptyset(\mathsf{String}, \sharp^1\mathtt{Nat})$, $\mathcal{S} \stackrel{\text{def}}{=} \{(1,2)\}$ and let the type of g be $\sharp^1_{\mathcal{S}}(T, T, \mathtt{Nat}, \mathsf{String})$, where String is the type for strings. We consider *nil* as a constant name of the language studied in this section and take it for the bottom element of any partial order $\mathcal{R} \subseteq \mathcal{N} \times \mathcal{N}$ with $T_{\mathcal{R}} = T$. For any $m \in \mathbb{N}$, process ST_m is well typed under $\mathcal{T}'''$ and thus terminating.

5.6 Summary

In this chapter we have proposed a core type system and three extensions of it to ensure termination of processes in the π-calculus. Based on the type systems we are able to prove the termination property of some challenging applications: the encodings of primitive recursive functions, the protocol for encoding separate choice in terms of parallel composition, a symbol table implemented as a dynamic chain of cells. For all (but one of) the type systems we also present upper bounds to the number of steps well-typed processes take to terminate.

We believe that the idea of using levels can be applied to other name-passing calculi. For instance, in Appendix C.6, we have checked that in the Join-calculus [Fou98] the type system presented in Section 5.4 can be simplified. Intuitively, this is because the Join-calculus can be encoded into a sublanguage of the asynchronous π-calculus with each input channel being unique, thus our assumption about asynchronous names in Section 5.4 is automatically met and recursive inputs are easier to be handled.

In Section 1.5 we have already discussed related work on termination, notably [San05] and [YBH04]. Our systems are incomparable with those in [San05] and [YBH04]. Roughly, in [San05] and [YBH04] processes are mainly "functional" and indeed include the standard encodings of the λ-calculus into the π-calculus. These processes are not typable in our type systems. In this chapter the processes are mainly "imperative". For instance, the examples in sections 5.4.2 and 5.5.2 are not typable in [San05] and [YBH04]. One way of interpreting the results of this chapter is to consider combinatory approach (on which our termination proofs are based) as a complementary technique to logical relations (on which [San05] and [YBH04] are based) for showing termination of processes. It would be interesting to see whether the two approaches can be successfully combined.

Chapter 6

Conclusions and Future Work

In this thesis we have investigated various issues on probabilistic processes and typed mobile processes. The major contributions are, briefly, the following:

1. A complete axiomatisation of a calculus which contains both nondeterministic and probabilistic choice, and recursion. We have axiomatized both strong and weak behavioural equivalences. It is the first time, as far as we know, that a complete axiomatisation of weak behavioural equivalences is presented for a language of this kind.

2. A complete axiomatisation of typed bisimilarity in the π-calculus with capability types. An indirect axiomatisation of a variant typed bisimilarity given in [HR04]. To our knowledge, this is the first attempt towards an algebraic theory of typed mobile processes.

3. A core type system and three refinements of it for guaranteeing termination property of well-typed processes in the π-calculus. In the termination proofs we have exploited two term rewriting techniques: lexicographic and multiset orderings. In contrast, the conventional proof techniques for concurrency, such as coinduction and structural induction, do not play an important role here.

In summary, we have developed algebraic techniques for reasoning about the behaviour of probabilistic processes and typed mobile processes. We have also studied a type-based technique for verifying the termination property of mobile processes. These results lay out the foundations for further study of more advanced models which may combine probability with typed mobility. They also highlight the robustness of the algebraic and typed-based techniques for behavioural reasoning.

In the rest of this chapter we discuss possible future work, including several problems that have been left open.

Generalisation of the results

Due to the difficulty discussed at the end of Section 4.4.1 we are only able to give an indirect axiomatisation of the bisimilarity proposed by Hennessy and Rathke [HR04]. We are not clear whether it is possible to directly axiomatize the equivalence in the language considered in Chapter 4.

We do not know at present how to adapt our results to the language in [BS98] either. We recall that the main differences are: (i) no distinction between channels and variables, (ii) no matching construct, (iii) the use of Pierce and Sangiorgi's types. Because of (i), some care is needed in a proof system, for instance in defining the appropriate rules for manipulating names that will later be bound in an input. Because of (ii), the expansion law cannot be used without appropriate modification. Another issue is axiomatisations of typed *weak* bisimilarities. In this case, however, types may not be so central, and the addition of the usual tau laws [Mil89a] might be sufficient.

For Hennessy and Rathke's bisimilarity, as well as the typed bisimilarity defined in [SW01], there are results that relate them to contextual equivalences such as barbed equivalence. It would be interesting to see what kind of contextual equivalence (if any) corresponds to our typed bisimilarity (Definition 4.9).

Our type system in Chapter 4 allows matching names to have arbitrary types. It is not clear how to restrict our use of matching. Limiting matching to names of compatible types might pose a problem for subject reduction. On the other hand, allowing matching only on names with types of the form $\mathtt{b}T$, as in [PS96], would seem difficult, for matching plays an important role in axiomatisations. For example, one would not be able to rewrite $x \mid \bar{y}$ as $x.\bar{y}+\bar{y}.x+[x=y]\tau$ under the type environment $\Delta = x : \mathtt{i}T, y : \mathtt{o}T$. In [HR04], a particular typing rule for matching is presented, which allows meet of types on successful matches. It might be interesting to know whether the presence of this typing rule would affect the validity of our proof systems.

Type inference

In Chapter 5 for the sake of simplicity we have given our type systems in the Church version. It is not difficult to transform them into the Curry version. For the Curry version of $\mathcal{T}$ and $\mathcal{T}'$, it is possible to check automatically whether a program is well-typed by using type inference, following for instance Vasconcelos and Honda's type inference algorithm for polyadic π-calculus [VH93]. Here one needs an extra constraint, which is a partial order between levels of names. By inspecting the structure of a process, this task can be done in linear time w.r.t. the size of the process. For $\mathcal{T}''$ and $\mathcal{T}'''$, however, type inference is not straightforward. In the future we would like to investigate efficient type inference algorithms for them.

Parallel composition

Parallel composition plays an important role for modelling distributed concurrent systems, as it allows to specify the structural properties of systems composed of several interacting parts. However, having both recursion and parallel composition in a process calculus complicates the matters to establish a complete axiomatisation, mostly because this can give rise to infinite-state systems even with the guardedness condition. For example, let E be the expression $\mu_X(a.(X \mid b))$, then we can easily see that there is an infinite transition graph starting from E, though it is guarded in the sense of Definition 3.2. Milner points out in [Mil89b] that in order to have a complete axiomatisation for CCS with both recursion and parallel composition, a sufficient condition is that the parallel composition does not occur in the body of any recursive expression.

In [DPP05] we relax this restriction by requiring, instead, that free variables do not appear in the scope of parallel composition[1]. In addition, due to the difficulty of defining parallel composition on probabilistic automata as discussed in [Seg95], we have refined the probabilistic process calculus given in Chapter 3. We restrict ourselves to simple probabilistic automata in [DPP05], and we have given complete axiomatisations for strong bisimilarity and observational equivalence. To obtain the completeness of the axiomatisations, we have developed a probabilistic version of the expansion law to eliminate all occurrences of parallel composition. In order to do that, we heavily rely on the condition that only closed terms are put in parallel. We are now considering how to adapt these results to axiomatize probabilistic branching bisimilarity.

Metric semantics of probabilistic processes

Usually probabilistic bisimulation is adapted from the classical notion of bisimulation by treating probabilities as labels (see for example [LS91, Seg95, PLS00, DP05]), but this does not provide a robust relation, since quantities are matched only when they are identical. Processes that differ for a very small probability, for instance, would be considered just as different as processes that perform completely different actions. This is particularly relevant to security systems where specifications can be given as perfect, but impractical processes and other, practical processes are considered safe if they only differ from the specification with a negligible probability.

To find a more flexible way to differentiate processes, researchers in this area have borrowed from pure mathematics the notion of metric [DJGP02, DJGP04, vBW04, vBW01]. A metric is defined as a function that associates a distance with a pair of processes. In [DCPP05] we have defined a notion of metric called *state-metric*. It turns out that in a probabilistic transition system each state-metric corresponds to a probabilistic bisimulation and that the greatest state-metric corresponds to probabilistic bisimilarity. Furthermore, the greatest state-metric can be characterised as the greatest fixed point of a monotonous function on state-metrics, which is closely analogous to Milner's characterisation of bisimilarity as the greatest fixed point of a monotonous function on bisimulations [Mil89a]. We would like to investigate whether it is possible to apply state-metrics to some fully-fledged probabilistic process calculus.

Implementation of the π-calculus

We consider it an interesting problem to develop a fully distributed implementation of the (synchronous) π-calculus (π) [Mil99] using a probabilistic asynchronous π-calculus (π_{pa}) [HP04] as an intermediate language. The reason of requiring a probabilistic calculus is that it has been shown impossible to implement certain mechanisms of the π-calculus without using randomization [Pal03]. Some results in this research direction are obtained in [PH04], but the part on implementation is very preliminary. A more realistic and efficient implementation remains to be worked out.

We believe it important that an implementation does not introduce livelocks (or other kinds of unintended outcomes), hence the translation from π to π_{pa} should preserve livelock-freedom, and

[1] A similar restriction is adopted, independently, in [BB05] for axiomatizing observational equivalence in a generic nonprobabilistic process algebra.

the semantics should be sensitive to divergency. For this reason, a probabilistic testing semantics is introduced in [PH04]. However, it turns out that probabilistic testing semantics is rather difficult to use. The correctness proofs are ad-hoc, by hand, and rather complicated. For the realistic (and necessarily more sophisticated) implementation, we need feasible and (at least in part) automatic proof methods. So it is appealing to investigate a divergency-sensitive *bisimulation-like* semantics. In the future, we plan to extend our results on divergency-sensitive equivalence obtained in Chapter 3 to the probabilistic asynchronous π-calculus.

Specification and verification of modern distributed systems

Unlike other probabilistic process algebras, π_{pa} has the advantage of being able to describe mobile systems. To equip π_{pa} with capability types might make it a good candidate language for specifying randomized, distributed, and mobile computational systems. Thus, as a natural development of our work, it is interesting to build an algebraic theory for this language by combining our results on probabilistic and mobile processes. A possible way to proceed is to first extend the results on finite processes in Chapter 3 to the setting of π_{pa}, then take type information into account as we have done in Chapter 4. As far as finite processes are concerned, this does not seem to be a difficult task. By contrast, we do not know how to extend our results in Chapter 5 so that probabilistic termination can be ensured by typability. We are not aware of any work on this problem.

Once an algebraic theory for typed π_{pa} is built, one might be able to exploit it to develop some automated verification tools, which would pave the way for verifying some useful randomized distributed algorithms and protocols. Therefore, another possible research direction is to develop automated tools that can check probabilistic and/or typed bisimulations, for which the results on axiomatisations in this thesis would be useful.

Appendix A

Proofs from Chapter 3

A.1 Proof of Lemma 3.14

We begin with several derived rules.

Lemma A.1 *The following rules are derivable:*

$$\textsf{wea2'}\ \frac{E \stackrel{\epsilon}{\Rightarrow}_c \{(\ell_i, E_i : p_i)\}_i \uplus \{(\ell, F : p)\} \quad F \stackrel{\epsilon}{\Rightarrow}_c \{(\tau, F_j : q_j)\}_j}{E \stackrel{\epsilon}{\Rightarrow}_c \{(\ell_i, E_i : p_i)\}_i \uplus \{(\ell, F_j : pq_j)\}_j}$$

$$\textsf{wea3'}\ \frac{E \stackrel{\epsilon}{\Rightarrow}_c \{(\ell_i, E_i : p_i)\}_i \uplus \{(\tau, F : p)\} \quad F \stackrel{\epsilon}{\Rightarrow}_c \{(h_j, F_j : q_j)\}_j}{E \stackrel{\epsilon}{\Rightarrow}_c \{(\ell_i, E_i : p_i)\}_i \uplus \{(h_j, F_j : pq_j)\}_j}$$

$$\textsf{wea4'}\ \frac{E \stackrel{\epsilon}{\Rightarrow}_c \{(\tau, E_i : p_i)\}_i \quad \forall i, E_i \stackrel{\epsilon}{\Rightarrow}_c \vartheta(X)}{E \stackrel{\epsilon}{\Rightarrow}_c \vartheta(X)}$$

Proof: By induction on inference. We also need to prove some other derived rules at first. For example, Before inferring wea2' we need to show its simpler version:

$$\textsf{wea2''}\ \frac{E \Rightarrow \{(\ell_i, E_i : p_i)\}_i \uplus \{(\ell, F : p)\} \quad F \stackrel{\epsilon}{\Rightarrow}_c \{(\tau, F_j : q_j)\}_j}{E \Rightarrow_c \{(\ell_i, E_i : p_i)\}_i \uplus \{(\ell, F_j : pq_j)\}_j}$$

The whole proof is tedious and non-instructive so it is omitted here. □

Lemma A.2 *Let $\mathcal{R}$ be a weak probabilistic bisimulation. If $E \mathcal{R} F$ then whenever $E \Rightarrow \eta$, there exists η' such that $F \stackrel{\epsilon}{\Rightarrow}_c \eta'$ and $\eta \equiv_{\mathcal{R}} \eta'$.*

Proof: By transition induction, on the depth of the inference by which the transition $E \Rightarrow \eta$ is inferred. We argue by cases on the last rule used.

- wea1: This is the induction basis. The result follows from the definition of weak probabilistic bisimulation.
- wea2: Let $\eta = \{(\ell_i, E_i : p_i)\}_{i \in I} \uplus \{(\ell, E_j : pq_j)\}_{j \in J}$, $\eta_1 = \{(\ell_i, E_i : p_i)\}_{i \in I} \uplus \{(\ell, E' : p)\}$, $\eta_2 = \{(\tau, E_j : q_j)\}_{j \in J}$, $E \Rightarrow \eta_1$ and $E' \Rightarrow \eta_2$. By induction hypothesis, there exists η_1' such

that $F \stackrel{\epsilon}{\Rightarrow}_c \eta_1'$ and $\eta_1 \equiv_{\mathcal{R}} \eta_1'$. Let $\eta_1(\ell, [E']_{\mathcal{R}}) = r$ for the equivalence class $[E']_{\mathcal{R}} \in \mathcal{E}/\mathcal{R}$ with E' as its representative. It is clear that $r \geq p$. Since $\eta_1 \equiv_{\mathcal{R}} \eta_1'$, we have η_1' in the form $\{(\ell, F_i : q_i)\}_{i \in I_1} \uplus \{(h_i, F_i : q_i)\}_{i \in I_2}$ such that

1. $I_1 \cap I_2 = \emptyset$;
2. for all $i \in I_1$, $F_i \; \mathcal{R} \; E'$;
3. for all $i \in I_2$, either $h_i \neq \ell$ or $(F_i, E') \notin \mathcal{R}$;
4. $\sum_{i \in I_1} q_i = r$.

From condition 2 and induction hypothesis, we know that for each $i \in I_1$ there exists η_{2i} s.t. $F_i \stackrel{\epsilon}{\Rightarrow}_c \eta_{2i}$, $\eta_2 \equiv_{\mathcal{R}} \eta_{2i}$ and η_{2i} in the form $\{(\tau, F_{ij} : q_{ij})\}_{j \in J_i}$. By repeated use of rule wea2' we can infer $F \stackrel{\epsilon}{\Rightarrow}_c \eta_2'$ where

$$\eta_2' = \{(\ell, F_{ij} : q_i q_{ij})\}_{i \in I_1, j \in J_i} \uplus \{(h_i, F_i : q_i)\}_{i \in I_2}.$$

Now let $\eta' = \frac{r-p}{r}\eta_1' + \frac{p}{r}\eta_2'$. By Lemma 3.5 we know that $F \stackrel{\epsilon}{\Rightarrow}_c \eta'$. We can verify that $\eta \equiv_{\mathcal{R}} \eta'$ as follows. For any $N \in \mathcal{E}/\mathcal{R}$ and $h \in \mathcal{L}$, there are three possibilities:

1. $h \neq \ell$: Then $\eta(h, N) = \eta_1(h, N) = \eta_1'(h, N) = \eta_2'(h, N)$. Hence

$$\eta'(h, N) = \frac{r-p}{r}\eta_1'(h, N) + \frac{p}{r}\eta_2'(h, N) = \frac{r-p}{r}\eta(h, N) + \frac{p}{r}\eta(h, N) = \eta(h, N).$$

2. $h = \ell$ and $E' \notin N$: Then we have

$$\begin{aligned}
\eta'(h, N) &= \tfrac{r-p}{r}\eta_1'(h, N) + \tfrac{p}{r}\eta_2'(h, N) \\
&= \tfrac{r-p}{r}\eta_1'(h, N) + \tfrac{p}{r}(\textstyle\sum_{i \in I_1} q_i \eta_{2i}(\tau, N) + \eta_1'(h, N)) \\
&= \tfrac{r-p}{r}\eta_1'(h, N) + \tfrac{p}{r}(\textstyle\sum_{i \in I_1} q_i \eta_2(\tau, N) + \eta_1'(h, N)) \\
&= \tfrac{r-p}{r}\eta_1'(h, N) + \tfrac{p}{r}(r\eta_2(\tau, N) + \eta_1'(h, N)) \\
&= \eta_1'(h, N) + p\eta_2(\tau, N) \\
&= \eta_1(h, N) + p\eta_2(\tau, N) \\
&= \eta(h, N)
\end{aligned}$$

3. $h = \ell$ and $N = [E']_{\mathcal{R}}$: Then we have

$$\begin{aligned}
\eta'(h, N) &= \tfrac{r-p}{r}\eta_1'(h, N) + \tfrac{p}{r}\eta_2'(h, N) \\
&= \tfrac{r-p}{r}\eta_1(h, N) + \tfrac{p}{r}(\textstyle\sum_{i \in I_1} q_i \eta_{2i}(\tau, N)) \\
&= \tfrac{r-p}{r} r + \tfrac{p}{r}(\textstyle\sum_{i \in I_1} q_i \eta_2(\tau, N)) \\
&= (r-p) + \tfrac{p}{r}(r\eta_2(\tau, N)) \\
&= (r-p) + p\eta_2(\tau, N) \\
&= (\eta_1(h, N) - p) + p\eta_2(\tau, N) \\
&= \eta(h, N)
\end{aligned}$$

- wea3: Similar to the last case.

- wea4: Then $\eta = \vartheta(X)$. Let $\eta_1 = \{(\tau, E_i : p_i)\}_i$, $E_i \Rightarrow \vartheta(X)$ for each i and $E \Rightarrow \eta_1$. By induction hypothesis there exists η_1' such that $F \stackrel{\epsilon}{\Rightarrow}_c \eta_1'$ and $\eta_1 \equiv_{\mathcal{R}} \eta_1'$. It is clear that η_1' must be in the form $\{(\tau, F_j : q_j)\}_j$ and by induction hypothesis $F_j \stackrel{\epsilon}{\Rightarrow}_c \vartheta(X)$ for each j. Therefore by rule wea4' we infer $F \stackrel{\epsilon}{\Rightarrow}_c \vartheta(X)$. By taking η' as $\vartheta(X)$, the desired result follows.

□

Now Lemma 3.14 follows immediately from Lemma A.2, 3.5 and 3.9.

A.2 Proof of Proposition 3.34

In [SS00] Stark and Smolka used a special function f that associates a probability to a nonprobabilistic transition so as to form a probabilistic transition. For example, let $E \equiv \frac{1}{3}a \oplus \frac{2}{3}b$, then $f(E \stackrel{a}{\longrightarrow} 0) = \frac{1}{3}$ and $f(E \stackrel{b}{\longrightarrow} 0) = \frac{2}{3}$. The function f can be characterised as $f = \sup_{i \geq 0} f_i$ for some functions $f_0, f_1, \ldots$ that take nonprobabilistic transitions to probabilities and respect some ordering. Therefore in the soundness proofs of some axioms, to show that $f(E \stackrel{a}{\longrightarrow} E') = f(F \stackrel{a}{\longrightarrow} F')$, it suffices to prove by induction on i that: (1) $f_i(E \stackrel{a}{\longrightarrow} E') \leq f(F \stackrel{a}{\longrightarrow} F')$ for all $i \geq 0$; (2) $f_i(F \stackrel{a}{\longrightarrow} F') \leq f(E \stackrel{a}{\longrightarrow} E')$ for all $i \geq 0$. In the presence of nondeterministic choice, however, this technique becomes unusable because now the probability with which an expression performs an action and evolves into another expression is not deterministic any more. For example, let $E \stackrel{\text{def}}{=} (\frac{1}{3}a \oplus \frac{2}{3}b) + (\frac{1}{2}a \oplus \frac{1}{2}c)$, then what is the value of $f(E \stackrel{a}{\longrightarrow} 0)$? Should it be $\frac{1}{3}$, $\frac{1}{2}$, or some value between them? Now the meaning of the function f is unclear because it depends on how the nondeterminism is resolved. Nevertheless our "bisimulation up to" techniques work well with Milner's transition induction technique, as can be seen in the proof of Proposition 3.34.

Lemma A.3 *1. If $E \rightarrow \{(\ell_i, E_i : p_i)\}_i$ then $E\{G/X\} \rightarrow \{(\ell_i, E_i\{G/X\} : p_i)\}_i$;*

2. If $E \Rightarrow \{(\ell_i, E_i : p_i)\}_i$ then $E\{G/X\} \Rightarrow \{(\ell_i, E_i\{G/X\} : p_i)\}_i$;

3. If $E \Rightarrow_c \{(\ell_i, E_i : p_i)\}_i$ then $E\{G/X\} \Rightarrow_c \{(\ell_i, E_i\{G/X\} : p_i)\}_i$;

4. If $E \stackrel{\epsilon}{\Rightarrow}_c \{(\ell_i, E_i : p_i)\}_i$ then $E\{G/X\} \stackrel{\epsilon}{\Rightarrow}_c \{(\ell_i, E_i\{G/X\} : p_i)\}_i$.

Proof: Straightforward by induction on inference. □

Lemma A.4 *1. If $E \rightarrow \vartheta(X)$ and $G \rightarrow \eta$ then $E\{G/X\} \rightarrow \eta$.*

2. If $E \Rightarrow \vartheta(X)$ and $G \rightarrow \eta$ then $E\{G/X\} \Rightarrow \eta$.

Proof: Straightforward by examining the structure of E. □

Lemma A.5 *If $E\{G/X\} \rightarrow \eta$ then one of the following two cases holds.*

1. $E \rightarrow \vartheta(X)$ and $G \rightarrow \eta$;

2. $\eta = \{(\ell_i, E_i\{G/X\} : p_i)\}_i$ and $E \rightarrow \{(\ell_i, E_i : p_i)\}_i$.

Proof: By induction on the depth of the inference of $E\{G/X\} \to \eta$. □

Proposition A.6 *If $E \approx F$ then $E\{G/X\} \approx F\{G/X\}$ for any $G \in \mathcal{E}$.*

Proof: Consider the relation $\mathcal{R} = \{(E\{G/X\}, F\{G/X\}) \mid E, F \in \mathcal{E} \text{ and } E \approx F\}$. Since $\approx$ is an equivalence relation, it follows that $\mathcal{R}$ is also an equivalence relation. So if we can show the assertion:

"If $E\{G/X\} \to \eta_1$ then there exists η_2 s.t. $F\{G/X\} \stackrel{\epsilon}{\Rightarrow}_c \eta_2$ and $\eta_1 \equiv_{\mathcal{R}} \eta_2$"

then it follows from Definition 3.13 that $\mathcal{R}$ is a weak probabilistic bisimulation.

We now prove the above assertion. From Lemma A.5 we know that there are two possibilities:

1. $E \to \vartheta(X)$ and $G \to \eta_1$. Thus $F \stackrel{\epsilon}{\Rightarrow}_c \vartheta(X)$ because $E \approx F$. From Lemma 3.51 we know that $F \Rightarrow \vartheta(X)$. By Lemma A.4 it follows that $F\{G/X\} \Rightarrow \eta_1$. We can simply take η_1 as η_2 and finish this case.

2. $\eta_1 = \{(\ell_i, E_i\{G/X\} : p_i)\}$ and $E \to \theta_1 = \{(\ell_i, E_i : p_i)\}_i$. Since $E \approx F$ there exists $\theta_2 = \{(h_j, F_j : q_j)\}_j$ s.t. $F \stackrel{\epsilon}{\Rightarrow}_c \theta_2$ and $\theta_1 \equiv_{\approx} \theta_2$. By Lemma A.3 we can derive $F\{G/X\} \stackrel{\epsilon}{\Rightarrow}_c \eta_2 = \{(h_j, F_j\{G/X\} : q_j)\}_j$. Observe that for any $E', F' \in \{E_i\}_i \cup \{F_j\}_j$ it holds that $E' \approx F'$ iff $E'\{G/X\}\ \mathcal{R}\ F'\{G/X\}$. Hence it follows from $\theta_1 \equiv_{\approx} \theta_2$ that $\eta_1 \equiv_{\mathcal{R}} \eta_2$ and we complete the proof of this case.

□

Proposition A.7 *If $E \simeq F$ then $E\{G/X\} \simeq F\{G/X\}$ for any $G \in \mathcal{E}$.*

Proof: Due to symmetry, it suffices to verify that if $E\{G/X\} \to \eta_1$ then there exists η_2 s.t. $F\{G/X\} \Rightarrow_c \eta_2$ and $\eta_1 \equiv_{\approx} \eta_2$. From Lemma A.5 we know that there are two possibilities:

1. $E \to \vartheta(X)$ and $G \to \eta_1$. Thus $F \Rightarrow_c \vartheta(X)$ because $E \simeq F$. From Lemma 3.51 we know that $F \Rightarrow \vartheta(X)$. By Lemma A.4 it follows that $F\{G/X\} \Rightarrow \eta_1$. We we can simply take η_1 as η_2 and finish this case.

2. $\eta_1 = \{(\ell_i, E_i\{G/X\} : p_i)\}$ and $E \to \theta_1 = \{(\ell_i, E_i : p_i)\}_i$. Since $E \simeq F$ there exists $\theta_2 = \{(h_j, F_j : q_j)\}_j$ s.t. $F \Rightarrow_c \theta_2$ and $\theta_1 \equiv_{\approx} \theta_2$. By Lemma A.3 we can derive $F\{G/X\} \Rightarrow_c \eta_2 = \{(h_j, F_j\{G/X\} : q_j)\}_j$. By Proposition A.6 it holds that for any $E', F' \in \{E_i\}_i \cup \{F_j\}_j$ if $E' \approx F'$ then $E'\{G/X\} \approx F'\{G/X\}$. Hence it follows from $\theta_1 \equiv_{\approx} \theta_2$ that $\eta_1 \equiv_{\approx} \eta_2$ and we complete the proof of this case.

□

Lemma A.8 *1. The following rules are derivable:*

$$\mathsf{D1}\ \frac{E_j \Rightarrow \eta}{\Sigma_{i\in 1..n} E_i \Rightarrow \eta}\quad \textit{for some } j \in 1..n \qquad \mathsf{D2}\ \frac{E\{\mu_X E/X\} \Rightarrow \eta}{\mu_X E \Rightarrow \eta}$$

$$\mathsf{D3}\ \frac{E_j \Rightarrow_c \eta}{\Sigma_{i\in 1..n} E_i \Rightarrow_c \eta}\quad \textit{for some } j \in 1..n \qquad \mathsf{D4}\ \frac{E\{\mu_X E/X\} \Rightarrow_c \eta}{\mu_X E \Rightarrow_c \eta}$$

$$\mathsf{D5}\ \frac{E \Rightarrow_c \{(\ell_i, E_i : p_i)\}_i \uplus \{(\ell, F : p)\} \quad F \Rightarrow_c \{(\tau, F_j : q_j)\}_j}{E \Rightarrow_c \{(\ell_i, E_i : p_i)\}_i \uplus \{(\ell, F_j : pq_j)\}_j}$$

$$\mathsf{D6}\ \frac{E \Rightarrow_c \{(\ell_i, E_i : p_i)\}_i \uplus \{(\tau, F : p)\} \quad F \Rightarrow_c \{(h_j, F_j : q_j)\}_j}{E \Rightarrow_c \{(\ell_i, E_i : p_i)\}_i \uplus \{(h_j, F_j : pq_j)\}_j}$$

2. *If* $\sum_{i\in 1..n} E_i \Rightarrow \eta$ *then* $E_j \Rightarrow \eta$ *for some* $j \in 1..n$*, with a shorter inference.*

3. *If* $\mu_X E \Rightarrow \eta$ *then* $E\{\mu_X E/X\} \Rightarrow \eta$*, with a shorter inference.*

Proof: Straightforward by induction on inference. □

Proof of Proposition 3.34 Let $\rho = \{\mu_X E/X\}$ and $\sigma = \{\mu_X F/X\}$. We show that the relation

$$\mathcal{R} = \{(G\rho, G\sigma) \mid E, F, G \in \mathcal{E} \text{ and } E \simeq F\}$$

is an observational equivalence up to $\simeq$. Because of symmetry we only need to show that if $G\rho \Rightarrow \eta$ there exists η' s.t. $G\sigma \Rightarrow_c \eta'$ and $\eta \equiv_{\mathcal{R}_\approx} \eta'$. The proof is carried out by induction on the depth of the inference of $G\rho \Rightarrow \eta$. There are several cases depending on the structure of G.

- $G \equiv X$: Then $G\rho \equiv \mu_X E \Rightarrow \eta$. By Lemma A.8 we have a shorter inference with the conclusion $E\rho \Rightarrow \eta$. By induction hypothesis there exists θ s.t. $E\sigma \Rightarrow_c \theta$ and $\eta \equiv_{\mathcal{R}_\approx} \theta$. Since $E \simeq F$ we have $E\sigma \simeq F\sigma$ by Proposition A.7. By Lemma 3.17 there exists η' s.t. $F\sigma \Rightarrow_c \eta'$ and $\theta \equiv_\approx \eta'$. By rule D4 it holds that $\mu_X F \Rightarrow_c \eta'$. At last it follows from Lemma 3.8 and the transitivity of $\equiv_{\mathcal{R}_\approx}$ that $\eta \equiv_{\mathcal{R}_\approx} \eta'$.

- $G \equiv \sum_{i\in 1..n} G_i$: If $G\rho \Rightarrow \eta$ then by Lemma A.8, $G_j\rho \Rightarrow \eta$ for some $j \in 1..n$ with a shorter inference. By induction hypothesis there exists η' s.t. $G_j\sigma \Rightarrow_c \eta'$ and $\eta \equiv_{\mathcal{R}_\approx} \eta'$. By rule D3 it holds that $G\sigma \Rightarrow_c \eta'$.

- $G \equiv \mu_Y G'$: If $G\rho \Rightarrow \eta$ then by Lemma A.8 there is a shorter inference with conclusion $G'\rho\{G\rho/Y\} \equiv G'\{G/Y\}\rho \Rightarrow \eta$. By induction hypothesis there exists η' s.t. $G'\{G/Y\}\sigma \Rightarrow_c \eta'$ and $\eta \equiv_{\mathcal{R}_\approx} \eta'$. By rule D4 it can be derived that $G\sigma \Rightarrow_c \eta'$.

- $G \equiv \bigoplus_{i\in I} p_i\ell_i.G_i$: In this case $G\rho \to \theta = \{(\ell_i, G_i\rho : p_i)\}_{i\in I}$. When $\eta = \vartheta(Y)$ for some variable Y the argument is simple. So we suppose that η is a distribution on $\mathcal{L} \times \mathcal{E}$. By induction on inference it can be proved that η is an extension of θ, i.e., there is a partition of I into three disjoint set I_1, I_2 and I_3 such that

  1. $\forall i \in I_2 \cup I_3,\ G_i\rho \Rightarrow \theta_i$ with a shorter inference than that of $G\rho \Rightarrow \eta$;
  2. $\forall i \in I_2, \theta_i = \{(\tau, E_{ij} : p_{ij})\}_j$;
  3. $\forall i \in I_3,\ \ell_i = \tau$ and $\theta_i = \{(\ell_{ij}, E_{ij} : p_{ij})\}_j$;

4. $\eta = \{(\ell_i, G_i\rho : p_i)\}_{i\in I_1} \uplus \biguplus_{i\in I_2}\{(\ell_i, E_{ij} : p_i p_{ij})\}_j \uplus \biguplus_{i\in I_3}\{(\ell_{ij}, E_{ij} : p_i p_{ij})\}_j$.

For each $i \in I_2 \cup I_3$, by induction hypothesis there exists θ'_i such that $G_i\sigma \Rightarrow_c \theta'_i$, $\theta_i \equiv_{\mathcal{R}_\approx} \theta'_i$ and

1. $\forall i \in I_2,\ \theta'_i = \{(\tau, F_{ik} : q_{ik})\}_k$;
2. $\forall i \in I_3,\ \theta'_i = \{(h_{ik}, F_{ik} : q_{ik})\}_k$.

Let m, n be the sizes of I_2 and I_3 respectively. Using rule D5 for m times and rule D6 for n times, we can derive $G\sigma \Rightarrow_c \eta'$, where

$$\eta' = \{(\ell_i, G_i\sigma : p_i)\}_{i\in I_1} \uplus \biguplus_{i\in I_2}\{(\ell_i, F_{ik} : p_i q_{ik})\}_k \uplus \biguplus_{i\in I_3}\{(h_{ik}, F_{ik} : p_i q_{ik})\}_k.$$

It remains to show that $\eta \equiv_{\mathcal{R}_\approx} \eta'$.

Let $p = \sum_{i\in I_1} p_i$, $\theta' = \{(\ell_i, G_i\rho : p_i/p)\}_{i\in I_1}$ and $\theta'' = \{(\ell_i, G_i\sigma : p_i/p)\}_{i\in I_1}$. It is immediate that $\theta' \equiv_{\mathcal{R}_\approx} \theta''$. For all $i \in I_2$, we let $\eta_i = \{(\ell_i, E_{ij} : p_{ij})\}_j$ and $\eta'_i = \{(\ell_i, F_{ik} : q_{ik})\}_k$. It follows from $\theta_i \equiv_{\mathcal{R}_\approx} \theta'_i$ that $\eta_i \equiv_{\mathcal{R}_\approx} \eta'_i$. Obviously we can rewrite η and η' as:

$$\eta = p\theta' + \textstyle\sum_{i\in I_2} p_i\eta_i + \sum_{i\in I_3} p_i\theta_i$$
$$\eta' = p\theta'' + \textstyle\sum_{i\in I_2} p_i\eta'_i + \sum_{i\in I_3} p_i\theta'_i$$

By Lemma 3.9 we have the desired result that $\eta \equiv_{\mathcal{R}_\approx} \eta'$.

□

A.3 Proof of Lemma 3.36

Lemma A.9 *Let $d_X(G) = n > 0$ and $\eta = \{(\ell_i, G_i : p_i)\}_{i\in I}$. Suppose $G\{E/X\} \to \eta$. For all $i \in I$, it holds that $G_i = G'_i\{E/X\}$ and*

1. *If $\ell_i = \tau$ then $d_X(G'_i) \geq n$;*
2. *If $\ell_i \neq \tau$ then $d_X(G'_i) \geq n - 1$.*

Proof: By induction on the depth of the inference of $G\{E/X\} \to \eta$. Let us examine the structure of G.

- $G \equiv X$ or Y: Impossible because $d_X(E) = 0$.
- $G \equiv \bigoplus_i p_i\ell_i.G_i$: Straightforward by definition.
- $G \equiv \sum_{i\in 1..n} G_i$: Then $G\{E/X\} \to \eta$ must be derived from a shorter inference with conclusion $G_j\{E/X\} \to \eta$ for some $j \in 1..n$. Thus the result follows from induction hypothesis, noting that $d_X(G_j) \geq d_X(G)$.
- $G \equiv \mu_Y G'$: Then $G\{E/X\} \to \eta$ is derived from the shorter inference of

  $$G'\{E/X\}\{G\{E/X\}/Y\} \equiv G'\{G/Y\}\{E/X\} \to \eta.$$

  So the result follows from induction hypothesis, by noting that $d_X(G'\{G/Y\}) = d_X(G)$.

□

Lemma A.10 *Let $d_X(G) = n$ and $\eta = \{(\ell_i, G_i : p_i)\}_{i\in I}$. Suppose $G\{E/X\} \Rightarrow \eta$. For all $i \in I$, it holds that*

1. *If $n > 0$ and $\ell_i = \tau$ then $G_i = G_i'\{E/X\}$ and $d_X(G_i') \geq n$;*
2. *If $n > 1$ and $\ell_i \neq \tau$ then $G_i = G_i'\{E/X\}$ and $d_X(G_i') \geq n - 1$.*

Proof: By induction on the depth of the inference of $G\{E/X\} \Rightarrow \eta$. There are three cases, depending on the last rule used in the inference.

- wea1: In this case $G\{E/X\} \to \eta$ and the result follows from Lemma A.9.
- wea2: Then $\eta = \{(\ell_i, G_i : p_i)\}_{i\in I} \uplus \{(\ell_0, H_j : p_0q_j)\}_{j\in J}$ and $G\{E/X\} \Rightarrow \eta$ is derived from the shorter inferences $G\{E/X\} \Rightarrow \{(\ell_i, G_i : p_i)\}_{i\in I} \uplus \{(\ell_0, G_0 : p_0)\}$ and $G_0 \Rightarrow \{(\tau, H_j : q_j)\}_{j\in J}$. By induction hypothesis, for each $i \in I \cup \{0\}$, it holds that

  1. If $n > 0$ and $\ell_i = \tau$ then $G_i = G_i'\{E/X\}$ and $d_X(G_i') \geq n$;
  2. If $n > 1$ and $\ell_i \neq \tau$ then $G_i = G_i'\{E/X\}$ and $d_X(G_i') \geq n - 1$.

  Particularly for G_0 there are two cases:

  1. if $\ell_0 = \tau$ then $G_0 = G_0'\{E/X\}$ and $d_X(G_0') \geq n > 0$. By induction hypothesis on the transition of $G_0'\{E/X\}$, we have $H_j = H_j'\{E/X\}$ and $d_X(H_j') \geq d_X(G_0') \geq n$ for each $j \in J$;
  2. if $\ell_0 \neq \tau$ then $G_0 = G_0'\{E/X\}$ and $d_X(G_0') \geq n - 1 > 0$. By induction hypothesis on the transition of $G_0'\{E/X\}$, we have $H_j = H_j'\{E/X\}$ and $d_X(H_j') \geq d_X(G_0') \geq n - 1$ for each $j \in J$.

- wea3: Then $\eta = \{(\ell_i, G_i : p_i)\}_{i\in I} \uplus \{(h_j, H_j : q_j)\}_{j\in J}$ and $G\{E/X\} \Rightarrow \eta$ is derived from the shorter inferences of $G\{E/X\} \Rightarrow \{(\ell_i, G_i : p_i)\}_{i\in I} \uplus \{(\tau, G_0 : p_0)\}$ and $G_0 \Rightarrow \{(h_j, H_j : q_j)\}_{j\in J}$. By induction hypothesis, for each $i \in I \cup \{0\}$, it holds that

  1. If $n > 0$ and $\ell_i = \tau$ then $G_i = G_i'\{E/X\}$ and $d_X(G_i') \geq n$;
  2. If $n > 1$ and $\ell_i \neq \tau$ then $G_i = G_i'\{E/X\}$ and $d_X(G_i') \geq n - 1$.

  Particularly for G_0 we have $G_0 = G_0'\{E/X\}$ and $d_X(G_0') \geq n > 0$. By induction hypothesis on the transition of $G_0'\{E/X\}$, it follows that for each $j \in J$

  1. if $h_j = \tau$ then $H_j = H_j'\{E/X\}$ and $d_X(H_j') \geq d_X(G_0') \geq n$ for each $j \in J$;
  2. $n > 1$ and $h_j \neq \tau$ then $H_j = H_j'\{E/X\}$ and $d_X(H_j') \geq d_X(G_0') - 1 \geq n - 1$.

□

Lemma A.11 *Suppose $d_X(G) > 1$, $\eta = \{(\ell_i, G_i : p_i)\}_{i\in I}$ and $G\{E/X\} \Rightarrow \eta$. Then $G_i = G_i'\{E/X\}$ for each $i \in I$. Moreover, $G\{F/X\} \Rightarrow \eta'$ and $\eta \equiv_{\mathcal{R}^*} \eta'$, where $\eta' = \{(\ell_i, G_i'\{F/X\} : p_i)\}_{i\in I}$ and $\mathcal{R} = \{(G\{E/X\}, G\{F/X\}) \mid$ for any $G \in \mathcal{E}\}$.*

Proof: A direct consequence of Lemma A.10. □

Proof of Lemma 3.36 Let $\eta = r_1\eta_1 + ... + r_n\eta_n$ and $G\{E/X\} \Rightarrow \eta_i$ for each $i \leq n$. By Lemma A.11, for each $i \leq n$, there exists η'_i s.t. $G\{F/X\} \Rightarrow \eta'_i$ and $\eta_i \equiv_{\mathcal{R}^*} \eta'_i$. Now let $\eta' = r_1\eta'_1 + ... + r_n\eta'_n$, thus $G\{F/X\} \Rightarrow_c \eta'$. By lemma 3.9 it follows that $\eta \equiv_{\mathcal{R}^*} \eta'$. □

A.4 Proof of Lemma 3.45

Proof:

1. We proceed by transition induction on the inference of $E \Rightarrow \eta$. There are three cases, concerning the last rules used.

   - wea1: Then $E \rightarrow \eta$ and there are several subcases.
     (a) psum: Then $E \equiv \bigoplus_i p_i\ell_i.E_i$ and the result is obvious by axiom **S2**.
     (b) nsum: Then $E \equiv \sum_{i\in I} F_i$ and $F_j \rightarrow \eta$ for some $j \in I$, with a shorter inference. By induction hypothesis we infer $\mathcal{A}_{gd} \vdash F_j = F_j + \bigoplus_i p_i\ell_i.E_i$, from which we have $\mathcal{A}_{gd} \vdash E \equiv \sum_{i\in I} F_i = \sum_{i\in I} F_i + \bigoplus_i p_i\ell_i.E_i = E + \bigoplus_i p_i\ell_i.E_i$.
     (c) rec: Then $E \equiv \mu_X E'$ and $E'\{E/X\} \rightarrow \eta$ for some E', with a shorter inference. By induction hypothesis $\mathcal{A}_{gd} \vdash E'\{E/X\} = E'\{E/X\} + \bigoplus_i p_i\ell_i.E_i$. By axiom **R1** we have $\mathcal{A}_{gd} \vdash E = E'\{E/X\} = E'\{E/X\} + \bigoplus_i p_i\ell_i.E_i = E + \bigoplus_i p_i\ell_i.E_i$.

   - wea2: Then $E \Rightarrow \{(\ell_i, E_i : p_i)\}_i \uplus \{(\ell, F : p)\}$, $F \Rightarrow \{(\tau, F_j : q_j)\}_j$ and $\eta \equiv \{(\ell_i, E_i : p_i)\}_i \uplus \{(\ell, F_j : pq_j)\}_j$. So we can infer as follows.

$$\begin{array}{rcl}
\mathcal{A}_{gd} \vdash E & \stackrel{IH}{=} & E + \bigoplus_i p_i\ell_i.E_i \oplus p\ell.F \\
& \stackrel{IH}{=} & E + \bigoplus_i p_i\ell_i.E_i \oplus p\ell.(F + \bigoplus_j q_j\tau.F_j) \\
& \stackrel{\mathbf{T3}}{=} & E + \bigoplus_i p_i\ell_i.E_i \oplus p\ell.(F + \bigoplus_j q_j\tau.F_j) + \bigoplus_i p_i\ell_i.E_i \oplus \bigoplus_j pq_j\ell.F_j \\
& = & E + \bigoplus_i p_i\ell_i.E_i \oplus \bigoplus_j pq_j\ell.F_j
\end{array}$$

   - wea3: Then $E \Rightarrow \{(\ell_i, E_i : p_i)\}_i \uplus \{(\tau, F : p)\}$, $F \Rightarrow \{(h_j, F_j : q_j)\}_j$ and $\eta \equiv \{(\ell_i, E_i : p_i)\}_i \uplus \{(h_j, F_j : pq_j)\}_j$. So we can infer as follows.

$$\begin{array}{rcl}
\mathcal{A}_{gd} \vdash E & \stackrel{IH}{=} & E + \bigoplus_i p_i\ell_i.E_i \oplus p\tau.F \\
& \stackrel{IH}{=} & E + \bigoplus_i p_i\ell_i.E_i \oplus p\tau.(F + \bigoplus_j q_jh_j.F_j) \\
& \stackrel{\mathbf{T2}}{=} & E + \bigoplus_i p_i\ell_i.E_i \oplus p\tau.(F + \bigoplus_j q_jh_j.F_j) + \bigoplus_i p_i\ell_i.E_i \oplus \bigoplus_j pq_jh_j.F_j \\
& = & E + \bigoplus_i p_i\ell_i.E_i \oplus \bigoplus_j pq_jh_j.F_j
\end{array}$$

2. Let $\eta = r_1\eta_1 + ... + r_n\eta_n$, $\eta_i \equiv \{(\ell_{ij}, E_{ij} : p_{ij})\}_j$ and $E \Rightarrow \eta_i$, for each $i \leq n$. We can do the following inference.

$$\begin{array}{rcl}
\mathcal{A}_{gd} \vdash E & \stackrel{(1)}{=} & E + \sum_{i\in 1..n} \bigoplus_j p_{ij}\ell_{ij}.E_{ij} \\
& \stackrel{\mathbf{C}}{=} & E + \sum_{i\in 1..n} \bigoplus_j p_{ij}\ell_{ij}.E_{ij} + \bigoplus_i \bigoplus_j r_ip_{ij}\ell_{ij}.E_{ij} \\
& = & E + \bigoplus_i \bigoplus_j r_ip_{ij}\ell_{ij}.E_{ij}
\end{array}$$

3. By induction on the inference $E \Rightarrow \vartheta(X)$. There are two cases, depending on the last rules used.

   - wea1: This case includes several subcases.
     - var: Then $E \equiv X$ and the result is obvious by axiom **S2**.
     - nsum: Then $E \equiv \sum_{i\in I} E_i$ and $E_j \Rightarrow \vartheta(X)$ for some $j \in I$. By induction hypothesis we infer $\mathcal{A}_{gd} \vdash E_j = E_j + X$, from which we have $\mathcal{A}_{gd} \vdash E \equiv \sum_{i\in I} E_i = \sum_{i\in I} E_i + X = E + X$.
     - rec: Then $E \equiv \mu_Y E'$ and $E'\{E/Y\} \rightarrow \vartheta(X)$ for some E' and $Y \neq X$. By induction hypothesis $\mathcal{A}_{gd} \vdash E'\{E/Y\} = E'\{E/Y\} + X$. By axiom **R1** we have $\mathcal{A}_{gd} \vdash E = E'\{E/Y\} = E'\{E/Y\} + X = E + X$.
   - wea4: Then $E \Rightarrow \{(\tau, E_i : p_i)\}_i$ and for each i it holds that $E_i \Rightarrow \vartheta(X)$. By the result of Clause 1 just proved above, we know that $\mathcal{A}_{gd} \vdash E = E + \bigoplus_i p_i\tau.E_i$. By induction hypothesis on each E_i we infer $\mathcal{A}_{gd} \vdash E = E + \bigoplus_i p_i\tau.(E_i + X)$. At last it follows from **T1** that $\mathcal{A}_{gd} \vdash E = E + \bigoplus_i p_i\tau.(E_i + X) + X = E + X$.

□

Appendix B

Proofs from Chapter 4

B.1 Some More Derived Rules

Cvn $[x = a]P =_\Delta [x = a][x \neq a_1] \cdots [x \neq a_n]P$ if $a \notin \{a_i \mid 1 \leq i \leq n\}$

Tv2 $P =_\Delta [x = a_1]P + [x = a_2]P + \cdots + [x = a_n]P$
if $\{b \in dom(\Delta_{\mathtt{c}}) \mid \Delta(b) \lessdot \Delta(x)\} = \{a_1, \cdots, a_n\}$

Tv3 If $P =_{\Delta, x:T} Q$ then $P =_{\Delta, x:S} Q$ for $S \lessdot T$

Iv1 If $P =_{\Delta, y:\Delta(x)_{\mathtt{i}}} Q$ then $x(y : T_1).P =_\Delta x(y : T_2).Q$

Iv2 If $P =_{\Delta \sqcap v:\Delta(x)_{\mathtt{o}}} Q$ then $\bar{x}v.P =_\Delta \bar{x}v.Q$

Proof: Among all the rules, the proof of **Iv2** is the hardest, so we report it below in details and omit the others.

Let $\{b \in dom(\Delta_{\mathtt{c}}) \mid \Delta(b) \lessdot \Delta(x)\} = \{a_1, \cdots, a_n\}$. When $n = 0$, the result is immediate by using **Tv1**. Suppose $n > 0$. For each $i \leq n$, $\Delta(a_i) \lessdot \Delta(x)$, there are two possibilities: (i) if $\Delta(a_i)\not\downarrow_{\mathtt{i}}$ then $\bar{a_i}b.P =_\Delta 0 =_\Delta \bar{a_i}b.Q$ by **Tout***; (ii) if $\Delta(a_i)\downarrow_{\mathtt{i}}$, then we have $\Delta(x)_{\mathtt{o}} \lessdot \Delta(a_i)_{\mathtt{o}} \lessdot \Delta(a_i)_{\mathtt{i}}$ by Proposition 4.2. There are two cases, depending on name v.

- v is a channel, say b. It follows from $P =_{\Delta \sqcap b:\Delta(x)_{\mathtt{o}}} Q$ that $P =_{\Delta \sqcap b:\Delta(a_i)_{\mathtt{i}}} Q$ by **Twea***. Using **Iout***, we have

$$\bar{a_i}b.P =_\Delta \bar{a_i}b.Q \tag{B.1}$$

  Finally,

$\bar{x}b.P$	$=_\Delta$	$[x = a_1]\bar{x}b.P + \cdots + [x = a_n]\bar{x}b.P$	by **Tv2**
	$=_\Delta$	$[x = a_1]\bar{a}_1b.P + \cdots + [x = a_n]\bar{a}_nb.P$	by **Tpre***
	$=_\Delta$	$[x = a_1]\bar{a}_1b.Q + \cdots + [x = a_n]\bar{a}_nb.Q$	by **(B.1)**
	$=_\Delta$	$\bar{x}b.Q$	by **Tpre*, Tv2**

- v is a variable, say y. By hypothesis, $\Delta \sharp \bar{x}y.P$ and $\Delta \sharp \bar{x}y.Q$ are configurations, then it holds that $\Delta(y) \lessdot \Delta(x)_{\mathtt{o}}$. By Proposition 4.1, it is easy to see that $\Delta \sqcap y : \Delta(x)_{\mathtt{o}} = \Delta$. Let the set $\{b \in dom(\Delta_{\mathtt{c}}) \mid \Delta(b) \lessdot \Delta(y)\} = \{b_1, \cdots, b_m\}$. We consider the non-trivial case that $m > 0$. For each $i \leq n, j \leq m$, by Proposition 4.2 we have

$$\Delta(b_j) \lessdot \Delta(y) \lessdot \Delta(x)_{\mathtt{o}} \lessdot \Delta(a_i)_{\mathtt{o}} \lessdot \Delta(a_i)_{\mathtt{i}}.$$

So $\Delta \sqcap b_j : \Delta(a_i)_{\mathtt{i}} = \Delta = \Delta \sqcap y : \Delta(x)_{\mathtt{o}}$. Therefore we can rewrite the hypothesis $P =_{\Delta \sqcap y:\Delta(x)_{\mathtt{o}}} Q$ as $P =_{\Delta \sqcap b_j:\Delta(a_i)_{\mathtt{i}}} Q$. Using **Iout***, we get the result

$$\bar{a}_i b_j.P =_{\Delta} \bar{a}_i b_j.Q \tag{B.2}$$

At last we can do the inference.

$\bar{x}y.P$

$=_{\Delta}$ $[x = a_1]\bar{x}y.P + \cdots + [x = a_n]\bar{x}y.P$ by **Tv2**

$=_{\Delta}$ $[x = a_1][y = b_1]\bar{x}y.P + \cdots + [x = a_1][y = b_m]\bar{x}y.P +$
$\cdots + [x = a_n][y = b_1]\bar{x}y.P + \cdots + [x = a_n][y = b_m]\bar{x}y.P$ by **Tv2**

$=_{\Delta}$ $[x = a_1][y = b_1]\bar{a}_1 b_1.P + \cdots + [x = a_1][y = b_m]\bar{a}_1 b_m.P +$
$\cdots + [x = a_n][y = b_1]\bar{a}_n b_1.P + \cdots + [x = a_n][y = b_m]\bar{a}_n b_m.P$ by **Tpre***

$=_{\Delta}$ $[x = a_1][y = b_1]\bar{a}_1 b_1.Q + \cdots + [x = a_1][y = b_m]\bar{a}_1 b_m.Q +$
$\cdots + [x = a_n][y = b_1]\bar{a}_n b_1.Q + \cdots + [x = a_n][y = b_m]\bar{a}_n b_m.Q$ by **(B.2)**

$=_{\Delta}$ $\bar{x}y.Q$ by **Tpre*, Tv2**

□

B.2 Proof of Theorem 4.36

Proof: We sketch the completeness proof of clause (ii), which is carried out by induction on the depth of $P + Q$; clause (i) can be shown in a similar way. Assume that P, Q are in hnf w.r.t. Δ and $\Delta = \Delta_{\mathtt{c}}, \widetilde{x} : \widetilde{T}$. Let $\Delta \sharp Q$ be a configuration respecting Γ. For some complete condition φ which are satisfiable by some legal substitution on Δ, let $P_{\varphi,a}$ be the sum of all active summands $\varphi_i \alpha_i.P_i$ of P such that $\{\mathbf{C1}, \mathbf{Tpre}*\} \vdash \varphi_i \alpha_i.P_i =_{\Delta} \varphi a(x : T_i).P_i$. We write

$$P_{\varphi,a} = \sum_{i=1}^{n} \varphi a(x : T_i).P_i \quad \text{and} \quad Q_{\varphi,a} = \sum_{j=1}^{m} \varphi a(x : S_j).Q_j$$

The key of the proof is to find, for each $1 \leq i \leq n$, a term R_i satisfying the following two properties.

$$\mathcal{A}_e \vdash \varphi a(x : T_i).P_i =_{\Delta} \varphi a(x : \Gamma(a)_{\mathtt{i}}).R_i \tag{B.3}$$

$$\mathcal{A}_e \vdash Q_{\varphi,a} =_{\Delta} Q_{\varphi,a} + \varphi a(x : \Gamma(a)_{\mathtt{i}}).R_i \tag{B.4}$$

Let $\sigma = \{\widetilde{b}/\widetilde{x}\}$ be a substitution which satisfies φ and $\Delta_{\mathtt{c}} \vdash \widetilde{b} : \widetilde{T}$. From $P\sigma \simeq^{e}_{\Delta_{\mathtt{c}}} Q\sigma$ we derive that $P_{\varphi,a}\sigma \simeq^{e}_{\Delta_{\mathtt{c}}} Q_{\varphi,a}\sigma$. Given $\Delta_{\mathtt{c}} \sharp P_{\varphi,a}\sigma \xrightarrow{a(x:T_i)} \Delta' \sharp P_i\sigma$, for each $b \in \{b \in dom(\Delta_{\mathtt{c}}) \mid \Delta_{\mathtt{c}}(b) <: \Delta_{\mathtt{c}}(a)_{\mathtt{o}}\} = \{c_1, \cdots, c_k\}$ we have a matching transition $\Delta_{\mathtt{c}} \sharp Q_{\varphi,a}\sigma \xrightarrow{a(x:S_{J(i,b)})} \Delta'' \sharp Q_{J(i,b)}\sigma$ such that

$$P_i\sigma\{b/x\} \simeq^{e}_{\Delta_{\mathtt{c}}} Q_{J(i,b)}\sigma\{b/x\}$$

for some function J from $[1, n]$ and $\{c_i \mid 1 \leq i \leq k\}$ to $[1, m]$. By the definition of hnf, P_i and $Q_{J(i,b)}$ are of the form $\varphi P'_i$ and $\varphi Q'_{J(i,b)}$ respectively. Here φ is complete on $dom(\Delta)$, but not on $dom(\Delta) \cup \{x\}$. We can complete it by adding conditions on the top which respects $\{b/x\}$. Let $\varphi_b = [x = b] \wedge \bigwedge_{u \in dom(\Delta) \setminus b} [x \neq u]$. It is easy to see that

$$([\varphi_b \wedge \varphi]P'_i)\sigma\{b/x\} \simeq^{e}_{\Delta_{\mathtt{c}}} ([\varphi_b \wedge \varphi]Q'_{J(i,b)})\sigma\{b/x\}.$$

By Lemma 4.20 we have $[\varphi_b \wedge \varphi]P'_i \doteq^{e}_{\Delta,x:\Delta(a)_\circ} [\varphi_b \wedge \varphi]Q'_{J(i,b)}$. By induction hypothesis

$$\mathcal{A}_e \vdash \varphi_b P_i =_{\Delta,x:\Delta(a)_\circ} \varphi_b Q_{J(i,b)}. \tag{B.5}$$

Now define $S_{i,l}$ for $l \leq k$ by

$$\begin{array}{rcl} S_{i,1} & = & Q_{J(i,c_1)} \\ S_{i,l} & = & [x = c_l]\ Q_{J(i,c_l)}\ S_{i,l-1} \quad \text{for } 1 < l \leq k \end{array}$$

Let R_i be defined as $S_{i,k}$. Using **C9** and **Cvn**, we decompose binary conditions in R_i into unary conditions.

$$\mathcal{A}_e \vdash R_i =_{\Delta,x:\Delta(a)_\circ} \varphi_{c_k} Q_{J(i,c_k)} + \varphi_{c_{k-1}} Q_{J(i,c_{k-1})} + \cdots + \varphi_{c_1} Q_{J(i,c_1)}$$

On the other hand by **Tv2** and **Cvn** we have

$$\mathcal{A}_e \vdash P_i =_{\Delta,x:\Delta(a)_\circ} \varphi_{c_k} P_i + \cdots + \varphi_{c_1} P_i.$$

By using (B.5) we have $\mathcal{A}_e \vdash P_i =_{\Delta,x:\Delta(a)_\circ} R_i$, from which we infer that $\mathcal{A}_e \vdash a(x : T_i).P_i =_\Delta a(x : \Gamma(a)_{\mathtt{i}}).R_i$ and $\mathcal{A}_e \vdash \varphi a(x : T_i).P_i =_\Delta \varphi a(x : \Gamma(a)_{\mathtt{i}}).R_i$ by **Iin*** and **Icon**. So we get the property in (B.3).

Finally with axiom **SP** we can show by induction on $0 < l \leq k$ that

$$\mathcal{A}_e \vdash Q_{\varphi,a} =_\Delta Q_{\varphi,a} + \varphi a(x : \Gamma(a)_{\mathtt{i}}).S_{i,l}. \tag{B.6}$$

Therefore (B.4) follows because it is the special case of (B.6) when $l = k$. □

Appendix C

Proofs from Chapter 5

C.1 Proofs from Section 5.2

Lemma C.1 *For two well-typed processes P and P', if $w : x$ (i.e., w and x have the same type) and $P' = P\{w/x\}$, then $wt(P) = wt(P')$.*

Proof: Trivial. □

Below we use $\mid wt(P) \mid$ to stand for the length of the vector $wt(P)$.

Lemma C.2 *Suppose $\mathcal{T} \vdash P$ and $P \xrightarrow{\alpha} P'$, then $\mid wt(P') \mid \leq \mid wt(P) \mid$.*

Proof: Straightforward by induction on the structure of P. □

Since the length of a vector can be extended by inserting zeros to the left end, we often assume implicitly, for simplicity of presentation, that several vectors have already been extended so as to be of equal length when discussing their relationship.

Lemma C.3 *Suppose $\mathcal{T} \vdash P, P \xrightarrow{aw} P'$, $lv(a) = i$, $wt(P) = \langle n_k, n_{k-1}, \cdots, n_1 \rangle$ and $wt(P') = \langle m_k, m_{k-1}, \cdots, m_1 \rangle$. Then $m_j \leq n_j$ for all j satisfying $i \leq j \leq k$.*

Proof: By induction on the transition of $P \xrightarrow{aw} P'$.

1. $P \equiv a(x).P_1 \xrightarrow{aw} P_1\{w/x\} \equiv P'$, in this case, $wt(P) = wt(P_1) = wt(P')$ by lemma C.1.

2. $P \equiv P_1 \mid P_2, P_1 \xrightarrow{aw} P_1'$ and $P' \equiv P_1' \mid P_2$, then we have

$$\begin{array}{rcccl} wt(P) & = & wt(P_1) + wt(P_2) & = & \langle n_k^1, n_{k-1}^1, \cdots, n_1^1 \rangle + \langle n_k^2, n_{k-1}^2, \cdots, n_1^2 \rangle \\ wt(P') & = & wt(P_1') + wt(P_2) & = & \langle m_k^1, m_{k-1}^1, \cdots, m_1^1 \rangle + \langle n_k^2, n_{k-1}^2, \cdots, n_1^2 \rangle \end{array}$$

   By induction hypothesis, $\forall j, i \leq j \leq k, m_j^1 \leq n_j^1$, it follows that $m_j = m_j^1 + n_j^2 \leq n_j^1 + n_j^2 = n_j$.

3. $P \equiv \nu b P_1, P_1 \xrightarrow{aw} P_1', P' \equiv \nu b P_1'$ and $b \neq a$, then $wt(P_1) = wt(P) = \langle n_k, n_{k-1}, \cdots, n_1 \rangle$, $wt(P_1') = wt(P') = \langle m_k, m_{k-1}, \cdots, m_1 \rangle$. By induction hypothesis, we know that $\forall j, i \leq j \leq k, m_j \leq n_j$.

4. $P \equiv P_1 + P_2, P_1 \xrightarrow{aw} P_1'$ and $P' \equiv P_1'$, then

$$\begin{array}{rcl} wt(P) & = & max\{wt(P_1), wt(P_2)\} = max\{\langle n_k^1, n_{k-1}^1, \cdots, n_1^1\rangle, \langle n_k^2, n_{k-1}^2, \cdots, n_1^2\rangle\} \\ wt(P') & = & wt(P_1') = \langle m_k^1, m_{k-1}^1, \cdots, m_1^1\rangle \end{array}$$

   By induction hypothesis, $\forall j, i \leq j \leq k, m_j^1 \leq n_j^1$, so $m_j^1 \leq n_j^1 \leq n_j$.

5. $P \equiv !a(x).P_1 \xrightarrow{aw} P \mid P_1\{w/x\} \equiv P'$. According to T-rep, any name which appears as subject of active output in P_1 has a level lower than that of a. Suppose $wt(P_1) = \langle n_l', n_{l-1}', \cdots, n_1'\rangle$, then $l < lv(a) = i$. Hence $wt(P') = wt(P) + wt(P_1\{w/x\}) = wt(P) + wt(P_1) = \langle n_k, \cdots, n_{l+1}, n_l + n_l', n_{l-1} + n_{l-1}', \cdots, n_1 + n_1'\rangle$. Therefore $m_j = n_j$ for all j satisfying $l \leq i \leq j \leq k$.

□

Lemma C.4 *Suppose* $\mathcal{T} \vdash P, P \xrightarrow{(\nu\widetilde{b})\bar{a}w} P'$, $lv(a) = i$, $wt(P) = \langle n_k, n_{k-1}, \cdots, n_1\rangle$ *and* $wt(P') = \langle m_k, m_{k-1}, \cdots, m_1\rangle$. *Then* $m_i < n_i$ *and* $m_j \leq n_j$ *for all* j *satisfying* $i < j \leq k$.

Proof: Similar to the proof of Lemma C.3. As an example, let us consider one case. Suppose $P \equiv \bar{a}w.P_1 \xrightarrow{\bar{a}w} P_1 \equiv P'$. After the transition, process P lost one output occurrence at level i previously contributed by name a. Other output occurrences remain unchanged. So it holds that $m_i = n_i - 1$ and $m_j = n_j$ for all $j \neq i$. □

Proof of Lemma 5.1

By induction on the transition system. We consider a typical case. Suppose $P \equiv P_1 \mid P_2$, $P_1 \xrightarrow{aw} P_1', P_2 \xrightarrow{(\nu\widetilde{b})\bar{a}w} P_2'$ and $P' \equiv (\nu\widetilde{b})(P_1' \mid P_2')$. Let $lv(a) = i$ and

$$\begin{array}{rcccl} wt(P) & = & wt(P_1) + wt(P_2) & = & \langle n_k^1, n_{k-1}^1, \cdots, n_1^1\rangle + \langle n_k^2, n_{k-1}^2, \cdots, n_1^2\rangle \\ wt(P') & = & wt(P_1') + wt(P_2') & = & \langle m_k^1, m_{k-1}^1, \cdots, m_1^1\rangle + \langle m_k^2, m_{k-1}^2, \cdots, m_1^2\rangle \end{array}$$

It follows from Lemma C.3 that $\forall j, i \leq j \leq k, m_j^1 \leq n_j^1$. From Lemma C.4 we infer that $m_i^2 < n_i^2$ and $\forall j, i < j \leq k, m_j^2 \leq n_j^2$. Combining the two results, we can draw the conclusion that $m_i < n_i$ and $\forall j, i < j \leq k, m_j \leq n_j$, in other words, $wt(P') \prec wt(P)$. □

C.2 Proofs from Section 5.3

When P is known or unimportant, we simply write $\mathcal{M}_i$ for $\mathcal{M}_{P,i}$. There are two additional special vectors widely used in this section.

1. $\mathbf{0}_i' = \langle(\mathcal{M}_k; n_k), \cdots, (\mathcal{M}_1; n_1)\rangle$ where (1) $\forall j \leq k, \mathcal{M}_j = [\,]$; (2) $\langle n_k, \cdots, n_1\rangle = \mathbf{0}_i$.

2. $\mathbf{0}_{ij}'' = \langle(\mathcal{M}_k; n_k), \cdots, (\mathcal{M}_1; n_1)\rangle$ where (1) $\mathcal{M}_i = [j]$ and $\mathcal{M}_l = [\,]$ for all l such that $l \neq i$; (2) $\langle n_k, \cdots, n_1\rangle = \mathbf{0}_i$.

The proofs of the following lemmas are carried out by induction on the transition $P \xrightarrow{\alpha} P'$. Here we write $a : \sharp^i\neg\mathtt{Nat}$ to mean that $a : \sharp^i T$ and $T \neq \mathtt{Nat}$ for some T.

Lemma C.5 *Suppose* $\mathcal{T}' \vdash P$ *and* $P \xrightarrow{aw} P'$.
1) If $a : \sharp^i\neg\texttt{Nat}$, *then* $t_{P'} \prec t_P + \mathbf{0}'_i$
2) If $a : \sharp^i\texttt{Nat}$, *then* $t_{P'} \prec t_P + \mathbf{0}''_{iw}$.

Proof: Let $t_P = \langle(\mathcal{M}_k; n_k), \cdots, (\mathcal{M}_1; n_1)\rangle$ and $t_{P'} = \langle(\mathcal{M}'_k; n'_k), \cdots, (\mathcal{M}'_1; n'_1)\rangle$. We consider two typical cases.

1. $P \equiv a(x).P_1 \xrightarrow{aw} P_1\{w/x\} \equiv P'$.

   (a) If $a : \sharp^i\neg\texttt{Nat}$, then all `Nat` values and output occurrences in P remain intact after the transition. So $t_{P'} = t_P \prec t_P + \mathbf{0}'_i$.

   (b) If $a : \sharp^i\texttt{Nat}$, there are two subcases.

      i. If $\forall \bar{b}u \in out(P_1), x \notin fvn(u)$ then no new `Nat` value is created in P_1. So we have $t_{P'} = t_P \prec t_P + \mathbf{0}''_{iw}$.

      ii. For each active output $\bar{b}u$ with $fvn(u) = \{x\}$, new constant values are generated. Let $u\{w/x\} = m \in \mathbb{N}$. Since u is consider as ∞ in $\mathcal{M}_{lv(b)}$ and it becomes m in $\mathcal{M}'_{lv(b)}$, we infer that $\mathcal{M}'_{lv(b)} \prec \mathcal{M}_{lv(b)}$ by the fact that $m < \infty$. As $wt(P)$ does not change, hence $t_{P'} \prec t_P \prec t_P + \mathbf{0}''_{iw}$.

2. $P \equiv !a(x).P_1 \xrightarrow{aw} P \mid P_1\{w/x\} \equiv P'$.

   (a) If $a : \sharp^i\neg\texttt{Nat}$, in this case only the first condition in Definition 5.4 is applicable, which ensures that all active outputs in P_1 have levels lower than i. So $wt(P') \prec wt(P) + \mathbf{0}_i$ and $\mathcal{M}_j = \mathcal{M}'_j$ for all $j \geq i$. Therefore it holds that $t_{P'} \prec t_P + \mathbf{0}'_i$.

   (b) If $a : \sharp^i\texttt{Nat}$, there are also two subcases.

      i. If $\forall b \in os(P_1), lv(b) < i$, then we are in the same situation as that of case 2.(a). So $t_{P'} \prec t_P + \mathbf{0}'_i \prec t_P + \mathbf{0}''_{iw}$.

      ii. If there are outputs at level i in P_1, say $\bar{b}u$, then rule T-rep requires that $u < x$, i.e., $u\{w/x\} < w$. It is easy to see that $\mathcal{M}_{P_1\{w/x\},i} \prec [w]$. It follows that $\mathcal{M}'_i \prec \mathcal{M}_i \uplus [w]$. Although it may occur that $n'_i > n_i$, the relation $t_{P'} \prec t_P + \mathbf{0}''_{iw}$ still holds because the compound vector is constructed in such a way that Nat-multisets are compared in a higher priority than output occurrences.

3. The other three cases can be analyzed by using induction hypothesis.

□

Lemma C.6 *Suppose* $\mathcal{T}' \vdash P$ *and* $P \xrightarrow{(\nu\tilde{b})\bar{a}w} P'$.
1) If $a : \sharp^i\neg\texttt{Nat}$, *then* $t_{P'} \preceq t_P - \mathbf{0}'_i$.
2) If $a : \sharp^i\texttt{Nat}$, *then* $t_{P'} \preceq t_P - \mathbf{0}''_{iw}$.

Proof: By induction on transitions. Consider the base case. Suppose that $P \equiv \bar{a}w.P_1 \xrightarrow{\bar{a}w} P_1 \equiv P'$. If $a : \sharp^i\neg\texttt{Nat}$, P lost one output occurrence after the transition. There is no change for `Nat` values in P_1. So $wt(P') = wt(P) - \mathbf{0}_i$ and $\mathcal{M}_{P',j} = \mathcal{M}_{P,j}$ for all $j \leq |\ wt(P)\ |$. In other words, we have

$t_{P'} = t_P - \mathbf{0}'_i$. If $a : \sharp^i\mathtt{Nat}$, P lost one output occurrence and a constant w at channel a. So $\mathcal{M}_{P',i} = \mathcal{M}_{P,i} - [w]$, $wt(P') = wt(P) - \mathbf{0}_i$ and $\forall j \neq i, \mathcal{M}_{P',j} = \mathcal{M}_{P,j}$, which means $t_{P'} = t_P - \mathbf{0}''_{iw}$. For other cases, induction hypothesis is applied. □

Proof of Lemma 5.6

Similar to the proof of Lemma 5.1. We consider the base case, the other cases follow from induction hypothesis. Let $P \equiv P_1 \mid P_2, P_1 \xrightarrow{aw} P'_1, P_2 \xrightarrow{(\nu\widetilde{b})\bar{a}w} P'_2$ and $P' \equiv (\nu\widetilde{b})(P'_1 \mid P'_2)$.

1. If $a : \sharp^i\neg\mathtt{Nat}$, then we have that $t_{P'_1} \prec t_{P_1} + \mathbf{0}'_i$ from Lemma C.5 and $t_{P'_2} \preceq t_{P_2} - \mathbf{0}'_i$ from Lemma C.6. So it can be derived that $t_{P'} = t_{P'_1} + t_{P'_2} \prec t_{P_1} + \mathbf{0}'_i + t_{P_2} - \mathbf{0}'_i = t_{P_1} + t_{P_2} = t_P$.

2. If $a : \sharp^i\mathtt{Nat}$, then from Lemma C.5 we have the result that $t_{P'_1} \prec t_{P_1} + \mathbf{0}''_{iw}$ and from Lemma C.6 we have $t_{P'_2} \preceq t_{P_2} - \mathbf{0}''_{iw}$. Hence it holds that $t_{P'} = t_{P'_1} + t_{P'_2} \prec t_{P_1} + \mathbf{0}''_{iw} + t_{P_2} - \mathbf{0}''_{iw} = t_{P_1} + t_{P_2} = t_P$. □

C.3 Extending $\mathcal{T}'$ with Polyadicity and Conditional

To allow for polyadic communication and if-then-else constructor, the extension of typing rules is straightforward.

$$\textsf{T-rep}\ \frac{\vdash u : \sharp^n\widetilde{V} \quad \widetilde{x} : \widetilde{V} \quad \vdash P \quad \forall\bar{v}\langle\widetilde{w}\rangle \in out(P), \bar{v}\langle\widetilde{w}\rangle \triangleleft u(\widetilde{x})}{\vdash !u(\widetilde{x}).P}$$

$$\textsf{T-if}\ \frac{\vdash w : \mathtt{bool} \quad \vdash P \quad \vdash Q}{\vdash\ \mathtt{if}\ w\ \mathtt{then}\ P\ \mathtt{else}\ Q}$$

The definition of $\triangleleft$ should be changed accordingly.

Definition C.7 *Suppose $u : \sharp^n(T_1, \cdots, T_k)$ and $v : \sharp^m(S_1, \cdots, S_l)$. We write $\bar{v}\langle\widetilde{w}\rangle \triangleleft u(\widetilde{x})$ if one of the two cases holds:*

1. *$m < n$*
2. *both of the following two conditions are met:*
   (a) *$m = n$ and $k = l$*
   (b) *there exists some $i \leq k$ such that $T_i = \mathtt{Nat}$, $w_i < x_i$ and $w_j \leq x_j$ for all $j \neq i$ with $T_j = \mathtt{Nat}$.*

In clause 2 we require that at least one argument of first-order should decrease its value, while in monadic case the unique first-order argument decreases.

In an input $u(\widetilde{x})$ or an output $\bar{v}\langle\widetilde{w}\rangle$, the order of arguments in the tuples $\widetilde{x}$ and $\widetilde{w}$ is not important. Without loss of generality, we assume that arguments of type $\mathtt{Nat}$ are always in the left end. In other words, we may consider that a tuple $\widetilde{x}$ is composed of two parts: $\widetilde{x} = \widetilde{x_1}; \widetilde{x_2}$, and x_i is of type $\mathtt{Nat}$ only if it is an element of $\widetilde{x_1}$. That is, all elements of $\widetilde{x_2}$ are of channel type or $\mathtt{bool}$ type.

Let $\bar{v}\langle w_n, \cdots, w_1; w'_m, \cdots, w'_1\rangle$ be an active output appearing in process P. Define $\underline{w_i}$ below for every w_i, where $i \in \{1, \cdots, n\}$.

$$\underline{w_i} = \begin{cases} w_i & \text{if } w_i \text{ is a constant, i.e., } fvn(w_i) = \emptyset \\ \infty & \text{otherwise.} \end{cases}$$

The definition of Nat-multiset, for the case of output, needs to be modified.

$$\mathcal{M}_{\bar{v}w.P,i} = \begin{cases} \mathcal{M}_{P,i} \uplus [\underline{w_n}, \cdots, \underline{w_1}] & \text{if } v : \sharp^i(\widetilde{\texttt{Nat}}; \widetilde{T}) \\ \mathcal{M}_{P,i} & \text{otherwise} \end{cases}$$

where $w = \langle w_n, \cdots, w_1; w'_m, \cdots, w'_1\rangle$. The intuition is that during a communication we consume an output $\bar{v}w$ and probably get some new outputs at level i, of the form $\bar{v}\langle w_n - m_n, \cdots, w_1 - m_1; w'_m, \cdots, w'_1\rangle$. As $\underline{w_i - m_i} < \underline{w_i}$ for some i and $\underline{w_j - m_j} \leq \underline{w_j}$ for all other j with $i, j \leq n$, we immediately infer that $\mathcal{M}_{P',i} <_{mul} \mathcal{M}_{P,i}$. The definition of compound vector remains unchanged. For conditionals, we can extend the definition of weight in this way: $wt(\texttt{ if } b \texttt{ then } P \texttt{ else } Q) = max\{wt(P), wt(Q)\}$. According to the new definition of Nat-multiset, properties similar to Lemma C.5 and C.6 are easy to prove. Lemma 5.6 and Theorem 5.7 still hold.

C.4 Proofs from Section 5.4

Proof of Lemma 5.12

1. There is a communication performed between a non-replicated input and an output message. That is, $P \equiv (\nu\widetilde{b})(a^{\epsilon}(x).P_1 \mid \bar{a}w.Q_1 \mid Q_2)$ for some a, P_1, Q_1, Q_2, w and $\widetilde{b}$, and $P' \equiv (\nu\widetilde{b})(P_1\{w/x\} \mid Q_1 \mid Q_2)$. Therefore we have that

$$\begin{array}{rcl} wt(P) & = & wt(P_1) + wt(\bar{a}w) + wt(Q_1) + wt(Q_2) \\ & \succ & wt(P_1) + wt(Q_1) + wt(Q_2) = wt(P') \end{array}$$

2. To derive this kind of transition, either if-t or if-f is used. If if-t is used then we have that $P \equiv (\nu\widetilde{b})((\texttt{ if } true \texttt{ then } P_1 \texttt{ else } Q_1) \mid Q_2)$ and $P' \equiv (\nu\widetilde{b})(P_1 \mid Q_2)$ for some $\widetilde{b}, P_1, Q_1$ and Q_2. Depending on the relation between $wt(P_1)$ and $wt(Q_1)$ we have $wt(P) \succ wt(P')$ if $wt(P_1) \prec wt(Q_1)$ and $wt(P) = wt(P')$ if $wt(P_1) \succeq wt(Q_1)$. The symmetric case for if-f is similar.

3. By the transition rule rep, each time a replicated process is invoked a fresh tag is produced. So there is no replicated process invoked in P_i for $1 \leq i \leq n-1$. Then there are two possibilities:

   (a) No replicated process invoked in P either. Therefore all communications take place between non-replicated inputs and outputs. Reasoning as in clause 1, one can derive that

   $$wt(P) \succ wt(P_1) \succ \cdots \succ wt(P')$$

   (b) A replicated process $!\kappa.Q$, with $\kappa = a_1(x_1).\cdots.a_n(x_n)$, is invoked in P and a new process $(a_2^{(l,2)}(x_2).\cdots.a_n^{(l,n)}(x_n).Q)\sigma$, for some σ, is spawned. The subsequent reductions consume the input prefixes from $a_2^{(l,2)}\sigma(x_2)$ to $a_n^{(l,n)}\sigma(x_n)$ and their corresponding outputs.

Thus we have the relation

$$wt(P') + wt(\kappa) = wt(P) + wt(Q\sigma').$$

Substitution of names does not affect the weight of a process, so $wt(Q\sigma') = wt(Q)$. The side condition of rule rep requires that $wt(\kappa) \succ wt(Q)$. Hence we have the conclusion that $wt(P) \succ wt(P')$.

□

Proof of Lemma 5.13

Let $n = \rho(l)$.

1. Since P is regular, the transition with tag (l, i) must originate from a communication between an active output and a replicated input. So R must be of the form:

$$\begin{cases} (\nu\widetilde{b})(!a_1(x_1).\cdots.a_n(x_n).P \mid \bar{a_1}w \mid Q) & \text{if } i = 1 \\ (\nu\widetilde{b})(!a_1(x_1).\cdots.a_n(x_n).P \mid (a_i^{(l,i)}(x_i).\cdots.a_n^{(l,n)}(x_n).P)\sigma \mid \bar{a_i'}w \mid Q) & \text{if } 1 < i < n \end{cases}$$

   with $a_i\sigma = a_i'$. To have a subsequent transition with tag ϵ, Q must be of the form: $c^{\epsilon}(x).Q_1 \mid \bar{c}w.Q_2 \mid Q_3$ for some c, w, Q_1, Q_2 and Q_3. It is evident that R also have the reduction path $R \xrightarrow{\epsilon} R_1' \xrightarrow{(l,i)} R'$. The case for $t = \epsilon'$ is also straightforward.

2. Let $m = \rho(l')$. As in the proof of clause 1 we know that the transitions with non-special tags come from replicated inputs. Depending on whether l and l' come from the same input pattern or not, we have the following two cases:

   (a) They are generated by two different input patterns, that is, there exist at least two replicated inputs in P, say $!a_1(x_1).\cdots.a_n(x_n).P_1$ and $!b_1(x_1).\cdots.b_m(x_m).P_2$ respectively. There are four possibilities. Let us consider the typical case that $j \neq 1$ and $i \neq 1$. Then R should be of the form

$$\begin{aligned} R \equiv\ & (\nu\widetilde{c})(!b_1(y_1).\cdots.b_n(y_n).P_2 \mid !a_1(x_1).\cdots.a_n(x_n).P_1 \\ & \mid (b_j^{(l',j)}(y_j).\cdots.b_m^{(l',m)}(y_m).P_2)\sigma_1 \mid (a_i^{(l,i)}(x_i).\cdots.a_n^{(l,n)}(x_n).P_1)\sigma_2 \\ & \mid \bar{b}_j'w' \mid Q) \end{aligned}$$

   with $b_j\sigma_1 = b_j'$. Since $j < \rho(l')$ the consumption of $b_j\sigma_1(y_j)$ does not liberate any output, and an output on $a_i\sigma_2$ should be directly available in Q so as to make the subsequent communication on $a_i\sigma_2$ possible, which means that

$$Q \equiv \begin{cases} \bar{a}_i'w \mid Q_2 & \text{if } i < n \\ \bar{a}_i'w.Q_1 \mid Q_2 & \text{if } i = n \end{cases}$$

   with $a_i\sigma_2 = a_i'$. Obviously in both cases R can take another reduction path: $R \xrightarrow{(l,i)} R_1' \xrightarrow{(l',j)} R'$ for some R_1'.

   (b) l and l' originate from the same input pattern $!a_1(x_1).\cdots.a_n(x_n).P_1$, which has been invoked two times. The arguments are similar to Case (a).

□

Proof of Lemma 5.14

We consider the inductive step. Suppose P has an infinite reduction sequence $P \equiv P_0 \xrightarrow{t_1} P_1 \xrightarrow{t_2} \cdots \xrightarrow{t_i} P_i \xrightarrow{t_{i+1}} \cdots$. We shall do case analysis to find some process Q satisfying the three conditions: (i) Q is also non-terminating; (ii) Q is regular; (iii) $wt(P) \succ wt(Q)$.

At first it is clear that if $t_j = (l, i)$ and $i < \rho(l)$, then the atomic tag l is generated by invoking an input pattern, since in P there are only special tags.

Case 1: If $t_1 = \epsilon'$, by Lemma 5.12 there are two possibilities. If $wt(P) \succ wt(P_1)$ we can set $Q = P_1$. If $wt(P) = wt(P_1)$, we need to start the search from t_2. Note that any reduction sequence by consecutively using rules if-t or if-f is finite since the size of the starting process decreases step by step. So we will find either a tag ϵ' that decreases weight or a tag of the form ϵ or (l, i), which directs the analysis to Case 2 or Case 3 accordingly.

Case 2: If $t_1 = \epsilon$, then by Lemma 5.12 we know that $wt(P) \succ wt(P_1)$. P_1 is just the process Q we are finding.

Case 3: If $t_1 = (l, i)$ and $\rho(l) > 0$, then $i = 1$ since P is regular. Let $n = \rho(l)$.
−If $n = 1$, then by Lemma 5.12 it holds that $wt(P) \succ wt(P_1)$. So we can set $Q = P_1$.
− If $n > 1$ and hence a new process $R \stackrel{\text{def}}{=} (a_2^{(l,2)}(x_2).\cdots.a_n^{(l,n)}(x_n).R_0)\sigma$ appears in P_1.

1. If R does not participate in any communication among the infinite sequence $P_1 \xrightarrow{t_2} \cdots \xrightarrow{t_i} P_i \xrightarrow{t_{i+1}} \cdots$, then replacing R with $\mathsf{0}$ does not affect the sequence. More precisely, let $P_1 = (\nu\widetilde{c})(!a_1(x_1).\cdots.a_n(x_n).R_0 \mid R \mid R_1)$, for some R_1, and $Q = (\nu\widetilde{c})(!a_1(x_1).\cdots.a_n(x_n).R_0 \mid \mathsf{0} \mid R_1)$. Q can produce the same infinite reduction sequence as that of P_1 with $\mathsf{0}$ in place of R at the top level, but with $wt(Q) \prec wt(P)$ because P consumes an output during the transition $P \xrightarrow{(l,1)} P_1$.

2. If R participates in a communication among the sequence, then there exists i such that $t_i = (l, 2)$. We need to classify all the reductions between P_1 and P_i. There are two subcases to consider.

   (a) If all t_j for $1 < j < i$ are of the forms ϵ or ϵ', then we use Lemma 5.13 for $(i-2)$ times and push $(l, 1)$ forward until to the proper left of $(l, 2)$. The resulting sequence is of the form:

   $$P \xrightarrow{t_2} P_2' \xrightarrow{t_3} \cdots \xrightarrow{t_{i-1}} P_{i-1}' \xrightarrow{(l,1)(l,2)} P_i' \longrightarrow \cdots$$

   By Lemma 5.12, we have the relations

   $$wt(P) \succeq wt(P_2') \succeq \cdots \succeq wt(P_{i-1}')$$

   (b) If there is a partition of the set $\{j \mid 1 < j < i\}$ by I_1 and I_2 such that all $t_j \in C_1 = \{t_i \mid i \in I_1\} = \{t_{11}, \cdots, t_{1k}\}$ are of the forms ϵ or ϵ' and all $t_j \in C_2 = \{t_i \mid i \in I_2\} = \{t_{21}, \cdots, t_{2k'}\}$ are of the form (l_j, n_j) with $\rho(l_j) > 0$.

      i. If $\forall j \in I_2, n_j < \rho(l_j)$, i.e., no input pattern is complete (since for each j not all tags from $(l_j, 1)$ to $(l_j, \rho(l_j))$ are in the set C_2), then by using Lemma 5.13 for finite many

times we can push all tags in C_1 to the left of $(l,1)$ and preserve their order. The sequence changes into this form:

$$P \xrightarrow{t_{11}} P_{11} \xrightarrow{t_{12}} \cdots \xrightarrow{t_{1k}} P_{1k} \xrightarrow{(l,1)} \xrightarrow{t_{21}} \cdots \xrightarrow{t_{2k'}} \xrightarrow{(l,2)} \cdots$$

Similarly, by using Lemma 5.13, we can push all tags in C_2 to the right of $(l,2)$.

$$P \xrightarrow{t_{11}} P_{11} \xrightarrow{t_{12}} \cdots \xrightarrow{t_{1k}} P_{1k} \xrightarrow{(l,1)}\xrightarrow{(l,2)} P'_i \xrightarrow{t_{21}} \cdots \xrightarrow{t_{2k'}} \cdots$$

By Lemma 5.12 it follows that

$$wt(P) \succeq wt(P_{11}) \succeq \cdots \succeq wt(P_{1k}).$$

ii. If there is a set $I'_2 \subseteq I_2$ such that $\forall j \in I'_2, t_j = (l_j, \rho(l_j))$, i.e., all tags in I'_2 are the tags of ending inputs in some input patterns. These patterns can be completed by tags between $(1,l)$ and $(l,2)$. We shall use Lemma 5.13 to sort out all complete patterns and push them to the left of $(l,1)$.

A. Starting from $(l,1)$ we scan the sequence forward to find the first tag $(l_1, \rho(l_1))$ for some atomic tag l_1 because we want to make all tags with atomic tag l_1 be in consecutive positions by "sequeezing out" other tags to the left of $(l_1,1)$ or to the right of $(l_1, \rho(l_1))$. All tags between $(l_1,1)$ and $(l_1,\rho(l_1))$ are of one of the three forms: ϵ, ϵ' or (l_j, n_j) with $n_j < \rho(l_j)$. As we did in Case i, it is feasible to push all ϵ and ϵ' backward and all (l_j, n_j) forward so that only tags with atomic tag l_1 are left between $(l_1,1)$ and $(l_1,\rho(l_1))$ (these tags are already in ascending order since they come from the same input pattern, say $a_1(x_1).\cdots.a_{\rho(l_1)}(x_{\rho(l_1)})$, and the consumption of these input prefixes goes from left to right). After the operations, we get a reduction sequence like

$$P \xrightarrow{(l,1)} \cdots \xrightarrow{\epsilon} \xrightarrow{\epsilon'} \cdots \underbrace{\xrightarrow{(l_1,1)}\xrightarrow{(l_1,2)} \cdots \xrightarrow{(l_1,\rho(l_1))}}_{\tau^{l_1}} \cdots \xrightarrow{(l_j,n_j)} \cdots \xrightarrow{(l,2)} \cdots$$

B. Find the next tag $(l_2, \rho(l_2))$ for some atomic tag l_2 and make all tags with atomic tag l_2 in consecutive positions. Now we can treat tags in group τ^{l_1} as a whole and push them backward just as what we do for tag ϵ. We repeat this operation for other group τ^{l_j} as long as $(l_j, \rho(l_j))$ lies between $(l,1)$ and $(l,2)$. At the end of this stage, we have a sequence as follows.

$$P \xrightarrow{(l,1)} \cdots \xRightarrow{\tau^{l_1}} \cdots \xRightarrow{\tau^{l_2}} \cdots \xRightarrow{\tau^{l_j}} \cdots \xrightarrow{(l,2)} \cdots$$

where $\xRightarrow{\tau^{l_j}}$ stands for $\xrightarrow{(l_j,1)}\xrightarrow{(l_j,2)} \cdots \xrightarrow{(l_j,\rho(l_j))}$.

C. For other tags t_j with $j \notin I'_2$ and $j \in I_2$, which do not belong to a complete group, we push them forward to the right of $(l,2)$, keeping their order. At this moment, there are still tags like ϵ and ϵ' between $(l,1)$ and $(l,2)$.

$$P \xrightarrow{(l,1)} \cdots \xrightarrow{t} \cdots \xRightarrow{\tau^{l_1}} \cdots \xrightarrow{t} \cdots \xRightarrow{\tau^{l_j}} \cdots \xrightarrow{(l,2)} \cdots$$

where $t \in \{\epsilon, \epsilon'\}$.

D. Push $(l,1)$ forward until to the proper left of $(l,2)$ so as to yield this sequence:

$$P \xrightarrow{t} P'_{11} \cdots \xrightarrow{t} \xRightarrow{\tau^{l_1}} \cdots \xrightarrow{t} \xRightarrow{\tau^{l_j}} \cdots \xrightarrow{t} P'_{j'k_{j'}} \xrightarrow{(l,1)} \xrightarrow{(l,2)} P'_i \cdots$$

where $t \in \{\epsilon, \epsilon'\}$. By Lemma 5.12 it follows that

$$wt(P) \succeq wt(P'_{11}) \succeq \cdots \succeq wt(P'_{j'k_{j'}})$$

In the above four steps, when we commute reductions like $\xrightarrow{(l_j,n_j)} \xrightarrow{t_i}$, the condition $n_j < \rho(l_j)$ is always satisfied. This ensures the correct use of Lemma 5.13.

If $n = 2$, by Lemma 5.12 and the transitivity of $\succ$, we have that $wt(P) \succ wt(P'_i)$ and so Q can be set as P'_i. If $n > 2$ we repeat the operations done for $(l,1)$ on (l,i) with $1 < i < \rho(l)$. There are two possibilities for the ultimate result:
1) either $(l, i+1)$ does not appear in the subsequent reductions, then we replace the process $R \stackrel{\text{def}}{=} (a_{i+1}^{(l,i+1)}(x_{i+1}).\cdots.a_n^{(l,n)}(x_n).R_0)\sigma$ with $\mathsf{0}$ and get a non-terminating process Q such that $wt(P) \succ wt(Q)$;
2) or we complete the input pattern with atomic tag l and have a sequence like

$$P \xrightarrow{t_i} \cdots \xrightarrow{(l,1)} \xrightarrow{(l,2)} \cdots \xrightarrow{(l,n)} Q \xrightarrow{t_j} \cdots$$

In this case we also have $wt(P) \succ wt(Q)$ according to previous operations and Lemma 5.12.

Note that there are possibly three kinds of tags lying in the ultimate sequence between P and Q:
1) tags ϵ or ϵ';
2) tags belonging to complete input patterns;
3) tags not belonging to complete input patterns, but the continuations of these incomplete input patterns are discarded in Q since we have substituted $\mathsf{0}$ for them.

Therefore each new atomic tag l with $\rho(l) > 0$ created by the derivatives of P is usded up when reaching Q. As P is regular, Q must be regular as well. Hence the induction hypothesis applies and it maintains that Q is terminating. At this point contradiction arises. □

C.5 Proofs from Section 5.5

Lemma C.8 *If $n(\mathcal{R}) \cap \widetilde{x} = \emptyset$ then $(\mathcal{R} + \mathcal{R}')\Downarrow_{\widetilde{x}} = \mathcal{R} + \mathcal{R}'\Downarrow_{\widetilde{x}}$.*

Proof: Let $\mathcal{R}'' = \mathcal{R} + \mathcal{R}'$.

$$\begin{aligned}
& (\mathcal{R} + \mathcal{R}')\Downarrow_{\widetilde{x}} \\
= \; & \{(a,b) \mid a,b \notin \widetilde{x} \text{ and } a\mathcal{R}''c_1\mathcal{R}'' \cdots \mathcal{R}''c_n\mathcal{R}''b \text{ for some } \widetilde{c} \subseteq \widetilde{x} \text{ and } n \geq 0\} \\
= \; & \{(a,b) \mid a,b \notin \widetilde{x} \text{ and } a\mathcal{R}b\} \\
& \cup \{(a,b) \mid a,b \notin \widetilde{x} \text{ and } a\mathcal{R}'c_1\mathcal{R}' \cdots \mathcal{R}'c_n\mathcal{R}'b \text{ for some } \widetilde{c} \subseteq \widetilde{x} \text{ and } n \geq 0\} \\
= \; & \mathcal{R} \cup \mathcal{R}'\Downarrow_{\widetilde{x}} \\
= \; & \mathcal{R} + \mathcal{R}'\Downarrow_{\widetilde{x}}
\end{aligned}$$

□

Let $\mathcal{R}$ be a partial order and σ be a substitution of names. We say $\mathcal{R}\sigma$ is well defined if $\mathcal{R}\sigma = \{(x\sigma, y\sigma) \mid (x,y) \in \mathcal{R}\}$ is a partial order. For the multiset $\mathcal{M} = [x_1, \cdots, x_n]$ we write $\mathcal{M}\sigma = [x_1\sigma, \cdots, x_n\sigma]$.

Lemma C.9 *If $\mathcal{M}_1\ \mathcal{R}_{mul}\ \mathcal{M}_2$ then*
(1) $\mathcal{M}_1\ \mathcal{R}'_{mul}\ \mathcal{M}_2$ with $\mathcal{R}' = \mathcal{R} + \mathcal{S}$.
(2) $(\mathcal{M}_1 \uplus \mathcal{M})\ \mathcal{R}_{mul}\ (\mathcal{M}_2 \uplus \mathcal{M})$ for any multiset $\mathcal{M}$ over $n(\mathcal{R})$.
(3) $\mathcal{M}_1\sigma\ \mathcal{R}\sigma_{mul}\ \mathcal{M}_2\sigma$ when $\mathcal{R}\sigma$ is well defined.

Proof: We only need the definition of multiset ordering. (1) Since $\mathcal{R}'$ is a superset of $\mathcal{R}$, it holds that $x\mathcal{R}y$ implies $x\mathcal{R}'y$. (2) Trivial. (3) Since $\mathcal{R}\sigma$ is well defined, it follows that $x\mathcal{R}y$ implies $x\sigma\ \mathcal{R}\sigma\ y\sigma$. □

Given a multiset $\mathcal{M}$ and a partial order $\mathcal{R}$ on names, we extract from $\mathcal{M}$ a sub-multiset in the following way:

$$\mathcal{M}_{\mathcal{R}}(x) \stackrel{\text{def}}{=} \begin{cases} \mathcal{M}(x) & x \in n(\mathcal{R}) \\ 0 & x \notin n(\mathcal{R}) \end{cases}$$

Note that here we consider a multiset $\mathcal{M}$ with elements from set V as a function $\mathcal{M} : V \mapsto \mathbb{N}$ (cf. [Bez03]). Clearly all elements in $\mathcal{M}_{\mathcal{R}}$ belong to $n(\mathcal{R})$.

The following lemma provides an alternative characterisation of the relation $\widehat{\mathcal{R}}$. It shows that names not in $n(\mathcal{R})$ are invariant with respect to the multiset ordering.

Lemma C.10 *Suppose $P\ \widehat{\mathcal{R}}\ Q$, $\mathcal{M}^1 = mos_{\mathcal{R}}(P)$ and $\mathcal{M}^2 = mos_{\mathcal{R}}(Q)$. Then $\mathcal{M}^1_{\mathcal{R}}\ \mathcal{R}_{mul}\ \mathcal{M}^2_{\mathcal{R}}$.*

Proof: From $P\ \widehat{\mathcal{R}}\ Q$ we know that: (i) $\mathcal{M}^1 = \mathcal{M} \uplus \mathcal{M}_1$; (ii) $\mathcal{M}^2 = \mathcal{M} \uplus \mathcal{M}_2$; (iii) $\mathcal{M}_1\ \mathcal{R}_{mul}\ \mathcal{M}_2$. Since all elements in $\mathcal{M}_1$ and $\mathcal{M}_2$ belong to $n(\mathcal{R})$, it is easy to see that $\mathcal{M}^1_{\mathcal{R}} = \mathcal{M}_{\mathcal{R}} \uplus \mathcal{M}_1$ and $\mathcal{M}^2_{\mathcal{R}} = \mathcal{M}_{\mathcal{R}} \uplus \mathcal{M}_2$. From Lemma C.9(2), it follows that $\mathcal{M}^1_{\mathcal{R}}\ \mathcal{R}_{mul}\ \mathcal{M}^2_{\mathcal{R}}$. □

Lemma C.11 *If the partial order $\mathcal{R}$ is finite, then there exists no infinite sequence like*

$$P_0\ \widehat{\mathcal{R}}\ P_1\ \widehat{\mathcal{R}}\ P_2\ \widehat{\mathcal{R}} \cdots$$

Proof: Since $\mathcal{R}$ is finite, it is well-founded, so is the induced multiset ordering $\mathcal{R}_{mul}$. Suppose there exists such an infinite sequence. Let $\mathcal{M}^i = mos_{\mathcal{R}}(P_i)$. By Lemma C.10, we would have the sequence

$$\mathcal{M}^0_{\mathcal{R}}\ \mathcal{R}_{mul}\ \mathcal{M}^1_{\mathcal{R}}\ \mathcal{R}_{mul}\ \mathcal{M}^2_{\mathcal{R}}\ \mathcal{R}_{mul} \cdots$$

which contradicts the well-foundedness of $\mathcal{R}_{mul}$. □

Lemma C.12 *If $P\ \widehat{\mathcal{R}}\ Q$ then*
(1) $P\ \widehat{\mathcal{R}'}\ Q$ with $\mathcal{R}' = \mathcal{R} + \mathcal{S}$
(2) $P \mid R\ \widehat{\mathcal{R}}\ Q \mid R$
(3) $P\sigma\ \widehat{\mathcal{R}\sigma}\ Q\sigma$ when $\widehat{\mathcal{R}\sigma}$ is well defined.
(4) $P'\ \widehat{\mathcal{R}}\ Q'$ with $mos_{\mathcal{R}}(P) = mos_{\mathcal{R}}(P')$ and $mos_{\mathcal{R}}(Q) = mos_{\mathcal{R}}(Q')$.

Proof: Straightforward. The first and third clause of Lemma C.9 are used to prove (1) and (3) respectively. □

The next two lemmas illustrate the basic properties of the type system $\mathcal{T}'''$.

Lemma C.13 *If $\mathcal{R} \vdash P$ then $n(\mathcal{R}) \subseteq fn(P)$.*

Proof: By trivial induction on the structure of P. □

Lemma C.14 *If $\mathcal{R} \vdash P$, $\widetilde{x} : \widetilde{w}$, $\sigma = \{\widetilde{w}/\widetilde{x}\}$ and $\mathcal{R}\sigma$ is well defined, then $\mathcal{R}\sigma \vdash P\sigma$.*

Proof: The derivation of $\mathcal{R} \vdash P$ forms a tree tr with the conclusion as root. If we replace all occurrences of x_i with w_i we get another tree tr'. By induction on the depth of tr' it can be shown that tr' is a valid derivation tree with root $\mathcal{R}\sigma \vdash P\sigma$. □

Proof of Theorem 5.20

By induction on the depth of the derivation $P \xrightarrow{\alpha} P'$. Let us consider the last rule used in the derivation.

1. Rule in In this case $P = a(\widetilde{x}).P_1$ and $P' = P_1\sigma$, where $\sigma = \{\widetilde{w}/\widetilde{x}\}$. From $\mathcal{R} \vdash P$ we infer that $a : \sharp^n_{\mathcal{S}}\widetilde{V}$, $\widetilde{x} : \widetilde{V}$, $\mathcal{R}' \vdash P_1$, $\mathcal{S} = \mathcal{R}'/\widetilde{x}$ and $\mathcal{R} = \mathcal{R}' \Downarrow_{\widetilde{x}}$.

   (a) If $\mathcal{S} = \emptyset$ then $n(\mathcal{R}') \cap \widetilde{x} = \emptyset$. Obviously $\mathcal{R}'\sigma$ is well defined since $\mathcal{R}'\sigma = \mathcal{R}'$. By Lemma C.14 we have $\mathcal{R}'\sigma \vdash P_1\sigma$. Observe that $\mathcal{S} * \widetilde{w} = \emptyset$ and $\mathcal{R}' \Downarrow_{\widetilde{x}} = \mathcal{R}'$, i.e., $\mathcal{R}'\sigma = \mathcal{R}' = \mathcal{R}' \Downarrow_{\widetilde{x}} + \emptyset = \mathcal{R} + \mathcal{S} * \widetilde{w}$. Therefore it holds that $\mathcal{R} + \mathcal{S} * \widetilde{w} \vdash P'$.

   (b) If $\mathcal{S} \neq \emptyset$, then $n(\mathcal{R}') \subseteq \widetilde{x}$ by definition and $\mathcal{S} * \widetilde{x} = \mathcal{R}'$ by Lemma 5.17. By hypothesis $\mathcal{S} * \widetilde{w}$ is a partial order, so $\mathcal{R}'\sigma$ is well defined since $\mathcal{R}'\sigma = (\mathcal{S} * \widetilde{x})\sigma = \mathcal{S} * \widetilde{w}$. By Lemma C.14 we have $\mathcal{R}'\sigma \vdash P_1\sigma$. The conclusion is straightforward by noting that $\mathcal{R} + \mathcal{S} * \widetilde{w} = \mathcal{R}' \Downarrow_{\widetilde{x}} + \mathcal{R}'\sigma = \emptyset + \mathcal{R}'\sigma = \mathcal{R}'\sigma$.

2. Rule com1 We have $P = P_1 \mid P_2, P_1 \xrightarrow{(\nu\widetilde{b})\bar{a}\widetilde{w}} P_1', P_2 \xrightarrow{a\widetilde{w}} P_2', \widetilde{b} \cap fn(P_2) = \emptyset$ and $P' = (\nu\widetilde{b})(P_1' \mid P_2')$. From $\mathcal{R} \vdash P$ we derive that $\mathcal{R}_1 \vdash P_1$, $\mathcal{R}_2 \vdash P_2$ and $\mathcal{R} = \mathcal{R}_1 + \mathcal{R}_2$. By induction hypothesis on the transition of P_1 we have the following results: (1) $a : \sharp^n_{\mathcal{S}}\widetilde{V}$ and $\widetilde{w} : \widetilde{V}$; (2) $\mathcal{R}_1' \vdash P_1'$; (3) $\mathcal{R}_1 = (\mathcal{R}_1' + \mathcal{S} * \widetilde{w}) \Downarrow_{\widetilde{b}}$. By inductive assumption on the transition of P_2 we infer that $\mathcal{R}_2 + \mathcal{S} * \widetilde{w} \vdash P_2'$. Using T-par it follows that $\mathcal{R}_2 + \mathcal{R}_1' + \mathcal{S} * \widetilde{w} \vdash P_1' \mid P_2'$. Using T-res we have that $(\mathcal{R}_2 + \mathcal{R}_1' + \mathcal{S} * \widetilde{w}) \Downarrow_{\widetilde{b}} \vdash (\nu\widetilde{b})(P_1' \mid P_2')$. By the condition $\widetilde{b} \cap fn(P_2) = \emptyset$ and Lemma C.13, $\widetilde{b} \cap n(\mathcal{R}_2) = \emptyset$ holds. By using Lemma C.8 we have that $(\mathcal{R}_2 + \mathcal{R}_1' + \mathcal{S} * \widetilde{w}) \Downarrow_{\widetilde{b}} = \mathcal{R}_2 + (\mathcal{R}_1' + \mathcal{S} * \widetilde{w}) \Downarrow_{\widetilde{b}} = \mathcal{R}_2 + \mathcal{R}_1 = \mathcal{R}$. Therefore $\mathcal{R} \vdash P'$ is valid.

3. Rule rep Suppose $P = !\kappa.P_1$ with $\kappa = a(\widetilde{x}).\kappa'$. Let $\sigma = \{\widetilde{w}/\widetilde{x}\}$. After the transition P changes into $P' = P \mid (\kappa'.P_1)\sigma$. From $\mathcal{R} \vdash !\kappa.P_1$ we have $\mathcal{R} \vdash \kappa.P_1$ according to the typing rule T-rep. Applying the arguments in Case 1 to $\kappa.P_1$ we have the results: (1) $a : \sharp^n_{\mathcal{S}}\widetilde{V}$ and $\widetilde{w} : \widetilde{V}$; (2) if $\mathcal{S} * \widetilde{w}$ is a partial order then $\mathcal{R} + \mathcal{S} * \widetilde{w} \vdash (\kappa'.P_1)\sigma$. Using T-par we can infer that $\mathcal{R} + \mathcal{S} * \widetilde{w} + \mathcal{R} \vdash P'$, i.e., $\mathcal{R} + \mathcal{S} * \widetilde{w} \vdash P'$.

4. Rule open Let $P = \nu c P_1$. The transition $P \xrightarrow{(\nu \widetilde{b}, c)\bar{a}\widetilde{w}} P'$ comes from $P_1 \xrightarrow{(\nu \widetilde{b})\bar{a}\widetilde{w}} P'$ with $c \in fn(\widetilde{w}) - \{\widetilde{b}, a\}$. From $\mathcal{R} \vdash P$ we have that $\mathcal{R}' \vdash P_1$ and $\mathcal{R} = \mathcal{R}' \Downarrow_c$. By induction hypothesis on the transition of P_1 we have the following results: (1) $a : \sharp^n_{\mathcal{S}} \widetilde{V}$ and $\widetilde{w} : \widetilde{V}$; (2) $\mathcal{R}'' \vdash P'$ (3) $\mathcal{R}' = (\mathcal{R}'' + \mathcal{S} * \widetilde{w}) \Downarrow_{\widetilde{b}}$. Therefore $\mathcal{R} = \mathcal{R}' \Downarrow_c = ((\mathcal{R}'' + \mathcal{S} * \widetilde{w}) \Downarrow_{\widetilde{b}}) \Downarrow_c = (\mathcal{R}'' + \mathcal{S} * \widetilde{w}) \Downarrow_{\{\widetilde{b}, c\}}$. Now all conditions required for P are satisfied and thus we complete this case.

5. Rule if-t Let $P =$ `if` *true* `then` P_1 `else` P_2 and $P' = P_1$. From $\mathcal{R} \vdash P$ we have that $\mathcal{R}_1 \vdash P_1$, $\mathcal{R}_2 \vdash P_2$ and $\mathcal{R} = \mathcal{R}_1 + \mathcal{R}_2$. By setting $\mathcal{R}' = \mathcal{R}_1$ and $\mathcal{R}'' = \mathcal{R}_2$ the conclusion is obvious. The symmetric rule if-f is similar.

6. Rule par1 and res Followed from induction hypothesis. □

Let $\mathcal{R} \vdash P$. If P appears underneath an input prefix as in $a(\widetilde{x}).P$, then either all names in $n(\mathcal{R})$ are shielded by the prefix or none of them is bound. In other words, $\widetilde{x}$ cannot include only a portion of names in $n(\mathcal{R})$. This observation is made explicit by the following lemma, where we write $\exists! i ...$ to mean that there exists a *unique* i satisfying the succeeding condition. Usually if name a is given type $\sharp^n_{\mathcal{S}} \widetilde{V}$ we say that the partial order of a is $\mathcal{S}$, written as $po(a) = \mathcal{S}$.

Lemma C.15 *Suppose $\mathcal{R}_0 \vdash P$ and $\mathcal{R} \vdash \kappa.P$ with $\kappa = a_1(\widetilde{x}_1).\cdots.a_n(\widetilde{x}_n)$ and $n \geq 1$. Then one of the following two cases holds.*

1. $\mathcal{R}_\kappa = \emptyset$
2. $\exists! i \leq n, \mathcal{R}_\kappa = po(a_i) * \widetilde{x}_i$

Proof: We prove a stronger proposition: when the conditions in the above hypothesis are met, then one of the following two cases holds:

1. $\forall i \leq n, po(a_i) = \emptyset \wedge n(\mathcal{R}_0) \cap \widetilde{x}_i = \emptyset \wedge \mathcal{R} = \mathcal{R}_0$.
2. $\exists! i \leq n, po(a_i) = \mathcal{S} \neq \emptyset \wedge n(\mathcal{R}_0) \subseteq \widetilde{x}_i \wedge \mathcal{R}_0 = \mathcal{S} * \widetilde{x}_i \wedge (\forall j \neq i, po(a_j) = \emptyset \wedge n(\mathcal{R}_0) \cap \widetilde{x}_j = \emptyset) \wedge \mathcal{R} = \emptyset$.

By induction on the length of κ. Since $\kappa.P$ is well-typed, the sub-process $a_n(\widetilde{x}_n).P$ must be well-typed as well. Let $\mathcal{R}_1 \vdash a_n(\widetilde{x}_n).P$. Then $\mathcal{R}_1 = \mathcal{R}_0 \Downarrow_{\widetilde{x}_n}$, $a_n : \sharp^m_{\mathcal{S}} \widetilde{V}$, $\widetilde{x}_n : \widetilde{V}$ and $\mathcal{S} = \mathcal{R}_0 / \widetilde{x}_n$. Let $\kappa' = a_1(x_1).\cdots.a_{n-1}(\widetilde{x}_{n-1})$.

1. If $\mathcal{R}_0 = \emptyset$ then $\mathcal{S} = \emptyset$, i.e., $po(a_n) = \emptyset$. We also have $\mathcal{R}_1 = \mathcal{R}_0 = \emptyset$. Now take $a(\widetilde{x}_n).P$ as P and κ' as κ, we can do similar reasoning to show that $po(a_{n-1}) = \emptyset$ and $\mathcal{R}_2 = \mathcal{R}_1 = \emptyset$ if $\mathcal{R}_2 \vdash a_{n-1}(\widetilde{x}_{n-1}).a_n(\widetilde{x}_n).P$. Repeat the game until a_1, it can be shown at last that $\forall i \leq n, po(a_i) = \emptyset \wedge \mathcal{R} = \mathcal{R}_0$.
2. If $\mathcal{R}_0 \neq \emptyset$ there are two possibilities.
   (a) $n(\mathcal{R}_0) \subseteq \widetilde{x}_n$. In this case we have $\mathcal{S} \neq \emptyset$ but $\mathcal{R}_0 \Downarrow_{\widetilde{x}_n} = \emptyset$ and $\mathcal{R}_0 = \mathcal{S} * \widetilde{x}_n$. So it holds that $po(a_n) \neq \emptyset$ and $\mathcal{R}_1 = \emptyset$. By the arguments of Case 1, it is easy to see that $\forall j \leq n-1, po(a_j) = \emptyset \wedge \mathcal{R}_j = \mathcal{R}_1 = \emptyset$. Since we assume that bound names are different from each other, $n(\mathcal{R}_0) \cap \widetilde{x}_j = \emptyset$ holds.

(b) $n(\mathcal{R}_0) \cap \widetilde{x}_n = \emptyset$. In this case $\mathcal{S} = \emptyset$ and $\mathcal{R}_1 = \mathcal{R}_0$. By induction hypothesis on $\mathcal{R} \vdash \kappa'.a_n(\widetilde{x}_n).P$, we have the following results: (1) either $\forall i \leq n-1, po(a_i) = \emptyset \wedge n(\mathcal{R}_0) \cap \widetilde{x}_i = \emptyset \wedge \mathcal{R} = \mathcal{R}_0$ (2) or $\exists! i \leq n-1, po(a_i) = \mathcal{S}' \neq \emptyset \wedge n(\mathcal{R}_0) \subseteq \widetilde{x}_i \wedge \mathcal{R}_0 = \mathcal{S}' * \widetilde{x}_i \wedge (\forall j \neq i, po(a_j) = \emptyset \wedge n(\mathcal{R}_0) \cap \widetilde{x}_j = \emptyset \wedge \mathcal{R} = \emptyset)$. The conclusion follows immediately.

□

Proof of Lemma 5.21

By the transition rule rep, each time a replicated process is invoked a fresh tag is produced. So there is no replicated process invoked in P_i for $1 \leq i \leq n-1$. Then there are two possibilities:

1. No replicated process is invoked in P either. Therefore all communications on a_i, with $1 \leq i \leq n$, take place between non-replicated inputs and outputs. By similar analysis in Lemma 5.12, one can derive that
$$wt(P) \succ wt(P_1) \succ \cdots \succ wt(P')$$

2. A replicated process $!\kappa.Q$, with $\kappa = a_1(\widetilde{x}_1).\cdots.a_n(\widetilde{x}_n)$, is invoked in P and a new process $(a_2^{(l,2)}(\widetilde{x}_2).\cdots.a_n^{(l,n)}(\widetilde{x}_n).Q)\sigma$ is spawned. The subsequent reductions consume the input prefixes from $a_2^{(l,2)}\sigma(\widetilde{x}_2)$ to $a_n^{(l,n)}\sigma(\widetilde{x}_n)$ and their corresponding outputs. Then we have the relation
$$wt(P') + wt(\kappa) = wt(P) + wt(Q\sigma')$$
Note that substitution of names does not affect the weight of a process, so $wt(Q\sigma') = wt(Q)$. According to the side condition of rule T-rep there are two cases:

   (a) $wt(\kappa) \succ wt(Q)$. It follows that $wt(P) \succ wt(P')$.

   (b) $wt(\kappa) = wt(Q)$, $\kappa\ \mathcal{R}_\kappa\ Q$ and $a_n : \iota_r$. First, observe that P must be of the following form in order to have the reduction sequence.
$$P = !a_1(\widetilde{x}_1).\cdots.a_n(\widetilde{x}_n).Q \mid \bar{b}_1\widetilde{w}_1 \mid \cdots \mid \bar{b}_n\widetilde{w}_n.R_1 \mid R_2$$
   with $a_1 = b_1$ and $b_{i+1} = a_{i+1}\sigma_1\cdots\sigma_i$ for $i \geq 1$ by letting $\sigma_i = \{\widetilde{w}_i/\widetilde{x}_i\}$. Let $\sigma = \sigma_1\cdots\sigma_n$. According to our bound name convention that bound names are different from each other, $\widetilde{x}_i \cap \widetilde{x}_j = \emptyset$ if $i \neq j$. If follows that $b_i = a_i\sigma$ for all $i \geq 1$. Hence we have the result that $mos_{\mathcal{R}}(\kappa\sigma) = mos_{\mathcal{R}}(\bar{b}_1\widetilde{w}_1 \mid \cdots \mid \bar{b}_n\widetilde{w}_n)$. We also have P' in the form:
$$P' = !a_1(\widetilde{x}_1).\cdots.a_n(\widetilde{x}_n).Q \mid Q\sigma \mid R_1 \mid R_2$$
   Let $P_1 = !a_1(\widetilde{x}_1).\cdots.a_n(\widetilde{x}_n).Q$, $P_2 = \bar{b}_1\widetilde{w}_1 \mid \cdots \mid \bar{b}_n\widetilde{w}_n.R_1$ and $P_2' = Q\sigma \mid R_1$. From $\mathcal{R} \vdash P$ we have the results that $\mathcal{R}_1 \vdash P_1$, $\mathcal{R}_2 \vdash P_2$ and $\mathcal{R}_3 \vdash R$ with $\mathcal{R} = \mathcal{R}_1 + \mathcal{R}_2 + \mathcal{R}_3$. Let $\mathcal{R}_{21} = \Sigma_{i=1}^n po(b_i) * \widetilde{w}_i$ and $\mathcal{R}_{22} \vdash R_1$. Then $\mathcal{R}_2 = \mathcal{R}_{21} + \mathcal{R}_{22}$. Note that $\mathcal{R}_1 \vdash \kappa.Q$ is valid and by Lemma C.15 there are two possibilities:

      i. $\mathcal{R}_\kappa = \emptyset$

      ii. $\exists! i \leq n, \mathcal{R}_\kappa = po(a_i) * \widetilde{x}_i$

From the condition $\kappa \ \widehat{\mathcal{R}_\kappa}\ Q$ we know that $\mathcal{R}_\kappa \neq \emptyset$, so the second possibility is true. It follows that $\mathcal{R}_{21} = po(b_i) * \widetilde{w}_i = \mathcal{R}_\kappa \sigma_i = \mathcal{R}_\kappa \sigma$ by bound name convention. Hence we have the following inference sequence

$$\begin{array}{lrll}
 & \kappa & \widehat{\mathcal{R}_\kappa} & Q & \\
\Rightarrow & \kappa\sigma & \widehat{\mathcal{R}_\kappa \sigma} & Q\sigma & \text{by Lemma C.12(3)} \\
\Rightarrow & \kappa\sigma & \widehat{\mathcal{R}_{21}} & Q\sigma & \mathcal{R}_\kappa\sigma = \mathcal{R}_{21} \\
\Rightarrow & (\bar{b_1}\widetilde{w}_1 \mid \cdots \mid \bar{b_n}\widetilde{w}_n) & \widehat{\mathcal{R}_{21}} & Q\sigma & \text{by Lemma C.12(4)} \\
\Rightarrow & (\bar{b_1}\widetilde{w}_1 \mid \cdots \mid \bar{b_n}\widetilde{w}_n) \mid R_1 & \widehat{\mathcal{R}_{21}} & Q\sigma \mid R_1 & \text{by Lemma C.12(2)} \\
\Rightarrow & P_2 & \widehat{\mathcal{R}_{21}} & P_2' & \text{by Lemma C.12(4)} \\
\Rightarrow & P_1 \mid P_2 \mid R_2 & \widehat{\mathcal{R}_{21}} & P_1 \mid P_2' \mid R_2 & \text{by Lemma C.12(2)} \\
\Rightarrow & P & \widehat{\mathcal{R}} & P' & \text{by Lemma C.12(1)}
\end{array}$$

Since $a_n : \iota_r$ we have that $ur(Q) = \emptyset$, thus $ur(Q\sigma) = \emptyset$ and no unguarded restriction is liberated by the reduction sequence. Note that b_n and a_n are of the same type, hence of the same sort, which means that $ur(R_1) = \emptyset$. Theorefore P' has no unguarded restrictions either. □

Proof of Lemma 5.22

Suppose that there exists an infinite reduction sequence like

$$P_0 \stackrel{\tau^{l_1}}{\Longrightarrow} P_1 \stackrel{\epsilon'}{\longrightarrow} P_2 \stackrel{\tau^{l_2}}{\Longrightarrow} \cdots \stackrel{\epsilon'}{\longrightarrow} P_{i-1} \stackrel{\tau^{l}}{\Longrightarrow} P_i \cdots \tag{C.1}$$

then there must be infinitely many transitions $\stackrel{\tau^{l_j}}{\Longrightarrow}$ because the transition $\stackrel{\epsilon'}{\longrightarrow}$ decreases the size of processes. Let $P_0 = \nu\widetilde{a}Q_0$, without unguarded restrictions in Q_0, i.e., $ur(Q_0) = \emptyset$. Suppose $\mathcal{R} \vdash P_0$, then Q_0 is also well-typed, say $\mathcal{R}_0 \vdash Q_0$ for some $\mathcal{R}_0$. There is a corresponding reduction sequence starting from Q_0:

$$Q_0 \stackrel{\tau^{l_1}}{\Longrightarrow} Q_1 \stackrel{\epsilon'}{\longrightarrow} Q_2 \stackrel{\tau^{l_2}}{\Longrightarrow} \cdots \stackrel{\epsilon'}{\longrightarrow} Q_{i-1} \stackrel{\tau^{l}}{\Longrightarrow} Q_i \cdots$$

By Lemma 5.21 and transition rules if-t and if-f we know that no unguarded restriction is created in the sequence, thus $\forall j \leq i, P_j = \nu\widetilde{a}Q_j$ and $wt(P_j) = wt(Q_j)$. From Lemma 5.21 and Subject Reduction Theorem we have that all Q_j are well-typed, noted as $\mathcal{R}_j \vdash Q_j$, and

- if $Q_j \stackrel{\tau^{l_n}}{\Longrightarrow} Q_{j+1}$ then $\mathcal{R}_j = \mathcal{R}_{j+1}$ and $Q_j \ \widehat{\mathcal{R}_j}\ Q_{j+1}$
- if $Q_j \stackrel{\epsilon'}{\longrightarrow} Q_{j+1}$ then $\mathcal{R}_j = \mathcal{R}_{j+1} + \mathcal{R}'_{j+1}$ for some $\mathcal{R}'_{j+1}$.

If follows that $\forall j \leq i, \mathcal{R} = \mathcal{R}_j + \mathcal{R}''_j$ for some $\mathcal{R}''_j$ and by Lemma C.12(1) if $Q_j \ \widehat{\mathcal{R}_j}\ Q_{j+1}$ then $Q_j \ \widehat{\mathcal{R}}\ Q_{j+1}$. Let $\mathcal{M}^j = mos_{\mathcal{R}}(Q_j)$. It can be derived that

- if $Q_j \stackrel{\tau^{l_n}}{\Longrightarrow} Q_{j+1}$ then $\mathcal{M}^j_{\mathcal{R}}\ \mathcal{R}_{mul}\ \mathcal{M}^{j+1}_{\mathcal{R}}$ by Lemma C.10.
- if $Q_j \stackrel{\epsilon'}{\longrightarrow} Q_{j+1}$ then $\mathcal{M}^j_{\mathcal{R}}\ \mathcal{R}^{=}_{mul}\ \mathcal{M}^{j+1}_{\mathcal{R}}$ by rules if-t and if-f

where the notation $\mathcal{M}\ \mathcal{R}^{=}_{mul}\ \mathcal{M}'$ means $\mathcal{M}\ \mathcal{R}_{mul}\ \mathcal{M}'$ or $\mathcal{M} = \mathcal{M}'$. Since there are infinitely many transitions $\stackrel{\tau^{l_j}}{\Longrightarrow}$ in (C.1), there are infinitely many $\mathcal{R}_{mul}$ in the sequence

$$\mathcal{M}^0_{\mathcal{R}}\ \mathcal{R}_{mul}\ \mathcal{M}^1_{\mathcal{R}}\ \mathcal{R}^{=}_{mul}\ \mathcal{M}^2_{\mathcal{R}}\ \mathcal{R}_{mul} \cdots$$

which contradicts the well-foundedness of $\mathcal{R}_{mul}$.

Consequently, by means of commuting reductions used in Lemma 5.14, we can always find a Q with $wt(P_0) \succ wt(Q)$ in finite number of steps. □

$$
\begin{array}{lcll}
P & ::= & \mathsf{0} \;\big|\; x\langle y\rangle \;\big|\; \texttt{def}\ D\ \texttt{in}\ P \;\big|\; P \mid P' & \text{processes} \\
D & ::= & \mathsf{T} \;\big|\; J \triangleright P \;\big|\; D \wedge D' & \text{definitions} \\
J & ::= & x\langle y\rangle \;\big|\; J \mid J' & \text{join-patterns}
\end{array}
$$

$$
\begin{array}{rcll}
\vdash P_1 \mid P_2 & \rightleftharpoons & \vdash P_1, P_2 & \textsf{Str-join} \\
\vdash \mathsf{0} & \rightleftharpoons & \vdash & \textsf{Str-null} \\
D_1 \wedge D_2 \vdash & \rightleftharpoons & D_1, D_2 \vdash & \textsf{Str-and} \\
\mathsf{T} \vdash & \rightleftharpoons & \vdash & \textsf{Str-nodef} \\
\vdash \texttt{def}\ D\ \texttt{in}\ P & \rightleftharpoons & D\sigma \vdash P\sigma & \textsf{Str-def} \\
\\
J \triangleright P \vdash J\sigma & \longrightarrow & J \triangleright P \vdash P\sigma & \textsf{Red}
\end{array}
$$

Table C.1: Syntax and semantics of the Join-calculus

C.6 Levels in the Join-calculus

The idea of introducing level information into type system so as to enforce termination is also applicable in other process calculi. In this section, we investigate termination of processes in the Join-calculus [Fou98] by taking advantage of levels as we did in Section 5.2. We recall the syntax and semantics of the Join-calculus in Table C.1. Detailed description about the calculus can be found in [Fou98].

For ease of understanding, we consider the monadic Join-calculus. The extension to allow polyadic communication is straightforward. We preserve all notations of [Fou98] for the syntax and semantics, but add two multisets $mdv[J]$ and $mdv[P]$ which are defined below.

$$
\begin{array}{rcl}
mdv[x\langle y\rangle] & \stackrel{\text{def}}{=} & [x] \\
mdv[J \mid J'] & \stackrel{\text{def}}{=} & mdv[J] \uplus mdv[J'] \\
mdv[\texttt{def}\ D\ \texttt{in}\ P] & \stackrel{\text{def}}{=} & mdv[P] \\
mdv[P \mid Q] & \stackrel{\text{def}}{=} & mdv[P] \uplus mdv[Q] \\
mdv[\mathsf{0}] & \stackrel{\text{def}}{=} & [\,]
\end{array}
$$

The reason of using multisets instead of the set $\mathsf{dv}[J]$ given in [Fou98] comes from the mechanism of inter-process synchronisation of the Join-calculus: pattern-matching. Consider the following two processes:

$$
\begin{array}{rcl}
Q & \stackrel{\text{def}}{=} & \texttt{def}\ x\langle\rangle \triangleright x\langle\rangle\ \texttt{in}\ x\langle\rangle \\
Q' & \stackrel{\text{def}}{=} & \texttt{def}\ x\langle\rangle \mid x\langle\rangle \triangleright x\langle\rangle\ \texttt{in}\ x\langle\rangle
\end{array}
$$

Obviously Q' is terminating while Q is not. Without multiset, we would not be able to distinguish Q' from Q and wrongly take both of them as illegal processes. For the type system, we assume that the only primitive type is `unit` and we do not consider polymorphism. Hence the concepts of type scheme and simple type environment in [Fou98] coincide with type and typing environment respectively. Due to these simplification our type system becomes less complicated than the original

$$\textsf{T-message}\ \frac{\Gamma \vdash x : \sharp^n V \quad \Gamma \vdash y : V}{\Gamma \vdash x\langle y\rangle} \qquad \textsf{T-par}\ \frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}$$

$$\textsf{T-def}\ \frac{\Gamma_1, \Gamma_2 \vdash D :: \Gamma_2 \quad \Gamma_1, \Gamma_2 \vdash P}{\Gamma_1 \vdash \texttt{def}\ D\ \texttt{in}\ P} \qquad \textsf{T-null}\ \frac{}{\Gamma \vdash \mathsf{0}}$$

$$\textsf{T-rule}\ \frac{\Gamma, \widetilde{y} : \widetilde{V} \vdash \prod_{i\in 1..n} x_i\langle y_i\rangle \quad \Gamma, \widetilde{y} : \widetilde{V} \vdash P \quad lv(\widetilde{x}) >_{mul} lv(mdv[P])}{\Gamma \vdash \prod_{i\in 1..n} x_i\langle y_i\rangle \triangleright P :: \Gamma \downarrow_{\widetilde{x}}}$$

$$\textsf{T-and}\ \frac{\Gamma \vdash D_1 :: \Gamma_1 \quad \Gamma \vdash D_2 :: \Gamma_2}{\Gamma \vdash D_1 \wedge D_2 :: \Gamma_1 \oplus \Gamma_2} \qquad \textsf{T-nodef}\ \frac{}{\Gamma \vdash \mathsf{T} :: \Gamma'}$$

$$\textsf{T-soup}\ \frac{\forall P \in \mathcal{P}, \Gamma \vdash P \quad \forall D \in \mathcal{D}, \Gamma \vdash D}{\Gamma \vdash \mathcal{D} \vdash \mathcal{P}} \qquad \textsf{T-multi}\ \frac{\Gamma \vdash D :: \Gamma'}{\Gamma \vdash D}$$

Figure C.1: Typing rules for the join calculus

one presented in [Fou98]. The syntax of types is the same as that of π-calculus studied in Section 5.2. Given a set of names N, the restriction of type environment Γ on N, written $\Gamma \downarrow_N$, is a new type environment which only binds names belonging to N. Let $N = \{x_1, \cdots, x_n\}$, we define $lv(N) = \{lv(x_1), \cdots, lv(x_n)\}$ as the multiset of levels for names in N. The typing rules are reported in Figure C.1, where $\prod_{i\in 1..n} P_i$ represents the parallel composition $P_1 \mid \cdots \mid P_n$ and $>_{mul}$ is the multiset ordering between two multisets of natural numbers.

The rule T-rule requires the condition $lv(mdv[J]) >_{mul} lv(mdv[P])$ in order to make $J \triangleright P$ typable. It means that some output channels in J are replaced by finite number of lower level channels in P. According to the semantics of the join calculus, the only effective reduction relation is

$$J \triangleright P \vdash J\sigma \longrightarrow J\triangleright \vdash P\sigma.$$

Since the substitution σ does not affect level information, as a whole the chemical soup will loose some level information after the reduction step. This phenomenon is reflected in the decrement of our measure, weight, which is now defined on both processes and soups.

$$\begin{array}{rclcrcl} wt(\mathsf{0}) & = & \mathbf{0} & & wt(x\langle y\rangle) & = & \mathbf{0}_i \ \text{ if } lv(x) = i \\ wt(P \mid Q) & = & wt(P) + wt(Q) & & wt(\texttt{def}\ D\ \texttt{in}\ P) & = & wt(P) \\ wt(J \mid J') & = & wt(J) + wt(J') & & & & \\ wt(\mathcal{D} \vdash \mathcal{P}) & = & \sum_{i\in 1..n} wt(P_i) \ \text{ if } \mathcal{P} = \{P_i \mid 1 \le i \le n\} & & & & \end{array}$$

As usual, the proofs of weakening and substitution lemmas are quite easy. The proof of subject reduction theorem is simpler than that in [Fou98] because no type variable is involved. Details are omitted.

Lemma C.16 *If $\Gamma \vdash J \triangleright P$ then $wt(J) \succ wt(P)$.*

Proof: By definitions it holds that $lv(mdv[J]) >_{mul} lv(mdv[P])$ iff $wt(J) \succ wt(P)$. □

Theorem C.17 *If $\mathcal{D} \vdash \mathcal{P}$ is a well-typed chemical soup, there is no infinite reduction sequence starting from the soup.*

Proof: We need to prove three claims.

1. Claim 1: If $D_1 \vdash P_1 \rightleftharpoons D_2 \vdash P_2$ then $wt(D_1 \vdash P_1) = wt(D_2 \vdash P_2)$. It is trivial by examining all structural rules.

2. Claim 2: If $D_1 \vdash P_1 \longrightarrow D_2 \vdash P_2$ then $wt(D_1 \vdash P_1) \succ wt(D_2 \vdash P_2)$. The only reduction rule is $J \triangleright P \vdash J\sigma \longrightarrow J \triangleright P \vdash P\sigma$. Following from Lemma C.16, it holds that $wt(J\sigma) = wt(J) \succ wt(P) = wt(P\sigma)$, thus $wt(D_1 \vdash P_1) \succ wt(D_2 \vdash P_2)$.

3. Claim 3: If $\mathcal{D}_1 \vdash \mathcal{P}_1 \rightleftharpoons^* \longrightarrow \rightleftharpoons^* \mathcal{D}_2 \vdash \mathcal{P}_2$ then $wt(\mathcal{D}_1 \vdash \mathcal{P}_1) \succ wt(\mathcal{D}_2 \vdash \mathcal{P}_2)$. This is easy by using the first two claims.

The required result follows from Claim 3. □

Bibliography

[AB01] Suzana Andova and J. C. M. Baeten. Abstraction in probabilistic process algebra. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2031 of *Lecture Notes in Computer Science*, pages 204–219. Springer, 2001.

[Aba99] Martin Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 46(5):749–786, 1999.

[AÉI02] Luca Aceto, Zoltán Ésik, and Anna Ingólfsdóttir. Equational axioms for probabilistic bisimilarity (preliminary report). Technical Report RS-02-6, BRICS, 2002.

[And99] Suzana Andova. Process algebra with probabilistic choice. Technical Report CSR 99-12, Eindhoven University of Technology, 1999.

[Bar84] Henk Barendregt. *The lambda Calculus: Its Syntax and Semantics*. North-Holland, 1984.

[BB05] Jos C. M. Baeten and Mario Bravetti. A ground-complete axiomatization of finite state processes in process algebra. In *Proceedings of the 16th International Conference on Concurrency Theory*, Lecture Notes in Computer Science. Springer, 2005. To appear.

[BBS95] Jos C. M. Baeten, Jan A. Bergstra, and Scott A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. *Information and Computation*, 121(2):234–255, 1995.

[BD95] Michele Boreale and Rocco De Nicola. Testing equivalences for mobile processes. *Information and Computation*, 120:279–303, 1995.

[Bec80] Frank S. Beckman. *Mathematical Foundations of Programming*. Addison-Wesley, 1980.

[Bez03] Marc Bezem. Mathematical background. In *Term Rewriting Systems*, pages 790–825. Cambridge University Press, 2003.

[BGW01] Nikita Borisov, Ian Goldberg, and David Wagner. Intercepting mobile communications: The insecurity of 802.11. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pages 180–189. ACM Press, 2001.

[BH97] Christel Baier and Holger Hermanns. Weak bisimulation for fully probabilistic processes. In *Proceedings of the 9th International Conference on Computer Aided Verification*, volume 1254 of *Lecture Notes in Computer Science*, pages 119–130. Springer, 1997.

[BK84] Jan A. Bergstra and Jan Willem Klop. Process algebra for synchronous communication. *Information and Computation*, 60:109–137, 1984.

[Bou92] Gérard Boudol. Asynchrony and the π-calculus (note). Technical Report RR-1702, INRIA Sophia-Antipolis, 1992.

[Bou03] Gérard Boudol. On strong normalization in the intersection type discipline. In *Proceedings of the 6th International Conference on Typed Lambda Calculi and Applications*, volume 2701 of *Lecture Notes in Computer Science*, pages 60–74. Springer, 2003.

[BS98] Michele Boreale and Davide Sangiorgi. Bisimulation in name-passing calculi without matching. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, pages 165–175. IEEE, Computer Society Press, 1998.

[BS01] Emanuele Bandini and Roberto Segala. Axiomatizations for probabilistic bisimulation. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 370–381. Springer, 2001.

[BW90] Jos C. M. Baeten and W. P. Weijland. *Process Algebra*, volume 18 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1990.

[CG00] Luca Cardelli and Andrew D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.

[Cod70] E.F. Codd. A relational model for large shared databanks. *Communications of the ACM*, 13(6):377–387, 1970.

[CS02] Stefano Cattani and Roberto Segala. Decision algorithms for probabilistic bisimulation. In *Proceedings of the 13th International Conference on Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*, pages 371–385. Springer, 2002.

[DCPP05] Yuxin Deng, Tom Chothia, Catuscia Palamidessi, and Jun Pang. Metrics for action-labelled quantitative transition systems. In *Proceedings of the 3rd Workshop on Quantitative Aspects of Programming Languages*, Electronic Notes in Theoretical Computer Science, 2005. To appear.

[DFP98] Rocco De Nicola, Gian Luigi Ferrari, and Rosario Pugliese. Kaim: a kernel language for agents interaction and mobility. *IEEE Transactions on Software Engineering*, 24(5):315–330, 1998.

[DH95] Nachum Dershowitz and Charles Hoot. Natural termination. *Theoretical Computer Science*, 142(2):179–207, 1995.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science*, chapter 6, pages 243–320. North-Holland, 1990.

[DJGP02] Josee Desharnais, Radha Jagadeesan, Vineet Gupta, and Prakash Panangaden. The metric analogue of weak bisimulation for probabilistic processes. In *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*, pages 413–422. IEEE Computer Society, 2002.

[DJGP04] Josee Desharnais, Radha Jagadeesan, Vineet Gupta, and Prakash Panangaden. Metrics for labelled markov processes. *Theoretical Computer Science*, 318(3):323–354, 2004.

[DM79] Nachum Dershowitz and Zohar Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

[DP05] Yuxin Deng and Catuscia Palamidessi. Axiomatizations for probabilistic finite-state behaviors. In *Proceedings of the 8th International Conference on Foundations of Software Science and Computation Structures*, volume 3441 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2005.

[DPP05] Yuxin Deng, Catuscia Palamidessi, and Jun Pang. Compositional reasoning for probabilistic finite-state behaviors, 2005. Submitted. A draft version is available at `http://www.pps.jussieu.fr/~yuxin/publications/par.ps`.

[DS04a] Yuxin Deng and Davide Sangiorgi. Ensuring termination by typability. In *Proceedings of the 3rd IFIP International Conference on Theoretical Computer Science*, pages 619–632. Kluwer, 2004.

[DS04b] Yuxin Deng and Davide Sangiorgi. Towards an algebraic theory of typed mobile processes. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming*, volume 3142 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 2004.

[DS05] Yuxin Deng and Davide Sangiorgi. Towards an algebraic theory of typed mobile processes. *Theoretical Computer Science*, 2005. To appear.

[Fou98] Cédric Fournet. *The Join-Calculus: A Calculus for Distributed Mobile Programming*. PhD thesis, École Polytechnique, Paris, France, 1998.

[Fu99] Yuxi Fu. Variations on mobile processes. *Theoretical Computer Science*, 221(1–2):327–368, 1999.

[FY03] Yuxi Fu and Zhenrong Yang. Tau laws for pi calculus. *Theoretical Computer Science*, 308:55–130, 2003.

[Gan80] Robin O. Gandy. Proofs of strong normalization. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980.

[GJS90] Alessandro Giacalone, Chi-Chang Jou, and Scott A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of IFIP TC2 Working Conference on Programming Concepts and Methods*, 1990.

[HJ90] Hans Hansson and Bengt Jonsson. A calculus for communicating systems with time and probabilities. In *Proceedings of IEEE Real-Time Systems Symposium*, pages 278–287. IEEE Computer Society Press, 1990.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[HP04] Oltea M. Herescu and Catuscia Palamidessi. Probabilistic asynchronous pi-calculus. Technical report, INRIA Futurs and LIX, 2004.

[HR02a] Matthew Hennessy and James Riely. Information flow vs. resource access in the asynchronous pi-calculus. *ACM Transactions on Programming Languages and Systems*, 24(5):566–591, 2002.

[HR02b] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, 173(1):82–120, 2002.

[HR04] Matthew Hennessy and Julian Rathke. Typed behavioural equivalences for processes in the presence of subtyping. *Mathematical Structures in Computer Science*, 14:651–684, 2004.

[HT91] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *Proceedings of the 5th European Conference on Object-Oriented Programming*, volume 512 of *Lecture Notes in Computer Science*, pages 133–147. Springer, 1991.

[HVY00] Kohei Honda, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Secure information flow as typed process behaviour. In *Proceedings of the 9th European Symposium on Programming Languages and Systems*, volume 1782 of *Lecture Notes in Computer Science*, pages 180–199. Springer, 2000.

[HY05] Kohei Honda and Nobuko Yoshida. Noninterference through flow analysis. *Journal of Functional Programming*, 2005. To appear.

[Jon93] Cliff B. Jones. A π-calculus semantics for an object-based design notation. In *Proceedings of the 4th International Conference on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 158–172. Springer, 1993.

[Kob98] Naoki Kobayashi. A partially deadlock-free typed process calculus. *ACM Transactions on Programming Languages and Systems*, 20(2):436–482, 1998.

[Kob00] Naoki Kobayashi. Type systems for concurrent processes: From deadlock-freedom to livelock-freedom, time-boundedness. In *Proceedings of the 1st IFIP International Conference on Theoretical Computer Science*, volume 1872 of *Lecture Notes in Computer Science*, pages 365–389. Springer, 2000.

[KPT99] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, 1999.

[Lin94] Huimin Lin. Symbolic bisimulation and proof systems for the π-calculus. Technical Report 7/94, School of Cognitive and Computing Sciences, University of Sussex, 1994.

[Lin03] Huimin Lin. Complete inference systems for weak bisimulation equivalences in the π-calculus. *Information and Computation*, 180(1):1–29, 2003.

[LL03] A. Laurie and B. Laurie. Serious flaws in bluetooth security lead to disclosure of personal data, 2003. `http://bluestumbler.org`.

[Loa98] Ralph Loader. Notes on simply typed lambda calculus. Technical Report 381, LFCS, University of Edinburgh, 1998.

[Low91] Gavin Lowe. *Probabilities and Priorities in Time CSP*. PhD thesis, Oxford, 1991.

[LS91] Kim G. Larsen and Aren Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[LS00] Francesca Levi and Davide Sangiorgi. Controlling interference in ambients. In *Proceedings of the 27th ACM symposium on Principles of Programming Languages*, pages 352–364. ACM Press, 2000.

[Mer00] Massimo Merro. *Locality in the π-calculus and applications to distributed objects*. PhD thesis, Ecole des Mines de Paris, France, 2000.

[Mil78] Robin Milner. Synthesis of communicating behaviour. In *Proceedings of the 7th Symposium on Mathematical Foundations of Computer Science*, volume 64 of *Lecture Notes in Computer Science*, pages 71–83. Springer, 1978.

[Mil80] Robin Milner. A calculus of communicating systems. volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.

[Mil84] Robin Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Science*, 28:439–466, 1984.

[Mil89a] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil89b] Robin Milner. A complete axiomatisation for observational congruence of finite-state behaviours. *Information and Computation*, 81:227–247, 1989.

[Mil91] Robin Milner. The polyadic π-calculus: A tutorial. Technical Report ECS-LFCS-91-180, Department of Computer Science, University of Edingburgh, 1991.

[Mil92] Robin Milner. Functions as processes. *Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Mil99] Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[MPW92] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, part I/II. *Information and Computation*, 100:1–77, 1992.

[Nes00] Uwe Nestmann. What is a 'good' encoding of guarded choice? *Information and Computation*, 156:287–319, 2000.

[Pal03] Catuscia Palamidessi. Comparing the expressive power of the synchronous and the asynchronous pi-calculus. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[Par01] Joachim Parrow. An introduction to the pi-calculus. In *Handbook of Process Algebra*, pages 479–543. Elsevier, 2001.

[PH04] Catuscia Palamidessi and Oltea M. Herescu. A randomized encoding of the π-calculus with mixed choice. Technical report, INRIA Futurs and LIX, 2004.

[Plo81] Gordon Plotkin. A structural approach operational semantics. Technical Report DAIMI-FN-19, Computer Science Department, Aarhus University, 1981.

[PLS00] Anna Philippou, Insup Lee, and Oleg Sokolsky. Weak bisimulation for probabilistic systems. In *Proceedings of the 11th International Conference on Concurrency Theory*, volume 1877 of *Lecture Notes in Computer Science*, pages 334–349. Springer, 2000.

[PS95] Joachim Parrow and Davide Sangiorgi. Algebraic theories for name-passing calculi. *Information and Computation*, 120(2):174–197, 1995.

[PS96] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–454, 1996.

[PV98] Joachim Parrow and Björn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science*, pages 176–185. IEEE, Computer Society Press, 1998.

[RG02] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, 2002.

[San93] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, Department of Computer Science, University of Edingburgh, 1993.

[San96a] Davide Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167:235–274, 1996.

[San96b] Davide Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33:69–97, 1996.

[San99] Davide Sangiorgi. The typed π-calculus at work: A proof of jones's parallelisation transformation on concurrent objects. *Theory and Practice of Object-Oriented Systems*, 5(1):25–33, 1999.

[San05] Davide Sangiorgi. Termination of processes. *Mathematical Structures in Computer Science*, 2005. To appear.

[SdV04] Ana Sokolova and Erik P. de Vink. Probabilistic automata: system types, parallel composition and comparison. In *Validation of Stochastic Systems: A Guide to Current Research*, volume 2925 of *Lecture Notes in Computer Science*, pages 1–43. Springer, 2004.

[Seg95] Roberto Segala. Modeling and verification of randomized distributed real-time systems. Technical Report MIT/LCS/TR-676, PhD thesis, MIT, Dept. of EECS, 1995.

[SL94] Roberto Segala and Nancy A. Lynch. Probabilistic simulations for probabilistic processes. In *Proceedings of the 5th International Conference on Concurrency Theory*, volume 836 of *Lecture Notes in Computer Science*, pages 481–496. Springer, 1994.

[SM92] Davide Sangiorgi and Robin Milner. The problem of "weak bisimulation up-to". In *Proceedings of the 3th International Conference on Concurrency Theory*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1992.

[SS00] Eugene W. Stark and Scott A. Smolka. A complete axiom system for finite-state probabilistic processes. In *Proof, language, and interaction: essays in honour of Robin Milner*, pages 571–595. MIT Press, 2000.

[Sta79] Richard Statman. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9(1):73–81, 1979.

[Sto02] Mariëlle Stoelinga. *Alea jacta est: verification of probabilistic, real-time and parametric systems*. PhD thesis, University of Nijmegen, 2002.

[SW01] Davide Sangiorgi and David Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[Tho95] Bent Thomsen. A theory of higher order communicating systems. *Information and Computation*, 116(1):38–57, 1995.

[Tof94] Chris Tofts. Processes with probabilities, priority and time. *Formal Aspects of Computing*, 6(5):536–564, 1994.

[vBW01] Franck van Breugel and James Worrell. An algorithm for quantitative verification of probabilistic transition systems. In *Proceedings of the 12th International Conference on Concurrency Theory*, volume 2154 of *Lecture Notes in Computer Science*, pages 336–350. Springer, 2001.

[vBW04] Franck van Breugel and James Worrell. A behavioural pseudometric for probabilistic transition systems. *Theoretical Computer Science*, 2004. In press.

[vdP01] Jaco van de Pol. A prover for the mucrl toolset with applications – version 0.1. Technical Report SEN-R0106, CWI, Amsterdam, 2001.

[vGSS95] Rob J. van Glabbeek, Scott A. Smolka, and Bernhard Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121(1):59–80, 1995.

[VH93] Vasco Thudichum Vasconcelos and Kohei Honda. Principal typing schemes in a polyadic π-calculus. In *Proceedings of the 4th International Conference on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 524–538. Springer, 1993.

[Wal95] David Walker. Objects in the π-calculus. *Information and Computation*, 116(2):253–271, 1995.

[YBH04] Nobuko Yoshida, Martin Berger, and Kohei Honda. Strong normalisation in the pi-calculus. *Information and Computation*, 191(2):145–202, 2004.

[YL92] Wang Yi and Kim Larsen. Testing probabilistic and nondeterministic processes. In *Proceedings of the 12th IFIP International Symposium on Protocol Specification, Testing and Verification*, pages 47–61. North Holland, 1992.

[Zan03] Hans Zantema. Termination. In *Term Rewriting Systems*, pages 181–259. Cambridge University Press, 2003.