Université Paris 7 — Denis Diderot
UFR d'Informatique

THÈSE

pour l'obtention du Diplôme de
Docteur de l'Université Paris 7
Spécialité Informatique

présentée et soutenue publiquement par

Francesco Zappa Nardelli

le 12 décembre 2003.

De la sémantique des processus d'ordre supérieur

Directeur de thèse :
Giuseppe Castagna

Jury :

MM.	Pierre-Louis Curien	Président
	Giuseppe Castagna	
	Matthew Hennessy	
	Jean-Jacques Lévy	
	Glynn Winskel	

Rapporteurs :

MM.	Matthew Hennessy
	Davide Sangiorgi

A big thank you to Giuseppe Castagna for his guidance during these years, to Matthew Hennessy and Davide Sangiorgi for reviewing my thesis, and to all the members of the *jury*.
A big thank you to Giorgio Ghelli.
A big thank you to Massimo Merro and Glynn Winskel for their friendship.
A big thank you to Jan Vitek, Gergana Markova, Chrislain Razafimahefa, Michel Pawlak.
A big thank you to the persons I shared an house with: Davide Malacaria, Alessandro Urpi, Pio Nardiello in Pisa, without forgetting Alessandro Casanovi; Cecilia Soderlund, Noemi Jacobs, and Shams in the most international flat I lived in; Marie Coris and Anne-Sophie Peyne in Brighton.
A big thank you to Amy Clemitshaw and Sophie Maltby for the long walks on Brighton seafront.
A big thank you to two great bars: chez Paco (*de profundis*) in Paris, and the Full Moon in Brighton — thank you, Paul Stones, and all the poets.
A big thank you to an old Fiat 500 (*de profundis*), to Ernesto Mistretta, to Carlos Muñoz Camacho.
A big thank you to my parents, Giuliana 'mamma' Nardelli and Paolo 'babbo' Zappa. A big thank you to my friends from Perugia: Daniele 'Monta' Montagnoli, Miriam Menichelli, Marco 'Basco' Bastianelli, Lorenzo 'Merio' Mariani, Marco 'Borgo' Borghesi, Giacomo Scorsipa, Stefania Cruciani.
A big thank you to the persons I shared an office with: Juliusz Chroboczek, Gabriele Santini, Samuel Hym, Alain Frisch, Jim Laird. And of course Frédéric De Jaeger and Cédric Lhouissane.
A big thank you to Eric Goubault, Vladimiro Sassone, Giuseppe Longo, Matteo Coccia, Marco Carbone, Sophie Lepine, Tom Hirschowitz, Marie Duflot, Olivier Glass, Marine Picon, Julian Rathke and especially to Katarzina 'Katy' Wozniak.
And the biggest thank you to Annick 'Piccina' Grandemange.

Un grand merci à Giuseppe Castagna pour avoir dirigé mes études, à Matthew Hennessy et Davide Sangiorgi pour avoir accepté d'être rapporteurs, à tous les membres du jury.
Un grand merci à Giorgio Ghelli.
Un grand merci à Massimo Merro et à Glynn Winskel pour leur amitié.
Un grand merci à Jan Vitek, Gergana Markova, Chrislain Razafimahefa, Michel Pawlak.
Un grand merci aux amis avec lesquels j'ai partagé une maison : Davide Malacaria, Alessandro Urpi, Pio Nardiello à Pise, sans oublier Alessandro Casanovi ; Cecilia Soderlund, Noemi Jacobs, et Shams dans l'appartement le plus international au monde ; Marie Coris et Anne-Sophie Peyne à Brighton.
Un grand merci à Amy Clemitshaw et Sophie Maltby pour les longues promenade sur la plage de Brighton.
Un grand merci à deux grands bars : chez Paco (*de profundis*) à Paris, et le *Full Moon* à Brighton — un grand merci à Paul Stones et à tous les poètes.
Un grand merci à une vieille Fiat 500 (*de profundis*), à Ernesto Mistretta et à Carlos Muñoz Camacho.
Un grand merci à mes parents, Giuliana 'Mamma' Nardelli et Paolo 'Babbo' Zappa. Un grand merci à mes amis de Pérouse : Daniele 'Monta' Montagnoli, Miriam Menichelli, Marco 'Basco' Bastianelli, Lorenzo 'Merio' Mariani, Marco 'Borgo' Borghesi, Giacomo Scorsipa, Stefania Cruciani.
Un grand merci aux personnes avec lesquelles j'ai partagé un bureau : Juliusz Chroboczek, Gabriele Santini, Samuel Hym, Alain Frisch, Jim Laird. Et bien sûr Frédéric De Jaeger et Cédric Lhouissane.
Un grand merci à Eric Goubault, Vladimiro Sassone, Giuseppe Longo, Matteo Coccia, Marco Carbone, Sophie Lepine, Tom Hirschowitz, Marie Duflot, Olivier Glass, Marine Picon, Julian Rathke et surtout à Katarzina 'Katy' Wozniak.
Et le merci le plus grand est pour Annick 'Piccina' Grandemange.

To *The man who killed Don Quijote.*

À *The man who killed Don Quijote.*

Contents

1 Résumé

Ce premier chapitre présente de manière exhaustive la démarche et les résultats des travaux de la présente thèse. Il ne saurait toutefois constituer qu'un aperçu, et nous renvoyons, pour de plus amples détails, explications et exemples, et pour toutes les preuves, aux chapitres suivants (2 à 6), rédigés en anglais.

Les théories du calcul séquentiel sont essentiellement des théories des fonctions calculables : un programme séquentiel transforme un ensemble de valeurs d'entrée en un ensemble de valeurs de sortie, en exécutant une séquence d'actions mécaniques. L'approche dénotationnelle de Scott et Strachey, par le cadre mathématique général du calcul séquentiel qu'elle offre, s'est révélée fondamentale, car elle a ainsi permis, notamment, la comparaison des langages de programmation, la mise en relation du calcul séquentiel et des domaines mathématiques que sont l'algèbre, la topologie et la logique, et la mise au point de nouveaux systèmes de typage, des nouvelles constructions de programmation, et de nouvelles méthodes d'analyse des programmes.

Cette approche fonctionnelle du calcul demeure cependant insuffisante pour analyser les systèmes concurrents. En effet, dans un système concurrent, des agents actifs influencent mutuellement leur action, par des interactions constantes et variées : l'intérêt de tels systèmes réside dans cette interaction, dans le flot de données, et pas dans la relation entrée-sortie. Ainsi, dans la mesure où, dans un tel contexte, les analyses de la concurrence ne bénéficient pas de la mise au point d'un cadre mathématique global analogue à celui de Scott et Strachey pour le calcul séquentiel, elles prennent d'ordinaire pour objet une notion jugée fondamentale, autour de laquelle un modèle est construit, puis testé.

Parmi ces notions, l'existence de « noms », a été au centre de nombreuses recherches, car celle-ci est inséparable du processus de communication, fondamental dans tout système concurrent. L'existence de noms suggère de plus un espace abstrait de processus connectés, dans lequel les noms représentent les connections ; seuls les processus qui partagent des noms sont alors en mesure d'interagir. La structure d'un système change donc de manière dynamique, car les liens entre processus sont sans cesse créés et détruits.

Si cette mobilité des liens est bien connue et analysée, il n'en va pas de même de la mobilité des processus au sein de l'espace abstrait de processus connectés. En effet, les possibilités de modélisation des processus mobiles sont multiples, tandis que les techniques habituellement utilisées en concurrence sont d'application difficile pour l'analyse des modèles obtenus. Ainsi, non seulement les modèles abondent, mais leur compréhension respective demeure extrêmement limitée. Leurs théories ne donnent ainsi des processus mobiles qu'une image fragmentée et fragmentaire.

Cette thèse se propose ainsi d'étudier les théories à la base de la mobilité, dans le but de développer des outils mathématiques de formalisation et de preuve des propriétés des systèmes mobiles. Nous nous concentrerons sur des « bisimulations étiquetées », afin de mettre au point des méthodes de preuve puissante permettant l'investigation des théories

comportementales de ces systèmes. En parallèle, nous donnerons une lecture opérationnelle d'un modèle dénotationnel de processus non-déterministes ; cela nous conduira à la définition d'un langage concis mais expressif, à même de représenter une grande variété de langages de processus. L'unité de ce travail réside dans la réflexion menée sur le concept de comportement, qui se propose de définir ce que l'on entend lorsque l'on pose que deux processus ont le même comportement.

Méthodes de preuve basées sur la bisimulation

On n'a défini la sémantique d'un langage qu'une fois que l'on est en mesure de clarifier la question : « ces deux termes sont-ils équivalents ? ». La définition de l'équivalence doit en effet répondre à une exigence pragmatique : on veut, lorsque l'on considère deux termes comme équivalents, pouvoir les remplacer l'un par l'autre dans un programme, sans que l'exécution du programme en soit visiblement affectée. En d'autres termes, une des propriétés fondamentales requises est celle de la contextualité de l'équivalence.

Il n'est pas surprenant que l'une des premières approches de l'étude de la sémantique de la concurrence ait été construite autour d'une caractérisation des interactions possibles entre les termes du langage et les environnements possibles. En effet, en CCS, les interactions qu'un terme peut avoir avec son environnement sont décrites efficacement par un système de transitions étiquetées (abrégé STE). Ainsi, $P \xrightarrow{\alpha} Q$ sera lu : « le processus P peut interagir avec un contexte avec l'action α, et continuer ensuite comme Q ». L'étiquette a représente alors une interaction possible avec un contexte $C_a[-] = - \mid \overline{a}.\mathbf{0}$. À partir de la caractérisation des interactions avec l'environnement, on peut définir une équivalence dans le spectre temps linéaire—temps arborescent. Dans cette thèse, nous concentrerons sur la bisimulation, une équivalence qui demande que deux processus équivalents puissent se simuler réciproquement dans les interactions qu'ils ont avec l'environnement. Formellement, on dit qu'une relation $\mathcal{R}$ entre processus est une bisimulation si $P \mathcal{R} Q$ implique que

- si $P \xrightarrow{\alpha} P'$ alors il existe un processus Q' tel que $Q \xrightarrow{\alpha} Q'$ et $P' \mathcal{R} Q'$;
- si $Q \xrightarrow{\alpha} Q'$ alors il existe un processus P' tel que $P \xrightarrow{\alpha} P'$ et $P' \mathcal{R} Q'$.

On dit que deux processus P et Q sont bisimilaires s'il existe une bisimulation $\mathcal{R}$ telle que $P \mathcal{R} Q$. Apparaît alors clairement le rôle centrale joué par le STE, qui doit incorporer dans sa définition une connaissance approfondie des interactions entre termes et environnements. Or, en présence de processus d'ordre supérieur, définir un STE approprié n'est pas toujours simple. Pour donner un aperçu des difficultés, il suffit de considérer une extension à l'ordre supérieur du π-calcul, où un processus P peut être envoyé sur un canal a avec la syntaxe $\overline{a}\langle P\rangle.Q$, et reçu et activé avec la construction $a(X).Q$. Il semble logique d'étendre l'un des STE standards du π-calcul avec des transitions de la forme $\overline{a}\langle P\rangle.Q \xrightarrow{\overline{a}\langle P\rangle} Q$, et de considérer ensuite la bisimulation qui en dérive naturellement. Cette approche se révèle désastreuse : l'équivalence que l'on obtient ne respecte même pas des lois basiques comme la symétrie de l'opérateur de composition parallèle (les processus $\overline{a}\langle P \mid Q\rangle.\mathbf{0}$ et $\overline{a}\langle Q \mid P\rangle.\mathbf{0}$ sont distingués, parce que les étiquettes $\overline{a}\langle P \mid Q\rangle$ et $\overline{a}\langle Q \mid P\rangle$ sont syntaxiquement différentes). Des définitions alternatives de bisimulations, nommées « bisimulations d'ordre supérieur », ont été proposées [AGR88, Bou89, Tho90], mais comme Sangiorgi l'a montré [San94] (voir

aussi page 48 et suivantes), elles ne sont pas adaptées à un calcul d'ordre supérieur avec portée de noms statique. Ces difficultés sont liées au fait qu'un contexte ne peut pas observer directement les processus transmis, mais doit d'abord les réactiver et interagir avec eux avant d'observer une différence entre les deux processus de départ.

Ces difficultés ont eu pour conséquence que les modèles pour le calcul mobile ont été souvent présentés comme des langages de programmation, plutôt que comme des calculs de processus. Cela signifie que, une fois leur syntaxe définie, leur modèle d'exécution est spécifié sous la forme d'un ensemble de règles pour une machine abstraite (très souvent il s'agit de la Machine Abstraite Chimique [BB92]). Il faut remarquer que même si cela donne une relation de réduction, notée $\rightarrowtriangle$, qui décrit comment exécuter un programme, cette approche ne donne directement aucune information sur le comportement des termes dans un contexte. La présentation du langage de programmation est complétée par la définition d'une notion d'équivalence : on identifie une *observation* basique, notée $P \downarrow \alpha$ (lire : on peut observer α dans le programme P), et on définit deux programmes comme équivalents si, à l'intérieur de tout contexte $C[-]$, ils donnent lieu aux mêmes observations. Ou bien, si l'on s'intéresse à des équivalences dites branching-time, on peut utiliser la *congruence barbue*, ou l'un de ses dérivés. Nous nous intéresserons à la *congruence barbue fermée par réduction* (abrégée CBR et notée $\cong$), définie comme la plus grande relation symétrique entre processus telle que, si $P \cong Q$, alors les faits suivants sont vérifiés :

- si $P \rightarrowtriangle P'$ alors il existe un processus Q' tel que $Q \rightarrowtriangle^* Q'$ et $P' \cong Q'$;
- $P \downarrow \alpha$ implique $Q \Downarrow \alpha$;
- pour tout contexte $C[-]$, on a $C[P] \cong C[Q]$;

où $\rightarrowtriangle^*$ est la fermeture réflexive et transitive de $\rightarrowtriangle$ et $P \Downarrow \alpha$ si $P \rightarrowtriangle^* P'$ et $P' \downarrow \alpha$. On souligne que l'équivalence CBR est une équivalence dite *faible*, qui ne s'intéresse pas aux réductions internes du programme mais uniquement aux interactions qu'il a avec l'environnement. En effet, dans un cadre distribué, seules nous intéressent les actions visibles des processus.

Il semble que cette présentation mène à une notion d'équivalence qui réponde à nos attentes initiales, mais à bien voir ce développement a uniquement soulevé une autre question, bien plus difficile à traiter que la première : « comment peut-on démontrer que deux termes sont équivalents ? ». En effet la contextualité de la relation CBR dépend d'une quantification universelle sur tous les contextes possibles, qui rend son application extrêmement difficile.

La mise au point des *bisimulations étiquetées* qui impliquent, ou mieux, qui coïncident avec la CBR, reste un outil fondamental pour l'étude de la théorie comportementale d'un langage de processus. Bien plus, leur développement permet d'obtenir des informations très détaillées sur le comportement des termes.

Programmer avec des localisations explicites Nous nous intéressons à deux langages de processus qui représentent explicitement l'existence d'objets spatiaux, nommés localisations, qui abstraient les lieux physiques, comme les ordinateurs, les réseaux, ou les logiques, comme les domaines de protections et les agents. Pour leur permettre de remplir tous ces rôles, les localisations sont nommées et structurées selon une hiérarchie en arbre. Une localisation k sera représentée par $k[\,+\,]$, où l'on a noté avec $+$ son contenu. Situés à l'intérieur des localisations, les processus exécutent des actions et sont les responsables de l'évolution globale du système. La hiérarchie des localisations peut être modifiée par l'activité des pro-

cessus : on parle de mobilité *forte*, pour dire qu'un processus peut à tout moment interrompre son activité et se déplacer (ou être interrompu et être déplacé) dans une autre localisation où l'activité reprend.

Les deux calculs que nous étudierons, le Seal Calcul et les Ambients Mobiles, présentent deux modèles de mobilité opposés. Dans les deux cas, les localisations sont l'unité du mouvement, et une localisation se déplace avec tout son contenu. À titre d'exemple, on considère une localisation n qui entre dans une localisation m, ce qui n'est possible que si m est frère de n dans l'arborescence :

$$k[n[\,-\,] \mid m[\,+\,]] \quad \rightarrowtriangle \quad k[m[n[\,-\,] \mid +]]$$

En Seal Calcul une localisation, nommée *seal*, est déplacée par les processus dans son environnement et n'a aucun contrôle sur la migration. L'exemple ci-dessus peut être implanté avec deux processus P et Q qui se synchronisent pour déplacer le seal nommé n :

$$k[P \mid n[\,-\,] \mid m[Q \mid +]] \quad \rightarrowtriangle \quad k[m[n[\,-\,] \mid +]]$$

Au contraire, dans les Ambients Mobiles l'instruction de migration est déclenchée par les processus localisés dans la localisation même. Donc, pour implanter notre exemple, on utilisera un seul processus localisé à l'intérieur de l'ambient n :

$$k[n[P \mid -] \mid m[\,+\,]] \quad \rightarrowtriangle \quad k[m[n[\,-\,] \mid +]]$$

Si l'on considère que la hiérarchie des localisations forme une arborescence, et que les communications entre localisations lointaines doivent être explicitement programmées, alors l'étude de ces deux langages, qui se révéleront profondément différents, est complémentaire.

Le Seal Calcul

Le Seal Calcul est un calcul de processus mobiles apte à modéliser une programmation sécurisée sur des systèmes distribués. Il a été défini par Castagna et Vitek [VC99] comme la contrepartie formelle d'une bibliothèque nommée JavaSeal [BV01], extension de la machine virtuelle JAVA destinée à la programmation des applications basées sur des agents mobiles dans des domaines qui requièrent un degré élevé de sécurité. Par conséquent, sa définition incluait des constructions indispensables dans l'implantation réelle JavaSeal, mais qui compliquent inutilement l'étude théorique du Seal calcul. Des versions simplifiées du Seal Calcul ont alors été proposées, et à l'heure actuelle le nom « Seal Calcul » désigne toute une famille de calculs de processus, offrant les mêmes abstractions mais présentant des différences subtiles dans leur sémantique.

Il est cependant possible de donner une présentation formelle de cette famille de langages qui fasse abstraction du dialecte particulier : cela nous permettra de mettre en relief les profondes différences entre les dialectes. Nous commencerons par introduire la syntaxe et la sémantique par réduction des Seal Calculi, quitte à en expliquer les intuitions calculatrices juste après.

La syntaxe du Seal Calcul est présentée figure 1.1. Parmi les processus, le processus inactif $\mathbf{0}$, la composition parallèle $P \mid Q$, la création de noms $(\boldsymbol{\nu} x)P$, le préfixe $\alpha.P$, la duplication

Processus :			*Actions :*				*Localisations :*			
P	::=	$\mathbf{0}$	α	::=	$\overline{x}^{\eta}(y_1,\ldots,y_n)$	sortie	η	::=	$*$	ici
	$\mid$	$P \mid P$		$\mid$	$x^{\eta}(y_1,\ldots,y_n)$	entrée		$\mid$	$\uparrow$	haut
	$\mid$	$!\alpha.P$		$\mid$	$\overline{x}^{\eta}\{y\}$	envoi		$\mid$	z	bas
	$\mid$	$(\boldsymbol{\nu} x)P$		$\mid$	$x^{\eta}\{y_1,\cdots,y_n\}$	réception				
	$\mid$	$\alpha.P$		pour $n \geq 0$						
	$\mid$	$x[P]$	*Noms :* $a,\ldots,u,v,x,y,\cdots \in \mathbf{N}$							

Fig. 1.1: La syntaxe du Seal Calcul

(a) Canaux localisés (b) Canaux partagés

Fig. 1.2: Interprétations des canaux dans le Seal Calcul

$!\alpha.P$, et les localisations $n[P]$: on remarquera qu'il s'agit d'une extension de la syntaxe du π-calcul avec l'ajout de localisations hiérarchiques, et qu'il s'agit de la même syntaxe que les processus des Ambients Mobiles. Ce qui est propre au modèle de calcul de Seal, ce sont les actions α que les processus peuvent exécuter. Dans le Seal Calcul, toute interaction a lieu sur des canaux nommés et localisés. Une caractéristique du modèle de calcul est que la distribution spatiale des processus limite leurs interactions : deux processus ne peuvent interagir qu'en utilisant un canal qui leur soit « suffisamment proche ». Nous proposons deux significations des termes « suffisamment proches », qui définissent deux dialectes différents du Seal : on a affaire à des canaux dits *localisés* lorsque ceux-ci sont associés aux seals, et à des canaux dits *partagés* lorsqu'ils sont partagés entre deux seals en relation père-fils dans la hiérarchie. La figure 1.2 illustre les deux interprétations. En figure 1.2(a), on a représenté une configuration dans l'interprétation localisée. Les canaux x et y se trouvent à l'intérieur de deux seals différents, et les processus P et Q peuvent se synchroniser sur indifféremment l'un ou l'autre de ces canaux. Pour le processus P, le canal x est local et il sera noté x^*, quand pour Q il est localisé dans le seal parent, et il sera noté $x^{\uparrow}$. Le canal y sera noté y^* par Q, et y^n par P. En figure 3.1(b), on a représenté une configuration similaire dans l'interprétation partagée : maintenant, seul le canal x peut être utilisé pour synchroniser P et Q, et il sera noté respectivement x^n et $x^{\uparrow}$. Le processus P ne peut pas accéder au canal y.

Les canaux sont utilisés soit pour la communication (échange de noms), soit pour la mobilité (échange de seals). Les actions qui permettent la communication sont $\overline{x}^{\eta}(y_1,\ldots,y_n)$ et $x^{\eta}(y_1,\ldots,y_n)$. La première désigne un processus qui veut envoyer les noms $\vec{y}$ sur le canal x^{η}, puis continuer son exécution P ; la seconde désigne un processus qui veut recevoir des valeurs sur le canal x^{η}, puis reprendre son exécution Q, où les occurrences libres des $\vec{z}$ ont

été remplacées par les valeurs reçues $\vec{y}$. Les actions qui permettent la mobilité sont $\overline{x}^{\eta}\{\!|\, y \,|\!\}$ et $x^{\eta}\{\!|\, y_1, \ldots, y_n \,|\!\}$. La première désigne un processus qui veut envoyer un seal nommé y, localisé dans la même localisation que lui, sur le canal x^{η} ; la seconde désigne un processus qui veut recevoir un seal sur le canal x^{η} et en réactiver n copies à l'intérieur des seals qu'il va créer et nommer $\vec{z_i}$. Il en suit que l'une des applications de la mobilité réside dans la possibilité de copier des seals, comme le montre le terme

$$(\boldsymbol{\nu}\, y)\,(\,\overline{y}^{*}\{\!|\, x \,|\!\} \;\mid\; y^{*}\{\!|\, x\,, z \,|\!\}\,) \mid x[P] \quad \text{qui évolue en} \quad x[P] \mid y[P]\ .$$

La sémantique par réduction est définie en style chimique à l'aide d'une relation de réduction et d'une relation auxiliaire nommée congruence structurelle, qui identifie les termes syntaxiquement équivalents. La sémantique par réduction est paramétrée sur l'interprétation des canaux. Pour cela, nous introduisons deux prédicats,

$$\mathsf{synch}^{S}, \mathsf{synch}^{L} : \mathbf{Var} \times \mathbf{Loc} \times \mathbf{Loc} \to \mathbf{Bool}$$

les deux abrégés par synch.

On suppose que la notion de *noms libres* d'un terme P, notée $\mathrm{fn}(P)$, est connue (les lieurs sont la restriction et le préfixe d'entrée).

On définit la *congruence structurelle* comme la plus petite relation entre processus qui est préservée par les contextes statiques et qui vérifie les axiomes suivants :

$$\begin{array}{rcl@{\qquad}rcll} P \mid \mathbf{0} & \equiv & P & (\boldsymbol{\nu}x)(\boldsymbol{\nu}y)P & \equiv & (\boldsymbol{\nu}y)(\boldsymbol{\nu}x)P & x \neq y \\ P \mid Q & \equiv & Q \mid P & (\boldsymbol{\nu}x)(P \mid Q) & \equiv & P \mid (\boldsymbol{\nu}x)Q & x \notin \mathrm{fn}(P) \\ P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & (\boldsymbol{\nu}x)\mathbf{0} & \equiv & \mathbf{0} & \\ !\alpha.P & \equiv & \alpha.P \mid !\alpha.P & & & & \end{array}$$

Il est intéressant de remarquer que la possibilité de copier les seals invalide la règle structurelle

$$(\boldsymbol{\nu}x)y[P] \quad \equiv \quad y[(\boldsymbol{\nu}x)P] \qquad x \neq y$$

typique des Ambients. En effet, les processus $(\boldsymbol{\nu}x)y[P]$ et $y[(\boldsymbol{\nu}x)P]$ ne sont pas équivalents : si on considère un contexte $C[-] = - \mid (\boldsymbol{\nu}\, y)\,(\,\overline{y}^{*}\{\!|\, n \,|\!\} \;\mid\; y^{*}\{\!|\, n\,, m \,|\!\}\,)$ on obtient

$$C[n[(\boldsymbol{\nu}x)P]] \;\twoheadrightarrow\; n[(\boldsymbol{\nu}x)P] \mid m[(\boldsymbol{\nu}x)P] \quad \text{et} \quad C[(\boldsymbol{\nu}x)n[P]] \;\twoheadrightarrow\; (\boldsymbol{\nu}x)(n[P] \mid m[P])\ .$$

Le premier processus donne lieu à une configuration dans laquelle les seals n et m ont chacun un canal privé x, tandis que le second processus donne lieu à une configuration dans laquelle n et m partagent un nom commun x.

La sémantique par réduction est définie en style chimique. On appellera *contexte statique* un contexte de processus où le trou ne figure pas sous un préfixe ou une duplication. La sémantique par réduction du calcul est donc définie par la *relation de réduction*, $\twoheadrightarrow$, qui est la relation la plus petite entre processus, fermée par contextes statiques et satisfaisant les règles reportées figure 1.3.

On peut maintenant souligner un autre critère qui permet de différencier les dialectes des Seals : la portée des noms est statique dans le Seal Calcul et, comme dans le π-calcul, la portée d'un nom peut être étendue après une interaction. Ce phénomène est essentiel à

$$
\begin{array}{lll}
\text{(write local)} & x^*(\vec{u}).P \mid \overline{x}^*(\vec{v}).Q & \rightarrowtriangle\ P[\vec{v}/\vec{u}] \mid Q \\
\text{(write in)} & \overline{x}^{\eta_1}(\vec{w}).P \mid y[(\boldsymbol{\nu}\vec{z})(x^{\eta_2}(\vec{u}).Q_1 \mid Q_2)] & \rightarrowtriangle\ P \mid y[(\boldsymbol{\nu}\vec{z})(Q_1[\vec{w}/\vec{u}] \mid Q_2)] \\
\text{(write out)} & x^{\eta_1}(\vec{u}).P \mid y[(\boldsymbol{\nu}\vec{z})(\overline{x}^{\eta_2}(\vec{v}).Q_1 \mid Q_2)] & \\
 & \qquad \rightarrowtriangle\ (\boldsymbol{\nu}\vec{v} \cap \vec{z})(P[\vec{v}/\vec{u}] \mid y[(\boldsymbol{\nu}\vec{z} \setminus \vec{v})(Q_1 \mid Q_2)]) & \\
\text{(move local)} & x^*\{\vec{u}\}.P_1 \mid \overline{x}^*\{v\}.P_2 \mid v[Q] & \rightarrowtriangle\ P_1 \mid u_1[Q] \mid \cdots \mid u_n[Q] \mid P_2 \\
\text{(move in)} & \overline{x}^{\eta_1}\{v\}.P \mid v[S] \mid y[(\boldsymbol{\nu}\vec{z})(x^{\eta_2}\{\vec{u}\}.Q_1 \mid Q_2)] & \\
 & \qquad \rightarrowtriangle\ P \mid y[(\boldsymbol{\nu}\vec{z})(Q_1 \mid Q_2 \mid u_1[S] \mid \cdots \mid u_n[S])] & \\
\text{(move out)} & x^{\eta_1}\{\vec{u}\}.P \mid y[(\boldsymbol{\nu}\vec{z})(\overline{x}^{\eta_2}\{v\}.Q_1 \mid v[R] \mid Q_2)] & \\
 & \quad \rightarrowtriangle\ P \mid (\boldsymbol{\nu}\,\mathrm{fn}(R) \cap \vec{z})(u_1[R] \mid \cdots \mid u_n[R] \mid y[(\boldsymbol{\nu}\vec{z} \setminus \mathrm{fn}(R))(Q_1 \mid Q_2)]) & \\
\text{(red struct)} & P \equiv P' \ \wedge\ P' \rightarrowtriangle Q' \ \wedge\ Q' \equiv Q \text{ implique } P \rightarrowtriangle Q &
\end{array}
$$

où $x \notin \vec{z}$, $\vec{w} \cap \vec{z} = \emptyset$, $\mathrm{fn}(S) \cap \vec{z} = \emptyset$ et $\mathsf{synch}_y(\eta_1, \eta_2)$.

Fig. 1.3: Règles de réduction pour le Seal Calcul

l'expressivité du calcul, mais il peut être préférable de le limiter quand la portée d'un nom sort des limites d'une localisation à la suite d'un mouvement. On nommera cette limitation condition-*e*. Ajouter la condition-*e* au langage revient à imposer la condition

$$\mathrm{fn}(R) \cap \vec{z} = \emptyset$$

dans la règle de réduction (move out).

On définit comme notion basique d'observation la présence au niveau supérieur d'un seal dont le nom est public : cet observable peut être interprété comme la capacité pour un contexte d'interagir avec ce seal. Formellement, nous écrirons $P \downarrow n$ si et seulement si existent Q, R, $\vec{x}$ tels que $P \equiv (\boldsymbol{\nu}\vec{x})(n[Q] \mid R)$ où $n \notin \vec{x}$. Étant fixé cet observable, nous considérons la congruence barbue fermée par réduction, notée $\cong$, comme l'équivalence naturelle.

Les dialectes de Seal, et l'équivalence comportementale La localisation des canaux et la condition-*e* sont deux caractéristiques orthogonales. On peut ainsi identifier quatre dialectes du Seal Calcul, définis comme suit :

- *L-Seal* : canaux localisés ; (presque) le Seal Calcul originel présenté dans [VC99] ;
- *eL-Seal* : canaux localisés et condition-*e* ; défini dans [CZN02] ;
- *S-Seal* : canaux partagés ; défini dans [CGZN01] ;
- *eS-Seal* : canaux partagés et condition-*e* ; défini dans [CZN02].

Grâce à la présentation uniforme des ces calculs qu'on vient de donner, il est possible de mettre en relief les différences subtiles entre ces calculs de processus apparemment similaires.

La surprise la plus grande a probablement été le fait que la condition-e donne aux contextes le pouvoir d'observer l'ensemble des noms libres d'un terme. Soient P et Q deux processus, et soit x un nom tel que $x \in \mathrm{fn}(P)$ mais $x \notin \mathrm{fn}(Q)$. Indépendamment du comportement des deux termes, le contexte

$$C[-] \;=\; z^{y}\{n\} \mid y[(\boldsymbol{\nu} x)(\overline{z}^{\uparrow}\{n\} \mid n[\,-\,])]$$

utilise le nom x pour distinguer P de Q : d'un côté, le seal $n[Q]$ peut sortir de y, de l'autre la présence du nom libre x dans P empêche le déplacement du seal n. La condition-e donne aux contextes un grand pouvoir de discrimination et il est difficile de justifier l'équivalence qu'on obtient seulement en termes de *comportement*. En effet, deux processus sont considérés comme n'étant pas équivalents dès lors qu'ils présentent des nom libres différents, y compris dans des sous-processus *dead-locked*.

Les dialectes qui présentent des canaux localisés sont caractérisés par des extrusions de noms assez subtiles, comme le montre la réduction suivante :

$$x^{*}(u).P \mid (\boldsymbol{\nu} z) z[\overline{x}^{\uparrow}(v).Q] \;\twoheadrightarrow\; (\boldsymbol{\nu} z)(P[u/v] \mid z[Q]) \;.$$

Ces comportements rendent le contrôle de l'interférence très difficile. Par exemple, les termes $(\boldsymbol{\nu} cxy)(\overline{c}\{x\} \mid c\{y\} \mid x[P])$ et $(\boldsymbol{\nu} cxy)y[P]$ ne sont pas équivalents : le processus P pourrait en effet exécuter une action $\overline{z}^{\uparrow}(c)$ qui rende le nom c connu à l'environnement, qui peut ensuite interférer dans la réduction. Comme conséquence, non seulement la théorie algébrique se révèle très pauvre, mais surtout programmer dans les dialectes localisés du Seal Calcul peut conduire à des erreurs difficilement identifiables.

Le Seal Calcul avec des canaux partagés et sans la condition-e se révèle le calcul le plus propice à une étude théorique : dans la suite nous nous concentrerons sur S-Seal, et on l'appellera tout simplement Seal. Nous chercherons des méthodes de preuve basées sur un STE, pour en apprendre plus sur son comportement et approfondir notre connaissance de sa théorie algébrique.

Un système de transitions étiquetées En figure 1.6 nous définissons un STE pour Seal. Les transitions ont la forme

$$A \vdash P \xrightarrow{\ell} P'$$

où A est un ensemble fini de noms tel que $\mathrm{fn}(P) \subseteq A$; l'interprétation intuitive est « dans un état où les noms dans A peuvent être connus par le processus P et son environnement, le processus P peut exécuter l'action ℓ et devenir P' ». Cette présentation, reprise depuis [SV99] et [Sew00], nous permet de simplifier les règles qui concernent l'extension de la porté des noms. Dans les environnements des noms, on écrit A, y pour indiquer $A \cup \{y\}$,

Les actions ℓ, définies figure 1.4, donnent une description précise de l'évolution d'un terme P. Les actions peuvent être regroupées en trois classes : les actions qui sont une conséquence directe de l'exécution d'une capacité, celles qui représentent une synchronisation partielle, et la réduction interne.

Un processus peut utiliser une capacité : les règles (IN), (OUT), (SND), et (RCV) en sont responsables. Les actions d'entrée $x^{\eta}(\vec{y})$ et de sortie $\overline{x}^{\eta}(\vec{y})$ sont analogues aux actions correspondantes dans un STE précoce de π-calcul. L'action d'envoi $\overline{x}^{\eta}\{y\}$ d'un seal exprime

Étiquettes

ℓ	::=	τ	action interne
	$\mid$	P_z	seal freeze
	$\mid$	P^z	seal chained
	$\mid$	$\gamma[a]$	activité a à γ

Activités

a	::=	$x^\eta(\vec{y})$	entrée
	$\mid$	$\overline{x}^\eta(\vec{y})$	sortie
	$\mid$	$\overline{x}^\eta\{\!\mid y \mid\!\}$	envoi
	$\mid$	$\overline{x}^\eta\{\!\mid P \mid\!\}$	capsule
	$\mid$	$x^\eta\{\!\mid P \mid\!\}$	réception
	$\mid$	x^η_z	lock

Localisations

γ	::=	$*$	ici
	$\mid$	z	dans z

Fig. 1.4: Actions

Soient

$$P \;=\; \overline{x}^\eta\{\!\mid z \mid\!\}.P' \qquad Q \;=\; x^\eta\{\!\mid \vec{y} \mid\!\}.Q' \qquad R \;=\; z[S]$$

En suivant la légende

$$(R \mid P) \mid Q \dashrightarrow \qquad (R \mid Q) \mid P \longrightarrow \qquad (P \mid Q) \mid R \cdots\!\!\rightarrow$$

le diagramme ci-dessous illustre les interactions possibles.

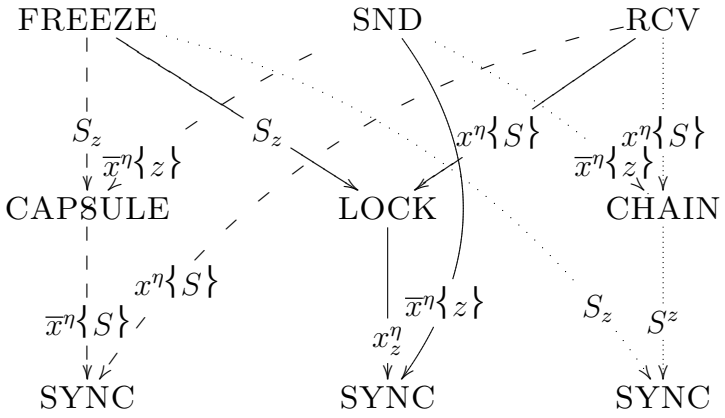


Fig. 1.5: Interactions pour la mobilité

le fait que le processus veut envoyer un seal nommé y sur le canal x^η. L'action de réception $x^\eta\{\!\mid Q \mid\!\}$ d'un seal exprime le fait que le processus est prêt à recevoir un seal dont le contenu est le processus Q sur le canal x^η. Les actions d'entrée et de réception mettent bien en relief la formulation en style précoce du STE. À ce groupe appartient aussi l'étiquette P_z : un seal peut être déplacé à tout moment par un contexte, et cela implique qu'un seal $z[P]$ doit pouvoir se signaler pour être déplacé. La règle (FREEZE) permet à un seal de se « congeler », en communicant à l'environnement son nom et son contenu, et en se transformant en processus inactif.

Le modèle de mobilité du Seal Calcul demande une synchronisation entre trois processus : le processus qui déplace le seal, celui qui le reçoit, et le seal qui est déplacé. Comme le diagramme reporté figure 1.5 le montre, le STE peut générer les τ-actions indépendamment de l'ordre dans lequel elle se synchronisent. Cela demande trois étiquettes supplémentaires : « capsule » $\overline{x}^{\eta}\{\!|P|\!\}$, « lock » x^{η}_{z}, et « seal chained » P^{z}. Ces trois étiquettes signifient que deux des trois intervenants dans une synchronisation ont déjà interagi et attendent le troisième processus pour terminer l'interaction.

L'étiquette τ modélise les réductions internes : étant données deux actions en correspondance, la règle (SYNC) construit le processus résultant de l'interaction. Les actions correspondantes sont identifiées par la relation $\curlyvee$.

Définition 3.3.1, page 72 *On définit $\curlyvee$ comme la plus petite relation symétrique entre étiquettes, contenant la relation suivante :*

$$\{\ (\gamma_1[\overline{x}^{\eta_1}(\vec{y})]\ ,\ \gamma_2[x^{\eta_2}(\vec{y})])\ \mid\ \mathcal{M}\ \}\ \cup\ \{\ (\gamma_1[\overline{x}^{\eta_1}\{\!|S|\!\}]\ ,\ \ \gamma_2[x^{\eta_2}\{\!|S|\!\}])\ \mid\ \mathcal{M}\ \}$$
$$\cup\ \{\ (\gamma_1[x^{\eta_1}_{z}]\ ,\ \gamma_2[\overline{x}^{\eta_2}\{\!|z|\!\}])\ \mid\ \mathcal{M}\ \}\ \cup\ \{\ (S_z\ ,\ S^z)\ \}$$

où $\mathcal{M} \stackrel{def}{=} (\gamma_1 = \eta_1 = \gamma_2 = \eta_2 = *) \vee (\gamma_1 = *\ \wedge\ \mathsf{synch}_{\gamma_2}(\eta_1, \eta_2)) \vee (\gamma_2 = *\ \wedge\ \mathsf{synch}_{\gamma_1}(\eta_2, \eta_1))$.

Une action qui peut être observée au niveau supérieur doit être exécutée ou bien au niveau supérieur, ou bien à l'intérieur d'une localisation qui est elle-même au niveau supérieur. Les étiquettes enregistrent l'endroit où l'action a est exécutée avec une étiquette : $*[a]$ signifie que l'action est une action exécutée au niveau supérieur, $x[a]$ signifie que l'action est exécutée à l'intérieur du seal x. La règle (SEAL LABEL) transforme une étiquette $*[a]$ en $x[a]$ sous la condition que l'action a soit observable à l'extérieur de la localisation x. Les autre règles de congruence ne nécessitent pas de commentaire.

La sémantique définie par le STE coïncide avec la sémantique par réduction :

Théorème 3.3.8, page 80 *Soit un processus P. (i) Si $\mathrm{fn}(P) \subseteq A$ et $A \vdash P \xrightarrow{\tau} Q$, alors $P \twoheadrightarrow Q$; (ii) si $P \twoheadrightarrow Q$ alors il existe $A \supseteq \mathrm{fn}(P)$ tel que $A \vdash P \xrightarrow{\tau} Q'$, où $Q' \equiv Q$.*

Une méthode de preuve Le STE qu'on vient d'introduire peut être utilisé comme base au-dessus de laquelle définir une bisimulation étiquetée. Le développement de cette notion de bisimulation non seulement nous donnera un outil pour étudier la théorie comportementale du Seal, mais surtout nous permettra de mieux approfondir certaines spécifités de son modèle de calcul.

Commençons tout d'abord par rappeler que le prédicat $P \downarrow n$ teste la capacité d'un processus P à interagir avec l'environnement en utilisant le seal n. Dans d'autres calculs de processus, comme le π-calcul, les barbes sont définies en utilisant les actions du STE. Il n'est pas difficile de prouver que la relation RBC ne change pas si l'on remplace les barbes naturelles par des barbes obtenues à partir d'une certaine classe d'actions du STE (théorème 3.4.6, page 83). Il est par contre extrêmement intéressant d'analyser quelles actions appartiennent à cette classe, et surtout lesquelles n'y figurent pas. En effet, la classe d'actions générée par la règle (Chain) n'y figure pas, parce que les étiquettes de la forme S^z peuvent, dans certains

Congruence

(PAR)
$$\frac{A \vdash P \xrightarrow{\ell} P'}{A \vdash P \mid Q \xrightarrow{\ell} P' \mid Q}$$

(RES) $u \notin \mathrm{fn}(\ell)$
$$\frac{A, u \vdash P \xrightarrow{\ell} P'}{A \vdash (\boldsymbol{\nu} u)P \xrightarrow{\ell} (\boldsymbol{\nu} u)P'}$$

(BANG)
$$\frac{A \vdash \alpha.P \xrightarrow{\ell} P'}{A \vdash !\alpha.P \xrightarrow{\ell} P \mid !\alpha.P}$$

(OPEN COM) $y, \eta, \gamma \neq u; u \in \vec{x}$
$$\frac{A, u \vdash P \xrightarrow{\gamma[\overline{y}^{\eta}(\vec{x})]} P'}{A \vdash (\boldsymbol{\nu} u)P \xrightarrow{\gamma[\overline{y}^{\eta}(\vec{x})]} P'}$$

(OPEN FREEZE) $z \notin u; u \in \mathrm{fn}(S)$
$$\frac{A, u \vdash P \xrightarrow{S_z} P'}{A \vdash (\boldsymbol{\nu} u)P \xrightarrow{S_z} P'}$$

(SEAL TAU)
$$\frac{A \vdash P \xrightarrow{\tau} P'}{A \vdash x[P] \xrightarrow{\tau} x[P']}$$

(OPEN CAPSULE) $y, \eta, \gamma \notin \vec{u}; u \in \mathrm{fn}(S)$
$$\frac{A, u \vdash P \xrightarrow{\gamma[\overline{y}^{\eta}\{S\}]} P'}{A \vdash (\boldsymbol{\nu} u)P \xrightarrow{\gamma[\overline{y}^{\eta}\{S\}]} P'}$$

(SEAL LABEL)
$$\frac{A \vdash P \xrightarrow{*[a]} P' \quad a \in \{y^{\uparrow}(\vec{z}), \overline{y}^{\uparrow}(\vec{z}), y^{\uparrow}\{Q\}, \overline{y}^{\uparrow}\{Q\}\}}{A \vdash x[P] \xrightarrow{x[a]} x[P']}$$

Communication

(OUT)
$$\frac{}{A \vdash \overline{x}^{\eta}(\vec{y}).P \xrightarrow{*[\overline{x}^{\eta}(\vec{y})]} P}$$

(IN)
$$\frac{}{A \vdash x^{\eta}(\vec{y}).P \xrightarrow{*[x^{\eta}(\vec{v})]} P[\vec{v}/\vec{y}]}$$

Mobilité

(SND)
$$\frac{}{A \vdash \overline{x}^{\eta}\{y\}.P \xrightarrow{*[\overline{x}^{\eta}\{y\}]} P}$$

(RCV)
$$\frac{}{A \vdash x^{\eta}\{\vec{y}\}.P \xrightarrow{*[x^{\eta}\{Q\}]} P \mid y_1[Q] \mid \cdots \mid y_n[Q]}$$

(CAPSULE)
$$\frac{A \vdash P \xrightarrow{S_z} P' \quad A \vdash Q \xrightarrow{*[\overline{x}^{\eta}\{z\}]} Q'}{A \vdash P \mid Q \xrightarrow{*[\overline{x}^{\eta}\{S\}]} P' \mid Q'}$$

(LOCK) $\gamma = \eta = *$ ou $(\gamma \neq * \wedge \eta = \uparrow)$
$$\frac{A \vdash P \xrightarrow{S_z} P' \quad A \vdash Q \xrightarrow{\gamma[x^{\eta}\{S\}]} Q'}{A \vdash P \mid Q \xrightarrow{\gamma[x^{\eta}_z]} (\boldsymbol{\nu}\,\mathrm{fn}(S) \setminus A)(P' \mid Q')}$$

(FREEZE)
$$\frac{}{A \vdash x[P] \xrightarrow{P_x} \mathbf{0}}$$

(CHAIN) $\gamma = \eta_1 = \eta_2 = *$ or $(\eta_1 = \gamma \wedge \eta_2 = \uparrow)$
$$\frac{A \vdash P \xrightarrow{*[\overline{x}^{\eta_1}\{y\}]} P' \quad A \vdash Q \xrightarrow{\gamma[x^{\eta_2}\{S\}]} Q'}{A \vdash P \mid Q \xrightarrow{S^y} P' \mid Q'}$$

Synchronisation

(SYNC) $\ell_1 \curlyvee \ell_2$
$$\frac{A \vdash P \xrightarrow{\ell_1} P' \qquad A \vdash Q \xrightarrow{\ell_2} Q'}{A \vdash P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}(\mathrm{fn}(\ell_1) \cup \mathrm{fn}(\ell_2)) \setminus A)(P' \mid Q')}$$

Les règles symétriques pour (PAR), (CAPSULE), (LOCK), et (CHAIN) ont été omises.

Fig. 1.6: Un système de transitions étiquetées pour Seal

cas, ne pas être observables. On reviendra sur ce point quand on parlera de la complétude de notre bisimulation.

Une seconde remarque est qu'une barbe $P \downarrow n$ correspond à l'émission d'une étiquette (FREEZE) par le processus en question : $A \vdash P \xrightarrow{R_n} P'$ pour un processus R arbitraire et un ensemble de noms $A \supseteq \mathrm{fn}(P)$. Par conséquent, deux processus P et Q peuvent émettre la même barbe naturelle $P, Q \downarrow n$, même si au niveau du STE cela correspond à deux actions *a priori* différentes, comme $A \vdash P \xrightarrow{R_n} P'$ et $A \vdash Q \xrightarrow{S_n} Q'$ où R peut être différent de S. Les actions générées par la règle (CAPSULE) se comportent de la même façon. Cela est typique du traitement de certaines actions d'ordre supérieur, qui demandent une correspondance faible entre les processus qui sont transmis.

À bien voir, la correspondance entre les barbes naturelles et les actions (FREEZE) nous montre que l'observation utilisée dans la RBC est insensible au processus R qui figure dans l'étiquette de l'action (FREEZE) correspondante. Une caractérisation coinductive de cette équivalence doit donc ne pas demander une correspondance stricte entre les étiquettes d'ordre supérieur. Nous reprenons l'approche de Sangiorgi dans sa définition de la *bisimulation contextuelle* pour HO π [San92, San96a], et nous demandons que deux processus qui exécutent des transitions d'ordre supérieur soient équivalents par rapport à toute interaction possible avec un contexte. Il y a quatre étiquettes d'ordre supérieur, $\gamma[x^\eta\{\!\!\{ S \}\!\!\}], S^z, R_z$, et $\gamma[\overline{x}^\eta\{\!\!\{ R \}\!\!\}]$, qui peuvent être classées en deux groupes selon leur comportement par rapport à des interactions avec le contexte.

Supposons que $A \vdash P \xrightarrow{\gamma[x^\eta\{\!\!\{ S \}\!\!\}]} P'$, et soit Q un processus qu'on veut montrer équivalent à P. Le processus P' est le résiduel de l'interaction entre P et un contexte $C[-]$ qui envoie un corps de seal S sur le canal x^η. Dans le jeu de la bisimulation le processus Q doit pouvoir recevoir un corps de seal arbitraire sur le canal x^η, sinon un contexte peut facilement distinguer les deux processus. En même temps, le processus S qui figure dans l'étiquette a été envoyé par le contexte : les processus P et Q n'ont aucun contrôle sur le choix de S. Par conséquent, il n'est pas restrictif de demander des processus syntaxiquement identiques dans les étiquettes. L'action S^z est analogue à la précédente, parce que le processus S est présenté par l'environnement.

Supposons maintenant que $A \vdash P \xrightarrow{R_z} P'$, c'est-à-dire que le processus P offre le seal $z[R]$ qui pourra être déplacé et copié par l'environnement. Il est indispensable de demander qu'il existe un processus Q' tel que $A \vdash Q \xrightarrow{S_z} Q'$ pour un corps de seal S arbitraire, sinon un contexte peut facilement distinguer P et Q. En même temps, demander que R et S soient le même processus est trop restrictif. En effet, le contexte ne peut pas analyser directement les processus R et S, mais il doit avant tout les réactiver pour interagir avec par la suite. Par conséquent, la bisimulation ne pose pas de conditions pour R et S, mais demande que les résiduels obtenus après une interaction avec un contexte qui reçoit le seal soient équivalents. Le cas $\overline{x}^\eta\{\!\!\{ R \}\!\!\}$ est tout à fait similaire.

Il est donc nécessaire de caractériser tous les processus que l'on peut obtenir après la migration d'un seal : ce rôle est accompli par les *contextes de réception*. Un exemple permet d'illustrer comment ils ont été définis. Soit A un ensemble de noms, et soit P un processus

tel que

$$A \vdash P \xrightarrow{z[\overline{x}^{\uparrow}\{R\}]} P' .$$

Un contexte qui se synchronise avec cette action de P doit être de la forme $C'[-] \equiv C[\ - \mid x^z\{\!|\vec{y}|\!\}.V\ ]$, pour un $C[-]$ et V. La réduction qui en dérive est

$$C'[P] \ \equiv\ C[\ P \mid x^z\{\!|\vec{y}|\!\}.V\ ] \quad \twoheadrightarrow \quad C[\ (\boldsymbol{\nu}\,\mathrm{fn}(R) \setminus A)(P' \mid y_1[R] \mid \cdots \mid y_n[R]) \mid V\ ] .$$

Une propriété basique que nous demandons à notre bisimulation, $\approx$, est que des contextes comme $C'[-]$ ci-dessus ne puissent pas distinguer des processus équivalents. Dans l'exemple ci-dessus cela se traduit par le processus $C'[Q]$ qui doit évoluer vers un processus équivalent à celui du résiduel dans (3.3). Une façon d'imposer cette condition est de demander que Q exécute une action qui se synchronise avec $x^z\{\!|\vec{y}|\!\}.V$, par exemple demander qu'existent S, Q' tels que

$$A \vdash Q \xrightarrow{z[\overline{x}^{\uparrow}\{S\}]} Q'$$

et que le résiduel de $C'[P]$ soit équivalent au résiduel de $C'[Q]$:

$$C[\ (\boldsymbol{\nu}\,\mathrm{fn}(R) \setminus A)(P' \mid y_1[R] \mid \cdots \mid y_n[R]) \mid V\ ] \approx C[\ (\boldsymbol{\nu}\,\mathrm{fn}(S) \setminus A)(Q' \mid y_1[S] \mid \cdots \mid y_n[S]) \mid V\ ]$$

On peut factoriser le contexte $C[[-] \mid V]$, et vérifier seulement que

$$(\boldsymbol{\nu}\,\mathrm{fn}(R) \setminus A)(P' \mid y_1[R] \mid \cdots \mid y_n[R]) \quad \approx \quad (\boldsymbol{\nu}\,\mathrm{fn}(S) \setminus A)(Q' \mid y_1[S] \mid \cdots \mid y_n[S]) .$$

Cette idée peut être généralisée : on appelle le contexte

$$(\boldsymbol{\nu}\,\mathrm{fn}(Y) \setminus A)(X \mid y_1[Y] \mid \cdots \mid y_n[Y])$$

un contexte de réception de z vers $\uparrow$ dans l'univers A, noté $\mathcal{D}^A_{z,\uparrow}[X, Y]$. Son *environnement associé* est l'ensemble de noms libres obtenus après avoir remplacé les trous X et Y par respectivement le résiduel dans X, et le corps du seal envoyé dans Y.

Définition 3.4.7, page 86 (Contexte de réception) *Soit A un ensemble de noms, et soient X et Y deux processus tels que* $\mathrm{fn}(X) \subseteq A$. *Un* contexte de réception $\mathcal{D}^A_{\gamma,\eta}[X, Y]$ *et son* contexte associé $A_{\mathcal{D}^A_{\gamma,\eta}[X,Y]}$ *sont respectivement un processus et un ensemble de noms définis comme suit :*

si $\gamma, \eta = *, *$, ou $\gamma, \eta = z, \uparrow$, alors

$$\mathcal{D}^A_{\gamma,\eta}[X, Y] \ =\ (\boldsymbol{\nu}\,\mathrm{fn}(Y) \setminus A)(\ X \mid z_1[Y] \mid \cdots \mid z_n[Y]\ )$$

où les noms $\vec{z}$ satisfont la condition $\mathrm{fn}(\mathcal{D}^A_{\gamma,\eta}[X, Y]) \subseteq A \cup \vec{z}$.
Son environnement associé $A_{\mathcal{D}^A_{\gamma,\eta}[X,Y]}$ est $A \cup \vec{z}$;

si $\gamma, \eta = *, z$, alors

$$\mathcal{D}^A_{*,z}[X, Y] = (\boldsymbol{\nu}\,\mathrm{fn}(Y) \setminus A)(\ X \mid z[(\ z_1[Y] \mid \cdots \mid z_n[Y] \mid U\ )]\ )$$

où les noms $\vec{z}$ et le processus U satisfont la condition $\mathrm{fn}(\mathcal{D}^A_{*,z}[X, Y]) \subseteq A \cup \vec{z}$.
Son environnement associé $A_{\mathcal{D}^A_{*,z}[X,Y]}$ est $A \cup \vec{z}$;

si $\gamma, \eta = *, \uparrow$*, alors*

$$\mathcal{D}^A_{*,\uparrow}[X,Y] \;=\; z[X \mid U] \mid z_1[Y] \mid \cdots \mid z_n[Y]$$

où les noms $z, \vec{z}$ *et les processus* U *satisfont la condition* $\mathrm{fn}(\mathcal{D}^A_{*,\uparrow}[X,Y]) \subseteq A \cup \vec{z} \cup \{z\}$*. Son environnement associé* $A_{\mathcal{D}^A_{*,\uparrow}[X,Y]}$ *est* $A \cup \vec{z} \cup \{z\}$*.*

Nous écrivons $\mathcal{D}^A[X,Y]$ *quand nous quantifions sur tous les* γ, η *et écrivons* $A_{\mathcal{D}}$ *à la place de* $A_{\mathcal{D}^A_{\gamma,\eta}[X,Y]}$ *si cela ne donne lieu à aucune ambiguïté.*

Nous pouvons enfin définir une bisimulation étiquetée sur les processus du Seal.

Définition 3.4.8, page 86 (Bisimilarité) *On appelle* bisimilarité*, notée* $\approx$*, la plus grande famille de relations indexées par des ensembles finis de noms telles que toute* $\approx_A$ *est une relation symétrique sur* $\{P \mid \mathrm{fn}(P) \subseteq A\}$ *et, si* $P \approx_A Q$*, alors les conditions suivantes sont vérifiés :*

1. *si* $A \vdash P \xrightarrow{\tau} P'$ *alors il existe un* Q' *tel que* $A \vdash Q \Rightarrow Q'$ *et* $P' \approx_A Q'$ *;*
2. *si* $A \vdash P \xrightarrow{\ell} P'$ *et* $\ell \in \{\, \gamma[x^\eta(\vec{y})], \gamma[\overline{x}^\eta(\vec{y})], *[\overline{x}^\eta\{y\}], \gamma[x^\eta\{S\}], S^z, \gamma[x^\eta_z] \,\}$*, alors il existe un* Q' *tel que* $A \vdash Q \Rightarrow \xrightarrow{\ell} Q'$ *et* $P' \approx_{A \cup \mathrm{fn}(\ell)} Q'$ *;*
3. *si* $A \vdash P \xrightarrow{R_z} P'$ *alors existent* Q', S *tels que* $A \vdash Q \Rightarrow \xrightarrow{S_z} Q'$ *et pour tout contexte admissible* $\mathcal{D}^A[-,-]$ *on a* $\mathcal{D}^A[P',R] \approx_{A_{\mathcal{D}}} \mathcal{D}^A[Q',S]$*;*
4. *si* $A \vdash P \xrightarrow{\gamma[\overline{x}^\eta\{R\}]} P'$ *alors existent* Q', S *tels que* $A \vdash Q \Rightarrow \xrightarrow{\gamma[\overline{x}^\eta\{S\}]} Q'$ *et pour tout contexte admissible* $\mathcal{D}^A_{\gamma,\eta}[-,-]$ *on a* $\mathcal{D}^A_{\gamma,\eta}[P',R] \approx_{A_{\mathcal{D}}} \mathcal{D}^A_{\gamma,\eta}[Q',S]$*;*
5. *pour toute substitution* σ *telle que* $\mathrm{dom}(\sigma) \subseteq A$ *et* $\mathrm{ran}(\sigma) \subseteq A$*, on a* $P\sigma \approx_A Q\sigma$ *;*

où un contexte de réception $\mathcal{D}^A[-,-]$ *est* admissible *si les deux substitutions* $\mathcal{D}^A[P',R]$ *et* $\mathcal{D}^A[Q',S]$ *sont bien formées (c'est-à-dire, s'il n'y a pas de phénomènes de capture de noms).*

Les techniques standard, adaptées aux relations indexées, nous permettent de prouver que la relation $\approx$ existe, est unique, et est une équivalence (proposition 3.4.9, page 87).

Il est plus difficile de montrer que la bisimilarité est préservée par tous les opérateurs du calcul. On doit particulièrement veiller à éviter des problèmes avec les index : nous utilisons pour cela la notion de *congruence indexée* [Sew00].

Définition 3.4.10, page 87 (Congruence indexée) *Soit* $\mathcal{R}$ *une famille de relations indexées par des ensembles finis de noms telles que toute* $\mathcal{R}_A$ *soit une relation définie sur* $\{P \mid \mathrm{fn}(P) \subseteq A\}$*. On dit que* $\mathcal{R}$ *est une* congruence indexée *si toute* $\mathcal{R}_A$ *est une équivalence et si les conditions suivantes sont vraies :*

$$\frac{P\ \mathcal{R}_{A,\vec{y}}\ Q \quad x, \mathrm{fn}(\eta) \subseteq A}{x^\eta(\vec{y}).P\ \mathcal{R}_A\ x^\eta(\vec{y}).Q} \qquad \frac{P\ \mathcal{R}_A\ Q \quad \alpha \neq x^\eta(y), \mathrm{fn}(\alpha) \subseteq A}{\alpha.P\ \mathcal{R}_A\ \alpha.Q}$$

$$\frac{P\ \mathcal{R}_{A,\vec{y}}\ Q \quad x, \mathrm{fn}(\eta) \subseteq A}{!x^\eta(\vec{y}).P\ \mathcal{R}_A\ !x^\eta(\vec{y}).Q} \qquad \frac{P\ \mathcal{R}_A\ Q \quad \alpha \neq x^\eta(y), \mathrm{fn}(\alpha) \subseteq A}{!\alpha.P\ \mathcal{R}_A\ !\alpha.Q}\ .$$

$$\frac{P\ \mathcal{R}_A\ Q \quad x \in A}{x[P]\ \mathcal{R}_A\ x[Q]} \qquad \frac{P\ \mathcal{R}_A\ Q \quad \mathrm{fn}(R) \subseteq A}{\begin{array}{l} P \mid R\ \mathcal{R}_A\ Q \mid R \\ R \mid P\ \mathcal{R}_A\ R \mid Q \end{array}} \qquad \frac{P\ \mathcal{R}_{A,x}\ Q}{(\boldsymbol{\nu} x)P\ \mathcal{R}_A\ (\boldsymbol{\nu} x)Q}$$

On peut prouver par une longue induction sur les contextes que

Théorème 3.4.14, page 91 *La bisimilarité est une congruence indexée.*

On peut aussi prouver que la bisimulation est préservée par des substitutions injectives

Lemme 3.4.15, page 92 *Si $P \approx_A Q$ et $f : A \to B$ est injective, alors $fP \approx_B fQ$.*

Ces deux propriétés nous permettent de conclure que la méthode de preuve basée sur la bisimulation est correcte.

Théorème 3.4.16, page 93 (Soundness) *La bisimilarité est correcte par rapport à la congruence barbue : s'il existe un A tel que $P \approx_A Q$, alors $P \cong Q$.*

À propos de la complétude La bisimilarité ne coïncide pas avec la RBC pour plusieurs raisons. Tout d'abord, il faut observer que la bisimilarité est en style « retardé » : les transitions faibles ne permettent pas l'exécution d'actions τ après une action visible. Cette formulation est requise, parce qu'un processus qui exécute une action d'ordre supérieur nécessite la contribution d'un contexte de réception avant de continuer.

On remarquera aussi que le π-calcul est un sous-calcul du Seal Calcul, et caractériser complètement la congruence barbue en π-calcul requiert un opérateur explicite pour comparer deux noms, au moins avec un STE analogue au nôtre.

C'est cependant le modèle de mobilité qui soulève les problème le plus intéressant. L'une des caractéristiques de Seal est qu'un seal ne peut pas détecter s'il est déplacé par l'environnement ou pas. Nous pouvons ainsi construire les deux processus :

$$P \;=\; (\boldsymbol{\nu} x)(\overline{x}^*\{\!|y|\!\} \mid x^*\{\!|y|\!\}) \quad \text{et} \quad Q \;=\; \mathbf{0}\,.$$

Soit $A \supseteq \mathrm{fn}(P)$. Pour tout processus S, on a $A \vdash P \xrightarrow{S^y}$; au contraire Q n'exécute aucune action et on en déduit que $P \not\approx Q$. En même temps, aucun contexte ne peut distinguer P et Q : par exemple $C[-] \;=\; - \mid y[S]$ n'est d'aucune aide, parce que le contexte ne peut pas déterminer si le seal y a été déplacé ou non. En termes techniques, cela signifie qu'aucun contexte peut efficacement observer une action S^z.

La théorie algébrique du Seal Calcul La bisimulation permet de prouver aisément un certain nombre de lois algébriques, comme nous le montrons page 94 et suivantes. De plus, son développement nous a fait remarquer plusieurs caractéristiques du modèle de calcul du Seal Calcul qui étaient restées inaperçues pendant longtemps.

Cependant, elle demeure insuffisante pour prouver d'autres équations, principalement à cause de l'absence de techniques de preuve plus raffinées, comme les techniques de preuve dites « up-to ». L'objectif de notre étude du Seal Calcul était de jeter des bases solides, et nous pensons l'avoir fait. En même temps, le développement de méthodes de preuve puissantes sera l'un des buts de notre travail dans le cadre fourni par les Ambients Mobiles.

Les Ambients Mobiles

Le calcul des Ambients Mobiles a été introduit par Cardelli et Gordon il y a quelques années [CG00b] : son influence dans le domaine a été très importante, mais le développement d'un traitement sémantique approprié s'est révélé (historiquement) assez difficile. En effet les Ambients Mobiles présentent des difficultés inhabituelles, qui peuvent être résumées comme suit :

- il est difficile pour une localisation n de contrôler les interférences causées par d'autres localisations, ou bien par le processus localisé dans n même [LS00a] ;
- le modèle de mobilité est asynchrone : aucune autorisation n'est demandée pour migrer à l'intérieur d'une localisation. Comme Sangiorgi l'a observé [San01], cela peut être à l'origine d'un phénomène dit de *stuttering* : un ambient peut rentrer et sortir plusieurs fois d'une localisation sans que cela soit observable. Ainsi, une caractérisation de l'équivalence observationnelle ne doit pas observer le *stuttering*.
- Une des lois algébriques de AM est la loi dite du « pare-feu parfait », [CG00b] :

$$(\boldsymbol{\nu} n)n[P] = \mathbf{0} \qquad n \text{ inédit pour } P.$$

  Si l'on suppose $P = \texttt{in\_}k.\mathbf{0}$ il est évident qu'une bisimilarité qui veut capturer cette loi ne doit pas observer le mouvement des localisations secrètes, c'est-à-dire des localisations dont le nom n'est pas connu par l'environnement.

Le but de notre étude est de résoudre ces difficultés, et de donner un traitement sémantique au calcul des Ambients Mobiles.

Les Ambients Mobiles Nous récrivons tout d'abord la syntaxe du Calcul des Ambients sur deux niveaux, en introduisant la distinction entre processus et systèmes. Nous appellerons système un processus qui présente au niveau supérieur seulement des ambients, qui peuvent éventuellement partager entre eux des noms secrets. Cette distinction nous permet de nous concentrer sur le mouvement des agents en évitant la complexité introduite par des capacités qui ne sont pas exercées à l'intérieur d'une localisation. La syntaxe et les règles de réduction sont reportées figure 1.7. La syntaxe des processus est standard [CG00b], avec pour seule différence que la duplication est ici remplacée par la duplication d'un processus sous préfixe. On utilise les conventions habituelles.

La sémantique par réduction est définie en style chimique. La sémantique par réduction du calcul est donc définie par la *relation de réduction*, $\rightarrowtriangle$, qui est la relation la plus petite entre processus, fermée par les contextes statiques et satisfaisant les règles reportées figure 1.7. La sémantique par réduction utilise la relation auxiliaire de congruence structurelle, $\equiv$, qui identifie les termes syntaxiquement équivalents et permet ainsi de rapprocher les sous-termes qui vont participer à une interaction. Sa définition est standard (voir page 100). Les systèmes sont des processus qui ont une structure caractéristique, et les règles de la figure 1.7 décrivent aussi leur évolution. On remarque que la classe des systèmes est fermée par réduction.

On définit comme notion basique d'observation la présence au niveau supérieur d'un ambient dont le nom est public ; formellement, nous écrivons $M \downarrow n$ si $M \equiv (\boldsymbol{\nu}\tilde{m})(n[P] \mid M')$ où $n \notin \{\tilde{m}\}$. Nous écrivons $M \Downarrow n$ s'il existe un système M' tel que $M \rightarrowtriangle^* M'$ et $M' \downarrow n$. Le

Noms : $n, h, \ldots \in \mathbf{N}$

Systèmes :	*Processus :*	*Capacités :*	
$M, N ::= \mathbf{0}$	$P, Q, R ::= \mathbf{0}$	$C ::= \texttt{in\_}n$	entrer dans n
$\mid M_1 \mid M_2$	$\mid P_1 \mid P_2$	$\mid \texttt{out\_}n$	sortir de n
$\mid (\boldsymbol{\nu} n)M$	$\mid (\boldsymbol{\nu} n)P$	$\mid \texttt{open\_}n$	ouvrir n
$\mid n[P]$	$\mid n[P]$		
	$\mid C.P$		
	$\mid\ !C.P$		

Règles de réduction :

$n[\texttt{in\_}m.P \mid Q] \mid m[R] \twoheadrightarrow m[\, n[P \mid Q] \mid R\,]$ $\qquad$ $\texttt{open\_}n.P \mid n[\,Q\,] \twoheadrightarrow P \mid Q$

$m[\, n[\texttt{out\_}m.P \mid Q] \mid R\,] \twoheadrightarrow n[P \mid Q] \mid m[R]$ $\qquad$ $P \equiv Q, Q \twoheadrightarrow R, R \equiv S$ implique $P \twoheadrightarrow S$

Fig. 1.7: Les Ambients Mobiles en deux niveaux

choix de travailler avec des systèmes induit naturellement une classe de contextes, nommés *contextes de système*, et générés par la grammaire suivante :

$$C[-] \ ::= \ - \ \mid \ M \mid C[-] \ \mid \ C[-] \mid M \ \mid \ (\boldsymbol{\nu} n)C[-] \ \mid \ n[C[-] \mid P] \ \mid \ n[P \mid C[-]]$$

L'équivalence naturelle sur laquelle nous nous concentrerons est la *congruence barbue fermée par réduction* : nous reportons sa définition ci-dessous pour éviter toute confusion. La relation CBR, notée $\cong$, est la plus grande relation symétrique entre systèmes telle que si $M \cong N$ alors les faits suivants sont vérifiés :

- si $M \twoheadrightarrow M'$ alors il existe un système N' tel que $N \twoheadrightarrow^* N'$ et $M' \cong N'$;
- $M \downarrow n$ implique $Q \Downarrow n$;
- pour tout contexte de système $C[-]$, on a $C[P] \cong C[Q]$.

Un système de transitions étiquetées Dans les Ambients Mobiles, les préfixes C donnent naturellement lieu à des transitions de la forme $P \xrightarrow{C} Q$; par exemple nous avons

$$\texttt{in\_}n.P_1 \mid P_2 \xrightarrow{\texttt{in\_}n} P_1 \mid P_2 \ .$$

En même temps, les capacités élémentaires induisent des actions plus complexes. Nous allons donc définir un STE pour les Ambients Mobiles. Le STE est défini sur les processus, même si nos équivalences mettent en relation uniquement des systèmes. Nous distinguons les pré-actions, les env-actions et les τ-actions : les premières modélisent la possibilité pour un processus d'exécuter des capacités, et les deuxièmes modélisent l'interaction d'un système avec son environnement. Les τ-actions représentent des réductions que les termes peuvent exécuter sans nécessiter une contribution de l'environnement. Seules les τ-actions et les env-actions modélisent l'évolution d'un système.

Les pré-actions, définies figure 1.9, donnent lieu à des transitions de la forme $P \xrightarrow{\pi} O$, où les étiquettes π et les résiduels O sont définis dans la figure 1.8. Un résiduel sera un processus

$$
\begin{array}{lrcl}
\textit{Pré-actions :} & \pi & ::= & \mathtt{in}\_n \mid \mathtt{out}\_n \mid \mathtt{open}\_n \mid \mathtt{enter}\_n \mid \mathtt{amb}\_n \mid \mathtt{exit}\_n \\
\textit{Env-actions :} & \sigma & ::= & k.\mathtt{enter}\_n \mid k.\mathtt{exit}\_n \mid *.\mathtt{enter}\_n \mid *.\mathtt{exit}\_n \\
 & & & \quad \mid n.\overline{\mathtt{enter}}\_k \mid k.\mathtt{open}\_n \\
\textit{Actions :} & \alpha & ::= & \sigma \cup \tau \\
\textit{Concrétions :} & K & ::= & (\boldsymbol{\nu}\tilde{m})\langle P\rangle Q \\
\textit{Résiduels :} & O & ::= & P \mid K
\end{array}
$$

Fig. 1.8: Pré-actions, Env-actions, Actions, Concrétions, et Résiduels

simple si π est un préfixe du langage, ou bien une concrétion de la forme $(\boldsymbol{\nu}\tilde{m})\langle P\rangle Q$ si le périmètre d'un ambient est mis en cause. Dans ce cas, intuitivement, P représente la partie du système qui est influencée par l'action π et Q la partie qui ne l'est pas, et $\tilde{m}$ est l'ensemble des noms secrets partagés par P et Q. On utilisera les conventions suivantes : si K est la concrétion $(\boldsymbol{\nu}\tilde{m})\langle P\rangle Q$, alors $(\boldsymbol{\nu}r)K$ est une abréviation pour $(\boldsymbol{\nu}\tilde{m})\langle P\rangle(\boldsymbol{\nu}r)Q$ si $r \notin \mathrm{fn}(P)$, et pour $(\boldsymbol{\nu}r\tilde{m})\langle P\rangle Q$ sinon. De même, $K \mid R$ est défini comme $(\boldsymbol{\nu}\tilde{m})\langle P\rangle(Q \mid R)$, où $\tilde{m}$ est tel que $\mathrm{fn}(R) \cap \{\tilde{m}\} = \emptyset$.

Les τ-actions, définies en figure 1.10, modélisent l'évolution autonome d'un processus.

Les env-actions, définies figure 1.11, donnet lieu à des transitions de la forme $M \xrightarrow{\sigma} N$, où les étiquettes σ sont définies en figure 1.8. En pratique, les env-actions transforment les concrétions en processus en introduisant explicitement la contribution de l'environnement à la réduction. Cette contribution est toujours un ambient qui contient un processus arbitraire : le STE ne spécifie pas directement ce processus, qui est spécifié seulement dans le jeu de la bisimulation. Pour cela nous étendons la syntaxe de processus avec le processus spécial $\circ$, pour marquer les ambients dont le contenu doit être spécifié dans le jeu de la bisimulation. Le comportement du processus $\circ$ est analogue à celui du processus inactif : il est seulement un marque-page. On remarque que, à la différence des pré-actions et des τ-actions, les env-actions ne présentent pas de règles structurelles : les env-actions doivent être exécutées par des systèmes qui interagissent directement avec l'environnement.

On appellera *actions* les env-actions étendues par l'action τ.

Proposition 4.2.1, page 102 *Si T est un système (respectivement, un processus) et $T \xrightarrow{\alpha} T'$, alors T' est un système (respectivement, un processus) qui peut contenir le processus spécial $\circ$.*

Pour illustrer le STE, nous expliquerons les règles induites par le préfixe `in`, l'immigration des ambients. Un exemple typique d'un ambient m qui entre dans un ambient n est le suivant :

$$(\boldsymbol{\nu}m)(m[\,\mathtt{in}\_n.P_1 \mid P_2\,] \mid M) \mid n[Q] \twoheadrightarrow (\boldsymbol{\nu}m)(M \mid n[\,m[\,P_1 \mid P_2\,] \mid Q]) \;.$$

L'activation du préfixe $\mathtt{in}\_n$ à l'intérieur de l'ambient m donne à l'ambient m la capacité d'entrer dans n : nous formalisons cela avec une nouvelle action $\mathtt{enter}\_n$. Ainsi, la règle

(π Pfx) $\dfrac{}{\pi.P \xrightarrow{\pi} P}$ (π Repl Pfx) $\dfrac{}{!\pi.P \xrightarrow{\pi} P \mid !\pi.P}$ (π Amb) $\dfrac{}{n[P] \xrightarrow{\mathtt{amb}\_n} \langle P\rangle \mathbf{0}}$ (π Exit) $\dfrac{P \xrightarrow{\mathtt{out}\_n} P_1}{m[P] \xrightarrow{\mathtt{exit}\_n} \langle m[P_1]\rangle \mathbf{0}}$

(π Enter) $\dfrac{P \xrightarrow{\mathtt{in}\_n} P_1}{m[P] \xrightarrow{\mathtt{enter}\_n} \langle m[P_1]\rangle \mathbf{0}}$ (π Res) $\dfrac{P \xrightarrow{\pi} O \qquad n \notin \mathrm{fn}(\pi)}{(\boldsymbol{\nu} n)P \xrightarrow{\pi} (\boldsymbol{\nu} n)O}$ (π Par) $\dfrac{P \xrightarrow{\pi} O}{\begin{array}{c} P \mid Q \xrightarrow{\pi} O \mid Q \\ Q \mid P \xrightarrow{\pi} Q \mid O \end{array}}$

Fig. 1.9: Système de transitions étiquetées - Pré-actions

(τ Enter) $\dfrac{P \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{p})\langle P_1\rangle P_2 \quad Q \xrightarrow{\mathtt{amb}\_n} (\boldsymbol{\nu}\tilde{q})\langle Q_1\rangle {Q_2}^{(*)}}{\begin{array}{c} P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{p})(\boldsymbol{\nu}\tilde{q})(n[P_1 \mid Q_1] \mid P_2 \mid Q_2) \\ Q \mid P \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{q})(\boldsymbol{\nu}\tilde{p})(n[Q_1 \mid P_1] \mid Q_2 \mid P_2) \end{array}}$ (τ Par) $\dfrac{P \xrightarrow{\tau} P'}{\begin{array}{c} P \mid Q \xrightarrow{\tau} P' \mid Q \\ Q \mid P \xrightarrow{\tau} Q \mid P' \end{array}}$

(τ Exit) $\dfrac{P \xrightarrow{\mathtt{exit}\_n} (\boldsymbol{\nu}\tilde{m})\langle k[P_1]\rangle P_2}{n[P] \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{m})(k[P_1] \mid n[P_2])}$ (τ Amb) $\dfrac{P \xrightarrow{\tau} Q}{n[P] \xrightarrow{\tau} n[Q]}$

(τ Open) $\dfrac{P \xrightarrow{\mathtt{open}\_n} P_1 \quad Q \xrightarrow{\mathtt{amb}\_n} (\boldsymbol{\nu}\tilde{m})\langle Q_1\rangle Q_2}{\begin{array}{c} P \mid Q \xrightarrow{\tau} P_1 \mid (\boldsymbol{\nu}\tilde{m})(Q_1 \mid Q_2) \\ Q \mid P \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{m})(Q_1 \mid Q_2) \mid P_1 \end{array}}$ (τ Res) $\dfrac{P \xrightarrow{\tau} P'}{(\boldsymbol{\nu} n)P \xrightarrow{\tau} (\boldsymbol{\nu} n)P'}$

(*) On impose la condition $((\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2)) \cap \{\tilde{q}\}) = ((\mathrm{fn}(Q_1) \cup \mathrm{fn}(Q_2)) \cap \{\tilde{p}\}) = \emptyset$

Fig. 1.10: Système de transitions étiquetées - τ-actions

(π Enter) nous donne

$$m[\mathtt{in}\_n.P_1 \mid P_2] \xrightarrow{\mathtt{enter}\_n} \langle m[P_1 \mid P_2]\rangle \mathbf{0}$$

et, en utilisant les règles structurelles (π Res) et (π Par) on obtient

$$(\boldsymbol{\nu} m)(m[\mathtt{in}\_n.P_1 \mid P_2] \mid M) \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu} m)\langle m[P_1 \mid P_2]\rangle M.$$

Cela signifie que l'ambient $m[\mathtt{in}\_n.P_1 \mid P_2]$ a la capacité d'entrer dans un ambient nommé n. Si cette capacité est utilisée, l'ambient m entrera dans n, et le système M sera le résiduel, dans la localisation originale. Clairement, cette action peut être exécutée uniquement s'il y a un ambient nommé n à coté de m. La règle (π Amb) permet de tester la présence d'un ambient : on a

$$n[Q] \xrightarrow{\mathtt{amb}\_n} \langle Q\rangle \mathbf{0}\ .$$

La concrétion $\langle Q\rangle \mathbf{0}$ dit que Q est dans n, et $\mathbf{0}$ est dehors. Enfin, la règle (τ Enter) permet à ces deux actions complémentaires d'être exécutées simultanément, en exécutant la migration

(Enter)
$$\frac{P \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{m})\langle k[P_1]\rangle P_2{}^{(\dagger)}}{P \xrightarrow{k.\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{m})(n[\circ \mid k[P_1]] \mid P_2)}$$

(Exit)
$$\frac{P \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{m})\langle k[P_1]\rangle P_2{}^{(\dagger)}}{P \xrightarrow{k.\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{m})(k[P_1] \mid n[\,\circ \mid P_2])}$$

(Co-Enter)
$$\frac{P \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{m})\langle P_1\rangle P_2{}^{(\dagger)}}{P \xrightarrow{n.\overline{\texttt{enter}}\_k} (\boldsymbol{\nu}\tilde{m})(n[P_1 \mid k[\,\circ\,]] \mid P_2)}$$

(Open)
$$\frac{P \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{m})\langle P_1\rangle P_2}{P \xrightarrow{k.\texttt{open\_}n} k[\circ \mid (\boldsymbol{\nu}\tilde{m})(P_1 \mid P_2)]}$$

(Enter Shh)
$$\frac{P \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{m})\langle k[P_1]\rangle P_2{}^{(\ddagger)}}{P \xrightarrow{*.\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{m})(n[\circ \mid k[P_1]] \mid P_2)}$$

(Exit Shh)
$$\frac{P \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{m})\langle k[P_1]\rangle P_2{}^{(\ddagger)}}{P \xrightarrow{*.\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{m})(k[P_1] \mid n[\circ \mid P_2])}$$

(†) On impose $k \notin \tilde{m}$. (‡) On impose $k \neq n$ et $k \in \tilde{m}$

Fig. 1.11: Système de transitions étiquetées - Env-actions

de l'ambient à l'intérieur de l'ambient n, donnant lieu à la réduction originale :

$$(\boldsymbol{\nu}m)(m[\,\texttt{in\_}n.P_1 \mid P_2\,] \mid M) \mid n[Q] \xrightarrow{\tau} (\boldsymbol{\nu}m)(M \mid n[\,m[\,P_1 \mid P_2\,] \mid Q]) \,.$$

Les env-actions modélisent l'interaction d'un agent avec son environnement. Par exemple, étant donnée la capacité

$$(\boldsymbol{\nu}m)(m[\texttt{in\_}n.P_1 \mid P_2] \mid M) \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}m)\langle m[P_1 \mid P_2]\rangle M$$

l'application de la règle (Enter Shh) nous donne

$$(\boldsymbol{\nu}m)(m[\texttt{in\_}n.P_1 \mid P_2] \mid M) \xrightarrow{*.\texttt{enter\_}n} (\boldsymbol{\nu}m)(n[\circ \mid m[P_1 \mid P_2]] \mid M) \,.$$

Cette transition modélise un ambient secret qui entre dans un ambient n fourni par l'environnement. Le processus contenu dans n peut être ensuite spécifié, en remplaçant le marque-page. Si le nom n n'était pas secret, alors la règle (Enter) nous aurait donné la transition

$$m[\texttt{in\_}n.P_1 \mid P_2] \mid M \xrightarrow{m.\texttt{enter\_}n} n[\circ \mid m[P_1 \mid P_2]] \mid M$$

pour modéliser un ambient public m qui entre dans un ambient n fourni par l'environnement.

Les règles pour l'émigration et l'ouverture d'un ambient sont analogues. Enfin, si un système offre un ambient public n au niveau supérieur, alors un contexte peut interagir avec le système en présentant un ambient qui entre dans n. La règle (Co-Enter) capture cette interaction entre le système et l'environnement.

La sémantique définie par le STE coïncide avec la sémantique par réduction :

Théorème 4.2.3, page 105 *Si* $P \xrightarrow{\tau} P'$ *alors* $P \twoheadrightarrow P'$. *Si* $P \twoheadrightarrow P'$ *alors* $P \xrightarrow{\tau} \equiv P'$.

On remarque que si $M \cong N$ alors (i) $M \Downarrow n$ si et seulement si $N \Downarrow n$ et (ii) $M \Rightarrow M'$ implique que il y a un système N' tel que $N \Rightarrow N'$ et $M' \cong N'$. Dans la suite on utilisera ces propriétés sans mention spécifique.

Une caractérisation de la congruence barbue fermée par réduction Notre but est la définition d'une bisimulation étiquetée qui coïncide avec la congruence barbue. Nous nous intéressons à des équivalences faibles, qui font abstraction des τ-actions : pour cela nous introduisons les actions faibles. Leur définition est standard : $\Rightarrow$ décrit la fermeture réflexive et transitive de $\stackrel{\tau}{\longrightarrow}$; $\stackrel{\alpha}{\Longrightarrow}$ décrit $\Rightarrow \stackrel{\alpha}{\longrightarrow} \Rightarrow$; $\stackrel{\hat{\alpha}}{\Longrightarrow}$ décrit $\Rightarrow$ si $\alpha = \tau$ et $\stackrel{\alpha}{\Longrightarrow}$ sinon. Avant d'introduire la bisimulation, nous avons encore besoin de clarifier comment remplacer le marque-page $\circ$ par un vrai processus : l'opérateur $\bullet$ en est responsable.

Définition 4.3.1, page 108 *Soient T et T_i des systèmes ou bien des processus. Alors, étant donné un processus P, on définit :*

$$\begin{array}{rclrclrcl}
\mathbf{0} \bullet P & \stackrel{\text{def}}{=} & \mathbf{0} & (T_1 \mid T_2) \bullet P & \stackrel{\text{def}}{=} & (T_1 \bullet P) \mid (T_2 \bullet P) & \circ \bullet P & \stackrel{\text{def}}{=} & P \\
n[R] \bullet P & \stackrel{\text{def}}{=} & n[R \bullet P] & (\boldsymbol{\nu} n)T \bullet P & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} n)(T \bullet P) \text{ if } n \notin \text{fn}(P) & & & \\
!C.R \bullet P & \stackrel{\text{def}}{=} & !C.(R \bullet P) & C.R \bullet P & \stackrel{\text{def}}{=} & C.(R \bullet P) & & &
\end{array}$$

Nous pouvons enfin introduire notre bisimulation étiquetée entre systèmes.

Définition 4.3.2, page 108 (Bisimilarité) *Une relation symétrique $\mathcal{R}$ est une* bisimulation *si $M \mathcal{R} N$ implique :*

- *si $M \stackrel{\alpha}{\longrightarrow} M'$, $\alpha \notin \{*.\texttt{enter}\_n, *.\texttt{exit}\_n\}$, alors il existe un système N' tel que $N \stackrel{\hat{\alpha}}{\Longrightarrow} N'$ et pour tous les processus P on a $M' \bullet P \mathcal{R} N' \bullet P$;*
- *si $M \stackrel{*.\texttt{enter}\_n}{\longrightarrow} M'$ alors il existe un système N' tel que $N \mid n[\circ] \Rightarrow N'$ et pour tous les processus P on a $M' \bullet P \mathcal{R} N' \bullet P$;*
- *si $M \stackrel{*.\texttt{exit}\_n}{\longrightarrow} M'$ alors il existe un système N' tel que $n[\circ \mid N] \Rightarrow N'$ et pour tous les processus P on a $M' \bullet P \mathcal{R} N' \bullet P$.*

Deux systèmes M et N sont dit bisimilaires*, notés $M \approx N$, s'il existe une bisimulation $\mathcal{R}$ telle que $M \mathcal{R} N$.*

La bisimilarité demande une quantification universelle des processus P fournis par l'environnement. Ce processus remplace le marque-page $\circ$ qui peut avoir été généré par les env-actions. La bisimilarité est définie en style tardif : la quantification existentielle précède l'universelle. On peut définir une version en style précoce de la bisimilarité, où la quantification universelle de la contribution de l'environnement P précède celle de l'existence du système N'. On peut montrer facilement que, comme dans HOπ, les versions tardive et précoce coïncident.

Il est important de remarquer que les actions $*.\texttt{enter}\_n$ et $*.\texttt{exit}\_n$ requièrent une correspondance moins stricte dans le jeu de la bisimulation : ces actions ne sont pas observables, et notre approche est similaire au traitement du préfixe d'entrée en π-calcul asynchrone [ACS98].

Nous démontrons que la bisimilarité est une relation contextuelle, ce qui permet de prouver que la bisimilarité est une méthode de preuve correcte pour RBC.

Théorème 4.3.7, page 122 (Soundness) *La bisimilarité est contenue dans la* RBC.

La difficulté principale pour prouver que la bisimilarité coïncide avec la RBC est la définition de contextes qui permettent d'observer les actions visibles. La définition de ces

$$
\begin{aligned}
C_{k.\texttt{enter}\_n}[-] &= n[\circ \mid \mathsf{done}[\texttt{in}\_k.\texttt{out}\_k.\texttt{out}\_n]] \mid - \\
C_{k.\texttt{exit}\_n}[-] &= (\boldsymbol{\nu} a)a[\texttt{in}\_k.\texttt{out}\_k.\mathsf{done}[\texttt{out}\_a]] \mid n[\circ \mid -] \\
C_{n.\overline{\texttt{enter}}\_k}[-] &= (\boldsymbol{\nu} a)a[\texttt{in}\_n.k[\texttt{out}\_a.(\circ \mid (\boldsymbol{\nu} b)b[\texttt{out}\_k.\texttt{out}\_n.\mathsf{done}[\texttt{out}\_b]])]] \mid - \\
C_{k.\texttt{open}\_n}[-] &= k[\circ \mid (\boldsymbol{\nu} a, b)(\texttt{open}\_b.\texttt{open}\_a.\mathsf{done}[\texttt{out}\_k] \mid a[- \mid \texttt{open}\_n.b[\texttt{out}\_a]])]
\end{aligned}
$$

où a,b et done sont inédits.

Fig. 1.12: Contextes pour les actions visibles

$$
\begin{aligned}
-_1 \oplus -_2 &= (\boldsymbol{\nu} o)(o[\,] \mid \texttt{open}\_o.-_1 \mid \texttt{open}\_o.-_2) \\
\mathrm{SPY}_\alpha\langle i, j, -\rangle &= (i[\texttt{out}\_n] \mid -) \oplus (j[\texttt{out}\_n] \mid -) \\
&\qquad \text{si } \alpha \in \{k.\texttt{enter}\_n, k.\texttt{exit}\_n, k.\texttt{open}\_n, *.\texttt{enter}\_n, *.\texttt{exit}\_n\} \\
\mathrm{SPY}_\alpha\langle i, j, -\rangle &= (i[\texttt{out}\_k.\texttt{out}\_n] \mid -) \oplus (j[\texttt{out}\_k.\texttt{out}\_n] \mid -) \text{ si } \alpha \in \{n.\overline{\texttt{enter}}\_k\}
\end{aligned}
$$

Fig. 1.13: Contextes et processus auxiliaires

contextes $C_\alpha[-]$, pour toute action visible α, est reportée figure 1.12. L'ambient nommé done est utilisé comme barbe fraîche pour signaler que l'action α a été observée. Nous démontrons qu'il existe une étroite correspondance entre les actions visibles α et leurs contextes correspondants $C_\alpha[-]$. Le lemme suivant montre que les contextes peuvent mimer l'exécution des actions visibles.

Lemme 4.3.8, page 123 *Soit M un système et soit $\alpha \in \{k.\texttt{enter}\_n, k.\texttt{exit}\_n, n.\overline{\texttt{enter}}\_k, k.\texttt{open}\_n\}$. Pour tout processus P, si $M \xrightarrow{\alpha} M'$, alors $C_\alpha[M] \bullet P \Rightarrow\cong M' \bullet P \mid \mathsf{done}[\,]$.*

L'inverse de ce lemme requiert des définitions techniques reportées figure 1.13. Le terme $\oplus$ implante une forme de choix interne, et les contextes $\mathrm{SPY}_\alpha\langle i, j, -\rangle$ sont des outils qui permettent de garantir que le processus P fourni par l'environnement n'exécute aucune action. Les processus $\mathrm{SPY}_\alpha\langle i, j, P\rangle$ peuvent observer les réductions de P du fait que l'une des deux barbes i et j est perdue si P exécute une action.

Lemme 4.3.11, page 126 *Soit M un système, soit $\alpha \in \{k.\texttt{enter}\_n, k.\texttt{exit}\_n, n.\overline{\texttt{enter}}\_k, k.\texttt{open}\_n\}$, et soient i, j des noms frais pour M. Pour tout processus P avec $\{i, j\} \cap \mathrm{fn}(P) = \emptyset$, si $C_\alpha[M] \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle \Rightarrow\cong O \mid \mathsf{done}[\,]$ et $O \Downarrow_{i,j}$, alors il existe un système M' tel que $O \cong M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle$ et $M \stackrel{\alpha}{\Longrightarrow} M'$.*

Théorème 4.3.12, page 131 (Complétude) *La relation* RBC *est contenue dans la bisimilarité.*

Démonstration [Esquisse] Nous démontrons que la relation $\mathcal{R} = \{(M, N) \mid M \cong N\}$ est une bisimulation. Pour cela, supposons que $M \mathcal{R} N$ et $M \xrightarrow{\alpha} M'$. Supposons aussi que $\alpha \in \{k.\texttt{enter\_}n, k.\texttt{exit\_}n, n.\overline{\texttt{enter}}\texttt{\_}k, k.\texttt{open\_}n\}$. Nous devons trouver un système N' tel que $N \stackrel{\alpha}{\Longrightarrow} N'$ et pour tout processus P on a $M' \bullet P \cong N' \bullet P$.

L'idée de la preuve est d'utiliser un contexte particulier qui puisse mimer l'effet de l'action α, et qui nous permette ensuite de comparer les résiduels des deux systèmes. Ce contexte prend la forme

$$D_\alpha\langle P\rangle[-] \;=\; (C_\alpha[-] \mid \mathsf{Flip}) \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle$$

où $C_\alpha[-]$ sont les contextes définis en figure 1.12 et Flip est le système

$$(\boldsymbol{\nu} k)k[\texttt{in\_done.out\_done.}(\mathsf{succ}[\texttt{out\_}k] \oplus \mathsf{fail}[\texttt{out\_}k])] \;.$$

Les noms succ et fail ont été choisis de sorte qu'ils soient frais. Intuitivement, l'existence de la barbe fail montre que l'action α n'a pas encore été exécutée ; au contraire, l'existence de succ avec l'absence de fail assure que l'action α a été exécutée, et que cette exécution a été attestée par la présence de l'ambient nommé done.

Du moment que $\cong$ est contextuelle, $M \cong N$ implique que pour tout processus P on a $D_\alpha\langle P\rangle[M] \;\cong\; D_\alpha\langle P\rangle[N]$. Grâce au lemme 4.3.8, et par examen des réductions du processus Flip, on observe que :

$$D_\alpha\langle P\rangle[M] \;\Rightarrow\!\cong\; M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle \mid \mathsf{done}[\,] \mid \mathsf{Flip} \;\Rightarrow\!\cong\; M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle \mid \mathsf{done}[\,] \mid \mathsf{succ}[\,]$$

où $M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle \mid \mathsf{done}[\,] \mid \mathsf{succ}[\,] \Downarrow_{i,j,\mathsf{succ}} \not\Downarrow_{\mathsf{fail}}$. Appelons ce résiduel O_1. À cette réduction doit correspondre une réduction $D_\alpha\langle P\rangle[N] \;\Rightarrow\; O_2$, où $O_1 \cong O_2$. En même temps, la correspondance doit préserver les barbes de O_1, parce que l'on doit avoir $O_2 \Downarrow_{i,j,\mathsf{succ}} \not\Downarrow_{\mathsf{fail}}$.

Les barbes $O_2 \Downarrow_{\mathsf{succ}} \not\Downarrow_{\mathsf{fail}}$ nous indiquent que $O_2 \cong \hat{N} \mid \mathsf{done}[\,] \mid \mathsf{succ}[\,]$, pour un système $\hat{N}$. Mais $O_2 \Downarrow_{i,j}$: l'observation précédente peut alors être combinée avec le lemme 4.3.11 pour obtenir l'existence d'un système N' tel que $\hat{N} \cong N' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle$ et d'une action faible $N \stackrel{\alpha}{\Longrightarrow} N'$.

Pour conclure, il nous faut montrer que pour tout processus P on a $M' \bullet P \;\cong\; N' \bullet P$. La relation RBC est préservée par restriction, et cela nous permet de dériver $(\boldsymbol{\nu}\mathsf{done}, \mathsf{succ})O_1 \cong (\boldsymbol{\nu}\mathsf{done}, \mathsf{succ})O_2$. Mais on a $(\boldsymbol{\nu}\mathsf{done})\mathsf{done}[\,] \;\cong\; (\boldsymbol{\nu}\mathsf{succ})\mathsf{succ}[\,] \;\cong\; \mathbf{0}$, et il s'ensuit que $M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle \;\cong\; N' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle$. Même raisonnement, $\cong$ est préservée par restriction et $(\boldsymbol{\nu} i, j)\mathrm{SPY}_\alpha\langle i, j, P\rangle \cong P$. Par conséquent, nous pouvons enfin dériver $M' \bullet P \;\; \mathcal{R} \;\; N' \bullet P$, pour tout processus P.

Pour compléter la preuve nous devons aussi considérer les actions $*.\texttt{enter\_}n$ et $*.\texttt{exit\_}n$: le fait qu'elles ne sont pas observables rend ces cas beaucoup plus simples. □

Nous pouvons ainsi conclure que la bisimilarité et la congruence barbue par réduction coïncident.

La communication synchrone de capacités peut être ajoutée au calcul des Ambients Mobiles. Le processus de sortie $\langle E\rangle.P$ envoie le message E et continue ensuite comme P ; le processus d'entrée $(x).Q$ reçoit un message, le lie à la variable x dans Q, qui est ensuite exécuté. Comme discuté dans [VC99, San01], la communication synchrone est raisonnable dans le modèle de calcul des Ambient Mobiles, parce que la communication est toujours

locale. Notre STE peut être facilement étendu pour gérer la communication, et la preuve du théorème 4.2.3 peut être aisément complétée. De plus, dans notre cadre la communication ne peut pas être observée au niveau supérieur : cela implique que la bisimulation peut être appliquée au calcul étendu et que tous nos résultats restent vrais.

Techniques de preuve dites « up-to » Nous adaptons des techniques de preuve dites « up-to » à notre cadre : ces techniques permettent de réduire grandement la taille de la relation qu'il faut construire pour prouver que deux processus sont bisimilaires.

Définition (Bisimulation up-to contexte et up-to $\equiv$) *Une relation symétrique $\mathcal{R}$ est une* bisimulation up-to contexte et up-to $\equiv$ *si $P\,\mathcal{R}\,Q$ implique :*

- *si $M \xrightarrow{\alpha} M''$, $\alpha \notin \{*.\texttt{enter}\_n, *.\texttt{exit}\_n\}$, alors il existe un système N'' tel que $N \stackrel{\hat{\alpha}}{\Longrightarrow} N''$, et pour tout processus P il existe un contexte de système $C[-]$ et des systèmes M' et N' tels que $M'' \bullet P \equiv C[M']$, $N'' \bullet P \equiv C[N']$, et $M'\,\mathcal{R}\,N'$;*
- *si $M \xrightarrow{*.\texttt{enter}\_n} M''$ alors il existe un système N'' tel que $N \mid n[\,\circ\,] \Rightarrow N''$, et pour tout processus P il existe un contexte de système $C[-]$ et des systèmes M' et N' tels que $M'' \bullet P \equiv C[M']$, $N'' \bullet P \equiv C[N']$, et $M'\,\mathcal{R}\,N'$;*
- *si $M \xrightarrow{*.\texttt{exit}\_n} M''$ alors il existe un système N'' tel que $n[\circ \mid N] \Rightarrow N''$, et pour tout processus P il existe un contexte de système $C[-]$ et des systèmes M' et N' tels que $M'' \bullet P \equiv C[M']$, $N'' \bullet P \equiv C[N']$, et $M'\,\mathcal{R}\,N'$.*

La technique de preuve up-to contexte et up-to congruence structurelle est une technique de preuve correcte :

Théorème *Si $\mathcal{R}$ est une bisimulation up-to contexte et up-to $\equiv$, alors $\mathcal{R} \subseteq \approx$.*

Nous introduisons aussi la relation d'expansion (définition 4.4.1, page 134), qui est essentiellement une variante asymétrique de la bisimulation permettant d'imposer des contraintes sur les réductions internes exécutées par les systèmes, et nous démontrons que la technique de preuve up-to contexte et up-to expansion est aussi correcte (théorème 4.4.7, page 135).

Théorie algébrique Pour montrer la puissance de nos méthodes de preuve nous reportons ici deux exemples : le premier est la preuve de l'équation dite du « pare-feu parfait », le second montrera comment notre bisimulation est insensible au *stuttering*.

Théorème 4.6.1, page 139 $(\boldsymbol{\nu} n)n[P] \;\cong\; \mathbf{0}$ *si $n \notin \mathrm{fn}(P)$.*

Démonstration [Esquisse] Soit $\mathcal{S}$ la fermeture symétrique de la relation $((\boldsymbol{\nu} n)n[Q], \mathbf{0}) \mid \forall Q \text{ s.t. } n \notin \mathrm{fn}(Q)\}$. La relation $\mathcal{S}$ est une bisimulation up-to contexte et up-to $\equiv$. Les cas les plus délicats concernent les mouvements invisibles $*.\texttt{enter}\_k$ et $*.\texttt{exit}\_k$. On ne détaille ici que le premier. Si

$$(\boldsymbol{\nu} n)n[P] \xrightarrow{*.\texttt{enter}\_k} (\boldsymbol{\nu} n)k[\circ \mid n[P']] \equiv k[\circ \mid (\boldsymbol{\nu} n)n[P']] \qquad \text{alors} \qquad \mathbf{0} \mid k[\,\circ\,] \Rightarrow \equiv k[\circ \mid \mathbf{0}]$$

et les deux résiduels sont encore dans $\mathcal{S}$ up-to contexte et up-to congruence structurelle. □

Dans [San01] on montre que les équivalences comportementales barbues sont insensibles à un phénomène dit de « stuttering », causé par des processus qui peuvent répétitivement entrer et sortir d'une localisation. En utilisant un opérateur de somme non-déterministe à la CCS, l'exemple suivant donne des intuitions sur les *stuttering*. Les systèmes

$$M = m[\texttt{in\_}n.\texttt{out\_}n.\texttt{in\_}n.R] \qquad \text{et} \qquad N = m[\texttt{in\_}n.\texttt{out\_}n.\texttt{in\_}n.R + \texttt{in\_}n.R]$$

sont en effet équivalents. Pour comprendre pourquoi le terme supplémentaire de N n'influence pas son comportement, on étudie la réduction :

$$N \mid n[S] \;\rightarrowtriangle\; n[S \mid m[R]] \ .$$

Le processus M peut répondre au cours du jeu de la bisimulation par une séquence de trois réductions :

$$M \mid n[S] \;\rightarrowtriangle\; n[S \mid m[\texttt{out\_}n.\texttt{in\_}n.R]] \;\rightarrowtriangle\; n[S] \mid m[\texttt{in\_}n.R] \;\rightarrowtriangle\; n[S \mid m[R]] \ .$$

Comme l'exemple le montre, la capacité $\texttt{in\_}n$ correspond à l'exécution de trois capacités $\texttt{in\_}n.\texttt{out\_}n.\texttt{in\_}n$. Même s'il peut apparaître que la bisimulation que nous avons définie fait correspondre à chaque action une seule action (éventuellement précédée et/ou suivie par des τ-transitions), elle est insensible au *stuttering*. Pour le montrer, nous utilisons une variante de l'exemple ci-dessus qui n'utilise pas la somme non-déterministe.

La duplication dans les deux systèmes M et N ci-dessous implante une boucle où l'ouverture de l'ambient l alterne avec le chemin $\texttt{in\_}n.\texttt{out\_}n$; un décalage d'un cycle entre les deux boucles semble différencier M de N mais le *stuttering* rend ce décalage négligeable.

Proposition 4.6.2, page 140 *Les systèmes M et N définis comme*

$$M = m[(\boldsymbol{\nu} l)(\texttt{in\_}n.l[] \mid !\texttt{open\_}l.\texttt{out\_}n.\texttt{in\_}n.l[])]$$
$$N = m[(\boldsymbol{\nu} l)(\texttt{in\_}n.\texttt{out\_}n.\texttt{in\_}n.l[] \mid !\texttt{open\_}l.\texttt{out\_}n.\texttt{in\_}n.l[])]$$

sont bisimilaires.

Démonstration [Esquisse] Soit $\mathcal{R}$ la fermeture symétrique de la relation

$$\{(\ k[O \mid (\boldsymbol{\nu} l)(\texttt{in\_}n.l[] \mid !\texttt{open\_}l.\texttt{out\_}n.\texttt{in\_}n.l[])]\ ,$$
$$\quad k[O \mid (\boldsymbol{\nu} l)(\texttt{in\_}n.\texttt{out\_}n.\texttt{in\_}n.l[] \mid !\texttt{open\_}l.\texttt{out\_}n.\texttt{in\_}n.l[])]\ )$$
$$\qquad\qquad \mid\ k \text{ et } O \text{ sont arbitraires}\} \ \cup\ \mathcal{I}$$

où $\mathcal{I}$ est la relation identique entre systèmes. La relation $\mathcal{R}$ est une bisimulation up-to contexte et up-to congruence structurelle. Nous détaillons le cas le plus intéressant, qui demande qu'à l'exercice d'une capacité corresponde une séquence de trois capacités. Supposons que $M\ \mathcal{R}\ N$, avec

$$M = k[O \mid (\boldsymbol{\nu} l)(\texttt{in\_}n.l[] \mid !\texttt{open\_}l.\texttt{out\_}n.\texttt{in\_}n.l[])]$$

et

$$N = k[O \mid (\boldsymbol{\nu} l)(\texttt{in\_}n.\texttt{out\_}n.\texttt{in\_}n.l[] \mid !\texttt{open\_}l.\texttt{out\_}n.\texttt{in\_}n.l[])] \ .$$

Supposons aussi que

$$M \xrightarrow{k.\mathtt{enter}\_n} n[\circ \mid k[O \mid (\boldsymbol{\nu} l)(l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])]] \ .$$

Alors N peut réaliser la séquence de transitions suivante :

$$\begin{aligned} N &\xrightarrow{k.\mathtt{enter}\_n} n[\circ \mid k[O \mid (\boldsymbol{\nu} l)(\mathtt{out}\_n.\mathtt{in}\_n.l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])]] \\ &\xrightarrow{\tau} n[\,\circ\,] \mid k[O \mid (\boldsymbol{\nu} l)(\mathtt{in}\_n.l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])] \\ &\xrightarrow{\tau} n[\circ \mid k[O \mid (\boldsymbol{\nu} l)(l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])]] \ . \end{aligned}$$

Nous pouvons factoriser le contexte $n[\circ \mid -]$: en travaillant up-to contexte, on voit que les résiduels sont toujours dans $\mathcal{R}$. □

Cette preuve montre clairement comment l'exécution des trois capacités $\mathtt{in}\_n.\mathtt{out}\_n.\mathtt{in}\_n$ requise pour répondre à la capacité $\mathtt{in}\_n$ donne lieu à une action $k.\mathtt{enter}\_n$ suivie par deux τ transitions. Les τ-transitions sont ensuite absorbées parce que la bisimilarité est une équivalence faible.

Nous avons ainsi développé un cadre sémantique approprié pour les Ambients Mobiles de Cardelli et Gordon, grâce à des idées innovatrices qui permettent de modéliser la mobilité asynchrone et le *stuttering*. Le cadre est complété par des techniques de preuve puissantes. Ces résultats permettent enfin de considérer les Ambients Mobiles comme un véritable *calcul* de processus.

Une théorie des domaines pour la concurrence

Le point de départ de notre étude du Seal Calcul et des Ambients Mobiles était leurs théories opérationnelles, et l'objectif était le développement de leur théories sémantiques. Indépendamment des résultats obtenus, l'approche essentiellement opérationnelle suivie se prête mal à faire ressortir les relations entre les langages, et, en général, le dessin sous-jacent les systèmes concurrents. C'est pour cela que dans la deuxième partie de cette thèse nous dédions nos efforts à un langage enraciné dans une théorie des domaines de la concurrence basée sur les chemins d'exécutions.

Notre investigation entre dans une direction de recherche initiée et étudiée principalement par Glynn Winskel et ses co-auteurs. Initialement, la théorie des catégories a été utilisée pour mettre en relation les modèles classiques de la concurrence : cela a permis non seulement de comprendre les similitudes et différences existantes parmi les constructions utilisés en concurrence, mais a aussi permis de découvrir une notion de bisimulation, nommée *bisimulation des applications ouvertes*, qui peut être appliquée uniformément à plusieurs modèles, même s'ils sont très différents. Une classe de catégories se prête naturellement à l'utilisation de cette bisimulation abstraite et s'est révélée une base intéressante pour développer une théorie des domaines pour la concurrence : Cattani et Winskel se sont ainsi intéressés à l'étude d'une 2-catégorie basée sur les pré-faisceaux [CW03]. Il faut en même temps admettre que les connaissances de théorie des catégories requises ne sont pas triviales. La nécessité d'avoir un système formel pour mieux étudier ces modèles a mené Winskel et Nygaard à la définition

d'un langage, nommé HOPLA (acronyme pour *higher-order process language*), pour l'une des catégories étudiées. Ce langage s'est révélé simple et expressif, et surtout adapté à l'étude de nombreux calculs de processus d'ordre supérieur.

Le langage HOPLA ne prévoit pas d'opérateurs pour modéliser la génération de noms, et, sous cet aspect, il est incomplet parce qu'il ne permet pas d'étudier les phénomènes spécifiques aux langages de processus modernes. Notre contribution a été d'étendre le langage HOPLA par des opérateurs pour manipuler la génération de noms, sans toutefois le dénaturer. Même si nous nous concentrerons sur ses aspects opérationnels, nous commençons par donner un aperçu du modèle dénotationnel qui nous a guidé dans sa définition.

Une théorie des domaines basée sur les chemins d'exécution Les processus peuvent exécuter des chemins d'exécution qui sont souvent extrêmement compliqués : cette simple observation suggère de construire une théorie des domaines pour la concurrence directement sur la notion de chemin, et de représenter les processus comme des collections de leur chemins d'exécution.

Les chemins sont les éléments des préordres $\mathbb{P}, \mathbb{Q}, \ldots$ nommés *ordres de chemins*. Les ordres de chemins jouent le rôle des types de processus, en décrivant la forme des chemins que les processus peuvent réaliser. La relation d'ordre permet de déterminer pour un chemin donné tous ses sous-chemins. Un processus de type $\mathbb{P}$ est alors représenté comme un sous-ensemble fermé vers le bas $X \subseteq \mathbb{P}$, nommé un *ensemble de chemins*. Les ensembles de chemins ordonnés par inclusion sont les éléments de l'ensemble partiellement ordonné $\widehat{\mathbb{P}}$, auquel on pensera comme à un domaine pour interpréter les processus de type $\mathbb{P}$. L'ordre partiel $\widehat{\mathbb{P}}$ a plusieurs propriétés intéressantes :

- il est un treillis complet avec bornes supérieures données par l'union ensembliste : en suivant Hennessy et Plotkin [HP79], $\widehat{\mathbb{P}}$ est un domaine non-déterministe, où les bornes supérieures sont utilisées pour interpréter la somme non-déterministe des processus ;
- tout $X \in \widehat{\mathbb{P}}$ est le sup de certains éléments *premiers* : les premiers ont la forme $y_{\mathbb{P}}p = \{p' \mid p' \leq_{\mathbb{P}} p\}$ et représentent un processus qui peut réaliser le chemin d'exécution p ;
- on peut caractériser abstraitement $\widehat{\mathbb{P}}$ comme la *sup-complétion* de $\mathbb{P}$.

Cette dernière propriété attire notre attention sur les fonctions $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ préservant les bornes supérieures. On écrira **Lin** pour la catégorie qui a les ordres de chemins $\mathbb{P}, \mathbb{Q}, \ldots$ pour objets, et les fonctions préservant les bornes supérieures $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ pour flèches. La catégorie **Lin** peut être vue comme un modèle de la logique linéaire classique : l'involution $\mathbb{P}^{\perp}$ est donnée par $\mathbb{P}^{\mathrm{op}}$, le tenseur $\mathbb{P} \otimes \mathbb{Q}$ par le produit des préordres $\mathbb{P} \times \mathbb{Q}$, le produit $\mathbb{P} \& \mathbb{Q}$ et le coproduit sont tous les deux donnés par la somme disjointe des préordres $\mathbb{P} + \mathbb{Q}$, et l'espace des fonctions $\mathbb{P} \multimap \mathbb{Q}$ est donné par $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$ (voir page 150 et suivantes). En même temps, **Lin** ne supporte pas toutes les opérations associées aux langages de processus : en effet, toutes les flèches préservent les bornes supérieures et, à la différence de l'opérateur de préfixe, envoient la somme vide (le processus inactif) vers elle-même.

En suivant la discipline de la logique linéaire, on peut obtenir des flèches non-linéaires à partir de fonctions linéaires dont le domaine est sous un exponentiel. Un choix possible dans **Lin** est de définir les exponentiels $\mathbb{P}$ l'ensemble des ensembles finis de chemins, ordonné par

$$P \preceq_{\mathbb{P}} P' \Leftrightarrow \forall p \in P. \exists p' \in P'. p \leq_{\mathbb{P}} p' \ .$$

On peut alors observer qu'une flèche linéaire $!\mathbb{P} \to \mathbb{Q}$ correspond à une fonction Scott-continue $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ (page 151). Les flèches continues permettent à plusieurs opérations (parmi lesquelles le préfixe) d'être interprétées : un élément de $P \in !\mathbb{P}$ contient plusieurs chemins de calcul de $\mathbb{P}$ et il peut être considéré comme décrivant l'exécution de plusieurs copies d'un processus de type $\mathbb{P}$.

Les fonctions continues peuvent être utilisées pour définir une catégorie qui soit un modèle complètement adéquat de HOPLA [NW03]. Pour gérer la génération de noms, nous devons faire un pas de plus. En suivant [Sta96], on peut faire deux observations importantes qui permettent de caractériser le modèle recherché.

1. Le comportement d'un processus dépend seulement d'un ensemble fini de noms, son *support*. Pour tout ensemble de noms s on aura donc un modèle $\mathbb{P}(s)$ pour interpréter le comportement des processus de type $\mathbb{P}$ qui utilisent uniquement des noms dans s.
2. Le comportement d'un processus est préservé par le renommage injectif de ses noms. Cela implique l'existence pour tout type $\mathbb{P}$ et pour toute injection $i : s \to_{\text{inj}} s'$ d'une « traduction » $\mathbb{P}(i) : \mathbb{P}(s) \to \mathbb{P}(s')$ qui mette en relation l'espace d'interprétation $\mathbb{P}(s')$ avec celui de $\mathbb{P}(s)$.

Un cadre adapté pour formaliser cette famille uniforme d'interprétations est fourni par les catégories de foncteurs. On écrira $\mathcal{I}$ pour la catégorie des ensembles finis et fonctions injectives, et on définit $\mathbf{Lin}^{\mathcal{I}}$ comme la catégorie de foncteurs et transformations naturelles entre $\mathcal{I}$ et **Lin**. On continuera à noter les objets de $\mathbf{Lin}^{\mathcal{I}}$ par $\mathbb{P}, \mathbb{Q}, ...$

Une large partie de la structure de **Lin** peut être étendue à $\mathbf{Lin}^{\mathcal{I}}$. Notamment le tenseur $\otimes$, le produit &, la somme $+$, et l'exponentiel ! s'étendent point-à-point. D'autres constructions sont propres à $\mathbf{Lin}^{\mathcal{I}}$. La construction *objet de noms* permet de voir un ensemble de noms comme un objet de **Lin** : son action sur s est de renvoyer l'ensemble s vu comme un ordre plat.

L'union disjointe des ensembles dans $\mathcal{I}$ génère une structure monoïdale symétrique sur $\mathbf{Lin}^{\mathcal{I}}$ [Day70]. On notera $\boxtimes$ (prononcer *tenseur bizarre*) le tenseur associé. L'idée derrière cet objet est que dans un objet $\mathbb{P} \boxtimes \mathbb{Q}$ les deux composantes $\mathbb{P}$ et $\mathbb{Q}$ doivent avoir des *supports disjoints*. Pour illustrer cela on peut comparer les deux objets $\mathbb{N} \otimes \mathbb{N}$ et $\mathbb{N} \boxtimes \mathbb{N}$:

$$\mathbb{N} \otimes \mathbb{N}(s) = \{(n_1, n_2) \mid n_1, n_2 \in s\} \qquad \mathbb{N} \boxtimes \mathbb{N}(s) = \{(n_1, n_2) \mid n_1, n_2 \in s \text{ et } n_1 \neq n_2\} \ .$$

On peut construire plusieurs espaces de fonctions dans $\mathbf{Lin}^{\mathcal{I}}$. Notre attention sera dirigée vers $\mathbb{N} \multimap \mathbb{P}$ et $!\mathbb{P} \multimap \mathbb{Q}$, où $\multimap$ est l'espace de fonctions associé à $\otimes$. L'espace de fonctions associé avec $\boxtimes$, si on restreint son domaine aux objets de noms, génère le foncteur $\delta : \mathbf{Lin}^{\mathcal{I}} \to \mathbf{Lin}^{\mathcal{I}}$, nommé *foncteur de décalage*. Notations : si s est un ensemble fini, nous notons $s + 1$ l'ensemble générique obtenu en ajoutant un nouvel élément, noté $*_s$, à l'ensemble s. Étant donnée $f : s \to s'$, nous notons $f + 1 : s + 1 \to s' + 1$ la fonction qui se comporte comme f sur les éléments de s, et qui envoie $*_s$ dans $*_{s'}$. Le foncteur de décalage est alors défini sur les objets comme $\delta\mathbb{P}(s) = \mathbb{P}(s + 1)$, et sur les flèches comme $\delta\mathbb{P}(i) = \mathbb{P}(i + 1)$. Intuitivement, il étend l'ensemble des noms publics s par un nom inédit : dans cette optique, δ représente des fonctions qui accepterons uniquement un nom inédit en entrée.

Les constructions dans $\mathbf{Lin}^{\mathcal{I}}$ que nous venons de décrire permettent d'interpréter les types du langage new-HOPLA : nous allons maintenant étudier leur interprétation opérationnelle.

Le langage new-HOPLA

Le langage new-HOPLA peut être décrit comme une extension du λ-calcul simplement typé, avec un opérateur de préfixe, une somme non-déterministe et des opérateurs pour manipuler des noms générés dynamiquement. Le langage est typé, et le type d'un terme décrit la forme des chemins d'exécution que le terme peut réaliser. En suivant la structure de $\mathbf{Lin}^{\mathcal{I}}$, on obtient :

– le type de noms est désigné par $\mathbb{N}$;
– les types de processus sont définis par la grammaire :

$$\begin{array}{rcl} \mathbb{P} & ::= & \mathbf{0} \;\mid\; \mathbb{N}\otimes\mathbb{P} \;\mid\; !\mathbb{P} \;\mid\; \delta\mathbb{P} \;\mid\; \mathbb{N}\to\mathbb{P} \;\mid\; \mathbb{P}\to\mathbb{Q} \;\mid \\ & & \Sigma_{i\in I}\mathbb{P}_i \;\mid\; \mu_j\vec{P}.(\mathbb{P}_1,\ldots,\mathbb{P}_k) \;\mid\; P\ . \end{array}$$

Le type $\Sigma_{i\in I}\mathbb{P}_i$ où I est un ensemble fini sera souvent noté $i_1{:}\mathbb{P}+\cdots+i_k{:}\mathbb{P}$. On remarque la présence de types récursifs : les équations récursives pour les ordres de chemins peuvent être résolues en utilisant les systèmes de l'information [Sco82, LW84].

On supposera l'existence d'un ensemble infini, dénombrable, de *noms constants*, désignés par $a, b, \ldots$ et d'un ensemble infini, dénombrable, de *variables de noms*, désignées par $\alpha, \beta, \ldots$. Les noms, constants ou variables, sont désignés par $m, n, \ldots$.

$m, n, \ldots$	$::=$	$a, b, \ldots$	noms constants
	$\mid$	$\alpha, \beta, \ldots$	variables de noms

On supposera aussi l'existence d'un ensemble infini, dénombrable, de *variables de processus*, désignées par $x, y, \ldots$.

Les *termes* sont définis par la grammaire suivante :

t, u, v	$::=$	$\mathbf{0}$	$!t$	processus inactif et action anonyme
	$\mid$	$n \cdot t$	$\pi_n t$	tenseur et projection
	$\mid$	$\lambda x.t$	tu	abstraction de processus et application
	$\mid$	$\lambda\alpha.t$	tn	abstraction de nom et application
	$\mid$	$new\alpha.t$	$t[n]$	abstraction de nom inédit et application
	$\mid$	$recx.t$	x	définition récursive et variable de processus
	$\mid$	$i{:}t$	$\pi_i t$	injection et projection
	$\mid$	$\Sigma_{i\in I} t_i$		somme
	$\mid$	$\Sigma_{\alpha\in\mathbb{N}} t$		somme sur les noms
	$\mid$	$[t > p(x) \Rightarrow u]$		filtrage

Le *support* d'un terme, noté $\mathsf{n}(t)$, est défini comme l'ensemble de ses nom constants.

Le comportement des termes t du langage est décrit par la relation de transition reportée figure 1.14. Celle-ci utilise les transitions de forme $s \vdash t \xrightarrow{p} t'$, où s est un ensemble fini de noms constants tels que $\mathrm{fn}(t) \subseteq s$ et p est une action (ou motif de chemin). L'ensemble s représente l'ensemble des noms connus dans l'univers où évolue t. La relation ci-dessus sera donc lue ainsi : « dans un état tel que les noms en s sont susceptibles d'être connus par le terme t et par son environnement, le terme t peut réaliser l'action p et devenir t' ».

$$\frac{}{!\mathbb{P}; s \vdash !t \xrightarrow{!x\ (x)} t} \qquad \frac{\mathbb{P}; s \vdash t \xrightarrow{p} t' \quad a \in s}{\mathbb{N} \otimes \mathbb{P}; s \vdash a \cdot t \xrightarrow{a \cdot p} t'} \qquad \frac{\mathbb{N} \otimes \mathbb{P}; s \vdash t \xrightarrow{a \cdot p} t'}{\mathbb{P}; s \vdash \pi_a t \xrightarrow{p} t'}$$

$$\frac{\mathbb{P}; s \vdash t_i \xrightarrow{p\ (x)} t'}{\mathbb{P}; s \vdash \Sigma_{i \in I} t_i \xrightarrow{p\ (x)} t'} \qquad \frac{\mathbb{P}; s \vdash t[a/\alpha] \xrightarrow{p\ (x)} u \quad a \in s}{\mathbb{P}; s \vdash \Sigma_{\alpha \in \mathbb{N}} t \xrightarrow{p\ (x)} u} \qquad \frac{\mathbb{P}; s \vdash t[recy.t/y] \xrightarrow{p\ (x)} u}{\mathbb{P}; s \vdash recy.t \xrightarrow{p\ (x)} u}$$

$$\frac{\mathbb{P}_i; s \vdash t \xrightarrow{p\ (x)} t'}{\Sigma_{i \in I} \mathbb{P}_i; s \vdash i{:}t \xrightarrow{i:p\ (x)} t'} \qquad \frac{\Sigma_{i \in I} \mathbb{P}_i; s \vdash t \xrightarrow{i:p\ (x)} t'}{\mathbb{P}_i; s \vdash \pi_i t \xrightarrow{p\ (x)} t'} \qquad \frac{\mathbb{Q}; s \vdash t[u/x] \xrightarrow{p\ (x)} v \quad s \vdash u : \mathbb{P}}{\mathbb{P} \to \mathbb{Q}; s \vdash \lambda x.t \xrightarrow{u \mapsto p\ (x)} v}$$

$$\frac{\mathbb{P} \to \mathbb{Q}; s \vdash t \xrightarrow{u \mapsto p\ (x)} v}{\mathbb{Q}; s \vdash tu \xrightarrow{p\ (x)} v} \qquad \frac{\mathbb{P}; s \vdash t[a/\alpha] \xrightarrow{p\ (x)} v \quad a \in s}{\mathbb{N} \to \mathbb{P}; s \vdash \lambda\alpha.t \xrightarrow{a \mapsto p\ (x)} v} \qquad \frac{\mathbb{N} \to \mathbb{P}; s \vdash t \xrightarrow{a \mapsto p\ (x)} v}{\mathbb{P}; s \vdash ta \xrightarrow{p\ (x)} v}$$

$$\frac{\mathbb{P}; s \dot\cup \{a\} \vdash t[a/\alpha] \xrightarrow{p[a/\alpha]\ (x)} u[a/\alpha]}{\delta\mathbb{P}; s \vdash new\alpha.t \xrightarrow{new\alpha.p[x'[\alpha]/x]\ (x')} new\alpha.u} \qquad \frac{\delta\mathbb{P}; s \vdash t \xrightarrow{new\alpha.p[x'[\alpha]/x]\ (x')} u}{\mathbb{P}; s \dot\cup \{a\} \vdash t[a] \xrightarrow{p[a/\alpha]\ (x)} u[a]}$$

$$\frac{\mathbb{P}; s \vdash t \xrightarrow{p\ (x)} t' \quad \mathbb{Q}; s \vdash u[t'/x] \xrightarrow{q\ (x')} v}{\mathbb{Q}; s \vdash [t > p(x) \Rightarrow u] \xrightarrow{q\ (x')} v}$$

Dans la règle pour l'abstraction d'un nom inédit, les conditions $a \notin \mathsf{n}(p)$ et $a \notin \mathsf{n}(u)$ doivent être vérifiées.

Fig. 1.14: new-HOPLA : système de transitions

Les *actions* (ou *motif de chemin*) sont définies par la grammaire :

$$p, q, r \quad ::= \quad x \;\mid\; !p \;\mid\; n \cdot p \;\mid\; i{:}p \;\mid\; new\alpha.p \;\mid\; n \mapsto p \;\mid\; u \mapsto p \;\mid\; p[n]\ .$$

Comme on le verra quand on parlera du typage de new-HOPLA, les actions bien typées présentent uniquement une action anonyme !, suivie par une variable de reprise : si un terme réalise une action, alors la variable de reprise enregistre la continuation du terme, et le type d'une action met en relation le type d'un terme avec les types de ses continuations.

On peut maintenant illustrer l'interprétation opérationnelle des types et des termes qui les habitent.

Le type vide, $\mathbf{0}$, représente l'ensemble vide de chemins d'exécution : l'unique terme de ce type est le terme inactif, $\mathbf{0}$, terme qui n'exécute aucune action.

Le type préfixe, $!\mathbb{P}$, décrit des chemins d'exécution dans lesquels une première action anonyme, qu'on désignera par ! (prononcer *bip*), est réalisée avant de se réduire à un chemin d'exécution de forme $\mathbb{P}$. Le type préfixe est associé à une opération de préfixe, qui envoie un processus t de type $\mathbb{P}$ dans le terme $!t$ de type $!\mathbb{P}$. Ainsi, le terme $!t$ réalise l'action anonyme ! et devient t. L'action anonyme est l'action visible élémentaire sur laquelle on construit des actions plus complexes et plus informatives.

Le type somme, construit à partir d'une famille de types $(\mathbb{P}_i)_{i\in I}$, est désigné par $\Sigma_{i\in I}\mathbb{P}_i$. Il est associé d'une part aux injections, qui donnent un terme $i : t$ de type $\Sigma_{i\in I}\mathbb{P}_i$ à partir d'un terme t de type $\mathbb{P}_i$, et d'autre part aux projections, qui à l'opposé donnent un terme $\pi_i t$ de type $\mathbb{P}_i$ à partir d'un terme t de type somme décrit ci-dessus. D'un point de vue opérationnel, si t réalise une action p, alors $i : t$ réalise la même action étiquetée par i. La projection $\pi_i t$ filtre les actions p réalisées par t : si p a la forme $j : p'$, et $j = i$, alors $\pi_i t$ réalise p'. Sinon, l'action est bloquée. La sémantique identifie donc les termes $\pi_i(i{:}t)$ et t, et, si $i \neq j$, les termes $\pi_i(j{:}t)$ et $\mathbf{0}$.

Le type tenseur $\mathbb{N}\otimes\mathbb{P}$ désigne l'action d'étiqueter avec un nom un processus de type $\mathbb{P}$. Le constructeur donne un terme $n \cdot t$ de type $\mathbb{N}\otimes\mathbb{P}$ à partir d'un terme t de type p et d'un nom n. De même, tous les chemins d'exécution de t sont étiquetés avec n. Le destructeur, désigné par $\pi_n t$, vérifie si une action réalisée par t est étiquetée par n. Si oui, il efface l'étiquette. Si non, il bloque l'action.

Le type de l'espace des fonctions $\mathbb{P} \to \mathbb{Q}$ est interprété comme $!\mathbb{P} \multimap \mathbb{Q}$ dans $\mathbf{Lin}^{\mathcal{I}}$ et est analogue à l'espace des fonctions dans le λ-calcul simplement typé. Un processus $\lambda x.t$ de type $\mathbb{P} \to \mathbb{Q}$ reçoit un processus u qui se comporte comme $\mathbb{P}$, le lie à la variable x, et se comporte ensuite comme $\mathbb{Q}$. En d'autres termes, si $t[u/x]$ réalise une action p et continue comme t', alors $\lambda x.t$ réalise une action $u \mapsto p$ (qui se lit « si appliqué à u on réalise p »), et continue comme t'. L'application de processus est notée tu, pour $t : \mathbb{P} \to \mathbb{Q}$ et $u : \mathbb{P}$. Informellement, son comportement peut être décrit comme tu réalise p et continue comme t' si t appliqué à u réalise p et continue comme t'. Le type de l'espace de fonctions $\mathbb{N} \to \mathbb{Q}$ est interprété comme $\mathbb{N} \multimap \mathbb{Q}$ dans $\mathbf{Lin}^{\mathcal{I}}$. Le comportement des termes $\lambda\alpha.t$ et $t\alpha$ est analogue à celui que l'on vient de décrire.

Le type $\delta\mathbb{P}$ représente un espace de fonctions dont le domaine est $\mathbb{N}$ et le codomaine en est $\mathbb{P}$, soumis à la contrainte que l'argument doit être un nom *inédit*. Étant donné un terme t de type $\mathbb{P}$, le terme $new\alpha.t$ sera typé comme $\delta\mathbb{P}$: opérationnellement il reçoit en entrée un nom inédit, le lie à α, et continue ensuite comme t. L'action correspondante est $new\alpha.p$, qui doit être interprétée comme « si appliqué à un nom inédit, il exécute p » : l'identité du nom est sans importance, pourvu qu'il soit inédit. La règle opérationnelle correspondante force cette condition en étendant l'univers des noms connus s par un nouveau nom. Le destructeur, nommé application de nom inédit, est écrit $t[n]$, où $t : \delta\mathbb{P}$ et n est un nom inédit pour t.

Le terme de filtrage $[t > p(x) \Rightarrow u]$ est fondamental pour l'expressivité de new-HOPLA. Ce terme filtre une action réalisée par t contre le motif p (on rappelle qu'actions et motifs sont représentés par la même syntaxe) : si les deux se correspondent, la continuation de t est passée à u grâce à la variable x. Par conséquent, si le terme t a pour type $\mathbb{P}$, alors l'action p est un chemin de type $\mathbb{P}$ avec une variable de reprise de type $\mathbb{R}$, et u en général contient x. Ce terme joue le rôle de destructeur du préfixe : notamment les processus $[!t > !x(x) \Rightarrow u]$ et $u[t/x]$ sont identifiés (voir page 164 pour d'autres exemples).

Le système de types Le système de types de new-HOPLA demande une attention particulière dans la gestion de noms inédits. Par exemple, considérons le terme $t = t'[\alpha]$. L'application du nom α au terme t' avec pour contrainte que α soit remplacée uniquement par des noms inédits pour t'. Considérons aussi le contexte $C[-] = \lambda\alpha.-$: dans le terme $C[t] = \lambda\alpha.(t'[\alpha])$ la variable α est abstraite par une lambda abstraction et pendant l'exécution peut être remplacée par un nom arbitraire, notamment un nom qui n'est pas inédit pour t'. On rencontre le même problème avec les contextes de la forme $C[-] = \Sigma_{\alpha\in\mathbb{N}}-$. Une autre difficulté surgit si une variable de processus x est libre dans t : un contexte comme $C[-] = \lambda x.-$ pourrait remplacer x par un terme u arbitraire. Mais α pourrait être remplacée par un nom qui figure dans le support de u, et rien ne garantit que le nom qui remplace α soit inédit pour le terme $t[u/x]$.

Un des rôles du système de types est d'assurer statiquement que des variables de noms supposées inédites ne soient jamais rempolacées par des noms déjà connus. Pour imposer ces restrictions, le contexte de typage ne contient pas seulement des hypothèses sur les types associés aux variables, comme α:$\mathbb{N}$ ou x:$\mathbb{P}$, mais aussi des *hypothèses de fraîcheur* entre elles, comme (α, β) ou (α, x). La présence d'une contrainte (α, β) révèle que les variables de noms α et β ont été supposées différentes et ne devront jamais être identifiées. Une contrainte (α, x) enregistre le fait que dans tout environnement le nom qui remplace α doit être inédit pour le terme qui remplace x.

À l'aide de ces informations auxiliaires, le système de types suppose qu'on peut abstraire une variable avec une lambda abstraction, ou une somme sur les noms, seulement si aucune hypothèse de fraîcheur n'a été faite sur elle.

Notations : on écrit $d\backslash\alpha$ pour l'ensemble de distinctions obtenu en effaçant tous les couples qui contiennent α de d ; l'ordre des variables dans une paire (α, β) n'est pas important.

Le système de types de new-HOPLA peut être défini avec des jugements de la forme

$$A; \Gamma; d \vdash t : \mathbb{P}$$

où

- $A \equiv \alpha_1$:$\mathbb{N}, \ldots, \alpha_k$:$\mathbb{N}$: ensemble de variables de noms ;
- $\Gamma \equiv x_1$:$\mathbb{P}_1, \ldots, x_k$:$\mathbb{P}_k$: fonction partielle entre variables de processus et types ;
- d : ensemble de couples $(\alpha, x) \in A \times \Gamma$, et $(\alpha, \beta) \in A \times A$, qui enregistre les hypothèses de fraîcheur.

Les actions sont typées en suivant les mêmes procédés, même si les jugements de typage signalent explicitement la variable de reprise :

$$A; \Gamma; d; ; x{:}\mathbb{R} \vdash p : \mathbb{P} \ .$$

L'environnement de typage suit les mêmes intuitions que ci-dessus. La variable x est la variable de reprise du motif p, et son type est $\mathbb{R}$: le jugement est syntaxiquement différent pour bien marquer que la variable de reprise n'est pas une variable libre de l'action. Le système de type de new-HOPLA est reporté figure 1.15 et figure 1.16.

La règle responsable de la génération des hypothèses de fraîcheur est la règle de l'application des nouveaux noms. Si un terme t a été typé dans l'environnement $A; \Gamma; d$ et α est une nouvelle variable de noms (c'est-à-dire $\alpha \notin A$), alors le terme $t[\alpha]$ est bien typé,

$$\frac{}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ !x : !\mathbb{R}} \qquad \frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ \alpha \cdot p : \mathbb{N} \otimes \mathbb{P}}\alpha \in A$$

$$\frac{\alpha{:}\mathbb{N}, A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}}{A;\Gamma;(d \setminus \alpha);;x'{:}\delta\mathbb{R} \Vdash\!\vdash\ new\alpha.p[x'[\alpha]/x] : \delta\mathbb{P}} \qquad \frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ \alpha \mapsto p : \mathbb{P}}\alpha \in A$$

$$\frac{A;\Gamma;d \vdash u : \mathbb{Q} \qquad A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ u \mapsto p : \mathbb{Q} \to \mathbb{P}} \qquad \frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}_j \quad j \in I}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ (j{:}p) : \Sigma_{i\in I}\mathbb{P}_i}$$

$$\frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ t : \mathbb{P}_j[\mu\vec{P}.\vec{\mathbb{P}}/\vec{P}]}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ t : \mu_j P : \vec{\mathbb{P}}} \qquad \frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}}{A';\Gamma';d';;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}} \begin{array}{l} A \subseteq A' \\ \Gamma \subseteq \Gamma' \\ d \subseteq d' \end{array}$$

Fig. 1.15: new-HOPLA : règles de typage pour les actions

à la condition que la variable α soit remplacée seulement par des noms inédits pour tous les termes qui remplacent les variables qui peuvent apparaître dans t ; cela est obtenu en ajoutant l'ensemble de distinctions $\{\alpha\} \times (\Gamma \cup A)$ à d.[1] La règle pour le filtrage modifie aussi l'ensemble de distinctions. D'un point de vue opérationnel, l'opérateur de filtrage substitue un sous-terme de t, dont les noms sont contenus dans A', dans x. De même, la règle de typage vérifie qu'aucun nom dans A' ne figure parmi les variables qui ont été supposées inédites pour x ; ensuite elle considère le terme $u[t'/x]$, où t' est un sous-terme de t : une variable de noms $\alpha \in A$ supposée différente de x dans le typage de u, doit maintenant être supposée différente de toutes les variables libres de t'. Cela justifie l'ensemble de distinctions $\{\{\alpha\} \times (A' \cup \Gamma') \mid (\alpha, x) \in d\}$. À part cela, le système de typage est plutôt standard et suit les grandes lignes des systèmes de typage du λ-calcul simplement typé.

On peut donner un aperçu de l'interprétation de new-HOPLA dans $\mathbf{Lin}^{\mathcal{I}}$. Un environnement de typage $\alpha_1{:}\mathbb{N}, \ldots, \alpha_n{:}\mathbb{N}; x_1{:}\mathbb{P}_1, \ldots, x_n{:}\mathbb{P}_n; \emptyset$ représente l'objet

$$[\![\, \alpha_1{:}\mathbb{N}, \ldots, \alpha_n{:}\mathbb{N}; x_1{:}\mathbb{P}_1, \ldots, x_n{:}\mathbb{P}_n; \emptyset \,]\!] : \mathbb{N} \otimes \cdots \otimes \mathbb{N} \otimes !\mathbb{P}_1 \otimes \cdots \otimes !\mathbb{P}_n$$

dans $\mathbf{Lin}^{\mathcal{I}}$. Cela montre que les dénotations des environnements, étant donné un univers s, sont des tuples. Si l'ensemble des hypothèses de fraîcheur n'est pas vide, alors la dénotation d'un environnement est un sous-ensemble de ces tuples, formé par les tuples qui vérifient les hypothèses de fraîcheur. Alors, un jugement de typage $A;\Gamma;d \vdash t : \mathbb{P}$ représente une fonction linéaire

$$[\![\, A;\Gamma;d \vdash t : \mathbb{P} \,]\!] : [\![\, A;\Gamma;d \,]\!] \to \mathbb{P} \ ,$$

et un jugement pour un motif de chemins $A;\Gamma;d;;x{:}\mathbb{R} \vdash p : \mathbb{P}$ représente une fonction linéaire

$$[\![\, A;\Gamma;d;;x{:}\mathbb{R} \vdash p : \mathbb{P} \,]\!] : [\![\, A;\Gamma;d \,]\!] \otimes \mathbb{R}^{\mathrm{op}} \to \mathbb{P}^{\mathrm{op}} \ .$$

[1] Pour simplifier la notation, quand nous écrivons $\{\alpha\} \times (\Gamma \cup A)$ nous négligeons implicitement toutes les informations concernant les types de variables dans Γ.

$$\frac{}{A;\Gamma;d\vdash\emptyset:\mathbb{P}} \qquad \frac{}{A;x{:}\mathbb{P},\Gamma;d\vdash x:\mathbb{P}} \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}}{A';\Gamma';d'\vdash t:\mathbb{P}}\ \begin{array}{l}A\subseteq A'\\ \Gamma\subseteq\Gamma'\\ d\subseteq d'\end{array}$$

$$\frac{\alpha{:}\mathbb{N},A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash\Sigma_{\alpha\in\mathbb{N}}t:\mathbb{P}}\alpha\notin d \qquad \frac{\alpha{:}\mathbb{N},A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash\lambda\alpha.t:\mathbb{N}\to\mathbb{P}}\alpha\notin d$$

$$\frac{A;x{:}\mathbb{Q},\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash\lambda x.t:\mathbb{Q}\to\mathbb{P}}x\notin d \qquad \frac{A;x{:}\mathbb{P},\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash rec x.t:\mathbb{P}}x\notin d$$

$$\frac{\alpha{:}\mathbb{N},A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;(d\setminus\alpha)\vdash new\alpha.t:\delta\mathbb{P}} \qquad \frac{A;\Gamma;d\vdash t:\delta\mathbb{P}}{\alpha{:}\mathbb{N},A;\Gamma;d\cup(\{\alpha\}\times(\Gamma\cup A))\vdash t[\alpha]:\mathbb{P}}$$

$$\frac{A;\Gamma;d\vdash t:\mathbb{N}\to\mathbb{P}}{A;\Gamma;d\vdash t\alpha:\mathbb{P}}\alpha\in A \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}\to\mathbb{Q}\quad A;\Gamma;d\vdash u:\mathbb{P}}{A;\Gamma;d\vdash tu:\mathbb{Q}}$$

$$\frac{A;\Gamma;d\vdash t_i:\mathbb{P}\quad\forall i\in I}{A;\Gamma;d\vdash\Sigma_{i\in I}t_i:\mathbb{P}} \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}_i}{A;\Gamma;d\vdash i{:}t:\Sigma_{i\in I}\mathbb{P}_i} \qquad \frac{A;\Gamma;d\vdash t:\Sigma_{i\in I}\mathbb{P}_i}{A;\Gamma;d\vdash\pi_i t:\mathbb{P}_i} \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash !t:!\mathbb{P}}$$

$$\frac{A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash\alpha\cdot t:\mathbb{N}\otimes\mathbb{P}}\alpha\in A \qquad \frac{A;\Gamma;d\vdash t:\mathbb{N}\otimes\mathbb{P}}{A;\Gamma;d\vdash\pi_\alpha t:\mathbb{P}}\alpha\in A \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}_j[\mu\vec{P}.\vec{\mathbb{P}}/\vec{P}]}{A;\Gamma;d\vdash t:\mu_j P:\vec{\mathbb{P}}}$$

$$\frac{A';\Gamma';d'\vdash t:\mathbb{P} \qquad A';\Gamma';d';;x{:}\mathbb{R}\Vdash p:\mathbb{P} \qquad A;x{:}\mathbb{R},\Gamma;d\vdash u:\mathbb{Q}}{A\cup A';\Gamma\cup\Gamma';\overline{d}\vdash[t>p(x)\Rightarrow u]:\mathbb{Q}}A'\cap\{\alpha\mid(\alpha,x)\in d\}=\emptyset$$

$$\text{où } \overline{d}=(d\setminus x)\cup d'\cup\{\{\alpha\}\times(A'\cup\Gamma')\mid(\alpha,x)\in d\}$$

Fig. 1.16: new-HOPLA : règles de typage pour les processus

Le système de typage suppose que les termes ne contiennent pas de noms constants. Cela permet d'éviter des complications liées à un système de types qui gère à la fois variables de noms et noms constants. Nous écrivons $s \vdash t : \mathbb{P}$ quand il existe un jugement $A; \emptyset; d \vdash \sigma t' : \mathbb{P}$ et une substitution σ pour A qui respecte les hypothèses de fraîcheur d telle que $t = \sigma t'$. De même pour les motifs.

Proposition 5.2.3, page 160 *Le jugement $s \vdash t : \mathbb{P}$ est valide si et seulement s'il existe un jugement canonique $A; \emptyset; \{(\alpha, \beta) \mid \alpha \neq \beta\} \vdash t' : \mathbb{P}$, où la substitution σ est une bijection entre les variables de noms et les noms constants, et $t = \sigma t'$.*

Le système de typage vérifie des propriétés attendues :

Lemme 5.2.4, page 160 (Substitution) *Soient $A'; \Gamma'; d' \vdash t : \mathbb{Q}$ et $A; x{:}\mathbb{Q}, \Gamma; d \vdash u : \mathbb{P}$ deux jugements valides, où $\Gamma \cup \Gamma'$ est bien défini et où $A' \cap \{\alpha \mid (\alpha, x) \in d\} = \emptyset$. Alors,*

$$A \cup A'; \Gamma \cup \Gamma'; \overline{d} \vdash u[t/x] : \mathbb{P}$$

où $\overline{d} = (d \setminus x) \cup d' \cup \{\{\alpha\} \times (A' \cup \Gamma') \mid (\alpha, x) \in d\}$.

Théorème 5.2.7, page 167 (Les transitions préservent les types) *Si $s \vdash t : \mathbb{P}$ et $s; ; x{:}\mathbb{Q} \Vdash p : \mathbb{P}$ et $\mathbb{P}; s \vdash t \xrightarrow{p\ (x)} t'$, alors $s \vdash t' : \mathbb{Q}$.*

La théorie algébrique de new-HOPLA Il est naturel d'utiliser le système de transitions pour définir une bisimulation : cette équivalence se révèle une sémantique de new-HOPLA très intéressante.

Le langage est typé, et il est naturel de considérer comme équivalents uniquement les termes qui ont le même type. Pour cela, on dit qu'une relation $\mathcal{R}$ entre jugements de typage *respecte les types* si chaque fois que $\mathcal{R}$ met en relation $E_1 \vdash t_1 : \mathbb{P}_1$ et $E_2 \vdash t_2 : \mathbb{P}_2$, on a $E_1 = E_2$ et $\mathbb{P}_1 = \mathbb{P}_2$. Du moment que nous sommes surtout intéressés par des relations entre des termes fermés, nous écrirons $s \vdash t \ \mathcal{R}\ u : \mathbb{P}$ comme abréviation pour $(s \vdash t : \mathbb{P}, s \vdash q : \mathbb{P}) \in \mathcal{R}$.

Définition 5.3.1, page 169 (Bisimilarité) *Une relation qui respecte les types $\mathcal{R}$ est une* bisimulation *si*

1. *$s \vdash t \ \mathcal{R}\ u : \mathbb{P}$ et $s' \vdash t \xrightarrow{p(x)} t'$ pour $s' \supseteq s$ impliquent qu'il existe un terme u' tel que $s' \vdash u \xrightarrow{p(x)} u'$ et $s' \vdash t' \ \mathcal{R}\ u' : \mathbb{R}$;*
2. *$s \vdash t \ \mathcal{R}\ u : \mathbb{P}$ et $s' \vdash u \xrightarrow{p(x)} u'$ pour $s' \supseteq s$ impliquent qu'il existe un terme t' tel que $s' \vdash t \xrightarrow{p(x)} t'$ et $s' \vdash t' \ \mathcal{R}\ u' : \mathbb{R}$;*

où $\mathbb{R}$ est le type de la variable de reprise x dans p. On définit la bisimilarité*, notée $\sim$, comme la plus grande bisimulation.*

Deux termes fermés t et q sont bisimilaires si existent s et $\mathbb{P}$ tels que $s \vdash t \sim q : \mathbb{P}$.

On souligne l'importance de la quantification universelle sur tous les ensembles de noms $s' \supseteq s$. En effet, l'une des propriétés du STE est que le support d'un terme ne change pas au cours de son exécution : l'existence d'univers qui contiennent d'autres noms doit donc être

explicitement prise en compte dans la bisimulation, sinon la relation resultante considererait comme équivalents les termes

$$\{a\} \vdash \lambda\alpha.[\alpha!\mathbf{0} > a!x \Rightarrow !\mathbf{0}] : \mathbb{N} \otimes !\mathbf{0} \qquad \text{et} \qquad \{a\} \vdash \lambda\alpha.!\mathbf{0} : \mathbb{N} \otimes !\mathbf{0}$$

Or, les deux termes ci-dessus se comportent différemment dans un univers où ils existent d'autres noms que a. Cette quantification est analogue à la quantification sur les mondes possibles des travaux sur les modèles de Kripke du λ-calcul.

La propriété fondamentale de la bisimilarité est qu'elle est une congruence par rapport aux contextes bien typés.

Théorème 5.3.14, page 176 *La bisimilarité $\sim$ est une congruence.*

Démonstration [Esquisse] La preuve utilise une variante de la méthode de Howe [How96] adaptée par Gordon au cadre typé [Gor95]. Pour cela nous introduisons une relation auxiliaire, nommée *candidat de précongruence*, qui, par construction, contient l'extension ouverte de la bisimilarité et est préservée par les opérateurs du langage. On prouve ensuite que le candidat de précongruence est une simulation ; il s'ensuit que la fermeture transitive et réflexive du candidat de précongruence est une bisimulation et la conclusion va de soi. □

La théorie algébrique de new-HOPLA est extrêmement riche, comme le montrent les équations page 177 et suivantes.

Exemples Nous étudions l'expressivité de new-HOPLA, notamment nous montrons comment il peut être utilisé pour donner une sémantique à des algèbres de processus bien connues. À titre d'exemple nous détaillons ici le codage de la sémantique tardive du π-calcul.

Tout d'abord nous définissons un type produit $\mathbb{P}\,\&\,\mathbb{Q}$ très utile comme $1{:}\mathbb{P} + 2{:}\mathbb{Q}$. Les projections sont $\mathit{fst}(t) = \pi_1(t)$ et $\mathit{snd}(t) = \pi_2(t)$, et le *pairing* est défini comme $(t, u) = 1{:}t + 2{:}u$. Pour les actions $(p, -) = 1{:}p$, $(-, q) = 2{:}q$. Il est simple de vérifier que $s \vdash \mathit{fst}(t, u) \sim t : \mathbb{P}$, que $s \vdash \mathit{snd}(t, u) \sim u : \mathbb{Q}$, et que $s \vdash (\mathit{fst}(t, u), \mathit{snd}(t, u)) \sim (t, u) : \mathbb{P}\,\&\,\mathbb{Q}$, pour tout $s \supseteq \mathsf{n}(t) \cup \mathsf{n}(u)$.

En utilisant les conventions habituelles sur noms et variables, les termes de π-calcul sont définis par la grammaire suivante :

$$P \;::=\; \mathbf{0} \;\mid\; P \mid P \;\mid\; (\boldsymbol{\nu}\alpha)P \;\mid\; \overline{n}m.P \;\mid\; n(\alpha).P\ .$$

(Une description du système de transitions étiquetées en style tardif et la définition de la bisimulation forte sont reportées page 179).

Le codage de la sémantique tardive est reporté figure 1.17. Les termes du π-calcul sont traduits en new-HOPLA par la fonction $[\![-]\!]$ définie par induction structurelle. L'opération de composition parallèle $||$: $\mathbb{P}\&\mathbb{P} \to \mathbb{P}$ (on utilise la notation infixe) implante la loi d'expansion tardive du π-calcul. Informellement, la restriction Res : $\delta\mathbb{P} \to \mathbb{P}$ pousse les restrictions vers l'intérieur des processus : les cinq termes correspondent aux cinq équations ci-dessous :

$$\begin{aligned}
(\boldsymbol{\nu}\alpha)\tau.P &\sim_l \tau.(\boldsymbol{\nu}\alpha)P \\
(\boldsymbol{\nu}\alpha)\overline{m}n.P &\sim_l \overline{m}n.(\boldsymbol{\nu}\alpha)P && \text{si } \alpha \neq m, n \\
(\boldsymbol{\nu}\alpha)\overline{m}\alpha.P &\sim_l \overline{m}(\alpha).P && \text{si } \alpha \neq m \\
(\boldsymbol{\nu}\alpha)\overline{m}(\beta).P &\sim_l \overline{m}(\beta).(\boldsymbol{\nu}\alpha)P && \text{si } \alpha \neq m \\
(\boldsymbol{\nu}\alpha)m\beta.P &\sim_l m\beta.(\boldsymbol{\nu}\alpha)P && \text{si } \alpha \neq m
\end{aligned}$$

$$\mathbb{P} \;=\; \tau{:}!\mathbb{P} \;+\; \mathsf{out}{:}\mathbb{N}\otimes\mathbb{N}\otimes!\mathbb{P} \;+\; \mathsf{bout}{:}\mathbb{N}\otimes!(\delta\mathbb{P}) \;+\; \mathsf{inp}{:}\mathbb{N}\otimes!(\mathbb{N}\to\mathbb{P})$$

$$\begin{array}{rcl}
[\![\,\mathbf{0}\,]\!] & = & \mathbf{0} \\
[\![\,\overline{\alpha}\beta.P\,]\!] & = & \mathsf{out}{:}\alpha\cdot\beta\cdot!\,[\![\,P\,]\!] \\
[\![\,\alpha(\beta).P\,]\!] & = & \mathsf{inp}{:}\alpha\cdot!(\lambda\beta.[\![\,P\,]\!]) \\
[\![\,(\boldsymbol{\nu}\alpha)P\,]\!] & = & \mathit{Res}\,(\mathit{new}\alpha.[\![\,P\,]\!]) \\
[\![\,P\mid Q\,]\!] & = & [\![\,P\,]\!] \;||\; [\![\,Q\,]\!]
\end{array}$$

$$\begin{array}{rcl}
\mathit{Res} & : & \delta\mathbb{P}\to\mathbb{P} \\
\mathit{Res}\;t & = & [t > \mathit{new}\alpha.\tau{:}!(x[\alpha])) \Rightarrow \tau{:}!\mathit{Res}\,x] \\
& + & \Sigma_{\beta\in\mathbb{N}}\Sigma_{\gamma\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{out}{:}\beta\cdot\gamma\cdot!(x[\alpha]) \Rightarrow \mathsf{out}{:}\beta\cdot\gamma\cdot!\mathit{Res}\,x] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{out}{:}\beta\cdot\alpha\cdot!(x[\alpha]) \Rightarrow \mathsf{bout}{:}\beta\cdot!x] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{bout}{:}\beta\cdot!(x[\alpha]) \Rightarrow \mathsf{bout}{:}\beta\cdot!\mathit{new}\gamma\cdot\mathit{Res}\,(\mathit{new}\eta.x[\eta][\gamma])] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{inp}{:}\beta\cdot!(x[\alpha]) \Rightarrow \mathsf{inp}{:}\beta\cdot!\lambda\gamma.\mathit{Res}\,(\mathit{new}\eta.x[\eta](\gamma))]
\end{array}$$

$$\begin{array}{rcl}
|| & : & \mathbb{P}\&\mathbb{P}\to\mathbb{P} \\
t \;||\; u & = & [t > \tau{:}!x \Rightarrow \tau{:}!(x \;||\; u)] \\
& + & \Sigma_{\beta\in\mathbb{N}}\Sigma_{\gamma\in\mathbb{N}}[t > \mathsf{out}{:}(\beta\cdot\gamma\cdot!x) \Rightarrow [u > \mathsf{inp}{:}(\beta\cdot!y) \Rightarrow \tau{:}!(x \;||\; y\gamma)]] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{bout}{:}(\beta\cdot!x) \Rightarrow [u > \mathsf{inp}{:}(\beta\cdot!y) \Rightarrow \tau{:}!\mathit{Res}\,(\mathit{new}\eta.(x[\eta] \;||\; y\eta))]] \\
& + & \Sigma_{\beta\in\mathbb{N}}\Sigma_{\gamma\in\mathbb{N}}[t > \mathsf{out}{:}\beta\cdot\gamma\cdot!x \Rightarrow \mathsf{out}{:}\beta\cdot\gamma\cdot!(x \;||\; u)] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{bout}{:}\beta\cdot!x \Rightarrow \mathsf{bout}{:}\beta\cdot!\mathit{new}\eta.(x[\eta] \;||\; u)] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{inp}{:}\beta\cdot!x \Rightarrow \mathsf{inp}{:}\beta\cdot!\lambda\eta.(x(\eta) \;||\; u)] \\
& + & \text{mêmes cas avec } t \text{ et } u \text{ échangés.}
\end{array}$$

où η ne figure pas dans u.

Fig. 1.17: π-calcul : codage de la sémantique tardive

où $\overline{m}(\alpha)$ est une abréviation de $(\boldsymbol{\nu}\alpha)\overline{m}\alpha$. La fonction *Res* implante aussi l'equation $(\boldsymbol{\nu}\alpha)P \sim \mathbf{0}$ si aucun des cas ci-dessus ne s'applique.

Il y a une correspondance stricte entre les actions exécutées par un processus de π-calcul fermé et les actions réalisées par son codage.

Théorème 5.4.3, page 180 *Soit P un terme fermé de π-calcul. Alors, (i) $P \xrightarrow{\tau}_l P'$ implique que $\mathsf{n}(P) \vdash [\![P]\!] \xrightarrow{\tau:!}\sim [\![P']\!]$; (ii) $\mathsf{n}(P) \vdash [\![P]\!] \xrightarrow{\tau:!} t$ implique que $P \xrightarrow{\tau}_l P'$ et $t \sim [\![P']\!]$*

De plus, le codage préserve et reflète la bisimulation forte.

Théorème 5.4.4, page 182 et théorème 5.4.5, page 185 *Soient P et Q deux processus fermés de π-calcul. Alors $P \sim_l Q$ si et seulement si $\mathsf{n}(P, Q) \vdash [\![P]\!] \sim [\![Q]\!] : \mathbb{P}$.*

Nous proposons aussi des codages de la sémantique précoce du π-calcul et de la sémantique tardive du π-calcul polyadique (page 187 et suivantes) : dans tous ces cas, nous conjecturons que le codage est complètement adéquat par rapport à la bisimulation forte. Ensuite, nous nous concentrons sur l'ordre supérieur, et nous proposons un codage des Ambients Mobiles (page 192). Le codage préserve et reflète les réductions des termes des Ambients Mobiles, mais il donne lieu à une équivalence qui est contenue dans l'équivalence contextuelle naturelle (on parle ici d'équivalences fortes), mais qui ne coïncide pas avec elle. Déterminer s'il existe un codage des Ambients Mobiles (comme de HOπ, ou de CCS avec communication de processus) dans new-HOPLA, qui puisse réconcilier la bisimulation naturelle déterminée par le codage avec la bisimulation contextuelle de Sangiorgi [San94], est un problème ouvert.

Travaux futurs

En guise de conclusion de cette excursion dans les résultats de cette thèse, nous montrons comment cette thèse ouvre plusieurs directions de recherche.

Nos résultats sur les théories à la base du Seal Calcul et des Ambients Mobiles nous donnent non seulement des fondations solides pour obtenir une meilleure compréhension de ces calculs, mais aussi des outils pratiques pour prouver qu'une implantation respecte sa spécification, ou qu'une machine abstraite est correcte. Une direction de recherche consiste à raffiner nos techniques pour des domaines spécifiques : notamment, nous croyons que des caractérisations étiquetées de la congruence barbue dans un cadre typé peuvent être obtenues pour les Ambients Mobiles en suivant les idées de [HMR03].

En même temps, ces résultats ne présentent pas encore de cadre complet pour comprendre la sémantique des calculs d'ordre supérieur. La bisimulation contextuelle s'est révélée un outil très robuste pour définir des bisimulations correctes pour des calculs d'ordre supérieur : le prochaine étape consiste à développer des techniques pour éliminer la quantification universelle sur tous les contextes avec lesquels un processus peut interagir. Nous avons montré comment la technique de preuve up-to contexte permet de prouver des équations, aussi complexes soient-elles, en se révélant très efficace. Cependant, il serait très intéressant de pouvoir étendre le STE proposé ici pour les Ambients Mobiles à l'approche que Jeffrey et Rathke ont développé pour HOπ [JR03].

Un projet majeur consiste dans la définition d'axiomatisations des équivalences comportementales dans les calculs d'ordre supérieur. En effet, même si le raisonnement équationel est une technique centrale dans les calculs de processus, cette direction est encore vierge. En utilisant les outils introduits pour les Ambients Mobiles, il semble possible de donner une axiomatisation complète de la CBR dans des calculs similaires aux Ambients Mobiles. Des premiers pas ont été effectués dans cette direction [MZN03a].

La prolongation naturelle de notre recherche pour new-HOPLA consiste à définir et prouver les relations entre le langage et le modèle dénotationnel $\mathbf{Lin}^{\mathcal{I}}$. Nous nous attendons à un résultat d'adéquation complète, où une équivalence contextuelle (non la bisimulation) serait caractérisées en termes d'égalité d'ensembles de chemins. Mettre en relation la sémantique opérationnelle avec un modèle dénotationnel plus informatif, basé sur les pré-faisceaux, semble présenter des difficultés additionnelles.

Un manque important de la théorie de new-HOPLA réside dans l'absence d'équivalences faibles. Le langage new-HOPLA ne présenta rien d'analogue aux réductions internes, et par conséquent on ne sait pas clairement comment introduire de équivalences plus abstraites. Fiore, Cattani et Winskel ont développé un cadre mathématique pour les équivalences faibles dans le contexte des modèles de pré-faisceaux [FCW99], qui pourrait se révéler un bon point de départ.

Pour terminer, nous voudrions préciser les buts poursuivis : nos études des théories opérationnelles des langages de processus d'ordre supérieur, et la définition de new-HOPLA, ont été conçues comme des étapes d'une part vers une comprehension approfondie du calcul concurrent, et d'autre part vers la construction d'une théorie des domaines de la concurrence, à même de modéliser l'ordre supérieur et la génération de noms. Tous ces travaux tendent vers le projet, certes ambitieux, mais que nous espérons fructueux, de donner à la concurrence des outils aussi féconds que les travux de Scott et Strachey l'ont été pour le calcul séquentiel.

2 Introduction

This thesis is about the *semantics of concurrent computation.*

Theories of sequential computation are essentially theories of computable functions: a sequential program transforms a set of input values into output values, performing a sequence of mechanizable steps. In this framework, the role of the denotational approach pioneered by Scott and Strachey is essential: by providing a global mathematical setting, it places programming languages in connection with each other; connects with mathematical worlds of algebra, topology, and logic; inspires type disciplines, programming constructs, and methods of reasoning.

The functional view of computation finds perhaps the most serious limitations in the analysis of concurrent systems. Here, concurrently active agents influence each other's activity on the fly by interacting in all sorts of different ways: the focus is on interaction, on the data-flow, not on the input-output relation. Even if the classical Church-Turing thesis may be seen as indicating that all general models of computation are equivalent, any translation from an expressive approach to concurrency to a formalism like that of Turing Machines would miss 'what really matters', much in the same way an 'isomorphism' (as sets) between the real line and the plane misses the mathematical understanding of space, e.g., the notion of neighbourhood.

In contrast to sequential computation, in concurrency there is no class of 'computable processes' with universal status, and global mathematical guidance is missing. Strategies of research usually seize upon some single notion which seems to be pervasive, make it the focus of a model, and then submit that model to various tests.

Among such notions, *naming* received a lot of attention. Using a name, or address, is inextricably confused with communication, an undoubtedly pervasive notion in concurrent systems. Naming suggests the existence of an abstract space of linked processes, where names are used to represent links and processes interact by using names that they share. The structure of the system changes dynamically because links between processes are created and discharged.

If mobility of links has a relatively developed theory, it is not the case when processes themselves move in the abstract space of linked processes. The design space of languages that aim at seizing the notion of 'mobile agents' is indeed broad. At the same time standard techniques used in concurrency do not lift smoothly to this framework. As a consequence, while there is a plethora of models that focus on mobility of active agents, their understanding is limited, and their theories form a rather fragmented picture.

This thesis addresses the theories underlying the mobile computation, with the aim to develop mathematical tools to express and reason about systems. We focus on labelled bisimilarities to provide tractable proof methods to investigate — that is, to calculate — behavioural theories of mobile systems. Along a different line, we give an operational reading of a denotational model of non-deterministic processes. This leads us to a compact but

expressive language, that directly encodes a rich variety of process languages. The unifying concept of this dissertation is *behaviour*, and what it means to say that two different processes behave the same.

In the next two sections we introduce the main contributions of this thesis.

Background We assume that the reader has a good knowledge of first-order process calculi like CCS and π-calculus, and that he is familiar with standard techniques used in concurrency. Basic notions of domain theory and category theory are required to understand the mathematical structures that led to the definition of new-HOPLA.

2.1 On the semantics of process languages

The basic theory of CCS was developed around a *labelled transition system* and related *bisimulation*.

A *labelled transition system*, or *LTS*, is an inductively defined relation of the form

$$P \xrightarrow{\alpha} Q \ .$$

Intuitively, the action α in the judgement $P \xrightarrow{\alpha} Q$ represents some small context P can interact with, and Q is the residual of P after the interaction. For instance, in CCS, a label a stands for a possible interaction with the context $C_a[-] = - \mid \overline{a}.\mathbf{0}$. An LTS incorporates deep knowledge of the operational semantics of a calculus in its definition.

Bisimulation has been introduced in concurrency by Park and Milner [Par81]. Since then it imposed itself as a popular reference equivalence, mainly thanks to the powerful proof methods associated. A relation between processes $\mathcal{R}$ is a *bisimulation* if whenever $P \mathcal{R} Q$ the following hold:

- if $P \xrightarrow{\alpha} P'$ then there exists Q' such that $Q \xrightarrow{\alpha} Q'$ and $P' \mathcal{R} Q'$;
- if $Q \xrightarrow{\alpha} Q'$ then there exists P' such that $P \xrightarrow{\alpha} P'$ and $P' \mathcal{R} Q'$.

Two processes are equivalent if and only if there is a bisimulation relating them. Accordingly, *bisimilarity*, denoted $\sim$, is defined as the largest bisimulation.

These pleasingly simple notions do not lift smoothly to naming, and even less to higher-order. As a first example, the paper that introduced π-calculus [MPW92] defines *two* bisimulation based equivalences, the *early* and the *late* semantics. *'Why two semantics?'* is a question that puzzled many researchers at that time. Early and late bisimilarity are today well understood. Yet, this anecdote suggests that grabbing a clear understanding of the semantics of a new process calculus can be far from straightforward. As we will see below, features offered by modern process calculi lead to a series of difficulties that can be hard to cope.

Bisimulation and higher-order processes Higher-order complicates the definition of a 'sound' labelled bisimilarity. To introduce higher-order process calculi in a gentle way, we focus on a simple calculus, and we follow in Sangiorgi's steps [San94].

Actions: $\ell \;::=\; \tau \;\mid\; (\boldsymbol{\nu}\vec{z})\overline{a}\langle P\rangle \;\mid\; a(P)$.

$$\frac{}{\overline{a}\langle P\rangle.Q \xrightarrow{\overline{a}\langle P\rangle} Q} \qquad \frac{}{a(X).Q \xrightarrow{a(P)} Q[P/X]} \qquad \frac{P \xrightarrow{(\boldsymbol{\nu}\vec{z})\overline{a}\langle P\rangle} P' \quad n \in (\mathrm{fn}(P) \setminus \vec{z}), n \neq a}{(\boldsymbol{\nu}n)P \xrightarrow{(\boldsymbol{\nu}n,\vec{z})\overline{a}\langle P\rangle} P'}$$

$$\frac{P \xrightarrow{\ell} P' \quad n \notin \mathrm{fn}(\ell)}{(\boldsymbol{\nu}n)P \xrightarrow{\ell} (\boldsymbol{\nu}n)P'} \qquad \frac{P \xrightarrow{\ell} P'}{\begin{array}{c} P \mid Q \xrightarrow{\ell} P' \mid Q \\ Q \mid P \xrightarrow{\ell} Q \mid P' \end{array}} \qquad \frac{P \xrightarrow{(\boldsymbol{\nu}\vec{z})\overline{a}\langle P\rangle} P' \quad Q \xrightarrow{a(P)} Q'}{\begin{array}{c} P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}\vec{z})(P' \mid Q') \\ Q \mid P \xrightarrow{\tau} (\boldsymbol{\nu}\vec{z})(Q' \mid P') \end{array}}$$

Table 2.1: LTS for the sample higher-order process calculus: first try

The language we consider is the second-order fragment of Higher-Order π-calculus, HOπ. It is defined by the grammar below:

$$P, Q \;::=\; \mathbf{0} \;\mid\; !P \;\mid\; (\boldsymbol{\nu}a)P \;\mid\; P \mid Q \;\mid\; \overline{a}\langle P\rangle.Q \;\mid\; a(X).P \;\mid\; X$$

where $a, \ldots m, n, \ldots$ range over an infinite set of names, and $X, Y, \ldots$ range over process variables. Informally, process $\overline{a}\langle Q\rangle.P$ performs an output action at a emitting process Q and then continues as P. Symmetrically, a process $a(X).P$ receives a process, say Q, at a and continues as $P[Q/X]$. The restriction operator $(\boldsymbol{\nu}a)P$ makes the scope of the name a local to process P; as in λ-calculus and in π-calculus the scoping is lexical. Writing $\vec{z}$ for the tuple of names $(z_1, z_2, \ldots, z_n)$, and using standard notations[1], the reduction semantics of the calculus (informally) obeys to the rule below:

$$(\boldsymbol{\nu}\vec{z})\overline{a}\langle P\rangle.Q \mid a(X).R \;\rightsquigarrow\; (\boldsymbol{\nu}\vec{z} \cap \mathrm{fn}(P))((\boldsymbol{\nu}\vec{z} \setminus \mathrm{fn}(P))Q \mid R[P/X]) \qquad \text{if } \vec{z} \cap \mathrm{fn}(R) = \emptyset \text{ and } a \notin \vec{z}\,.$$

Along the lines of the LTS for polyadic π-calculus (see for instance [SW01], Section 3.1), in Table 2.1 we propose a natural transition system. The reduction semantics defined by the LTS corresponds with the expected one. On the other hand, the definition of bisimilarity requires matching actions to be syntactically the same. With the above LTS, bisimilarity breaks basic algebraic laws, such as the commutativity of parallel composition. In fact, the processes $\overline{a}\langle P \mid Q\rangle.\mathbf{0}$ and $\overline{a}\langle Q \mid P\rangle.\mathbf{0}$ emit syntactically different processes and as such are distinguished by bisimulation.

The approach to higher-order computation of several early works [AGR88, Bou89, Tho90] is to require the processes being emitted being bisimilar rather than identical. We refer to the resulting bisimulation as *higher-order bisimulation*. Unfortunately this approach turns to be disastrous with the LTS define above. Let P be a deadlocked process with m free in

[1] All the notations used here are defined at page 63.

Actions: $\ell \;::=\; \tau \;\mid\; \overline{a}\langle P\rangle \;\mid\; a(P)$.

$$\frac{}{A \vdash \overline{a}\langle P\rangle.Q \xrightarrow{\overline{a}\langle P\rangle} Q} \qquad \frac{}{A \vdash a(X).Q \xrightarrow{\overline{a}(P)} Q[P/Q]} \qquad \frac{A, n \vdash P \xrightarrow{\overline{a}\langle P\rangle} P' \quad n \in \mathrm{fn}(P), n \neq a}{A \vdash (\boldsymbol{\nu} n)P \xrightarrow{\overline{a}\langle P\rangle} P'}$$

$$\frac{A, n \vdash P \xrightarrow{\ell} P' \quad n \notin \mathrm{fn}(\ell)}{A \vdash (\boldsymbol{\nu} n)P \xrightarrow{\ell} (\boldsymbol{\nu} n)P'} \qquad \frac{A \vdash P \xrightarrow{\ell} P'}{A \vdash P \mid Q \xrightarrow{\ell} P' \mid Q} \qquad \frac{A \vdash P \xrightarrow{\overline{a}\langle P\rangle} P' \quad A \vdash Q \xrightarrow{a(P)} Q'}{A \vdash P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}\, \mathrm{fn}(P) \setminus A)(P' \mid Q')}$$

Table 2.2: LTS for the higher-order process calculus: second try

it, like $(\boldsymbol{\nu} n)n(X).m(Z).0$, and Q be an arbitrary process with $Y \notin \mathrm{fv}(Q)$. Then this choice equates the processes

$$(\boldsymbol{\nu} m)(\overline{a}\langle m(Y).Q\rangle.\overline{m}\langle\mathbf{0}\rangle.\mathbf{0}) \quad \text{and} \quad (\boldsymbol{\nu} m)(\overline{a}\langle P\rangle.\overline{m}\langle\mathbf{0}\rangle.\mathbf{0})$$

which have completely different possibilities of interactions. In fact,

$$(\boldsymbol{\nu} m)(\overline{a}\langle m(Y).Q\rangle.\overline{m}\langle\mathbf{0}\rangle.\mathbf{0}) \mid a(X).X \xrightarrow{\tau} (\boldsymbol{\nu} m)(\overline{m}\langle\mathbf{0}\rangle.\mathbf{0} \mid m(Y).Q)$$
$$(\boldsymbol{\nu} m)(\overline{a}\langle P\rangle.\overline{m}\langle\mathbf{0}\rangle.\mathbf{0}) \mid a(X).X \xrightarrow{\tau} (\boldsymbol{\nu} m)(\overline{m}\langle\mathbf{0}\rangle.\mathbf{0} \mid P)$$

where $(\boldsymbol{\nu} m)(\overline{m}\langle\mathbf{0}\rangle.\mathbf{0} \mid P)$ is deadlocked, while $(\boldsymbol{\nu} m)(\overline{m}\langle\mathbf{0}\rangle.\mathbf{0} \mid m(Y).Q) \xrightarrow{\tau} (\boldsymbol{\nu} m)(\mathbf{0} \mid Q)$ and Q can engage in arbitrary interactions. Also, this treatment of extruded names yields the law

$$(\boldsymbol{\nu} m)(\overline{a}\langle P\rangle.Q) \;\sim\; \overline{a}\langle(\boldsymbol{\nu} m)P\rangle.Q$$

which is difficult to justify because in the first process all copies of P activated by its recipient share the name m, whereas in the second one, the name m is private to each copy.

The roots of these problems seem to be related to the fact that scope extrusions figure in the labels: in Table 2.2 we define an alternative LTS, extending the LTS for π-calculus defined in [Sew00]. Transitions have the form $A \vdash P \xrightarrow{\ell} Q$, where A is a finite set of names such that $\mathrm{fn}(P) \subseteq A$; they should be read as "in a state where the names A may be known by process P and by its environment, the process P can do ℓ to become Q". The two proposed LTS agree on internal reductions, and they define the same reduction semantics. If we define a higher-order bisimulation on top of the LTS above[2], we end up with a semantics that does not equate

$$\overline{a}\langle\mathbf{0}\rangle.\mathbf{0} \quad \text{and} \quad (\boldsymbol{\nu} m)\overline{a}\langle\overline{m}\langle\mathbf{0}\rangle.\mathbf{0}\rangle.\mathbf{0}$$

as $\overline{m}\langle\mathbf{0}\rangle.\mathbf{0}$ is not bisimilar to $\mathbf{0}$. On the other hand, in any context the two processes above give rise to the same sequence of transitions because the process $\overline{m}\langle\mathbf{0}\rangle.\mathbf{0}$ is deadlocked as no other process knows the name m needed to 'unlock' it.

[2]A little care is required to keep the indexing straight, see Section 3.4.8.

Even if we forget about scope extrusions, higher-order bisimulation, in any of the LTSs above, appears over-discriminating. In fact, it does not equate the processes

$$\overline{a}\langle \mathbf{0} \rangle.!\,\overline{m}\langle \mathbf{0} \rangle.\mathbf{0} \quad \text{and} \quad \overline{a}\langle \overline{m}\langle \mathbf{0} \rangle.\mathbf{0} \rangle.!\,\overline{m}\langle \mathbf{0} \rangle.\mathbf{0}\ .$$

Yet, the replication $!\,\overline{m}\langle \mathbf{0} \rangle.\mathbf{0}$ covers the difference between $\mathbf{0}$ and $\overline{m}\langle \mathbf{0} \rangle.\mathbf{0}$ regardless of how many copies of them are used.

Barbed congruence Because of these complications, new process calculi are often presented as *programming languages*. Technically this means to give their syntax, and specify how a program can be evaluated with respect to a given abstract machine. Once a clear evaluation strategy has been sketched, a notion of observation is introduced, and an equivalence is then derived using *contextual equivalence* or, if branching time is taken into account, *barbed equivalence* [MS92, HY95].

A good example is offered by the original paper on Mobile Ambients by Cardelli and Gordon [CG00b]. They define the syntax of Mobile Ambients:

$$\begin{array}{rcl} P, Q & ::= & \mathbf{0} \;\mid\; P \mid Q \;\mid\; !P \;\mid\; (\boldsymbol{\nu} z)P \;\mid\; n[P] \;\mid\; \pi.P \\ \pi & ::= & \mathtt{in}\_n.P \;\mid\; \mathtt{out}\_n.P \;\mid\; \mathtt{open}\_n.P \end{array}$$

and they give its operational semantics[3] as a set of reduction rules for the Chemical Abstract Machine [BB92]:

$$\begin{array}{rcl} n[\mathtt{in}\_m.P \mid Q] \mid m[R] & \rightarrowtriangle & m[\,n[P \mid Q] \mid R\,] \\ m[\,n[\mathtt{out}\_m.P \mid Q] \mid R\,] & \rightarrowtriangle & n[P \mid Q] \mid m[R] \\ \mathtt{open}\_n.P \mid n[\,Q\,] & \rightarrowtriangle & P \mid Q \end{array}$$

Shortly thereafter, Cardelli and Gordon introduce a Morris-style contextual equivalence (otherwise known as may-testing equivalence) for Mobile Ambients: two processes are contextually equivalent if and only if they admit the same elementary observations whenever they are inserted inside any arbitrary enclosing process. They identify the presence of an ambient whose name is not restricted at the top-level of a process as basic observation. This notion of observation is reminiscent of the report of the successful outcome of an experiment in a theory of *testing* equivalence [DH84, Hen88]. In such a theory the basis for comparing processes is the results of experiments in which the processes are tested by composing them with special terms.

More in detail, a process P *converges to a name* n if after some reductions, P exhibits an ambient named n. Formally,

$$\begin{array}{rcl} P \downarrow n & = & P \equiv (\boldsymbol{\nu}\, \vec{z})\,(n[P_1] \mid P_2) \text{ for some } P_1, P_2 \text{ and set of names } \vec{z} \text{ such that } n \notin \vec{z}; \\ P \Downarrow n & = & \text{either } P \downarrow n \text{ or } P \rightarrowtriangle P' \text{ for some } P' \text{ and } P' \Downarrow n. \end{array}$$

[3]Mobile Ambients is the object of Chapter 4: a complete description its syntax and semantics (in particular the definition of the structural equivalence relation, $\equiv$) can be found there.

Then *contextual equivalence* is the relation $P \cong Q$ defined by:

$$P \simeq Q \qquad \text{iff} \qquad \text{for all } n \text{ and } C[-], C[P] \Downarrow n \Leftrightarrow C[Q] \Downarrow n \ .$$

Contextual equivalence does not take into account the branching structure of processes. Branching time informations can be added to testing theories in several different ways. *Barbed equivalence* provides a simple method to construct a branching time equivalence relation. The idea is to impose a bisimulation structure over reduction steps, and also ask that at each step observations are preserved. As in general barbed equivalence is far from being a congruence for the language, *barbed congruence* is commonly used.

We introduce a variant of barbed congruence, called *reduction barbed congruence*. Reduction barbed congruence, denoted $\cong$, is the largest symmetric relation such that whenever $P \cong Q$ the following hold:

- if $P \rightarrowtriangle P'$ then there exists a process Q' such that $Q \rightarrowtriangle^* Q'$ and $P' \cong Q'$;
- $P \downarrow n$ implies $Q \Downarrow n$;
- for all contexts $C[-]$, it holds $C[P] \cong C[Q]$;

where $\rightarrowtriangle^*$ denotes the reflexive and transitive closure of $\rightarrowtriangle$.

Observe that reduction barbed congruence, as defined above, abstracts from internal reduction, that is, it is a *weak* equivalence. In a distributed setting, it is the visible actions that are of most interest.

The definition of barbed congruence is simple and robust. Yet, it involves quantification over all contexts, making it harder to prove instances of the equivalence (even if Fournet's thesis [Fou98] presents useful techniques). Also, as all ready-made recipes, its definition does not provide in a direct way any information on the behavioural theory of the calculus.

The need for *labelled bisimilarities* Let us go back to Mobile Ambients. Even a notion of equivalence as simple as contextual equivalence suffices to point out some peculiarities of its behavioural theory. Consider a process $(\boldsymbol{\nu}\, n)\, n[P]$ where n is not free in P. Since the name n is know neither inside the ambient $n[P]$, nor outside it, the ambient $n[P]$ is a 'perfect firewall' that neither allows another ambient to enter nor to exit. This is formalised by the equation below (also known as *the perfect firewall equation*):

$$(\boldsymbol{\nu}\, n)\, n[(\boldsymbol{\nu}\, n)\, P] \ \cong \ \mathbf{0} \ .$$

This equation has been proved in [GC02] by a long induction over contexts. For quite a long time the perfect firewall equation has been (almost) the only equation formally proved, and summarised all what was known about Mobile Ambients semantics.

The desire to investigate more in detail the behavioural theory of Mobile Ambients, led Levi and Sangiorgi to define a dialect of Ambients, called *Safe Ambients*: in this novel calculus interaction is more constrained than in Ambients. As a consequence, more equations are valid, and in general their proofs are easier. In [MH01], Merro and Hennessy develop a labelled bisimilarity that coincides with reduction barbed congruence for (a variant) of

Safe Ambients. Along a parallel line of research, for the subset of Mobile Ambients without name restriction, Sangiorgi [San01] shows that the equivalence induced by the spatial logics by Cardelli and Gordon [CG00a] coincides with structural congruence. This means that the ambient logic captures the intensional structure of a process, but not its behaviour.

All these attempts show how elusive can be capturing contextual equivalence or barbed congruence with a labelled bisimilarity. They also show how many efforts are devoted to the quest. In fact, the name *calculus* implies the ability to do *equational reasoning*, that is, to calculate. The definition of *labelled bisimilarities* that imply, or better coincide with, reduction barbed congruence is fundamental tool to investigate the behavioural theory of the calculus. More than that, their development usually provides a great deal of insight into the meaning of barbed congruence, and very fine grained information on equivalent processes [San92, ACS98, FG98].

The Seal Calculus and Mobile Ambients The first part of this dissertation is devoted to the study of the behavioural theories of two process calculi that present two orthogonal models of agent mobility. Both assume the existence of basic spatial objects, called locations. A location abstracts either a physical place, such as hosts or networks, or a logical entity, such as operating system processes or finer grained isolated software components like wrappers [SV99]. To play these different roles, locations are structured in a tree-like hierarchy in which each level models some 'real world' entity. Mobility amounts to perform local changes in the tree-like hierarchy of locations. Both offer 'strong' mobility: a process can be interrupted during its execution and moved together with its current environment to another location where execution is pursued. In the Seal Calculus a location is displaced by processes in its environment and has no control on migration: a location is a passive component of the reduction. On the other hand, in Mobile Ambients the migration instruction comes from the migrating entity itself: locations are active components, and move under the control of the processes located inside them. Under the assumptions that the 'reference frame' is hierarchical, and that routing of messages across distant locations must be explicitly programmed, the study of these two models covers a wide range of the spectrum of possible mobility primitives.

Contributions The name Seal Calculus has been used to denote a family of dialects that evolved from the Seal Calculus as introduced in [VC99]. These dialects share the same design choices and provide the programmer with the same set of abstractions, but little (and surprisingly subtle) differences can found in their reduction semantics. Our investigation aims to tidy up the confused picture offered by these apparently similar dialects, as well as to develop solid foundations for Seal. We give a uniform presentation of the syntax and of the reduction semantics of all of the dialects. A preliminary investigation of their behavioural theories highlights the deep differences between these process calculi, only superficially similar. We concentrate on the dialect that appears at the same time a suitable kernel for a programming language and a process calculus amenable of theoretical investigation. We propose a labelled transition system: its definition points out some of the particularities of the Seal model of computation. On top of the LTS we build a labelled bisimilarity and

we prove that it is a sound proof technique for reduction barbed congruence. The labelled bisimilarity is subsequently used to prove some algebraic laws.

In the framework of Mobile Ambients, we devote our efforts to develop powerful bisimulation based proof techniques. First, as in the Distributed π-calculus [HR98], we rewrite the syntax of Mobile Ambients in two levels: *processes* and *systems*. We are interested in studying systems and movement of code, rather than processes: the two level syntax allows us to focus on higher-order actions, avoiding the overhead caused by actions fired at top-level. We introduce a new labelled transition system for Mobile Ambients which is used to define a labelled bisimilarity over systems. The resulting bisimilarity can be defined either in *late* or in *early* style. However, as in HOπ [San96a], the two formulations coincide, and we concentrate on the easier late version, denoted $\approx$. The definition of $\approx$ reminds us the asynchronous bisimilarity found in [ACS98]. Indeed, as for inputs in asynchronous π, our bisimilarity does not observe the movements of secret ambients. The relation $\approx$ completely characterises reduction barbed congruence over systems. Then, we enhance our proof methods by defining two *up-to proof techniques*, along the lines of [MS92, San98, SW01]. More precisely, we develop both *up-to expansion* and *up-to context* proof techniques and prove their soundness. We are not aware of other forms of up-to proof techniques for higher-order calculi. Finally, we apply our bisimulation proof methods to prove a collection of both old and new *algebraic laws* (including the perfect firewall equation); we also prove the correctness of the protocol, introduced in [CG00b], for controlling access through a firewall.

2.2 A process language out of a semantic model

In the first part of the thesis bisimulation based proof methods are used as a common ground to study the intrinsic theory of two already existing process calculi. That is, we start from their definitions and we develop their foundations. In the second part of the dissertation, the syntax and the operational theory of an expressive language are derived *starting from* its roots in a domain theory for concurrency based on paths, that is, from its foundations.

We put aside operational semantics for a while, and we review a line of research aiming to provide a common mathematical framework to investigate the semantics of concurrency.

Bisimulation from open maps Starting from the work [WN95], Winskel and Nielsen concentrate on understanding the structure of different models for concurrency and how they relate. Category theory has been the natural tool for this task. Their approach turns each class of models into a category, whose objects are models, and whose morphisms stand for simulations. As an example consider the category of transition systems[4] $\mathbf{T}$ (for brevity we assume a common set of labels A), and let $T_1 = (S_1, i_1, A, \rightarrow_1)$, $T_2 = (S_2, i_2, A, \rightarrow_2)$ be two transition systems. A morphism $f : T_1 \rightarrow T_2$ in $\mathbf{T}$ is a function $f : S_1 \rightarrow S_2$ such that $f(i_1) = i_2$ and $t_1 \rightarrow_1 t_2$ implies $ft_1 \rightarrow_2 ft_2$. Observe that f expresses how T_1 may be simulated by T_2. Synchronisations trees arise as a full subcategory of transitions systems. The familiar operation of unfolding a transition system into a synchronisation tree appears

[4] A transition system is a structure $(S, i, A, \rightarrow)$ where S is a set of states, i a distinguished initial state, A a set of labels, and $\rightarrow$ is the transition relation, that is a subset of $S \times A \times S$.

as a right adjoint to the inclusion functor $\mathbf{S} \to \mathbf{T}$. Similar constructions can be carried over also in causal models, like event structures and Petri nets. This presentation not only makes precise how different models relate, but also exposes operations like nondeterministic sum and parallel composition found in process calculi as universal constructions.

Morphisms in a category are understood as kinds of functional simulations. A definition of bisimulation across the categorical models can be built by identifying those morphisms that not only preserve but also reflect behaviour. Consider again the two transitions systems defined above. A morphism $f : T_1 \to T_2$ is said to be a *zig-zag morphism* (or *abstraction homomorphism* [Cas85]) if whenever $fs \xrightarrow{\alpha}_2 t'$ there is a state $t \in S_1$ such that $s \xrightarrow{\alpha} t$ and $ft = t'$. It is clear that if f is a zig-zag morphism then its graph is a bisimulation between the two transition systems. In categorical terms, suppose we are given a sequence of transitions in T_1:

$$p_1 : i_1 \xrightarrow{\alpha_1} r_1 \xrightarrow{\alpha_2} r_2 \xrightarrow{\alpha_3} \ldots \xrightarrow{\alpha_n} s_1$$

Since $f : T_1 \to T_2$ is a morphism, this induces a simulating sequence in T_2:

$$p_2 : i_2 \xrightarrow{\alpha_1} fr_1 \xrightarrow{\alpha_2} fr_2 \xrightarrow{\alpha_3} \ldots \xrightarrow{\alpha_n} fs_1$$

If f is zig-zag, then any extension q_2 of p_2 with an extra transition $fs_1 \xrightarrow{\alpha} s_2'$, implies that also p_1 can be extended to a sequence

$$q_1 : i_1 \xrightarrow{\alpha_1} r_1 \xrightarrow{\alpha_2} r_2 \xrightarrow{\alpha_3} \ldots \xrightarrow{\alpha_n} s_1 \xrightarrow{\alpha} s_1' \ ,$$

where s_1' is such that $fs_1' = s_2'$. Now, the four sequences above are themselves transition systems, and objects of $\mathbf{T}$. Clearly $p_1 \cong p_2$ and $q_1 \cong q_2$, and we will refer to them as p and q. As q is an extension of p there is an obvious morphism $e : p \to q$, and because p is a sequence of T_1 and q a sequence of T_2 we have morphisms $x : p \to T_1$ and $y : q \to T_2$, so that the square on the left below commutes. Commutativity rephrases the fact the simulations by T_2 of p can be extended to the simulation of q.

$$\begin{array}{ccc} p & \xrightarrow{x} & T_1 \\ {\scriptstyle e}\downarrow & & \downarrow{\scriptstyle f} \\ q & \xrightarrow[y]{} & T_2 \end{array} \qquad\qquad \begin{array}{ccc} p & \xrightarrow{x} & T_1 \\ {\scriptstyle e}\downarrow & \overset{z}{\nearrow} & \downarrow{\scriptstyle f} \\ q & \xrightarrow[y]{} & T_2 \end{array}$$

The zig-zag requirement on f can then be stated by saying that any such commutative square can be split into two commutative diagrams as on the right above. Commutativity of the upper triangle expresses that the sequence in T_1 that simulates q is an extension of the sequence simulating p. The lower triangle amounts to saying that this extension is compatible under f with the sequence simulating q in T_2.

It is reasonable to think to objects p and q as *computation paths*. In the model of transition systems they correspond to sequences of transitions, that is, traces. In general, computation paths may have more structure than traditional traces, e.g., allowing path semantics to take nondeterministic branching into account in a limited way.

Abstractly, let $\mathbf{M}$ be any category of models and be $\mathbb{P}$ be a subcategory $\mathbb{P} \to \mathbf{M}$ thought of as the category of the computation paths of $\mathbf{M}$. We define $\mathbb{P}$-*open maps* to be those

morphisms f of $\mathbf{M}$ such that for all morphisms $e : p \to q$ in $\mathbb{P}$, any commuting square like that on the left can be split into two commuting triangles as on the right. We say that two models M and N of $\mathbf{M}$ are open map bisimilar if there exists a span $M \leftarrow O \rightarrow N$ of open maps between them. In the case of transition systems and the subcategory of finite sequences, open-map bisimilarity coincides with Park and Milner's bisimulation. Reasonable equivalence are obtained for other models as well, see [JNW94].

Presheaf models Open map bisimulation provides an abstract way to uniformly define bisimulation on models for concurrency, once a suitable subcategory of paths has been chosen. There is much freedom in the choice of the subcategory $\mathbb{P}$, whose only requirement is to be a subcategory of the model.

But an important class of categories are equipped with a canonical choice for a path category: these are the so-called *presheaf categories*. The objects of a presheaf category, called *presheaves*, can be understood as the collection of processes constructed from its canonical underlying path category by freely adding coproduct (nondeterministic sums of paths) and coequalisers (gluing paths together by identifying subpaths). An alternative view is that a presheaf, seen as a process, assigns to each computation path the set of ways it is simulated by, or 'realised by', the process. In the light of the above, Cattani and Winskel have drawn attention to a 2-categorical model of linear logic and associated comonads, based on presheaves [CW03]. The categorical overhead involved is quite conspicuous.

In general it is quite inconvenient to work directly in a specific category or with a specific monad. Mathematical logic provides a simple recipe to abstract away from specific models: fix a language, define what is an interpretation of the language in a model, find a formal system (on the language) that captures the desired properties of models. When the formal system is sound, one can forget about the models and use the formal systems instead. Moreover, if the formal system is also complete, then nothing is lost (as far as one is concerned with properties expressible in the language, and valid in all models).

This led Nygaard and Winskel to the discovery of two expressive metalanguages for concurrency, called HOPLA (Higher Order Process LAnguage) and AL (Affine Language). The presheaf semantics led to operational semantics, guided by the idea that derivations of transitions in the operational semantics, associated with paths, should correspond to realisers in the presheaf definitions. Process calculi such as CCS, CCS enriched with process passing, and Mobile Ambients with public names can be encoded in HOPLA.

The language new-HOPLA Although surprisingly expressive, HOPLA cannot model *name generation*, one basic ingredient of mobile systems, and, in general, of modern process calculi. In the second part of this thesis, we study an extension to equip HOPLA with name generation.

At the level of the domain theory, the idea is to work in a functor category indexed by the category of finite sets and injective functions. We give an account of the denotational model using path sets, rather than presheaves. Even if path sets are considerably simpler than presheaves, they furnish a model which is sufficiently rich in structure to show how new-HOPLA arises from canonical constructions on path sets. For the most part new-HOPLA lends itself to an operational account. As for HOPLA, the intuitions on the

interpretation lead us to the development of a syntax, of a type system, and of an operational semantics. The language new-HOPLA can be viewed as an extension of the simply typed λ-calculus with prefixing, sums, and constructs to manipulate dynamically generated names. We investigate its operational theory. Particular attention is devoted to the bisimulation equivalence that arises naturally, and we show that it is a congruence for the language. Rich process algebras can be faithfully encoded into new-HOPLA. We define an encoding of π-calculus that not only preserves and reflects the reduction semantics, but also the behavioural theory. We also provide encodings of polyadic π-calculus and of Mobile Ambients.

2.3 Overview

Outline This dissertation is composed by three central chapters, plus an introduction and a conclusion. Each chapter focuses on a particular process language.

Chapter 3 presents the family of languages commonly referred to as 'Seal Calculus'. A preliminary investigation of their behavioural theories highlights the deep differences among these dialects, and focus our attention on one of them. To investigate more in depth its behavioural theory, we develop a bisimulation based proof method, subsequently used to prove some basic equivalences.

Chapter 4 is devoted to the development of bisimulation based proof methods for Cardelli and Gordon's Mobile Ambients. We give an *LTS* based operational semantics, and a labelled *bisimulation* based equivalence that coincides with reduction barbed congruence. We also provide *up-to proof techniques* and prove a set of *algebraic laws*, including the perfect firewall equation and the correctness of a more complex protocol.

Chapter 5 introduces a typed language for higher-order nondeterministic processes. Its roots in a domain theory of concurrency are sketched though for the most part it lends itself to a more operational account. Its operational theory is investigated. It is also shown how the language can directly encode π-calculus, polyadic π-calculus, and Mobile Ambients.

Each part ends by setting the specific results in the context of what we see as a promising broader enterprise towards a full understanding of the semantics of higher-order processes.

How to read this thesis As a PhD thesis, this document includes the details of proofs too malignant (or simply too boring) to inflict upon readers. We believe that at a first reading most of the proofs can be safely skipped. There are some notable exceptions though. In particular, we point the reader to the proof of completeness of bisimilarity reported in Section 4.3.2, which is definitely worth an accurate study.

Provenance of the material This dissertation is partially based on published material.

Investigation of dialects of the Seal Calculus amenable of theoretical investigation, with the distinction of shared and localised channels, began in

[CGZN01]: *Typing Mobility in the Seal Calculus.*
In *Proceedings of CONCUR '01.*
Coauthored with Giuseppe Castagna, and Giorgio Ghelli.

The presentation of the syntax of the Seal Calculus abstracting over the localisation of channels, the restriction on scope extrusions, and a very preliminary characterisation of reduction barbed congruence in *eS*-Seal (a dialect different from that we focus on in this dissertation) are reported in

[CZN02]: *The Seal Calculus Revisited: Contextual Equivalence and Bisimilarity.*
In *Proceedings of FSTTCS '02.*
Coauthored with Giuseppe Castagna.

The study of bisimulation based proof methods for Mobile Ambients appeared in

[MZN03b]: *Bisimulation Proof Methods for Mobile Ambients.*
In *Proceedings of ICALP '03.*
Coauthored with Massimo Merro.

The development of new-HOPLA is fruit of a collaboration with Glynn Winskel [WZN03].

Material not included in this document The uniform presentation of the reduction semantics of the Seal Calculus had a somewhat surprising side effect. In [ZN00] (and later in [CGZN01]), we study a type system for the shared variant of the Seal Calculus that aims to define what 'typing mobility' means. Only little modifications are required to build an analogous type system on top of the located dialect of the Seal Calculus. The resulting type system has a nice interpretation as a system for resource access control: the type of a location declares to the environment the local resources that are made available, all the others being reserved for private use. This result has not been included in this dissertation as it clearly belongs to a completely different direction of research. It can be found in [CVZN03].

On the semantics of process languages:

bisimulation proof methods

3 The Seal Calculus

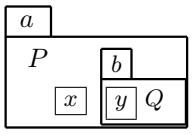
The Seal Calculus distills the main concepts of distributed computing down to three abstractions: *processes*, *locations*, and *resources*. Processes are sequential threads of control modelled on the π-calculus with terms to denote the inert process, sequential and parallel composition, and replication. Locations are basic spatial objects that can be nested, thus creating a tree like hierarchy. Resources are modelled by *channels*, named structures over which processes synchronise.

The Seal Calculus has been introduced in [VC99] as formal counterpart to a kernel for programming mobile agents [BV01] on top of the Java Virtual Machine. Since then, the name Seal Calculus has been used to denote a family of process calculi that originate from the same design choices and provide the programmer with the same set of abstractions.

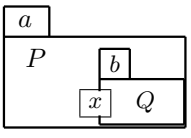
Located and Shared Seal A distinctive feature of the Seal model of computation is that processes can interact only over channels that are 'close enough' in the seal hierarchy. We identified two interpretations, called respectively *located channels* and *shared channels*, of what 'close enough' means. In the first case, channels are associated to seals and channel denotations specify the seal in which a channel is located. In the second interpretation, channels are shared between communicating entities, so that channel denotations specify the partner the channel is shared with.

Figure 3.1(a) represents a configuration in the located interpretation. Channels x and y are localised in two different seals. Synchronisation between these seals can happen over both channels: to synchronise over x, process P must specify that it refers to a local channel, while Q will specify access to a channel located in its direct parent. Thus, channel x will be referred as x^* by process P, and as channel $x^\uparrow$ by Q, while channel y is referred as y^n by P, and as y^* by Q.

Figure 3.1(b) depicts a similar configuration in the shared interpretation. In this interpretation only the channel x can be used to synchronise processes P and Q, and it will be referred to as x^n and $x^\uparrow$. Process P cannot access to channel y.



(a) Localised Channels



(b) Shared Channels

Figure 3.1: Channels in the Seal Calculus

Mobility and Extrusion of Names Scoping is lexical in the Seal Calculus and, as in π-calculus, the scope of a name can be enlarged after an interaction. Although scope extrusion is fundamental to the expressivity of the calculus, it may be desirable to limit it in some cases. In particular there are arguments to prevent scope extrusion when a name exits from the boundaries of the enclosing seal as a consequence of a mobility interaction. We refer to this constraint as the *e*-condition.

Four dialects As localisation of channels and the *e*-condition are orthogonal features, we build four dialects of the Seal Calculus. All of them already appeared in published papers, and contributed to the confusion that reigns over the name Seal Calculus.

- *L-Seal*: located channels; (almost) the original Seal Calculus, as presented in [VC99];
- *eL-Seal*: located channels and *e*-condition; introduced in [CZN02];
- *S-Seal*: shared channels; appears in [CGZN01];
- *eS-Seal*: shared channels and *e*-condition; introduced in [CZN02].

This chapter aims to tidy up the scattered picture offered by these apparently similar calculi, and to develop solid foundations for the Seal Calculus.

Overview In an uniform framework, we present the four dialects of the Seal Calculus. A preliminary investigation of their behavioural theories highlights the deep differences among these apparently similar calculi. More than that, it directs our attention to one of the dialects. This dialect is not only suited for theoretical investigation, but also prevents subtle errors in the programming practise. To investigate more in depth its behavioural theory, we develop a bisimulation based proof method, and we use it to prove some basic equivalences. A discussion of the limits of the proposed proof method lead us to the next chapter.

3.1 Syntax and operational semantics

The syntax of the Seal Calculus is defined in Table 3.1, where $\mathbf{N}$ denotes an infinite set of names. Following the π-calculus, the inert process is denoted by $\mathbf{0}$. A process $\alpha.P$ is composed of an action α and a continuation P; it denotes a process waiting to performing α and then behaving as P. Actions consist of communication and moves and are explained later on. The term $P \mid Q$ denotes a process composed of two subprocesses, P and Q running in parallel. Locations, called *seals*[1], are represented by the term $a[P]$, where the process P is running at a. A location is itself a process. The replicated process $!\alpha.P$ creates an unbound number of copies of $\alpha.P$ running in parallel. We consider guarded replication instead of full replication because it allows us to derive a simpler LTS in Section 3.3. We also recall that in the π-calculus (i) replicated input has the same expressive power as full replication [HY94] and recursion [Mil91, SW01]; (ii) replicated input has a simpler semantics and is handy for implementations.

[1]The name *seal* is a contraction of *Sealed Object*, and derives from the original implementation of the kernel of JavaSeal.

Names: $a, \ldots, u, v, x, y, z, \cdots \in \mathbf{N}$

Locations:

η	::=	$*$	local
	$\mid$	$\uparrow$	up
	$\mid$	z	down

Actions:

α	::=	$\overline{x}^{\eta}(y_1, \cdots, y_n)$	output
	$\mid$	$x^{\eta}(y_1, \cdots, y_n)$	input
	$\mid$	$\overline{x}^{\eta}\{y\}$	send
	$\mid$	$x^{\eta}\{y_1, \cdots, y_n\}$	receive

for $n \geq 0$.

Processes:

P, Q, R	::=	$\mathbf{0}$	inactivity
	$\mid$	$P \mid Q$	parallel composition
	$\mid$	$(\boldsymbol{\nu} x)P$	restriction
	$\mid$	$\alpha.P$	prefixing
	$\mid$	$n[P]$	seal
	$\mid$	$!\alpha.P$	replication

Table 3.1: The syntax of the Seal Calculus

We work modulo α-conversion and thus suppose all bound variables in the same process to be distinct. Some useful notations: we write $\vec{x}_n$ (or simply $\vec{x}$) to denote the tuple $x_1, \cdots, x_n$. We also write $(\boldsymbol{\nu}\vec{x}_n)P$ (or $(\boldsymbol{\nu}\vec{x})P$), for $(\boldsymbol{\nu}x_1)\ldots(\boldsymbol{\nu}x_n)P$. We omit $*$ locations and trailing $\mathbf{0}$ processes. In the input action $x^{\eta}(y_1, \cdots, y_n)$ the y_i's are required to be pairwise distinct (however this is not required in the receiving action: see § 3.1).

The free names of actions and processes are defined as:

$$
\begin{array}{rclcrcl}
\mathrm{fn}(\mathbf{0}) & = & \varnothing & & \mathrm{fn}(\uparrow) & = & \varnothing \\
\mathrm{fn}(*) & = & \varnothing & & \mathrm{fn}(x) & = & \{x\} \\
\mathrm{fn}(P \mid Q) & = & \mathrm{fn}(P) \cup \mathrm{fn}(Q) & & \mathrm{fn}(!\alpha.P) & = & \mathrm{fn}(\alpha.P) \\
\mathrm{fn}(x[P]) & = & \mathrm{fn}(P) \cup \{x\} & & \mathrm{fn}((\boldsymbol{\nu}x)P) & = & \mathrm{fn}(P) \setminus \{x\} \\
\mathrm{fn}(x^{\eta}(\vec{y}).P) & = & (\mathrm{fn}(P) \setminus \vec{y}) \cup \{x\} \cup \mathrm{fn}(\eta) & & \mathrm{fn}(\overline{x}^{\eta}(\vec{y}).P) & = & \mathrm{fn}(P) \cup \vec{y} \cup \{x\} \cup \mathrm{fn}(\eta) \\
\mathrm{fn}(\overline{x}^{\eta}\{y\}.P) & = & \mathrm{fn}(P) \cup \{y\} \cup \{x\} \cup \mathrm{fn}(\eta) & & \mathrm{fn}(x^{\eta}\{\vec{y}\}.P) & = & \mathrm{fn}(P) \cup \vec{y} \cup \{x\} \cup \mathrm{fn}(\eta)
\end{array}
$$

Contrary to what happens with the input action, the receive action is not a binding operation[2]. As we will see shortly, a name specified in the receive action denotes the name of a seal that will run in parallel with the receiving process. It seems an undue restriction to

[2]As a consequence, the names y_i are not required to be pairwise distinct.

limit the scope of the names of incoming seals to the sole continuation of the receive action. If a similar behaviour is needed, it can be easily programmed.

We use $P[y/x]$ to denote the process obtained from P by substituting y for all free occurrences of x, and use $P[\vec{y}_n/\vec{x}_n]$ to denote the process obtained from P by simultaneous substitution of y_i for x_i. The latter is not defined for vectors of different arity.

Actions Every interaction takes place over named, localised, channels. The Seal Calculus differentiates between local and remote interaction. Interaction is local if the processes that synchronise are located in the same seal. Interaction is remote if the processes are located in seals in parent-child relationship: depending on the interpretation, they synchronise over a shared channel or a channel located in one of the host seals. Processes that are not in seals in parent-child relationship can not communicate: any interaction that spans more than a single seal boundary has to be encoded explicitly.

Channel synchronisation is used both for communication (exchange of names) and for mobility (exchange of seals).

***Communication*:** $\overline{x}^{\eta}(\vec{y}).P$ denotes a process ready to output $\vec{y}$ on channel x^{η} and then continue as P, and $x^{\eta}(\vec{z}).Q$ denotes a process that will read from channel x^{η} and then continue as Q where the free occurrences of $\vec{z}_i$ are replaced by $\vec{y}_i$.

As an example, in the configuration of Figure 3.1(a), the reduction below illustrates the exchange of the name w between P and Q over the channel y located inside the seal b.

$$a[\ \underbrace{\overline{y}^{b}(w).R}_{P} \mid b[\underbrace{y^{*}(z).S}_{Q}]\ ] \quad \rightarrowtriangle \quad a[R \mid b[S[w/z]]]$$

Referring to Figure 3.1(b), a similar interaction carried on in the shared interpretation looks like:

$$a[\ \underbrace{x^{b}(z).R}_{P} \mid b[\underbrace{\overline{x}^{\uparrow}(w).S}_{Q}]\ ] \quad \rightarrowtriangle \quad a[R[w/z] \mid b[S]]$$

Remark that the (sum and matching free) polyadic π-calculus is a sub-calculus of the Seal Calculus as it can be obtained by forbidding the use of seal processes, of send and receive actions, and of up and down locations.

***Mobility*:** $\overline{x}^{\eta}\{\!|\, y \,|\!\}.P$ denotes a process ready to send a child seal named y along channel x^{η}; $x^{\eta}\{\!|\, \vec{z}_n \,|\!\}.P$ denotes a process that will receive a seal along channel x^{η} and reactivate n copies of it inside newly created seals called respectively z_i.

The nesting of seals can be represented as a tree, and mobility corresponds to a tree rewriting operation, as shown in Figure 3.2. A move disconnects a subtree rooted at some seal y and grafts it either onto the parent of y (configuration (b)), onto one of the children of y (configuration (c)), or back onto y itself (configuration (d)). The rewriting operation relabels the edge associated to the moved seal, and can also create a finite number of copies of the subtree rooted at the moved seal. These tree transformations correspond to mobility reduction rules of the calculus.

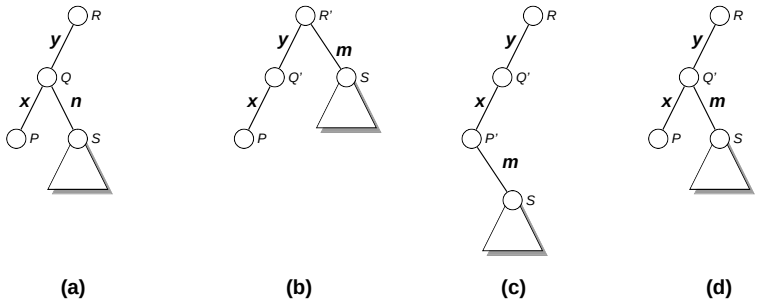


Figure 3.2: Mobility as restructuring the tree of locations

Seals can be copied A straightforward application of the semantics of mobility is the implementation of a copy operation:

$$(\mathsf{copy}\, x\, \mathsf{as}\, z).P \quad \stackrel{def}{=} \quad (\boldsymbol{\nu}\, y)\,(\,\overline{y}^*\{x\} \mid y^*\{x\,, z\}.P\,)$$

The term $\mathsf{copy}\, n\, \mathsf{as}\, m \mid n[P]$ reduces to $n[P] \mid m[P]$. The copy process creates a brand new channel name y to prevent other processes from interfering with the protocol. Then, the left hand side subprocess attempts to move n over the local channel, while the right hand side process receives it and instantiates two copies of it under names n and m. The same technique can be used to destroy a seal:

$$(\mathsf{destroy}\, x\,).P \quad \stackrel{def}{=} \quad (\boldsymbol{\nu}\, y)\,(\,\overline{y}^*\{x\} \mid y^*\{\}.P\,)$$

Here the receive action instantiates zero copies of the sent seal, thus destroying it. Duplication differs from replication, $!P$, because replication creates copies of inert processes while duplication copies running processes. Higher-order π-calculus [San92, San96a] is also restricted to inactive processes.

Synchronisation We define the reduction semantics of the calculus in chemical style, using a structural congruence relation and a set of reduction rules. The semantics is parametric on the interpretation of channels. For that, we introduce two predicates

$$\mathsf{synch}^S, \mathsf{synch}^L : \mathbf{Var} \times \mathbf{Loc} \times \mathbf{Loc} \to \mathbf{Bool}$$

ranged over by synch. Intuitively, $\mathsf{synch}_y(\eta_1, \eta_2)$ holds if and only if for any channel x an action on x^{η_1} performed in some parent seal may synchronise with a coaction on x^{η_2} performed in a child seal named y (in this case we say that η_1 and η_2 are 'y-corresponding' locations).

Definition 3.1.1 (y-correspondence) *Let η_1, η_2 be locations and y a name. We define:*

1. Shared channels: $\mathsf{synch}^S_y(\eta_1, \eta_2) \stackrel{def}{=} (\eta_1 = y \wedge \eta_2 = \uparrow)$;
2. Located channels: $\mathsf{synch}^L_y(\eta_1, \eta_2) \stackrel{def}{=} (\eta_1 = y \wedge \eta_2 = *) \vee (\eta_1 = * \wedge \eta_2 = \uparrow)$.

Definition 3.1.2 (Contexts) *A* context *is a term containing a hole, denoted $-$, generated by the grammar below:*

$$C[-] \quad ::= \quad - \quad \mid \quad C[-] \mid P \quad \mid \quad P \mid C[-] \quad \mid \quad n[C[-]] \quad \mid \quad \alpha.C[-] \quad \mid \quad (\boldsymbol{\nu} n)C[-]$$

where P is an arbitrary process, α an arbitrary action, and n an arbitrary name.

A static context *is a context where the hole does not appear under prefix.*

Definition 3.1.3 (Structural Congruence) *Structural congruence is the smallest relation over processes that is preserved by static contexts and that satisfies the following axioms:*

$$\begin{array}{rcl} P \mid \mathbf{0} & \equiv & P \\ P \mid Q & \equiv & Q \mid P \\ P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R \\ !\alpha.P & \equiv & \alpha.P \mid !\alpha.P \end{array} \qquad \begin{array}{rcll} (\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P & \equiv & (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P & x \neq y \\ (\boldsymbol{\nu} x)(P \mid Q) & \equiv & P \mid (\boldsymbol{\nu} x)Q & x \notin \mathrm{fn}(P) \\ (\boldsymbol{\nu} x)\mathbf{0} & \equiv & \mathbf{0} & \end{array}$$

Seal duplication has an interesting consequence on the semantics of the Seal Calculus: the structural rule

$$(\boldsymbol{\nu} x)y[P] \quad \equiv \quad y[(\boldsymbol{\nu} x)P] \qquad x \neq y \tag{3.1}$$

found for instance in the semantics of Mobile Ambients is not sound. In fact the processes $(\boldsymbol{\nu} x)y[P]$ and $y[(\boldsymbol{\nu} x)P]$ are not equivalent: if we compose the terms with the process $Q =$ copy n as m we obtain

$$n[(\boldsymbol{\nu} x)P] \mid Q \;\rightarrowtriangle\; n[(\boldsymbol{\nu} x)P] \mid m[(\boldsymbol{\nu} x)P] \quad \text{and} \quad (\boldsymbol{\nu} x)n[P] \mid Q \;\rightarrowtriangle\; (\boldsymbol{\nu} x)(n[P] \mid m[P]) \quad .$$

The first process yields a configuration where seals n and m have each a private channel x, while the second process reduces to a configuration where n and m share a common channel x. This unsoundness is formally proved in Section 3.2.2, after defining our reference equivalence.

Since we work modulo α-conversion we can suppose that in the term $(\boldsymbol{\nu}\vec{x})P$ the names $x_1, \ldots, x_n$ are all distinct. Structural congruence states that their order is not important as they can be freely permuted. This justifies the use of set-theoretic operations such as $(\boldsymbol{\nu}\vec{x} \cap \vec{y})P$ or $(\boldsymbol{\nu}\vec{x} \setminus \vec{y})P$, with the convention that $(\boldsymbol{\nu}\varnothing)P = P$.

Definition 3.1.4 (Reduction Relation) *The reduction relation, $\rightarrowtriangle$ is the smallest relation between processes that satisfies the rules reported in Table 3.2 and is preserved by all* static contexts*. Its reflexive and transitive closure is denoted $\rightarrowtriangle^*$.*

As structural congruence cannot move restrictions across seal boundaries, the reduction rules (write out) and (move out) explicitly extrude restrictions through seal boundaries. The non-local rules are parametric in synch: different remote interaction patterns are obtained according to whether synch is replaced by synch^S (shared channels), or synch^L (located channels).

The first three rules define the semantics of communications. Rule (write local) defines local communication: it is the same rule found in polyadic π-calculus. Rule (write in) describes the communication of a tuple $\vec{v}$ from a parent to its child y. Reduction takes place provided

$$
\begin{array}{lll}
\text{(write local)} & x^*(\vec{u}).P \mid \overline{x}^*(\vec{v}).Q & \rightarrowtriangle\ P[\vec{v}/\vec{u}] \mid Q \\
\text{(write in)} & \overline{x}^{\eta_1}(\vec{w}).P \mid y[(\boldsymbol{\nu}\vec{z})(x^{\eta_2}(\vec{u}).Q_1 \mid Q_2)] & \rightarrowtriangle\ P \mid y[(\boldsymbol{\nu}\vec{z})(Q_1[\vec{w}/\vec{u}] \mid Q_2)] \\
\text{(write out)} & x^{\eta_1}(\vec{u}).P \mid y[(\boldsymbol{\nu}\vec{z})(\overline{x}^{\eta_2}(\vec{v}).Q_1 \mid Q_2)] & \\
 & \qquad \rightarrowtriangle\ (\boldsymbol{\nu}\vec{v} \cap \vec{z})(P[\vec{v}/\vec{u}] \mid y[(\boldsymbol{\nu}\vec{z} \setminus \vec{v})(Q_1 \mid Q_2)]) & \\
\text{(move local)} & x^*\{\vec{u}\}.P_1 \mid \overline{x}^*\{v\}.P_2 \mid v[Q] & \rightarrowtriangle\ P_1 \mid u_1[Q] \mid \cdots \mid u_n[Q] \mid P_2 \\
\text{(move in)} & \overline{x}^{\eta_1}\{v\}.P \mid v[S] \mid y[(\boldsymbol{\nu}\vec{z})(x^{\eta_2}\{\vec{u}\}.Q_1 \mid Q_2)] & \\
 & \qquad \rightarrowtriangle\ P \mid y[(\boldsymbol{\nu}\vec{z})(Q_1 \mid Q_2 \mid u_1[S] \mid \cdots \mid u_n[S])] & \\
\text{(move out)} & x^{\eta_1}\{\vec{u}\}.P \mid y[(\boldsymbol{\nu}\vec{z})(\overline{x}^{\eta_2}\{v\}.Q_1 \mid v[R] \mid Q_2)] & \\
 & \quad \rightarrowtriangle\ P \mid (\boldsymbol{\nu}\,\mathrm{fn}(R) \cap \vec{z})(u_1[R] \mid \cdots \mid u_n[R] \mid y[(\boldsymbol{\nu}\vec{z} \setminus \mathrm{fn}(R))(Q_1 \mid Q_2)]) & \\
\text{(red struct)} & P \equiv P' \ \wedge\ P' \rightarrowtriangle Q' \ \wedge\ Q' \equiv Q \text{ implies } P \rightarrowtriangle Q &
\end{array}
$$

where $x \notin \vec{z}$, $\vec{w} \cap \vec{z} = \emptyset$, $\mathrm{fn}(S) \cap \vec{z} = \emptyset$ and $\mathsf{synch}_y(\eta_1, \eta_2)$ holds.

Table 3.2: Reduction rules for the Seal calculus

that (*i*) η_1 and η_2 are y-corresponding locations, (*ii*) channel x is not locally restricted (i.e., $x \notin \vec{z}$), and (*iii*) no name capture arises. (i.e., $\vec{w} \cap \vec{z} = \emptyset$). Rule (write out) corresponds to the case where a child y communicates to its parent a vector $\vec{v}$ of names. Again η_1 and η_2 must be y-corresponding locations and x must not be locally restricted (i.e., $x \notin \vec{z}$). Names local to y in $\vec{v}$ may be extruded across the boundaries of y.

The three remaining rules define the semantics of mobility. In local mobility (rule (move local)) the body of the seal specified by the send action is copied n times, named as specified by the receive action. A seal can be moved inside a child y (rule (move in)) provided that (*i*) η_1 and η_2 are y-corresponding locations, (*ii*) channel x is not locally restricted (i.e., $x \notin \vec{z}$), and (*iii*) no variable free in the moved process is captured (i.e., $\mathrm{fn}(R) \cap \vec{z} = \varnothing$). A seal can be moved out of its enclosing seal (rule (move out)). Again η_1 and η_2 must be y-corresponding locations and x must not be locally restricted (i.e., $x \notin \vec{z}$). Names local to y but free in R may be extruded across the boundaries of y.

The *e*-condition There is a tension whether to allow extrusion of names as a consequence of a mobility action. Names private to a seal can be considered its private resources, and it is legitimate to prevent these names to be extruded to the enclosing seal as a consequence of a mobility action. In fact, in a distributed implementation locally restricted channels would correspond to local variables (e.g. pointers). Moving these channels outside the enclosing location then requires to implicitly transform them into globally unique identifiers.

If this more restrictive option is chosen, then all variables free in an exiting seal must already be known by the parent either because they are non-local or because they have been

previously communicated to it. In the example below, the output action extrudes the name y, so that there is no harm in moving the seal m out of n

$$n[(\boldsymbol{\nu} y)(\overline{u}^{\uparrow}(y).\overline{x}^{\uparrow}\{\!|m|\!\} \mid m[\overline{y}^{\uparrow}()])] \ .$$

Extrusion of names across seal boundaries as a consequence of mobility can be forbidden by adding the condition

$$\mathrm{fn}(R) \cap \vec{z} = \emptyset \tag{3.2}$$

to rule (move out). We refer to the equation 3.2 as the *e-condition*. In this case, the (move out) rule can be simplified, and rewritten as:

$$x^{\eta_1}\{\!|\vec{u}|\!\}.P \mid y[(\boldsymbol{\nu}\vec{z})(\overline{x}^{\eta_2}\{\!|v|\!\}.Q_1 \mid v[R] \mid Q_2)] \ \twoheadrightarrow \ P \mid u_1[R] \mid \cdots \mid u_n[R] \mid y[(\boldsymbol{\nu}\vec{z})(Q_1 \mid Q_2)]$$

where $\mathrm{fn}(R) \cap \vec{z} = \varnothing$, $x \notin \vec{z}$, and $\mathsf{synch}_y(\eta_1, \eta_2)$ holds.

Adding the e-condition to the semantics of the Seal Calculus has surprising effects on its behavioural theory, as we will see shortly.

3.2 Behavioural theories

In this section we define the basic equivalence relation we consider, and we discuss the behaviour between our dialects of Seal and their equational theories.

3.2.1 Reduction barbed congruence

We introduce an equivalence constructed from natural, desirable, properties, that will be our reference equivalence notion. We focus on a slight variant of Milner and Sangiorgi's barbed congruence[MS92], called *reduction barbed congruence*. This relation was first studied by Honda and Yoshida under the name of *maximum sound theory* [HY95].

Definition 3.2.1 (Reduction closed) *A relation $\mathcal{R}$ over processes is* reduction closed *if $P \ \mathcal{R} \ Q$ and $P \twoheadrightarrow P'$ implies the existence of some Q' such that $Q \twoheadrightarrow^* Q'$ and $P' \ \mathcal{R} \ Q'$.*

Definition 3.2.2 (Contextual) *A relation $\mathcal{R}$ over processes is* contextual *if $P \ \mathcal{R} \ Q$ implies $C[P] \ \mathcal{R} \ C[Q]$ for all contexts $C[-]$.*

We build our analysis on the assumption that the basic observable entity in Seal is the presence of a seal whose name is public at top-level. This observation, originally due to Cardelli and Gordon [CG00b, GC02], can be interpreted as the ability of the top-level process to interact with that seal. In Section 3.4.1 we will see that the reduction barbed congruence is insensible to the exact observation chosen.

Definition 3.2.3 (Barbs) *We write $P \downarrow n$ if and only if there exist Q, R, $\vec{x}$ such that $P \equiv (\boldsymbol{\nu}\vec{x})(n[Q] \mid R)$ where $n \notin \vec{x}$. We write $P \Downarrow n$ if there exists P' such that $P \twoheadrightarrow^* P'$ and $P' \downarrow n$.*

Definition 3.2.4 (Barb preserving) *We say that a relation $\mathcal{R}$ over processes is* barb preserving *if $P\ \mathcal{R}\ Q$ and $P \downarrow n$ implies $Q \Downarrow n$.*

Definition 3.2.5 (Reduction barbed congruence) *Reduction barbed congruence, written $\cong$, is the largest symmetric relation over terms which is reduction closed, contextual, and barb preserving.*

In the sequel we will use the following properties of reduction barbed congruence.

Lemma 3.2.6 *If $P \cong Q$ then*

1. *$P \Downarrow n$ if and only if $Q \Downarrow n$;*
2. *$P \rightarrow^* P'$ implies that there exists a process Q' such that $Q \rightarrow^* Q'$ and $P' \cong Q'$.*

3.2.2 Seal dialects and reduction barbed congruence

To acquaint familiarity with reduction barbed congruence, we show that in all Seal dialects the process $P = (\boldsymbol{\nu} x)n[R]$ is not equivalent to $Q = n[(\boldsymbol{\nu} x)R]$. In this way we prove the unsoundness of rule (3.1) informally discussed in Section 3.1. Take

$$R \ = \ \overline{y}^{\eta}(x) \mid x^{\eta}()$$

and consider the context

$$C[-] \ = \ \mathsf{copy}\, n\, \mathsf{as}\, m.y^{n}(u).\overline{u}^{m}().b[] \mid [-]$$

where b is fresh, and η is such that $\mathsf{synch}_u(n, \eta)$ holds in the desired channel interpretation (this also implies $\mathsf{synch}_u(m, \eta)$). Then $C[P] \rightarrow^* P'$ and $P' \downarrow b$ while there is no Q' such that $C[Q] \rightarrow^* Q'$ and $Q' \Downarrow b$.

Although it may appear that the four dialects of Seal define essentially the same language, deep differences are exposed by the study of their behavioural theory. To avoid getting lost in the multitude of semantics, we label $\cong$ with the name of the calculus we are considering. So, $\cong_{eS}$ stands for reduction barbed congruence over Seal with shared channels and the e-condition; similarly for the other dialects.

Observing free names Perhaps the biggest surprise: the e-condition gives a context the power to observe the set of free names of a term. Let P and Q be two processes, and let x be a name such that $x \in \mathrm{fn}(P)$ but $x \notin \mathrm{fn}(Q)$. Independently of their behaviour, the context

$$C[-] \ = \ z^{y}\{\!|n|\!\} \mid y[(\boldsymbol{\nu} x)(\overline{z}^{\uparrow}\{\!|n|\!\} \mid n[\,-\,])]$$

exploits the name x to tell P and Q apart. In fact, while the move of $n[Q]$ out of y can freely occur, the presence of x free in P forbids the move of $n[P]$. The same example applies to *eL*-Seal.

Thus, the e-condition gives contexts a great discriminating power, and it is difficult to justify the resulting equivalence purely in terms of *behaviour*. In fact two processes are not equivalent if they contain different free names even in deadlocked subprocesses.

As an aside remark, this is similar to what happens to object-oriented languages that offer reflection (the capability to examine classes definitions at runtime): contextual equivalence becomes dependent not only on the runtime behaviour of a program, but also on the raw code of the program.

Located Channels The extrusion in Located Seal is subtle, as shown by the following reduction:

$$x^*(u).P \mid (\boldsymbol{\nu} z)z[\overline{x}^{\uparrow}(v).Q] \;\rightarrowtriangle\; (\boldsymbol{\nu} z)(P[u/v] \mid z[Q]) \;.$$

The subtle extrusions rules of the located dialects make controlling interference to be quite difficult. For instance, $(\boldsymbol{\nu} cxy)\overline{c}\{x\} \mid c\{y\} \mid x[P]$ is *not* equivalent to $(\boldsymbol{\nu} cxy)y[P]$: the process P may contain an action $\overline{z}^{\uparrow}(c)$ that notifies the name c to the environment making interference possible. As a consequence, not only the resulting algebraic theory is extremely poor, but programming in located versions of the Seal Calculus is an error prone activity.

***S*-Seal** The Seal Calculus with shared channels and without the e-condition appears to be the calculus more amenable of theoretical investigation. Our intuitions about its semantics have not been revolutionised by basic observations on its behavioural theory. Also it seems to constitute the core of a more than reasonable programming language. For these reasons, in the rest of this Chapter we will focus exclusively on S-Seal, and we will abbreviate its name to Seal. In particular, we will look for a bisimulation based proof method, to learn more about its behaviour, and further investigate its algebraic theory.

3.3 A labelled transition semantics

In Table 3.4 we report a labelled transition system for Seal. Our labelled transition system uses actions of the form

$$A \vdash P \xrightarrow{\ell} P'$$

where A is a finite set of names such that $\mathrm{fn}(P) \subseteq A$; the judgement should be read as 'in a state where names in A may be known by process P and by its environment, the process P can perform ℓ and become P''. This presentation, borrowed from [SV99] and [Sew00], allows us to drop many side conditions when dealing with the extrusion of names.

In the name environment A we use commas to denote disjoint union, that is, A, y means $A \dot\cup \{y\}$ and $A, \vec{y}$ means $A \dot\cup \vec{y}$. We write $P \xrightarrow{\ell}$ as a shorthand for 'there exist a set of names A and a process Q such that $A \vdash P \xrightarrow{\ell} Q$'. We write $\Rightarrow$ to denote the reflexive and transitive closure of $\xrightarrow{\tau}$. We denote the composition of the relations $\mathcal{R}$ and $\mathcal{S}$ as $\mathcal{R}\mathcal{S}$; this notation is also extended to transitions.

Labels The ℓ labels, defined in Table 3.3, give a precise description of how the process P evolves, either autonomously, or by interacting with a possible context. We define an *early* semantics, that is, a semantics where receiving actions are instantiated with their actual arguments when the capability is unleashed, rather than when interaction occurs. This avoids dealing explicitly with process substitutions, and also simplifies the definition of the

Labels				**Activities**				**Locations**			
ℓ	::=	τ	internal action	a	::=	$x^\eta(\vec{y})$	input	γ	::=	$*$	here
	$\mid$	P_z	seal freeze		$\mid$	$\overline{x}^\eta(\vec{y})$	output		$\mid$	z	inside z
	$\mid$	P^z	seal chained		$\mid$	$\overline{x}^\eta\{\!\mid y \mid\!\}$	send				
	$\mid$	$\gamma[a]$	activity a at γ		$\mid$	$\overline{x}^\eta\{\!\mid P \mid\!\}$	capsule				
					$\mid$	$x^\eta\{\!\mid P \mid\!\}$	receive				
					$\mid$	x^η_z	lock				

Table 3.3: Actions

labelled bisimilarity. This is not surprising, as our bisimulation will be a *context bisimulation* in the sense of [San96a], and, in general, a transition in an early LTS clearly identifies the context the process is interacting with.
The free names of a label, $\mathrm{fn}(\ell)$, are defined by the rules below:

$$\mathrm{fn}(\tau) = \emptyset \qquad \mathrm{fn}(P_z) = \mathrm{fn}(P^z) = \{z\} \cup \mathrm{fn}(P) \qquad \mathrm{fn}(\gamma[a]) = \mathrm{fn}(\gamma) \cup \mathrm{fn}(a)$$
$$\mathrm{fn}(x^\eta(\vec{y})) = \mathrm{fn}(\overline{x}^\eta(\vec{y})) = \{x, \vec{y}\} \cup \mathrm{fn}(\eta) \qquad \mathrm{fn}(\overline{x}^\eta\{\!\mid y \mid\!\}) = \{x, y\} \cup \mathrm{fn}(\eta)$$
$$\mathrm{fn}(\overline{x}^\eta\{\!\mid P \mid\!\}) = \mathrm{fn}(x^\eta\{\!\mid P \mid\!\}) = \{x\} \cup \mathrm{fn}(\eta) \cup \mathrm{fn}(P) \qquad \mathrm{fn}(x^\eta_z) = \{x, z\} \cup \mathrm{fn}(\eta)$$

The Seal model of mobility requires a three-party synchronisation between the sender, the receiver, and the seal being moved. The labelled transition system we propose uses intermediate transitions to express partial synchronisations. Actions can be roughly grouped into three sets: the actions that are the direct consequence of the exercise of a capability, those that denote a partial synchronisation, and the internal reduction.

Exercise of capabilities A process can exert a capability, emitting the corresponding action. The corresponding rules are (IN), (OUT), (SND), and (RCV). The actions input $x^\eta(\vec{y})$ and output $\overline{x}^\eta(\vec{y})$ account for communication. They are analogous to the corresponding rules of an early LTS for π-calculus.

The actions send $\overline{x}^\eta\{\!\mid y \mid\!\}$ and receive $x^\eta\{\!\mid Q \mid\!\}$ account for mobility. The first one simply expresses that the process is willing to send a seal named y over the channel x^η. The latter expresses that the process is willing to receive a seal over the channel x^η. The early LTS supposes that the arbitrary seal body Q is provided by the context, and the outcome is exactly the process that would be obtained if P received the seal body Q.

To this group belong also the P_z label. A consequence of the Seal's mobility model is that a seal can at any time be moved by a context. This implies that a seal $z[P]$ must be able to offer itself to be moved. This is accomplished by the (FREEZE) rule: a seal freezes itself, emitting the P_z label, that is by communicating to the environment its name and its content, and turning into an inactive process.

Partial synchronisation Three labels denote that two of the three components of a synchronisation already interacted and are waiting for an action of the third component to

Let

$$P \;=\; \overline{x}^{\eta}\{\!|\, z \,|\!\}.P' \qquad Q \;=\; x^{\eta}\{\!|\, \vec{y} \,|\!\}.Q' \qquad R \;=\; z[S]$$

According to the legend

$$(R \mid P) \mid Q \dashrightarrow \qquad (R \mid Q) \mid P \longrightarrow \qquad (P \mid Q) \mid R \cdots\!\!\rightarrow$$

the diagram below depicts the possible interaction paths.

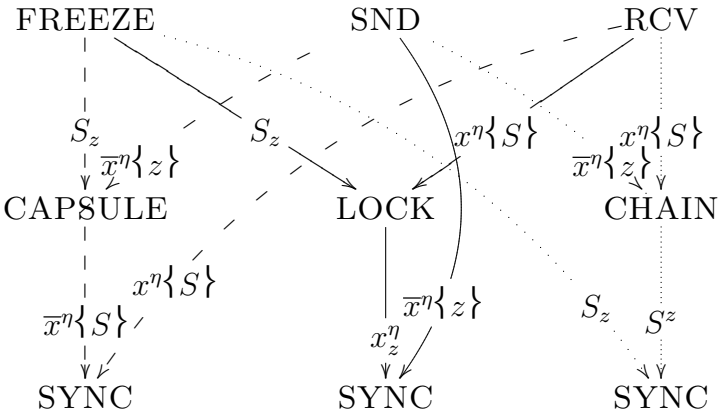


Figure 3.3: Synchronisation paths for mobility

yield an internal reduction.

As the diagram in Figure 3.3 illustrates, the LTS can generate τ transition independently from the order in which labels match. This requires three intermediate labels, called 'capsule' $\overline{x}^{\eta}\{\!|\, P \,|\!\}$, 'lock' x^{η}_z, and 'seal chained' P^z. The label $\overline{x}^{\eta}\{\!|\, P \,|\!\}$ represents the action of serialising a seal: its emission indicates that a process willing to send a seal over a channel found it, serialised it, and is now waiting to synchronise with a receiver. The label x^{η}_z denotes that a process willing to receive a seal at x^{η} synchronised with the corresponding frozen seal named z, and is now looking for a sender. Remark that the names to be extruded are concretized by the (LOCK) rule: the semantics defined by the LTS performs the smallest number of extrusions needed. The label P^z denotes that a process willing to move a seal named z and a process willing to receive a seal with body P synchronised, and are now looking for the frozen seal (named z and whose body is P) to be moved.

Internal reduction The τ label denotes internal synchronisation. Given two matching actions, the (SYNC) rule constructs the process resulting from interaction: it puts the outcomes in parallel and concretizes the extruded names. Matching actions are identified by the Υ relation.

Definition 3.3.1 *Let Υ be the smallest binary* symmetric *relation on labels containing the*

following relation:

$$\{\ (\gamma_1[\overline{x}^{\eta_1}(\vec{y})]\ ,\ \gamma_2[x^{\eta_2}(\vec{y})])\ \mid\ \mathcal{M}\ \}\ \cup\ \{\ (\gamma_1[\overline{x}^{\eta_1}\{\!|S|\!\}]\ ,\ \ \gamma_2[x^{\eta_2}\{\!|S|\!\}])\ \mid\ \mathcal{M}\ \}$$
$$\cup\ \{\ (\gamma_1[x_z^{\eta_1}]\ ,\ \gamma_2[\overline{x}^{\eta_2}\{\!|z|\!\}])\ \mid\ \mathcal{M}\ \}\ \cup\ \{\ (S_z\ ,\ S^z)\ \}$$

where $\mathcal{M} \stackrel{def}{=} (\gamma_1 = \eta_1 = \gamma_2 = \eta_2 = *) \vee (\gamma_1 = * \wedge \mathsf{synch}_{\gamma_2}(\eta_1, \eta_2)) \vee (\gamma_2 = * \wedge \mathsf{synch}_{\gamma_1}(\eta_2, \eta_1))$.

Congruence rules An action observable at top-level can take place either at top-level, or inside a top-level seal. The labels keep track of where an action a takes place by tagging it with $*[a]$ if the action is a top level action, and with $x[a]$ if the action is unleashed inside a seal x. The (SEAL LABEL) rule transforms a $*[a]$ label into a $x[a]$ label provided that the action a can be observed from the outside of the seal x. For instance, in the process $x[z[P]]$ the seal $z[P]$ cannot be observed and this is reflected in LTS by the absence of a transition $x[z[P]] \xrightarrow{x[P_z]}$. The rules (OPEN FREEZE) and (OPEN CAPSULE) are responsible for the extrusion of names as a consequence of mobility actions.

The remaining congruence rules are fairly standard.

3.3.1 Basic properties of the LTS

We prove some properties of the labelled transition semantics. In particular, we investigate how transitions are preserved and reflected by injective renaming, and we prove the correspondence between the LTS and the semantics in chemical style.

First of all, we remark that for a given transition $A \vdash P \xrightarrow{\ell} O$ the structure of the process P and of the outcome O can be characterised up to structural congruence.

Lemma 3.3.2 (Transition analysis) *For* $\mathrm{fn}(P) \subseteq A$*, consider the derivation* $A \vdash P \xrightarrow{\ell} O$*.*

$\ell = *[\overline{x}^{\eta}(\vec{z})]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(\overline{x}^{\eta}(\vec{z}).R_1 \mid R_2)$*, with* $x, \eta \notin \vec{u}$*;*
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \vec{z})(R_1 \mid R_2)$*;*

$\ell = a[\overline{x}^{\uparrow}(\vec{z})]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(\overline{x}^{\uparrow}(\vec{z}).R_1 \mid R_2)] \mid R_3)$*, with* $x \notin (\vec{u} \cup \vec{v})$ *and* $a \notin \vec{u}$*;*
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \vec{z})(a[(\boldsymbol{\nu}\vec{v} \setminus \vec{z})(R_1 \mid R_2)] \mid R_3)$*;*

$\ell = *[x^{\eta}(\vec{z})]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(x^{\eta}(\vec{y}).R_1 \mid R_2)$*, with* $x, \eta, \vec{y}, \vec{z} \notin \vec{u}$*;*
$O \equiv (\boldsymbol{\nu}\vec{u})(R_1[\vec{z}/\vec{y}] \mid R_2)$*;*

$\ell = a[x^{\uparrow}(\vec{z})]$ *implies*

Congruence

(PAR)
$$\frac{A \vdash P \xrightarrow{\ell} P'}{A \vdash P \mid Q \xrightarrow{\ell} P' \mid Q}$$

(RES) $u \notin \mathrm{fn}(\ell)$
$$\frac{A, u \vdash P \xrightarrow{\ell} P'}{A \vdash (\boldsymbol{\nu} u)P \xrightarrow{\ell} (\boldsymbol{\nu} u)P'}$$

(BANG)
$$\frac{A \vdash \alpha.P \xrightarrow{\ell} P'}{A \vdash !\alpha.P \xrightarrow{\ell} P \mid !\alpha.P}$$

(OPEN COM) $y, \eta, \gamma \neq u; u \in \vec{x}$
$$\frac{A, u \vdash P \xrightarrow{\gamma[\overline{y}^{\eta}(\vec{x})]} P'}{A \vdash (\boldsymbol{\nu} u)P \xrightarrow{\gamma[\overline{y}^{\eta}(\vec{x})]} P'}$$

(OPEN FREEZE) $z \notin u; u \in \mathrm{fn}(S)$
$$\frac{A, u \vdash P \xrightarrow{S_z} P'}{A \vdash (\boldsymbol{\nu} u)P \xrightarrow{S_z} P'}$$

(SEAL TAU)
$$\frac{A \vdash P \xrightarrow{\tau} P'}{A \vdash x[P] \xrightarrow{\tau} x[P']}$$

(OPEN CAPSULE) $y, \eta, \gamma \notin \vec{u}; u \in \mathrm{fn}(S)$
$$\frac{A, u \vdash P \xrightarrow{\gamma[\overline{y}^{\eta}\{S\}]} P'}{A \vdash (\boldsymbol{\nu} u)P \xrightarrow{\gamma[\overline{y}^{\eta}\{S\}]} P'}$$

(SEAL LABEL)
$$\frac{A \vdash P \xrightarrow{*[a]} P' \quad a \in \{y^{\uparrow}(\vec{z}), \overline{y}^{\uparrow}(\vec{z}), y^{\uparrow}\{Q\}, \overline{y}^{\uparrow}\{Q\}\}}{A \vdash x[P] \xrightarrow{x[a]} x[P']}$$

Communication

(OUT)
$$\frac{}{A \vdash \overline{x}^{\eta}(\vec{y}).P \xrightarrow{*[\overline{x}^{\eta}(\vec{y})]} P}$$

(IN)
$$\frac{}{A \vdash x^{\eta}(\vec{y}).P \xrightarrow{*[x^{\eta}(\vec{v})]} P[\vec{v}/\vec{y}]}$$

Mobility

(SND)
$$\frac{}{A \vdash \overline{x}^{\eta}\{y\}.P \xrightarrow{*[\overline{x}^{\eta}\{y\}]} P}$$

(RCV)
$$\frac{}{A \vdash x^{\eta}\{\vec{y}\}.P \xrightarrow{*[x^{\eta}\{Q\}]} P \mid y_1[Q] \mid \cdots \mid y_n[Q]}$$

(CAPSULE)
$$\frac{A \vdash P \xrightarrow{S_z} P' \quad A \vdash Q \xrightarrow{*[\overline{x}^{\eta}\{z\}]} Q'}{A \vdash P \mid Q \xrightarrow{*[\overline{x}^{\eta}\{S\}]} P' \mid Q'}$$

(LOCK) $\gamma = \eta = *$ or $(\gamma \neq * \wedge \eta = \uparrow)$
$$\frac{A \vdash P \xrightarrow{S_z} P' \quad A \vdash Q \xrightarrow{\gamma[x^{\eta}\{S\}]} Q'}{A \vdash P \mid Q \xrightarrow{\gamma[x_z^{\eta}]} (\boldsymbol{\nu}\, \mathrm{fn}(S) \setminus A)(P' \mid Q')}$$

(FREEZE)
$$\frac{}{A \vdash x[P] \xrightarrow{P_x} \mathbf{0}}$$

(CHAIN) $\gamma = \eta_1 = \eta_2 = *$ or $(\eta_1 = \gamma \wedge \eta_2 = \uparrow)$
$$\frac{A \vdash P \xrightarrow{*[\overline{x}^{\eta_1}\{y\}]} P' \quad A \vdash Q \xrightarrow{\gamma[x^{\eta_2}\{S\}]} Q'}{A \vdash P \mid Q \xrightarrow{S^y} P' \mid Q'}$$

Synchronisation

(SYNC) $\ell_1 \curlyvee \ell_2$
$$\frac{A \vdash P \xrightarrow{\ell_1} P' \qquad A \vdash Q \xrightarrow{\ell_2} Q'}{A \vdash P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}(\mathrm{fn}(\ell_1) \cup \mathrm{fn}(\ell_2)) \setminus A)(P' \mid Q')}$$

The symmetric rules for (PAR), (CAPSULE), (LOCK), and (CHAIN) are omitted.

Table 3.4: A labelled transition system for the Seal Calculus

$P \equiv (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(x^{\uparrow}(\vec{y}).R_1 \mid R_2)] \mid R_3)$, *with* $x, \vec{y}, \vec{z} \notin (\vec{u} \cup \vec{v})$, *and* $a \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \{a\})(a[(\boldsymbol{\nu}\vec{v})(R_1[\vec{z}/\vec{y}] \mid R_2)] \mid R_3)$;

$\ell = S_z$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(z[S] \mid R_1)$, *with* $z \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \mathrm{fn}(S))R_1$;

$\ell = S^z$ *implies either*

$P \equiv (\boldsymbol{\nu}\vec{u})(\overline{x}^{*}\{z\}.R_1 \mid x^{*}\{\vec{y}\}.R_2 \mid R_3)$ *with* $z, \mathrm{fn}(S) \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u})(R_1 \mid R_2 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_3)$;

or

$P \equiv (\boldsymbol{\nu}\vec{u})(\overline{x}^{a}\{z\}.R_1 \mid R_2 \mid a[x^{\uparrow}\{\vec{y}\}.R_3 \mid R_4])$ *with* $z \notin \vec{u}$, $\mathrm{fn}(S) \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u})(R_1 \mid R_2 \mid a[y_1[S] \mid \cdots \mid y_n[S] \mid R_3 \mid R_4])$;

$\ell = *[\overline{x}^{\eta}\{y\}]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(\overline{x}^{\eta}\{y\}.R_1 \mid R_2)$, *with* $x, \eta, y \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u})(R_1 \mid R_2)$;

$\ell = *[\overline{x}^{\eta}\{S\}]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(\overline{x}^{\eta}\{y\}.R_1 \mid y[S] \mid R_2)$, *with* $x, \eta \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \mathrm{fn}(S))(R_1 \mid R_2)$;

$\ell = a[\overline{x}^{\uparrow}\{S\}]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(\overline{x}^{\uparrow}\{y\}.R_1 \mid y[S] \mid R_2)] \mid R_3)$, *with* $x \notin (\vec{u} \cup \vec{v})$, *and* $a \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \mathrm{fn}(S))(a[(\boldsymbol{\nu}\vec{v} \setminus \mathrm{fn}(S))(R_1 \mid R_2)] \mid R_3)$;

$\ell = *[x^{\eta}\{S\}]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(x^{\eta}\{\vec{y}\}.R_1 \mid R_2)$, *with* $x, \eta, \mathrm{fn}(S) \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u})(R_1 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_2)$;

$\ell = a[x^{\uparrow}\{S\}]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(x^{\uparrow}\{\vec{y}\}.R_1 \mid R_2)] \mid R_3)$, *with* $x, y, \mathrm{fn}(S) \notin \vec{u} \cup \vec{v}$, *and* $a \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u} \setminus \{a\})(a[(\boldsymbol{\nu}\vec{v})(R_1 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_2)] \mid R_3)$;

$\ell = *[x^{*}_{z}]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(z[S] \mid x^{*}\{\vec{y}\}.R_1 \mid R_2)$, *with* $x, z \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u})(R_1 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_2)$;

$\ell = a[x^{\uparrow}_{z}]$ *implies*

$P \equiv (\boldsymbol{\nu}\vec{u})(z[S] \mid R_1 \mid a[(\boldsymbol{\nu}\vec{v})(x^{\uparrow}\{\vec{y}\}.R_2 \mid R_3)])$, *with* $z \notin \vec{u}$, $x \notin \vec{u} \cup \vec{v}$, *and* $a \notin \vec{u}$;
$O \equiv (\boldsymbol{\nu}\vec{u})(R_1 \mid a[(\boldsymbol{\nu}\vec{v})(R_2 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_3)])$;

where $\vec{u}$ *and* $\vec{v}$ *are disjoint.*

Proof Induction on derivations of transitions.[3] □

We now show how injective renaming preserves and reflects transitions. This theorems are fundamental to prove that the bisimulation based equivalence that we will build on top of the LTS is preserved by injective substitutions. It is convenient to introduce some special notations for injective substitutions. Given an injective function $f : A \to B$ between two finite sets of names $A = \{a_1, \ldots, a_n\}$ and B, and a process P such that $\mathrm{fn}(P) \subseteq A$, we write fP for $P[f(a_1), \ldots, f(a_n)/a_1, \ldots, a_n]$. We denote $\mathrm{dom}(f)$ and $\mathrm{ran}(f)$ respectively the domain and the range of f. Given $f : A \to B$ and $A' \subseteq A$, we write $f(\boldsymbol{\nu}A')$ for f restricted to A'. Given $f : A_1 \to B_1$ and $g : A_2 \to B_2$ with $A_1 \cap A_2 = \emptyset$, we write $f+g : A_1 \dot\cup A_2 \to B_1 \cup B_2$ for their set theoretic union.

Lemma 3.3.3 (Injective substitution) *If* $A \vdash P \xrightarrow{\ell} P'$, *and* $f : A \to B$ *and* $g : (\mathrm{fn}(\ell) \setminus A) \to (\mathbf{N} \setminus B)$ *are injective, then* $B \vdash fP \xrightarrow{(f+g)\ell} (f+g)P'$.

Proof Induction on derivations of transitions.

(IN) Consider $A \vdash x^{\eta}(\vec{y}).P \xrightarrow{*[x^{\eta}(\vec{v})]} P[\vec{v}/\vec{y}]$. By alpha conversion, we suppose $\vec{y} \notin (A \cup B \cup (\mathrm{fn}(\ell) \setminus A) \cup \mathrm{ran}(g))$. Then $f(x^{\eta}(\vec{y}).P) = fx^{f\eta}(\vec{y}).fP$ and $\mathrm{fn}(fx^{f\eta}(\vec{y}).fP) \subseteq B$. By (IN) $B \vdash fx^{f\eta}(\vec{y}).fP \xrightarrow{*[fx^{f\eta}((f+g)\vec{v})]} fP[(f+g)\vec{v}/\vec{y}]$. Now $\mathrm{fn}(P) \subseteq A, \vec{y}$ so $\mathrm{fn}(P) \cap \mathrm{dom}(g) = \emptyset$, so $fP = (f+g)P$. Hence $fP[(f+g)\vec{v}/\vec{y}] = (f+g)P[(f+g)\vec{v}/\vec{y}] = (f+g)(P[\vec{v}/\vec{y}])$, so we have the transition $B \vdash fx^{f\eta}(\vec{y}).fP \xrightarrow{(f+g)*[x^{\eta}(\vec{v})]} (f+g)P[\vec{v}/\vec{y}]$.

(RCV) Consider $A \vdash x^{\eta}\{\vec{y}\}.P \xrightarrow{*[x^{\eta}\{Q\}]} P \mid y_1[Q] \mid \cdots \mid y_n[Q]$. By (RCV) $B \vdash fx^{f\eta}\{f\vec{y}\}.fP \xrightarrow{*[fx^{f\eta}\{Q'\}]} fP \mid fy_1[Q'] \mid \cdots \mid fy_n[Q']$ for all processes Q'. In particular, taking $Q' = (f+g)Q$, $B \vdash fx^{f\eta}\{f\vec{y}\}.fP \xrightarrow{*[fx^{f\eta}\{(f+g)Q\}]} fP \mid fy_1[(f+g)Q] \mid \cdots \mid fy_n[(f+g)Q]$. As $\mathrm{fn}(P) \subseteq A$ and $\vec{y} \subseteq A$, we have the transition $B \vdash f(x^{\eta}\{\vec{y}\}.P) \xrightarrow{(f+g)(*[x^{\eta}\{Q\}])} (f+g)(P \mid y_1[Q] \mid \cdots \mid y_n[Q])$.

(OPEN COM) Define $f' : (A, u) \to (B, g(u))$ as $f + (u \mapsto g(u))$, and $g' = g \restriction (\mathrm{fn}(\gamma[\overline{y}^{\eta}(\vec{x})]) \setminus \{u\})$. By the induction hypothesis $B, g(u) \vdash f'P \xrightarrow{(f'+g')\gamma[\overline{y}^{\eta}(\vec{x})]} (f'+g')P'$, so by (OPEN COM) $B \vdash (\boldsymbol{\nu}gu)f'P \xrightarrow{(f'+g')\gamma[\overline{y}^{\eta}(\vec{x})]} (f'+g')P'$. So, as $f'+g' = f+g$ we have $B \vdash f(\boldsymbol{\nu}u)P \xrightarrow{(f+g)\gamma[\overline{y}^{\eta}(\vec{x})]} (f+g)P'$.

(OPEN FREEZE),(OPEN CAPSULE) Similar to (OPEN COM).

(SYNC) The argument depends on the shape of the matching labels; we detail the case when synchronisation is communication. Other cases are similar. We also suppose $\ell_2 = \gamma[x^{\eta_2}(\vec{y})]$. $\mathrm{fn}(\tau) = \emptyset$, so we have $f : A \to B$ and $g : \emptyset \to (\mathbf{N} \setminus B)$. Take some

[3] To avoid unnecessary complications in the grammar of the labels, we included the label $[\overline{x}^{\eta}\{y\}]$ even if no process can emit it.

$g' : (\mathrm{fn}(\gamma_2[x^{\eta_2}(\vec{y})]) \setminus A) \to (\mathbf{N} \setminus B)$ injective. By the induction hypothesis and (COM) we have

$$\frac{B \vdash fP \xrightarrow{(f+g')\gamma_1[\overline{x}^{\eta_1}(\vec{y})]} (f+g')P' \qquad B \vdash fQ \xrightarrow{(f+g')\gamma_2[x^{\eta_2}(\vec{y})]} (f+g')Q'}{B \vdash f(P \mid Q) \xrightarrow{\tau} (\boldsymbol{\nu}((f+g')\vec{y}) \setminus B)(f+g')(P' \mid Q')}$$

Now $(f+g')\vec{y} \setminus B = \mathrm{ran}(g')$, so $B \vdash f(P \mid Q) \xrightarrow{\tau} (\boldsymbol{\nu}\,\mathrm{ran}(g'))(f+g')(P' \mid Q')$. We have $f((\boldsymbol{\nu}\,\mathrm{dom}(g'))(P' \mid Q')) = (\boldsymbol{\nu}\,\mathrm{ran}(g'))(f+g')(P' \mid Q')$, so $B \vdash f(P \mid Q) \xrightarrow{\tau} f((\boldsymbol{\nu}(\vec{y} \setminus B))(P' \mid Q'))$.

(LOCK) Similar to (SYNC).

(OUT),(SND),(FREEZE) Immediate.

(CAPSULE), (CHAIN), (PAR), (BANG), (SEAL TAU), (SEAL LABEL) Straightforward use of the induction hypothesis. □

Lemma 3.3.4 (Shifting)

1. $(A \vdash P \xrightarrow{\gamma[x^{\eta}(\vec{u})]} P' \wedge v \in (\vec{u} \setminus A))$ *if and only if* $(A, v \vdash P \xrightarrow{\gamma[x^{\eta}(\vec{u})]} P' \wedge v \in \vec{u} \setminus \mathrm{fn}(P))$;
2. $(A \vdash P \xrightarrow{\gamma[x^{\eta}\{Q\}]} P' \wedge v \in (\mathrm{fn}(Q) \setminus A))$ *if and only if* $(A, v \vdash P \xrightarrow{\gamma[x^{\eta}\{Q\}]} P' \wedge v \in \mathrm{fn}(Q) \setminus \mathrm{fn}(P))$;
3. $(A \vdash P \xrightarrow{Q^y} P' \wedge v \in (\mathrm{fn}(Q) \setminus A))$ *if and only if* $(A, v \vdash P \xrightarrow{Q^y} P' \wedge v \in \mathrm{fn}(Q) \setminus \mathrm{fn}(P))$.

Proof We detail the second part, others are similar (Part 3. depends on Part 2.).

We want to show that $A \vdash P \xrightarrow{\gamma[x^{\eta}\{Q\}]} P'$ if and only if $A, v \vdash P \xrightarrow{\gamma[x^{\eta}\{Q\}]} P'$, where $v \in \mathrm{fn}(Q) \setminus A$. We perform two inductions on derivations of transitions.

(RCV) Straightforward.

(PAR),(BANG) Straightforward use of the induction hypothesis.

(RES) Consider

$$\frac{A, y \vdash P \xrightarrow{\gamma[x^{\eta}\{Q\}]} P'}{A \vdash (\boldsymbol{\nu}y)P \xrightarrow{\gamma[x^{\eta}\{Q\}]} (\boldsymbol{\nu}y)P'} \quad v \in (\mathrm{fn}(Q) \setminus A),\ y \notin \gamma, x, \eta \qquad \frac{A, v, y \vdash P \xrightarrow{\gamma[x^{\eta}\{Q\}]} P'}{A, v \vdash (\boldsymbol{\nu}y)P \xrightarrow{\gamma[x^{\eta}\{Q\}]} (\boldsymbol{\nu}y)P'} \quad v \in (\mathrm{fn}(Q) \setminus \mathrm{fn}((\boldsymbol{\nu}y)P)),\ y \notin \gamma, x, \eta$$

For the left-to-right implication, note that $v \in \mathrm{fn}(Q) \setminus (A, y)$, so by the induction hypothesis $A, v, y \vdash P \xrightarrow{\gamma[x^{\eta}\{Q\}]} P'$ and $v \in (\mathrm{fn}(Q) \setminus \mathrm{fn}((\boldsymbol{\nu}y)P))$. For the right-to-left implication, note that as A, v, y is well-formed we have $x \in \mathrm{fn}(Q) \setminus \mathrm{fn}(P)$, so by induction hypothesis $A, y \vdash P \xrightarrow{\gamma[x^{\eta}\{Q\}]} P'$ and $v \in \mathrm{fn}(Q) \setminus (A, y)$.

The other cases are vacuous. □

The transitions of an injectively substituted process fP are determined by the transitions of P as follows.

Lemma 3.3.5 (Converse to injective substitution) *For $f : A \to B$ injective, if $B \vdash fP \xrightarrow{\ell'} Q'$ then*

1. *there exist a label ℓ and a process Q,*
2. *there exist two sets of names H, I with $H \cap I = \emptyset$, $H \cup I = \mathrm{fn}(\ell) \setminus A$,*
3. *there exists $g : I \to B'$ bijective with $B' \cap B = \emptyset$,*
4. *there exists $h : H \to (B \setminus \mathrm{ran}(f))$ injective,*

such that

a. *$A \vdash P \xrightarrow{\ell} Q$, and $\ell' = (f + g + h)\ell$, and $Q' = (f + g + h)Q$; and*

b. *if $\ell' \notin \{\gamma[x^\eta(\vec{y})], \gamma[x^\eta\{R\}], R^y \mid \text{for some } \gamma, x, \eta, \vec{y}, R\}$ then $H = \emptyset$.*

Proof Induction on derivations of transitions.

(OUT) We have $B \vdash f(\overline{x}^\eta(\vec{y}).P) \xrightarrow{f(*[\overline{x}^\eta(\vec{y})])} fP$. Take $Q = P$, $\ell = *[\overline{x}^\eta(\vec{y})]$. Also take $H = I = \emptyset$, $g : \emptyset \to \emptyset$, $h : \emptyset \to B \setminus \mathrm{ran}(f)$. Then $A \vdash \overline{x}^\eta(\vec{y}).P \xrightarrow{*[\overline{x}^\eta(\vec{y})]} P$.

(IN) We can suppose without loss of generality that $\vec{y} \notin A \cup B$. Then we have $B \vdash f(x^\eta(\vec{y}).P) \xrightarrow{*[fx^{f\eta}(\vec{z})])} (fP)[\vec{z}/\vec{y}]$. We scan $\vec{z}$ according to the rules below, and in doing so we construct a new set of names $\vec{v}$, and H, I, B', g, h. Start with $I = H = B' = \emptyset$, and g, h always undefined functions. Then, for all i:

- if $z_i \in \mathrm{ran}(f)$ then $v_i = u$ for $u \in A$ such that $f(u) = z_i$;
- if $z_i \in B \setminus \mathrm{ran}(f)$ and for all $j < i$, $z_j \neq z_i$, then take some fresh o_i such that $o_i \notin A$. Add o_i to H, add $h : o_i \mapsto z_i$ to the current graph of h, and let $v_i = o_i$; if $z_j = z_i$ for some $j < i$, then let $v_i = v_j$;
- if $z_i \notin B$ and for all $j < i$, $z_j \neq z_i$, then take some new o_i such that $o_i \notin A$. Add o_i to I, add z_i to B', add $g : o_i \mapsto z_i$ to the current graph of g, and let $v_i = o_i$; if $z_j = z_i$ for some $j < i$, then let $v_i = v_j$.

Take $\ell = *[x^\eta(\vec{v})]$, $Q = P[\vec{v}/\vec{y}]$. Also take the H, I, g, h constructed while generating $\vec{v}$. Then $H \cap I = \emptyset$, $H \cup I = \mathrm{fn}(\ell) \setminus A$, $g : I \to B'$ is a bijection, $h : H \to (B \setminus \mathrm{ran}(f))$ is injective. Moreover, $\ell' = (f + g + h)\ell$, $Q' = (f + g + h)Q$, and $A \vdash x^\eta(\vec{y}).P \xrightarrow{*[x^\eta(\vec{v})]} P[\vec{v}/\vec{y}]$.

(SND), (FREEZE) Similar to (OUT).

(RCV) We have $B \vdash f(x^\eta\{\vec{y}\}.P) \xrightarrow{fx^{f\eta}\{R'\}} fP \mid fy_1[R'] \mid \ldots \mid fy_n[R']$. We construct $\vec{v}, H, I, B', g, h$ as we have done in (INP), but where $\vec{z}$ is replaced by $\overrightarrow{\mathrm{fn}(R')}$ (the ordering is not important). Then let $R = R'[\vec{v}/\overrightarrow{\mathrm{fn}(R')}]$, and take $\ell = *[x^\eta\{R\}]$, and $Q = P \mid y_1[R] \mid \ldots \mid y_n[R]$.

(LOCK) Consider an instance

$$\frac{B \vdash fP \xrightarrow{S_z} P' \quad B \vdash fQ \xrightarrow{\gamma[x^\eta\{S\}]} Q'}{B \vdash f(P \mid Q) \xrightarrow{\gamma[x^\eta_z]} (\boldsymbol{\nu}\, \mathrm{fn}(S) \setminus B)(P' \mid Q')}$$

By induction hypothesis, there exist P_0', S_0, z_0 and $g : (\mathrm{fn}(S_0) \setminus A) \to B'$ such that $A \vdash P \xrightarrow{S_0 z_0} P_0'$ and $(f+g)S_0 = S$, $f(z_0) = z$, $(f+g)P_0' = P'$. This also forces H to be empty. Again, by induction hypothesis, there exist $Q_1', \gamma_1, x_1, \eta_1, S_1$ and $g' : (\mathrm{fn}(S_1) \setminus A) \to B'$ such that $A \vdash Q \xrightarrow{\gamma_1[x_1 \eta_1 \{\!\{ S_1 \}\!\}]} Q_1'$ and $(f+g')S_1 = S$, $(f+g')Q_1' = Q'$, $f\gamma_1 = \gamma$, $f\eta_1 = \eta$. By Lemma 3.3.3, as g and g' are bijection, $A \vdash Q \xrightarrow{\gamma_1[x_1 \eta_1 \{\!\{ S_0 \}\!\}]} g^{-1}(g'(Q_1'))$. By (LINK), $A \vdash P \mid Q \xrightarrow{\gamma_1[x_1 \eta_1 z_1]} (\boldsymbol{\nu}\, \mathrm{fn}(S_0) \setminus A)(P' \mid Q')$. It remains only to note that $(f+g)((\boldsymbol{\nu}\, \mathrm{fn}(S_0) \setminus A)(P_0' \mid g^{-1}(g'(Q_1'))) = (\boldsymbol{\nu}\, \mathrm{fn}(S) \setminus B)(P' \mid Q')$.

(CHAIN),(CAPSULE), (SYNC) These cases are similar to (LOCK).

(PAR), (BANG), (SEAL TAU), (SEAL LABEL) By the induction hypothesis.

(RES) Consider an instance

$$\frac{A, u' \vdash P' \xrightarrow{\ell'} Q'}{A \vdash (\boldsymbol{\nu} u')P' \xrightarrow{\ell'} (\boldsymbol{\nu} u')Q'}\ u' \notin \mathrm{fn}(\ell)$$

where $fP = (\boldsymbol{\nu} u')P'$. There exists $u \notin A$ such that $P = (\boldsymbol{\nu} u)P_0$ and $P' = (f + [u'/u])P_0$. By the induction hypothesis there exist ℓ, Q_0, H, I and $g' : I \to B'$, $h : H \to (B \setminus \mathrm{ran}(f)$, where $H \cup I = \mathrm{fn}(\ell) \setminus (A \cup \{u\})$, such that $A, u \vdash P_0 \xrightarrow{\ell} Q_0$, and $\ell' = (f + [u'/u] + g' + h)\ell$, and $Q' = (f + [u'/u] + g' + h)Q_0$. Since $u' \notin \mathrm{fn}(\ell')$, it follows that $u \not/\mathrm{fn}(\ell)$. By (RES), $A \vdash P \xrightarrow{\ell} (\boldsymbol{\nu} u)Q_0$. The case follows taking $g = g' + [u'/u]$ and $Q = (\boldsymbol{\nu} u)Q_0$.

(OPEN COM) Consider an instance

$$\frac{B, u' \vdash P' \xrightarrow{\gamma[\overline{y}^{\eta}(\vec{x}')]} Q'}{B \vdash (\boldsymbol{\nu} u')P' \xrightarrow{\gamma[\overline{y}^{\eta}(\vec{x}')]} Q'} y, \eta, \gamma \neq u'; u' \in \vec{x}'$$

where $fP = (\boldsymbol{\nu} u')P'$. There exist $u \notin A$ and P_0 such that $P = (\boldsymbol{\nu} u)P_0$ and $P' = (f + [u'/u])P_0$. By induction hypothesis, there exist $\ell, Q, g' : \mathrm{fn}(\ell \setminus (A, u)) \to B'$ such that $A, u \vdash P_0 \xrightarrow{\ell} Q$, where $\ell = \gamma_0[\overline{y_0}^{\eta_0}(\vec{x}_0)]$, and $f\gamma_0 = \gamma$, $fy_0 = y$, $f\eta_0 = \eta$ and $(f + g' + [u'/u])\vec{x}_0 = \vec{x}$. By (OPEN COM) $A \vdash P \xrightarrow{\gamma_0[\overline{y_0}^{\eta_0}(\vec{x}_0)]} Q$, and the case follows taking $g = g' + [u'/u]$.

(OPEN FREEZE), (OPEN CAPSULE) Similar to (OPEN COM). □

Corollary 3.3.6 *Let $f : A \to B$ injective, and $\mathrm{fn}(P) \subseteq A$. $A \vdash P \xrightarrow{\tau} P'$ if and only if $B \vdash fP \xrightarrow{\tau} fP'$.*

To conclude this section, we state and prove the equivalence between the LTS and the semantics in chemical style.

Lemma 3.3.7 *If $A \vdash P \xrightarrow{\ell} Q$ and $P' \equiv P$ then $A \vdash P' \xrightarrow{\ell} Q'$ for some $Q' \equiv Q$.*

Proof Induction on the size of the derivation of $P' \equiv P$. □

Theorem 3.3.8 *Let P be a process:*

1. *if* $\mathrm{fn}(P) \subseteq A$ *and* $A \vdash P \xrightarrow{\tau} Q$, *then* $P \twoheadrightarrow Q$;
2. *if* $P \twoheadrightarrow Q$ *then there exists* $A \supseteq \mathrm{fn}(P)$ *such that* $A \vdash P \xrightarrow{\tau} Q'$, *where* $Q' \equiv Q$.

Proof Both parts are proved by transition induction.

Part 1. If there is a derivation of $A \vdash P \xrightarrow{\tau} Q$, then one of the following applies:

1. the last rule applied to derive $A \vdash P \xrightarrow{\tau} Q$ is (SYNC);
2. there are processes P', Q' and a context $C[-]$ such that $P = C[P']$, $Q = C[Q']$, and $A \vdash P' \xrightarrow{\tau} Q'$.

We proceed by induction on the structure of the context $C[-]$.

Base Case There are two processes P_1 and P_2 such that $P = P_1 \mid P_2$, $A \vdash P_1 \xrightarrow{\ell_1} P_1'$, $A \vdash P_2 \xrightarrow{\ell_2} P_2'$, and $\ell_1 \curlyvee \ell_2$. Lemma 3.3.2 describes the structure of a process P and the outcome O for each label ℓ after each transition. It remains to verify that the LTS and the $\curlyvee$ relation characterise matching processes in the reduction relation. This requires checking all the possible matching labels in the $\curlyvee$ relation. We detail the case for $\ell_1 = *[\overline{x}^a\{\!\{ S \}\!\}]$, $\ell_2 = a[x^\uparrow\{\!\{ S \}\!\}]$. Other cases are similar.

The derivation $P_1 \xrightarrow{*[\overline{x}^a\{\!\{S\}\!\}]} P_1'$ implies that $P_1 \equiv (\boldsymbol{\nu}\vec{z})(\overline{x}^a\{\!\{ y \}\!\}.R_1 \mid y[S] \mid R_2)$ and $P_1' \equiv (\boldsymbol{\nu}\vec{z} \setminus \mathrm{fn}(S))(R_1 \mid R_2)$, with $x, a \notin \vec{z}$.

The derivation $P_2 \xrightarrow{a[x^\uparrow\{\!\{S\}\!\}]} P_2'$ implies that $P_2 \equiv (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(x^\uparrow\{\!\{ \vec{y} \}\!\}.R_1 \mid R_2)] \mid R_3)$ and $P_2' \equiv (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(R_1 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_2)] \mid R_3)$, with $x, y, \mathrm{fn}(S) \notin (\vec{u} \cup \vec{v})$ and $a \notin \vec{u}$.

Observe that $\mathrm{fn}(S) \setminus A = \mathrm{fn}(S) \cap \vec{z}$. It holds

$$\begin{aligned} P_1 \mid P_2 \;&\equiv\; (\boldsymbol{\nu}\vec{z})(\overline{x}^a\{\!\{ y \}\!\}.R_1 \mid y[S] \mid R_2) \mid (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(x^\uparrow\{\!\{ \vec{y} \}\!\}.R_1 \mid R_2)] \mid R_3) \\ &\twoheadrightarrow\; (\boldsymbol{\nu}(\mathrm{fn}(S) \cup \{a\}) \setminus A)((\boldsymbol{\nu}\vec{z} \setminus \mathrm{fn}(S))(R_1 \mid R_2) \\ &\qquad\qquad \mid (\boldsymbol{\nu}\vec{u})(a[(\boldsymbol{\nu}\vec{v})(R_1 \mid y_1[S] \mid \cdots \mid y_n[S] \mid R_2)] \mid R_3)) \\ &\equiv\; P_1' \mid P_2' \;. \end{aligned}$$

Inductive step Follows easily from the congruence laws of reduction.

Part 2. We detail one of the base cases, the others are similar. Rule (move out) says:

$$\begin{aligned} x^a\{\!\{ \vec{u} \}\!\}.P \mid a[(\boldsymbol{\nu}\vec{z})(\overline{x}^\uparrow\{\!\{ v \}\!\}.Q_1 \mid v[R] \mid Q_2)] \\ \twoheadrightarrow\quad (\boldsymbol{\nu}\,\mathrm{fn}(R) \cap \vec{z})(P \mid u_1[R] \mid \cdots \mid u_n[R] \mid a[(\boldsymbol{\nu}\vec{z} \setminus \mathrm{fn}(R))(Q_1 \mid Q_2)]) \end{aligned}$$

where $x \notin \vec{z}$. Let $A = \mathrm{fn}(P)$. From the LTS, we can derive:

1. $A \vdash x^a\{\!\{ \vec{u} \}\!\}.P \xrightarrow{*[x^a\{\!\{R\}\!\}]} P \mid u_1[R] \mid \cdots \mid u_n[R]$

2. As $x \notin \vec{z}$ the following is a valid derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A \vdash \overline{x}^{\uparrow}\{\!|v|\!\}.Q_1 \xrightarrow{*[\overline{x}^{\uparrow}\{\!|v|\!\}]} Q_1 \qquad A \vdash v[R] \xrightarrow{R_v} \mathbf{0}}{A \vdash \overline{x}^{\uparrow}\{\!|v|\!\}.Q_1 \mid v[R] \xrightarrow{*[\overline{x}^{\uparrow}\{\!|R|\!\}]} Q_1 \mid \mathbf{0}}}{A \vdash \overline{x}^{\uparrow}\{\!|v|\!\}.Q_1 \mid v[R] \mid Q_2 \xrightarrow{*[\overline{x}^{\uparrow}\{\!|R|\!\}]} Q_1 \mid \mathbf{0} \mid Q_2}}{A \vdash (\boldsymbol{\nu}\vec{z})(\overline{x}^{\uparrow}\{\!|v|\!\}.Q_1 \mid v[R] \mid Q_2) \xrightarrow{*[\overline{x}^{\uparrow}\{\!|R|\!\}]} (\boldsymbol{\nu}\vec{z} \setminus \mathrm{fn}(R))(Q_1 \mid \mathbf{0} \mid Q_2)}}{A \vdash a[(\boldsymbol{\nu}\vec{z} \setminus \mathrm{fn}(R))(\overline{x}^{\uparrow}\{\!|v|\!\}.Q_1 \mid v[R] \mid Q_2)] \xrightarrow{a[\overline{x}^{\uparrow}\{\!|R|\!\}]} a[(\boldsymbol{\nu}\vec{z} \setminus \mathrm{fn}(R))(Q_1 \mid \mathbf{0} \mid Q_2)]}$$

As $*[x^a\{\!|R|\!\}] \curlyvee a[\overline{x}^{\uparrow}\{\!|R|\!\}]$, we can apply the rule (SYNC) to derive

$$\begin{aligned} A \vdash x^a\{\!|\vec{u}|\!\}.P \mid a[(\boldsymbol{\nu}\vec{z})(\overline{x}^{\uparrow}\{\!|v|\!\}.Q_1 \mid v[R] \mid Q_2)] \\ \xrightarrow{\tau} (\boldsymbol{\nu}\,\mathrm{fn}(R) \cap \vec{z})(P \mid u_1[R] \mid \cdots \mid u_n[R] \mid a[(\boldsymbol{\nu}\vec{z} \setminus \mathrm{fn}(R))(Q_1 \mid \mathbf{0} \mid Q_2)]) \end{aligned}$$

as desired.

To complete the inductive step, observe that the congruence cases follow because the label τ can cross every context (rules (PAR), (RES), (SEAL TAU)), and the insensitivity to structural congruence follows from Lemma 3.3.7. □

3.4 A proof method

In this section we use the LTS defined in the previous section to define a bisimulation based relation contained in reduction barbed congruence. This will be an useful proof method to ensure equivalence between terms, and will be subsequently used to explore the behavioural theory of S-Seal. At the same time, its development will point out some peculiarities of Seal semantics.

3.4.1 LTS and observable actions

We first reexamine the definition of reduction barbed congruence, and we show that it is very robust under changes to the precise definition of barbs.

The predicate $P \downarrow n$ detects the ability of a process P to interact with the environment via the seal n. In other process calculi, like the π-calculus, barbs are defined using visible actions. We show that reduction barbed congruence is not affected if we change our definition of barb with barbs inherited from the *visible* actions of the LTS. We refer to barbs of Definition 3.2.3 as *natural barbs*.

First, we remark that the exposure of a barb corresponds to the emission of a 'freeze' label in the labelled transition system:

Lemma 3.4.1 *$P \downarrow n$ if and only if $A \vdash P \xrightarrow{Q_n} P'$ for some P', Q, and A, with* $\mathrm{fn}(P) \subseteq A$.

Proof The 'if' part is a consequence of Lemma 3.3.2, while the 'only if' part follows straightforwardly from Definition 3.2.5. □

$$C_{*[\overline{x}^*\{y\}]}[-] = y[\overline{a}^\uparrow()] \mid x^*\{i\}.a^i().o[\,] \mid - \qquad C_{*[x_z^*]}[-] = \overline{x}^*\{z\}.o[\,] \mid -$$
$$C_{*[\overline{x}^\uparrow\{y\}]}[-] = y[\overline{a}^\uparrow()] \mid x^u\{i\}.a^i().o[\,] \mid u[-] \qquad C_{k[x_z^\uparrow]}[-] = \overline{x}^k\{z\}.o[\,] \mid -$$

$$C_{*[x^*\{S\}]}[-] = a[S] \mid \overline{x}^*\{a\}.o[\,] \mid - \qquad C_{*[\overline{x}^*\{\star\}]}[-] = x^*\{a\}.o[\,] \mid -$$
$$C_{*[x^\uparrow\{S\}]}[-] = a[S] \mid \overline{x}^u\{a\}.o[\,] \mid u[-] \qquad C_{*[\overline{x}^\uparrow\{\star\}]}[-] = x^u\{a\}.o[\,] \mid u[-]$$
$$C_{k[x^\uparrow\{S\}]}[-] = a[S] \mid \overline{x}^k\{a\}.o[\,] \mid - \qquad C_{k[\overline{x}^\uparrow\{\star\}]}[-] = x^k\{a\}.o[\,] \mid -$$

where a, i, o, u are fresh names.

Table 3.5: Contexts that transform a λ-barb into a natural barb

We now prove that for all classes of visible actions of our LTS the resulting definitions of barbed congruence collapse and coincide with $\cong$. For that we classify the actions of the LTS according to the axiom or rule responsible for generating them (that is, according to their 'shape'):

Definition 3.4.2 (λ-barbs) *We define the following sets of actions*

$$\mathcal{L}_{*SND^*} = \{\ *[\overline{x}^*\{y\}] \mid \textit{for all } x, y\ \} \qquad \mathcal{L}_{*LOCK^*} = \{\ *[x_z^*] \mid \textit{for all } x, z\ \}$$
$$\mathcal{L}_{*SND^\uparrow} = \{\ *[\overline{x}^\uparrow\{y\}] \mid \textit{for all } x, y\ \} \qquad \mathcal{L}_{*LOCK^\uparrow} = \{\ *[x_z^\uparrow] \mid \textit{for all } x, z\ \}$$

$$\mathcal{L}_{*RCV^*} = \{\ *[x^*\{S\}] \mid \textit{for all } x, S\ \} \qquad \mathcal{W}_{*CAPS^*} = \{\ *[\overline{x}^*\{S\}] \mid \textit{for all } x, S\ \}$$
$$\mathcal{L}_{*RCV^\uparrow} = \{\ *[x^\uparrow\{S\}] \mid \textit{for all } x, S\ \} \qquad \mathcal{W}_{*CAPS^\uparrow} = \{\ *[\overline{x}^\uparrow\{S\}] \mid \textit{for all } x, S\ \}$$
$$\mathcal{L}_{kRCV^\uparrow} = \{\ k[x^\uparrow\{S\}] \mid \textit{for all } x, S\ \} \qquad \mathcal{W}_{kCAPS^\uparrow} = \{\ k[\overline{x}^\uparrow\{S\}] \mid \textit{for all } x, S\ \}$$

ans we refer to each of them as a class of actions.

Let $\mathcal{L} = \bigcup_\lambda \mathcal{L}_\lambda$. For $\omega \in \mathcal{L}$, we write $P \downarrow \omega$ if $P \xrightarrow{\omega}$, and $P \Downarrow \omega$ if $P \Rightarrow \xrightarrow{\omega}$.

Let $\mathcal{W} = \bigcup_\lambda \mathcal{W}_\lambda$. We write $P \downarrow \gamma[\overline{x}^\eta\{\star\}]$ if there exists a process S such that $P \xrightarrow{\gamma[\overline{x}^\eta\{S\}]}$. We write $P \Downarrow \gamma[\overline{x}^\eta\{\star\}]$ if there exists a process S such that $P \Rightarrow \xrightarrow{\gamma[\overline{x}^\eta\{S\}]}$.

Let $\mathcal{A} = \mathcal{L} \cup \mathcal{W}$, and let λ range over the classes of actions. We call λ-barbs *all the barbs that belong to the class λ.*

Definition 3.4.3 *For each class of actions λ, let $\cong_\lambda$ be the largest congruence over processes that is* reduction closed *and* preserves λ-barbs.

For example, if $P \cong_{*SND^*} Q$ and $P \downarrow *[\overline{a}^*\{z\}]$ (that is, $P \xrightarrow{*[\overline{a}^*\{z\}]}$), then $Q \Downarrow *[\overline{a}^*\{z\}]$ (that is, $Q \Rightarrow \xrightarrow{*[\overline{a}^*\{z\}]}$). And if $P \cong_{*CAPS^*} Q$ and $P \downarrow *[\overline{a}^*\{\star\}]$ (that is, $P \xrightarrow{*[\overline{a}^*\{R\}]}$ for some R), then $Q \Downarrow *[\overline{a}^*\{\star\}]$ (that is, $Q \Rightarrow \xrightarrow{*[\overline{a}^*\{S\}]}$ for some S).

$$D_n^{*[\overline{x}^*\{y\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.\overline{x}^*\{y\} \qquad D_n^{*[x_z^*]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.(x^*\{z\} \mid z[\mathbf{0}])$$
$$D_n^{*[\overline{x}^\uparrow\{y\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.\overline{x}^\uparrow\{y\} \qquad D_n^{k[x_z^\uparrow]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.(k[x^*\{z\}] \mid z[\mathbf{0}])$$

$$D_n^{*[x^*\{S\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.x^*\{\} \qquad D_n^{*[\overline{x}^*\{\star\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.\overline{x}^*\{\mathbf{0}\}$$
$$D_n^{*[x^\uparrow\{S\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.x^\uparrow\{\} \qquad D_n^{*[\overline{x}^\uparrow\{\star\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.\overline{x}^\uparrow\{\mathbf{0}\}$$
$$D_n^{k[x^\uparrow\{S\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.k[x^\uparrow\{\}] \qquad D_n^{k[\overline{x}^\uparrow\{\star\}]}[-] = - \mid \overline{a}^*\{n\} \mid a^*\{n\}.k[\overline{x}^\uparrow\{\mathbf{0}\}]$$

where a is a fresh name.

Table 3.6: Contexts that transform a natural barb into a λ-barb

Lemma 3.4.4 *Consider the contexts defined in Table 3.5. Then $C_\omega[P] \Downarrow o$ if and only if $P \Downarrow \omega$.*

Proof The *if* direction is a direct consequence of Lemma 3.3.2.

For the *only if* direction, the proof depends on the action ω. The main argument is that the process $C_\omega[P]$ can unleash the seal named o only if P performs the action ω.

The most interesting case is when $\omega = *[\overline{x}^*\{y\}]$. The term

$$C_{*[\overline{x}^*\{y\}]}[P] \;=\; y[\overline{a}^\uparrow()] \mid x^*\{i\}.a^i().o[\,] \mid P$$

must consume the two actions $x^*\{i\}$ and $a^i()$ to activate the seal named o. The first one synchronises either with $\overline{x}^*\{y\}$ (as we desire) or $\overline{x}^*\{S\}$. In this last case the moved seal is provided by P and it cannot perform an interaction over the channel a, that has been chosen fresh for P. In turn, the second action can be consumed only if the seal y provided by the context is moved and renamed into i. This guarantees that $P \Downarrow \overline{x}^*\{y\}$.

The other cases are simpler. □

Lemma 3.4.5 *Consider the contexts defined in Table 3.6. Then $D_n^\omega[P] \Downarrow \omega$ if and only if $P \Downarrow n$.*

Proof Each context can be rewritten as $- \mid \overline{a}^*\{n\} \mid a^*\{n\}.X$ where X is depends on the particular context. As the name a is fresh, the prefix $a^*\{n\}$ is consumed only if the process that fills the hole offers a seal named n at top-level, that is, it emits $\downarrow n$. The continuation X is then responsible for generating the appropriate λ-barb. □

Theorem 3.4.6 *Let P and Q be two processes, and let λ be a class of actions. Then $P \cong Q$ if and only if $P \cong_\lambda Q$.*

Proof Since $\cong$ and $\cong_\lambda$ differ only in the barb which is used, it suffices to show $\downarrow n$ and $\downarrow \lambda$ imply each other.

First we prove that $P \cong_\lambda Q$ implies $P \cong Q$. Let $P \cong Q$ and $P \downarrow \omega$ for $\omega \in \lambda$. We want to conclude that $Q \Downarrow \omega$. As $\cong$ is contextual, it holds $C_\omega[P] \cong C_\omega[Q]$. Lemma 3.4.4 guarantees

that $C_\omega[P] \Downarrow o$. As $\cong$ preserves natural barbs, it must hold $C_\omega[Q] \Downarrow o$. Lemma 3.4.4 allows us to conclude $Q \Downarrow \omega$.

For the other implication, suppose $P \cong_\lambda Q$ and $P \downarrow n$. We want to conclude that $Q \Downarrow \mathrm{n}()$. Let ω be a label in λ. As $\cong_\lambda$ is contextual, it holds $D_n^\omega[P] \cong C_n^\omega[Q]$. Lemma 3.4.5 guarantees that $C_n^\omega[P] \Downarrow \omega$. As $\cong$ preserves λ-barbs, it must hold $C_n^\omega[Q] \Downarrow \omega$. Lemma 3.4.5 allows us to conclude $Q \Downarrow n$. □

To shorten the presentation, in the Definition of λ-barbs we did not include actions related to communication. Unsurprisingly, the Theorem above can easily be expanded to take into account also barbs exposing communication actions.

Theorem 3.4.6 brings out two peculiarities of the LTS proposed. First, the class of action generated by the (Chain) rule do not appear among the classes for which reduction λ-barbed congruence coincides with reduction barbed congruence. As we will discuss in Section 3.4.3, these actions cannot be observed by a context. Second, analogously to what happens with natural barbs, the classes of action $\gamma CAPS^\eta$ do not report the body of the moved seal in their prototypical action. This is typical of our treatment of some of the higher order actions, where weak matching requirements on the transmitted processes are imposed.

3.4.2 Higher-order actions and bisimulation

Lemma 3.4.1 shows that the observation used in the contextual equivalence is insensitive to the particular process Q occurring in the label of the corresponding 'freeze' transition. Thus we expect a LTS based characterisation of this equivalence not to be strict in matching higher-order labels.

To avoid the difficulties we pointed out in the introduction we resort to the intuition underlying the definition of Sangiorgi's *context bisimulation* for HO-π [San92, San96a], and require that the outcomes of two bisimilar processes emitting higher order transitions must be equivalent with respect to every possible interaction with a context.

Processes appear in four kinds of higher-order labels: $\gamma[x^\eta\{\!\!\{ S \}\!\!\}], S^z, R_z$, and $\gamma[\overline{x}^\eta\{\!\!\{ R \}\!\!\}]$. We focus on the contexts with which a process emitting a higher-order label can interact, with the aim of finding out how to apply context bisimulation to our framework.

Receiving a seal body Suppose that $A \vdash P \xrightarrow{\gamma[x^\eta\{\!\!\{ S \}\!\!\}]} P'$, and let Q be a process supposed equivalent to P. The process P' is the outcome of the interaction between P and a context $C[-]$ sending a seal body S over the channel x^η. In the bisimulation game, the process Q must be able to receive an arbitrary seal body over the channel x^η, otherwise a context can easily separate P from Q. At the same time, it should be stressed that the process S appearing in the label is offered by the context, and as such the processes P and Q must both be ready to receive it. Therefore when comparing processes that receive a seal body, it is not restrictive to require syntactically identical processes in higher-order labels.

The case for S^z is analogous to the previous one, as the process S is offered by the environment.

Emitting a seal body Suppose that $A \vdash P \xrightarrow{R_z} P'$. This can be seen as the offer of an interaction with a context that can move and/or duplicate the seal z. We must at least require that there exists a process Q' such that $A \vdash Q \xrightarrow{S_z} Q'$ for some seal body S, otherwise a context can easily separate P from Q. At the same time, imposing S to be syntactically the same as R is overly restrictive. This is a consequence of the fact that the context can not directly investigate the seal body S, but it must move and/or duplicate it before being able to interact with it. Thus, we do not impose directly any condition on S. Rather, we ask that the outcomes obtained after interacting with an arbitrary context that receives (that is, moves and/or copies) the seal are equivalent.

The case for $\overline{x}^{\eta}\{\!| R |\!\}$ is similar: the only advantage is that it is known where the interacting context is going to reactivate the copies of the seal.

To state the equivalence of processes emitting a seal body, we must characterise all the possible outcomes obtained after an interaction between a mobility offer and a context willing to receive the seal body being sent. This role is played by the *receiving contexts*.

Receiving contexts represent all the processes that may result from the migration of a seal. An example is probably helpful here. Let A be a set of names. In this 'universe', consider a process P that emits the label $z[\overline{x}^{\uparrow}\{\!| R |\!\}]$ and becomes P':

$$A \vdash P \xrightarrow{z[\overline{x}^{\uparrow}\{\!| R |\!\}]} P' \ .$$

A context that synchronises with P over this label must have the shape $C'[-] \equiv C[\ - \mid x^z\{\!| \vec{y} |\!\}.V\ ]$, for some $C[-]$ and V. The resulting reduction is

$$C'[P] \ \equiv \ C[\ P \mid x^z\{\!| \vec{y} |\!\}.V\ ] \quad \twoheadrightarrow \quad C[\ (\boldsymbol{\nu}\, \mathrm{fn}(R) \setminus A)(P' \mid y_1[R] \mid \cdots \mid y_n[R] \mid V)\ ] \ .$$

The process V cannot know any name in $\mathrm{fn}(R) \setminus A$, so we rewrite the above reduction as

$$C[\ P \mid x^z\{\!| \vec{y} |\!\}.V\ ] \quad \twoheadrightarrow \quad C[\ (\boldsymbol{\nu}\, \mathrm{fn}(R) \setminus A)(P' \mid y_1[R] \mid \cdots \mid y_n[R]) \mid V\ ] \ . \tag{3.3}$$

Our aim is to define a bisimulation based equivalence, $\approx$, such that contexts like $C'[-]$ cannot distinguish equivalent processes. In the example above, this means that the process $C'[Q]$ must reduce to a process equivalent to the outcome of (3.3). An easy way to achieve this is to require that in the bisimulation game Q emits a label that consumes the action $x^z\{\!| \vec{y} |\!\}.V$, that is

$$A \vdash Q \xrightarrow{z[\overline{x}^{\uparrow}\{\!| S |\!\}]} Q' \quad \text{for some } S, Q',$$

and that the outcome of $C'[P]$ is equivalent to the outcome of $C'[Q]$:

$$C[\ (\boldsymbol{\nu}\, \mathrm{fn}(R) \setminus A)(P' \mid y_1[R] \mid \cdots \mid y_n[R]) \mid V\ ] \approx C[\ (\boldsymbol{\nu}\, \mathrm{fn}(S) \setminus A)(Q' \mid y_1[S] \mid \cdots \mid y_n[S]) \mid V\ ]$$

Some housekeeping now: we factor out the context common to both members of the equation, $C[[-] \mid V]$, and we check that the following equivalence holds:

$$(\boldsymbol{\nu}\, \mathrm{fn}(R) \setminus A)(P' \mid y_1[R] \mid \cdots \mid y_n[R]) \quad \approx \quad (\boldsymbol{\nu}\, \mathrm{fn}(S) \setminus A)(Q' \mid y_1[S] \mid \cdots \mid y_n[S]) \ .$$

This idea can be generalised. Consider two processes (like P and Q) that in an universe A send two (possibly different) seal bodies from a location z toward their parent $\uparrow$. We

consider these two processes as equivalent only if the substitutions of the sent seal bodies for Y and of the outcomes for X in the following pattern

$$(\boldsymbol{\nu}\,\mathrm{fn}(Y) \setminus A)(X \mid y_1[Y] \mid \cdots \mid y_n[Y])$$

yield equivalent processes. The pattern process above is called a *receiving context*, from z toward $\uparrow$ in A, and is noted as $\mathcal{D}^A_{z,\uparrow}[X,Y]$. Its *associated environment* is the set of free variables of the process obtained after instantiating X and Y.

Definition 3.4.7 (Receiving Context) *Let A be a set of variables. Given two processes X and Y such that $\mathrm{fn}(X) \subseteq A$, a* receiving context $\mathcal{D}^A_{\gamma,\eta}[X,Y]$ *and its* associated environment $A_{\mathcal{D}^A_{\gamma,\eta}[X,Y]}$ *are respectively a process and a set of names defined as:*

if $\gamma,\eta = *,*$, or $\gamma,\eta = z,\uparrow$, then

$$\mathcal{D}^A_{\gamma,\eta}[X,Y] \;=\; (\boldsymbol{\nu}\,\mathrm{fn}(Y) \setminus A)(\; X \mid z_1[Y] \mid \cdots \mid z_n[Y] \;)$$

where the names $\vec{z}$ satisfy $\mathrm{fn}(\mathcal{D}^A_{\gamma,\eta}[X,Y]) \subseteq A \cup \vec{z}$.

Its associated environment $A_{\mathcal{D}^A_{\gamma,\eta}[X,Y]}$ is $A \cup \vec{z}$;

if $\gamma,\eta = *,z$, then

$$\mathcal{D}^A_{*,z}[X,Y] \;=\; (\boldsymbol{\nu}\,\mathrm{fn}(Y) \setminus A)(\; X \mid z[(\; z_1[Y] \mid \cdots \mid z_n[Y] \mid U \;)] \;)$$

where the names $\vec{z}$ and the process U satisfy $\mathrm{fn}(\mathcal{D}^A_{*,z}[X,Y]) \subseteq A \cup \vec{z}$.

Its associated environment $A_{\mathcal{D}^A_{*,z}[X,Y]}$ is $A \cup \vec{z}$;

if $\gamma,\eta = *,\uparrow$, then

$$\mathcal{D}^A_{*,\uparrow}[X,Y] \;=\; z[X \mid U] \mid z_1[Y] \mid \cdots \mid z_n[Y]$$

where the names $z,\vec{z}$ and the process U satisfy $\mathrm{fn}(\mathcal{D}^A_{*,\uparrow}[X,Y]) \subseteq A \cup \vec{z} \cup \{z\}$.

Its associated environment $A_{\mathcal{D}^A_{*,\uparrow}[X,Y]}$ is $A \cup \vec{z} \cup \{z\}$.

We write $\mathcal{D}^A[X,Y]$ when we quantify over all γ,η and abbreviate $A_{\mathcal{D}^A_{\gamma,\eta}[X,Y]}$ by $A_{\mathcal{D}}$ when no ambiguity arises.

Seal bisimulation According to our analysis, we are now ready to define a bisimulation based equivalence relation over seal processes.

Definition 3.4.8 (Bisimilarity) *Let bisimilarity, denoted $\approx$, be the largest family of relations indexed by finite sets of names such that each $\approx_A$ is a symmetric relation over $\{P \mid \mathrm{fn}(P) \subseteq A\}$ and for all $P \approx_A Q$ the following conditions hold:*

1. *if $A \vdash P \xrightarrow{\tau} P'$ then there exists a Q' such that $A \vdash Q \Rightarrow Q'$ and $P' \approx_A Q'$;*
2. *if $A \vdash P \xrightarrow{\ell} P'$ and $\ell \in \{\; \gamma[x^\eta(\vec{y})], \gamma[\overline{x}^\eta(\vec{y})], *[\overline{x}^\eta\{\!\!\{ y \}\!\!\}], \gamma[x^\eta\{\!\!\{ S \}\!\!\}], S^z, \gamma[x^\eta_z] \;\}$, then there exists a Q' such that $A \vdash Q \Rightarrow \xrightarrow{\ell} Q'$ and $P' \approx_{A \cup \mathrm{fn}(\ell)} Q'$;*

3. *if* $A \vdash P \xrightarrow{R_z} P'$ *then there exist* Q', S *such that* $A \vdash Q \Rightarrow \xrightarrow{S_z} Q'$ *and for all admissible contexts* $\mathcal{D}^A[-,-]$ *it holds* $\mathcal{D}^A[P', R] \approx_{A_\mathcal{D}} \mathcal{D}^A[Q', S]$;

4. *if* $A \vdash P \xrightarrow{\gamma[\overline{x}^\eta\{R\}]} P'$ *then there exist* Q', S *such that* $A \vdash Q \Rightarrow \xrightarrow{\gamma[\overline{x}^\eta\{S\}]} Q'$ *and for all admissible contexts* $\mathcal{D}^A_{\gamma,\eta}[-,-]$ *it holds* $\mathcal{D}^A_{\gamma,\eta}[P', R] \approx_{A_\mathcal{D}} \mathcal{D}^A_{\gamma,\eta}[Q', S]$;

5. *for all substitutions* σ *such that* $\mathrm{dom}(\sigma) \subseteq A$ *and* $\mathrm{ran}(\sigma) \subseteq A$, $P\sigma \approx_A Q\sigma$;

where a context $\mathcal{D}^A[-,-]$ *is* admissible *if both process substitutions* $\mathcal{D}^A[P', R]$ *and* $\mathcal{D}^A[Q', S]$ *are well-formed (i.e. no name capture arises).*

Intuitively, the first two cases of Definition 3.4.8 handle all first-order labels, as well as labels originating from receive actions: these do not deserve a special treatment because our early semantics implicitly tests all possible interactions. The cases 3. and 4. check processes that offer a seal body. Condition 5. ensures that $\approx$ is preserved by contexts where the hole in under a input prefix.

Standard reasoning, albeit here for an indexed relation, guarantees that

Proposition 3.4.9 *The relation* $\approx$ *exists uniquely and is an equivalence.*

Bisimilarity is preserved by all the constructors of the calculus, albeit working with an indexed relation requires a little care even to state the congruence property, to keep the A-indexing straight. Following [Sew00], we resort to the notion of *indexed congruence*.

Definition 3.4.10 (Indexed congruence) *Consider a family* $\mathcal{R}$ *of relations indexed by finite sets of names such that each* $\mathcal{R}_A$ *is a relation over* $\{P \mid \mathrm{fn}(P) \subseteq A\}$. $\mathcal{R}$ *is an* indexed congruence *if each* $\mathcal{R}_A$ *is an equivalence relation and the following hold:*

$$\frac{P\ \mathcal{R}_{A,\vec{y}}\ Q \quad x, \mathrm{fn}(\eta) \subseteq A}{x^\eta(\vec{y}).P\ \mathcal{R}_A\ x^\eta(\vec{y}).Q} \qquad \frac{P\ \mathcal{R}_A\ Q \quad \alpha \neq x^\eta(y), \mathrm{fn}(\alpha) \subseteq A}{\alpha.P\ \mathcal{R}_A\ \alpha.Q}$$

$$\frac{P\ \mathcal{R}_{A,\vec{y}}\ Q \quad x, \mathrm{fn}(\eta) \subseteq A}{!x^\eta(\vec{y}).P\ \mathcal{R}_A\ !x^\eta(\vec{y}).Q} \qquad \frac{P\ \mathcal{R}_A\ Q \quad \alpha \neq x^\eta(y), \mathrm{fn}(\alpha) \subseteq A}{!\alpha.P\ \mathcal{R}_A\ !\alpha.Q}\ .$$

$$\frac{P\ \mathcal{R}_A\ Q \quad x \in A}{x[P]\ \mathcal{R}_A\ x[Q]} \qquad \frac{P\ \mathcal{R}_A\ Q \quad \mathrm{fn}(R) \subseteq A}{\begin{array}{c} P \mid R\ \mathcal{R}_A\ Q \mid R \\ R \mid P\ \mathcal{R}_A\ R \mid Q \end{array}} \qquad \frac{P\ \mathcal{R}_{A,x}\ Q}{(\boldsymbol{\nu} x)P\ \mathcal{R}_A\ (\boldsymbol{\nu} x)Q}$$

Our aim here is to prove that bisimilarity is an indexed congruence. It is easier to prove separately that bisimilarity is preserved by the different constructs of the language. Before proceeding, we introduce an elementary but useful up-to proof technique.

Lemma 3.4.11 *If* $\mathcal{R}$ *is a bisimulation up to structural congruence, then* $\mathcal{R} \subseteq \approx$.

Proof Let $\mathcal{S}_A = \{(P, Q) \mid (P_1, Q_1) \in \mathcal{R}_A, P_1 \equiv P \text{ and } Q_1 \equiv Q\}$, where A is a set of names. Clearly, for every A, it holds $\mathcal{R}_A \subseteq \mathcal{S}_A$, and $\bigcup_A \mathcal{R}_A \subseteq \bigcup_A \mathcal{S}_A$. Because of Lemma 3.3.7, a diagram chasing argument allows us to show that for every set of names A, the relation $\mathcal{S}_A$ is closed under the conditions to be a bisimulation, as required. □

Lemma 3.4.12 *If $P \approx_A Q$, then $x[P] \approx_A x[Q]$ for all $x \in A$.*

Proof Define $\mathcal{S}_A = \{(x[P], x[Q]) \mid P \approx_A Q, x \in A\} \cup \approx_A$ and let $\mathcal{S} = \bigcup_A \mathcal{S}_A$. We show that $\mathcal{S}$ is a bisimulation. Consider $(R, S) \in \mathcal{S}$. Then there exists A such that either $R \approx_A S$ holds, or $R = x[P]$ and $S = x[Q]$ for $x \in A$ and $P \approx_A Q$. In the former case the result follows straightforwardly. We next consider the possible transitions in the latter case. A simple inspection of the rules of LTS shows that if $x[P]$ emits a label, then one of the following cases holds:

- $A \vdash x[P] \xrightarrow{x[a]} x[P']$ for some P', where $a \in \{\tau, \overline{y}^{\uparrow}(\vec{z}), y^{\uparrow}(\vec{z}), y^{\uparrow}\{\!|Q|\!\}\}$ and $A \vdash P \xrightarrow{*[a]} P'$. We have $A \vdash Q \Rightarrow \xrightarrow{*[a]} Q'$ and $P' \approx_{A \cup \mathrm{fn}(a)} Q'$ because of the definition of bisimulation. Since $x \in A$, we have $A \vdash x[Q] \Rightarrow \xrightarrow{x[a]} x[Q']$, and $x[P']\ \mathcal{S}_{A \cup \mathrm{fn}(a)}\ x[Q']$ follows from the definition of $\mathcal{S}$.
- $A \vdash x[P] \xrightarrow{x[\overline{y}^{\uparrow}\{\!|R|\!\}]} x[P']$ for some P', where $A \vdash P \xrightarrow{*[\overline{y}^{\uparrow}\{\!|R|\!\}]} P'$. The definition of bisimulation ensures that (i) $A \vdash Q \Rightarrow \xrightarrow{*[\overline{y}^{\uparrow}\{\!|S|\!\}]} Q'$ for some Q', S; and (ii) $\mathcal{D}^A_{*,\uparrow}[P', R] \approx_{A_{\mathcal{D}}} \mathcal{D}^A_{*,\uparrow}[Q', S]$ for every possible such a pair of receiving contexts. From (i) we deduce that $A \vdash x[Q] \Rightarrow \xrightarrow{x[\overline{y}^{\uparrow}\{\!|S|\!\}]} x[Q']$. Now use (ii) and notice that every process that is a receiving context $\mathcal{D}^A_{x,\uparrow}[x[O], -]$ is also a receiving context $\mathcal{D}^A_{*,\uparrow}[O, -]$. Therefore (ii) implies that $\mathcal{D}^A_{x,\uparrow}[x[P'], R] \approx_{A_{\mathcal{D}}} \mathcal{D}^A_{x,\uparrow}[x[Q'], S]$, which, by definition of $\mathcal{S}$ implies
$$\mathcal{D}^A_{x,\uparrow}[x[P'], R]\ \mathcal{S}_{A_{\mathcal{D}}}\ \mathcal{D}^A_{x,\uparrow}[x[Q'], S]\ .$$

The last property we have to check for is closure under substitutions. Let σ be a substitution defined on A. Let $(R, S) \in \mathcal{S}$. Then there exists A such that either $R \approx_A S$ holds, or $R = x[P]$ and $S = x[Q]$ for $x \in A$ and $P \approx_A Q$. In the first case $P\sigma \approx_{A\sigma} Q\sigma$ holds because $\approx$ is closed under arbitrary substitution. Then $P\sigma\ \mathcal{S}_{A\sigma}\ Q\sigma$ follows from the construction of $\mathcal{S}$. In the latter, observe that $x\sigma \in A\sigma$ because $x \in A$, and $R\sigma \approx_{A\sigma} S\sigma$ holds because $\approx$ is closed under arbitrary substitution. The definition of substitution assures that $x\sigma[R\sigma] = (x[R])\sigma$ and $x\sigma[S\sigma] = (x[S])\sigma$. Then $(x[R])\sigma\ \mathcal{S}_{A\sigma}\ (x[S])\sigma$ follows because of the construction of $\mathcal{S}$. □

Lemma 3.4.13 *If $P \approx_A Q$, then $P \mid R \approx_A Q \mid R$ and $(\boldsymbol{\nu} x)P \approx_{A \setminus x} (\boldsymbol{\nu} x)Q$, for all $x \in A$, and for all R such that $\mathrm{fn}(R) \subseteq A$.*

Proof Define $\mathcal{S}_A = \{(\,(\boldsymbol{\nu}\vec{u})(P \mid R)\,, (\boldsymbol{\nu}\vec{u})(Q \mid R)\,) \mid P \approx_{A,\vec{u}} Q,\ (\mathrm{fn}(R) \setminus \vec{u}) \subseteq A\}$, and let $\mathcal{S} = \bigcup_A \mathcal{S}_A$. We show that $\mathcal{S}$ is a bisimulation up to structural congruence. Suppose $P \approx_A Q$. As a useful shorthand, we write $\bar{P}$ for $(\boldsymbol{\nu}\vec{u})(P \mid R)$, and $\bar{Q}$ for $(\boldsymbol{\nu}\vec{u})(Q \mid R)$.

We perform a case analysis on the transitions performed by $\bar{P}$. We detail all the most difficult and interesting situations (namely higher-order synchronisation and scope extrusion): the others follow accordingly.

- Suppose $A \vdash \bar{P} \xrightarrow{\gamma[x^{\eta}_z]} O_1$ because $A \cdot \vec{u} \vdash P \xrightarrow{\gamma[x^{\eta}\{\!|F|\!\}]} P'$ and $A \cdot \vec{u} \vdash R \xrightarrow{F_z} R'$. From the definition of bisimulation it follows that $A \cdot \vec{u} \vdash Q \Rightarrow \xrightarrow{\gamma[x^{\eta}\{\!|F|\!\}]} Q'$ and
$$P' \approx_{A \cup \vec{u} \cup \mathrm{fn}(F)} Q'\ . \tag{3.4}$$

Remark that $\mathrm{fn}(R') \subseteq A \cup \vec{u} \cup \mathrm{fn}(F_z) = A \cup \vec{u} \cup \mathrm{fn}(F)$ since $z \in A$. Thus, $A \vdash \bar{Q} \Rightarrow \xrightarrow{\gamma[x_z^\eta]} O_2$. The outcomes are respectively of the form $O_1 \equiv (\boldsymbol{\nu}\vec{u})(\boldsymbol{\nu}\,\mathrm{fn}(F) \setminus (A \cup \vec{u}))(P' \mid R')$ and $O_2 \equiv (\boldsymbol{\nu}\vec{u})(\boldsymbol{\nu}\,\mathrm{fn}(F) \setminus (A \cup \vec{u}))(Q' \mid R')$. Thus, from (3.4) we can conclude $O_1 \; \mathcal{S}_A \; O_2$ because of the construction of $\mathcal{S}$.

- Suppose $A \vdash \bar{P} \xrightarrow{*[\overline{x}^\eta\{F\}]} O_1$ because $A \cdot \vec{u} \vdash P \xrightarrow{F_z} P'$ and $A \cdot \vec{u} \vdash R \xrightarrow{*[\overline{x}^\eta\{z\}]} R'$. From the definition of bisimulation it follows that $A \vdash Q \Rightarrow \xrightarrow{G_z} Q'$ and

$$\mathcal{D}^{A,\vec{u}}[P', F] \approx_{A_\mathcal{D}} \mathcal{D}^{A,\vec{u}}[Q', G] \; . \tag{3.5}$$

Thus, $A \vdash \bar{Q} \Rightarrow \xrightarrow{*[\overline{x}^\eta\{G\}]} O_2$. We must prove that for all admissible contexts $\mathcal{E}^A_{*,\eta}[-,-]$ it holds $\mathcal{E}^A_{*,\eta}[O_1, F] \; \mathcal{S}_{A_\mathcal{E}} \; \mathcal{E}^A_{*,\eta}[O_2, G]$. We have to consider different cases according to whether the names in $\vec{u}$ are extruded or not.

Suppose $\mathrm{fn}(F) \subseteq A$ and $\mathrm{fn}(G) \subseteq A$. In this case, the outcomes are respectively of the form $O_1 \equiv (\boldsymbol{\nu}\vec{u})(P' \mid R')$ and $O_2 \equiv (\boldsymbol{\nu}\vec{u})(Q' \mid R')$ (no extrusion occurs). The structure of the context $\mathcal{E}$ depends on the localisation η of the mobility offer:

- If $\eta = *$, then (3.5) states that for all names $\vec{z}$ it holds:

$$(\boldsymbol{\nu}\,\mathrm{fn}(F) \setminus (A, \vec{u}))(P' \mid z_1[F] \mid \cdots \mid z_n[F]) \\ \approx_{A_\mathcal{D}} \; (\boldsymbol{\nu}\,\mathrm{fn}(G) \setminus (A, \vec{u}))(Q' \mid z_1[G] \mid \cdots \mid z_n[G])$$

  Since $\mathrm{fn}(F),\ \mathrm{fn}(G) \subseteq A$, then the leading restrictions are empty. Combining this with the observation that $\approx_{A_\mathcal{D}} \subseteq \mathcal{S}_{A_\mathcal{D}}$, we obtain:

$$P' \mid z_1[F] \mid \cdots \mid z_n[F] \; \mathcal{S}_{A_\mathcal{D}} \; Q' \mid z_1[G] \mid \cdots \mid z_n[G] \tag{3.6}$$

  Remark that $A_\mathcal{D} = A \cup \vec{z} \cup \vec{u}$ and recall that $A \cap \vec{u} = \varnothing$. Since $\mathcal{S}$ is closed under parallel composition and restriction we can write:

$$(\boldsymbol{\nu}\vec{u})(P' \mid R' \mid z_1[F] \mid \cdots \mid z_n[F]) \; \mathcal{S}_{A,\vec{z}} \; (\boldsymbol{\nu}\vec{u})(Q' \mid R' \mid z_1[G] \mid \cdots \mid z_n[G])$$

  Since (3.5) holds for all admissible receiving contexts, in particular it holds for those receiving contexts of form (3.6) where $\vec{z} \cap \vec{u} = \emptyset$. For these particular receiving contexts we can rearrange both sides of the equation (3.6) by structural congruence, obtaining:

$$(\boldsymbol{\nu}\vec{u})(P' \mid R') \mid z_1[F] \mid \cdots \mid z_n[F] \; \mathcal{S}_{A,\vec{z}} \; (\boldsymbol{\nu}\vec{u})(Q' \mid R') \mid z_1[G] \mid \cdots \mid z_n[G]$$

  that is $\mathcal{E}^A_{*,*}[O_1, F] \; \mathcal{S}_{A,\vec{z}} \; \mathcal{E}^A_{*,*}[O_2, G]$. The result $\mathcal{E}^{A,\vec{u}}_{*,*}[O_1, F] \; \mathcal{S}_{A_\mathcal{E}} \; \mathcal{E}^{A,\vec{u}}_{*,*}[O_2, G]$ follows because the receiving context is generated by $\gamma = *, \eta = *$, and so $A_\mathcal{E} = A, \vec{z}$.

- The case $\eta = z$ is analogous to the previous one.

– If $\eta = \uparrow$ then from (3.5), case $\gamma, \eta = *, \uparrow$ we can deduce that for all U' it holds

$$z[P' \mid R' \mid U'] \mid z_1[F] \mid \cdots \mid z_n[F] \;\approx_{A \cup \vec{z} \cup \vec{u}}\; z[Q' \mid R' \mid U'] \mid z_1[G] \mid \cdots \mid z_n[G]$$

(it suffices to take $U = U' \mid R'$). The construction of $\mathcal{S}$ then guarantees that

$$(\boldsymbol{\nu}\vec{u})z[P' \mid R' \mid U'] \mid z_1[F] \mid \cdots \mid z_n[F] \\ \approx_{A \cup \vec{z}}\; (\boldsymbol{\nu}\vec{u})z[Q' \mid R' \mid U'] \mid z_1[G] \mid \cdots \mid z_n[G]$$

and up to structural congruence we obtain

$$z[(\boldsymbol{\nu}\vec{u})(P' \mid R' \mid U')] \mid z_1[F] \mid \cdots \mid z_n[F] \\ \approx_{A \cup \vec{z}}\; z[(\boldsymbol{\nu}\vec{u})(Q' \mid R' \mid U')] \mid z_1[G] \mid \cdots \mid z_n[G]\ .$$

This can be written using receiving contexts as $\mathcal{E}^A_{*,\uparrow}[O_1, F]\ \mathcal{S}_{A_\mathcal{E}}\ \mathcal{E}^A_{*,\uparrow}[O_2, G]$, as required.

Suppose now that $\mathrm{fn}(F) \not\subseteq A$ and $\mathrm{fn}(G) \subseteq A$. This means that in $\bar{P}$ some of the names in $\vec{u}$ are extruded by the mobility offer. condition As before, we perform a case analysis on the localisation η. We detail only the case $\eta = *$: the others are similar. If $\eta = *$, then the outcomes are respectively of the form $O_1 \equiv (\boldsymbol{\nu}\vec{u} \setminus \mathrm{fn}(F))(P' \mid R')$ and $O_2 \equiv (\boldsymbol{\nu}\vec{u})(Q' \mid R')$. We spell out (3.5) for $\gamma, \eta = *, *$. It holds $P' \mid z_1[F] \mid \cdots \mid z_n[F] \approx_{A \cup \vec{u} \cup \vec{z}} Q' \mid z_1[G] \mid \cdots \mid z_n[G]$. This implies that for all $\vec{z}$ it holds

$$(\boldsymbol{\nu}\vec{u})(P' \mid R' \mid z_1[F] \mid \cdots \mid z_n[F])\ \mathcal{S}_{A \cup \vec{z}}\ (\boldsymbol{\nu}\vec{u})(Q' \mid R' \mid z_1[G] \mid \cdots \mid z_n[G])$$

Now notice that since $A \vdash \bar{P} = (\boldsymbol{\nu}\vec{u})(P \mid R) \xrightarrow{*[\overline{x}^\eta\{F\}]}$ then F occurs in $\bar{P}$ and therefore $\mathrm{fn}(F) \subseteq A \cup \vec{u}$. From the last formula we obtain $\mathrm{fn}(F) \setminus A \subseteq \vec{u}$ from which it is possible to deduce that $\vec{u} = (\mathrm{fn}(F) \setminus A) \cup (\vec{u} \setminus \mathrm{fn}(F))$. Therefore we have:

$$(\boldsymbol{\nu}\,\mathrm{fn}(F) \setminus A)(\boldsymbol{\nu}\vec{u} \setminus \mathrm{fn}(F))(P' \mid R' \mid z_1[F] \mid \cdots \mid z_n[F]\ ) \\ \mathcal{S}_{A \cup \vec{z}}\ (\boldsymbol{\nu}\vec{u})(Q' \mid R' \mid z_1[G] \mid \cdots \mid z_n[G]\ )$$

Since the equation above holds for all $\vec{z}$, then in particular it holds for those $\vec{z}$ such that $\vec{u} \cap \vec{z} = \varnothing$. In this case then we can extrude the z_i's from the scope of the inner restriction in the left hand-side of the above equation above. Furthermore recall that $A \cap \vec{u} = \varnothing$ and $\mathrm{fn}(G) \subseteq A$, which implies $\mathrm{fn}(G) \cap \vec{u} = \varnothing$. Then we can extrude these seals in the right hand-side,too. Up to structural congruence we have:

$$(\boldsymbol{\nu}\,\mathrm{fn}(F) \setminus A)(\ ((\boldsymbol{\nu}\vec{u} \setminus \mathrm{fn}(F))(P' \mid R'))) \mid z_1[F] \mid \cdots \mid z_n[F]\ ) \\ \mathcal{S}_{A \cup \vec{z}}\quad (\boldsymbol{\nu}\vec{u})(Q' \mid R') \mid z_1[G] \mid \cdots \mid z_n[G]$$

that is $\mathcal{E}^A_{*,*}[O_1, F]\ \mathcal{S}_{A_\mathcal{E}}\ \mathcal{E}^A_{*,*}[O_2, G]$.

Finally, the remaining two cases, that is $\mathrm{fn}(F) \subseteq A$, $\mathrm{fn}(G) \not\subseteq A$ and $\mathrm{fn}(F) \not\subseteq A$, $\mathrm{fn}(G) \not\subseteq A$, are similar to the previous one.

- Suppose $A \vdash \bar{P} \xrightarrow{\tau} O_1$, where $A \cdot \vec{u} \vdash P \xrightarrow{F_z} P'$ and $A \cdot \vec{u} \vdash R \xrightarrow{F^z} R_1$.
  By Lemma 3.3.2 either $R_1 \equiv (\boldsymbol{\nu}\vec{w})(R_3 \mid z_1[F] \mid \cdots \mid z_n[F])$ or $R_1 \equiv (\boldsymbol{\nu}\vec{w})(R_3 \mid z[R_4 \mid z_1[F] \mid \cdots \mid z_n[F]])$ hold. From the definition of bisimulation it follows that $A \cdot \vec{u} \vdash Q \Rightarrow \xrightarrow{G_z} Q'$, and for all $\mathcal{D}^{A,\vec{u}}[-,-]$ admissible it holds

$$\mathcal{D}^{A,\vec{u}}[P', F] \approx_{A_{\mathcal{D}}} \mathcal{D}^{A,\vec{u}}[Q', G]. \tag{3.7}$$

  The early nature of the LTS allows process R to receive G as well: $A \cdot \vec{u} \vdash R \xrightarrow{G^z} R_2$, with R_2 either in the form $R_2 \equiv (\boldsymbol{\nu}\vec{w})(R_3 \mid z_1[G] \mid \cdots \mid z_n[G])$, or $R_2 \equiv (\boldsymbol{\nu}\vec{w})(R_3 \mid z[R_4 \mid z_1[G] \mid \cdots \mid z_n[G]])$. Thus, $A \vdash \bar{Q} \Rightarrow \xrightarrow{\tau} O_2$.

  Consider the first case.

  Before going on, we point out that as $A \cdot \vec{u} \vdash R \xrightarrow{F^z} \equiv (\boldsymbol{\nu}\vec{w})(R_3 \mid z_1[F] \mid \cdots \mid z_n[F])$, the condition $\vec{z} \subseteq A \cup \vec{u} \cup \vec{w}$ must hold. Also, $\mathrm{fn}(R_3) \subseteq A \cup \vec{u} \cup \vec{w}$. To see why, remember that R_3 is formed by the parallel composition of two residuals: the continuation of the receiving action, and the continuation of the send action. The free variables of the first process must be contained in $A \cup \vec{u} \cup \vec{z} \cup \{x\}$ where x is the channel on which the move takes place; those of the second process are instead contained in $A \cup \vec{u} \cup \{z\} \cup \{x\}$. Thus $\mathrm{fn}(R_3) \subseteq A \cup \vec{u} \cup \vec{z} \cup \{z\} \cup \{x\}$. But $\{z\} \cup \{x\} \subseteq A \cup \vec{u}$ as they obviously occur free in the sending process (cf. rule (SND)). So, $\vec{z} \subseteq A \cup \vec{w}$ and we conclude that $\mathrm{fn}(R_3) \subseteq A \cup \vec{u} \cup \vec{w}$.

  The outcomes are respectively of the form

$$\begin{aligned} O_1 &\equiv (\boldsymbol{\nu}\vec{u})(\boldsymbol{\nu}\,\mathrm{fn}(F) \setminus (A \cup \vec{u}))(P' \mid (\boldsymbol{\nu}\vec{w})(R_3 \mid z_1[F] \mid \cdots \mid z_n[F])) \\ O_2 &\equiv (\boldsymbol{\nu}\vec{u})(\boldsymbol{\nu}\,\mathrm{fn}(G) \setminus (A \cup \vec{u}))(Q' \mid (\boldsymbol{\nu}\vec{w})(R_3 \mid z_1[G] \mid \cdots \mid z_n[G])) \,. \end{aligned}$$

  Since $\mathrm{fn}(R_3) \subseteq A \cup \vec{u} \cup \vec{w}$ we have $\mathrm{fn}(R_3) \cap (\mathrm{fn}(F) \setminus (A \cup \vec{u})) = \emptyset$, and similarly $\mathrm{fn}(R_3) \cap (\mathrm{fn}(G) \setminus (A \cup \vec{u})) = \emptyset$. We can then extrude R_3 from the scope of the inner restriction as follows:

$$\begin{aligned} O_1 &\equiv (\boldsymbol{\nu}\vec{u})(\boldsymbol{\nu}\vec{w})(R_3 \mid (\boldsymbol{\nu}\,\mathrm{fn}(F) \setminus (A \cup \vec{u}))(P' \mid z_1[F] \mid \cdots \mid z_n[F])) \\ O_2 &\equiv (\boldsymbol{\nu}\vec{u})(\boldsymbol{\nu}\vec{w})(R_3 \mid (\boldsymbol{\nu}\,\mathrm{fn}(G) \setminus (A \cup \vec{u}))(Q' \mid z_1[G] \mid \cdots \mid z_n[G])) \,. \end{aligned}$$

  The case $\gamma, \eta = *, *$ of (3.7) implies that

$$\begin{aligned} &(\boldsymbol{\nu}\,\mathrm{fn}(F) \setminus (A \cup \vec{u}))(P' \mid z_1[F] \mid \cdots z_n[F]) \\ &\qquad\qquad \approx_{A \cup \vec{z} \cup \vec{u}} \ (\boldsymbol{\nu}\,\mathrm{fn}(G) \setminus (A \cup \vec{u}))(Q' \mid z_1[G] \mid \cdots z_n[G]) \end{aligned} \tag{3.8}$$

  The construction of $\mathcal{S}$ allows us to derive $O_1 \ \mathcal{S}_{(A \cup \vec{u} \cup \vec{z}) \setminus (\vec{u} \cup \vec{w})} \ O_2$ from 3.8. As $\vec{z} \subseteq A \cup \vec{u} \cup \vec{w}$, $(A \cup \vec{u} \cup \vec{z}) \setminus (\vec{u} \cup \vec{w}) = A$, and we conclude $O_1 \ \mathcal{S}_A \ O_2$, as required.

  The latter case follows accordingly. □

Theorem 3.4.14 *Bisimilarity is an indexed congruence.*

Proof It is straightforward to prove that $\approx$ is preserved by prefix and replication. The other cases are covered by Lemma 3.4.12 and Lemma 3.4.13.[4] □

[4]The proof of the symmetric case of congruence with respect to parallel composition is analogous to the proof of Lemma 3.4.13.

Lemma 3.4.15 (Injective substitution — bisimulation) *If $P \approx_A Q$ and $f : A \to B$ is injective then $fP \approx_B fQ$.*

Proof We check

$$\mathcal{R}_B = \{\ (fP, fQ) \mid f : A \to_{\text{inj}} B \text{ and } P \approx_A Q\ \}$$

is a bisimulation.

Suppose $B \vdash fP \xrightarrow{\ell'} P'_1$. We perform a case analysis on the label ℓ'.

Case $\ell = \tau$. By Corollary 3.3.6, there exists P' such that $A \vdash P \xrightarrow{\tau} P'$ and $P'_1 = fP'$. By bisimulation, there exists Q' such that $A \vdash Q \Rightarrow Q'$ and $P' \approx_A Q'$. By Corollary 3.3.6, $B \vdash fQ \Rightarrow fQ'$. Finally $fP' \mathcal{R}_B fQ'$.

Case $\ell \in \{\ \gamma[x^\eta(\vec{y})], \gamma[\overline{x}^\eta(\vec{y})], *[\overline{x}^\eta\{\!| y |\!\}], \gamma[x^\eta\{\!| S |\!\}], S^z, \gamma[x^\eta_z]\ \}$. By Lemma 3.3.5, there exist $\ell, P', H, I, g : I \to_{\text{bij}} B', h : H \to_{\text{inj}} (B \setminus \text{ran}(f))$: such that $B' \cap B = \emptyset$, $H \cup I = \text{fn}(\ell) \setminus A$, $H \cap I = \emptyset$, and

$$A \vdash P \xrightarrow{\ell} P' \qquad P'_1 = (f+g+h)P' \qquad \ell' = (f+g+h)\ell\ .$$

By bisimulation, there exists Q' such that $A \vdash Q \Rightarrow \xrightarrow{l} Q'$ and $P' \approx_{A \cup \text{fn}(\ell)} Q'$.

Then, for f, g, h as above, we have $B \vdash fQ \Rightarrow \xrightarrow{(f+g+h)\ell} (f+g+h)Q'$. Finally, $(f+g+h)P'\ \mathcal{R}_B\ (f+g+h)Q'$ follows from the construction of $\mathcal{R}$.

Case $\ell = R_z$. Then, $B \vdash fP \xrightarrow{R_z} P'_1$. By Lemma 3.3.5, there exist ℓ, P' and $I, g : I \to_{\text{bij}} B'$ with $I \cap A = \emptyset$ and $B' \cap B = \emptyset$, such that

$$A \vdash P \xrightarrow{\ell} P' \qquad (f+g)\ell = R_z \qquad (f+g)P' = P'_1\ .$$

Observe that ℓ must be of the form $R'_{z'}$ for some R', z' such that $(f+g)R' = R$ and $fz' = z$. Then by bisimulation there exists a Q' such that $A \vdash Q \xrightarrow{S'_{z'}} Q'$ and for all admissible contexts $\mathcal{D}^A[-,-]$ it holds $\mathcal{D}^A[P', R'] \approx_{A_\mathcal{D}} \mathcal{D}^A[Q', S']$. By Lemma 3.3.3 we have $B \vdash fQ \xrightarrow{S_z} Q'_1$, where $S = (f+g)S'$ and $Q'_1 = (f+g)Q'$. To conclude that for all admissible contexts $\mathcal{E}^B[-,-]$ it holds $\mathcal{E}^B[P'_1, R]\ \mathcal{R}_{A_\mathcal{E}}\ \mathcal{E}^B[Q'_1, S]$, observe that every context $\mathcal{E}^B[-,-]$ can be written as $(f+g)\mathcal{D}^A[-,-]$ for a receiving context $\mathcal{D}^A[-,-]$ because $(f+g)$ is injective and because $g(\text{fn}(R') \setminus A) = \text{fn}((f+g)R) \setminus B$.

Case $\ell = \gamma[\overline{x}^\eta\{\!| R |\!\}]$. Similar to $\ell = R_z$.

It remains to prove that $\mathcal{R}$ is closed under substitution. For that, suppose $fP\ \mathcal{R}_B\ fQ$ and let $\sigma : B \to C$ be a substitution. We want to show that $(fP)\sigma\ \mathcal{R}_{B\sigma}\ (fQ)\sigma$. By definition of $\mathcal{R}$, $P\ \mathcal{R}_A\ Q$ for some A, $f : A \to_{\text{inj}} B$. The composition of σ and f is a substitution $\sigma f : A \to C$. As $\approx_A$ is closed under substitution, $P(\sigma f)\ \mathcal{R}_{A(\sigma f)}\ Q(\sigma f)$, that is, $(fP)\sigma\ \mathcal{R}_{(fA)\sigma}\ (fQ)\sigma$. Let $i : (fA)\sigma \to B\sigma$ be the injection of the set $(fA)\sigma$ into $B\sigma$ (the injection exists because $fA \subseteq B$). Then

$$(fP)\sigma = i((fP)\sigma)\ \mathcal{R}_{B\sigma}\ i((fQ)\sigma) = (fQ)\sigma$$

as required. □

The soundness of our bisimulation based proof method is an easy consequence of the two Lemmas above.

Theorem 3.4.16 (Soundness) *Bisimilarity is sound with respect to barbed congruence: if $P \approx_A Q$ for some A, then $P \cong Q$.*

Proof Suppose $P \approx_A Q$ for some A. By Definition 3.4.8.1 bisimilarity $\approx$ is reduction closed. bisimilarity is preserved by arbitrary contexts because it is an indexed congruence (Lemma 3.4.14), and because it is preserved by injective renaming (Lemma 3.4.15). It remains to prove that if $P \downarrow n$ then $Q \Downarrow n$. For this, note that the first part of condition 3 in Definition 3.4.8 ensures that whenever $A \vdash P \xrightarrow{F_n}$ for some process F, $A \vdash Q \Rightarrow \xrightarrow{G_n}$ for some process G. Therefore, the thesis follows by Lemma 3.4.1. □

3.4.3 On completeness

Bisimilarity is not complete with respect to barbed congruence, for several reasons. First of all, there is a technical cause: bisimilarity is a delay bisimilarity, that is the weak transitions $\Rightarrow \xrightarrow{\ell}$ do not allow τ moves after a visible action. We keep the delay formulation of the bisimulation because it is easier to apply in practise. Also, the delay formulation captures the intuition that a process that performs an higher-order action needs the contribution of a receiving context before continuing.

Deep reasons lurk in the design of the calculus, both in the communication and in the mobility subsystems. The communication subsystem of the Seal Calculus is an extension of the π-calculus, and, with an LTS analogous to ours, in the π-calculus the matching operator is necessary to obtain completeness: the same operator may be required in Seal.

But the mobility subsystem raises the most interesting point. A direct consequence of the Seal's model of computation, is that a seal can not detect if the environment moves it or not. Exploiting this feature, we construct the two processes:

$$P \;=\; (\boldsymbol{\nu} x)(\overline{x}^*\{\!|\, y \,|\!\}\mid x^*\{\!|\, y \,|\!\}) \quad \text{and} \quad Q \;=\; \mathbf{0}\ .$$

Let $A \supseteq \mathrm{fn}(P)$. Then for all processes S, it holds $A \vdash P \xrightarrow{S^y}$, while Q does not emit anything. Thus $P \not\approx Q$. At the same time no context can separate P from Q: even $C[-] \;=\; - \mid y[S]$, that is offering a seal to be moved, does not help, because the context cannot determine if the seal y has been moved or not. In technical terms, this example shows that no context can reliably test for a S^z action. As a consequence, reduction barbed congruence is not insensible to replacing natural barbs with barbs inherited from the S^z action.

This observation might suggest that for the same reasons it should be difficult for a context to distinguish the label $\gamma[\overline{x}^\eta\{\!|\, y \,|\!\}]$ from $\gamma[\overline{x}^\eta\{\!|\, S \,|\!\}]$, and $*[x^\eta\{\!|\, S \,|\!\}]$ from $*[x_z^\eta]$. It is not the case: a careful use of fresh names leads to the definition of contexts that differentiate these actions, as seen in Section 3.4.1.

3.5 Algebraic theory

The following equation, named *the perfect firewall equation* holds in S-Seal without any supplementary condition.

Lemma 3.5.1 (Perfect firewall) *For every process P, it holds:*

$$(\boldsymbol{\nu} n)n[P] \;\cong\; \mathbf{0}\ .$$

Proof Let $\mathcal{S}_A = \{\, ((\boldsymbol{\nu} n)n[P], \mathbf{0}) \mid \mathrm{fn}(P) \subseteq A \,\}^{=}$, where $\mathcal{R}^{=}$ denotes the symmetric closure of $\mathcal{R}$. Let $\mathcal{S} = \cup_A \mathcal{S}_A$. The process $(\boldsymbol{\nu} n)n[P]$ can not emit any label: if $n[P]$ performs a visible action, then it is of the form $n[a]$ or P'_n, where $P \Rightarrow P'$. In both cases n belongs to the free names of the label, and it can not be observed under $(\boldsymbol{\nu} n)$. So $\mathcal{S}$ is a bisimulation. The Lemma follows because $\approx \subseteq \cong$. □

This equation encodes the fact that it is possible to prevent a seal to have any interaction with its environment simply by restricting its name. This justifies processes like

$$\begin{array}{lcl} (\mathsf{halt}\ x.P) & \stackrel{def}{=} & (\boldsymbol{\nu} n)(x\ \mathsf{be}\ n.(P \mid x^*(z).n\ \mathsf{be}\ x)) \\ (\mathsf{restart}\ x) & \stackrel{def}{=} & \overline{x}^*(x).P \end{array}$$

where be is the renaming operator defined as

$$(n\ \mathsf{be}\ m).P \;\stackrel{def}{=}\; (\boldsymbol{\nu} x)(\overline{x}^*\{n\} \mid x^*\{m\}.P)\ .$$

These operators do not modify the process running within the seal, rather they affect its communication capabilities. In fact,

$$\mathsf{halt}\ x.P \mid x[Q] \;\rightarrowtriangle\; P \mid (\boldsymbol{\nu} n)(n[Q] \mid x^*().n\ \mathsf{be}\ x)$$

and the previously active seal x is prevented from interacting with the environment because it has been renamed into n, a secret name. The same seal can be freed by executing $\mathsf{restart}\ x$:

$$(\boldsymbol{\nu} n)(n[Q] \mid x^*().n\ \mathsf{be}\ x) \mid \mathsf{restart}\ x \;\rightarrowtriangle\; x[Q]\ .$$

Thus, the perfect firewall equation is an useful aid to the programmer.

In the proposition below, we enumerate a collection of algebraic laws that describe the use of restriction to avoid interference in mobility and communication. The same equations hold if seals are duplicated, and vector of names transmitted.

Proposition 3.5.2 *For all processes P, Q, and R, the following equivalences hold:*

1. $(\boldsymbol{\nu} m, x)(m[P] \mid \overline{x}^*\{m\}.Q \mid x^*\{n\}.R) \;\approx\; (\boldsymbol{\nu} m, x)(n[P] \mid Q \mid R)$
2. $(\boldsymbol{\nu} m, x, o)(m[P] \mid \overline{x}^o\{m\}.Q \mid o[x^\uparrow\{n\}.R]) \;\approx\; (\boldsymbol{\nu} m, x, o)(Q \mid o[n[P] \mid R])$
3. $(\boldsymbol{\nu} x, o)(x^o\{n\}.Q \mid o[m[P] \mid \overline{x}^\uparrow\{m\}.R]) \;\approx\; (\boldsymbol{\nu} x, o)(n[P] \mid Q \mid o[R])$
4. $(\boldsymbol{\nu} x)(\overline{x}^*(v).Q \mid x^*(u).R) \;\approx\; (\boldsymbol{\nu} x)(Q \mid R[v/u])$

5. $(\boldsymbol{\nu} x, o)(\overline{x}^o(v).Q \mid o[x^\uparrow(u).R]) \;\approx\; (\boldsymbol{\nu} x, o)(Q \mid o[R[v/u]])$

6. $(\boldsymbol{\nu} x, o)(x^o(u).R \mid o[\overline{x}^\uparrow(v).R]) \;\approx\; (\boldsymbol{\nu} x, o)(Q[v/u] \mid o[Q])$

Proof The proofs of the above laws are by exhibiting the appropriate bisimulation. In all cases the bisimulation has a similar form:

$$\mathcal{S} = \{(lhs, rhs), (rhs, lhs)\} \cup \mathcal{I}$$

where *lhs* and *rhs* denote respectively the left hand side and the right hand side of the equation, and $\mathcal{I}$ is the identical relation over processes. □

Limits of the proof method In the other dialects of the Seal Calculus, the perfect firewall equation does not hold. But in all dialects, a safe way to isolate a seal consists in enclosing it in a secret sandbox.

Lemma 3.5.3 (Secret sandbox) *For every process P and name n, it holds:*

$$(\boldsymbol{\nu} a)a[n[P]] \;\cong\; \mathbf{0} \ .$$

In S-Seal, it is easy to prove the perfect sandbox equation using our proof method (actually, it is a trivial consequence of the perfect firewall).

We conjecture that a much stronger equation holds.

Lemma 3.5.4 (Sandbox) *For every process P and name n, it holds:*

$$a[n[P]] \;\cong\; a[\mathbf{0}] \ .$$

This equation captures the key idea that all interactions a seal can perform are under control of its enclosing seal. Unfortunately, it is far from easy to guess the smallest bisimulation relating $a[n[P]]$ and $a[\mathbf{0}]$. In fact, the seal a can be duplicated by an arbitrary context in almost arbitrary locations (receiving contexts can have very complex shapes, especially after few iterations), and the bisimulation candidate quickly becomes intractable.

The aim of this chapter was to present and to develop a foundation to the Seal Calculus. We believe that the basis of the Seal Calculus have been laid, and we will not investigate further in this direction.

At the same time, in CCS and in π-calculus, *up-to proof techniques* are extremely useful for proving bisimilarity whenever exhibiting a bisimulation is awkward. Probably suitable up-to proof techniques built on top of our bisimilarity would make easier the proof of the sandbox. We will address the development of up-to proof techniques for higher-order process calculi in the next chapter, in the context of Mobile Ambients.

4 Mobile Ambients

The calculus of *Mobile Ambients* [CG00b], abbreviated MA, has been introduced as a process calculus for describing *mobile agents*. In MA, the term $n[P]$ represents an agent, or *ambient*, named n, executing the code P. The ambient n is a bounded, protected, and (potentially) mobile space where the computation P takes place. In turn P may contain other ambients, may perform (local) *communications*, or may exercise *capabilities*, that allow entry to or exit from named ambients. *Ambient names*, such as n, are used to control access to the ambient's computation space and may be dynamically created as in the π-calculus, [MPW92], using the construct $(\boldsymbol{\nu} n)P$. A *system* in MA consists of a collection of ambients running in parallel where the knowledge of certain ambient names may be restricted.

As we have seen in the introduction, the definition of an appropriate notion of bisimilarity for Mobile Ambients revealed harder than expected, mainly because:

- It is difficult for an ambient n to control interferences that may originate either from other ambients in its environment or from the computation running at n itself, [LS00a].
- Ambient mobility is asynchronous — no permission is required to migrate into an ambient. As noticed in [San01], this may cause a *stuttering* phenomenon originated by ambients that may repeatedly enter and exit another ambient. Stuttering cannot be observed, and any characterisation of reduction barbed congruence should be insensitive to stuttering.
- One of the main algebraic laws of MA is the *perfect firewall equation*, [CG00b]:

$$(\boldsymbol{\nu} n)n[P] = \mathbf{0} \qquad \text{for } n \text{ not free in } P.$$

  If you suppose $P = \mathtt{in}\_k.\mathbf{0}$, it is evident that a bisimilarity that wants to capture this law must not observe the movements of *secret ambients*, that is those ambients, like n, whose names are not known by the rest of the system.

In [MH02], it is introduced a labelled bisimilarity for an 'easier' variant of MA, called *Safe Ambients with Passwords*, (SAP), equipped with (i) *synchronous mobility*, as in Levi and Sangiorgi's *Safe Ambients* [LS00a], and (ii) *passwords* to exercise control over, and differentiate between, different ambients that may wish to exercise a capability. The main result in [MH02] is the characterisation of reduction barbed congruence in terms of the labelled bisimilarity. The result holds only in SAP and crucially relies on the two features (i) and (ii) mentioned above.

This chapter is the natural continuation of [MH02] where, now, we tackle the original problem: *to provide bisimulation proof methods for Mobile Ambients.*

Overview First, as in the Distributed π-calculus [HR98], we rewrite the syntax of MA in two levels: *processes* and *systems*. We introduce a new labelled transition system for

MA which is used to define a labelled bisimilarity over systems. The labelled bisimilarity is proved sound and complete with respect to reduction barbed congruence. We enhance our proof methods by introducing *up-to expansion* and *up-to context* proof techniques. We apply our bisimulation proof methods to prove a collection of both old and new algebraic laws.

4.1 Mobile Ambients in two levels

As in the Distributed π-calculus [HR98], we rewrite the syntax of MA in two levels: *processes* and *systems*. We are interested in studying systems rather than processes. So, our behavioural equalities are defined over systems. This little expedient allows us to focus on higher-order actions, which involve movement of code, avoiding the overhead caused by actions fired at top-level.

In Table 4.1 we report the syntax of MA, where $\mathbf{N}$ denotes an infinite set of names. Unlike other definitions of MA in the literature, our syntax is defined in a two-level structure, a lower one for *processes*, and an upper one for *systems*.

As regards processes, the constructs for inactivity, parallel composition, restriction and replicated prefixing are inherited from mainstream concurrent calculi, most notably the π-calculus [MPW92]. The inactive process, $\mathbf{0}$, does nothing. Parallel composition is denoted by a binary operator, $P \mid Q$, that is commutative and associative. The restriction operator, $(\boldsymbol{\nu} n)P$, creates a new (unique) name n within a scope P. We have replicated prefixing, $!C.P$, (rather than full replication) to create as many parallel replicas as needed.

Specific of the ambient calculus are the *ambient*, $n[P]$, and the *prefix* via capabilities, $C.P$. In $n[P]$, n is the name of the ambient and P is the process running inside the ambient. The process $C.P$ executes an action regulated by the capability C, and then continues as the process P. Capabilities are obtained from names; given a name n, the capability $\texttt{in\_}n$ allows entry into n, the capability $\texttt{out\_}n$ allows exit out of n, and the capability $\texttt{open\_}n$ allows the destruction of the boundary of ambient n. For the sake of simplicity, at this stage, we omit *communication*; it will be added in Section 4.5.

A system is a collection of ambients running in parallel where the knowledge of certain ambient names may be restricted among two or more ambients.

We use a number of notational conventions. Parallel composition has the lowest precedence among the operators. The process $C.C'.P$ is read as $C.(C'.P)$. We omit trailing dead processes, writing C for $C.\mathbf{0}$, and $n[\,]$ for $n[\mathbf{0}]$. Restriction $(\boldsymbol{\nu} n)P$ acts as binder for name n, P, $\mathrm{fn}(P)$, is defined accordingly. We write $\tilde{n}$ for the tuple $(n_1, \ldots, n_k)$, and $(\boldsymbol{\nu}\tilde{n})$ as a shorthand for $(\boldsymbol{\nu} n_1) \ldots (\boldsymbol{\nu} n_k)$.

Operational semantics A *static context* is a process context where the hole does not appear under a prefix or a replication. The dynamics of the calculus is specified by the *reduction relation* over processes described in Table 4.2. As systems are processes with a special structure, the rules of Table 4.2 also describe the evolution of systems. The *reduction semantics* relies on an auxiliary relation called *structural congruence* that brings the participants of a potential interaction into contiguous positions. It is easy to check that the reduction relation is closed under systems, that is, systems always reduce to systems.

Names: $a, b, \ldots, k, l, m, n, \ldots \in \mathbf{N}$

Systems:

M, N	$::= \mathbf{0}$	termination
	$\mid M_1 \mid M_2$	parallel composition
	$\mid (\boldsymbol{\nu} n)M$	restriction
	$\mid n[P]$	ambient

Capabilities:

C	$::= \texttt{in\_}n$	may enter into n
	$\mid \texttt{out\_}n$	may exit out of n
	$\mid \texttt{open\_}n$	may open n

Processes:

P, Q, R	$::= \mathbf{0}$	nil process
	$\mid P_1 \mid P_2$	parallel composition
	$\mid (\boldsymbol{\nu} n)P$	restriction
	$\mid C.P$	prefixing
	$\mid n[P]$	ambient
	$\mid !C.P$	replication

Table 4.1: Mobile Ambients in two levels

Behavioural semantics One of the main motivation of our work is the definition of a labelled bisimilarity for MA. Rather than simply defining an ad-hoc bisimulation based equivalence over systems we first introduce our reference equivalence. We consider the same notion of equivalence that we have used working with the Seal Calculus, that is, a generalisation of barbed congruence. We remind its definition below.

Definition 4.1.1 *A relation $\mathcal{R}$ over systems is* reduction closed *if $M \mathcal{R} N$ and $M \rightarrow M'$ implies the existence of some N' such that $N \rightarrow^* N'$ and $M' \mathcal{R} N'$, where $\rightarrow^*$ denotes the reflexive and transitive closure of $\rightarrow$.*

Definition 4.1.2 (System Context) *A* system context *is a context generated by the following grammar:*

$$C[-] ::= - \mid M \mid C[-] \mid C[-] \mid M \mid (\boldsymbol{\nu} n)C[-] \mid n[C[-] \mid P] \mid n[P \mid C[-]]$$

where M is an arbitrary system, and P is an arbitrary process.

Definition 4.1.3 *A relation $\mathcal{R}$ over systems is* contextual *if $M \mathcal{R} N$ implies $C[M] \mathcal{R} C[N]$ for all system contexts $C[-]$.*

$P \mid Q \equiv P \mid Q$	(Struct Par Comm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(Struct Par Assoc)
$P \mid \mathbf{0} \equiv P$	(Struct Zero Par)
$(\boldsymbol{\nu} n)\mathbf{0} \equiv \mathbf{0}$	(Struct Zero Res)
$!C.P \equiv C.P \mid !C.P$	(Struct Repl Par)
$(\boldsymbol{\nu} n)(\boldsymbol{\nu} m)P \equiv (\boldsymbol{\nu} m)(\boldsymbol{\nu} n)P$	(Struct Res Res)
$n \notin \mathrm{fn}(P)$ implies $(\boldsymbol{\nu} n)(P \mid Q) \equiv P \mid (\boldsymbol{\nu} n)Q$	(Struct Res Par)
$n \neq m$ implies $(\boldsymbol{\nu} n)(m[P]) \equiv m[(\boldsymbol{\nu} n)P]$	(Struct Res Amb)

$\equiv$ is the least equivalence relation which i) satisfies the axioms and rules above and ii) is preserved by all contexts.

$n[\mathtt{in\_}m.P \mid Q] \mid m[R] \rightarrowtriangle m[\, n[P \mid Q] \mid R \,]$	(Red In)
$m[\, n[\mathtt{out\_}m.P \mid Q] \mid R \,] \rightarrowtriangle n[P \mid Q] \mid m[R]$	(Red Out)
$\mathtt{open\_}n.P \mid n[\, Q \,] \rightarrowtriangle P \mid Q$	(Red Open)
$P \equiv Q \quad Q \rightarrowtriangle R \quad R \equiv S$ implies $P \rightarrowtriangle S$	(Red Struct)

$\rightarrowtriangle$ is the least relation which i) satisfies the rules above and ii) is preserved by all static contexts.

Table 4.2: Structural congruence and reduction rules

In Mobile Ambients the observation predicate $M \downarrow n$ denotes the possibility of the system M of interacting with the environment via the ambient n. We write $M \downarrow n$ if $M \equiv (\boldsymbol{\nu} \tilde{m})(n[P] \mid M')$ where $n \notin \{\tilde{m}\}$. We write $M \Downarrow n$ if there exists M' such that $M \rightarrowtriangle^* M'$ and $M' \downarrow n$.

Definition 4.1.4 *We say that a relation $\mathcal{R}$ over systems is* barb preserving *if $M \,\mathcal{R}\, N$ and $M \downarrow n$ implies $N \Downarrow n$.*

Definition 4.1.5 (Reduction barbed congruence) *Reduction barbed congruence, written $\cong$, is the largest symmetric relation over systems which is reduction closed, contextual, and barb preserving.*

4.2 A labelled transition semantics

In our language, the prefixes C give rise, in the standard manner, [Mil89], to transitions of the form $P \xrightarrow{C} Q$. For example we have

$$\mathtt{in\_}n.P_1 \mid P_2 \xrightarrow{\mathtt{in\_}n} P_1 \mid P_2.$$

However, similarly to [MH02], each of the capability C induces different and more complicated actions. The LTS is defined over processes, although in the labelled bisimilarity we

$$
\begin{array}{lll}
\textit{Pre-actions:} & \pi & ::= \mathtt{in}\_n \mid \mathtt{out}\_n \mid \mathtt{open}\_n \mid \mathtt{enter}\_n \mid \mathtt{amb}\_n \mid \mathtt{exit}\_n \\
\textit{Env-actions:} & \sigma & ::= k.\mathtt{enter}\_n \mid k.\mathtt{exit}\_n \mid *.\mathtt{enter}\_n \mid *.\mathtt{exit}\_n \\
 & & \quad \mid n.\overline{\mathtt{enter}}\_k \mid k.\mathtt{open}\_n \\
\textit{Actions:} & \alpha & ::= \sigma \cup \tau \\
\textit{Concretions:} & K & ::= (\boldsymbol{\nu}\tilde{m})\langle P\rangle Q \\
\textit{Outcomes:} & O & ::= P \mid K
\end{array}
$$

Table 4.3: Pre-actions, env-actions, actions, concretions, and outcomes

only consider actions going from systems to systems. We make a distinction between *pre-actions* and *env-actions*: the former denote the possibility to exercise certain capabilities whereas the latter model the interaction of a system with its environment. As usual, we also have *τ-actions* to model internal computations. Only env-actions and τ-actions model the evolution of a system at run-time.

The pre-actions, defined in Table 4.4, are of the form $P \xrightarrow{\pi} O$ where the ranges of π and of O, the *outcomes*, are reported in Table 4.3. An outcome may be a simple process Q, if for example π is a prefix of the language, or a *concretion*, of the form $(\boldsymbol{\nu}\tilde{m})\langle P\rangle Q$, when an ambient boundary is somehow involved. In this case, intuitively, P represents the part of the system affected by the action, while Q is not affected, and $\tilde{m}$ is the set of private names shared by P and Q. We adopt the convention that if K is the concretion $(\boldsymbol{\nu}\tilde{m})\langle P\rangle Q$, then $(\boldsymbol{\nu}r)K$ is a shorthand for $(\boldsymbol{\nu}\tilde{m})\langle P\rangle(\boldsymbol{\nu}r)Q$, if $r \notin \mathrm{fn}(P)$, and the concretion $(\boldsymbol{\nu}r\tilde{m})\langle P\rangle Q$ otherwise. We have a similar convention for the rule (π Par): $K \mid R$ is defined to be the concretion $(\boldsymbol{\nu}\tilde{m})\langle P\rangle(Q \mid R)$, where $\tilde{m}$ are chosen, using α-conversion if necessary, so that $\mathrm{fn}(R) \cap \{\tilde{m}\} = \emptyset$. Occasionally, we omit dead processes when they are in parallel with processes, writing P for $P \mid \mathbf{0}$.

The τ-actions, formally defined in Table 4.5, model the internal evolution of processes. Actually, there are three possible interactions: entering, exiting, and opening. Structural rules complete the LTS.

The env-actions, formally defined in Table 4.6, are of the form $M \xrightarrow{\sigma} M'$, where the range of σ is given in Table 4.3. Env-actions turn concretions into running systems by explicitly introducing the environment's ambient interacting with the process in question. The content of this ambient will be instantiated later, in the bisimilarity, with a process. For convenience, we extend the syntax of processes with the special process $\circ$ to pinpoint those ambients whose content will be instantiated later. The process $\circ$ does not reduce: it is simply a placeholder. Notice that, unlike pre-actions and τ-actions, env-actions do not have structural rules; this is because env-actions are supposed to be performed by systems that can directly interact with the environment.

We call *actions* the set of env-actions extended with τ. Actions always go from systems to systems and, in general, from processes to processes, even if the outcome may possibly involve the special process $\circ$. As our bisimilarity will be defined over systems, we will only

$$(\pi \text{ Pfx})\ \frac{}{\pi.P \xrightarrow{\pi} P} \qquad (\pi \text{ Repl Pfx})\ \frac{}{!\pi.P \xrightarrow{\pi} P \mid !\pi.P}$$

$$(\pi \text{ Enter})\ \frac{P \xrightarrow{\texttt{in\_}n} P_1}{m[P] \xrightarrow{\texttt{enter\_}n} \langle m[P_1]\rangle \mathbf{0}} \qquad (\pi \text{ Amb})\ \frac{}{n[P] \xrightarrow{\texttt{amb\_}n} \langle P\rangle \mathbf{0}}$$

$$(\pi \text{ Exit})\ \frac{P \xrightarrow{\texttt{out\_}n} P_1}{m[P] \xrightarrow{\texttt{exit\_}n} \langle m[P_1]\rangle \mathbf{0}} \qquad (\pi \text{ Res})\ \frac{P \xrightarrow{\pi} O \qquad n \notin \mathrm{fn}(\pi)}{(\boldsymbol{\nu} n)P \xrightarrow{\pi} (\boldsymbol{\nu} n)O}$$

$$(\pi \text{ Par})\ \frac{P \xrightarrow{\pi} O}{\begin{array}{l} P \mid Q \xrightarrow{\pi} O \mid Q \\ Q \mid P \xrightarrow{\pi} Q \mid O \end{array}}$$

Table 4.4: Labelled transition system: pre-actions

$$(\tau \text{ Enter})\ \frac{P \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{p})\langle P_1\rangle P_2 \quad Q \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{q})\langle Q_1\rangle Q_2 \quad {}^{(*)}}{\begin{array}{l} P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{p})(\boldsymbol{\nu}\tilde{q})(n[P_1 \mid Q_1] \mid P_2 \mid Q_2) \\ Q \mid P \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{q})(\boldsymbol{\nu}\tilde{p})(n[Q_1 \mid P_1] \mid Q_2 \mid P_2) \end{array}}$$

$$(\tau \text{ Exit})\ \frac{P \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{m})\langle k[P_1]\rangle P_2}{n[P] \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{m})(k[P_1] \mid n[P_2])} \qquad (\tau \text{ Amb})\ \frac{P \xrightarrow{\tau} Q}{n[P] \xrightarrow{\tau} n[Q]}$$

$$(\tau \text{ Open})\ \frac{P \xrightarrow{\texttt{open\_}n} P_1 \quad Q \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{m})\langle Q_1\rangle Q_2}{\begin{array}{l} P \mid Q \xrightarrow{\tau} P_1 \mid (\boldsymbol{\nu}\tilde{m})(Q_1 \mid Q_2) \\ Q \mid P \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{m})(Q_1 \mid Q_2) \mid P_1 \end{array}} \qquad (\tau \text{ Res})\ \frac{P \xrightarrow{\tau} P'}{(\boldsymbol{\nu} n)P \xrightarrow{\tau} (\boldsymbol{\nu} n)P'}$$

$$(\tau \text{ Par})\ \frac{P \xrightarrow{\tau} P'}{\begin{array}{l} P \mid Q \xrightarrow{\tau} P' \mid Q \\ Q \mid P \xrightarrow{\tau} Q \mid P' \end{array}}$$

(*) In rule (τ Enter) we require $((\mathrm{fn}(P_1) \cup \mathrm{fn}(P_2)) \cap \{\tilde{q}\}) = ((\mathrm{fn}(Q_1) \cup \mathrm{fn}(Q_2)) \cap \{\tilde{p}\}) = \emptyset$.

Table 4.5: Labelled transition system: τ-actions

consider actions (and not pre-actions) in its definition.

Proposition 4.2.1 *If T is a system (resp. a process), and $T \xrightarrow{\alpha} T'$, then T' is a system (resp. a process), possibly containing the special process $\circ$.*

We explain the rules induced by the the prefix `in`, the *immigration* of ambients. A typical

(Enter)
$$\frac{P \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{m})\langle k[P_1]\rangle P_2{}^{(\dagger)}}{P \xrightarrow{k.\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{m})(n[\circ \mid k[P_1]] \mid P_2)}$$

(Exit)
$$\frac{P \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{m})\langle k[P_1]\rangle P_2{}^{(\dagger)}}{P \xrightarrow{k.\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{m})(k[P_1] \mid n[\,\circ \mid P_2])}$$

(Co-Enter)
$$\frac{P \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{m})\langle P_1\rangle P_2{}^{(\dagger)}}{P \xrightarrow{n.\overline{\texttt{enter}}\_k} (\boldsymbol{\nu}\tilde{m})(n[P_1 \mid k[\,\circ\,]] \mid P_2)}$$

(Open)
$$\frac{P \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{m})\langle P_1\rangle P_2}{P \xrightarrow{k.\texttt{open\_}n} k[\circ \mid (\boldsymbol{\nu}\tilde{m})(P_1 \mid P_2)]}$$

(Enter Shh)
$$\frac{P \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{m})\langle k[P_1]\rangle P_2{}^{(\ddagger)}}{P \xrightarrow{*.\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{m})(n[\circ \mid k[P_1]] \mid P_2)}$$

(Exit Shh)
$$\frac{P \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{m})\langle k[P_1]\rangle P_2{}^{(\ddagger)}}{P \xrightarrow{*.\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{m})(k[P_1] \mid n[\circ \mid P_2])}$$

(†) In rules (Enter), (Co-Enter), and (Exit) we require $k \notin \tilde{m}$;

(‡) In rules (Enter Shh) and (Exit Shh) we require $k \neq n$ and $k \in \tilde{m}$.

Table 4.6: Labelled transition system: env-actions

example of an ambient m migrating into an ambient n follows:

$$(\boldsymbol{\nu}m)(m[\,\texttt{in\_}n.P_1 \mid P_2\,] \mid M) \mid n[Q] \;\rightarrowtriangle\; (\boldsymbol{\nu}m)(M \mid n[\,m[\,P_1 \mid P_2\,] \mid Q])$$

The driving force behind the migration is the activation of the prefix $\texttt{in\_}n$, within the ambient m. It induces a capability in the ambient m to migrate into n, that we formalise as a new action $\texttt{enter\_}n$. Thus an application of (π Enter) gives

$$m[\texttt{in\_}n.P_1 \mid P_2] \xrightarrow{\texttt{enter\_}n} \langle m[P_1 \mid P_2]\rangle \mathbf{0}$$

and more generally, using the structural rules (π Res) and (π Par),

$$(\boldsymbol{\nu}m)(m[\texttt{in\_}n.P_1 \mid P_2] \mid M) \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}m)\langle m[P_1 \mid P_2]\rangle M.$$

This means that the ambient $m[\texttt{in\_}n.P_1 \mid P_2]$ has the capability to enter an ambient n; if the capability is exercised, the ambient $m[P_1 \mid P_2]$ will enter n while M will be the residual where execution started. Of course the action can only be executed if there is an ambient n in parallel. The rule (π Amb) allows to check for the presence of ambients. So for example, we have

$$n[Q] \xrightarrow{\texttt{amb\_}n} \langle Q\rangle \mathbf{0}.$$

Here, the concretion $\langle Q\rangle\mathbf{0}$ says that Q is in n, while $\mathbf{0}$ is outside. Finally, the communication (τ Enter) allows these two complementary actions to occur simultaneously, executing the migration of the ambient $m[P_1 \mid P_2]$ from its current computation space into the ambient n, giving rise to the original move above:

$$(\boldsymbol{\nu}m)(m[\,\texttt{in\_}n.P_1 \mid P_2\,] \mid M) \mid n[Q] \xrightarrow{\tau} (\boldsymbol{\nu}m)(M \mid n[\,m[\,P_1 \mid P_2\,] \mid Q]).$$

Note that this is a *higher-order* interaction, as the ambient $m[P_1 \mid P_2]$ is transferred between two computation spaces.

We have not said yet what env-actions are useful for. They model the interaction of mobile agents with their environment. So, for instance, using the rule (Enter Shh), we derive from

$$(\boldsymbol{\nu} m)(m[\mathtt{in}\_n.P_1 \mid P_2] \mid M) \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu} m)\langle m[P_1 \mid P_2]\rangle M.$$

the transition

$$(\boldsymbol{\nu} m)(m[\mathtt{in}\_n.P_1 \mid P_2] \mid M) \xrightarrow{*.\mathtt{enter}\_n} (\boldsymbol{\nu} m)(n[\circ \mid m[P_1 \mid P_2]] \mid M).$$

This transition denotes a private (*secret*) ambient entering an ambient n provided by the environment. The computation running at n can be added later by instantiating the placeholder $\circ$.

Had the ambient name m not been restricted, we would have used the rule (Enter) to derive

$$m[\mathtt{in}\_n.P_1 \mid P_2] \mid M \xrightarrow{m.\mathtt{enter}\_n} n[\circ \mid m[P_1 \mid P_2]] \mid M$$

to model a global ambient m that enters an ambient n provided by the environment.

The rule for *emigration* follows the same line. A typical example of ambient m emigrating from ambient n is as follows:

$$n[\, m[\mathtt{out}\_n.P_1 \mid P_2] \mid Q] \rightarrowtriangle n[Q] \mid m[P_1 \mid P_2].$$

The driving force behind the emigration is the activation of the prefix $\mathtt{out}\_n$ within the ambient m. It induces a capability in the ambient m to emigrate from n, which we formalise as a new action $\mathtt{exit}\_n$. Thus an application of the rule (π Exit), followed by (π Par), gives

$$m[\mathtt{out}\_n.P_1 \mid P_2] \mid Q \xrightarrow{\mathtt{exit}\_n} \langle m[P_1 \mid P_2]\rangle Q.$$

Here when this capability is exercised the code Q will remain inside the ambient n while the ambient $m[P_1 \mid P_2]$ will move outside. However to actually effect the emigration of m we need a further context, namely the ambient n from which to emigrate. This leads to the rule (τ Exit); an application of which gives the original move above:

$$n[\, m[\mathtt{out}\_n.P_1 \mid P_2] \mid Q] \xrightarrow{\tau} n[Q] \mid m[P_1 \mid P_2].$$

Again, env-actions can model the exiting of both private and global ambients from an ambient provided by the environment.

Whenever a system offers a public ambient n at top-level, a context can interact with the system by providing an ambient entering n. The rule (Co-Enter) captures this interaction between system and environment. The rule that controls the *opening* follows along the same lines.

We end this section with a theorem that asserts that the LTS-based semantics coincides with the reduction semantics of Section 4.1.

For any process P, outcome O and pre-action π such that $P \xrightarrow{\pi} O$, the structure of P and O can be determined up to structural congruence. We write $O \equiv (\boldsymbol{\nu}\tilde{r})\langle P\rangle Q$ as a shorthand for $O = (\boldsymbol{\nu}\tilde{r})\langle P'\rangle Q'$ with $P \equiv P'$ and $Q \equiv Q'$.

Lemma 4.2.2

- *If* $P \xrightarrow{\texttt{in\_}n} O$ *then there exist* $\tilde{p}, P_1, P_2$, *with* $n \notin \tilde{p}$, *such that*

$$P \equiv (\boldsymbol{\nu}\tilde{p})(\texttt{in\_}n.P_1 \mid P_2) \quad \textit{and} \quad O \equiv (\boldsymbol{\nu}\tilde{p})(P_1 \mid P_2)\ ;$$

- *if* $P \xrightarrow{\texttt{out\_}n} O$ *then there exist* $\tilde{p}, P_1, P_2$, *with* $n \notin \tilde{p}$, *such that*

$$P \equiv (\boldsymbol{\nu}\tilde{p})(\texttt{out\_}n.P_1 \mid P_2) \quad \textit{and} \quad O \equiv (\boldsymbol{\nu}\tilde{p})(P_1 \mid P_2)$$

- *if* $P \xrightarrow{\texttt{open\_}n} O$ *then there exist* $\tilde{p}, P_1, P_2$, *with* $n \notin \tilde{p}$, *such that*

$$P \equiv (\boldsymbol{\nu}\tilde{p})(\texttt{open\_}n.P_1 \mid P_2) \quad \textit{and} \quad O \equiv (\boldsymbol{\nu}\tilde{p})(P_1 \mid P_2)\ ;$$

- *if* $P \xrightarrow{\texttt{enter\_}n} O$ *then there exist* $\tilde{p}, \tilde{r}, k, P_1, P_2, P_3$, *with* $n \notin \tilde{p}, \tilde{r}$, *such that*

$$P \equiv (\boldsymbol{\nu}\tilde{p})(k[(\boldsymbol{\nu}\tilde{r})(\texttt{in\_}n.P_1 \mid P_2)] \mid P_3) \quad \textit{and} \quad O \equiv (\boldsymbol{\nu}\tilde{p})\langle k[(\boldsymbol{\nu}\tilde{r})(P_1 \mid P_2)]\rangle P_3\ ;$$

- *if* $P \xrightarrow{\texttt{exit\_}n} O$ *then there exist* $\tilde{p}, \tilde{r}, k, P_1, P_2, P_3$, *with* $n \notin \tilde{p}, \tilde{r}$, *such that*

$$P \equiv (\boldsymbol{\nu}\tilde{p})(k[(\boldsymbol{\nu}\tilde{r})(\texttt{out\_}n.P_1 \mid P_2)] \mid P_3) \quad \textit{and} \quad O \equiv (\boldsymbol{\nu}\tilde{p})\langle k[(\boldsymbol{\nu}\tilde{r})(P_1 \mid P_2)]\rangle P_3\ ;$$

- *if* $P \xrightarrow{\texttt{amb\_}n} O$ *then there exist* $\tilde{p}, P_1, P_2$, *with* $n \notin \tilde{p}$, *such that*

$$P \equiv (\boldsymbol{\nu}\tilde{p})(n[P_1] \mid P_2) \quad \textit{and} \quad O \equiv (\boldsymbol{\nu}\tilde{p})\langle P_1\rangle P_2\ .$$

Proof By inspection of the LTS. □

Theorem 4.2.3

1. *If* $P \xrightarrow{\tau} P'$ *then* $P \rightarrowtriangle P'$
2. *If* $P \rightarrowtriangle P'$ *then* $P \xrightarrow{\tau} \equiv P'$.

Proof

Part 1. By induction of the derivation of $P \xrightarrow{\tau} P'$. τ-transitions can only be generated by the rules in Table 4.5.

(τ Enter) We know that $P \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{p})\langle P_1\rangle P_2$, and $Q \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{q})\langle Q_1\rangle Q_2$. From Lemma 4.2.2 we deduce that $P \equiv (\boldsymbol{\nu}\tilde{p})(k[(\boldsymbol{\nu}\tilde{r})(\texttt{in\_}n.P_3 \mid P_4)] \mid P_2)$, where $P_1 \equiv k[(\boldsymbol{\nu}\tilde{r})(P_3 \mid P_4)]$, for some processes P_3, P_4 and names $\tilde{r}$. Lemma 4.2.2 also guarantees that $Q \equiv (\boldsymbol{\nu}\tilde{q})(n[Q_1] \mid Q_2$. Then,

$$\begin{aligned} P \mid Q &\equiv (\boldsymbol{\nu}\tilde{p})(k[(\boldsymbol{\nu}\tilde{r})(\texttt{in\_}n.P_3 \mid P_4)] \mid P_2) \mid (\boldsymbol{\nu}\tilde{q})(n[Q_1] \mid Q_2) \\ &\equiv (\boldsymbol{\nu}\tilde{p})(\boldsymbol{\nu}\tilde{r})(\boldsymbol{\nu}\tilde{q})(k[\texttt{in\_}n.P_3 \mid P_4] \mid n[Q_1] \mid P_2 \mid Q_2) \\ &\rightarrowtriangle (\boldsymbol{\nu}\tilde{p})(\boldsymbol{\nu}\tilde{r})(\boldsymbol{\nu}\tilde{q})(n[k[P_3 \mid P_4] \mid Q_1] \mid P_2 \mid Q_2) \\ &\equiv (\boldsymbol{\nu}\tilde{p})(\boldsymbol{\nu}\tilde{q})(n[P_1 \mid Q_1] \mid P_2 \mid Q_2) \end{aligned}$$

as desired.

(τ Exit) We know that $P \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{p})\langle k[P_1]\rangle P_2$. From Lemma 4.2.2 we deduce that $P \equiv (\boldsymbol{\nu}\tilde{p})(k[(\boldsymbol{\nu}\tilde{r})(\texttt{out\_}n.P_3 \mid P_4)] \mid P_2)$, where $P_1 \equiv (\boldsymbol{\nu}\tilde{r})(P_3 \mid P_4)$, for some processes P_3, P_4 and names $\tilde{r}$. Then,

$$\begin{aligned} n[P] &\equiv n[(\boldsymbol{\nu}\tilde{p})(k[(\boldsymbol{\nu}\tilde{r})(\texttt{out\_}n.P_3 \mid P_4)] \mid P_2)] \\ &\equiv (\boldsymbol{\nu}\tilde{p})(\boldsymbol{\nu}\tilde{r})n[k[\texttt{out\_}n.P_3 \mid P_4] \mid P_2)] \\ &\rightarrowtriangle (\boldsymbol{\nu}\tilde{p})(\boldsymbol{\nu}\tilde{r})(n[P_2]k[P_3 \mid P_4]) \\ &\equiv (\boldsymbol{\nu}\tilde{p})(n[P_2]k[(\boldsymbol{\nu}\tilde{r})(P_3 \mid P_4)]) \end{aligned}$$

as desired.

(τ Open) We know that $P \xrightarrow{\texttt{open\_}n} P_1$ and $Q \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{q})\langle Q_1\rangle Q_2$. From Lemma 4.2.2 we deduce that $P \equiv (\boldsymbol{\nu}\tilde{p})(\texttt{open\_}n.P_2 \mid P_3)$, where $P_1 \equiv (\boldsymbol{\nu}\tilde{p})(P_2 \mid P_3)$ for for some processes P_2, P_3. Lemma 4.2.2 also guarantees that $Q \equiv (\boldsymbol{\nu}\tilde{q})(n[Q_1] \mid Q_2)$. Then,

$$\begin{aligned} P \mid Q &\equiv (\boldsymbol{\nu}\tilde{p})(\texttt{open\_}n.P_2 \mid P_3) \mid (\boldsymbol{\nu}\tilde{q})(n[Q_1] \mid Q_2) \\ &\equiv (\boldsymbol{\nu}\tilde{p})(\boldsymbol{\nu}\tilde{q})(\texttt{open\_}n.P_2 \mid n[Q_1] \mid P_3 \mid Q_2) \\ &\rightarrowtriangle (\boldsymbol{\nu}\tilde{p})(\boldsymbol{\nu}\tilde{q})(P_2 \mid Q_1 \mid P_3 \mid Q_2) \\ &\equiv (\boldsymbol{\nu}\tilde{p})(P_2 \mid P_3) \mid (\boldsymbol{\nu}\tilde{q})(Q_1 \mid Q_2) \end{aligned}$$

as desired.

The other cases follows straightforwardly from the congruence rules of the reduction relation.

Part 2. By induction on the derivation of $P \rightarrowtriangle Q$. There are three base cases.

(Red In) We know that

$$n[\texttt{in\_}m.P \mid Q] \mid m[R] \rightarrowtriangle m[n[P \mid Q] \mid R] \ .$$

The derivation below is valid.

$$\dfrac{\dfrac{\dfrac{\texttt{in\_}m.P \xrightarrow{\texttt{in\_}m} P}{\texttt{in\_}m.P \mid Q \xrightarrow{\texttt{in\_}m} P \mid Q}}{n[\texttt{in\_}m.P \mid Q] \xrightarrow{\texttt{enter\_}m} \langle P \mid Q\rangle \mathbf{0}} \qquad m[R] \xrightarrow{\texttt{amb\_}m} \langle R\rangle \mathbf{0}}{n[\texttt{in\_}m.P \mid Q] \mid m[R] \xrightarrow{\tau} m[n[P \mid Q] \mid R]}$$

(Red Out) We know that

$$m[n[\texttt{out\_}m.P \mid Q] \mid R] \rightarrowtriangle n[P \mid Q] \mid m[R]$$

The derivation below is valid.

$$\dfrac{\dfrac{\dfrac{\dfrac{\texttt{out}\_m.P \xrightarrow{\texttt{out}\_m} P}{\texttt{out}\_m.P \mid Q \xrightarrow{\texttt{out}\_m} P \mid Q}}{n[\texttt{out}\_m.P \mid Q] \xrightarrow{\texttt{exit}\_m} \langle n[P \mid Q]\rangle \mathbf{0}}}{n[\texttt{out}\_m.P \mid Q] \mid R \xrightarrow{\texttt{exit}\_m} \langle n[P \mid Q]\rangle R}}{m[n[\texttt{out}\_m.P \mid Q] \mid R] \xrightarrow{\tau} n[P \mid Q] \mid m[R]}$$

(Red Open) We know that

$$\texttt{open}\_n.P \mid n[Q] \rightarrowtriangle P \mid Q$$

The derivation below is valid.

$$\dfrac{\texttt{open}\_n.P \xrightarrow{\texttt{open}\_n} P \qquad n[Q] \xrightarrow{\texttt{amb}\_n} \langle Q\rangle \mathbf{0}}{\texttt{open}\_n.P \mid n[Q] \xrightarrow{\tau} P \mid Q}$$

The induction step, rule (Red Struct), follows because τ-transitions are preserved by all static contexts. □

Since we are interested in *weak bisimilarities*, that abstract over τ-actions, we introduce the notion of weak action. The definition is standard: $\Rightarrow$ denotes the reflexive and transitive closure of $\xrightarrow{\tau}$; $\stackrel{\alpha}{\Longrightarrow}$ denotes $\Rightarrow \xrightarrow{\alpha} \Rightarrow$; $\stackrel{\hat{\alpha}}{\Longrightarrow}$ denotes $\Rightarrow$ if $\alpha = \tau$ and $\stackrel{\alpha}{\Longrightarrow}$ otherwise.

Corollary 4.2.4 *If $M \equiv N$ and $M \xrightarrow{\tau} M'$, then there is N' such that $N \xrightarrow{\tau} N'$ and $M' \equiv N'$.*

From the results above, it is easy to establish that if $M \cong N$ then

- $M \Downarrow n$ iff $N \Downarrow n$
- $M \Rightarrow M'$ implies there is N' such that $N \Rightarrow N'$ and $M' \cong N'$.

In the sequel we will use these properties without comment.

4.3 Characterising reduction barbed congruence

In this section we define a labelled bisimilarity for MA that completely characterises reduction barbed congruence.

In the previous section we said that actions (and more precisely env-actions) introduce a special process $\circ$ to pinpoint those ambients whose content will be specified in the bisimilarity. The $\bullet$ operator instantiates the placeholder with a process, as defined below. It performs a name-capture avoiding substitution. It should be pointed out that we allow structural congruence to rearrange terms containing $\circ$: with respect to structural congruence, $\circ$ behaves like the inactive process $\mathbf{0}$. This motivates the introduction of the special process $\circ$, instead of relying on standard process substitutions.

Definition 4.3.1 *Let T, T_1, and T_2 range over both systems and processes. Then, given a process P, we define:*

$$\begin{array}{rcl@{\qquad}rcl}
\mathbf{0} \bullet P & \stackrel{\text{def}}{=} & \mathbf{0} & (T_1 \mid T_2) \bullet P & \stackrel{\text{def}}{=} & (T_1 \bullet P) \mid (T_2 \bullet P) \\
n[R] \bullet P & \stackrel{\text{def}}{=} & n[R \bullet P] & (\boldsymbol{\nu} n)T \bullet P & \stackrel{\text{def}}{=} & (\boldsymbol{\nu} n)(T \bullet P) \text{ if } n \notin \mathrm{fn}(P) \\
\circ \bullet P & \stackrel{\text{def}}{=} & P & C.R \bullet P & \stackrel{\text{def}}{=} & C.(R \bullet P) \\
!C.R \bullet P & \stackrel{\text{def}}{=} & !C.(R \bullet P). & & &
\end{array}$$

Everything is in place to define our bisimilarity.

Definition 4.3.2 (Late bisimilarity) *A symmetric relation $\mathcal{R}$ between systems is a* late bisimulation *if $M \mathcal{R} N$ implies:*

- *if $M \xrightarrow{\alpha} M'$, $\alpha \notin \{*.\texttt{enter}\_n, *.\texttt{exit}\_n\}$, then there is a system N' such that $N \stackrel{\hat{\alpha}}{\Longrightarrow} N'$ and for all processes P it holds $M' \bullet P \mathcal{R} N' \bullet P$;*

- *if $M \xrightarrow{*.\texttt{enter}\_n} M'$ then there is a system N' such that $N \mid n[\,\circ\,] \Rightarrow N'$ and for all processes P it holds $M' \bullet P \mathcal{R} N' \bullet P$;*

- *if $M \xrightarrow{*.\texttt{exit}\_n} M'$ then there is a system N' such that $n[\circ \mid N] \Rightarrow N'$ and for all processes P it holds $M' \bullet P \mathcal{R} N' \bullet P$.*

Systems M and N are late bisimilar*, written $M \approx N$, if $M \mathcal{R} N$ for some late bisimulation $\mathcal{R}$.*

The bisimilarity above has a universal quantification over the process P provided by the environment. This process instantiates the special process $\circ$ generated via env-actions. The bisimilarity is defined in a *late* style as the existential quantification precedes the universal one. Another possibility would be to define the bisimilarity in *early* style where the universal quantification over the environment's contribution P precedes that over the derivative N'. We write $\approx_e$ to denote this early variant. By definition, every late bisimulation is also a early one, while the converse, in general, does not hold. In our case, as in HOπ [San96a], we will prove that late and early bisimilarity actually coincide. As a consequence, late bisimilarity will be our main labelled bisimilarity because the derivatives N' do not depend on processes P.

In the definition of bisimilarity, actions $*.\texttt{enter}\_n$ and $*.\texttt{exit}\_n$ are treated apart asking for weaker matching requirements. This is because both actions are not observable. Somehow, this is very similar to what happens with input actions in the asynchronous π-calculus [HT91, Bou92].

Finally, the π-calculus experience suggests that late bisimulations like the one we defined might fail to be transitive. As we will see, processes reveal to be more 'tractable' than names, and in our framework late bisimilarity turns out to be an equivalence relation.

4.3.1 Soundness

Late and early bisimilarity represent two proof techniques for reduction barbed congruence. More precisely we prove that they are both contextual and contained in reduction barbed congruence.

The following lemma is crucial for proving that $\approx$ is contextual. This lemma will be also used for proving the soundness of the up-to context proof techniques in Section 4.4.

Lemma 4.3.3 *Let $\mathcal{S}$ be a contextual symmetric relation between systems. Let $(M, N) \in \mathcal{S}$ be a pair satisfying the bisimulation conditions in $\mathcal{S}$, that is,*

- *if $M \xrightarrow{\alpha} M'$, $\alpha \notin \{*.\texttt{enter}\_n, *.\texttt{exit}\_n\}$, then there is a system N' such that $N \stackrel{\hat{\alpha}}{\Longrightarrow} N'$ and for all processes P it holds $M' \bullet P \; \mathcal{S} \; N' \bullet P$;*
- *if $M \xrightarrow{*.\texttt{enter}\_n} M'$ then there is a system N' such that $N \mid n[\,\circ\,] \Rightarrow N'$ and for all processes P it holds $M' \bullet P \; \mathcal{S} \; N' \bullet P$;*
- *if $M \xrightarrow{*.\texttt{exit}\_n} M'$ then there is a system N' such that $n[\circ \mid N] \Rightarrow N'$ and for all processes P it holds $M' \bullet P \; \mathcal{S} \; N' \bullet P$.*

Then, for any system context $C[-]$, all the pairs $(C[M], C[N])$ also satisfy the bisimulation conditions in $\mathcal{S}$.

Proof The relation $\mathcal{S}$ is contextual, and as such it is the smallest relation between systems such that:

- if $M \; \mathcal{S} \; N$, then $M \mid H \; \mathcal{S} \; N \mid H$ and $H \mid M \; \mathcal{S} \; H \mid N$ for all systems H;
- if $M \; \mathcal{S} \; N$, then $(\boldsymbol{\nu} m)M \; \mathcal{S} \; (\boldsymbol{\nu} m)N$ for all names m;
- if $M \; \mathcal{S} \; N$, then $m[M \mid P] \; \mathcal{S} \; m[N \mid P]$ and $m[P \mid M] \; \mathcal{S} \; m[P \mid N]$ for all names m and processes P.

We prove the closure of $C[M] \; \mathcal{S} \; C[N]$ under the conditions for being a bisimulation by induction on the structure of $C[-]$ (we will not discuss the symmetric cases: they follow accordingly).

- $C[-] = -$.

  This case holds because $M \; \mathcal{S} \; N$ satisfies the bisimulation conditions in $\mathcal{S}$.

- $C[-] = (\boldsymbol{\nu} m)D[-]$.

  We know that $D[M] \; \mathcal{S} \; D[N]$ satisfies the bisimulation conditions in $\mathcal{S}$, and we want to prove that $(\boldsymbol{\nu} m)D[M] \; \mathcal{S} \; (\boldsymbol{\nu} m)D[N]$ satisfies the bisimulation conditions in $\mathcal{S}$ as well. Suppose $(\boldsymbol{\nu} m)D[M] \xrightarrow{\alpha}$. We perform a case analysis on α.

  - $(\boldsymbol{\nu} m)D[M] \xrightarrow{\tau} O_1$.

    This can only be derived from $D[M] \xrightarrow{\tau} O_1$, where $O_1 = (\boldsymbol{\nu} m)O_1$. The induction hypothesis tells us that there exists a system O_2 such that $D[N] \Rightarrow O_2$ and $O_1 \; \mathcal{S} \; O_2$. We can derive $(\boldsymbol{\nu} m)D[N] \Rightarrow (\boldsymbol{\nu} m)O_2$ and conclude $(\boldsymbol{\nu} m)O_1 \; \mathcal{S} \; (\boldsymbol{\nu} m)O_2$ because $\mathcal{S}$ is closed under restriction.

– $(\boldsymbol{\nu} m)D[M] \xrightarrow{k.\mathtt{enter}\_n} O_1$.
Observe that this must have been derived from

$$\frac{\dfrac{D[M] \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2}{(\boldsymbol{\nu} m)D[M] \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu} m)(\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2}}{(\boldsymbol{\nu} m)D[M] \xrightarrow{k.\mathtt{enter}\_n} O_1 \;\equiv\; (\boldsymbol{\nu} m)(\boldsymbol{\nu}\tilde{r})(n[\circ \mid k[M_1]] \mid M_2)}$$

for some process M_1 and system M_2. Remark that this implies $m \neq n$ and $m \neq k$. As $D[M] \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2$ then $D[M] \xrightarrow{k.\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid k[M_1]] \mid M_2) = M'$. The induction hypothesis then tells us that there exist systems N', A, B such that $D[N] \Rightarrow A \xrightarrow{k.\mathtt{enter}\_n} B \Rightarrow N'$, and for all processes P it holds $M' \bullet P \;\mathcal{S}\; N' \bullet P$. As $A \xrightarrow{k.\mathtt{enter}\_n} B$, the system B must be of the form $(\boldsymbol{\nu}\tilde{s})(n[\circ \mid k[N_1]] \mid N_2)$, for some process N_1 and system N_2. It also holds $A \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{s})\langle k[N_1]\rangle N_2$. This implies $(\boldsymbol{\nu} m)A \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu} m)(\boldsymbol{\nu}\tilde{s})\langle k[N_1]\rangle N_2$, from which we can derive $(\boldsymbol{\nu} m)A \xrightarrow{k.\mathtt{enter}\_n} C \equiv (\boldsymbol{\nu} m)B = (\boldsymbol{\nu} m)(\boldsymbol{\nu}\tilde{s})(n[\circ \mid k[N_1]] \mid N_2)$. We obtain $(\boldsymbol{\nu} m)D[N] \Rightarrow (\boldsymbol{\nu} m)A \xrightarrow{k.\mathtt{enter}\_n} C \Rightarrow\equiv (\boldsymbol{\nu} m)N'$. Call $(\boldsymbol{\nu} m)N' = O_2$. We can conclude that for all processes P, it holds $O_1 \bullet P \;\mathcal{S}\; O_2 \bullet P$ up to structural congruence, because $\mathcal{S}$ is closed under restriction.

– $(\boldsymbol{\nu} m)D[M] \xrightarrow{k.\mathtt{exit}\_n} O_1$.
Observe that this must have been derived from

$$\frac{\dfrac{D[M] \xrightarrow{\mathtt{exit}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2}{(\boldsymbol{\nu} m)D[M] \xrightarrow{\mathtt{exit}\_n} (\boldsymbol{\nu} m)(\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2}}{(\boldsymbol{\nu} m)D[M] \xrightarrow{k.\mathtt{exit}\_n} O_1 \;\equiv\; (\boldsymbol{\nu} m)(\boldsymbol{\nu}\tilde{r})(n[\circ \mid M_2] \mid k[M_1])}$$

for some process M_1 and system M_2. Remark that this implies $m \neq n$ and $m \neq k$. As $D[M] \xrightarrow{\mathtt{exit}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2$ then $D[M] \xrightarrow{k.\mathtt{exit}\_n} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid M_2] \mid k[M_1]) = M'$. The induction hypothesis then tells us that there exist systems N', A, B such that $D[N] \Rightarrow A \xrightarrow{k.\mathtt{exit}\_n} B \Rightarrow N'$, and for all processes P it holds $M' \bullet P \;\mathcal{S}\; N' \bullet P$. As $A \xrightarrow{k.\mathtt{exit}\_n} B$, the system B must be of the form $(\boldsymbol{\nu}\tilde{s})(n[\circ \mid N_2] \mid k[N_1])$, for some process N_1 and system N_2. It also holds $A \xrightarrow{\mathtt{exit}\_n} (\boldsymbol{\nu}\tilde{s})\langle k[N_1]\rangle N_2$. This implies $(\boldsymbol{\nu} m)A \xrightarrow{\mathtt{exit}\_n} (\boldsymbol{\nu} m)(\boldsymbol{\nu}\tilde{s})\langle k[N_1]\rangle N_2$, from which we can derive $(\boldsymbol{\nu} m)A \xrightarrow{k.\mathtt{exit}\_n} C \equiv (\boldsymbol{\nu} m)B = (\boldsymbol{\nu} m)(\boldsymbol{\nu}\tilde{s})(n[\circ \mid N_2] \mid k[N_1])$. We obtain $(\boldsymbol{\nu} m)D[N] \Rightarrow (\boldsymbol{\nu} m)A \xrightarrow{k.\mathtt{exit}\_n} C \Rightarrow\equiv (\boldsymbol{\nu} m)N'$. Call $(\boldsymbol{\nu} m)N' = O_2$. We can conclude that for all processes P, it holds $O_1 \bullet P \;\mathcal{S}\; O_2 \bullet P$ up to structural congruence, because $\mathcal{S}$ is closed under restriction.

– $(\boldsymbol{\nu} m)D[M] \xrightarrow{n.\overline{\mathtt{enter}}\_k} O_1$.

Observe that this must have been derived from

$$\dfrac{\dfrac{D[M] \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle M_1\rangle M_2}{(\boldsymbol{\nu}m)D[M] \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{r})\langle M_1\rangle M_2}}{(\boldsymbol{\nu}m)D[M] \xrightarrow{n.\overline{\texttt{enter}}\_k} O_1 \equiv (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{r})(n[k[\,\circ\,] \mid M_1] \mid M_2)}$$

for some process M_1 and system M_2. Remark that this implies $m \neq n$ and $m \neq k$. As $D[M] \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle M_1\rangle M_2$ then $D[M] \xrightarrow{n.\overline{\texttt{enter}}\_k} (\boldsymbol{\nu}\tilde{r})(n[k[\,\circ\,] \mid M_1] \mid M_2) = M'$. The induction hypothesis then tells us that there exist systems N', A, B such that $D[N] \Rightarrow A \xrightarrow{n.\overline{\texttt{enter}}\_k} B \Rightarrow N'$, and for all processes P it holds $M' \bullet P \mathcal{S} N' \bullet P$. As $A \xrightarrow{n.\overline{\texttt{enter}}\_k} B$, the system B must be of the form $(\boldsymbol{\nu}\tilde{s})(n[k[\,\circ\,] \mid N_1] \mid N_2)$, for some process N_1 and system N_2. It also holds $A \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{s})\langle N_1\rangle N_2$. This implies $(\boldsymbol{\nu}m)A \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{s})\langle N_1\rangle N_2$, from which we can derive $(\boldsymbol{\nu}m)A \xrightarrow{n.\overline{\texttt{enter}}\_k} C \equiv (\boldsymbol{\nu}m)B = (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{s})(n[k[\,\circ\,] \mid N_1] \mid N_2)$. We obtain $(\boldsymbol{\nu}m)D[N] \Rightarrow (\boldsymbol{\nu}m)A \xrightarrow{n.\overline{\texttt{enter}}\_k} C \Rightarrow\equiv (\boldsymbol{\nu}m)N'$. Call $(\boldsymbol{\nu}m)N' = O_2$. We can conclude that for all processes P, it holds $O_1 \bullet P \mathcal{S} O_2 \bullet P$ up to structural congruence, because $\mathcal{S}$ is closed under restriction.

– $(\boldsymbol{\nu}m)D[M] \xrightarrow{k.\texttt{open}\_n} O_1$.
Observe that this must have been derived from

$$\dfrac{\dfrac{D[M] \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle M_1\rangle M_2}{(\boldsymbol{\nu}m)D[M] \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{r})\langle M_1\rangle M_2}}{(\boldsymbol{\nu}m)D[M] \xrightarrow{k.\texttt{open}\_n} O_1 \equiv k[\circ \mid (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{r})(M_1 \mid M_2)]}$$

for some process M_1 and system M_2. Remark that this implies $m \neq n$ and $m \neq k$. As $D[M] \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle M_1\rangle M_2$ then $D[M] \xrightarrow{k.\texttt{open}\_n} k[\circ \mid (\boldsymbol{\nu}\tilde{r})(M_1 \mid M_2)] = M'$. Also observe that $O_1 \equiv (\boldsymbol{\nu}m)k[\circ \mid (\boldsymbol{\nu}\tilde{r})(M_1 \mid M_2)] = (\boldsymbol{\nu}m)M'$. The induction hypothesis then tells us that there exist systems N', A, B such that $D[N] \Rightarrow A \xrightarrow{k.\texttt{open}\_n} B \Rightarrow N'$, and for all processes P it holds $M' \bullet P \mathcal{S} N' \bullet P$. As $A \xrightarrow{k.\texttt{open}\_n} B$, the system B must be of the form $k[\circ \mid (\boldsymbol{\nu}\tilde{s})(N_1 \mid N_2)]$, for some process N_1 and system N_2. It also holds $A \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{s})\langle N_1\rangle N_2$. This implies $(\boldsymbol{\nu}m)A \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{s})\langle N_1\rangle N_2$, from which we can derive $(\boldsymbol{\nu}m)A \xrightarrow{k.\texttt{open}\_n} C \equiv k[\circ \mid (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{s})(N_1 \mid N_2)] \equiv (\boldsymbol{\nu}m)k[\circ \mid (\boldsymbol{\nu}\tilde{s})(N_1 \mid N_2)] = (\boldsymbol{\nu}m)N'$. We obtain $(\boldsymbol{\nu}m)D[N] \Rightarrow (\boldsymbol{\nu}m)A \xrightarrow{k.\texttt{open}\_n} C \Rightarrow\equiv (\boldsymbol{\nu}m)N'$. Call $(\boldsymbol{\nu}m)N' = O_2$. We can conclude that for all processes P, it holds $O_1 \bullet P \mathcal{S} O_2 \bullet P$ up to structural congruence, because $\mathcal{S}$ is closed under restriction.

– $(\boldsymbol{\nu}m)D[M] \xrightarrow{*.\texttt{enter}\_n} O_1$.
Observe that there are two possible derivations: the entering ambient may be called m or not.

∗ Suppose:

$$\dfrac{\dfrac{D[M] \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{r})\langle m[M_1]\rangle M_2}{(\boldsymbol{\nu}m)D[M] \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{r})\langle m[M_1]\rangle M_2}}{(\boldsymbol{\nu}m)D[M] \xrightarrow{*.\texttt{enter\_}n} O_1 \equiv (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{r})(n[\circ \mid m[M_1]] \mid M_2)}$$

where $m \notin \tilde{r}$, for some process M_1 and system M_2. Remark that this implies $n \notin r$. As $D[M] \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{r})\langle m[M_1]\rangle M_2$ then $D[M] \xrightarrow{m.\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid m[M_1]] \mid M_2) = M'$. The induction hypothesis then tells us that there exist systems N', A, B such that $D[N] \Rightarrow A \xrightarrow{m.\texttt{enter\_}n} B \Rightarrow N'$, and for all processes P it holds $M' \bullet P \;\mathcal{S}\; N' \bullet P$. As $A \xrightarrow{m.\texttt{enter\_}n} B$, the system B must be of the form $(\boldsymbol{\nu}\tilde{s})(n[\circ \mid m[N_1]] \mid N_2)$, for some process N_1 and system N_2, where $m \notin \tilde{s}$. It also holds $A \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{s})\langle m[N_1]\rangle N_2$. This implies $(\boldsymbol{\nu}m)A \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{s})\langle m[N_1]\rangle N_2$, from which we can derive $(\boldsymbol{\nu}m)A \mid n[\,\circ\,] \xrightarrow{\tau} C \equiv (\boldsymbol{\nu}m)B = (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{s})(n[\circ \mid N_2] \mid m[N_1])$. We obtain $(\boldsymbol{\nu}m)(D[N] \mid n[\,\circ\,]) \equiv (\boldsymbol{\nu}m)D[N] \mid n[\,\circ\,] \Rightarrow (\boldsymbol{\nu}m)A \mid n[\,\circ\,] \xrightarrow{\tau} C \Rightarrow\equiv (\boldsymbol{\nu}m)N'$. Call $(\boldsymbol{\nu}m)N' = O_2$. We can conclude that for all processes P, it holds $O_1 \bullet P \;\mathcal{S}\; O_2 \bullet P$ up to structural congruence, because $\mathcal{S}$ is closed under restriction.

∗ Suppose:

$$\dfrac{\dfrac{D[M] \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2}{(\boldsymbol{\nu}m)D[M] \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2}}{(\boldsymbol{\nu}m)D[M] \xrightarrow{*.\texttt{enter\_}n} O_1 \equiv (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{r})(n[\circ \mid k[M_1]] \mid M_2)}$$

where $k \neq m$ and thus $k \in \tilde{r}$, for some process M_1 and system M_2. Remark that $n \notin \tilde{r}$. Since $D[M] \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2$, then $D[M] \xrightarrow{*.\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid k[M_1]] \mid M_2) = M'$. The induction hypothesis then tells us that there exists a system N' such that $D[N] \mid n[\,\circ\,] \Rightarrow N'$, and for all processes P it holds $M' \bullet P \;\mathcal{S}\; N' \bullet P$. We can derive $(\boldsymbol{\nu}m)D[N] \mid n[\,\circ\,] \equiv (\boldsymbol{\nu}m)(D[N] \mid n[\circ]) \Rightarrow (\boldsymbol{\nu}m)N'$. Call $(\boldsymbol{\nu}m)N' = O_2$. We can conclude that for all processes P, it holds $O_1 \bullet P \;\mathcal{S}\; O_2 \bullet P$ up to structural congruence, because $\mathcal{S}$ is closed under restriction.

– $(\boldsymbol{\nu}m)D[M] \xrightarrow{*.\texttt{exit\_}n} O_1$.

Observe that there are two possible derivations.

∗ Suppose:

$$\dfrac{\dfrac{D[M] \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle m[M_1]\rangle M_2}{(\boldsymbol{\nu}m)D[M] \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{r})\langle m[M_1]\rangle M_2}}{(\boldsymbol{\nu}m)D[M] \xrightarrow{*.\texttt{exit\_}n} O_1 \equiv (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{r})(n[\circ \mid M_2] \mid m[M_1])}$$

where $m \notin \tilde{r}$, for some process M_1 and system M_2. Remark that this implies $n \notin r$. As $D[M] \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle m[M_1]\rangle M_2$ then $D[M] \xrightarrow{m.\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid M_2] \mid m[M_1]) = M'$. The induction hypothesis then tells us that there exist systems N', A, B such that $D[N] \Rightarrow A \xrightarrow{m.\texttt{exit\_}n} B \Rightarrow N'$, and for all processes P it holds $M' \bullet P \mathcal{S} N' \bullet P$. As $A \xrightarrow{m.\texttt{exit\_}n} B$, the system B must be of the form $(\boldsymbol{\nu}\tilde{s})(n[\circ \mid N_2] \mid m[N_1])$, for some process N_1 and system N_2, where $m \notin \tilde{s}$. It also holds $A \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{s})\langle k[N_1]\rangle N_2$. This implies $(\boldsymbol{\nu}m)A \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{s})\langle m[N_1]\rangle N_2$, from which we can derive $(\boldsymbol{\nu}m)n[\circ \mid A] \xrightarrow{\tau} C \equiv (\boldsymbol{\nu}m)B = (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{s})(n[\circ \mid N_2] \mid m[N_1])$. We obtain $(\boldsymbol{\nu}m)(D[N] \mid n[\,\circ\,]) \equiv (\boldsymbol{\nu}m)D[N] \mid n[\,\circ\,] \Rightarrow (\boldsymbol{\nu}m)A \mid n[\,\circ\,] \xrightarrow{\tau} C \Rightarrow\equiv (\boldsymbol{\nu}m)N'$. Call $(\boldsymbol{\nu}m)N' = O_2$. We can conclude that for all processes P, it holds $O_1 \bullet P \mathcal{S} O_2 \bullet P$ up to structural congruence, because $\mathcal{S}$ is closed under restriction.

∗ Suppose:

$$\dfrac{\dfrac{D[M] \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2}{(\boldsymbol{\nu}m)D[M] \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2}}{(\boldsymbol{\nu}m)D[M] \xrightarrow{*.\texttt{exit\_}n} O_1 \;\equiv\; (\boldsymbol{\nu}m)(\boldsymbol{\nu}\tilde{r})(n[\circ \mid M_2] \mid k[M_1])}$$

where $k \neq m$ and thus $k \in \tilde{r}$, for some process M_1 and system M_2. Remark that $n \notin \tilde{r}$. Since $D[M] \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2$, then $D[M] \xrightarrow{*.\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid M_2] \mid k[M_1]) = M'$. The induction hypothesis then tells us that there exists a system N' such that $n[\circ \mid D[N]] \Rightarrow N'$, and for all processes P it holds $M' \bullet P \mathcal{S} N' \bullet P$. We can derive $(\boldsymbol{\nu}m)D[N] \mid n[\,\circ\,] \equiv (\boldsymbol{\nu}m)(D[N] \mid n[\circ]) \Rightarrow (\boldsymbol{\nu}m)N'$. Call $(\boldsymbol{\nu}m)N' = O_2$. We can conclude that for all processes P, it holds $O_1 \bullet P \mathcal{S} O_2 \bullet P$ up to structural congruence, because $\mathcal{S}$ is closed under restriction.

- $C[-] = D[-] \mid H$.

  We know that $D[M] \mathcal{S} D[N]$ satisfies the bisimulation conditions in $\mathcal{S}$, and we want to prove that $D[M] \mid H \mathcal{S} D[N] \mid H$ satisfies the bisimulation conditions in $\mathcal{S}$ as well. We perform a case analysis on the transition $D[M] \mid H \xrightarrow{\alpha} O_1$.

  We consider first the cases when there is no interaction between $D[M]$ and H.

  – $D[M] \mid H \xrightarrow{\tau} O_1$, because $D[M] \xrightarrow{\tau} M'$ and $O_1 \equiv M' \mid H$. The induction hypothesis tells us that there exists a N' such that $D[N] \Rightarrow N'$ and $M' \mathcal{S} N'$. Thus, $D[N] \mid H \Rightarrow O_2 \equiv N' \mid H$ and $O_1 \equiv M' \mid H \mathcal{S} N' \mid H \equiv O_2$ because $\mathcal{S}$ is closed under parallel composition.

  – $D[M] \mid H \xrightarrow{\tau} O_1$, because $H \xrightarrow{\tau} H'$ and $O_1 \equiv D[M] \mid H'$. Let $O_2 = D[N] \mid H'$: it holds $D[N] \mid H \xrightarrow{\tau} O_2$, and $O_1 \mathcal{S} O_2$ because $D[M] \mathcal{S} D[N]$ and $\mathcal{S}$ is closed under parallel composition.

– $D[M] \mid H \xrightarrow{k.\mathtt{enter}\_n} O_1$.

There are two possible derivations.

∗ Suppose:

$$\dfrac{\dfrac{D[M] \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2}{D[M] \mid H \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2 \mid H}}{D[M] \mid H \xrightarrow{k.\mathtt{enter}\_n} O_1 \;\equiv\; (\boldsymbol{\nu}\tilde{r})(n[\circ \mid k[M_1]] \mid M_2 \mid H)}$$

for some process M_1 and system M_2. Remark that $k \notin \tilde{r}$. As $D[M] \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2$ then $D[M] \xrightarrow{k.\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid k[M_1]] \mid M_2) = M'$. The induction hypothesis then tells us that there exist systems N', A, B such that $D[N] \Rightarrow A \xrightarrow{k.\mathtt{enter}\_n} B \Rightarrow N'$, and for all processes P it holds $M' \bullet P \;\mathcal{S}\; N' \bullet P$. As $A \xrightarrow{k.\mathtt{enter}\_n} B$, the system B must be of the form $(\boldsymbol{\nu}\tilde{s})(n[\circ \mid k[N_1]] \mid N_2)$, for some process N_1 and system N_2. It also holds $A \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{s})\langle k[N_1]\rangle N_2$. This implies $A \mid H \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{s})\langle k[N_1]\rangle N_2 \mid H$, from which we can derive $A \mid H \xrightarrow{k.\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{s})(n[\circ \mid k[N_1]] \mid N_2 \mid H) \equiv B \mid H$. We obtain $D[N] \mid H \Rightarrow A \mid H \xrightarrow{k.\mathtt{enter}\_n}\equiv B \mid H \Rightarrow\equiv N' \mid H$. Call $N' \mid H = O_2$. We can conclude that for all processes P, it holds $O_1 \bullet P \;\mathcal{S}\; O_2 \bullet P$ up to structural congruence, because $\mathcal{S}$ is closed under parallel composition.

∗ Suppose:

$$\dfrac{\dfrac{H \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2}{D[M] \mid H \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2 \mid D[M]}}{D[M] \mid H \xrightarrow{k.\mathtt{enter}\_n} O_1 \;\equiv\; (\boldsymbol{\nu}\tilde{r})(n[\circ \mid k[H_1]] \mid H_2 \mid M)}$$

for some process H_1 and system H_2. Remark that $k \notin \tilde{r}$. We can construct the following derivation:

$$\dfrac{\dfrac{H \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2}{D[N] \mid H \xrightarrow{\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2 \mid D[N]}}{D[N] \mid H \xrightarrow{k.\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid k[H_1]] \mid H_2 \mid D[N]) \;=\; O_2}$$

We can conclude that for all processes P, it holds $O_1 \bullet P \;\mathcal{S}\; O_2 \bullet P$ up to structural congruence, because $D[M] \;\mathcal{S}\; D[N]$ and $\mathcal{S}$ is closed under parallel composition.

– $D[M] \mid H \xrightarrow{k.\mathtt{exit}\_n} O_1$.

There are two possible derivations.

∗ Suppose:

$$\dfrac{\dfrac{D[M] \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2}{D[M] \mid H \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2 \mid H}}{D[M] \mid H \xrightarrow{k.\texttt{exit\_}n} O_1 \equiv (\boldsymbol{\nu}\tilde{r})(n[\circ \mid M_2 \mid H] \mid k[M_1])}$$

for some process M_1 and system M_2. Remark that $k \notin \tilde{r}$. As $D[M] \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2$ then $D[M] \xrightarrow{k.\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid M_2] \mid k[M_1]) = M'$. The induction hypothesis then tells us that there exist systems N', A, B such that $D[N] \Rightarrow A \xrightarrow{k.\texttt{exit\_}n} B \Rightarrow N'$, and for all processes P it holds $M' \bullet P \mathcal{S} N' \bullet P$. Remark that $N' \equiv (\boldsymbol{\nu}\tilde{h})n[\circ \mid N_3] \mid N_4$, for some N_3, N_4. As $A \xrightarrow{k.\texttt{exit\_}n} B$, the system B must be of the form $(\boldsymbol{\nu}\tilde{s})(n[\circ \mid N_2] \mid k[N_1])$, for some process N_1 and system N_2. It also holds $A \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{s})\langle k[N_1]\rangle N_2$. This implies $A \mid H \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{s})\langle k[N_1]\rangle N_2 \mid H$, from which we can derive $A \mid H \xrightarrow{k.\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{s})(n[\circ \mid N_2 \mid H] \mid k[N_1]) \equiv B \bullet (\circ \mid H)$. We obtain $D[N] \mid H \Rightarrow A \mid H \xrightarrow{k.\texttt{exit\_}n} B \bullet (\circ \mid H) \Rightarrow\equiv N' \bullet (\circ \mid H)$. Call $N' \bullet (\circ \mid H) = O_2$. As for all processes P it holds $M' \bullet P \mathcal{S} N' \bullet P$, we can conclude that for all processes Q, it holds $O_1 \bullet Q \mathcal{S} O_2 \bullet Q$ up to structural congruence, because $O_1 \bullet Q \equiv M' \bullet (Q \mid H) \mathcal{S} N' \bullet (Q \mid H) \equiv O_2 \bullet Q$.

∗ Suppose:

$$\dfrac{\dfrac{H \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2}{D[M] \mid H \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2 \mid D[M]}}{D[M] \mid H \xrightarrow{k.\texttt{exit\_}n} O_1 \equiv (\boldsymbol{\nu}\tilde{r})(n[\circ \mid H_2 \mid D[M]] \mid k[H_1])}$$

for some process H_1 and system H_2. Remark that $k \notin \tilde{r}$. We can construct the following derivation:

$$\dfrac{\dfrac{H \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2}{D[N] \mid H \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2 \mid D[N]}}{D[N] \mid H \xrightarrow{k.\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid H_2 \mid D[N]] \mid k[H_1]) = O_2}$$

We can conclude that for all processes P, it holds $O_1 \bullet P \mathcal{S} O_2 \bullet P$ up to structural congruence, because $D[M] \mathcal{S} D[N]$ and $\mathcal{S}$ is closed under parallel composition and ambient.

– $D[M] \mid H \xrightarrow{n.\overline{\texttt{enter}}\_k} O_1$.

There are two possible derivations.

∗ Suppose:

$$\dfrac{\dfrac{D[M] \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle M_1\rangle M_2}{D[M] \mid H \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle M_1\rangle M_2 \mid H}}{D[M] \mid H \xrightarrow{n.\overline{\texttt{enter}}\_k} O_1 \;\equiv\; (\boldsymbol{\nu}\tilde{r})(n[k[\,\circ\,] \mid M_1] \mid M_2 \mid H)}$$

for some process M_1 and system M_2. Remark that $k, n \notin \tilde{r}$. As $D[M] \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle M_1\rangle M_2$ then $D[M] \xrightarrow{n.\overline{\texttt{enter}}\_k} (\boldsymbol{\nu}\tilde{r})(n[k[\,\circ\,] \mid M_1] \mid M_2) = M'$. The induction hypothesis then tells us that there exist systems N', A, B such that $D[N] \Rightarrow A \xrightarrow{n.\overline{\texttt{enter}}\_k} B \Rightarrow N'$, and for all processes P it holds $M' \bullet P \;\mathcal{S}\; N' \bullet P$. As $A \xrightarrow{n.\overline{\texttt{enter}}\_k} B$, the system B must be of the form $(\boldsymbol{\nu}\tilde{s})(n[k[\,\circ\,] \mid N_1] \mid N_2)$, for some process N_1 and system N_2. It also holds $A \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{s})\langle N_1\rangle N_2$. This implies $A \mid H \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{s})\langle N_1\rangle N_2 \mid H$, from which we can derive $A \mid H \xrightarrow{n.\overline{\texttt{enter}}\_k} (\boldsymbol{\nu}\tilde{s})(n[k[\circ] \mid N_1] \mid N_2 \mid H) \equiv B \mid H$. We obtain $D[N] \mid H \Rightarrow A \mid H \xrightarrow{n.\overline{\texttt{enter}}\_k}\equiv B \mid H \Rightarrow\equiv N' \mid H$. Call $N' \mid H = O_2$. We can conclude that for all processes P, it holds $O_1 \bullet P \;\mathcal{S}\; O_2 \bullet P$ up to structural congruence, because $\mathcal{S}$ is closed under parallel composition.

∗ Suppose:

$$\dfrac{\dfrac{H \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle H_1\rangle H_2}{D[M] \mid H \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle H_1\rangle H_2 \mid D[M]}}{D[M] \mid H \xrightarrow{n.\overline{\texttt{enter}}\_k} O_1 \;\equiv\; (\boldsymbol{\nu}\tilde{r})(n[k[\,\circ\,] \mid H_1] \mid H_2 \mid D[M])}$$

for some process H_1 and system H_2. Remark that $k \notin \tilde{r}$. We can construct the following derivation:

$$\dfrac{\dfrac{H \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle H_1\rangle H_2}{D[N] \mid H \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle H_1\rangle H_2 \mid D[N]}}{D[N] \mid H \xrightarrow{n.\overline{\texttt{enter}}\_k} (\boldsymbol{\nu}\tilde{r})(n[k[\,\circ\,] \mid H_1] \mid H_2 \mid D[N]) \;=\; O_2}$$

We can conclude that for all processes P, it holds $O_1 \bullet P \;\mathcal{S}\; O_2 \bullet P$ up to structural congruence, because $D[M] \;\mathcal{S}\; D[N]$ and $\mathcal{S}$ is closed under parallel composition.

– $D[M] \mid H \xrightarrow{k.\texttt{open}\_n} O_1$.

There are two possible derivations.

∗ Suppose:

$$\dfrac{\dfrac{D[M] \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle M_1\rangle M_2}{D[M] \mid H \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle M_1\rangle M_2 \mid H}}{D[M] \mid H \xrightarrow{k.\texttt{open\_}n} O_1 \;\equiv\; k[\circ \mid (\boldsymbol{\nu}\tilde{r})(M_1 \mid M_2) \mid H]}$$

for some process M_1 and system M_2. Remark that $k, n \notin \tilde{r}$. As $D[M] \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{r})\langle M_1\rangle M_2$ then $D[M] \xrightarrow{k.\texttt{open\_}n} k[\circ \mid (\boldsymbol{\nu}\tilde{r})(M_1 \mid M_2)]$. The induction hypothesis then tells us that there exist systems N', A, B such that $D[N] \Rightarrow A \xrightarrow{k.\texttt{open\_}n} B \Rightarrow N'$, and for all processes P it holds $M' \bullet P \;\mathcal{S}\; N' \bullet P$. As $A \xrightarrow{k.\texttt{open\_}n} B$, the system B must be of the form $k[\circ \mid (\boldsymbol{\nu}\tilde{s})(N_1 \mid N_2)]$, for some process N_1 and system N_2. It also holds $A \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{s})\langle N_1\rangle N_2$. This implies $A \mid H \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{s})\langle N_1\rangle N_2 \mid H$, from which we can derive $A \mid H \xrightarrow{k.\texttt{open\_}n} k[\circ \mid (\boldsymbol{\nu}\tilde{s})(N_1 \mid N_2) \mid H] \equiv B \bullet (\circ \mid H)$. We obtain $D[N] \mid H \Rightarrow A \mid H \xrightarrow{k.\texttt{open\_}n} \equiv B \bullet (\circ \mid H) \Rightarrow \equiv N' \bullet (\circ \mid H)$. Call $N' \bullet (\circ \mid H) = O_2$. We can conclude that for all processes P, it holds $O_1 \bullet P \;\mathcal{S}\; O_2 \bullet P$ up to structural congruence, because for all processes P it holds $M' \bullet (P \mid H) \;\mathcal{S}\; N' \bullet (P \mid H)$.

∗ Suppose:

$$\dfrac{\dfrac{H \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{h})\langle H_1\rangle H_2}{D[M] \mid H \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{h})\langle H_1\rangle H_2 \mid D[M]}}{D[M] \mid H \xrightarrow{k.\texttt{open\_}n} O_1 \;\equiv\; k[\circ \mid (\boldsymbol{\nu}\tilde{h})(H_1 \mid H_2) \mid D[M]]}$$

for some process H_1 and system H_2. Remark that $k \notin \tilde{h}$. We can construct the following derivation:

$$\dfrac{\dfrac{H \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{h})\langle H_1\rangle H_2}{D[N] \mid H \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{h})\langle H_1\rangle H_2 \mid D[N]}}{D[N] \mid H \xrightarrow{k.\texttt{open\_}n} k[\circ \mid (\boldsymbol{\nu}\tilde{h})(H_1 \mid H_2) \mid D[N]] \;=\; O_2}$$

We can conclude that for all processes P, it holds $O_1 \bullet P \;\mathcal{S}\; O_2 \bullet P$ up to structural congruence, because $D[M] \;\mathcal{S}\; D[N]$ and $\mathcal{S}$ is closed under parallel composition and ambient.

– $D[M] \mid H \xrightarrow{*.\texttt{enter\_}n} O_1$.

There are two possible derivations.

∗ Suppose:

$$\dfrac{\dfrac{D[M] \xrightarrow{\texttt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2}{D[M] \mid H \xrightarrow{\texttt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2 \mid H}}{D[M] \mid H \xrightarrow{*.\texttt{enter}\_n} O_1 \equiv (\boldsymbol{\nu}\tilde{r})(n[\circ \mid k[M_1]] \mid M_2 \mid H)}$$

where $k \in \tilde{r}$, for some process M_1 and system M_2. Remark that $n \notin \tilde{r}$. As $D[M] \xrightarrow{\texttt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2$ then $D[M] \xrightarrow{*.\texttt{enter}\_n} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid k[M_1]] \mid M_2) = M'$. The induction hypothesis then tells us that there exist a system N' such that $D[N] \mid n[\,\circ\,] \Rightarrow N'$, and for all processes P it holds $M' \bullet P \mathcal{S} N' \bullet P$. We can derive $D[N] \mid n[\circ] \mid H \Rightarrow N' \mid H$. Call $N' \mid H = O_2$. We can conclude that for all processes P, it holds $O_1 \bullet P \mathcal{S} O_2 \bullet P$ up to structural congruence, because $\mathcal{S}$ is closed under parallel composition.

∗ Suppose:

$$\dfrac{\dfrac{H \xrightarrow{\texttt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2}{D[M] \mid H \xrightarrow{\texttt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2 \mid D[M]}}{D[M] \mid H \xrightarrow{*.\texttt{enter}\_n} O_1 \equiv (\boldsymbol{\nu}\tilde{r})(n[\circ \mid k[H_1]] \mid H_2 \mid D[M])}$$

where $k \in \tilde{r}$ for some process H_1 and system H_2. We can construct the following derivation:

$$\dfrac{\dfrac{H \xrightarrow{\texttt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2}{D[N] \mid H \xrightarrow{\texttt{enter}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2 \mid D[N]} \quad n[\,\circ\,] \xrightarrow{\texttt{amb}\_n} \langle\,\circ\,\rangle \mathbf{0}}{D[N] \mid H \mid n[\,\circ\,] \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid k[H_1]] \mid H_2 \mid D[N]) = O_2}$$

We can conclude that for all processes P, it holds $O_1 \bullet P \mathcal{S} O_2 \bullet P$ up to structural congruence, because $D[M] \mathcal{S} D[N]$ and $\mathcal{S}$ is closed under parallel composition.

– $D[M] \mid H \xrightarrow{*.\texttt{exit}\_n} O_1$.

There are two possible derivations.

∗ Suppose:

$$\dfrac{\dfrac{D[M] \xrightarrow{\texttt{exit}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2}{D[M] \mid H \xrightarrow{\texttt{exit}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2 \mid H}}{D[M] \mid H \xrightarrow{*.\texttt{exit}\_n} O_1 \equiv (\boldsymbol{\nu}\tilde{r})(n[\circ \mid M_2 \mid H] \mid k[M_1])}$$

for some process M_1 and system M_2. Remark that $k \in \tilde{r}$. As $D[M] \xrightarrow{\texttt{exit}\_n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2$ then $D[M] \xrightarrow{*.\texttt{exit}\_n} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid M_2] \mid k[M_1]) = M'$. The induction hypothesis then tells us that there exist systems N' such that $n[\circ \mid$

$D[N]] \Rightarrow N'$, and for all processes P it holds $M' \bullet P \; \mathcal{S} \; N' \bullet P$. Remark that $N' \equiv (\boldsymbol{\nu}\tilde{s})n[\circ \mid N_3] \mid N_4$, for some N_3, N_4. We can derive $n[\circ D[N] \mid H] \Rightarrow (\boldsymbol{\nu}\tilde{s})n[\circ \mid N_3 \mid H] \mid N_4$. Call $(\boldsymbol{\nu}\tilde{s})n[\circ \mid N_3 \mid H] \mid N_4 = O_2$. As for all processes P it holds $M' \bullet P \; \mathcal{S} \; N' \bullet P$, we can conclude that for all processes Q, it holds $O_1 \bullet Q \; \mathcal{S} \; O_2 \bullet Q$ up to structural congruence, because $O_1 \bullet Q \equiv M' \bullet (Q \mid H) \; \mathcal{S} \; N' \bullet (Q \mid H) \equiv O_2 \bullet Q$.

∗ Suppose:

$$\dfrac{\dfrac{H \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2}{D[M] \mid H \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2 \mid D[M]}}{D[M] \mid H \xrightarrow{*.\texttt{exit\_}n} O_1 \;\equiv\; (\boldsymbol{\nu}\tilde{r})(n[\circ \mid H_2 \mid D[M]] \mid k[H_1])}$$

for some process H_1 and system H_2. Remark that $k \in \tilde{r}$. We can construct the following derivation:

$$\dfrac{\dfrac{H \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2}{D[N] \mid H \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[H_1]\rangle H_2 \mid D[N]}}{n[\circ \mid D[N] \mid H] \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{r})(n[\circ \mid H_2 \mid D[N]] \mid k[H_1]) = O_2}$$

We can conclude that for all processes P, it holds $O_1 \bullet P \; \mathcal{S} \; O_2 \bullet P$ up to structural congruence, because $D[M] \; \mathcal{S} \; D[N]$ and $\mathcal{S}$ is closed under parallel composition and ambient.

Then, we consider the cases when there is interaction between $D[M]$ and H.

– $D[M] \mid H \xrightarrow{\tau} O_1$, because

$$D[M] \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{m})\langle k[M_1]\rangle M_2 \text{ and } H \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{h})\langle H_1\rangle H_2.$$

Then $O_1 \equiv (\boldsymbol{\nu}\tilde{h}, \tilde{m})(n[k[M_1] \mid H_1] \mid M_2 \mid H_2)$. We distinguish the cases $k \in \tilde{m}$, and $k \notin \tilde{m}$.

∗ $k \notin \tilde{m}$. As $D[M] \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{m})\langle k[M_1]\rangle M_2$, it also holds $D[M] \xrightarrow{k.\texttt{enter\_}n} M' \equiv (\boldsymbol{\nu}\tilde{m})(n[\circ \mid k[M_1]] \mid M_2)$. The induction hypothesis tells us that there exists a system N' such that $D[N] \xRightarrow{k.\texttt{enter\_}n} N' \equiv (\boldsymbol{\nu}\tilde{m})(n[\circ \mid k[N_1]] \mid N_2)$, and for all processes P, it holds $M' \bullet P \; \mathcal{S} \; N' \bullet P$. But if $D[N] \xRightarrow{k.\texttt{enter\_}n} N'$, then $D[N] \xRightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{m})\langle k[N_1]\rangle N_2$. This implies that $D[N] \mid H \xRightarrow{\tau} O_2 \equiv (\boldsymbol{\nu}\tilde{h}, \tilde{n})(n[k[N_1] \mid H_1] \mid N_2 \mid H_2)$. Since for all processes P, $M' \bullet P \; \mathcal{S} \; N' \bullet P$, it also holds $M' \bullet H_1 \; \mathcal{S} \; N' \bullet H_1$, and $O_1 \; \mathcal{S} \; O_2$ follows because $\mathcal{S}$ is closed under parallel composition and restriction.

∗ $k \in \tilde{m}$. As $D[M] \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{m})\langle k[M_1]\rangle M_2$, it also holds $D[M] \xrightarrow{\texttt{enter\_}n} M' \equiv (\boldsymbol{\nu}\tilde{m})(n[\circ \mid k[M_1]] \mid M_2)$. The induction hypothesis tells us that there

exists a system N' such that $D[N] \mid n[\,\circ\,] \Rightarrow N' \equiv (\boldsymbol{\nu}\tilde{n})(n[\circ \mid N_1] \mid N_2)$, and for all processes P, it holds $M' \bullet P \;\mathcal{S}\; N' \bullet P$. We can derive $D[N] \mid H \Rightarrow O_2 \equiv (\boldsymbol{\nu}\tilde{h}, \tilde{n})(n[N_1 \mid H_1] \mid N_2 \mid H_2)$. Since for all processes P, $M' \bullet P \;\mathcal{S}\; N' \bullet P$, it also holds $M' \bullet H_1 \;\mathcal{S}\; N' \bullet H_1$, and $O_1 \;\mathcal{S}\; O_2$ follows because $\mathcal{S}$ is closed under parallel composition and restriction.

– $D[M] \mid H \xrightarrow{\tau} O_1$, because

$$D[M] \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{m})\langle M_1\rangle M_2 \text{ and } H \xrightarrow{\texttt{enter\_}n} (\boldsymbol{\nu}\tilde{h})\langle k[H_1]\rangle H_2.$$

Then $O_1 \equiv (\boldsymbol{\nu}\tilde{h}, \tilde{m})(n[k[H_1] \mid M_1] \mid M_2 \mid H_2)$. As $D[M] \xrightarrow{\texttt{amb\_}n} (\boldsymbol{\nu}\tilde{m})\langle M_1\rangle M_2$, it also holds $D[M] \xrightarrow{n.\overline{\texttt{enter}}\_k} M' \equiv (\boldsymbol{\nu}\tilde{m})(n[k[\,\circ\,] \mid M_1] \mid M_2)$. The induction hypothesis tells us that there exists a system N' such that $D[N] \stackrel{n.\overline{\texttt{enter}}\_k}{\Longrightarrow} N' \equiv (\boldsymbol{\nu}\tilde{n})(n[k[\,\circ\,] \mid N_1] \mid N_2)$, and for all processes P, it holds $M' \bullet P \;\mathcal{S}\; N' \bullet P$. As $D[N] \stackrel{n.\overline{\texttt{enter}}\_k}{\Longrightarrow} N'$, we can derive $D[N] \stackrel{\texttt{amb}\_k}{\Longrightarrow} (\boldsymbol{\nu}\tilde{n})\langle N_1\rangle N_2$. It follows $D[N] \mid H \Rightarrow (\boldsymbol{\nu}\tilde{h}, \tilde{n})(n[k[H_1] \mid N_1] \mid N_2 \mid H_2) = O_2$. Since for all processes P, it holds $M' \bullet P \;\mathcal{S}\; N' \bullet P$, we have $M' \bullet h[H_1] \;\mathcal{S}\; N' \bullet h[H_1]$, and $O_1 \;\mathcal{S}\; O_2$ follows because $\mathcal{S}$ is closed under parallel composition and restriction.

• $C[-] = n[D[-] \mid P]$, where P is an arbitrary process.

We know that $D[M] \;\mathcal{S}\; D[N]$ satisfies the bisimulation conditions in $\mathcal{S}$, and we want to prove that $n[D[M] \mid P] \;\mathcal{S}\; n[D[N] \mid P]$ behaves as a bisimulation as well. We perform a case analysis on the transition $n[D[M] \mid P] \xrightarrow{\alpha} O_1$.

– $n[D[M] \mid P] \xrightarrow{\tau} O_1$, because $D[M] \xrightarrow{\tau} M'$. Then $O_1 \equiv n[M' \mid P]$. The induction hypothesis tells us that there exists a system N' such that $D[N] \Rightarrow N'$ and $M' \;\mathcal{S}\; N'$. We can derive $n[D[N] \mid P] \Rightarrow n[N' \mid P]$ and conclude $n[M' \mid P] \;\mathcal{S}\; n[N' \mid P]$ because $\mathcal{S}$ is closed under ambient.

– $n[D[M] \mid P] \xrightarrow{\tau} O_1$, because $P \xrightarrow{\tau} P'$. Then $O_1 \equiv n[D[M] \mid P']$. Call $O_2 = n[D[N] \mid P']$. Then $O_1 \;\mathcal{S}\; O_2$ because $D[M] \;\mathcal{S}\; D[N]$, and $\mathcal{S}$ is closed under the contexts of the form $C[-] = n[- \mid Q]$ where Q is a process.

– $n[D[M] \mid P] \xrightarrow{\tau} O_1$, because $D[M] \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2$. Then $O_1 \equiv (\boldsymbol{\nu}\tilde{r})(k[M_1] \mid n[M_2 \mid P])$. We distinguish the two cases $k \in \tilde{r}$ and $k \notin \tilde{r}$.

∗ $k \notin \tilde{r}$. From $D[M] \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2$ we can derive $D[M] \xrightarrow{k.\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})(k[M_1] \mid n[\circ \mid M_2])$. The induction hypothesis tells us that there exists a system N' such that $D[N] \stackrel{k.\texttt{exit\_}n}{\Longrightarrow} N' \equiv (\boldsymbol{\nu}\tilde{s})(k[N_1] \mid n[\circ \mid N_2])$ and for all processes Q, it holds $M' \bullet Q \;\mathcal{S}\; N' \bullet Q$. But $D[N] \stackrel{k.\texttt{exit\_}n}{\Longrightarrow} N'$ can only be derived from $D[N] \stackrel{\texttt{exit\_}n}{\Longrightarrow} (\boldsymbol{\nu}\tilde{s})\langle k[N_1]\rangle N_2$ and thus $n[D[N] \mid P] \Rightarrow N' \bullet P$. As for all processes Q, it holds $M' \bullet Q \;\mathcal{S}\; N' \bullet Q$, we can derive $(\boldsymbol{\nu}\tilde{r})(k[M_1] \mid n[P \mid M_2]) \;\mathcal{S}\; (\boldsymbol{\nu}\tilde{s})(k[N_1] \mid n[P \mid N_2])$, as required.

  - $k \in \tilde{r}$. From $D[M] \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[M_1]\rangle M_2$ we can derive $D[M] \xrightarrow{*.\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})(k[M_1] \mid n[\circ \mid M_2])$. The induction hypothesis tells us that there exists a system N' such that $n[\circ \mid D[N]] \Rightarrow N' \equiv (\boldsymbol{\nu}\tilde{s})(k[N_1] \mid n[\circ \mid N_2])$, and for all processes Q, it holds $M' \bullet Q \mathcal{S} N' \bullet Q$. We can instantiate the placeholder $\circ$ with the process P, thus obtaining the transition $n[D[N] \mid P] \Rightarrow N' \bullet P$. As for all processes Q, it holds $M' \bullet Q \mathcal{S} N' \bullet Q$, we have $O_1 = (\boldsymbol{\nu}\tilde{m})(k[M_1] \mid n[P \mid M_2]) \equiv M' \bullet P \mathcal{S} N' \bullet P \equiv (\boldsymbol{\nu}\tilde{s})(k[N_1] \mid n[P \mid N_2]) = O_2$, as required.

- $n[D[M] \mid P] \xrightarrow{\tau} O_1$, because $P \xrightarrow{\texttt{exit\_}n} (\boldsymbol{\nu}\tilde{r})\langle k[P_1]\rangle P_2$. This implies $O_1 \equiv (\boldsymbol{\nu}\tilde{r})(k[P_1] \mid n[D[M] \mid P_2])$. It also holds $n[D[N] \mid P] \xrightarrow{\tau} \equiv (\boldsymbol{\nu}\tilde{r})(k[P_1] \mid n[D[N] \mid P_2])$. Call this last term O_2. The relation $O_1 \mathcal{S} O_2$ follows because $D[M] \mathcal{S} D[N]$ and from the closure properties of $\mathcal{S}$.

- $n[D[M] \mid P] \xrightarrow{\tau} O_1$, and the τ action is generated by an interaction between $D[M]$ and P. There are three cases.

  - $D[M] \xrightarrow{\texttt{amb\_}m} (\boldsymbol{\nu}\tilde{r})\langle M_1\rangle M_2$ and $P \xrightarrow{\texttt{open\_}m} P'$. Then $O_1 = n[(\boldsymbol{\nu}\tilde{r})(M_1 \mid M_2) \mid P']$. It holds $D[M] \xrightarrow{n.\texttt{open\_}m} n[\circ \mid (\boldsymbol{\nu}\tilde{r})(M_1 \mid M_2)]$. The induction hypothesis tells us that there exists a system N' such that $D[N] \xrightarrow{n.\texttt{open\_}m} N'$, and for all processes Q it holds $M' \bullet Q \mathcal{S} N' \bullet Q$. The system N' must be of the form $n[\circ \mid (\boldsymbol{\nu}\tilde{s})(N_1 \mid N_2)]$. The transition $D[N] \xRightarrow{n.\texttt{open\_}m} N'$ must have been derived from $D[N] \xrightarrow{\texttt{amb\_}m} (\boldsymbol{\nu}\tilde{s})\langle N_1\rangle N_2$. This implies that $n[D[N] \mid P] \Rightarrow n[(\boldsymbol{\nu}\tilde{s})(N_1 \mid N_2) \mid P']$. Call this last term O_2. We can instantiate the placeholder $\circ$ with the process P', thus obtaining the transition $n[D[N] \mid P] \Rightarrow N' \bullet P$. As for all processes Q, it holds $M' \bullet Q \mathcal{S} N' \bullet Q$, we have $O_1 = n[(\boldsymbol{\nu}\tilde{r})(M_1 \mid M_2) \mid P'] \equiv M' \bullet P' \mathcal{S} N' \bullet P' \equiv n[(\boldsymbol{\nu}\tilde{s})(N_1 \mid N_2) \mid P'] = O_2$, as required.
  - $D[M] \xrightarrow{\texttt{enter\_}m}$ and $P \xrightarrow{\texttt{amb\_}m}$, or $D[M] \xrightarrow{\texttt{amb\_}m}$ and $P \xrightarrow{\texttt{enter\_}m}$. Call A_1 the outcome of the interaction between $D[M]$ and P. In both cases, by an analysis carried on previously, we know that there is a process A_2 such that $D[N] \mid P \Rightarrow A_2$, with $A_1 \mathcal{S} A_2$. We obtain $n[D[M] \mid P] \xrightarrow{\tau} n[A_1] = O_1$, and $n[D[N] \mid P] \Rightarrow n[A_2]$. The relation $n[A_1] \mathcal{S} n[A_2]$ follows from the closure of $\mathcal{S}$ under ambient.

- $n[D[M] \mid P] \xrightarrow{n.\overline{\texttt{enter}}\_k} O_1$. Then $O_1 \equiv n[k[\circ] \mid D[M] \mid P]$. But $n[D[N] \mid P] \xrightarrow{n.\overline{\texttt{enter}}\_k} O_2$, where $O_2 \equiv n[k[\circ] \mid D[N] \mid P]$. For all processes Q, $O_1 \bullet Q \mathcal{S} O_2 \bullet Q$ follows from $D[M] \mathcal{S} D[N]$ because of the closure properties of $\mathcal{S}$.

- $n[D[M] \mid P] \xrightarrow{n.\texttt{exit\_}m} m[\circ] \mid n[D[M] \mid P'] = O_1$, because $P \xrightarrow{\texttt{out\_}m} P'$. It also holds $n[D[N] \mid P] \xrightarrow{n.\texttt{exit\_}m} m[\circ] \mid n[D[N] \mid P']$. Call this last term O_2. Then, for all processes Q, the relation $O_1 \bullet Q \mathcal{S} O_2 \bullet Q$ follows from $D[M] \mathcal{S} D[N]$ because of the closure properties of $\mathcal{S}$.

□

Theorem 4.3.4 *Late bisimilarity is contextual.*

Proof Let $\mathcal{S}$ be the smallest binary relation between systems such that:

1. $\approx \subseteq \mathcal{S}$;
2. if $M \mathcal{S} N$, then $C[M] \mathcal{S} C[N]$ for all system contexts $C[-]$.

Remark that $\mathcal{S}$ is symmetric because of the symmetry of $\approx$. We prove that $\mathcal{S}$ is a late bisimilarity up to $\equiv$ by induction on the definition of $\mathcal{S}$.

- $M \mathcal{S} N$ because $M \approx N$.

  Immediate.

- $C[M] \mathcal{S} C[N]$ because $M \mathcal{S} N$.

  The induction hypothesis assures that $(M, N) \in \mathcal{S}$ is a pair satisfying the bisimulation conditions in $\mathcal{S}$. Lemma 4.3.3 assures that the pair $(C[M], C[N])$ satisfies the bisimulation conditions in $\mathcal{S}$.

□

It is easy to adapt Lemma 4.3.3 and the above proof to show that early bisimilarity is contextual.

Proposition 4.3.5 *Early bisimilarity is contextual.*

As stated in the following Lemma, there is a close relationship between the observation predicate $M \downarrow n$ and a particular action that M can emit.

Lemma 4.3.6

1. *If* $M \xrightarrow{n.\overline{\texttt{enter}}\_k} M'$ *then* $M \downarrow n$;
2. *if* $M \downarrow n$ *then there exists a system* M' *such that* $M \xrightarrow{n.\overline{\texttt{enter}}\_k} M'$, *for some* k.

We conclude that both late and early bisimilarity are contained in the reduction barbed congruence.

Theorem 4.3.7 (Soundness) *The following inclusions hold* $\approx \subseteq \approx_e \subseteq \cong$.

Proof The first inclusion holds by definition. The second one comes from Proposition 4.3.5 and the fact that early bisimilarity is reduction closed and barb-preserving. □

$$
\begin{aligned}
C_{k.\mathtt{enter}\_n}[-] &= n[\circ \mid \mathsf{done}[\mathtt{in}\_k.\mathtt{out}\_k.\mathtt{out}\_n]] \mid - \\
C_{k.\mathtt{exit}\_n}[-] &= (\boldsymbol{\nu} a)a[\mathtt{in}\_k.\mathtt{out}\_k.\mathsf{done}[\mathtt{out}\_a]] \mid n[\circ \mid -] \\
C_{n.\overline{\mathtt{enter}}\_k}[-] &= (\boldsymbol{\nu} a)a[\mathtt{in}\_n.k[\mathtt{out}\_a.(\circ \mid (\boldsymbol{\nu} b)b[\mathtt{out}\_k.\mathtt{out}\_n.\mathsf{done}[\mathtt{out}\_b]])]] \mid - \\
C_{k.\mathtt{open}\_n}[-] &= k[\circ \mid (\boldsymbol{\nu} a, b)(\mathtt{open}\_b.\mathtt{open}\_a.\mathsf{done}[\mathtt{out}\_k] \mid a[- \mid \mathtt{open}\_n.b[\mathtt{out}\_a]])]
\end{aligned}
$$

where a,b and done are fresh names.

Table 4.7: Contexts for visible actions

4.3.2 Completeness

We now prove that late and early bisimilarity are more than proof techniques. They actually characterise reduction barbed congruence. The main challenge here is to design the contexts capable to observe our visible actions. The definition of these contexts, $C_\alpha[-]$, for every visible action α, is given in Table 4.7. The special ambient name done is used as *fresh* barb to signal the consumption of actions.

To prove our characterisation result it suffices to show that reduction barbed congruence is contained in the late bisimilarity. Then, by Theorem 4.3.7 we can conclude that late, early, and reduction barbed congruence, they all coincide. The proof that reduction barbed congruence implies the late bisimilarity requires the correspondence between visible actions α and their corresponding contexts $C_\alpha[-]$.

The context for $n.\overline{\mathtt{enter}}\_k$ deserves some comments. The secret ambient b assures that if the ambient done arrives at top-level, then it is empty. This allows a uniform formulation of Lemma 4.3.11. The need for the secret ambient a is more elusive. It prevents that the ambient k is used to enter into the ambient n by a Trojan horse hidden in the system plugged into the hole. This could invalidate Lemma 4.3.11.

The following lemma says that the defining contexts are sound, that is, they can successfully mimic the execution of visible actions.

Lemma 4.3.8 *Let M be a system. Let $\alpha \in \{k.\mathtt{enter}\_n, k.\mathtt{exit}\_n, n.\overline{\mathtt{enter}}\_k, k.\mathtt{open}\_n\}$. For all processes P, if $M \xrightarrow{\alpha} M'$ then $C_\alpha[M] \bullet P \Rightarrow\cong M' \bullet P \mid \mathsf{done}[\,]$.*

Proof The proof is by case analysis on α.

Case $\alpha = k.\mathtt{enter}\_n$. Let P be a process. We know that $M \xrightarrow{k.\mathtt{enter}\_n} M'$. Then

$$M \equiv (\boldsymbol{\nu}\tilde{m})(k[\mathtt{in}\_n.M_1 \mid M_2] \mid M_3)$$

where $(\{n, k\} \cup \mathrm{fn}(P)) \cap \{\tilde{m}\} = \emptyset$, and

$$M' \equiv (\boldsymbol{\nu}\tilde{m})(n[\circ \mid k[M_1 \mid M_2]] \mid M_3).$$

Now,

$$\begin{array}{rl} & C_{k.\mathtt{enter}\_n}[M] \bullet P \\ \equiv & (\boldsymbol{\nu}\tilde{m})(n[P \mid \mathsf{done}[\mathtt{in}\_k.\mathtt{out}\_k.\mathtt{out}\_n]] \mid k[\mathtt{in}\_n.M_1 \mid M_2] \mid M_3) \\ \rightarrowtriangle & (\boldsymbol{\nu}\tilde{m})(n[P \mid \mathsf{done}[\mathtt{in}\_k.\mathtt{out}\_k.\mathtt{out}\_n] \mid k[M_1 \mid M_2]] \mid M_3) \\ \rightarrowtriangle & (\boldsymbol{\nu}\tilde{m})(n[P \mid k[M_1 \mid M_2 \mid \mathsf{done}[\mathtt{out}\_k.\mathtt{out}\_n]]] \mid M_3) \\ \rightarrowtriangle & (\boldsymbol{\nu}\tilde{m})(n[P \mid \mathsf{done}[\mathtt{out}\_n] \mid k[M_1 \mid M_2]] \mid M_3) \\ \rightarrowtriangle & (\boldsymbol{\nu}\tilde{m})(\mathsf{done}[\,] \mid n[P \mid k[M_1 \mid M_2]] \mid M_3) \\ \equiv & (\boldsymbol{\nu}\tilde{m})(n[\circ \mid k[M_1 \mid M_2] \mid M_3]) \bullet P \mid \mathsf{done}[\,] \\ = & M' \bullet P \mid \mathsf{done}[\,] \end{array}$$

This implies $C_{k.\mathtt{enter}\_n}[M] \bullet P \Rightarrow\!\cong M' \bullet P \mid \mathsf{done}[\,]$.

Case $\alpha = k.\mathtt{exit}\_n$**.** Let P be a process. We know that $M \xrightarrow{k.\mathtt{exit}\_n} M'$. Then

$$M \equiv (\boldsymbol{\nu}\tilde{m})(k[\mathtt{out}\_n.M_1 \mid M_2] \mid M_3)$$

where $(\{n,k\} \cup \mathrm{fn}(P)) \cap \{\tilde{m}\} = \emptyset$, and

$$M' \equiv (\boldsymbol{\nu}\tilde{m})(k[M_1 \mid M_2] \mid n[\circ \mid M_3]).$$

Now,

$$\begin{array}{rl} & C_{k.\mathtt{exit}\_n}M \bullet P \\ \equiv & (\boldsymbol{\nu}\tilde{m})((\boldsymbol{\nu}a)a[\mathtt{in}\_k.\mathtt{out}\_k.\mathsf{done}[\mathtt{out}\_a]] \mid n[P \mid k[\mathtt{out}\_n.M_1 \mid M_2] \mid M_3)] \\ \rightarrowtriangle & (\boldsymbol{\nu}\tilde{m})((\boldsymbol{\nu}a)a[\mathtt{in}\_k.\mathtt{out}\_k.\mathsf{done}[\mathtt{out}\_a]] \mid k[M_1 \mid M_2] \mid n[P \mid M_3]) \\ \rightarrowtriangle & (\boldsymbol{\nu}\tilde{m})((\boldsymbol{\nu}a)k[a[\mathtt{out}\_k.\mathsf{done}[\mathtt{out}\_a]] \mid M_1 \mid M_2] \mid n[P \mid M_3]) \\ \rightarrowtriangle & (\boldsymbol{\nu}\tilde{m})((\boldsymbol{\nu}a)a[\mathsf{done}[\mathtt{out}\_a]] \mid k[M_1 \mid M_2] \mid n[P \mid M_3]) \\ \rightarrowtriangle & (\boldsymbol{\nu}\tilde{m})((\boldsymbol{\nu}a)(\mathsf{done}[\,] \mid a[\,]) \mid k[M_1 \mid M_2] \mid n[P \mid M_3]) \\ \cong & (\boldsymbol{\nu}\tilde{m})(k[M_1 \mid M_2] \mid n[\circ \mid P_3]) \bullet P \mid \mathsf{done}[\,] \\ = & M' \bullet P \mid \mathsf{done}[\,] \end{array}$$

This implies $C_{k.\mathtt{exit}\_n}[M] \bullet P \Rightarrow\!\cong M' \bullet P \mid \mathsf{done}[\,]$.

Case $\alpha = n.\overline{\mathtt{enter}}\_k$**.** Let P be a process. We know that $M \xrightarrow{n.\overline{\mathtt{enter}}\_k} M'$. Then

$$M \equiv (\boldsymbol{\nu}\tilde{m})(n[M_1] \mid M_2)$$

where $(\{n,k\} \cup \mathrm{fn}(P)) \cap \{\tilde{m}\} = \emptyset$, and

$$M' \equiv (\boldsymbol{\nu}\tilde{m})(n[M_1 \mid k[\circ\,]] \mid M_2).$$

$$
\begin{array}{lcl}
-_1 \oplus -_2 & = & (\boldsymbol{\nu} o)(o[\,] \mid \mathtt{open}\_o.-_1 \mid \mathtt{open}\_o.-_2) \\
\mathrm{SPY}_\alpha\langle i, j, -\rangle & = & (i[\mathtt{out}\_n] \mid -) \oplus (j[\mathtt{out}\_n] \mid -) \\
 & & \quad \text{if } \alpha \in \{k.\mathtt{enter}\_n, k.\mathtt{exit}\_n, k.\mathtt{open}\_n, *.\mathtt{enter}\_n, *.\mathtt{exit}\_n\} \\
\mathrm{SPY}_\alpha\langle i, j, -\rangle & = & (i[\mathtt{out}\_k.\mathtt{out}\_n] \mid -) \oplus (j[\mathtt{out}\_k.\mathtt{out}\_n] \mid -) \text{ if } \alpha \in \{n.\overline{\mathtt{enter}}\_k\}
\end{array}
$$

Table 4.8: Auxiliary contexts and processes

Now,

$$
\begin{array}{ll}
 & C_{n.\overline{\mathtt{enter}}\_k}[M] \bullet P \\
\equiv & (\boldsymbol{\nu}\tilde{m})((\boldsymbol{\nu} a)a[\mathtt{in}\_n.k[\mathtt{out}\_a.(P \mid (\boldsymbol{\nu} b)b[\mathtt{out}\_k.\mathtt{out}\_n.\mathsf{done}[\mathtt{out}\_b]])]] \mid n[M_1] \mid M_2) \\
\rightarrowtriangle & (\boldsymbol{\nu}\tilde{m})(n[M_1 \mid (\boldsymbol{\nu} a)a[k[\mathtt{out}\_a.(P \mid (\boldsymbol{\nu} b)b[\mathtt{out}\_k.\mathtt{out}\_n.\mathsf{done}[\mathtt{out}\_b]])]]] \mid M_2) \\
\rightarrowtriangle & (\boldsymbol{\nu}\tilde{m})(n[M_1 \mid (\boldsymbol{\nu} a)a[\,] \mid k[P \mid (\boldsymbol{\nu} b)b[\mathtt{out}\_k.\mathtt{out}\_n.\mathsf{done}[\mathtt{out}\_b]]]] \mid M_2) \\
\rightarrowtriangle & (\boldsymbol{\nu}\tilde{m})(n[M_1 \mid (\boldsymbol{\nu} a)a[\,] \mid k[P] \mid (\boldsymbol{\nu} b)b[\mathtt{out}\_n.\mathsf{done}[\mathtt{out}\_b]]] \mid M_2) \\
\rightarrowtriangle & (\boldsymbol{\nu}\tilde{m})(n[M_1 \mid (\boldsymbol{\nu} a)a[\,] \mid k[P]] \mid (\boldsymbol{\nu} b)b[\mathsf{done}[\mathtt{out}\_b]] \mid M_2) \\
\rightarrowtriangle & (\boldsymbol{\nu}\tilde{m})(n[M_1 \mid (\boldsymbol{\nu} a)a[\,] \mid k[P]] \mid (\boldsymbol{\nu} b)b[\,] \mid \mathsf{done}[\,] \mid M_2) \\
\cong & (\boldsymbol{\nu}\tilde{m})(n[M_1 \mid k[\,\circ\,]] \mid M_2) \bullet P \mid \mathsf{done}[\,] \\
= & M' \bullet P \mid \mathsf{done}[\,]
\end{array}
$$

This implies $C_{n.\overline{\mathtt{enter}}\_k}[M] \bullet P \Rightarrow\!\!\cong M' \bullet P \mid \mathsf{done}[\,]$.

Case $\alpha = k.\mathtt{open}\_n$**.** Let P be a process. We know that $M \xrightarrow{k.\mathtt{open}\_n} M'$. Then $M \equiv (\boldsymbol{\nu}\tilde{m})(n[M_1] \mid M_2)$, where $n \in \{\tilde{m}\}$, and $M' \equiv k[\circ \mid (\boldsymbol{\nu}\tilde{m})(M_1 \mid M_2)]$. Names a and b are fresh for M. Now,

$$
\begin{array}{ll}
 & C_{k.\mathtt{open}\_n}[M] \bullet P \\
\equiv & k[P \mid (\boldsymbol{\nu} a, b)(\mathtt{open}\_b.\mathtt{open}\_a.\mathsf{done}[\mathtt{out}\_k] \mid \\
 & \qquad a[(\boldsymbol{\nu}\tilde{m})(n[M_1] \mid M_2) \mid \mathtt{open}\_n.b[\mathtt{out}\_a]])] \\
\rightarrowtriangle & k[P \mid (\boldsymbol{\nu} a, b)(\mathtt{open}\_b.\mathtt{open}\_a.\mathsf{done}[\mathtt{out}\_k] \mid a[(\boldsymbol{\nu}\tilde{m})(M_1 \mid M_2) \mid b[\mathtt{out}\_a]])] \\
\rightarrowtriangle & k[P \mid (\boldsymbol{\nu} a, b)(\mathtt{open}\_b.\mathtt{open}\_a.\mathsf{done}[\mathtt{out}\_k] \mid a[(\boldsymbol{\nu}\tilde{m})(M_1 \mid M_2)] \mid b[\,])] \\
\rightarrowtriangle & k[P \mid (\boldsymbol{\nu} a, b)(\mathtt{open}\_a.\mathsf{done}[\mathtt{out}\_k] \mid a[(\boldsymbol{\nu}\tilde{m})(M_1 \mid M_2)])] \\
\rightarrowtriangle & k[P \mid (\boldsymbol{\nu} a, b)(\mathsf{done}[\mathtt{out}\_k] \mid (\boldsymbol{\nu}\tilde{m})(M_1 \mid M_2))] \\
\rightarrowtriangle & k[P \mid (\boldsymbol{\nu}\tilde{m})(M_1 \mid M_2))]\mathsf{done}[] \\
\equiv & k[\circ \mid (\boldsymbol{\nu}\tilde{m})(M_1 \mid M_2)] \bullet P \mid \mathsf{done}[\,] \\
= & M' \bullet P \mid \mathsf{done}[\,]
\end{array}
$$

This implies $C_{k.\mathtt{open}\_n}[M] \bullet P \Rightarrow\!\!\cong M' \bullet P \mid \mathsf{done}[\,]$. □

To complete the correspondence proof between actions α and their contexts $C_\alpha[-]$, we have to prove the converse of Lemma 4.3.8, formalised in Lemma 4.3.11. Such result requires a few technical definitions given in Table 4.8.

The symbol $\oplus$ denotes a form of internal choice, whereas the context $\mathrm{SPY}_\alpha\langle i, j, -\rangle$ is a technical tool to guarantee that the process P provided by the environment does not perform any action. This is necessary when proving completeness to guarantee that the contribution of P is the same on both sides. The ability of $\mathrm{SPY}_\alpha\langle i, j, P\rangle$ to 'spy' on P stems from the fact that one of the two fresh barbs i and j is lost when P performs any action. The property of $\mathrm{SPY}_\alpha\langle i, j, -\rangle$ is captured in the following lemma. As a useful shorthand, we write $M \Downarrow i_1, \ldots, i_n$ for $M \Downarrow i_1$ *and* $\cdots$ *and* $M \Downarrow i_n$.

Lemma 4.3.9 *Let M be a system which possibly contains an occurrence of the special process $\circ$. If $M \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle \xrightarrow{\tau} O$ and $O \Downarrow i, j$, where i, j are fresh for P and M, then there exists a system M' such that:*

1. $O = M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle$;
2. $M \xrightarrow{\tau} M'$.

Proof For 1), the definition of $\bullet$ assures that there exists an arbitrary context $C[-]$ such that $C[\mathrm{SPY}_\alpha\langle i, j, P\rangle] = M \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle$, and names in P are not bound in $C[-]$. The construction of $\mathrm{SPY}_\alpha\langle i, j, P\rangle$ assures that if $C[\mathrm{SPY}_\alpha\langle i, j, P\rangle] \xrightarrow{\tau} Q$, then either there is an arbitrary context C' such that $Q = C'[\mathrm{SPY}_\alpha\langle i, j, P\rangle]$, or $Q = C[P']$ where $\mathrm{SPY}_\alpha\langle i, j, P\rangle \xrightarrow{\tau} P'$. But if $\mathrm{SPY}_\alpha\langle i, j, P\rangle \xrightarrow{\tau} P'$, then $P' \Downarrow i \not\Downarrow j$, or $P' \Downarrow j \not\Downarrow i$. As $O \Downarrow i, j$, O must be the outcome of the first reduction, and as such there exists an arbitrary context $C'[-]$ such that $O = C'[\mathrm{SPY}_\alpha\langle i, j, P\rangle]$. Let $M' = C'[\circ]$. As $C[\mathrm{SPY}_\alpha\langle i, j, P\rangle] \xrightarrow{\tau} C'[\mathrm{SPY}_\alpha\langle i, j, P\rangle]$, names in P cannot be bound in $C'[-]$. This implies $O = C'[\mathrm{SPY}_\alpha\langle i, j, P\rangle] = M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle$, as required for 1).

For 2), $M \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle = C[\mathrm{SPY}_\alpha\langle i, j, P\rangle] \xrightarrow{\tau} C'[\mathrm{SPY}_\alpha\langle i, j, P\rangle] = M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle$ implies $M = C[\circ] \xrightarrow{\tau} C'[\circ] = M'$, as required. □

We also need a simple result on arbitrary contexts.

Lemma 4.3.10 *Let $C[-]$ and $C'[-]$ be arbitrary contexts, P, P' processes, and r a fresh name for $C[-]$ and P, such that $C[r[P]] \xrightarrow{\tau} C'[r[P']]$. Then $C[\mathbf{0}] \xrightarrow{\tau} C'[\mathbf{0}]$.*

We can finally prove the correspondence between actions and contexts.

Lemma 4.3.11 *Let M be a system, let $\alpha \in \{k.\mathtt{enter}\_n, k.\mathtt{exit}\_n, n.\overline{\mathtt{enter}}\_k, k.\mathtt{open}\_n\}$, and let i, j be fresh names for M. For all processes P with $\{i, j\} \cap \mathrm{fn}(P) = \emptyset$, if $C_\alpha[M] \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle \Rightarrow\cong O \mid \mathsf{done}[\,]$ and $O \Downarrow i, j$, then there exists a system M' such that $O \cong M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle$ and $M \overset{\alpha}{\Longrightarrow} M'$.*

Proof The proof depends on the precise definition of the context. The main argument is that in the reduction

$$C_\alpha[M] \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle \Rightarrow\cong O \mid \mathsf{done}[\,]$$

the fresh ambient $\mathsf{done}[\,]$ can only be unleashed if M performs the action α, possibly preceded or followed by some internal actions. The fresh barbs i, j assure that the process P does not

take part in the reduction, and that the component $\mathrm{SPY}_\alpha\langle i,j,P\rangle$ is found intact after the reduction. The barbed congruence is used to garbage collect a possible $(\boldsymbol{\nu}a)a[\,]$ ambient. We proceed by case analysis on α.

Case $\alpha = k.\mathtt{enter}\_n$**.** Observe that

$$C_\alpha[M] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle = n[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid \mathsf{done}[\mathtt{in}\_k.\mathtt{out}\_k.\mathtt{out}\_n]] \mid M\ .$$

As $O \Downarrow i,j$ and done is fresh, by several applications of Lemma 4.3.9, there must be a system context $D[-]$ such that $O \mid \mathsf{done}[\,] \equiv D[\mathsf{done}[\,]] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle$ and $C_\alpha[M] \Rightarrow D[\mathsf{done}[\,]]$. As P cannot reduce and done is fresh, the ambient n does not migrate during the reduction. Moreover, as M is a system, the ambient n cannot be opened. Also observe that the ambient done must consume the prefix $\mathtt{in}\_k$, thus requiring the presence of an ambient k inside the ambient n during the reduction. More precisely, there exist systems M_1 and M_2 and a static context $C[-]$ such that:

$$\begin{array}{rl}
 & C_\alpha[M] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \\
= & n[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid \mathsf{done}[\mathtt{in}_k.\mathtt{out}_k.\mathtt{out}_n]] \mid M \\
\Rightarrow \xrightarrow{\tau} & (\boldsymbol{\nu}\tilde{m})(n[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid \mathsf{done}[\mathtt{in}_k.\mathtt{out}_k.\mathtt{out}_n] \mid M_1] \mid M_2) \\
\xrightarrow{\tau} & (\boldsymbol{\nu}\tilde{m})(n[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid C[\mathsf{done}[\mathtt{out}_k.\mathtt{out}_n]]] \mid M_2) \\
\Rightarrow & D[\mathsf{done}[\,]] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \\
\equiv & D[\mathbf{0}] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \mid \mathsf{done}[\,] \\
\equiv & O \mid \mathsf{done}[\,]
\end{array}$$

Examining the above reductions sequence from $C_\alpha[M] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle$ we conclude that

$$M \Rightarrow \xrightarrow{k.\mathtt{enter}\_n} (\boldsymbol{\nu}\tilde{m})(n[\circ \mid M_1] \mid M_2).$$

As the name done is fresh for M, by several applications of Lemma 4.3.10, we also have that

$$(\boldsymbol{\nu}\tilde{m})(n[\circ \mid \mathbf{0} \mid M_1] \mid M_2) \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \Rightarrow D[\mathbf{0}] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle.$$

Repeated application of Lemma 4.3.9(2) gives $(\boldsymbol{\nu}\tilde{m})(n[\circ \mid \mathbf{0} \mid M_1] \mid M_2) \Rightarrow D[\mathbf{0}]$, and therefore, as $\equiv$ is closed under reduction, there is a M', $M' \equiv D[\mathbf{0}]$, such that $M \xRightarrow{k.\mathtt{enter}\_n} M'$, as desired.

Case $\alpha = k.\mathtt{exit}\_n$**.** Observe that

$$C_{k.\mathtt{exit}\_n}[M] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle = (\boldsymbol{\nu}a)a[\mathtt{in}\_k.\mathtt{out}\_k.\mathsf{done}[\mathtt{out}\_a]] \mid n[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid M]\ .$$

To unleash the ambient done, the ambient a must perform both its capabilities, and as its name is restricted the ambient a will be empty at the end of reduction. As P cannot reduce, and M is a system, the ambient n does not migrate during the reduction. Also observe that the ambient a must consume the prefix $\mathtt{in}\_k$, thus requiring the presence of an ambient k at

top-level. More precisely, there exist a system M_1 and a static contexts $D[-]$ and $E[-]$ such that:

$$\begin{array}{ll} & C_{k.\mathtt{exit}\_n}[M] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \\ = & (\boldsymbol{\nu} a)a[\mathtt{in}\_k.\mathtt{out}\_k.\mathsf{done}[\mathtt{out}\_a]] \mid n[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid M] \\ \Rightarrow & (\boldsymbol{\nu} a)a[\mathtt{in}\_k.\mathtt{out}\_k.\mathsf{done}[\mathtt{out}\_a]] \mid M_1 \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \\ \xrightarrow{\tau} & (\boldsymbol{\nu} a)D[a[\mathtt{out}\_k.\mathsf{done}[\mathtt{out}\_a]]] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \\ \Rightarrow & (\boldsymbol{\nu} a)E[\mathsf{done}[\,] \mid a[\,]] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \qquad (\star) \\ \cong & E[\mathbf{0}] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \mid \mathsf{done}[\,] \\ \equiv & O \mid \mathsf{done}[\,] \end{array}$$

Examining the above reductions sequence from $C_{k.\mathtt{exit}\_n}[M] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle$ we conclude that

$$M \Rightarrow \xrightarrow{k.\mathtt{exit}\_n} M_1.$$

As the name done is fresh for M, by several applications of Lemma 4.3.10 to the reduction marked by $(\star)$ we have:

$$(\boldsymbol{\nu} a)a[\mathtt{in}\_k.\mathtt{out}\_k.\mathbf{0}] \mid M_1 \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \Rightarrow (\boldsymbol{\nu} a)E[\mathbf{0} \mid a[\,]] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \ .$$

Again, as a is fresh, by several applications of Lemma 4.3.10, and reducing underneath $(\boldsymbol{\nu} a)$, we obtain:

$$(\boldsymbol{\nu} a)(\mathbf{0} \mid M_1) \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \Rightarrow (\boldsymbol{\nu} a)E[\mathbf{0} \mid \mathbf{0}] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \ .$$

Summarising,

$$M_1 \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \equiv (\boldsymbol{\nu} a)(\mathbf{0} \mid M_1) \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \Rightarrow (\boldsymbol{\nu} a)E[\mathbf{0} \mid \mathbf{0}] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle$$

and, as $\equiv$ is closed under reductions,

$$M_1 \Rightarrow\!\equiv E[\mathbf{0}] \ .$$

So, assuming $M' = E[\mathbf{0}]$, we can conclude.

Case $\alpha = n.\overline{\mathtt{enter}}\_k$**.** Observe that

$$\begin{array}{l} C_\alpha[M] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle = \\ \quad (\boldsymbol{\nu} a)a[\mathtt{in}\_n.k[\mathtt{out}\_a.(\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} b)b[\mathtt{out}\_k.\mathtt{out}\_n.\mathsf{done}[\mathtt{out}\_b]])]] \mid M \end{array}$$

To unleash the ambient done, the ambient a must use its $\mathtt{in}\_n$ capability, and the ambient b must exit from a. Moreover the ambient b must unleash all its capabilities. This implies that at the end of the reduction both secret ambients a and b will be empty. Also observe that the prefix $\mathtt{in}\_n$ must be consumed, thus requiring the presence of an ambient n at top-level.

More precisely, there exist a system M_1 and static contexts $D[-]$ and $E[-]$ such that

$$\begin{array}{lll} & C_{n.\overline{\texttt{enter}}\_k}[M] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle & \\ = & (\boldsymbol{\nu} a)a[\texttt{in}\_n.(k[\texttt{out}\_a.(\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} b)b[\texttt{out}\_k.\texttt{out}\_n.\mathsf{done}[\texttt{out}\_b]])])] \mid M & \\ \Rightarrow & (\boldsymbol{\nu} a)a[\texttt{in}\_n.(k[\texttt{out}\_a.(\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} b)b[\texttt{out}\_k.\texttt{out}\_n.\mathsf{done}[\texttt{out}\_b]])])] \mid M_1 & \\ \xrightarrow{\tau} & D[(\boldsymbol{\nu} a)a[k[\texttt{out}\_a.(\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} b)b[\texttt{out}\_k.\texttt{out}\_n.\mathsf{done}[\texttt{out}\_b]])]] & \\ \Rightarrow & E[\mathsf{done}[] \mid (\boldsymbol{\nu} b)b[\,]] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \quad (\star) & \\ \cong & E[\mathsf{done}[]] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle & \\ \cong & E[\mathbf{0}] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \mid \mathsf{done}[] & \\ = & O \mid \mathsf{done}[\,] & \end{array}$$

Observe that,

$$D[(\boldsymbol{\nu} a)a[k[\texttt{out}\_a.(\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} b)b[\texttt{out}\_k.\texttt{out}\_n.\mathsf{done}[\texttt{out}\_b]])]] \cong D[k[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} b)b[\texttt{out}\_k.\texttt{out}\_n.\mathsf{done}[\texttt{out}\_b]]]]$$

Thus, by examining the above reductions sequence from $C_{n.\overline{\texttt{enter}}\_k}[M] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle$ we conclude that

$$M \Rightarrow \xrightarrow{n.\overline{\texttt{enter}}\_k} \cong D[k[\,\circ\,]].$$

As the name done is fresh, several applications of Lemma 4.3.10 to the above reduction marked by $(\star)$ gives:

$$D[(\boldsymbol{\nu} a)a[k[\texttt{out}\_a.(\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} b)b[\texttt{out}\_k.\texttt{out}\_n.\mathbf{0}])]]] \Rightarrow E[\mathbf{0} \mid (\boldsymbol{\nu} b)b[\,]] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle\ .$$

Again, as b is fresh, by several applications of Lemma 4.3.10 and reducing underneath $(\boldsymbol{\nu} b)$, we have:

$$D[(\boldsymbol{\nu} a)a[k[\texttt{out}\_a.(\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid \mathbf{0})]]] \Rightarrow E[\mathbf{0} \mid \mathbf{0}] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle\ .$$

Summarising,

$$D[k[\mathrm{SPY}_\alpha\langle i,j,P\rangle]] \cong D[(\boldsymbol{\nu} a)a[k[\texttt{out}\_a.(\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid \mathbf{0})]]] \Rightarrow\cong E[\mathbf{0}] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle.$$

and, as $\cong$ is closed under reduction, this implies:

$$D[k[\,\circ\,]] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle = D[k[\mathrm{SPY}_\alpha\langle i,j,P\rangle]] \Rightarrow\equiv E[\mathbf{0}] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle.$$

Now, by applying Lemma 4.3.9 there must be a system $\hat{M}$ such that: $D[k[\,\circ\,]] \Rightarrow \hat{M}$ and $\hat{M} \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \cong E[\mathbf{0}] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle$. Finally, as

$$M \Rightarrow \xrightarrow{n.\overline{\texttt{enter}}\_k} \cong D[k[\,\circ\,]]$$

and $\cong$ is reduction closed, there must exist M' such that $M \Rightarrow M'$ and

$$M' \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \cong E[\mathbf{0}] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \cong O$$

as required.

Case $\alpha = k.\mathtt{open\_}n$**.** Observe that

$$C_{k.\mathtt{open}\_n}[M] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle = \\ k[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} a,b)(\mathtt{open\_}b.\mathtt{open\_}a.\mathsf{done}[\mathtt{out\_}k] \mid a[M \mid \mathtt{open\_}n.b[\mathtt{out\_}a]])]$$

where a and b are fresh. To unleash the ambient done, the ambient a must use its $\mathtt{open\_}n$ capability, and the ambient b must exit from a. Moreover both the empty ambients a and b will be opened before done is activated. Also observe that the prefix $\mathtt{open\_}n$ must be consumed, thus requiring the presence of an ambient n inside the ambient a. More precisely, there exist a system M_1, processes Q_i, and a static context $D[-]$ such that:

$$\begin{array}{rl}
 & C_{k.\mathtt{open}_n}[M] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \\
= & k[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} a,b)(\mathtt{open_}b.\mathtt{open_}a.\mathsf{done}[\mathtt{out_}k] \mid a[M \mid \mathtt{open_}n.b[\mathtt{out_}a]])] \\
\Rightarrow & k[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} a,b)(\mathtt{open_}b.\mathtt{open_}a.\mathsf{done}[\mathtt{out_}k] \mid a[M_1 \mid \mathtt{open_}n.b[\mathtt{out_}a]])] \\
\xrightarrow{\tau} & k[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} a,b)(\mathtt{open_}b.\mathtt{open_}a.\mathsf{done}[\mathtt{out_}k] \mid a[Q \mid b[\mathtt{out_}a]])] \\
\Rightarrow & k[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} a,b)(\mathtt{open_}b.\mathtt{open_}a.\mathsf{done}[\mathtt{out_}k] \mid a[Q_1 \mid b[\mathtt{out_}a]])] \\
\xrightarrow{\tau} & k[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} a,b)(\mathtt{open_}b.\mathtt{open_}a.\mathsf{done}[\mathtt{out_}k] \mid b[\,] \mid a[Q_1])] \\
\Rightarrow & k[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} a,b)(\mathtt{open_}b.\mathtt{open_}a.\mathsf{done}[\mathtt{out_}k] \mid b[\,] \mid a[Q_2])] \\
\xrightarrow{\tau} & k[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} a,b)(\mathtt{open_}a.\mathsf{done}[\mathtt{out_}k] \mid \mathbf{0} \mid a[Q_2])] \\
\Rightarrow & k[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} a,b)(\mathtt{open_}a.\mathsf{done}[\mathtt{out_}k] \mid \mathbf{0} \mid a[Q_3])] \\
\Rightarrow & k[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} a,b)(\mathsf{done}[\mathtt{out_}k] \mid \mathbf{0} \mid Q_3)] \\
\Rightarrow & D[\mathsf{done}[\,]] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \\
\equiv & D[\mathbf{0}] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle \mid \mathsf{done}[\,] \\
= & O \mid \mathsf{done}[\,]
\end{array}$$

Examining the above reductions sequence from $C_{k.\mathtt{open}\_n}[M] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle$ we conclude that

$$M \Rightarrow \xrightarrow{k.\mathtt{open}\_n} k[\circ \mid Q].$$

As

$$k[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} a,b)(\mathtt{open\_}b.\mathtt{open\_}a.\mathsf{done}[\mathtt{out\_}k] \mid a[Q \mid b[\mathtt{out\_}a]])] \\ \Rightarrow D[\mathsf{done}[\,]] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle$$

and the name done is fresh, by several applications of Lemma 4.3.10 we have

$$k[\mathrm{SPY}_\alpha\langle i,j,P\rangle \mid (\boldsymbol{\nu} a,b)(\mathtt{open\_}b.\mathtt{open\_}a.\mathbf{0} \mid a[Q \mid b[\mathtt{out\_}a]])] \\ \Rightarrow D[\mathbf{0}] \bullet \mathrm{SPY}_\alpha\langle i,j,P\rangle.$$

By Lemma 4.3.9, this implies

$$k[\circ \mid (\boldsymbol{\nu} a,b)(\mathtt{open\_}b.\mathtt{open\_}a.\mathbf{0} \mid a[Q \mid b[\mathtt{out\_}a]])] \Rightarrow D[\mathbf{0}].$$

Applying our proof techniques we can easily prove that:

$$k[\circ \mid (\boldsymbol{\nu} a,b)(\mathtt{open\_}b.\mathtt{open\_}a.\mathbf{0} \mid a[Q \mid b[\mathtt{out\_}a]])] \cong k[\circ \mid Q].$$

As $\cong$ is closed under reduction, it follows that there is M' such that

$$k[\circ \mid Q] \;\Rightarrow M' \cong D[\mathbf{0}].$$

So, there is M' such that $M \;\Rightarrow M'$ and $O \cong M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle$, as desired.

□

Theorem 4.3.12 (Completeness) *Reduction barbed congruence is contained in late bisimilarity.*

Proof We prove that the relation $\mathcal{R} = \{(M, N) \mid M \cong N\}$ is a late bisimulation. The result will then follow by co-induction.

- Suppose $M \;\mathcal{R}\; N$. Suppose also that $M \xrightarrow{\alpha} M'$, with $\alpha \in \{k.\mathtt{enter}\_n, k.\mathtt{exit}\_n, n.\overline{\mathtt{enter}}\_k, k.\mathtt{open}\_n\}$. We must find a system N' such that $N \stackrel{\alpha}{\Longrightarrow} N'$ and for all P, $M' \bullet P \cong N' \bullet P$.

  The idea of the proof is to use a particular context which mimics the effect of the action α, and also allows us to subsequently compare the residuals of the two systems. This context has the form

  $$D_\alpha\langle P\rangle[-] \;=\; (C_\alpha[-] \mid \mathsf{Flip}) \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle$$

  where $C_\alpha[-]$ are the contexts in Table 4.7 and Flip is the system:

  $$(\boldsymbol{\nu} k)k[\mathtt{in}\_\mathsf{done}.\mathtt{out}\_\mathsf{done}.(\mathsf{succ}[\mathtt{out}\_k] \oplus \mathsf{fail}[\mathtt{out}\_k])]$$

  with succ and fail are fresh names. Intuitively, the existence of the fresh barb fail indicates that the action α has not yet happened, whereas the presence of succ together with the absence of fail ensures that the action α has been performed, and has been reported via done.

  As $\cong$ is contextual, $M \cong N$ implies that, for all processes P, it holds

  $$D_\alpha\langle P\rangle[M] \;\cong\; D_\alpha\langle P\rangle[N]\ .$$

  By Lemma 4.3.8, and by inspecting the reduction of the Flip process, we observe that:

  $$\begin{array}{rll} D_\alpha\langle P\rangle[M] & \Rightarrow\cong & M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle \mid \mathsf{done}[\,] \mid \mathsf{Flip} \\ & \Rightarrow\cong & M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle \mid \mathsf{done}[\,] \mid \mathsf{succ}[\,] \end{array}$$

  where $M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle \mid \mathsf{done}[\,] \mid \mathsf{succ}[\,] \Downarrow i, j, \mathsf{succ} \not\Downarrow \mathsf{fail}$. Call this outcome O_1.

  This reduction must be matched by a corresponding reduction

  $$D_\alpha\langle P\rangle[N] \;\Rightarrow\; O_2$$

  where $O_1 \cong O_2$. However, the possible matching reductions are constrained by the barbs of O_1, because it must hold $O_2 \Downarrow i, j, \mathsf{succ} \not\Downarrow \mathsf{fail}$.

  As $O_2 \Downarrow \mathsf{succ} \not\Downarrow \mathsf{fail}$, it must be $O_2 \cong \hat{N} \mid \mathsf{done}[\,] \mid \mathsf{succ}[\,]$, for some systems $\hat{N}$.

As $O_2 \Downarrow i, j$, the previous observation can be combined with Lemma 4.3.11 to derive the existence of a system (over the extended process syntax) N' such that $\hat{N} \cong N' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle$ and a weak action

$$N \stackrel{\alpha}{\Longrightarrow} N'.$$

To conclude we must establish that for all P, it holds $M' \bullet P \;\cong\; N' \bullet P$. As barbed congruence is preserved by restriction, we have $(\boldsymbol{\nu}\mathsf{done}, \mathsf{succ})O_1 \cong (\boldsymbol{\nu}\mathsf{done}, \mathsf{succ})O_2$. As $(\boldsymbol{\nu}\mathsf{done})\mathsf{done}[\,] \cong (\boldsymbol{\nu}\mathsf{succ})\mathsf{succ}[\,] \cong \mathbf{0}$, it follows that

$$M' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle \;\cong\; N' \bullet \mathrm{SPY}_\alpha\langle i, j, P\rangle.$$

Again, $\cong$ is preserved by restriction and $(\boldsymbol{\nu} i, j)\mathrm{SPY}_\alpha\langle i, j, P\rangle \cong P$. So, we can finally derive $M' \bullet P \;\mathcal{R}\; N' \bullet P$, for all processes P.

- Suppose now $M \;\mathcal{R}\; N$ and $M \xrightarrow{*.\mathtt{enter}\_n} M'$, We must find a system N' such that $N \mid n[\,\circ\,] \;\Rightarrow N'$ and for all P, $M' \bullet P \cong N' \bullet P$.

  We consider the context

  $$C\langle P\rangle[-] \;=\; - \mid n[\mathrm{SPY}_{*.\mathtt{enter}\_n}\langle i, j, P\rangle] \;.$$

  Because $\cong$ is contextual, for all processes P it holds

  $$C\langle P\rangle[M] \;\cong\; C\langle P\rangle[N] \;.$$

  By inspecting the reduction rules of $C\langle P\rangle[M]$ we observe that,

  $$C\langle P\rangle[M] \;\Rightarrow\; M' \bullet \mathrm{SPY}_{*.\mathtt{enter}\_n}\langle i, j, P\rangle$$

  where $M' \bullet \mathrm{SPY}_{*.\mathtt{enter}\_n}\langle i, j, P\rangle \Downarrow i, j$. Call this outcome O_1.

  This reduction must be matched by a corresponding reduction

  $$C\langle P\rangle[N] \;\Rightarrow\; O_2$$

  where $O_1 \cong O_2$ and $O_2 \Downarrow i, j$. By several applications of Lemma 4.3.9 it follows that there is a system N' such that $O_2 = N' \bullet \mathrm{SPY}_{*.\mathtt{enter}\_n}\langle i, j, P\rangle$ and $N \mid n[\,\circ\,] \;\Rightarrow N'$. Again, as $\cong$ is preserved by restriction and $(\boldsymbol{\nu} i, j)\mathrm{SPY}_{*.\mathtt{enter}\_n}\langle i, j, P\rangle \cong P$, from $O_1 \cong O_2$ and the freshness of i and j we can derive $M' \bullet P \cong N' \bullet P$, for all P, as required.

- Suppose at last $M\mathcal{R}N$ and $M \xrightarrow{*.\mathtt{exit}\_n} M'$. In this case we must find a system N' such that $n[\circ \mid N] \;\Rightarrow N'$ and for all P, $M' \bullet P \cong N' \bullet P$.

  We consider the context

  $$C\langle P\rangle[-] \;=\; n[- \mid \mathrm{SPY}_{*.\mathtt{exit}\_n}\langle i, j, P\rangle] \;.$$

  Because $\cong$ is contextual, for all processes P it holds

  $$C\langle P\rangle[M] \;\cong\; C\langle P\rangle[N] \;.$$

By inspecting the reduction rules of $C\langle P\rangle[M]$ we observe that,

$$C\langle P\rangle[M] \;\Rightarrow\; M' \bullet \mathrm{SPY}_{*.\mathtt{exit}\_n}\langle i, j, P\rangle$$

where $M' \bullet \mathrm{SPY}_{*.\mathtt{exit}\_n}\langle i, j, P\rangle \Downarrow i, J$. Call this outcome O_1.

This reduction must be matched by a corresponding reduction

$$C\langle P\rangle[N] \;\Rightarrow\; O_2$$

where $O_1 \cong O_2$ and $O_2 \Downarrow A, B$. By several applications of Lemma 4.3.9 it follows that there is a system N' such that $O_2 = N' \bullet \mathrm{SPY}_{*.\mathtt{enter}\_n}\langle i, j, P\rangle$ and $n[\circ \mid N] \;\Rightarrow\; N'$. Again, as $\cong$ is preserved by restriction and $(\boldsymbol{\nu} i, j)\mathrm{SPY}_{*.\mathtt{exit}\_n}\langle i, j, P\rangle \cong P$, from $O_1 \cong O_2$ and the freshness of i and j we can derive $M' \bullet P \cong N' \bullet P$, for all P, as required.

This concludes the analysis. □

As a consequence:

Theorem 4.3.13 *Late bisimilarity, early bisimilarity, and reduction barbed congruence coincide.*

Proof Theorem 4.3.7 states that $\approx \subseteq \approx_e$ and $\approx_e \subseteq \cong$. Theorem 4.3.12 states the reduction barbed congruence is contained in late bisimilarity, that is $\cong \subseteq \approx$. We hence have the following chain of inclusions $\cong \subseteq \approx \subseteq \approx_e \subseteq \cong$. □

A remark on transitivity of bisimilarity. Giving a direct proof that bisimilarity is a transitive relation seems to be awkward. At the same time, none of the Lemmas and Theorems used to prove that bisimilarity coincides with reduction barbed congruence requires the transitivity of the bisimilarity. This implies that late bisimilarity and early bisimilarity are equivalence relations.

The LTS can be presented in 'very early' style as done in [MH01]. See [ZN03]. With this formulation, transitions report explicitly the process running inside the interacting ambient. The LTS in 'very early' style allows a straightforward proof of transitivity of bisimilarity. Again, the resulting labelled bisimilarity coincides with reduction barbed congruence, thus strengthening the conjecture that in higher-order process calculi early and late bisimilarity coincide.

4.4 Up-to proof techniques

In the previous section we gave a labelled characterisation of reduction barbed congruence to prove that two systems have the same behaviour. In this section we adapt some well-known *up-to* proof techniques [MS92, San98] to our setting. These techniques allow us to reduce the size of the relation $\mathcal{R}$ to exhibit to prove that two processes are bisimilar. We focus on the *up-to expansion* [SM92] and the *up-to context* techniques [San96b]. As in the π-calculus, these two techniques can also be merged.

The expansion [AKH92], written $\lesssim$, is an asymmetric variant of the bisimilarity that allows us to count the number of silent moves performed by a process. More precisely, $M \lesssim N$ holds if M and N are bisimilar and N has at least as many τ-moves as M. Formally,

Definition 4.4.1 (Expansion) *A relation $\mathcal{R}$ over systems is an* expansion *if $M\ \mathcal{R}\ N$ implies:*

- *if $M \xrightarrow{\alpha} M'$, $\alpha \notin \{*.\texttt{enter}\_n, *.\texttt{exit}\_n\}$, then there exists a system N' such that $N \stackrel{\hat{\alpha}}{\Longrightarrow} N'$ and for all processes P it holds $M' \bullet P\ \mathcal{R}\ N' \bullet P$;*
- *if $M \xrightarrow{*.\texttt{enter}\_n} M'$ then there exists a system N' such that $N \mid n[\,\circ\,] \Rightarrow N'$ and for all processes P it holds $M' \bullet P\ \mathcal{R}\ N' \bullet P$;*
- *if $M \xrightarrow{*.\texttt{exit}\_n} M'$ then there exists a system N' such that $n[\circ \mid N] \Rightarrow N'$ and for all processes P it holds $M' \bullet P\ \mathcal{R}\ N' \bullet P$;*
- *if $N \xrightarrow{\alpha} N'$, $\alpha \notin \{*.\texttt{enter}\_n, *.\texttt{exit}\_n\}$, then there exists a system M' such that $M \xrightarrow{\hat{\alpha}} M'$ and for all processes P it holds $M' \bullet P\ \mathcal{R}\ N' \bullet P$;*
- *if $N \xrightarrow{*.\texttt{enter}\_n} N'$ then $(M \mid n[P])\ \mathcal{R}\ N' \bullet P$, for all processes P;*
- *if $N \xrightarrow{*.\texttt{exit}\_n} N'$ then $n[M \mid P]\ \mathcal{R}\ N' \bullet P$, for all processes P.*

We write $M \lesssim N$, if $M\ \mathcal{R}\ N$ for some expansion $\mathcal{R}$.

Definition 4.4.2 (Bisimulation up to $\gtrsim$) *A symmetric relation $\mathcal{R}$ is a* bisimulation up to $\gtrsim$ *if $M\ \mathcal{R}\ N$ implies:*

- *if $M \xrightarrow{\alpha} M'$, $\alpha \notin \{*.\texttt{enter}\_n, *.\texttt{exit}\_n\}$, then there exists a system N' such that $N \stackrel{\hat{\alpha}}{\Longrightarrow} N'$ and for all processes P it holds $M' \bullet P \gtrsim \mathcal{R} \lesssim N' \bullet P$;*
- *if $M \xrightarrow{*.\texttt{enter}\_n} M'$ then there exists a system N' such that $N \mid n[\,\circ\,] \Rightarrow N'$ and for all processes P it holds $M' \bullet P \gtrsim \mathcal{R} \lesssim N' \bullet P$;*
- *if $M \xrightarrow{*.\texttt{exit}\_n} M'$ then there exists a system N' such that $n[\circ \mid N] \Rightarrow N'$ and for all processes P it holds $M' \bullet P \gtrsim \mathcal{R} \lesssim N' \bullet P$.*

Theorem 4.4.3 *If $\mathcal{R}$ is a bisimulation up to $\gtrsim$, then $\mathcal{R} \subseteq \approx$.*

The proofs of Theorem 4.4.3 and of Theorem 4.4.5 below can be easily derived from the proof of Theorem 4.4.7.

Definition 4.4.4 (Bisimulation up to context) *A symmetric relation $\mathcal{R}$ is a* bisimulation up to context *if $M\ \mathcal{R}\ N$ implies:*

- *if $M \xrightarrow{\alpha} M''$, $\alpha \notin \{*.\texttt{enter}\_n, *.\texttt{exit}\_n\}$, then there exists a system N'' such that $N \stackrel{\hat{\alpha}}{\Longrightarrow} N''$, and for all processes P there is a system context $C[-]$ and systems M' and N' such that $M'' \bullet P = C[M']$, $N'' \bullet P = C[N']$, and $M'\ \mathcal{R}\ N'$;*
- *if $M \xrightarrow{*.\texttt{enter}\_n} M''$ then there exists a system N'' such that $N \mid n[\,\circ\,] \Rightarrow N''$, and for all processes P there is a system context $C[-]$ and systems M' and N' such that $M'' \bullet P = C[M']$, $N'' \bullet P = C[N']$, and $M'\ \mathcal{R}\ N'$;*

- *if* $M \xrightarrow{*.\texttt{exit}\_n} M''$ *then there exists a system* N'' *such that* $n[\circ \mid N] \Rightarrow N''$, *and for all processes* P *there is a system context* $C[-]$ *and systems* M' *and* N' *such that* $M'' \bullet P = C[M']$, $N'' \bullet P = C[N']$, *and* $M' \mathcal{R} N'$.

Theorem 4.4.5 *If* $\mathcal{R}$ *is a bisimulation up to context, then* $\mathcal{R} \subseteq \approx$.

Definition 4.4.6 (Bisimulation up to context and up to $\gtrsim$) *A symmetric relation* $\mathcal{R}$ *is a* bisimulation up to context and up to $\gtrsim$ *if* $M \mathcal{R} N$ *implies:*

- *if* $M \xrightarrow{\alpha} M''$, $\alpha \notin \{*.\texttt{enter}\_n, *.\texttt{exit}\_n\}$, *then there exists a system* N'' *such that* $N \stackrel{\hat{\alpha}}{\Longrightarrow} N''$, *and for all processes* P *there is a system context* $C[-]$ *and systems* M' *and* N' *such that* $M'' \bullet P \gtrsim C[M']$, $N'' \bullet P \gtrsim C[N']$, *and* $M' \mathcal{R} N'$;

- *if* $M \xrightarrow{*.\texttt{enter}\_n} M''$ *then there exists a system* N'' *such that* $N \mid n[\circ] \Rightarrow N''$, *and for all processes* P *there is a system context* $C[-]$ *and systems* M' *and* N' *such that* $M'' \bullet P \gtrsim C[M']$, $N'' \bullet P \gtrsim C[N']$, *and* $M' \mathcal{R} N'$;

- *if* $M \xrightarrow{*.\texttt{exit}\_n} M''$ *then there exist a system* N'' *such that* $n[\circ \mid N] \Rightarrow N''$, *and for all processes* P *there is a system context* $C[-]$ *and systems* M' *and* N' *such that* $M'' \bullet P \gtrsim C[M']$, $N'' \bullet P \gtrsim C[N']$, *and* $M' \mathcal{R} N'$.

Theorem 4.4.7 *If* $\mathcal{R}$ *is a bisimulation up to context and up to* $\gtrsim$, *then* $\mathcal{R} \subseteq \approx$.

Proof We define the relation $\mathcal{S}$ as the smallest relation such that:

1. $M \mathcal{R} N$ implies $M \mathcal{S} N$;
2. $M \gtrsim A, A \mathcal{S} B, B \lesssim N$ implies $M \mathcal{S} N$;
3. $M \mathcal{S} N$ implies $C[M] \mathcal{S} C[N]$, for all system contexts $C[-]$.

We prove by induction on its definition, that $\mathcal{S}$ is a late bisimulation. This will assure the soundness of the relation $\mathcal{R}$, because $M \mathcal{R} N$ implies $M \mathcal{S} N$ which implies $M \approx N$. Observe that $\mathcal{S}$ is symmetric because $\mathcal{R}$ is.

- $M \mathcal{S} N$ because $M \mathcal{R} N$.

  Suppose that $M \xrightarrow{\alpha} M''$, with $\alpha \notin \{*.\texttt{enter}\_n, *.\texttt{exit}\_n\}$. As $\mathcal{R}$ is a bisimulation up to context and up-to $\gtrsim$, we know that there exists a system N'' such that $N \stackrel{\alpha}{\Longrightarrow} N''$. We also know that for all process P, there exist a system context $C[-]$ and systems M' and N' such that $M'' \bullet P \gtrsim C[M']$, $N'' \bullet P \gtrsim C[N']$, and $M' \mathcal{R} N'$. This implies $M' \mathcal{S} N'$. By construction $\mathcal{S}$ is contextual and $C[M'] \mathcal{S} C[N']$ holds. By construction $\mathcal{S}$ is closed under expansion, and therefore $M'' \mathcal{S} N''$, as required.

  Suppose that $M \xrightarrow{*.\texttt{enter}\_n} M''$. As $\mathcal{R}$ is a bisimulation up to context and up to $\gtrsim$, we know that there exists a system N'' such that $N \mid n[\circ] \xRightarrow{*.\texttt{enter}\_n} N''$. We also know that for all process P, there exist a system context $C[-]$ and systems M' and N' such that $M'' \bullet P \gtrsim C[M']$, $N'' \bullet P \gtrsim C[N']$, and $M' \mathcal{R} N'$. This implies $M' \mathcal{S} N'$. By

construction, $\mathcal{S}$ is contextual, and $C[M']\ \mathcal{S}\ C[N']$ holds. By construction $\mathcal{S}$ is closed under expansion, and therefore $M''\ \mathcal{S}\ N''$, as required.

Suppose that $M \xrightarrow{*.\texttt{exit}\_n} M''$. As $\mathcal{R}$ is a bisimulation up to context and up to $\gtrsim$, we know that there exists a system N'' such that $n[\circ \mid N] \xRightarrow{*.\texttt{exit}\_n} N''$. We also know that for all process P, there exist a system context $C[-]$ and systems M' and N' such that $M'' \bullet P \gtrsim C[M']$, $N'' \bullet P \gtrsim C[N']$, and $M'\ \mathcal{R}\ N'$. This implies $M'\ \mathcal{S}\ N'$. By construction, $\mathcal{S}$ is contextual, and $C[M']\ \mathcal{S}\ C[N']$ holds. By construction $\mathcal{S}$ is closed under expansion, and we conclude $M''\ \mathcal{S}\ N''$, as required.

- $M\ \mathcal{S}\ N$ because $M \gtrsim A, A\ \mathcal{S}\ B, B \lesssim N$.

  The induction hypothesis tells us that $A\ \mathcal{S}\ B$ behaves like a late bisimulation.

  Suppose $M \xrightarrow{\alpha} M'$, with $\alpha \notin \{*.\texttt{enter}\_n, *.\texttt{exit}\_n\}$. A simple diagram chasing allows us to conclude that there are systems A', B', N' such that for all process P it holds $M' \bullet P \gtrsim A' \bullet P\ \mathcal{S}\ B' \bullet P \lesssim N' \bullet P$, and in turn, by construction of $\mathcal{S}$, $M' \bullet P\ \mathcal{S}\ N' \bullet P$.

  Suppose $M \xrightarrow{*.\texttt{enter}\_n} M'$. As $M \gtrsim A$, for all process P, it holds $M' \bullet P \gtrsim A \mid n[P]$. As $A\ \mathcal{S}\ B$, the closure properties of $\mathcal{S}$ assure that $A \mid n[P]\ \mathcal{S}\ B \mid n[P]$. The expansion relation is a congruence, and since $B\ \mathcal{S}\ N$ we conclude that $B \mid n[P] \lesssim N \mid n[P]$. But $N \mid n[P] \Rightarrow N \mid n[P]$, and $M' \bullet P \gtrsim \mathcal{S} \lesssim (N \mid n[\,\circ\,]) \bullet P$. This, by construction of $\mathcal{S}$, implies $M' \bullet P\ \mathcal{S}\ (N \mid n[\,\circ\,]) \bullet P$.

  Suppose $M \xrightarrow{*.\texttt{exit}\_n} M'$. As $M \gtrsim A$, for all process P, it holds $M' \bullet P \gtrsim n[P \mid A]$. As $A\ \mathcal{S}\ B$, the closure properties of $\mathcal{S}$ assure that $n[P \mid A]\ \mathcal{S}\ n[P \mid B]$. The expansion relation is a congruence, and since $B\ \mathcal{S}\ N$ we conclude that $n[P \mid A] \lesssim n[P \mid N]$. But $n[P \mid B] \Rightarrow n[P \mid N]$, and $M' \bullet P \gtrsim \mathcal{S} \lesssim n[\circ \mid N] \bullet P$. This, by construction of $\mathcal{S}$, implies $M' \bullet P\ \mathcal{S}\ n[\circ \mid N] \bullet P$.

- $C[M]\ \mathcal{S}\ C[N]$ because $M\ \mathcal{S}\ N$ and $C[-]$ is a system context.

  The induction hypothesis tells us that $(M, N) \in \mathcal{S}$ is a pair satisfying the bisimulation conditions in $\mathcal{S}$. Lemma 4.3.3 assures that the pair $(C[M], C[N]) \in \mathcal{S}$ satisfies the bisimulation conditions in $\mathcal{S}$.

This completes the induction. □

4.5 Adding communication

The basic idea is to have an *output process* such as $\langle E \rangle.P$, which outputs the message E and then continues as P, and an input process $(x)Q$ which on receiving a message binds it to x in Q which then executes; here occurrences of x in Q are bound. Notice that we have synchronous output; as discussed in [VC99, San01, BCC01a] this is not unrealistic because communication in MA is always local. The syntax of our extended language, together with the reduction rule for communication, is given in Table 4.9.

The LTS is extended by the introduction of two new pre-actions (E) for input, $\langle - \rangle$ for output, and a new form of concretions $(\boldsymbol{\nu}\tilde{m})\langle E \rangle Q$. In Table 4.11 we give all the defining rules

Names: $a, b, \ldots, k, l, m, n, \ldots \in \mathbf{N}$

Capabilities:

C	::=	$\texttt{in\_}n$	may enter into n
	$\mid$	$\texttt{out\_}n$	may exit out of n
	$\mid$	$\texttt{open\_}n$	may open n

Expressions:

E, F	::=	x	variable
	$\mid$	C	capability
	$\mid$	$E.F$	path
	$\mid$	ε	empty path

Guards:

G	::=	E	expression
	$\mid$	(x)	input
	$\mid$	$\langle E \rangle$	output

Systems:

M, N	::=	$\mathbf{0}$	termination
	$\mid$	$M_1 \mid M_2$	parallel composition
	$\mid$	$(\boldsymbol{\nu} n)M$	restriction
	$\mid$	$n[P]$	ambient

Processes:

P, Q, R	::=	$\mathbf{0}$	nil process
	$\mid$	$P_1 \mid P_2$	parallel composition
	$\mid$	$(\boldsymbol{\nu} n)P$	restriction
	$\mid$	$G.P$	prefixing
	$\mid$	$n[P]$	ambient
	$\mid$	$!G.P$	replication

Structural and Reduction rules for Communication:

$E.(F.P) \equiv (E.F).P$	(Struct Path)
$\varepsilon.P \rightarrowtriangle P$	(Red Empty Path)
$(x).P \mid \langle M \rangle.Q \rightarrowtriangle P[M/x] \mid Q$	(Red Comm)

Table 4.9: Message-passing Mobile Ambients in two levels

$$\textit{Pre-actions: } \pi ::= \ldots \mid (E) \mid \langle - \rangle \qquad \textit{Concretions: } K ::= (\boldsymbol{\nu}\tilde{m})\langle P \rangle Q \mid (\boldsymbol{\nu}\tilde{m})\langle E \rangle Q$$

Table 4.10: Pre-actions and concretions for communication

$$(\pi \text{ Output}) \ \frac{}{\langle E \rangle.P \xrightarrow{\langle - \rangle} \langle E \rangle P} \qquad (\pi \text{ Input}) \ \frac{}{(x).P \xrightarrow{(E)} P[E/x]}$$

$$(\pi \text{ Path}) \ \frac{E.(F.P) \xrightarrow{\pi} Q}{(E.F).P \xrightarrow{\pi} Q} \qquad (\tau \text{ Eps}) \ \frac{}{\epsilon.P \xrightarrow{\tau} P}$$

$$(\tau \text{ Comm}) \ \frac{P \xrightarrow{\langle - \rangle} (\boldsymbol{\nu}\tilde{m})\langle E \rangle P' \quad Q \xrightarrow{(E)} Q' \quad \text{fn}(Q') \cap \{\tilde{m}\} = \emptyset}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}\tilde{m})(P' \mid Q')}$$

Table 4.11: Labelled transition system: communication

that should be added to those of Table 4.4 and Table 4.5 to obtain the LTS for the extended calculus. Note that in the structural rules of Table 4.4 we are now assuming that parallel composition and restriction distribute over the new form of concretions $(\boldsymbol{\nu}\tilde{m})\langle E \rangle Q$ in the same manner as $(\boldsymbol{\nu}\tilde{m})\langle P \rangle Q$. The slightly unusual pre-action for output allows a uniform treatment of extrusion of names.

The proof of Theorem 4.2.3 can be easily completed to take into account the extended calculus. A consequence of working with MA in two levels is that communication capabilities cannot be observed at top-level. Moreover, the free variables of a system cannot be instantiated by a system context, because in a system context the hole cannot appear under a prefix. This in turn implies that our bisimulations can be applied to the extended calculus, and all the results of Section 4.3 and Section 4.4 hold without modifications.

Theorem 4.5.1 *Late bisimilarity, early bisimilarity, and reduction barbed congruence coincide in the Message Passing Calculus.*

Theorem 4.5.2 *The up-to expansion, up-to context, and up-to context and expansion proof techniques are sound proof techniques in the Message Passing Calculus.*

4.6 Algebraic theory

In this section we prove a collection of algebraic properties using our bisimulation proof methods. Then, we prove the correctness of a protocol for controlling access through a *firewall*, first proposed in [CG00b].

We briefly comment on the laws of Theorem 4.6.1. We recall that M, N range over systems and P, Q, R over processes. The first two laws are two examples of local communication

within private ambients without interference. The third law is the well-known perfect firewall law. The following four laws represent non-interference properties about movements of private ambients. Finally, the last two laws describe when opening cannot be interfered.

As a convenient notation, we write $\prod_{j\in J} P_j$ for $P_1 \mid \ldots \mid P_j$. We also write $\mathcal{R}^=$ for the symmetric closure of $\mathcal{R}$.

Theorem 4.6.1

1. $(\boldsymbol{\nu} n)n[\langle W\rangle.P \mid (x).Q \mid M] \;\cong\; (\boldsymbol{\nu} n)n[P \mid Q[W/x] \mid M]$ *if* $n \notin \mathrm{fn}(M)$
2. $(\boldsymbol{\nu} n)n[\langle W\rangle.P \mid (x).Q \mid \prod_{j\in J} \mathtt{open}\_k_j.R_j] \;\cong\; (\boldsymbol{\nu} n)n[P \mid Q[W/x] \mid \prod_{j\in J} \mathtt{open}\_k_j.R_j]$
3. $(\boldsymbol{\nu} n)n[P] \;\cong\; \mathbf{0}$ *if* $n \notin \mathrm{fn}(P)$
4. $(\boldsymbol{\nu} n)((\boldsymbol{\nu} m)m[\mathtt{in}\_n.P] \mid n[M]) \;\cong\; (\boldsymbol{\nu} n)n[(\boldsymbol{\nu} m)m[P] \mid M]$ *if* $n \notin \mathrm{fn}(M)$
5. $(\boldsymbol{\nu} m, n)(m[\mathtt{in}\_n.P] \mid n[\prod_{j\in J} \mathtt{open}\_k_j.R_j]) \;\cong\; (\boldsymbol{\nu} m, n)n[m[P] \mid \prod_{j\in J} \mathtt{open}\_k_j.R_j]$
6. $(\boldsymbol{\nu} n)n[(\boldsymbol{\nu} m)m[\mathtt{out}\_n.P] \mid M] \;\cong\; (\boldsymbol{\nu} n)((\boldsymbol{\nu} m)m[P] \mid n[M])$ *if* $n \notin \mathrm{fn}(M)$
7. $(\boldsymbol{\nu} n)n[m[\mathtt{out}\_n.P] \mid \prod_{j\in J} \mathtt{open}\_k_j.R_j] \;\cong\; (\boldsymbol{\nu} n)(m[P] \mid n[\prod_{j\in J} \mathtt{open}\_k_j.R_j])$ *if* $m \neq k_j$, *for* $j \in J$
8. $n[(\boldsymbol{\nu} m)(\mathtt{open}\_m.P \mid m[N]) \mid Q] \;\cong\; n[(\boldsymbol{\nu} m)(P \mid N) \mid Q]$ *if* $Q \equiv M \mid \prod_{j\in J}\langle W_j\rangle.R_j$ *and* $m \notin \mathrm{fn}(N)$
9. $(\boldsymbol{\nu} n)n[(\boldsymbol{\nu} m)(\mathtt{open}\_m.P \mid m[Q]) \mid R] \;\cong\; (\boldsymbol{\nu} n)n[(\boldsymbol{\nu} m)(P \mid Q) \mid R]$ *if* $R \equiv \prod_{i\in I}\langle W_i\rangle.S_i \mid \prod_{j\in J} \mathtt{open}\_k_j.R_j$ *and* $m, n \notin \mathrm{fn}(Q)$.

Proof To prove the above laws, except (3) and (9), we exhibit the appropriate bisimulation. In all cases the bisimulation has a similar form:

$$\mathcal{S} = \{(lhs, rhs)\}^= \cup \approx$$

where *lhs* and *rhs* denote respectively the left hand side and the right hand side of the equation, parameterised over names, processes and systems. For proving the laws (3) and (9) we show that the above $\mathcal{S}$ is a bisimulation up to context and up to structural congruence. We illustrate the proof of the law (3). Let $\mathcal{S} = \{((\boldsymbol{\nu} n)n[Q], \mathbf{0}) \mid \forall Q \text{ s.t. } n \notin \mathrm{fn}(Q)\}^=$. We show that $\mathcal{S}$ is a bisimulation up to context and up to structural congruence. The most delicate cases are those regarding the silent moves $*.\mathtt{enter}\_k$ and $*.\mathtt{exit}\_k$. For instance, if

$$(\boldsymbol{\nu} n)n[P] \xrightarrow{*.\mathtt{enter}\_k} (\boldsymbol{\nu} n)k[\circ \mid n[P']] \equiv k[\circ \mid (\boldsymbol{\nu} n)n[P']]$$

then

$$\mathbf{0} \mid k[\,\circ\,] \;\Rightarrow\equiv k[\circ \mid \mathbf{0}]$$

and up to context and structural congruence we are still in $\mathcal{S}$. □

On stuttering In [San01] it is shown that barbed behavioural equivalence is insensitive to *stuttering* phenomena, originated by processes that may repeatedly enter and exit an ambient. Using a sum operator *à la* CCS, the next example conveys some intuitions about stuttering. The systems

$$M = m[\mathtt{in}\_n.\mathtt{out}\_n.\mathtt{in}\_n.R] \qquad \text{and} \qquad N = m[\mathtt{in}\_n.\mathtt{out}\_n.\mathtt{in}\_n.R + \mathtt{in}\_n.R]$$

are indeed reduction barbed congruent. To see why the extra summand of N does not affect its behaviour, consider a reduction produced by this summand:

$$N \mid n[S] \;\rightarrow\; n[S \mid m[R]] \;.$$

The process M can match it using three reductions:

$$M \mid n[S] \;\rightarrow\; n[S \mid m[\mathtt{out}\_n.\mathtt{in}\_n.R]] \;\rightarrow\; n[S] \mid m[\mathtt{in}\_n.R] \;\rightarrow\; n[S \mid m[R]] \;.$$

The crucial point is that the exercise of the capability $\mathtt{in}\_n$ is matched by the exercise of three capabilities, $\mathtt{in}\_n.\mathtt{out}\_n.\mathtt{in}\_n$. Although it might seem that our bisimilarity matches each action with only one action (possibly preceded and/or followed by τ transitions), our bisimilarity is actually insensitive to stuttering. To illustrate why, we use a variant of the example above that does not rely on internal sum. Replication in the systems M and N below implements a loop with an alternation between input/output and the path $\mathtt{in}\_n.\mathtt{out}\_n$. There is a 1-cycle shift, however, between the two loops. Stuttering makes the shift irrelevant.

Proposition 4.6.2 *The systems M and N defined as*

$$\begin{aligned} M &= m[(\boldsymbol{\nu} l)(\mathtt{in}\_n.l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])] \\ N &= m[(\boldsymbol{\nu} l)(\mathtt{in}\_n.\mathtt{out}\_n.\mathtt{in}\_n.l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])] \end{aligned}$$

are bisimilar.

Proof Let

$$\begin{aligned} \mathcal{R} = \{(\; & k[O \mid (\boldsymbol{\nu} l)(\mathtt{in}\_n.l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])] \;, \\ & k[O \mid (\boldsymbol{\nu} l)(\mathtt{in}\_n.\mathtt{out}\_n.\mathtt{in}\_n.l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])] \;) \\ & \qquad \mid \; k \text{ and } O \text{ are arbitrary}\}^{=} \;\cup\; \mathcal{I} \;. \end{aligned}$$

where $\mathcal{I}$ is the identity relation between systems. The relation $\mathcal{R}$ is a bisimulation up to context and up to structural congruence. We detail the most interesting case, where the exercise of one capability must be matched by the exercise of three capabilities. Suppose $M \;\mathcal{R}\; N$, with

$$M = k[O \mid (\boldsymbol{\nu} l)(\mathtt{in}\_n.l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])]$$

and

$$N = k[O \mid (\boldsymbol{\nu} l)(\mathtt{in}\_n.\mathtt{out}\_n.\mathtt{in}\_n.l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])] \;.$$

Also suppose that

$$M \;\xrightarrow{k.\mathtt{enter}\_n}\; n[\circ \mid k[O \mid (\boldsymbol{\nu} l)(l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])]] \;.$$

Then N can perform the following sequence of transitions:

$$\begin{aligned} N &\xrightarrow{k.\mathtt{enter}\_n} n[\circ \mid k[O \mid (\boldsymbol{\nu} l)(\mathtt{out}\_n.\mathtt{in}\_n.l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])]] \\ &\xrightarrow{\tau} n[\,\circ\,] \mid k[O \mid (\boldsymbol{\nu} l)(\mathtt{in}\_n.l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])] \\ &\xrightarrow{\tau} n[\circ \mid k[O \mid (\boldsymbol{\nu} l)(l[] \mid !\mathtt{open}\_l.\mathtt{out}\_n.\mathtt{in}\_n.l[])]] \ . \end{aligned}$$

For all instantiations of $\circ$ we can factor out the context $n[\circ \mid -]$ and up to context we are still in $\mathcal{R}$. □

The proof above clearly shows how the exercise of the three capabilities $\mathtt{in}\_n.\mathtt{out}\_n.\mathtt{in}\_n$ needed to match the capability $\mathtt{in}\_n$ give rise to a $k.\mathtt{enter}\_n$ action followed by two internal transitions. The internal actions are subsequently absorbed by the weak formulation of the equivalence.

Crossing a firewall A protocol is discussed in [CG00b] for controlling access through a firewall. The ambient w represents the firewall; the ambient m, a trusted agent containing a process Q that is supposed to cross the firewall. The firewall ambient sends into the agent a pilot ambient k with the capability $\mathtt{in}\_w$ for entering the firewall. The agent acquires the capability by opening k. The process Q carried by the agent is finally liberated inside the firewall by the opening of ambient m. Names m and k act like passwords which guarantee the access only to authorised agents. Here is the protocol in MA:

$$\begin{aligned} AG &\stackrel{\text{def}}{=} m[\mathtt{open}\_k.(x).x.Q] \\ FW &\stackrel{\text{def}}{=} (\boldsymbol{\nu} w)w[\mathtt{open}\_m.P \mid k[\mathtt{out}\_w.\mathtt{in}\_m.\langle \mathtt{in}\_w \rangle]] \end{aligned}$$

The correctness (of a mild variant) of the protocol above is shown in [CG00b] for may-testing [DH84] proving that

$$(\boldsymbol{\nu} m, k)(AG \mid FG) \ \cong \ (\boldsymbol{\nu} w)w[Q \mid P]$$

under the conditions that $w \notin \mathrm{fn}(Q)$, $x \notin \mathrm{fv}(Q)$, $\{m, k\} \cap (\mathrm{fn}(P) \cup \mathrm{fn}(Q)) = \emptyset$. The proof relies on non-trivial contextual reasonings. In what follows, we show how it can be established using our bisimulation proof methods.

The system on the right can be obtained from that one on the left by executing six τ-actions. So, it suffices to prove that $\cong$ is insensitive to all these τ-actions. The result follows from the algebraic laws of Theorem 4.6.1 and the following two laws:

Lemma 4.6.3 *Let P, Q, and R be processes. Then*

1. $(\boldsymbol{\nu} k, m, w)(k[\mathtt{in}\_m.P] \mid m[\mathtt{open}\_k.Q] \mid w[\mathtt{open}\_m.R])$
   $\cong \ (\boldsymbol{\nu} k, m, w)(m[k[P] \mid \mathtt{open}\_k.Q] \mid w[\mathtt{open}\_m.R])$

2. $(\boldsymbol{\nu} m, w)(m[\langle \mathtt{in}\_w \rangle \mid (x).P] \mid w[\mathtt{open}\_m.Q])$
   $\cong \ (\boldsymbol{\nu} m, w)(m[P[\mathtt{in}\_w/x]] \mid w[\mathtt{open}\_m.Q])$

Proof By exhibiting the appropriate bisimulation. Again, in all cases the bisimulation has a similar form:

$$\mathcal{S} = \{(lhs, rhs)\}^{=} \cup \approx$$

where *lhs* and *rhs* denote respectively the left hand side and the right hand side of the equation. □

Theorem 4.6.4 *If* $w \notin \mathrm{fn}(Q)$, $x \notin \mathrm{fv}(Q)$, *and* $\{m, k\} \cap (\mathrm{fn}(P) \cup \mathrm{fn}(Q)) = \emptyset$, *then*

$$(\boldsymbol{\nu} m, k)(AG \mid FG) \cong (\boldsymbol{\nu} w) w[Q \mid P].$$

Proof It suffices to apply the algebraic laws of Theorem 4.6.1 and Lemma 4.6.3. More precisely, we apply, in sequence, Law (7) of Theorem 4.6.1, Law (1) of Lemma 4.6.3, Law (9) of Theorem 4.6.1, Law (2) of Lemma 4.6.3, and Laws (5) and (9) of Theorem 4.6.1. □

Notes and References

The Seal Calculus and Mobile Ambients are built on the idea that some 'network awareness' is required to model wide-area distributed mobile computation. An excellent survey of process algebras with localities is offered by Section 4 of Castellani's 'Process Algebras with Localities' [Cas01]. We limit ourselves to a panoramic view of the state of the art in bisimulation proof techniques for languages involving mobility of code.

HOπ Work towards developing a clear understanding of the use of higher-order features in process languages began several years ago with various proposals for higher-order versions of known calculi. In the introduction we have discussed at length the difficulties encountered by these 'early approaches'. The difficulty arises in establishing the soundness of the proof technique, which is tantamount to establishing some sort of contextuality property.

Sangiorgi studied intensively HOπ and introduced *context bisimulation*. Context bisimulation involves quantifications over all contexts a process can interact with, and by doing so it 'circumvents' the difficulties to establish the contextuality of the bisimulation. All the first part of this thesis relies on the ideas behind context bisimulation. Furthermore, one of Sangiorgi's achievements was to provide a translation of the higher-order language, which supports code mobility, to a first-order π-calculus which supports only name mobility.[1] With the restriction to a language of finite types, the translation led to *normal bisimulation*, a bisimulation that coincides with context bisimulation but does not involve the universal quantification over the interacting contexts.

Recently, Jeffrey and Rathke [JR03] developed an alternative presentation of the labelled transition system and a novel proof technique which allows them to remove the limitation on recursive types. The basic idea behind their work is to mimic the contribution of the interacting context directly in the LTS. In this way, they can rely on a standard, well-understood, CCS-like bisimulation. To give an account of their technique, consider again the LTS for MA we proposed in Section 4.2. Transitions introduce a placeholder which is instantiated by an arbitrary process in the bisimulation game. Jeffrey and Rathke's approach is to add rules to the LTS so that the placeholder behaves as an arbitrary process. As for now, Jeffrey and Rathke's novel labelled transition system is deeply inspired by the translation of HOπ into π-calculus, and this makes it difficult to export their ideas to different process calculi.

Mobile Ambients Higher-order LTSs for Mobile Ambients can be found in [CG96, GC02, Vig99, FMT01], but we are not aware of any form of bisimilarity defined using these LTSs. In [Sew03], Sewell addresses the problem of uniformly deriving LTSs and bisimulation congruences from the reduction rules of a calculus. The transitions generated for a fragment

[1] The existence of the translation indirectly implies that up-to proof techniques can be applied to HOπ, even if their correctness has not been proved in a direct way. (Sangiorgi, personal communication).

of Mobile Ambients require the same universal quantifications on the content of the interacting ambient as ours. Sewell's techniques only apply to strong equivalences. A simple first-order labelled transition system for Mobile Ambients without restriction is proposed by Sangiorgi in [San01]. Using this LTS the author defines an *intensional* bisimilarity for MA that separates terms on the basis of their internal structure.

A new impetus to research on labelled bisimilarities for Ambient-like languages, has been given by Merro and Hennessy in [MH02]. They give an LTS and a characterisation of reduction barbed congruence for a variant of Levi and Sangiorgi's Safe Ambients, called SAP.

Our work on Mobile Ambients is the natural prosecution of [MH02], where we tackle the original problem: to provide bisimulation proof methods for Mobile Ambients. The main differences with respect to [MH02] are the following:

- SAP differs from MA for having co-capabilities and passwords, both features are essential to prove the characterisation result in SAP.
- Our env-actions, unlike those in [MH02], are truly late, as they do not mention the process provided by the environment. This process can be added *later*, when playing the bisimulation game.
- Our actions for ambient's movement, unlike those in SAP, report the name of the migrating ambient. For instance, in $k.\texttt{enter}\_n$ we say that ambient k enters n. The knowledge of k is necessary to make the action observable by the environment. This is not needed in SAP, because movements can be observed by means of co-capabilities.
- Co-capabilities also allow the observation of the movement of an ambient whose name is private. As a consequence, the perfect firewall equation does not hold in SAP or in Safe Ambients. In MA, the movements of an ambient whose name is private cannot be observed. This is why the perfect firewall equation holds.

It is interesting to notice that dialects of Mobile Ambients evolved closer to the Seal calculus. SafeAmbients [LS00a] requires mobility to be the consequence of a process synchronisation. SafeAmbients with Passwords [MH02] requires that the mobility takes place on (what we can assimilate to) channels[2]. The very last step that distinguishes these Ambient variations from Seal is that Seal uses objective mobility — the agent is sent by the surrounding environment — while in Ambient-based calculi mobility is subjective — the agent sends itself —. It seemed quite natural that results similar to those of Merro and Hennessy can be stated easily for Seals. This turned out to be 'wishful thinking', as we have seen in Sections 3.3 and 3.4.

Locations and failures Locations are the basic observable entities in both Mobile Ambients and the Seal Calculus and they play a fundamental role in interaction. A different role is played by locations in the Distributed Join Calculus [FGL$^+$96] and in the simplest form of Dπ [RH97]. In these calculi locations provide an explicit support for specifying the distribution of resources. The locality structure is mostly descriptive: the routing of messages across remote locations is transparent, and locations cannot be directly observed.

[2]Merro and Hennessy also modify Levi and Sangiorgi's calculus so that the coaction of an $\texttt{out}\_n$ must be placed exactly as a receive action in Seal.

In other words, it is equivalent to run processes P and Q at two different machines, or to run the compound process $P \mid Q$ at a single machine. As a consequence, to reason about processes it is possible to erase all the locality informations[3] and relate the simple semantics (respectively the Join Calculus semantics or CCS semantics) to the distributed semantics. This is not longer true if these models are refined so that locations can fail. The crash of a physical site causes the permanent failure of its processes. As a consequence, locations become (partially) observable.

New calculi and labelled bisimilarities In Chapter 3, an investigation of the expected behavioural theories of the Seal's dialects suggested us to focus on a particular dialect. In general, the definition of a new process calculus can only benefit from a study of its behavioural theory, and a tractable and/or rich behavioural theory can be used as a strong argument to advertise a new process calculus.

This is the case for the *Calculus of Mobile Resources* [GHS02] and for *New Boxed Ambients* [BCMS]. In fact, together with its syntax and operational semantics, a characterisation of barbed congruence in terms of a labelled bisimilarity (in both cases a contextual bisimulation), is given. More than that, when research on the Calculus of Mobile Resources was in a preliminary stage, difficulties encountered in the definition of the labelled bisimilarity put forward alternative directions.

Typed equivalences Programming practise usually imposes a discipline on how variables or names are used. When this discipline is not explicit in the language, 'expected' properties may not hold, or the theory of the language may reveal not strong enough to prove them. *Types* can be used to make such disciplines explicit.

The use of types affects contextually-defined behavioural equivalences such as reduction barbed congruence, because in a typed calculus the processes being compared must obey to the same typing, and the contexts in which they are tested must be compatible with this typing. Typically, in a typed calculus the class of legal contexts in which two processes may be tested is smaller than in the untyped calculus.

The literature of types for mobile systems is huge, but most of the works define type systems to ensure *behavioural properties* of terms, like security properties. We limit our review to 'types to ensure *behavioural equalities*'. Among the works that deserve to be mentioned figure the already cited [LS00b], where a type system is used to assure that processes inside ambients are single threaded. This rule out some interferences, and allow the proof of some simple behavioural equalities between terms of Safe Ambients. The characterisation of typed barbed congruence with a labelled bisimilarity presents new difficulties. Seminal work in this area has been carried over by Boreale and Sangiorgi for IO types [BS98], and to a lesser extent by Pierce and Sangiorgi for polymorphic types [PS00]. In presence of higher-order computation, characterisations of typed barbed congruence have been developed by Hennessy, Merro, and Rathke [HMR03] and by Hennessy, Rathke, and Yoshida [HRY03]. In these works, types implement dynamic policies of resource access control.

[3] In the Distributed Join Calculus a little care is required to rule out migration attempts towards one's own sublocations, which may delay or block some processes.

The Heritage of the Seal Calculus

As one of the contributions of this thesis has been the development of the Seal Calculus, we will review the influence that Seal had on other works in the field.

Location mobility as a result of process synchronisation was first introduced in Seal as a natural extension of π-calculus communication primitives. It was later introduced for ambients by Levi and Sangiorgi in [LS00a], and then in several other Ambient variants. The Seal calculus also had a direct influence on the design of Boxed Ambients [BCC01a, BCC01b], a variant of the Ambient calculus obtained by suppressing the open capability and where non local communication are made possible by enriching the calculus with the communication primitives of the *located* Seal calculus. The communication primitives of the *shared* Seal calculus, first defined in [CGZN01], were adapted to Boxed Ambients in [CBC02] in order to ease static detection of insecure information flows. The difference in expressiveness of the communication primitives of the located and shared variant of the primitives is at the origin of the NBA (New Boxed Ambients) calculus defined in [BCMS], which enriches the shared channel communication of Seal they borrow from [CGZN01] with name capturing receiving actions (however these are useful only because they conform to the ambient calculus model where names are tightly bound to agents and are not modified by mobility: since in Seal the receiving agents decide the name of incoming seals this feature is useless). The interaction pattern introduced in Seal for mobility of exiting agents is first used for ambients in [MH02]. Seal has inspired the calculus of mobile resources of [GHS02] which inherits the interaction pattern for exiting agents and the fact that mobility takes place on the anonymous content of locations. Finally Seal primitives are also at the basis of the definition of Boxed-π, an extension of the π-calculus for securely integrating trusted and untrusted off-the-shelf software components [SV99, PJ03].

A process language out of a semantic model:

towards a domain theory for concurrency

5 new-HOPLA

The language new-HOPLA is a compact but expressive language for higher-order nondeterministic processes. It extends the language HOPLA of Nygaard and Winskel ([NW02, NW03, NW, Nyg03]) with operators to handle dynamically generated names. Its definition has been guided by the development of a domain theory for concurrency: while in the previous chapters we started from pre-existent process languages with the goal of equipping them with a reasonable semantics, our aim here is to build a language on top of (what appear to be) solid basis. This fits within a broader project of research, towards a fully fledged domain theory for concurrency based on presheaf categories [NW, CW03].

The language new-HOPLA has an operational interpretation, and it is on its operational account that we concentrate: behaviour of terms is again our main concern.

The language new-HOPLA can be roughly described as an extension of the simply typed λ-calculus with prefixing, sums, and constructs to manipulate dynamically generated names. The type of a process describes the possible (shapes of) computation paths that the process can perform. More in detail, let s be the set of names currently known by both the process and the environment. Then a typing judgement

$$\alpha_1{:}\mathbb{N}, \ldots, \alpha_n{:}\mathbb{N}; x_1{:}\mathbb{P}_1, \ldots, x_n{:}\mathbb{P}_n \vdash t : \mathbb{Q}$$

means that the process t yields computation paths in $\mathbb{Q}$ once name variables $\alpha_1, \ldots, \alpha_n$ are instantiated with names in s, and processes with computation paths in $\mathbb{P}_1, \ldots, \mathbb{P}_n$ are assigned to the process variables $x_1, \ldots, x_n$. The types $\mathbb{P}$ are built using function spaces, sum, an anonymous action prefix type, (a limited form of) product, and recursive definition. A special function space accounts for name-generation.

It is notable that although we can express many kinds of concurrent processes in the language, the language itself does not have many features typical of process calculi built-in, beyond nondeterministic sum and prefix operations. In particular, parallel composition of processes, *à la* CCS, and restriction, *à la* π-calculus, can *be defined* in new-HOPLA, but are not primitive constructs.

Overview The reader may like to look ahead to the operational presentation in Section 5.2 and to the study of its behavioural theory in Section 5.3. In addition, Section 5.4 reports encodings of π-calculus, polyadic π-calculus, and Mobile Ambients into new-HOPLA. But we start in Section 5.1 with an account of a domain theory for concurrency built on path sets, mostly taken from [NW]. Indexing the denotational model allows us to take into account name generation, and this lead us to an overview of the denotational semantics of new-HOPLA, fundamental to understand its design choices.

5.1 Domain theory for path sets

In the path semantics, processes are intuitively represented as collections of their computation paths. Paths are elements of preorders $(\mathbb{P}, \leq_{\mathbb{P}}), (\mathbb{Q}, \leq_{\mathbb{Q}}), \ldots$ (also denoted $\mathbb{P}, \mathbb{Q}, \ldots$) called *path orders*. Path orders function as process types, each describing the set of possible paths for processes of that type together with their sub-path ordering. A process of type $\mathbb{P}$ is then represented as a downwards-closed subset $X \subseteq \mathbb{P}$, called a *path set*. Path sets ordered by inclusion form the elements of the poset $\widehat{\mathbb{P}}$ which we will think of as a domain of meanings of processes of type $\mathbb{P}$.

The poset $\widehat{\mathbb{P}}$ has many interesting properties. First of all, it is a complete lattice with joins given by union. In the sense of Hennessy and Plotkin [HP79], $\widehat{\mathbb{P}}$ is a nondeterministic domain, with joins used to interpret nondeterministic sums of processes. Accordingly, given a family $(X_i)_{i\in I}$ of elements of $\widehat{\mathbb{P}}$, we will often write $\Sigma_{i\in I} X_i$ for their join. A typical finite join is written $X_1 + \cdots + X_n$ while the empty join is the empty set, the inactive process, written $\varnothing$.

A second important property of $\widehat{\mathbb{P}}$ is that any $X \in \widehat{\mathbb{P}}$ is the join of certain prime elements below it; $\widehat{\mathbb{P}}$ is a *prime algebraic complete lattice* [NPW81]. Primes are down-closures $y_{\mathbb{P}}p = \{p' \mid p' \leq_{\mathbb{P}} p\}$ of individual elements of $\widehat{\mathbb{P}}$, representing a process that may perform the computation path p. The map $y_{\mathbb{P}}$ reflects as well as preserves order, so that $p \leq_{\mathbb{P}} q$ if and only if $y_{\mathbb{P}}p \subseteq y_{\mathbb{P}}q$, and thus $y_{\mathbb{P}}$ embeds $\mathbb{P}$ in $\widehat{\mathbb{P}}$. We clearly have $y_{\mathbb{P}}p \subseteq X$ if and only if $p \in X$, and prime algebraicity of $\widehat{\mathbb{P}}$ amounts to saying that any $X \in \widehat{\mathbb{P}}$ is the union of its elements:

$$X = \bigcup_{p\in X} y_{\mathbb{P}}p \ . \tag{5.1}$$

Finally, $\widehat{\mathbb{P}}$ is characterised abstractly as the *free join-completion* of $\mathbb{P}$, meaning it is join-complete, and given any join complete poset C and a monotone map $f : \mathbb{P} \to C$, there is a unique join-preserving map $f^\dagger : \widehat{\mathbb{P}} \to C$ such that the diagram on the left below commutes.

$$\begin{array}{ccc} \mathbb{P} & \xrightarrow{y_{\mathbb{P}}} & \widehat{\mathbb{P}} \\ & \searrow^{f} & \downarrow{\scriptstyle f^\dagger} \\ & & C \end{array} \qquad\qquad f^\dagger X = \textstyle\bigcup_{p\in X} fp \ . \tag{5.2}$$

We call $f^\dagger$ the *extension of f along $y_{\mathbb{P}}$*. Uniqueness of $f^\dagger$ follows from equation (5.1). Notice that we may instantiate C to any poset of the form $\widehat{\mathbb{Q}}$, drawing our attention to join-preserving maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$. We will now investigate what additional type structure is at hand.

Linear category Write **Lin** for the category with path orders $\mathbb{P}, \mathbb{Q}, \ldots$ as objects, and join-preserving maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ as arrows. **Lin** can be understood as a categorical model of classical linear logic, and accordingly we call arrows of **Lin** *linear* maps. The involution of linear logic, $\mathbb{P}^\perp$, is given by $\mathbb{P}^{\mathrm{op}}$. The tensor product of $\mathbb{P}$ and $\mathbb{Q}$, denoted $\mathbb{P} \otimes \mathbb{Q}$ is given by the product of preorders, $\mathbb{P} \times \mathbb{Q}$. The function space from $\mathbb{P}$ to $\mathbb{Q}$, denoted $\mathbb{P} \multimap \mathbb{Q}$, is given by $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$. To justify this, observe that by the freeness property (5.2), join preserving maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ are in bijective correspondence with monotone maps $\mathbb{P} \to \widehat{\mathbb{Q}}$. Also, each element Y of

$\mathbb{Q}$ can be represented using its characteristic function, a monotone map $f_Y : \mathbb{Q}^{\mathrm{op}} \to \mathbf{2}$ from the opposite order to the simple poset $0 < 1$ such that $Y = \{q \mid f_Y q = 1\}$ and $\widehat{\mathbb{Q}} \cong [\mathbb{Q}^{\mathrm{op}}, \mathbf{2}]$. We then obtain the following chain by uncurrying:

$$[\mathbb{P}, \widehat{\mathbb{Q}}] \cong [\mathbb{P}, [\mathbb{Q}^{\mathrm{op}}, \mathbf{2}]] \cong [\mathbb{P} \times \mathbb{Q}^{\mathrm{op}}, \mathbf{2}] = [(\mathbb{P}^{\mathrm{op}} \times \mathbb{Q})^{\mathrm{op}}, \mathbf{2}] \cong \widehat{\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}} \ . \tag{5.3}$$

So the preorder $\mathbb{P}^{\mathrm{op}} \times \mathbb{Q}$ provides a type for a function space from $\mathbb{P}$ to $\mathbb{Q}$. On objects $\mathbb{P}$ and $\mathbb{Q}$, products (written $\mathbb{P}\&\mathbb{Q}$) and coproducts are both given by $\mathbb{P} + \mathbb{Q}$, the disjoint juxtaposition of preorders. An element of $\widehat{\mathbb{P} + \mathbb{Q}}$ can be identified with a pair (X, Y), with $X \in \widehat{\mathbb{P}}$ and $Y \in \widehat{\mathbb{Q}}$, which provides the projections $\pi_1 : \mathbb{P} + \mathbb{Q} \to \mathbb{P}$ and $\pi_2 : \mathbb{P} + \mathbb{Q} \to \mathbb{Q}$. Injection maps in **Lin** derive from the obvious injections into the disjoint sum of preorders $\mathbb{P} + \mathbb{Q}$. The empty order $\mathbb{O}$ plays the roles of empty product and empty coproduct. Actually, arbitrary products and coproducts are obtained by the construction above. This collapse of products and coproducts highlights that **Lin** has arbitrary *biproducts*.

While rich in structure, **Lin** does not support all operations associated with process languages. In fact, all maps are join preserving, and, unlike e.g. prefixing, always send the empty sum (the inactive process) to itself.

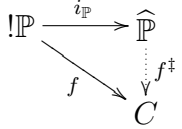
Continuous maps As a solution, following the discipline of linear logic, nonlinear maps can be obtained as linear maps whose domain is under an exponential. One choice of a suitable exponential for **Lin** is got by taking $(!\mathbb{P}, \preceq_{\mathbb{P}})$ to be the set of finite path sets of $\mathbb{P}$ with order

$$P \preceq_{\mathbb{P}} P' \Leftrightarrow \forall p \in P . \exists p' \in P' . p \leq_{\mathbb{P}} p'$$

(e.g., the finite join-completion of $\mathbb{P}$)—so $!\mathbb{P}$ can be thought of as consisting of compund paths associated with several runs. There is an obvious map $i_{\mathbb{P}} : !\mathbb{P} \to \widehat{\mathbb{P}}$ sending a finite set $\{p_1, \ldots, p_n\}$ to the join $y_{\mathbb{P}} p_1 + \cdots + y_{\mathbb{P}} p_n$. Such finite sums of primes are the finite (compact, isolated) elements of $\widehat{\mathbb{P}}$. The map $i_{\mathbb{P}}$ assumes the role of $y_{\mathbb{P}}$ above. For any $X \in \widehat{\mathbb{P}}$ and $P \in !\mathbb{P}$, we have $i_{\mathbb{P}} P \subseteq X$ if and only if $P \preceq_{\mathbb{P}} X$, and X is the directed join of the finite elements below it:

$$X = \bigcup_{P \preceq_{\mathbb{P}} X} i_{\mathbb{P}} P \ . \tag{5.4}$$

Further, $\widehat{\mathbb{P}}$ is the *free directed join completion* of $!\mathbb{P}$ (also called *ideal completion*). This means that given any monotone map $f : !\mathbb{P} \to C$ for some directed-join complete poset C, there is a unique directed-join preserving (i.e. Scott continuous) map $f^\ddagger : \widehat{\mathbb{P}} \to C$ such that the diagram below commutes.



$$f^\ddagger X = \textstyle\bigcup_{P \preceq X} f P \ . \tag{5.5}$$

We call $f^\ddagger$ the *extension of f along $i_{\mathbb{P}}$*. Uniqueness of $f^\ddagger$ follows from 5.4. As before we can replace C by a nondeterministic domain $\widehat{\mathbb{Q}}$ and by the freeness properties (5.1) and (5.4) we obtain a bijective correspondence between linear maps $!\mathbb{P} \to \mathbb{Q}$ and continuous maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$.

Continuous maps allow more process operations, including prefixing, to be expressed. An element of $P \in !\mathbb{P}$ consists of several, possibly no, computation paths of $\mathbb{P}$. Therefore it can intuitively be understood as describing a compound computation path associated with running several copies of a process of type $\mathbb{P}$.

Our choice of exponentials defines a functor $! : \mathbf{Lin} \to \mathbf{Lin}$, that sends an object $\mathbb{P}$ into $!\mathbb{P}$, and sends a linear arrow $f : \mathbb{P} \to \mathbb{Q}$ into $!f : !\mathbb{P} \to !\mathbb{Q}$, the linear map sending a finite path set X to the set of finite paths sets y such that $y \subseteq f(X)$. Via (5.5) the map $y_{!\mathbb{P}} : !\mathbb{P} \to \widehat{!\mathbb{P}}$ extends to a map $\eta_{\mathbb{P}} = y^{\ddagger}_{!\mathbb{P}} : \widehat{\mathbb{P}} \to \widehat{!\mathbb{P}}$:

$$\eta_{\mathbb{P}} X = \bigcup_{P \preceq X} y_{!\mathbb{P}} P$$

that is, $\eta_{\mathbb{P}} : \mathbb{P} \to !\mathbb{P}$ is a continuous map that sends a path set X to the set of the finite paths sets contained in X. This map provides a denotational counterpart of prefixing, see for instance [NW].

Indexing Lin Continuous maps can be used to define a category[1] that gives a fully abstract model for HOPLA [NW03]. Yet, one more step must be taken to take into account name-generation.

Following the ideas of [Sta96], there are two important observations which underlie the nature of the sought model.

1. The behaviour of a process depends only on a *finite* set of names which it contains, i.e. its *support*. So, for each set of names s we will have a model $\mathbb{P}(s)$ to interpret the behaviour of processes of type $\mathbb{P}$ which only use the names in s.
2. The behaviour of a process is preserved by injective renaming of its (free) names. This implies the existence, for each type $\mathbb{P}$ and for each injection $i : s \rightarrowtail_{\text{inj}} s'$, of a *translation* $\mathbb{P}(i) : \mathbb{P}(s) \to \mathbb{P}(s')$ that relates the space of meanings $\mathbb{P}(s')$ to that of $\mathbb{P}(s)$.

A convenient framework in which to formalise this uniform family of interpretations is in terms of functor categories.

Definition 5.1.1 *Define $\mathcal{I}$ to be the category of finite sets and injective functions. Define $\mathbf{Lin}^{\mathcal{I}}$ to be the category of functors from $\mathcal{I}$ to* **Lin** *and natural transformations.*

To see why $\mathbf{Lin}^{\mathcal{I}}$ is a good candidate for a domain theory of concurrency able to express also dynamically generated names, recall that to give a functor $\mathbb{P} : \mathcal{I} \to \mathbf{Lin}$ is to give an indexed family $(\mathbb{P}(s))_{s \in \text{obj}(\mathcal{I})}$ of path orders, together with families of linear maps $(\mathbb{P}(i) : \mathbb{P}(s) \to \mathbb{P}(s'))_{i : s \to s'}$ such that the naturality conditions hold. Functoriality and naturality give uniformity over varying name sets.

We will now investigate the type structure of the category $\mathbf{Lin}^{\mathcal{I}}$. Overloading the notation, we will denote objects of $\mathbf{Lin}^{\mathcal{I}}$ with $\mathbb{P}, \mathbb{Q}, ...$

[1] The category **Cts** has path orders $\mathbb{P}, \mathbb{Q}, \ldots$ as objects, and continuous maps $\widehat{\mathbb{P}} \to \widehat{\mathbb{Q}}$ as arrows.

As usual with functor categories, much (but not all) of the structure of **Lin** extends to $\mathbf{Lin}^{\mathcal{I}}$. In particular, tensor $\otimes$, product &, sum $+$, and exponential ! extend pointwise.[2]

Some constructions are instead proper to $\mathbf{Lin}^{\mathcal{I}}$. A basic one is the *object of names* construction, that allows to see a set of names as an object of **Lin**. For that, define $\mathbb{N} : \mathcal{I} \to \mathbf{Lin}$ to be the functor whose action at s returns s itself, seen as a flat order, and whose action on arrows is $\mathbb{N}(i)X = \{i(n) \mid n \in X\}$ for $i : s \to s'$ (remember that $X \in \widehat{\mathbb{N}(s)}$). In other words, $\mathbb{N}$ is given by the chain of inclusions $\mathcal{I} \hookrightarrow \mathbf{Set} \hookrightarrow \mathbf{Lin}$.

The *disjoint union* of sets in $\mathcal{I}$ lifts to a symmetric monoidal closed structure on $\mathbf{Lin}^{\mathcal{I}}$, following a Day-like construction [Day70]. The associated tensor is denoted $\boxtimes$ (pronounce *funny tensor*). The intuition behind this construction is that in an object $\mathbb{P} \boxtimes \mathbb{Q}$ the two components $\mathbb{P}$ and $\mathbb{Q}$ must have *disjoint supports*. A good illustration of this is provided by the comparison of the objects $\mathbb{N} \otimes \mathbb{N}$ and $\mathbb{N} \boxtimes \mathbb{N}$:

$$\mathbb{N} \otimes \mathbb{N}(s) = \{(n_1, n_2) \mid n_1, n_2 \in s\} \qquad \mathbb{N} \boxtimes \mathbb{N}(s) = \{(n_1, n_2) \mid n_1, n_2 \in s \text{ and } n_1 \neq n_2\} \ .$$

It is possible to construct several function spaces in $\mathbf{Lin}^{\mathcal{I}}$. Our attention is directed to $\mathbb{N} \multimap \mathbb{P}$, and $!\mathbb{P} \multimap \mathbb{Q}$, where $\multimap$ is the function space associated to $\otimes$.[3]

The function space associated to $\boxtimes$, when its domain its restricted to the object of names, gives rise to the functor $\delta : \mathbf{Lin}^{\mathcal{I}} \to \mathbf{Lin}^{\mathcal{I}}$, called the *shift functor*. If s is a finite set, we write $s + 1$ for the 'generic' set obtained by adding a new element, denoted $*_s$, to the set s. For $f : s \to s'$, we write $f + 1 : s + 1 \to s' + 1$ for the function that acts like f on the elements of s, and sends $*_s$ onto $*_{s'}$. The shift functor is then defined on objects as $\delta\mathbb{P}(s) = \mathbb{P}(s + 1)$, and as $\delta\mathbb{P}(i) = \mathbb{P}(i + 1)$ on arrows. Intuitively, it extends the set of the public names s with a new, *fresh*, name: regarded in this way, δ represents functions that will only accept a new name as input.

The constructions we have highlighted in $\mathbf{Lin}^{\mathcal{I}}$ provide an interpretation for the types of new-HOPLA: we move further to examine their operational interpretation.

5.2 The language

The language new-HOPLA is a typed process language directly suggested by the structure of the category $\mathbf{Lin}^{\mathcal{I}}$. Before formally defining the language, we discuss informally the constructs associated with the types, and their operational interpretation.

The empty type **0** denotes the empty set of computation paths: the only term of this type is the inactive term **0**, a term that performs no actions.

A prefix type has the form $!\mathbb{P}$; it describes computation paths in which first a prototypical action, which we call '!' (pronounce *beep*), is performed before resuming as a computation path in $\mathbb{P}$. The prefix type is associated with a *prefix operation* taking a process t in $\mathbb{P}$ to $!t$ of type $!\mathbb{P}$. The behaviour of the term $!t$ (and, in general, of the terms of the language) is

[2] As an example, given $\mathbb{P}$ and $\mathbb{Q}$, their tensor $\mathbb{P} \otimes \mathbb{Q}$ is the functor defined as $(\mathbb{P} \otimes \mathbb{Q})(s) = \mathbb{P}(s) \otimes \mathbb{Q}(s)$ for $s \in \mathcal{I}$, and for $i : s \to s'$, $(\mathbb{P} \otimes \mathbb{Q})(i) : (\mathbb{P}(s) \otimes \mathbb{Q}(s)) \to (\mathbb{P}(s') \otimes \mathbb{Q}(s'))$ is defined as $(\mathbb{P} \otimes \mathbb{Q})(i) = \mathbb{P}(i) \otimes \mathbb{Q}(i)$. Same with $\&, +, !$.

[3] Glynn Winskel recently proved that, under certain conditions satisfied in our framework, the function space $!\mathbb{P} \multimap \mathbb{Q}$ does exist in $\mathbf{Lin}^{\mathcal{I}}$ (personal communication).

specified by the transition relation formally defined in Section 5.2.3. For the time being, we will simply say that a term t may perform an action p (which is a 'fragment' of a computation path) and resume as the continuation. So the term $!t$ performs the prototypical action ! and resumes as t. The prototypical action is the basic visible action on top of which more complex and more informative actions can be built.

The sum type constructed from a family of types $(\mathbb{P}_i)_{i\in I}$ is written $\Sigma_{i\in I}\mathbb{P}_i$. Interpreted using the byproduct of $\mathbf{Lin}^{\mathcal{I}}$, the sum is both a product and a coproduct. It is associated with *injection* (tagging) term constructors, producing a term $i{:}t$ of type $\Sigma_{i\in I}\mathbb{P}_i$ from a term t of type $\mathbb{P}_i$, for $i \in I$. Conversely, *projection* term constructors produce a term $\pi_i t$ of type $\mathbb{P}_i$ from a term t of the sum type above. Operationally, if t performs an action p, then $i{:}t$ performs the same action tagged with i, e.g., $i{:}p$. Projection $\pi_i t$ filters the actions p performed by t: if p has the form $j{:}p'$ for some p' and $j = i$ then $\pi_i t$ emits p', otherwise it emits nothing. Thus, the semantics identifies the term $\pi_i(i{:}t)$ with t, and $\pi_i(j{:}t)$ with $\mathbf{0}$, for $i \neq j$.

The tensor type $\mathbb{N} \otimes \mathbb{P}$ denotes tagging a process of type $\mathbb{P}$ with a name. The term constructor produces a term $n \cdot t$ of type $\mathbb{N} \otimes \mathbb{P}$ from a term t of type $\mathbb{P}$ and a name n. Accordingly all computation paths of t are tagged with n: if p is an action performed by t, then $n \cdot p$ is an action performed by $n \cdot t$. The destructor, denoted $\pi_n t$ checks if an action performed by t is tagged by n: if yes it erases the tag, otherwise it blocks the action. Observe that there is a close correspondence between the operational behaviour of tensor an sum. In HOPLA, where it is assumed an immutable set of public names, there is no tensor type and its role is played by the sum type.

The function space type $\mathbb{P} \to \mathbb{Q}$ corresponds to $!\mathbb{P} \multimap \mathbb{Q}$ in $\mathbf{Lin}^{I}$ and is analogous to function space in simply typed λ-calculus. A process $\lambda x.t$ of type $\mathbb{P} \to \mathbb{Q}$ takes a process u that behaves as $\mathbb{P}$, binds it to x, and then behaves as $\mathbb{Q}$. That is, if $t[u/x]$ performs an action p and continues as t', then $\lambda x.t$ performs an action $u \mapsto p$ (read 'if applied to u it performs p'), and resumes as t'. Process application is denoted tu, for $t : \mathbb{P} \to \mathbb{Q}$ and $u : \mathbb{P}$. The informal behaviour is that if t upon being applied to u performs p and becomes t', then tu performs p and resumes as t'.

The function space type $\mathbb{N} \to \mathbb{Q}$ corresponds to $\mathbb{N} \multimap \mathbb{Q}$ in $\mathbf{Lin}^{\mathcal{I}}$: a process $\lambda\alpha.t$ takes a name a, binds it to α, and then behaves as $\mathbb{Q}$. The operational interpretation is analogous to that described above, and the corresponding action is denoted $a \mapsto p$ ('if applied to a it performs p'), for p an action performed by $t[a/\alpha]$. Name application is denoted tn for $t : \mathbb{N} \to \mathbb{P}$ and n a name.

The type $\delta\mathbb{P}$ corresponds to a function space with domain $\mathbb{N}$ and codomain $\mathbb{P}$, subject to the constraint that the argument must be a *fresh* name. For a term t of type $\mathbb{P}$, the term $new\alpha.t$ has type $\delta\mathbb{P}$: it takes a fresh name, binds it to α, and then continues as t. The corresponding computation path is $new\alpha.p$, and it reads 'if applied to a fresh name, it performs p': the identity of the name is irrelevant, as soon as it is fresh. The destructor, called new name application, is denoted $t[n]$ for $t : \delta\mathbb{P}$ and n a name fresh for t.

Central to the expressivity of new-HOPLA is the *pattern matching* term $[t > p(x) \Rightarrow u]$, in which an action performed by t is matched against the path pattern p: if the match is successful, the continuation of t is passed on to u via the variable x. So, if the term t has type $\mathbb{P}$, the pattern p is a path of type $\mathbb{P}$ with a resumption variable x of type $\mathbb{R}$, and u

generally involves x. This term plays the role of prefix destructor: in particular first prefixing and then matching yields a single successful match in that the process $[!t > !x(x) \Rightarrow u]$ and $u[t/x]$ are identified.

5.2.1 Syntax

Types The type of names is denoted by $\mathbb{N}$. The types of processes are defined by the grammar below.

$$\begin{array}{rcl} \mathbb{P} & ::= & \mathbf{0} \;\mid\; \mathbb{N} \otimes \mathbb{P} \;\mid\; !\mathbb{P} \;\mid\; \delta\mathbb{P} \;\mid\; \mathbb{N} \to \mathbb{P} \;\mid\; \mathbb{P} \to \mathbb{Q} \;\mid \\ & & \Sigma_{i \in I}\mathbb{P}_i \;\mid\; \mu_j \vec{P}.(\mathbb{P}_1 \ldots \mathbb{P}_k) \;\mid\; P \end{array}$$

The type $\Sigma_{i \in I}\mathbb{P}_i$ when I is a finite set, is most often written $i_1{:}\mathbb{P} + \cdots + i_k{:}\mathbb{P}$. The symbol P is drawn from a set of type variables used in defining recursive types; closed type expressions are interpreted as path orders. Using vector notation, $\mu_j \vec{P}.\vec{\mathbb{P}}$ abbreviates $\mu_j P_1 \ldots P_k.(\mathbb{P}_1 \ldots \mathbb{P}_k)$ and is interpreted as the j-component, for $1 \leq j \leq k$, of the 'least' solution to the defining equations $P_1 = \mathbb{P}_1, \ldots, P_k = \mathbb{P}_k$, where the expressions $\mathbb{P}_1, \ldots, \mathbb{P}_k$ may contain the P_j's. We shall write $\mu \vec{P}.\vec{\mathbb{P}}$ as an abbreviation for the k-tuple with j-component $\mu_j \vec{P}.\vec{\mathbb{P}}$. Simultaneous recursive equations for path orders can be solved using information systems [Sco82, LW84].

Terms and paths We assume an infinite, countable, set of *name constants*, ranged over by $a, b, \ldots$ and an infinite, countable, set of *name variables*, ranged over by $\alpha, \beta, \ldots$ Names, either constants or variables, are ranged over by $m, n, \ldots$.

$$\begin{array}{rcll} m, n, \ldots & ::= & a, b, \ldots & \text{name constants} \\ & \mid & \alpha, \beta, \ldots & \text{name variables} \end{array}$$

We assume an infinite, countable, set of *process variables*, ranged over by $x, y, \ldots$

Every type is associated with actions processes of that type may do. The *actions* are defined by the grammar below:

$$p, q, r \quad ::= \quad x \;\mid\; !p \;\mid\; n \cdot p \;\mid\; i{:}p \;\mid\; new\alpha.p \;\mid\; n \mapsto p \;\mid\; u \mapsto p \;\mid\; p[n] \;.$$

As we will see shortly, well-typed actions are constructed so that they involve exactly one prototypical action !, and exactly one 'resumption variable' x. Whenever a term performs the action, the variable of the action matches the resumption of the term: the type of an action thus relates the type of a term with the type of its resumption.

According to the transition rules a process of prefix type $!\mathbb{P}$ may do actions of the form $!p$, while a process of tensor or sum type may do actions of the form $n \cdot p$ or $i{:}p$ respectively. A process of type $\delta\mathbb{P}$ does actions of the form $new\alpha.p$ meaning that at the generation of a new name, a say, as input the action $p[a/\alpha]$ is performed. Actions of function type $n \mapsto p$ or $u \mapsto p$ express the dependency of the action on the input of a name n or process u respectively. The final clause is necessary in building up actions because we sometimes need to apply a resumption variable to a new name.

The *terms* are defined by the grammar below:

t, u, v	$::=$	$\mathbf{0}$	$!t$	inactive process and protypical action
		$n \cdot t$	$\pi_n t$	tensor and projection
		$\lambda x.t$	tu	process abstraction and application
		$\lambda\alpha.t$	tn	name abstraction and application
		$new\alpha.t$	$t[n]$	new name abstraction and application
		$recx.t$	x	recursive definition and process variables
		$i{:}t$	$\pi_i t$	injection and projection
		$\Sigma_{i\in I} t_i$		sum
		$\Sigma_{\alpha\in\mathbb{N}} t$		sum over names
		$[t > p(x) \Rightarrow u]$		pattern matching

In new-HOPLA actions are used as patterns in terms $[t > p(x) \Rightarrow u]$ where we explicitly note the resumption variable x. If the term t can perform the action p the resumption of t is passed on to u via the variable x.

Freshness We end this section by defining what a *fresh name* is. For that we first classify the free variables of a term (we assume an understanding of free variables):

- *free name variables*, denoted $\mathrm{fnv}(-)$: the binders of name variables are $\lambda\alpha.-$, $new\alpha.-$, and $\Sigma_{\alpha\in\mathbb{N}}-$;
- *free process variables*, denoted $\mathrm{fpv}(-)$: the binders of process variables are $\lambda x.-$, and $[t > p(x) \Rightarrow -]$;
- *name constants*, denoted $\mathrm{nc}(-)$: set of name constants of a term.

We say that a name is *fresh* for a closed term if it does not belong to the set of its name constants, also called its *support*. Formally:

Definition 5.2.1 (Support) *The* support *of a closed term t, denoted $\mathsf{n}(t)$, is the set of the its name constants $nc(t)$.*

Definition 5.2.2 (Freshness) *A name n is* fresh *for a closed term t if $n \notin \mathsf{n}(t)$.*

Our definition of freshness corresponds to FreshML semantic notion of 'not-in-the-support-of' (see [PG00]).

5.2.2 Typing judgements

As the old saying goes, *well-typed terms do not go wrong*. Compared to what happens in the simply typed λ-calculus, or even in HOPLA, a type system that ensures the old saying in new-HOPLA requires special care to deal with fresh names.

Consider the term $t = t'[\alpha]$. As we have informally discussed in the previous section, this denotes new-name application: any name instantiating α should be fresh for the term t'. Consider now the context $C[-] = \lambda\alpha.-$. In the term $C[t] = \lambda\alpha.(t'[\alpha])$, the variable α is abstracted via a lambda abstraction, and may be instantiated with any current name. In

particular it may be instantiated with names that belong to the support of t', thus breaking the hypothesis that t' has been applied to a fresh name. The same problem arises with contexts of the form $C[-] = \Sigma_{\alpha\in\mathbb{N}}-$. Moreover, if the process variable x is free in t, a context like $C[-] = \lambda x.-$ might replace x with an arbitrary term u. As the name instantiating α might belong to the support of u, nothing ensures that it is still fresh for the term $t[u/x]$.

The type system must sometimes ensure that name variables are instantiated by fresh names. To impose this restriction, the typing context contains not only typing assumptions about name and process variables, such as $\alpha{:}\mathbb{N}$ or $x{:}\mathbb{P}$, but also *freshness assumptions* (or *distinctions*) about them, written (α, β) or (α, x). Here the intended meaning of statements like (α, β) is that, in the any environment, the name instantiating the variables α and β must be *distinct*, and should never be identified. A freshness assumption like (α, x), where x is a process variable, records that in any environment the name instantiating α must be fresh for the term instantiating x.

Using this auxiliary information, the type system assumes that it is safe to abstract a variable, using lambda abstraction or sum over names, only if no freshness assumptions have been made on it.

The type system of new-HOPLA terms can be specified using judgements of the form:

$$A; \Gamma; d \vdash t : \mathbb{P}$$

where

- $A \equiv \alpha_1{:}\mathbb{N}, \ldots, \alpha_k{:}\mathbb{N}$ is a collection of name variables;
- $\Gamma \equiv x_1{:}\mathbb{P}_1, \ldots, x_k{:}\mathbb{P}_k$ is a partial function from process variables to types;
- d is a set of pairs $(\alpha, x) \in A \times \Gamma$, and $(\alpha, \beta) \in A \times A$, keeping track of the *freshness assumptions*.

The following notations are convenient for expressing properties of distinctions. We write $d \setminus \alpha$ for the set of distinctions obtained from d by deleting all pairs containing α. Many set of distinctions arising in practise are of the form $(A \times A) \setminus I$ where I is the identity relation. Therefore we write A itself as an abbreviation for $(A \times A) \setminus I$. The order in which variables appear in a distinction is irrelevant: we will write $(\alpha, \beta) \in d$ as a shorthand for $(\alpha, \beta) \in d$ *or* $(\beta, \alpha) \in d$. When we write $\Gamma \cup \Gamma'$ we allow the environments to overlap; the variables need not be disjoint provided the environments are consistent.

Actions are typed along the same lines, even if type judgements explicitly report the resumption variable:

$$A; \Gamma; d; ; x{:}\mathbb{R} \vdash p : \mathbb{P} \ .$$

The meaning of the environment $A; \Gamma; d$ is exactly the same as above. The variable x is the resumption variable of the pattern p, and its type is $\mathbb{R}$. The syntactically different judgements highlight that the resumption variable is not a free variable of an action.[4]

The type system of new-HOPLA is reported in Table 5.1 and Table 5.2.

The rule responsible for generating freshness assumptions is the rule for new-name application. If the term t has been typed in the environment $A; \Gamma; d$ and α is a new name

[4]...even if, with an abuse of notation, it is convenient to manipulate the resumption variables using standard substitutions when no ambiguity arises.

$$\frac{}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ !x : !\mathbb{R}} \qquad \frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ \alpha \cdot p : \mathbb{N} \otimes \mathbb{P}}\alpha \in A$$

$$\frac{\alpha{:}\mathbb{N}, A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}}{A;\Gamma;(d \setminus \alpha);;x'{:}\delta\mathbb{R} \Vdash\!\vdash\ new\alpha.p[x'[\alpha]/x] : \delta\mathbb{P}} \qquad \frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ \alpha \mapsto p : \mathbb{P}}\alpha \in A$$

$$\frac{A;\Gamma;d \vdash u : \mathbb{Q} \qquad A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ u \mapsto p : \mathbb{Q} \to \mathbb{P}} \qquad \frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}_j \quad j \in I}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ (j{:}p) : \Sigma_{i\in I}\mathbb{P}_i}$$

$$\frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ t : \mathbb{P}_j[\mu\vec{P}.\vec{\mathbb{P}}/\vec{P}]}{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ t : \mu_j P : \vec{\mathbb{P}}} \qquad \frac{A;\Gamma;d;;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}}{A';\Gamma';d';;x{:}\mathbb{R} \Vdash\!\vdash\ p : \mathbb{P}} \begin{array}{l} A \subseteq A' \\ \Gamma \subseteq \Gamma' \\ d \subseteq d' \end{array}$$

Table 5.1: new-HOPLA: typing rules for actions (or path patterns)

variable (that is, $\alpha \notin A$), then the term $t[\alpha]$ is well-typed under the hypothesis that any name instantiating the variable α is distinct from all the names in terms instantiating the variables that can appear in t. This is achieved adding the set of freshness assumptions $\{\alpha\} \times (\Gamma \cup A)$ to d.[5]

The rule for pattern matching also modifies the freshness assumptions. The operational rule of pattern matching substitutes a subterm of t, whose names are contained in A', for x. Accordingly, the typing rule initially checks that no name in A' belongs to the set of the variables supposed fresh for x. Our attention is then drawn to the term $u[t'/x]$, where t' is a subterm of t. A name variable $\alpha \in A$ supposed fresh from x when typing u, must now be supposed fresh from all the free variables of t'. This justifies the freshness assumptions $\{\{\alpha\} \times (A' \cup \Gamma') \mid (\alpha, x) \in d\}$.

The rest of the type system is fairly standard along the lines of type systems for the simply typed λ-calculus.

To link-up with the denotational model introduced in Section 5.1, we sketch the interpretation of new-HOPLAin $\mathbf{Lin}^{\mathcal{I}}$. A type environment $\alpha_1{:}\mathbb{N}, \ldots, \alpha_n{:}\mathbb{N}; x_1{:}\mathbb{P}_1, \ldots, x_n{:}\mathbb{P}_n; \emptyset$ stands for the object

$$[\![\, \alpha_1{:}\mathbb{N}, \ldots, \alpha_n{:}\mathbb{N}; x_1{:}\mathbb{P}_1, \ldots, x_n{:}\mathbb{P}_n; \emptyset \,]\!] : \mathbb{N} \otimes \cdots \otimes \mathbb{N} \otimes\, !\mathbb{P}_1 \otimes \cdots \otimes\, !\mathbb{P}_n$$

in $\mathbf{Lin}^{\mathcal{I}}$. As tensor is given pointwise by product, denotations of environments at a set of names s are tuples. If the set of freshness assumptions is not empty, then the denotation of a type environment is a subset of such tuples, containing exactly those tuples satisfying the freshness assumptions. Then, a syntactic type judgement $A;\Gamma;d \vdash t : \mathbb{P}$ stands for a linear map

$$[\![\, A;\Gamma;d \vdash t : \mathbb{P} \,]\!] : [\![\, A;\Gamma;d \,]\!] \to \mathbb{P} \ .$$

[5]When convenient, as here, we will confuse an environment with its domain.

$$\frac{}{A;\Gamma;d\vdash\emptyset:\mathbb{P}} \qquad \frac{}{A;x{:}\mathbb{P},\Gamma;d\vdash x:\mathbb{P}} \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}}{A';\Gamma';d'\vdash t:\mathbb{P}}\begin{array}{l}A\subseteq A'\\ \Gamma\subseteq\Gamma'\\ d\subseteq d'\end{array}$$

$$\frac{\alpha{:}\mathbb{N},A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash\Sigma_{\alpha\in\mathbb{N}}t:\mathbb{P}}\alpha\notin d \qquad \frac{\alpha{:}\mathbb{N},A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash\lambda\alpha.t:\mathbb{N}\to\mathbb{P}}\alpha\notin d$$

$$\frac{A;x{:}\mathbb{Q},\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash\lambda x.t:\mathbb{Q}\to\mathbb{P}}x\notin d \qquad \frac{A;x{:}\mathbb{P},\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash rec x.t:\mathbb{P}}x\notin d$$

$$\frac{\alpha{:}\mathbb{N},A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;(d\setminus\alpha)\vdash new\alpha.t:\delta\mathbb{P}} \qquad \frac{A;\Gamma;d\vdash t:\delta\mathbb{P}}{\alpha{:}\mathbb{N},A;\Gamma;d\cup(\{\alpha\}\times(\Gamma\cup A))\vdash t[\alpha]:\mathbb{P}}$$

$$\frac{A;\Gamma;d\vdash t:\mathbb{N}\to\mathbb{P}}{A;\Gamma;d\vdash t\alpha:\mathbb{P}}\alpha\in A \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}\to\mathbb{Q}\quad A;\Gamma;d\vdash u:\mathbb{P}}{A;\Gamma;d\vdash tu:\mathbb{Q}}$$

$$\frac{A;\Gamma;d\vdash t_i:\mathbb{P}\quad\forall i\in I}{A;\Gamma;d\vdash\Sigma_{i\in I}t_i:\mathbb{P}} \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}_i}{A;\Gamma;d\vdash i{:}t:\Sigma_{i\in I}\mathbb{P}_i} \qquad \frac{A;\Gamma;d\vdash t:\Sigma_{i\in I}\mathbb{P}_i}{A;\Gamma;d\vdash\pi_i t:\mathbb{P}_i} \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash !t:!\mathbb{P}}$$

$$\frac{A;\Gamma;d\vdash t:\mathbb{P}}{A;\Gamma;d\vdash\alpha\cdot t:\mathbb{N}\otimes\mathbb{P}}\alpha\in A \qquad \frac{A;\Gamma;d\vdash t:\mathbb{N}\otimes\mathbb{P}}{A;\Gamma;d\vdash\pi_\alpha t:\mathbb{P}}\alpha\in A \qquad \frac{A;\Gamma;d\vdash t:\mathbb{P}_j[\mu\vec{P}.\vec{\mathbb{P}}/\vec{P}]}{A;\Gamma;d\vdash t:\mu_j P:\vec{\mathbb{P}}}$$

$$\frac{A';\Gamma';d'\vdash t:\mathbb{P} \qquad A';\Gamma';d';;x{:}\mathbb{R}\Vdash p:\mathbb{P} \qquad A;x{:}\mathbb{R},\Gamma;d\vdash u:\mathbb{Q}}{A\cup A';\Gamma\cup\Gamma';\overline{d}\vdash[t>p(x)\Rightarrow u]:\mathbb{Q}}A'\cap\{\alpha\mid(\alpha,x)\in d\}=\emptyset$$

where $\overline{d}=(d\setminus x)\cup d'\cup\{\{\alpha\}\times(A'\cup\Gamma')\mid(\alpha,x)\in d\}$

Table 5.2: new-HOPLA: typing rules for processes

and a pattern judgement $A;\Gamma;d;;x{:}\mathbb{R} \vdash p : \mathbb{P}$ stands for a linear map

$$[\![\, A;\Gamma;d;;x{:}\mathbb{R} \vdash p : \mathbb{P} \,]\!] : [\![\, A;\Gamma;d \,]\!] \otimes \mathbb{R}^{\mathrm{op}} \to \mathbb{P}^{\mathrm{op}} \ .$$

The type system assumes that terms do not contain name constants. This is to avoid the complications in a type system coping with both name variables and constants at the same time. We write $s \vdash t : \mathbb{P}$ when there is a judgement $A;\emptyset;d \vdash \sigma t' : \mathbb{P}$ and a substitution σ for A respecting the freshness assumptions d such that t is $\sigma t'$. Similarly for patterns.

Proposition 5.2.3 *The judgement $s \vdash t : \mathbb{P}$ holds iff there is a canonical judgement $A;\emptyset;\{(\alpha,\beta) \mid \alpha \neq \beta\} \vdash t' : \mathbb{P}$, in which the substitution σ is a bijection between name variables and names and t is $\sigma t'$.*

We end this section by stating and proving some basic properties of the type system.

Lemma 5.2.4 (Substitution Lemma)

1. *Suppose that $A';\Gamma';d' \vdash t : \mathbb{Q}$ and $A;x{:}\mathbb{Q},\Gamma;d;;y{:}\mathbb{R} \Vdash p : \mathbb{P}$, where $\Gamma \cup \Gamma'$ is consistent and $A' \cap \{\alpha \mid (\alpha,x) \in d\} = \emptyset$. Then,*

   $$A \cup A';\Gamma \cup \Gamma';\overline{d};;y{:}\mathbb{R} \Vdash p[t/x] : \mathbb{P}$$

   where $\overline{d} = (d \setminus x) \cup d' \cup \{\{\alpha\} \times (A' \cup \Gamma') \mid (\alpha,x) \in d\}$.

2. *Suppose that $A';\Gamma';d' \vdash t : \mathbb{Q}$ and $A;x{:}\mathbb{Q},\Gamma;d \vdash u : \mathbb{P}$, where $\Gamma \cup \Gamma'$ is consistent and $A' \cap \{\alpha \mid (\alpha,x) \in d\} = \emptyset$. Then,*

   $$A \cup A';\Gamma \cup \Gamma';\overline{d} \vdash u[t/x] : \mathbb{P}$$

   where $\overline{d} = (d \setminus x) \cup d' \cup \{\{\alpha\} \times (A' \cup \Gamma') \mid (\alpha,x) \in d\}$.

Proof By rule induction.

Inaction. Suppose $A;x{:}\mathbb{Q},\Gamma;d \vdash \mathbf{0} : \mathbb{P}$. As $\mathbf{0}[t/x] = \mathbf{0}$, we derive $A \cup A';\Gamma \cup \Gamma';\overline{d} \vdash \mathbf{0}[t/x] : \mathbb{P}$ by the rule for inactive process.

Variable. Suppose $A;x{:}\mathbb{Q},\Gamma;d \vdash y : \mathbb{P}$. If $y \neq x$ then $y[t/x] = y$ and we derive $A \cup A';\Gamma \cup \Gamma';\overline{d} \vdash y[t/x] : \mathbb{P}$ by the rule for identity. If $y = x$, then $y[t/x] = t$ and $\mathbb{P} = \mathbb{Q}$: we derive and $A \cup A';\Gamma \cup \Gamma';\overline{d} \vdash y[t/x] : \mathbb{P}$ from $A';\Gamma';d' \vdash t : \mathbb{Q}$ by weakening.

Weakening. Suppose that $A'';x{:}\mathbb{Q},\Gamma'';d'' \vdash u : \mathbb{P}$ has been derived from $A;\Gamma;d \vdash u : \mathbb{P}$ for $A'' \supseteq A, \Gamma'' \supseteq \Gamma, d'' \supseteq d$. As $\Gamma'' \cup \Gamma'$ is consistent, then $\Gamma \cup \Gamma'$ is consistent as well. If $x{:}\mathbb{Q} \in \Gamma$, then by the induction hypothesis we have $A \cup A';\Gamma \cup \Gamma';\overline{d} \vdash u[t/x] : \mathbb{P}$ where $\overline{d} = (d \setminus x) \cup d' \cup \{\{\alpha\} \times (A' \cup \Gamma') \mid (\alpha,x) \in d\}$. By weakening we conclude $A'' \cup A';\Gamma'' \cup \Gamma';(d'' \setminus x) \cup d' \cup \{\{\alpha\} \times (A' \cup \Gamma') \mid (\alpha,x) \in d''\} \vdash u[t/x] : \mathbb{P}$. If $x{:}\mathbb{Q} \notin \Gamma$, then $x \notin \mathrm{fv}(u)$ and $u[t/x] = u$. The result follows from $A;\Gamma;d \vdash u : \mathbb{P}$ by weakening.

Tensor. Suppose that $A;x{:}\mathbb{Q};d \vdash \alpha \cdot u : \mathbb{P}$ has been derived from $A;x{:}\mathbb{Q};d \vdash u : \mathbb{P}$ for $\alpha \in A$. By the induction hypothesis we have $A \cup A';\Gamma \cup \Gamma';\overline{d} \vdash \alpha \cdot u[t/x] : \mathbb{P}$. As $\alpha \in A$, $\alpha \in A \cup A'$ as well, and we conclude by the rule for tensor.

Name projection. Analogous to tensor.

Name abstraction. Suppose that $A; x{:}\mathbb{Q}, \Gamma; d \vdash \lambda\alpha.u : \mathbb{N} \to \mathbb{P}$ has been derived from $\alpha{:}\mathbb{N}, A; x{:}\mathbb{Q}, \Gamma; d \vdash u : \mathbb{P}$ with $\alpha \notin d$. By the induction hypothesis we get $(\alpha{:}\mathbb{N}, A) \cup A'; \Gamma \cup \Gamma'; \overline{d} \vdash u[t/x] : \mathbb{P}$. As substitution is capture-avoiding, $\alpha{:}\mathbb{N} \notin A'$, and $(\alpha{:}\mathbb{N}, A) \cup A' = \alpha{:}\mathbb{N}, (A \cup A')$. It remains to show that $\alpha \notin \overline{d}$. But $\alpha \notin d'$ because $\alpha \notin A'$, and $\alpha \notin d$ because of the hypothesis. Thus, we can conclude by the rule for name abstraction.

Name application. Suppose that $A; x{:}\mathbb{Q}, \Gamma; d \vdash u\alpha : \mathbb{P}$ has been derived from $A; x{:}\mathbb{Q}, \Gamma; d \vdash u : \mathbb{N} \to \mathbb{P}$, with $\alpha \in A$. By the induction hypothesis we get $A \cup A'; \Gamma \cup \Gamma'; \overline{d} \vdash u[t/x] : \mathbb{N} \to \mathbb{P}$. As $\alpha \in A$, $\alpha \in (A \cup A')$. As $u[t/x]\alpha = u\alpha[t/x]$, by the rule for name application we get $A \cup A'; \Gamma \cup \Gamma'; \overline{d} \vdash u\alpha[t/x] : \mathbb{P}$, as required.

New name abstraction. Suppose that $A; x{:}\mathbb{Q}, \Gamma; (d \setminus \alpha) \vdash new\alpha.u : \delta\mathbb{P}$ has been derived from $\alpha{:}\mathbb{N}, A; x{:}\mathbb{Q}, \Gamma; d \vdash u : \mathbb{P}$ with $\alpha \notin d$. By the induction hypothesis we get $(\alpha{:}\mathbb{N}, A) \cup A'; \Gamma \cup \Gamma'; \overline{d} \vdash u[t/x] : \mathbb{P}$. As substitution is capture-avoiding, $\alpha{:}\mathbb{N} \notin A'$, and $(\alpha{:}\mathbb{N}, A) \cup A' = \alpha{:}\mathbb{N}, (A \cup A')$. It also holds $\overline{d} \setminus \alpha = ((d \setminus x) \cup d' \cup \{(\alpha, z) \mid z \in \Gamma' \text{ and } (\alpha, x) \in d\} \cup \{(\alpha, \beta) \mid \beta \in A' \text{ and } (\alpha, x) \in d\}) \setminus \alpha = ((d \setminus \alpha) \setminus x) \cup d' \cup \{(\alpha, z) \mid z \in \Gamma' \text{ and } (\alpha, x) \in (d \setminus \alpha)\} \cup \{(\alpha, \beta) \mid \beta \in A' \text{ and } (\alpha, x) \in (d \setminus \alpha)\}$. Then we can conclude by the rule for new name abstraction.

New name application. Suppose that $\alpha{:}\mathbb{N}, A; x{:}\mathbb{Q}, \Gamma; d \cup (\{\alpha\} \times ((x{:}\mathbb{Q}, \Gamma) \cup A)) \vdash u[\alpha] : \mathbb{P}$ has been derived $A; x{:}\mathbb{Q}, \Gamma; d \vdash u : \delta\mathbb{P}$. We want to show that $\alpha{:}\mathbb{N}, (A \cup A'); \Gamma \cup \Gamma'; \overline{d} \vdash u[\alpha][t/x] : \mathbb{P}$ where $\overline{d}$ is the set of distinctions:

$$((d \cup (\{\alpha\} \times ((x{:}\mathbb{Q}, \Gamma) \cup A))) \setminus x) \cup d' \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in (d \cup (\{\alpha\} \times ((x{:}\mathbb{Q}, \Gamma) \cup A)))\} .$$

By the induction hypothesis we get $A \cup A'; \Gamma \cup \Gamma'; (d \setminus x) \cup d' \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in d\} \vdash u[t/x] : \delta\mathbb{P}$. As $(\alpha, x) \in (d \cup (\{\alpha\} \times ((x{:}\mathbb{Q}, \Gamma) \cup A)))$, the hypothesis guarantee that $\alpha \notin A'$. By the rule for new name application we derive $\alpha{:}\mathbb{N}, (A \cup A'); \Gamma \cup \Gamma'; d_1 \vdash u[t/x][\alpha] : \mathbb{P}$, where the set of distinctions d_1 is

$$(d \setminus x) \cup d' \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in d\} \cup (\{\alpha\} \times (\Gamma \cup A)) .$$

Now, $u[t/x][\alpha] = u[\alpha][t/x]$. A little care is needed to show that d_1 is equal to $\overline{d}$. The set $\overline{d}$ can be rewritten as

$$\begin{aligned}&(d \setminus x) \cup (((\{\alpha\} \times ((x{:}\mathbb{Q}, \Gamma) \cup A)) \setminus x) \cup d' \\ &\quad \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in d\} \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in (\{\alpha\} \times ((x{:}\mathbb{Q}, \Gamma) \cup A))\}\end{aligned}$$

and in turn as

$$(d \setminus x) \cup (\{\alpha\} \times (\Gamma \cup A)) \cup d' \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in d\} \cup (\{\alpha\} \times (A' \cup \Gamma'))$$

and this is equal to d_1, as required.

Process abstraction. Suppose that $A; x{:}\mathbb{Q}, \Gamma; d \vdash \lambda y.u : \mathbb{R} \to \mathbb{P}$ has been derived from $A; y{:}\mathbb{R}, x{:}\mathbb{Q}, \Gamma; d \vdash u : \mathbb{P}$ with $y \notin d$. By the induction hypothesis we get $A \cup A'; (y{:}\mathbb{R}, \Gamma) \cup \Gamma'; \overline{d} \vdash u[t/x] : \mathbb{P}$. As substitution is capture-avoiding, $y{:}\mathbb{R} \notin \Gamma'$, and $(x{:}\mathbb{Q}, \Gamma) \cup \Gamma' = x{:}\mathbb{Q}, (\Gamma \cup \Gamma')$. It remains to show that $x \notin \overline{d}$. But $x \notin d'$ because $x \notin \Gamma'$, and $x \notin d$ because of the hypothesis. Thus, we can conclude by the rule for process abstraction.

Process application. Suppose that $A; x{:}\mathbb{Q}, \Gamma; d \vdash u : \mathbb{P}$ has been derived from $A; x{:}\mathbb{Q}, \Gamma; d \vdash u : \mathbb{R} \to \mathbb{P}$ and $A; x{:}\mathbb{Q}, \Gamma; d \vdash v : \mathbb{R}$. By the induction hypothesis we get $A \cup A'; \Gamma \cup \Gamma'; \overline{d} \vdash u[t/x] : \mathbb{R} \to \mathbb{P}$ and $A \cup A'; \Gamma \cup \Gamma'; \overline{d} \vdash v[t/x] : \mathbb{R}$. As $u[t/x]v[t/x] = uv[t/x]$, by the rule for process application we get $A \cup A'; \Gamma \cup \Gamma'; \overline{d} \vdash uv[t/x] : \mathbb{P}$, as required.

Recursion. Analogous to process abstraction.

Sum. Suppose that $A; x{:}\mathbb{Q}, \Gamma; d \vdash \Sigma_{i\in I} u_i : \mathbb{P}$ has been derived from $A; x{:}\mathbb{Q}, \Gamma; d \vdash u_i : \mathbb{P}$ for all $i \in I$. By the induction hypothesis we get $A \cup A'; \Gamma \cup \Gamma'; \overline{d} \vdash u_i[t/x] : \mathbb{P}$ for all $i \in I$. As $\Sigma_{i\in I}(u_i[t/x]) = \Sigma_{i\in I} u_i[t/x]$, by the rule for sum we get $A \cup A'; \Gamma \cup \Gamma'; \overline{d}\Sigma_{i\in I} \vdash u_i[t/x] : \mathbb{P}$, as required.

Injection, projection, lifting, recursive type. Straightforward use of the induction hypothesis, along the lines of the case of tensor.

Pattern matching. A little care is required to avoid getting confused by the type environments and the sets of distinctions. We detail the case in which the variable x belongs both to the type environment of the matched term, and to the type environment of the continuation. The other cases are simpler.

Suppose that

$$B \cup B; (x{:}\mathbb{Q}, \Delta) \cup (x{:}\mathbb{Q}, \Delta'); d \vdash [w > p(y) \Rightarrow v] : \mathbb{P}$$

has been derived from

$$B'; x{:}\mathbb{Q}, \Delta'; \delta' \vdash w : \mathbb{R} \qquad B'; x{:}\mathbb{Q}, \Delta'; \delta'; ; y{:}\mathbb{S} \vdash p : \mathbb{R} \qquad B; y{:}\mathbb{S}, x{:}\mathbb{Q}, \Delta; \delta \vdash v{:}\mathbb{P}$$

where

$$d = (\delta \setminus y) \cup \delta' \cup \{\{\beta\} \times (B' \cup (\Delta', x{:}\mathbb{Q})) \mid (\beta, y) \in \delta\}$$

and

$$B' \cap \{\beta \mid (\beta, y) \in \delta\} \;=\; \emptyset \; . \tag{5.6}$$

With respect to the hypothesis of the Lemma, $A = B \cup B'$ and $x{:}\mathbb{Q}, \Gamma = (x{:}\mathbb{Q}, \Delta) \cup (x{:}\mathbb{Q}, \Delta')$. Also remember that $A'; \Gamma'; d' \vdash t{:}\mathbb{Q}$ and

$$A' \cap \{\beta \mid (\beta, x) \in d\} = \emptyset \; . \tag{5.7}$$

We want to show that $B \cup B' \cup A'; \Delta \cup \Delta' \cup \Gamma'; \overline{d} \vdash [w > p(y) \Rightarrow v][t/x] : \mathbb{P}$ where $\overline{d} = (d \setminus x) \cup d' \cup \{\beta \times (A', \Gamma') \mid (\beta, x) \in d\}$.

By the induction hypothesis we have:

$$B' \cup A'; \Delta' \cup \Gamma'; d_1 \vdash w[t/x] : \mathbb{R} \qquad B' \cup A'; \Delta' \cup \Gamma'; d_1; ; y{:}\mathbb{S} \vdash p[t/x] : \mathbb{R}$$

where

$$d_1 \;=\; (\delta' \setminus x) \cup d' \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in \delta'\} \; .$$

We also have:

$$B \cup A'; (y{:}\mathbb{S}, \Delta) \cup \Gamma'; d_2 \vdash v[t/x] : \mathbb{P}$$

where

$$d_2 \;=\; (\delta \setminus x) \cup d' \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in \delta\} \; .$$

We show now that $(B' \cup A') \cap \{\gamma \mid (\gamma, y) \in d_2\} = \emptyset$. For that, we rewrite the set $\{\gamma \mid (\gamma, y) \in d_2\}$ can be rewritten as the union of the sets S_i defined as

$$S_1 = \{\gamma \mid (\gamma, y) \in (\delta \setminus x)\} \qquad S_2 = \{\gamma \mid (\gamma, y) \in d'\}$$
$$S_3 = \{\gamma \mid (\gamma, y) \in \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in \delta\}\} .$$

Now, $B' \cap S_1 = \emptyset$ because of hypothesis (5.6), and $A' \cap S_1 = \emptyset$ because of (5.7). This implies $(A' \cup B') \cap S_1 = \emptyset$. The sets S_2 and S_3 are empty because y is a bound variable and without loss of generality we can suppose $y \notin \Gamma'$.

Then, we can apply the pattern matching typing rule to obtain

$$B' \cup A' \cup B; \Delta' \cup \Gamma' \cup \Delta; d_3 \vdash [w[t/x] > p[t/x](y) \Rightarrow v[t/x]] : \mathbb{P}$$

where

$$d_3 = (d_2 \setminus y) \cup d_1 \cup \{\{\beta\} \times (B' \cup A' \cup \Delta' \cup \Gamma') \mid (\beta, y) \in d_2\} .$$

By definition of substitution $[w[t/x] > p[t/x](y) \Rightarrow v[t/x]] = [w > p(y) \Rightarrow v][t/x]$. It remains to prove that $d_3 = \overline{d}$. With a lot of patience, we see that

$$\begin{aligned}
\overline{d} &= (d \setminus x) \cup d' \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in d\} \\
&= (((\delta \setminus y) \cup \delta' \cup \{\{\beta\} \times (B' \cup (\Delta', x{:}\mathbb{Q})) \mid (\beta, y) \in \delta\}) \setminus x) \cup d' \cup \\
&\qquad \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in (\delta \setminus y)\} \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in \delta'\} \cup \\
&\qquad \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in \{\gamma \times (B' \cup (\Delta', x{:}\mathbb{Q})) \mid (\gamma, y) \in \delta\}\} \\
&= (\delta \setminus y \setminus x) \cup (\delta' \setminus x) \cup \{\beta \times (B' \cup \Delta') \mid (\beta, y) \in \delta\} \cup d' \cup \\
&\qquad \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in (\delta \setminus y)\} \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in \delta'\} \cup \\
&\qquad \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, y) \in \delta\} \\
&= (\delta \setminus y \setminus x) \cup (\delta' \setminus x) \cup d' \cup \\
&\qquad \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in (\delta \setminus y)\} \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in \delta'\} \cup \\
&\qquad \{\{\beta\} \times (B' \cup \Delta' \cup A' \cup \Gamma') \mid (\beta, y) \in \delta\}
\end{aligned}$$

In the last expression, the set $\{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in (\delta \setminus y)\}$ can be rewritten as $\{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in (\delta \setminus y)\}$. With more patience, we calculate:

$$\begin{aligned}
d_3 &= (d_2 \setminus y) \cup d_1 \cup \{\{\beta\} \times (B' \cup A' \cup \Delta' \cup \Gamma') \mid (\beta, y) \in d_2\} \\
&= (((\delta \setminus x) \cup d' \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in \delta\}) \setminus y) \cup (\delta' \setminus x) \cup d' \cup \\
&\qquad \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in \delta'\} \cup \\
&\qquad \{\{\beta\} \times (B' \cup A' \cup \Delta' \cup \Gamma') \mid (\beta, y) \in (\delta \setminus x)\} \cup \\
&\qquad \{\{\beta\} \times (B' \cup A' \cup \Delta' \cup \Gamma') \mid (\beta, y) \in d'\} \\
&\qquad \{\{\beta\} \times (B' \cup A' \cup \Delta' \cup \Gamma') \mid (\beta, y) \in \{\gamma \times (A' \cup \Gamma') \mid (\gamma, x) \in \delta\}\}
\end{aligned}$$

As $(d' \setminus y) \subseteq d'$, and since y is a bound variable an without loss of generality we can suppose $y \notin \Gamma'$ (and in turn $y \notin d'$), we have

$$\begin{aligned} d_3 = (\delta \setminus x \setminus y) \cup (\delta' \setminus x) \cup d' \cup \\ \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in \delta\} \cup \{\{\beta\} \times (A' \cup \Gamma') \mid (\beta, x) \in \delta'\} \cup \\ \{\{\beta\} \times (B' \cup A' \cup \Delta' \cup \Gamma') \mid (\beta, y) \in (\delta \setminus x)\} \ . \end{aligned}$$

We conclude that $\overline{d} = d_3$ because they are union of the same subsets.

Typing rules for patterns follow along the same lines (they are easier). The induction is now complete. □

Lemma 5.2.5 (Contraction Rules) *The rules below can be derived (also for patterns):*

$$\frac{\alpha{:}\mathbb{N}, \beta{:}\mathbb{N}, A; \Gamma; d \vdash p : \mathbb{P}}{\alpha{:}\mathbb{N}, A; \Gamma; d[\alpha/\beta] \vdash p[\alpha/\beta] : \mathbb{P}} (\alpha, \beta) \notin d \qquad \frac{A; y{:}\mathbb{Q}, z{:}\mathbb{Q}, \Gamma; d \vdash p : \mathbb{P}}{A; y{:}\mathbb{Q}, \Gamma; d[y/z] \vdash p[y/z] : \mathbb{P}}$$

5.2.3 Transition rules

The behaviour of terms is defined by a transition relation of the form

$$s \vdash t \xrightarrow{p\ (x)} t'$$

where s is a finite set of name constants such that $\mathsf{n}(t) \subseteq s$. Intuitively, the set s represents the set of the names known in the 'universe' where t is evolving. The transition above should be read as 'with current names s the term t can perform the action p and resume as t''. We generally note the action's resumption variable (enclosed in parentheses, e.g. (x)) in the transitions: this simplifies the transition rules in which the resumption variable must be explicitly manipulated.

So the transition relation is given at stages indexed by the set of current names s.

This implies that interactions with a context (with the only exception of new-name abstractions) are limited to names in s. For instance, the body of an abstraction over names $\lambda\alpha.t$ can only be instantiated with a name in s, and an abstraction over processes $\lambda x.t$ can only be instantiated with a process whose support is contained in s. As the transition relation is indexed by the current set of names, it is possible to generate new names at run-time. Indeed, the transition rule for new-name abstraction $new\alpha.t$ extends the set s of current names with a new name $a \notin s$: this name a is then passed to t via the variable α. Formally:

Definition 5.2.6 (Transition relation) *For closed terms t such that $s \vdash t : \mathbb{P}$ and path patterns such that $s; ; x{:}\mathbb{Q} \Vdash p : \mathbb{P}$ the rules reported in Table 5.3 define a relation $\mathbb{P}; s \vdash t \xrightarrow{p\ (x)} u$, called the* transition relation.

To help familiarise the reader with the transition relation, we present some fragments of derivations.

$$\frac{}{!\mathbb{P}; s \vdash !t \xrightarrow{!x\ (x)} t} \qquad \frac{\mathbb{P}; s \vdash t \xrightarrow{p} t' \quad a \in s}{\mathbb{N} \otimes \mathbb{P}; s \vdash a \cdot t \xrightarrow{a \cdot p} t'} \qquad \frac{\mathbb{N} \otimes \mathbb{P}; s \vdash t \xrightarrow{a \cdot p} t'}{\mathbb{P}; s \vdash \pi_a t \xrightarrow{p} t'}$$

$$\frac{\mathbb{P}; s \vdash t_i \xrightarrow{p\ (x)} t'}{\mathbb{P}; s \vdash \Sigma_{i \in I} t_i \xrightarrow{p\ (x)} t'} \qquad \frac{\mathbb{P}; s \vdash t[a/\alpha] \xrightarrow{p\ (x)} u \quad a \in s}{\mathbb{P}; s \vdash \Sigma_{\alpha \in \mathbb{N}} t \xrightarrow{p\ (x)} u} \qquad \frac{\mathbb{P}; s \vdash t[recy.t/y] \xrightarrow{p\ (x)} u}{\mathbb{P}; s \vdash recy.t \xrightarrow{p\ (x)} u}$$

$$\frac{\mathbb{P}_i; s \vdash t \xrightarrow{p\ (x)} t'}{\Sigma_{i \in I} \mathbb{P}_i; s \vdash i{:}t \xrightarrow{i:p\ (x)} t'} \qquad \frac{\Sigma_{i \in I} \mathbb{P}_i; s \vdash t \xrightarrow{i:p\ (x)} t'}{\mathbb{P}_i; s \vdash \pi_i t \xrightarrow{p\ (x)} t'} \qquad \frac{\mathbb{Q}; s \vdash t[u/x] \xrightarrow{p\ (x)} v \quad s \vdash u : \mathbb{P}}{\mathbb{P} \to \mathbb{Q}; s \vdash \lambda x.t \xrightarrow{u \mapsto p\ (x)} v}$$

$$\frac{\mathbb{P} \to \mathbb{Q}; s \vdash t \xrightarrow{u \mapsto p\ (x)} v}{\mathbb{Q}; s \vdash tu \xrightarrow{p\ (x)} v} \qquad \frac{\mathbb{P}; s \vdash t[a/\alpha] \xrightarrow{p\ (x)} v \quad a \in s}{\mathbb{N} \to \mathbb{P}; s \vdash \lambda \alpha.t \xrightarrow{a \mapsto p\ (x)} v} \qquad \frac{\mathbb{N} \to \mathbb{P}; s \vdash t \xrightarrow{a \mapsto p\ (x)} v}{\mathbb{P}; s \vdash ta \xrightarrow{p\ (x)} v}$$

$$\frac{\mathbb{P}; s \dot\cup \{a\} \vdash t[a/\alpha] \xrightarrow{p[a/\alpha]\ (x)} u[a/\alpha]}{\delta\mathbb{P}; s \vdash new\alpha.t \xrightarrow{new\alpha.p[x'[\alpha]/x]\ (x')} new\alpha.u} \qquad \frac{\delta\mathbb{P}; s \vdash t \xrightarrow{new\alpha.p[x'[\alpha]/x]\ (x')} u}{\mathbb{P}; s \dot\cup \{a\} \vdash t[a] \xrightarrow{p[a/\alpha]\ (x)} u[a]}$$

$$\frac{\mathbb{P}; s \vdash t \xrightarrow{p\ (x)} t' \quad \mathbb{Q}; s \vdash u[t'/x] \xrightarrow{q\ (x')} v}{\mathbb{Q}; s \vdash [t > p(x) \Rightarrow u] \xrightarrow{q\ (x')} v}$$

In the rule for new name abstraction, the conditions $a \notin \mathsf{n}(p)$ and $a \notin \mathsf{n}(u)$ must hold.

Table 5.3: new-HOPLA: transition rules

- The operational semantics validates β-equivalence:

$$\frac{\mathbb{Q}; s \vdash t[u/x] \xrightarrow{p\,(y)} t'}{\mathbb{P} \to \mathbb{Q}; s \vdash \lambda x.t \xrightarrow{u \mapsto p\,(y)} t'} \qquad \frac{\mathbb{Q}; s \vdash t[a/\alpha] \xrightarrow{p\,(y)} t'}{\mathbb{N} \to \mathbb{Q}; s \vdash \lambda \alpha.t \xrightarrow{a \mapsto p\,(y)} t'}$$

$$\frac{}{\mathbb{Q}; s \vdash (\lambda x.t)u \xrightarrow{p\,(y)} t'} \qquad \frac{}{\mathbb{Q}; s \vdash (\lambda \alpha.t)a \xrightarrow{p\,(y)} t'}$$

These derivations illustrate how β-equivalence is validated by the transition relation in the sense that a term $(\lambda x.t)u$ (resp. $(\lambda \alpha.t)a$) has the same transition capabilities as the term $t[u/x]$ (resp. $t[a/\alpha]$): for example, a term $(\lambda x.t)u$ performs an action p and resumes as t' iff the term $t[u/x]$ performs the same action p resuming as t'—the 'only if' part follows by the uniqueness of the derivations.

- The action of the tensor and sum type:

$$\frac{\mathbb{P}; s \vdash t \xrightarrow{p\,(y)} t'}{\mathbb{N} \otimes \mathbb{P}; s \vdash a \cdot t \xrightarrow{a \cdot p\,(y)} t'} \qquad \frac{\mathbb{P}_i; s \vdash t \xrightarrow{p\,(y)} t'}{\Sigma_{i \in I} \mathbb{P}_i; s \vdash i{:}t \xrightarrow{i:p\,(y)} t'}$$

$$\mathbb{P}; s \vdash \pi_a(a \cdot t) \xrightarrow{p\,(y)} t' \qquad \mathbb{P}_i; s \vdash \pi_i(i{:}t) \xrightarrow{p\,(y)} t'$$

  Tagging a term with a name, e.g. $a \cdot t$, has the effect of tagging all the computation paths p of t with the name a, e.g. $a \cdot p$. Conversely, projection over a name filters computation paths: if matching is successful projection erases the name tag. Injection and projection for sum types work along similar lines. In both cases, by uniqueness of the derivations, a term t has the same transition capabilites as the terms $\pi_a(a \cdot t)$ and $\pi_i(i{:}t)$.

- New-name abstraction and β-equivalence:

$$\frac{\mathbb{Q}; s \dot\cup \{a\} \vdash t[a/\alpha] \xrightarrow{p[a/\alpha]\,(y)} t'[a/\alpha]}{\delta\mathbb{Q}; s \vdash new\alpha.t \xrightarrow{new\alpha.\alpha \mapsto p[y'[\alpha]/y]\,(y')} new\alpha.t'}$$

$$\mathbb{Q}; s \dot\cup \{a\} \vdash (new\alpha.t)[a] \xrightarrow{p[a/\alpha]\,(y)} (new\alpha.t')[a]$$

  Again by uniqueness of the derivation, $(new\alpha.t)[a]$ and $t[a/\alpha]$ have the same transition capabilities for a fresh name a. Indeed, suppose that the set of current names is s, and that a is a name constant that does not appear in s. Transitions of the term $new\alpha.t$ are then derived from the transitions of the term $t[a/\alpha]$, that is, the term obtained by instantiating the variable α in t with a fresh name a. For instance, suppose that $s \dot\cup \{a\} \vdash t[a/\alpha] \xrightarrow{p'\,(y)} t''$. The name a can be 'factored out' from both p' and t'', that is, there exist a pattern p such that $p[a/\alpha] = p'$ and $a \notin \mathsf{n}(p)$, and a term t' such that $t'[a/\alpha] = t''$ and $a \notin \mathsf{n}(t')$. Then, abstracting the fresh name a, we obtain the transition $s \vdash new\alpha.t \xrightarrow{new\alpha.p\,[y'[\alpha]/y]} new\alpha.t'$. The need for the substitution of the resumption variable is related to typing issues and it is easier to explain together with pattern matching (see below).

  New name application has the expected action on patterns: the abstracted variable α is replaced with the fresh name a (typing constraints guarantee that the name a does not belong to the support of $new\alpha.t$ and in turn to the support of p).

- Matching the anonymous action:

$$\frac{!\mathbb{P}; s \vdash !t \xrightarrow{!x\,(x)} t \qquad \mathbb{Q}; s \vdash u[t/x] \xrightarrow{p\,(y)} u'}{\mathbb{Q}; s \vdash [!t > !x(x) \Rightarrow u] \xrightarrow{p\,(y)} u'}$$

  The derivation above illustrates the simplest case of pattern matching. The term being tested $!t$ emits the prototypical action ! and resumes as t. This matches the pattern $!x$

and the resumption t is bound to x in u, which then executes. Pattern matching allows for the testing of arbitrary actions, even if a little care is needed when new names are involved.

- Matching and new-name abstraction:

$$\frac{!\mathbb{P}; s \dot\cup \{a\} \vdash !t[a/\alpha] \xrightarrow{!x\,(x)} t[a/\alpha]}{\delta!\mathbb{P}; s \vdash new\alpha.!t \xrightarrow{new\alpha.!x[\alpha]\,(x)} new\alpha.t \qquad \mathbb{Q}; s \vdash u[new\alpha.t/x] \xrightarrow{p\,(y)} u'}$$
$$\mathbb{Q}; s \vdash [new\alpha.!t > new\alpha.!x[\alpha](x) \Rightarrow u] \xrightarrow{p\,(y)} u'$$

The continuation of a term that generates a new name and then performs some visible action, e.g. $new\alpha.!t$, invariably has the form $new\alpha.t'$ for some term t'. This implies that in case of successful match, like in the derivation above, the resumption variable x will be bound to a term of type $\delta\mathbb{R}$, for some $\mathbb{R}$. This justifies substituting the resumption variable with a resumption variable applied to a new name, both in the pattern type rule for new name abstraction type rule and in the transition rule for new name abstraction. In fact, suppose $new\alpha.!t$ has type $\delta!\mathbb{P}$: the type of its continuation is $\delta\mathbb{P}$. Then in the pattern $new\alpha.!x[\alpha]$, the type of the resumption variable must be $\delta\mathbb{P}$. The pattern $x[\alpha]$ has type $\mathbb{P}$ and consequently the whole pattern $new\alpha.!x[\alpha]$ has type $\delta!\mathbb{P}$, as the process it is supposed to test.

Remark that the new name generated in testing a pattern (above it is a) is local to the test.

We end this section by proving several properties of the transition relation. First of all, the operational rules are type correct:

Theorem 5.2.7 (Transitions preserve types) *If* $s \vdash t : \mathbb{P}$ *and* $s;;x{:}\mathbb{Q} \Vdash p : \mathbb{P}$ *and* $\mathbb{P}; s \vdash t \xrightarrow{p\ (x)} t'$*, then* $s \vdash t' : \mathbb{Q}$.

Proof Rule induction on the derivation of $\mathbb{P}; s \vdash t \xrightarrow{p\ (x)} u$.

Prefixing. Suppose that $!\mathbb{P}; s \vdash !t \xrightarrow{!x\ (x)} t$. Then $s \vdash !t : !\mathbb{P}$ and so by the typing rules, $s \vdash t : \mathbb{P}$ as wanted.

Process abstraction. Suppose that $\mathbb{P} \to \mathbb{Q}; \lambda y.t \xrightarrow{u \mapsto p\ (x)} t'$ with $s;;x{:}\mathbb{R} \vdash u \mapsto p : \mathbb{P} \to \mathbb{Q}$. By typing of patterns, we have $s \vdash u : \mathbb{P}$ and $s;;x{:}\mathbb{R} \vdash p : \mathbb{Q}$. The induction hypothesis then yields $s \vdash t' : \mathbb{R}$ as wanted. Note that the substitution $t[u/x]$ is well-formed because $A; y{:}\mathbb{Q}; A \vdash t : \mathbb{P}$ follows from $s \vdash \lambda y.t : \mathbb{P} \to \mathbb{Q}$, for a bijection $\sigma : s \to A$.

Process application. Suppose that $\mathbb{Q}; s \vdash tu \xrightarrow{p\ (x)} t'$ has been derived from $\mathbb{P} \to \mathbb{Q} : s \vdash t \xrightarrow{u \mapsto p(x)} t'$ with $s;;x{:}\mathbb{R} \vdash p(x) : \mathbb{Q}$. By the premise and the typing rules, we have $s \vdash t : \mathbb{P} \to \mathbb{Q}$ and $s \vdash u : \mathbb{P}$, such that $s;;x{:}\mathbb{R} \vdash u \mapsto p : \mathbb{P} \to \mathbb{Q}$. The induction hypothesis then yields $s \vdash t' : \mathbb{R}$ as wanted.

New name abstraction. Suppose that $\delta\mathbb{P}; s \vdash new\alpha.t \xrightarrow{new\alpha.p[x'[\alpha]/x]\ (x')} new\alpha.t'$ has been derived from $\mathbb{P}; s \dot\cup \{a\} \vdash t[a/\alpha] \xrightarrow{p[a/\alpha]\ (x)} t'[a/\alpha]$ with $s;;x'{:}\delta\mathbb{R} \vdash new\alpha.p[x'[\alpha]/x] : \delta\mathbb{P}$.

By the typing rules, we get $A, \alpha{:}\mathbb{N}; \emptyset; A, d \vdash t : \mathbb{P}$, for $\sigma : s + \{a\} \to A + \{\alpha\}$. This implies $s \dot\cup \{a\} \vdash t[a/\alpha] : \mathbb{P}$. Again, by typing rules we get $A, \alpha{:}\mathbb{N}; \emptyset; A, d; ; x{:}\mathbb{R} \vdash p : \mathbb{P}$ and then $s \dot\cup \{a\}; ; x{:}\mathbb{R} \vdash p[a/\alpha] : \mathbb{P}$. By the induction hypothesis we have $s \dot\cup \{a\} \vdash t'[a/\alpha] : \mathbb{R}$. This implies that $A, \alpha{:}\mathbb{N}; \emptyset; A, \alpha \vdash t' : \mathbb{R}$ and we get $s \vdash new\alpha.t' : \delta\mathbb{R}$ by the typing rules, as wanted.

New name application. Suppose that $\mathbb{P}; s \dot\cup \{a\} \vdash t[a] \xrightarrow{p[a/\alpha]\ (x)} t'[a]$ has been derived from $\delta\mathbb{P}; s \vdash t \xrightarrow{new\alpha.p[x'[\alpha]/x]\ (x')} t'$, with $s \dot\cup \{a\}; ; x{:}\mathbb{R} \vdash p[a/\alpha] : \mathbb{P}$. By the typing rules we get $s \vdash t : \delta\mathbb{P}$. Also, as the pattern is well-typed, $A, \alpha{:}\mathbb{N}; \emptyset; A, \alpha; ; x{:}\mathbb{R} \vdash p : \mathbb{P}$ must hold for $\sigma : s + \{a\} \to A + \{\alpha\}$. By the typing rules we get $s; ; x'{:}\delta\mathbb{R} \vdash new\alpha.p[x'[\alpha]/x] : \delta\mathbb{P}$. By the induction hypothesis, $s \vdash t' : \delta\mathbb{R}$, and by the typing rules we conclude $s \dot\cup \{a\} \vdash t'[a] : \mathbb{R}$.

Pattern matching. Suppose that $\mathbb{Q}; s \vdash [t > p(y) \Rightarrow u] \xrightarrow{q\ (x)} u'$ has been derived from $\mathbb{P}; s \vdash t \xrightarrow{p(y)} t'$ and $\mathbb{Q}; s \vdash u[t'/y] \xrightarrow{q(x)} u'$ with $s; ; y{:}\mathbb{S} \vdash p : \mathbb{P}$ and $s; ; x{:}\mathbb{R} \vdash q : \mathbb{Q}$. By the induction hypothesis applied to the first premise, we get $s \vdash t' : \mathbb{S}$. Thus, as $A; x{:}\mathbb{R}; A \vdash u : \mathbb{Q}$ for $\sigma : s \to A$, the substitution $u[t'/x]$ is well-formed. By the induction hypothesis applied to the second premise, we get $s \vdash u' : \mathbb{R}$, as wanted.

The remaining cases are handled similarly. □

An useful convention is to omit types in transitions and resumption variables in transitions when it does not cause confusion.

Some simple results follow. They can be proved by routine inductions on derivations of transition.

Lemma 5.2.8 *If* $\mathsf{n}(t) \subseteq s$ *and* $\mathbb{P}; s \vdash t \xrightarrow{p} t'$, *then* $\mathsf{n}(t') \subseteq s$.

Lemma 5.2.9 *If* $\mathbb{P}; s \vdash t \xrightarrow{p} t'$, *then* $\mathsf{n}(t) \cup \mathsf{n}(p) \cup \mathsf{n}(t') \vdash t \xrightarrow{p} t'$.

Lemma 5.2.10 (Injective renaming) *If* $\mathbb{P}; s \vdash t \xrightarrow{p} t'$ *and* $f : s \to s'$ *is injective, then* $\mathbb{P}; s' \vdash ft \xrightarrow{fp} ft'$.

Lemma 5.2.11 (Converse of injective renaming) *Let* $f : s \to s'$ *injective. If* $\mathbb{P}; s' \vdash ft \xrightarrow{p'} u'$, *then there exists* p, u *and* $g : s'' \to \mathsf{n}(p', u') \setminus \mathrm{ran}(f)$ *bijective, with* $s'' \cap s = \emptyset$, *such that* $\mathbb{P}; s, s'' \vdash t \xrightarrow{p} u$ *and* $p' = (f + g)p$ *and* $u' = (f + g)u$.

Observe that if $s' \vdash ft \xrightarrow{p'} u'$ then ft can chose nondeterministically some names in $(s' \setminus s)$ before performing p' (because of the semantics of the *sum over names*). This motivates the need for s'' in the Lemma above. As a consequence, bisimulation techniques based on injective renaming (like those of Section 3.3) cannot be applied to new-HOPLA.

5.3 Equivalences

After introducing some notations regarding relations, we explore the bisimulation equivalence that arises from the transition semantics.

A relation $\mathcal{R}$ between typing judgements is said to respect types if, whenever $\mathcal{R}$ relates $E_1 \vdash t_1 : \mathbb{P}_1$ and $E_2 \vdash t_2 : \mathbb{P}_2$, we have $E_1 = E_2$ and $\mathbb{P}_1 = \mathbb{P}_2$. We are mostly interested in relations between closed terms. We write $s \vdash t \;\mathcal{R}\; u : \mathbb{P}$ to denote $(s \vdash t : \mathbb{P}, s \vdash q : \mathbb{P}) \in \mathcal{R}$.

Definition 5.3.1 (Bisimilarity) *A type-respecting relation on closed terms, $\mathcal{R}$, is a* bisimulation *if*

1. $s \vdash t \;\mathcal{R}\; u : \mathbb{P}$ *and* $s' \vdash t \xrightarrow{p(x)} t'$ *for* $s' \supseteq s$ *imply that there exists a term* u' *such that* $s' \vdash u \xrightarrow{p(x)} u'$ *and* $s' \vdash t' \;\mathcal{R}\; u' : \mathbb{R}$*;*

2. $s \vdash t \;\mathcal{R}\; u : \mathbb{P}$ *and* $s' \vdash u \xrightarrow{p(x)} u'$ *for* $s' \supseteq s$ *imply that there exists a term* t' *such that* $s' \vdash t \xrightarrow{p(x)} t'$ *and* $s' \vdash t' \;\mathcal{R}\; u' : \mathbb{R}$*;*

where $\mathbb{R}$ is the type of the resumption variable x in p. Let bisimilarity*, denoted $\sim$, be the largest bisimulation.*

We say that two closed terms t and q are bisimilar if $s \vdash t \sim q : \mathbb{P}$ for some s and $\mathbb{P}$. We call a type-respecting relation a *simulation* if it is closed under condition 1.

In the definition of bisimulation, the universal quantification on sets of names s' is required, otherwise the resulting relation would equate

$$\{a\} \vdash \lambda\alpha.[\alpha!\mathbf{0} > a!x \Rightarrow !\mathbf{0}] : \mathbb{N} \otimes !\mathbf{0} \qquad \text{and} \qquad \{a\} \vdash \lambda\alpha.!\mathbf{0} : \mathbb{N} \otimes !\mathbf{0}$$

while the two terms above behave differently in a world where a is not the only existing name. This is reminiscent of works on Kripke λ-models.

Lemma 5.3.2 *If $s \vdash t \sim u : \mathbb{P}$, then for all $s' \supseteq s$ it holds $s' \vdash t \sim u : \mathbb{P}$.*

Proof Suppose $s \vdash t \sim u : \mathbb{P}$ and $s' \supseteq s$. We want to show that $s' \vdash t \sim u : \mathbb{P}$. For that let $s'' \supseteq s'$ and suppose $s'' \vdash t \xrightarrow{p} t'$. As $s \vdash t \sim u : \mathbb{P}$ and $s'' \supseteq s$, there exists a term u' such that $s'' \vdash u \xrightarrow{p} u'$ and $s'' \vdash t' \sim u'$, as required. The symmetric case is handled similarly. □

Lemma 5.3.3 *Bisimilarity is an equivalence relation.*

Proof It is easy to see that bisimilarity is reflexive and symmetric. For transitivity, suppose $s \vdash t \sim u : \mathbb{P}$ and $s \vdash u \sim v : \mathbb{P}$. We want to show that $s \vdash t \sim v : \mathbb{P}$. Let $s' \supseteq s$ and suppose $s' \vdash t \xrightarrow{p} t'$. As $s \vdash t \sim u : \mathbb{P}$, there exists a term u' such that $s' \vdash u \xrightarrow{t} u'$ and $s' \vdash t' \sim u' : \mathbb{P}$. In turn, as $s \vdash u \sim v : \mathbb{P}$, there exists a term v' such that $s' \vdash v \xrightarrow{t} v'$ and $s' \vdash u' \sim v' : \mathbb{P}$. The result follows by coinduction. □

5.3.1 Congruence of bisimilarity

Using an extension of Howe's method [How96] as adapted by Gordon and Pitts to a typed setting [Gor95, Pit97], we show that bisimilarity is preserved by well typed contexts.

Definition 5.3.4 (Closure) *A* $(A; \Gamma; d)$-closure *is a triple* $(s, \rho, [\vec{u}/\vec{x}])$*, where*

1. *s is a set of names;*
2. *$\rho : A \to s$ is a map such that $\rho(\alpha) \neq \rho(\beta)$ whenever $(\alpha, \beta) \in d$, and $\rho(\alpha) \notin \mathsf{n}(u_i)$ whenever $(\alpha, u_i) \in d$ (that is, ρ substitutes name constants in s for name variables in A and respects distinctions);*
3. *$[\vec{u}/\vec{x}]$ is a substitution assigning closed terms to the process variables in Γ such that $s \vdash u_i : \Gamma(x_i)$ for all i.*

Closures are ranged over by Ξ. Given a type judgement $A; \Gamma; d \vdash t : \mathbb{P}$ and a $(A; \Gamma; d)$-closure $\Xi = (s, \rho, [\vec{u}/\vec{x}])$, we write $t[\Xi]$ for the term $\rho t[\vec{u}/\vec{x}]$ and s_Ξ for s. Remark that $s \vdash \rho t[\vec{u}/\vec{x}] : \mathbb{P}$ is a valid type judgement.

Definition 5.3.5 (Open extension) *If $\mathcal{R}$ relates closed terms, we write $\mathcal{R}^\circ$ for its open extension, relating $A; \Gamma; d \vdash t : \mathbb{P}$ and $A; \Gamma; d \vdash u : \mathbb{P}$ if $s_\Xi \vdash t[\Xi] \,\mathcal{R}\, u[\Xi] : \mathbb{P}$ holds for all $(A; \Gamma; d)$-closures Ξ.*

We write $\mathcal{R}_c$ for the restriction of a type-respecting relation to closed terms. For a type-respecting relation $\mathcal{R}$ we write $\mathcal{R}$ also for the relation induced on actions, given inductively by

$$\frac{A; \Gamma; d; ; x{:}\mathbb{R} \vdash p \;\mathcal{R}\; q : \mathbb{P}}{A'; \Gamma'; d'; ; x{:}\mathbb{R} \vdash p \;\mathcal{R}\; q : \mathbb{P}} \begin{array}{l} A \subseteq A' \\ \Gamma \subseteq \Gamma' \\ d \subseteq d' \end{array} \qquad \frac{}{E; ; x{:}\mathbb{R} \vdash !x \;\mathcal{R}\; !x : !\mathbb{R}}$$

$$\frac{A; \Gamma; d; ; x{:}\mathbb{R} \vdash p \;\mathcal{R}\; q : \mathbb{P} \quad \alpha \in A}{A; \Gamma; d; ; x{:}\mathbb{R} \vdash \alpha \cdot p \;\mathcal{R}\; \alpha \cdot q : \mathbb{N} \otimes \mathbb{P}} \qquad \frac{E; ; x{:}\mathbb{R} \vdash p \;\mathcal{R}\; q : \mathbb{P}_j \quad j \in I}{E; ; x{:}\mathbb{R} \vdash j{:}p \;\mathcal{R}\; j{:}q : \Sigma_{i \in I} \mathbb{P}_i}$$

$$\frac{A; \Gamma; d; ; x{:}\mathbb{R} \vdash p \;\mathcal{R}\; q : \mathbb{P} \quad \alpha \in A}{A; \Gamma; d; ; x{:}\mathbb{R} \vdash \alpha \mapsto p \;\mathcal{R}\; \alpha \mapsto q : \mathbb{P}} \qquad \frac{A; \Gamma; d \vdash u \;\mathcal{R}\; v : \mathbb{P} \qquad A; \Gamma; d; ; x{:}\mathbb{R} \vdash p \;\mathcal{R}\; q : \mathbb{Q}}{A; \Gamma; d; ; x{:}\mathbb{R} \vdash u \mapsto p \;\mathcal{R}\; v \mapsto q : \mathbb{P}}$$

$$\frac{\alpha : \mathbb{N}, A; \Gamma; d; ; x{:}\mathbb{R} \vdash p \;\mathcal{R}\; q : \mathbb{P}}{A; \Gamma; (d \setminus \alpha); ; x'{:}\delta\mathbb{R} \vdash \mathit{new}\alpha.p[x'[\alpha]/x] \;\mathcal{R}\; \mathit{new}\alpha.q[x'[\alpha]/x] : \delta\mathbb{P}}$$

The open extension of $\sim$ is closed under weakening:

Lemma 5.3.6 *If $A; \Gamma; d \vdash t \sim^\circ u : \mathbb{P}$, then for all $A' \supseteq A, \Gamma' \supseteq \Gamma, d' \supseteq d$ we have $A'; \Gamma'; d' \vdash t \sim^\circ u : \mathbb{P}$.*

Proof Follows from the definition of open extension and from Lemma 5.3.2. □

Some terminology: a type respecting relation is said *operator respecting* if it is preserved by all the operators of the language. A *congruence* is an operator respecting relation that is also an equivalence.

Following Howe, we define an auxiliary relation, called the *precongruence candidate*, that, by construction, contains $\sim^\circ$ and is operator preserving. In what follows, we omit the $\mathbb{N}$ type in the environment of name variables, and we occasionally use E as a concise abbreviation for a typing environment $A; \Gamma; d$. Also, when no ambiguity arises, we use commas to denote the disjoint union of sets.

$$\frac{A;\Gamma;d \vdash t \mathrel{\hat{\sim}} w : \mathbb{P}}{A';\Gamma';d' \vdash t \mathrel{\hat{\sim}} w : \mathbb{P}} \begin{array}{l} A' \supseteq A \\ \Gamma' \supseteq \Gamma \\ d' \supseteq d \end{array} \qquad \frac{E \vdash t \mathrel{\hat{\sim}} w : \mathbb{P}_j[\mu\vec{P}.\vec{\mathbb{P}}/\vec{P}]}{E \vdash t \mathrel{\hat{\sim}} w : \mu_j P : \vec{\mathbb{P}}} \qquad \frac{E \vdash \mathbf{0} \sim^\circ w : \mathbb{P}}{E \vdash \mathbf{0} \mathrel{\hat{\sim}} w : \mathbb{P}} \qquad \frac{E \vdash x \sim^\circ w : \mathbb{P}}{E \vdash x \mathrel{\hat{\sim}} w : \mathbb{P}}$$

$$\frac{E \vdash t \mathrel{\hat{\sim}} t' : \mathbb{P} \qquad E \vdash !t' \sim^\circ w : !\mathbb{P}}{E \vdash !t \mathrel{\hat{\sim}} w : !\mathbb{P}} \qquad \frac{A;\Gamma,x{:}\mathbb{P};d \vdash t \mathrel{\hat{\sim}} t' : \mathbb{P} \qquad A;\Gamma;d \vdash \mathit{rec}x.t' \sim^\circ w : \mathbb{P}}{A;\Gamma;d \vdash \mathit{rec}x.t \mathrel{\hat{\sim}} w : \mathbb{P}}$$

$$\frac{E \vdash t \mathrel{\hat{\sim}} t' : \mathbb{P} \qquad E \vdash n \cdot t' \sim^\circ w : \mathbb{N} \otimes \mathbb{P}}{E \vdash n \cdot t \mathrel{\hat{\sim}} w : \mathbb{N} \otimes \mathbb{P}} \qquad \frac{E \vdash t \mathrel{\hat{\sim}} t' : \mathbb{N} \otimes \mathbb{P} \qquad E \vdash \pi_n t' \sim^\circ w : \mathbb{P}}{E \vdash \pi_n t \mathrel{\hat{\sim}} w : \mathbb{P}}$$

$$\frac{A;\Gamma,x{:}\mathbb{P};d \vdash t \mathrel{\hat{\sim}} t' : \mathbb{Q} \qquad A;\Gamma;d \vdash \lambda x.t' \sim^\circ w : \mathbb{P} \to \mathbb{Q}}{A;\Gamma;d \vdash \lambda x.t \mathrel{\hat{\sim}} w : \mathbb{P} \to \mathbb{Q}}$$

$$\frac{E \vdash t \mathrel{\hat{\sim}} t' : \mathbb{P} \to \mathbb{Q} \qquad E \vdash u \mathrel{\hat{\sim}} u' : \mathbb{P} \qquad E \vdash t'u' \sim^\circ w : \mathbb{P}}{E \vdash tu \mathrel{\hat{\sim}} w : \mathbb{P}}$$

$$\frac{A,\alpha;\Gamma;d \vdash t \mathrel{\hat{\sim}} t' : \mathbb{P} \quad A;\Gamma;d \vdash \lambda\alpha.t' \sim^\circ w : \mathbb{N} \to \mathbb{P}}{A;\Gamma;d \vdash \lambda\alpha.t \mathrel{\hat{\sim}} w : \mathbb{N} \to \mathbb{P}} \qquad \frac{E \vdash t \mathrel{\hat{\sim}} t' : \mathbb{N} \to \mathbb{P} \quad E \vdash t'\alpha \sim^\circ w : \mathbb{P}}{E \vdash t\alpha \mathrel{\hat{\sim}} w : \mathbb{P}}$$

$$\frac{A,\beta;\Gamma;d,(\{\beta\} \times (A,\Gamma)) \vdash t[\beta/\alpha] \mathrel{\hat{\sim}} t'[\beta/\alpha] : \mathbb{P} \qquad A;\Gamma;d \vdash \mathit{new}\alpha.t' \sim^\circ w : \delta\mathbb{P}}{A;\Gamma;d \vdash \mathit{new}\alpha.t \mathrel{\hat{\sim}} w : \delta\mathbb{P}}$$

$$\frac{A;\Gamma;d \vdash t \mathrel{\hat{\sim}} t' : \delta\mathbb{P} \qquad A,\alpha;\Gamma;d \cup (\{\alpha\} \times (A,\Gamma)) \vdash t'[\alpha] \sim^\circ w : \mathbb{P}}{A,\alpha;\Gamma;d \cup (\{\alpha\} \times (A,\Gamma)) \vdash t[\alpha] \mathrel{\hat{\sim}} w : \mathbb{P}}$$

$$\frac{E \vdash t \mathrel{\hat{\sim}} t' : \mathbb{P}_i \qquad E \vdash i{:}t' \sim^\circ w : \Sigma_i\mathbb{P}_i}{E \vdash i{:}t \mathrel{\hat{\sim}} w : \Sigma_i\mathbb{P}_i} \qquad \frac{E \vdash t \mathrel{\hat{\sim}} t' : \Sigma_{i\in I}\mathbb{P}_i \qquad E \vdash \pi_i t' \sim^\circ w : \mathbb{P}_i}{E \vdash \pi_i t \mathrel{\hat{\sim}} w : \mathbb{P}_i}$$

$$\frac{\forall i \quad E \vdash t_i \mathrel{\hat{\sim}} t_i' : \mathbb{P} \quad E \vdash \Sigma_{i\in I} t_i' \sim^\circ w : \mathbb{P}}{E \vdash \Sigma_{i\in I} t_i \mathrel{\hat{\sim}} w : \mathbb{P}} \qquad \frac{A,\alpha;\Gamma;d \vdash t \mathrel{\hat{\sim}} t' : \mathbb{P} \quad A;\Gamma;d \vdash \Sigma_{\alpha\in\mathbb{N}} t' \sim^\circ w : \mathbb{P}}{A;\Gamma;d \vdash \Sigma_{\alpha\in\mathbb{N}} t \mathrel{\hat{\sim}} w : \mathbb{P}}$$

$$\frac{A';\Gamma';d' \vdash t \mathrel{\hat{\sim}} t' : \mathbb{P} \quad A;\Gamma,x{:}\mathbb{R};d \vdash u \mathrel{\hat{\sim}} u' : \mathbb{Q} \quad A \cup A';\Gamma \cup \Gamma';\overline{d} \vdash [t' > p(x) \Rightarrow u'] \sim^\circ w : \mathbb{Q}}{A \cup A';\Gamma \cup \Gamma';\overline{d} \vdash [t > p(x) \Rightarrow u] \mathrel{\hat{\sim}} w : \mathbb{Q}}$$

In the last rule, $\overline{d}$ stands for $(d \setminus x) \cup d' \cup \{\{\alpha\} \times (A',\Gamma') \mid (\alpha,x) \in d\}$. Also, the pattern p is supposed to be well-type, with resumption type $\mathbb{R}$.
In the rule for new-name abstraction, we assume $A,\alpha;\Gamma;d' \vdash t : \mathbb{P}$, with $d = d' \setminus \alpha$. In particular, this implies that $\alpha \notin A$.

Figure 5.1: The precongruence candidate

Definition 5.3.7 (The precongruence candidate) *The* precongruence candidate, $\hat{\sim}$, *is the smallest type-respecting relation closed under the rules reported in Figure 5.1.*

We need several technical lemmas. The lemma below is fundamental to show that in a derivation of $s \vdash new\alpha.t \;\hat{\sim}_c\; u : \delta\mathbb{P}$ the choice of the name a used to derive $s \dot{\cup} \{a\} \vdash t[a/\alpha] \;\hat{\sim}_c\; v : \mathbb{P}$ is irrelevant, as far as it is fresh.

Lemma 5.3.8 *If there is a derivation of $A, \alpha; \Gamma; d, (\{\alpha\} \times (A \cup \Gamma)) \vdash t \;\hat{\sim}\; u : \mathbb{P}$, then there is a derivation of the same height of $A, \beta; \Gamma; d, (\{\beta\} \times (A, \Gamma)) \vdash t[\beta/\alpha] \;\hat{\sim}\; u[\beta/\alpha] : \mathbb{P}$.*

We then prove that the precongruence candidate is closed under substitutions.

Lemma 5.3.9

1. *if $A; \Gamma, x{:}\mathbb{Q}; d \vdash u \;\hat{\sim}\; u' : \mathbb{P}$ and $A'; \Gamma'; d' \vdash t \;\hat{\sim}\; t' : \mathbb{Q}$ with $A' \cap \{\gamma \mid (\gamma, x) \in d\} = \emptyset$, then $A \cup A'; \Gamma \cup \Gamma'; \overline{d} \vdash u[t/x] \;\hat{\sim}\; u'[t'/x] : \mathbb{P}$, where $\overline{d} = (d \setminus x) \cup d' \cup \{\{\beta\} \times (A', \Gamma') \mid (\beta, x) \in d\}$;*
2. *if $A, \alpha; \Gamma; d \vdash u \;\hat{\sim}\; u' : \mathbb{P}$, and $N = \{\gamma \mid (\alpha, \gamma) \in d\}$, then for all name variables $\beta \in (A \setminus N)$ it holds $A; \Gamma; d[\beta/\alpha] \vdash u[\beta/\alpha] \;\hat{\sim}\; u'[\beta/\alpha] : \mathbb{P}$.*

Proof Both parts are proved by induction on the depth of the derivation respectively of $A; \Gamma, x{:}\mathbb{P}; d \vdash u \;\hat{\sim}\; u' : \mathbb{P}$ and $A, \alpha; \Gamma; d \vdash u \;\hat{\sim}\; u' : \mathbb{P}$.

To illustrate the proof, we focus on part 1, and we detail the case when the last rule of the derivation is new-name abstraction. Suppose that the conclusion of the derivation is $A; \Gamma, x{:}\mathbb{Q}; d \vdash new\alpha.u \;\hat{\sim}\; u' : \delta\mathbb{P}$. This must have been derived from $A, \beta; \Gamma, x{:}\mathbb{Q}; d, (\{\beta\} \times (A \cup (\Gamma, x{:}\mathbb{Q}))) \vdash u[\beta/\alpha] \;\hat{\sim}\; v[\beta/\alpha] : \mathbb{P}$ and $A; \Gamma, x{:}\mathbb{Q}; d \vdash new\alpha.v \sim^{\circ} u' : \delta\mathbb{P}$ for some term v and for some fresh name β. As α is bound in $new\alpha.u$, we assume without loss of generality that $\alpha \notin A'$. More interestingly, as a consequence of Lemma 5.3.8, we can also suppose $\beta \notin A'$. By the induction hypothesis we get $(A, \beta) \cup A'; \Gamma \cup \Gamma'; d_1 \vdash u[\beta/\alpha][t/x] \;\hat{\sim}\; v[\beta/\alpha][t'/x] : \mathbb{P}$, where $d_1 = ((d, (\{\beta\} \times (A \cup (\Gamma, x{:}\mathbb{Q})))) \setminus x) \cup d' \cup \{\{\alpha\} \times (A' \cup \Gamma') \mid (\alpha, x) \in (d, (\beta \times (A \cup (\Gamma, x{:}\mathbb{Q}))))\}$. The set of distinctions d' can be rearranged so that it is of the form $d_1 = d_2, (\{\beta\} \times (A \cup A' \cup \Gamma \cup \Gamma'))$ for a set of distinctions d_2. Then we have $(A \cup A'), \beta; \Gamma \cup \Gamma'; d_1 \vdash u[t/x][\beta/\alpha] \;\hat{\sim}\; v[t'/x][\beta/\alpha] : \mathbb{P}$. Since $\sim^{\circ}$ is defined as the open extension of $\sim$, and because of Lemma 5.3.6, it holds $A \cup A'; \Gamma \cup \Gamma'; d_2 \vdash (new\alpha.v)[t/x] \sim^{\circ} u'[t'/x] : \delta\mathbb{P}$ and hence $A; \Gamma; \overline{d} \vdash new\alpha.(v[t/x]) \sim^{\circ} u'[t'/x] : \delta\mathbb{P}$, where $\overline{d} = d_2 \setminus \alpha$. We conclude $A \cup A'; \Gamma \cup \Gamma'; \overline{d} \vdash (new\alpha.u)[t/x] \;\hat{\sim}\; u'[t'/x] : \delta\mathbb{P}$. □

Part 2 of Lemma 5.3.9 (closure under name substitutions) allows us to prove some basic properties of the precongruence candidate. The proof of these properties involves the closure under name substitution to deal with the rule of new-name abstraction: we will detail one case to illustrate the proof strategy.

Lemma 5.3.10 *Some properties of the precongruence candidate:*

1. *$\hat{\sim}$ is reflexive;*
2. *$\hat{\sim}$ is operator respecting;*
3. *$\sim^{\circ} \subseteq \hat{\sim}$;*
4. *if $E \vdash t \;\hat{\sim}\; u : \mathbb{P}$ and $E \vdash u \sim^{\circ} v : \mathbb{P}$ then $E \vdash t \;\hat{\sim}\; v : \mathbb{P}$.*

Proof

1. follows from reflexivity of $\sim^\circ$ (by induction on the structure of the term t).

2. follows from the definition of $\hat{\sim}$, from the definition of operator respecting relation, and from reflexivity of $\sim^\circ$. The case of new-name abstraction deserves to be detailed. Suppose $A, \alpha; \Gamma; d \vdash t \;\hat{\sim}\; u : \mathbb{P}$. We want to conclude that $A; \Gamma; (d \setminus \alpha) \vdash new\alpha.t \;\hat{\sim}\; new\alpha.u : \delta\mathbb{P}$. As $A, \alpha; \Gamma; d \vdash t \;\hat{\sim}\; u : \mathbb{P}$, by Lemma 5.3.6 we have $A, \alpha, \beta; \Gamma; d, (\{\beta\} \times (A \cup \Gamma)) \vdash t \;\hat{\sim}\; u : \mathbb{P}$. Then, by Lemma 5.3.9 we have $A, \beta; \Gamma; (d, (\{\beta\} \times (A \cup \Gamma)))[\beta/\alpha] \vdash t[\beta/\alpha] \;\hat{\sim}\; u[\beta/\alpha] : \mathbb{P}$. Now, $(d \cup (\{\beta\} \times (A \cup \Gamma)))[\beta/\alpha] = (d \setminus \alpha) \cup (\{\beta\} \times (A \cup \Gamma))$. As $\sim^\circ$ is reflexive, $A; \Gamma; (d \setminus \alpha) \vdash new\alpha.u \sim^\circ new\alpha.u : \delta\mathbb{P}$. Hence $A; \Gamma; (d \setminus \alpha) \vdash new\alpha.t \;\hat{\sim}\; new\alpha.u : \delta\mathbb{P}$ follows from the definition of $\hat{\sim}$.

3. follows from the reflexivity of $\hat{\sim}$ and the definition of $\hat{\sim}$.

4. induction on the derivation of $E \vdash t \;\hat{\sim}\; u$, using the fact that $\sim$ (and $\sim^\circ$) is transitive. □

Lemma 5.3.11 *If $s \vdash t \;\hat{\sim}_c\; u : \mathbb{P}$, then for all $s' \supseteq s$ we have $s' \vdash t \;\hat{\sim}_c\; u : \mathbb{P}$.*

Proof Consequence of the weakening rule in the definition of the precongruence candidate. □

Proposition 5.3.12 *Since $\sim$ is an equivalence relation, the transitive closure $\hat{\sim}^*$ of $\hat{\sim}$ is symmetric, and therefore so is ${\hat{\sim}_c}^*$.*

In the next lemma, we heavily rely on the correspondence between the type judgement $s \vdash t : \mathbb{P}$ and the judgement $A; \emptyset; A \vdash \sigma t : \mathbb{P}$ for A a set of fresh name variables and $\sigma : s \to A$ a bijection between s and A.

Lemma 5.3.13 *$\hat{\sim}_c$ is a simulation.*

Proof We prove that $\hat{\sim}_c$ is a simulation by induction on the derivations of the operational semantics. Actually, we prove a stronger property:

> if $s \vdash t \;\hat{\sim}_c\; u : \mathbb{P}$ and $s' \vdash t \xrightarrow{p} t'$ for some $s' \supseteq s$, then for all p' with $s'; ; x{:}\mathbb{R} \vdash p \;\hat{\sim}\; p' : \mathbb{P}$, there exists a term u' such that $s' \vdash u \xrightarrow{p'} u'$ and $s' \vdash t' \;\hat{\sim}_c\; u'$.

Since $\hat{\sim}$ is reflexive, $s'; ; x{:}\mathbb{R} \vdash p \;\hat{\sim}\; p : \mathbb{P}$ for all actions, and so $\hat{\sim}_c$ is a simulation if the above holds. This stronger induction hypothesis is needed in the case of process application.

Most of the cases are proved in the same way. Consider $E \vdash t : \mathbb{P}$ and $s \vdash C(t) \sim_c u : \mathbb{Q}$ for some term constructor C, possibly involving binding. From the definition of $\hat{\sim}$ we obtain the existence of a term v with $E \vdash t \;\hat{\sim}\; v : \mathbb{P}$ and $s \vdash C(v) \sim u : \mathbb{Q}$. Under the assumption $s' \vdash C(t) \xrightarrow{p(x)} t'$ with $s' \supseteq s$ and with q any action such that $s'; ; x{:}\mathbb{R} \vdash p \sim q : \mathbb{Q}$, we show that there is a transition $s' \vdash C(v) \xrightarrow{q(x)} v'$ with $s' \vdash t' \;\hat{\sim}_c\; v' : \mathbb{R}$. Having showed this, *in all cases* we conclude as follows: since $s \vdash C(v) \sim u : \mathbb{Q}$, there is a transition $s' \vdash u \xrightarrow{q(x)} u'$

with $s' \vdash v' \sim u'$. Hence $s' \vdash t' \mathrel{\hat{\sim}_c} u' : \mathbb{R}$ follows from $s' \vdash t' \mathrel{\hat{\sim}_c} v' : \mathbb{R}$ by Lemma 5.3.10.4. To avoid repetition, this latter part will be left out below.

Cases *sum*, *process application*, and *pattern matching* differ from the above pattern because the constructor C takes more than one term: apart from this, their proof follows the aforementioned pattern.

Prototypical action (prefixing). Suppose $s \vdash !t \mathrel{\hat{\sim}_c} u : !\mathbb{P}$, and $s' \vdash !t \xrightarrow{!(x)} t$ for $s' \supseteq s$. Since $s \vdash !t \mathrel{\hat{\sim}_c} u : \mathbb{P}$ there exists a term v such that $s \vdash t \mathrel{\hat{\sim}_c} v : \mathbb{P}$ and $s \vdash !v \sim u : !\mathbb{P}$. By Corollary 5.3.11 we have $s' \vdash t \mathrel{\hat{\sim}_c} v : \mathbb{P}$. We get a transition $s' \vdash !v \xrightarrow{!(x)} v$ from the operational rules.

Tensor. Suppose $s \vdash a \cdot t \mathrel{\hat{\sim}_c} u : \mathbb{N} \otimes \mathbb{P}$ and that $s' \vdash a \cdot t \xrightarrow{a \cdot p(x)} t'$ because $s' \vdash t \xrightarrow{p(x)} t'$ for some $s' \supseteq s$. Let q be any action such that $s';;x{:}\mathbb{R} \vdash p \mathrel{\hat{\sim}} q : \mathbb{P}$. This implies $s';;x{:}\mathbb{R} \vdash a \cdot p \mathrel{\hat{\sim}} a \cdot q : \mathbb{N} \otimes \mathbb{P}$. Since $s \vdash a \cdot t \mathrel{\hat{\sim}_c} u : \mathbb{N} \otimes \mathbb{P}$ there exists a term v such that $s \vdash t \mathrel{\hat{\sim}_c} v : \mathbb{P}$ and $s \vdash a \cdot v \sim u : \mathbb{N} \otimes \mathbb{P}$. By the induction hypothesis we get $s' \vdash v \xrightarrow{q(x)} v'$ with $s' \vdash t' \mathrel{\hat{\sim}_c} v' : \mathbb{R}$, and hence $s' \vdash a \cdot v \xrightarrow{a \cdot q(x)} v'$.

Projection over names. Suppose $s \vdash \pi_a t \mathrel{\hat{\sim}_c} u : \mathbb{P}$ and that $s' \vdash \pi_a t \xrightarrow{p(x)} t'$ because $s' \vdash t \xrightarrow{a \cdot p(x)} t'$ for some $s' \supseteq s$. Let q be any action such that $s';;x{:}\mathbb{R} \vdash p \mathrel{\hat{\sim}} q : \mathbb{P}$. This implies $s';;x{:}\mathbb{R} \vdash a \cdot p \mathrel{\hat{\sim}} a \cdot q : \mathbb{N} \otimes \mathbb{P}$. Since $s \vdash \pi_a t \mathrel{\hat{\sim}_c} u : \mathbb{P}$ there exists a term v such that $s \vdash t \mathrel{\hat{\sim}_c} v : \mathbb{N} \otimes \mathbb{P}$ and $s \vdash \pi_a v \sim u : \mathbb{P}$. By the induction hypothesis we get $s' \vdash v \xrightarrow{a \cdot q(x)} v'$ with $s' \vdash t' \mathrel{\hat{\sim}_c} v' : \mathbb{R}$, and hence $s' \vdash \pi_a v \xrightarrow{q(x)} v'$.

Sum. Suppose $s \vdash \Sigma_{i \in I} t_i \mathrel{\hat{\sim}_c} u : \mathbb{P}$ and that $s' \vdash \Sigma_{i \in I} t_i \xrightarrow{p(x)} t'$ is derived from $s' \vdash t_i \xrightarrow{p(x)} t'$, for some $s' \supseteq s$. Let q be any action such that $s';;x{:}\mathbb{R} \vdash p \mathrel{\hat{\sim}} q : \mathbb{P}$. Since $s \vdash \Sigma_{i \in I} t_i \mathrel{\hat{\sim}_c} u : \mathbb{P}$, there is a family of terms $\{v_i\}_{i \in I}$ such that $s \vdash t_i \mathrel{\hat{\sim}_c} v_i : \mathbb{P}$ for each $i \in I$, and $s \vdash \Sigma_{i \in I} v_i \sim u : \mathbb{P}$. By the induction hypothesis we get $s' \vdash v_i \xrightarrow{q(x)} v'$ with $s' \vdash t' \mathrel{\hat{\sim}_c} v' : \mathbb{R}$, and hence $s' \vdash \Sigma_{i \in I} v_i \xrightarrow{q(x)} v'$.

Sum over names. Suppose $s \vdash \Sigma_{\alpha \in \mathbb{N}} t \mathrel{\hat{\sim}_c} u : \mathbb{P}$ and that $s' \vdash \Sigma_{\alpha \in \mathbb{N}} t \xrightarrow{p(x)} t'$ is derived from $s' \vdash t[a/\alpha] \xrightarrow{p(x)} t'$ where $a \in s'$, for some $s' \supseteq s$. Let q be any action such that $s';;x{:}\mathbb{R} \vdash p \mathrel{\hat{\sim}} q : \mathbb{P}$. Since $s \vdash \Sigma_{\alpha \in \mathbb{N}} t \mathrel{\hat{\sim}_c} u : \mathbb{P}$, there exists a term v such that $A, \alpha; \emptyset; A \vdash \sigma t \mathrel{\hat{\sim}} \sigma v : \mathbb{P}$ for a bijection $\sigma : s \to A$, and $s \vdash \Sigma_{\alpha \in \mathbb{N}} v \sim u : \mathbb{P}$. Using weakening and Lemma 5.3.9.2, we obtain $s' \vdash t[a/\alpha] \mathrel{\hat{\sim}_c} v[a/\alpha]$. By the induction hypothesis we get $s' \vdash v[a/\alpha] \xrightarrow{q(x)} v'$ with $s' \vdash t' \mathrel{\hat{\sim}_c} v' : \mathbb{R}$, and hence $s' \vdash \Sigma_{\alpha \in \mathbb{N}} v \xrightarrow{q(x)} v'$.

Recursion. Suppose $s \vdash \mathit{recy}.t \mathrel{\hat{\sim}_c} u : \mathbb{P}$ and that $s' \vdash \mathit{recy}.t \xrightarrow{p(x)} t'$ is derived from $s' \vdash t[\mathit{recy}.t/y] \xrightarrow{p(x)} t'$. Let q be any action with $s';;x{:}\mathbb{R} \vdash p \mathrel{\hat{\sim}} q : \mathbb{P}$. Since $s \vdash \mathit{recy}.t \mathrel{\hat{\sim}_c} u : \mathbb{P}$ there exists a term v such that $A; y{:}\mathbb{P}; A \vdash \sigma t \mathrel{\hat{\sim}} \sigma v : \mathbb{Q}$ for a bijection $\sigma : s \to A$, and $s \vdash \mathit{recy}.v \sim u : \mathbb{P} \to \mathbb{Q}$. As $\hat{\sim}$ is operator respecting, we have $s \vdash \mathit{recy}.t \mathrel{\hat{\sim}_c} \mathit{recy}.v : \mathbb{P}$, and using Lemma 5.3.9.1 we obtain $s \vdash t[\mathit{recy}.t/y] \mathrel{\hat{\sim}_c} v[\mathit{recy}.v/y] : \mathbb{P}$. By the induction hypothesis we get $s' \vdash v[\mathit{recy}.v/y] \xrightarrow{q(x)} v'$ with $s' \vdash t' \mathrel{\hat{\sim}_c} v' : \mathbb{R}$, and hence also $s' \vdash \mathit{recy}.v \xrightarrow{q(x)} v'$.

Process abstraction. Suppose $s \vdash \lambda y.t \mathrel{\hat{\sim}_c} u : \mathbb{P} \to \mathbb{Q}$ and that $s' \vdash \lambda y.t \xrightarrow{w_1 \mapsto p(x)} t'$ is derived from $s' \vdash t[w_1/y] \xrightarrow{p(x)} t'$. Let $w_2 \mapsto q$ be any action with $s';;x{:}\mathbb{R} \vdash w_1 \mapsto p \mathrel{\hat{\sim}} w_2 \mapsto q : \mathbb{P} \to \mathbb{Q}$. This implies $s' \vdash w_1 \mathrel{\hat{\sim}_c} w_2 : \mathbb{P}$. Since $s \vdash \lambda y.t \mathrel{\hat{\sim}_c} u : \mathbb{P} \to \mathbb{Q}$ there exists a term v such that $A; y{:}\mathbb{P}; A \vdash \sigma t \mathrel{\hat{\sim}} \sigma v : \mathbb{Q}$ for $\sigma : s \to A$, and $s \vdash \lambda y.v \sim u : \mathbb{P} \to \mathbb{Q}$. Using weakening and Lemma 5.3.9.1 we have $s' \vdash t[w_1/y] \mathrel{\hat{\sim}_c} v[w_2/y] : \mathbb{Q}$. By the induction hypothesis we get $s' \vdash v[w_2/y] \xrightarrow{q(x)} v'$ with $s' \vdash t' \mathrel{\hat{\sim}_c} v' : \mathbb{R}$, and hence also $s' \vdash \lambda y.v \xrightarrow{w_2 \mapsto q(x)} v'$.

Process application. Suppose $s \vdash t_1 t_2 \mathrel{\hat{\sim}_c} u : \mathbb{Q}$ and that $s' \vdash t_1 t_2 \xrightarrow{p(x)} t'$ is derived from $s' \vdash t_1 \xrightarrow{t_2 \mapsto p(x)} t'$ for some $s' \supseteq s$. Let q be any action such that $s';;x{:}\mathbb{R} \vdash p \mathrel{\hat{\sim}} q : \mathbb{Q}$. Since $s \vdash t_1 t_2 \mathrel{\hat{\sim}_c} u : \mathbb{Q}$ there exists terms v_1 and v_2 such that $s \vdash t_1 \mathrel{\hat{\sim}_c} v_1 : \mathbb{P} \to \mathbb{Q}$ and $s \vdash t_2 \mathrel{\hat{\sim}_c} v_2 : \mathbb{P}$ and $s \vdash v_1 v_2 \sim u : \mathbb{Q}$. By the induction hypothesis we get $s' \vdash v_1 \xrightarrow{v_2 \mapsto q(x)} v'$ with $s' \vdash t' \mathrel{\hat{\sim}_c} v' : \mathbb{R}$, and hence $s' \vdash v_1 v_2 \xrightarrow{v_2 \mapsto q(x)} v'$. Notice how the stronger induction hypothesis allows us to choose the label $v_2 \mapsto q$ rather than $u_2 \mapsto p$, so that we could obtain a transition from $v_1 v_2$.

Name abstraction. Suppose $s \vdash \lambda\alpha.t \mathrel{\hat{\sim}_c} u : \mathbb{N} \to \mathbb{Q}$ and that $s' \vdash \lambda\alpha.t \xrightarrow{a \mapsto p(x)} t'$ is derived from $s' \vdash t[a/\alpha] \xrightarrow{p(x)} t'$. Let $a \mapsto q$ be any action with $s';;x{:}\mathbb{R} \vdash a \mapsto p \mathrel{\hat{\sim}} a \mapsto q : \mathbb{N} \to \mathbb{Q}$. Since $s \vdash \lambda\alpha.t \mathrel{\hat{\sim}_c} u : \mathbb{N} \to \mathbb{Q}$ there exists a term v such that $A, \alpha; \emptyset; A \vdash \sigma t \mathrel{\hat{\sim}} \sigma v : \mathbb{Q}$ for a bijection $\sigma : s \to A$, and $s \vdash \lambda\alpha.v \sim u : \mathbb{N} \to \mathbb{Q}$. Using weakening and Lemma 5.3.9.2 we have $s' \vdash t[a/\alpha] \mathrel{\hat{\sim}_c} v[a/\alpha] : \mathbb{Q}$. By the induction hypothesis we get $s' \vdash v[a/\alpha] \xrightarrow{q(x)} v'$ with $s' \vdash t' \mathrel{\hat{\sim}_c} v' : \mathbb{R}$, and hence also $s' \vdash \lambda\alpha.v \xrightarrow{a \mapsto q(x)} v'$.

Name application. Suppose $s \vdash ta \mathrel{\hat{\sim}_c} u : \mathbb{Q}$ and that $s' \vdash ta \xrightarrow{p(x)} t'$ is derived from $s' \vdash t \xrightarrow{a \mapsto p(x)} t'$ for some $s' \supseteq s$. Let q be any action such that $s';;x{:}\mathbb{R} \vdash p \mathrel{\hat{\sim}} q : \mathbb{Q}$. Since $s \vdash ta \mathrel{\hat{\sim}_c} u : \mathbb{Q}$ there exists a term v such that $s \vdash t \mathrel{\hat{\sim}_c} v : \mathbb{N} \to \mathbb{Q}$ and $s \vdash va \sim u : \mathbb{Q}$. By the induction hypothesis we get $s' \vdash v \xrightarrow{a \mapsto q(x)} v'$ with $s' \vdash t' \mathrel{\hat{\sim}_c} v' : \mathbb{R}$, and hence $s' \vdash va \xrightarrow{a \mapsto q(x)} v'$.

Injection. Suppose $s \vdash i{:}t \mathrel{\hat{\sim}_c} u : \Sigma_{i \in I} \mathbb{P}_i$ and that $s' \vdash i{:}t \xrightarrow{i:p(x)} t'$ because $s' \vdash t \xrightarrow{p(x)} t'$ for some $s' \supseteq s$. Let q be any action such that $s';;x{:}\mathbb{R} \vdash p \mathrel{\hat{\sim}} q : \mathbb{P}_i$. This implies $s';;x{:}\mathbb{R} \vdash i{:}p \mathrel{\hat{\sim}} i{:}q : \Sigma_{i \in I} \mathbb{P}_i$. Since $s \vdash i{:}t \mathrel{\hat{\sim}_c} u : \Sigma_{i \in i} \mathbb{P}_i$ there exists a term v such that $s \vdash t \mathrel{\hat{\sim}_c} v : \mathbb{P}_i$ and $s \vdash i{:}v \sim u : \Sigma_{i \in I} \mathbb{P}_i$. By the induction hypothesis we get $s' \vdash v \xrightarrow{q(x)} v'$ with $s' \vdash t' \mathrel{\hat{\sim}_c} v' : \mathbb{R}$, and hence $s' \vdash a \cdot v \xrightarrow{i:q(x)} v'$.

Projection. Suppose $s \vdash \pi_i t \mathrel{\hat{\sim}_c} u : \mathbb{P}_i$ and that $s' \vdash \pi_i t \xrightarrow{p(x)} t'$ because $s' \vdash t \xrightarrow{i:p(x)} t'$ for some $s' \supseteq s$. Let q be any action such that $s';;x{:}\mathbb{R} \vdash p \mathrel{\hat{\sim}} q : \mathbb{P}_i$. This implies $s';;x{:}\mathbb{R} \vdash i{:}p \mathrel{\hat{\sim}} i{:}q : \Sigma_{i \in I} \mathbb{P}_i$. Since $s \vdash \pi_i t \mathrel{\hat{\sim}_c} u : \mathbb{P}_i$ there exists a term v such that $s \vdash t \mathrel{\hat{\sim}_c} v : \Sigma_{i \in I} \mathbb{P}_i$ and $s \vdash \pi_i v \sim u : \mathbb{P}_i$. By the induction hypothesis we get $s' \vdash v \xrightarrow{i:q(x)} v'$ with $s' \vdash t' \mathrel{\hat{\sim}_c} v' : \mathbb{R}$, and hence $s' \vdash \pi_i v \xrightarrow{q(x)} v'$.

New-name abstraction. Suppose $s \vdash new\alpha.t \mathrel{\hat{\sim}_c} u : \delta\mathbb{P}$. Suppose also that $s' \vdash new\alpha.t \xrightarrow{new\alpha.p[x'[\alpha]/x](x')} new\alpha.t'$ is derived from $s' \cup \{a\} \vdash t[a/\alpha] \xrightarrow{p[a/\alpha](x)} t'[a/\alpha]$ for

some $s' \supseteq s$. Let q be any action such that $A', \alpha; \emptyset; A', d; ; x : \mathbb{R} \vdash \sigma p \hat{\sim} \sigma q : \mathbb{P}$ for a bijection $\sigma : s' \to A'$. This implies $s' \dot{\cup} \{a\}; ; x{:}\mathbb{R} \vdash p[a/\alpha] \hat{\sim} q[a/\alpha] : \mathbb{P}$ and $s'; ; x'{:}\delta\mathbb{R} \vdash new\alpha.p \hat{\sim} new\alpha.q : \delta\mathbb{P}$. Since $s \vdash new\alpha.t \hat{\sim}_c u : \delta\mathbb{P}$ there exists a term v such that $s \dot{\cup} \{a\} \vdash t[a/\alpha] \hat{\sim}_c v[a/\alpha] : \mathbb{P}$ and $s \vdash new\alpha.v \sim u : \delta\mathbb{P}$. As $s' \cup \{a\} \vdash t[a/\alpha] \xrightarrow{p[a/\alpha](x)} t'[a/\alpha]$, by the induction hypothesis we have $s' \cup \{a\} \vdash v[a/\alpha] \xrightarrow{q[a/\alpha](x)} v'[a/\alpha]$ with $s' \dot{\cup} \{a\} \vdash t'[a/\alpha] \hat{\sim}_c v'[a/\alpha] : \mathbb{P}$. By the operational rules $s' \vdash new\alpha.v \xrightarrow{new\alpha.q[x'[\alpha]/x](x')} new\alpha.v'$. As $\sim$ is reflexive, we can deduce $s' \vdash new\alpha.t' \hat{\sim}_c new\alpha.v' : \delta\mathbb{P}$ from $s' \dot{\cup} \{a\} \vdash t'[a/\alpha] \hat{\sim}_c v'[a/\alpha] : \mathbb{P}$, as desired.

New name application. Suppose $s \dot{\cup} \{a\} \vdash t[a] \hat{\sim}_c u : \mathbb{P}$. Suppose also that $s' \dot{\cup} \{a\} \vdash t[a] \xrightarrow{p[a/\alpha](x)} t'[a]$ is derived from $s' \vdash t \xrightarrow{new\alpha.p[x'[\alpha]/x](x')} t'$, for some $s' \supseteq s$. Let q be any action such that $A', \alpha; \emptyset; A', d; ; x : \mathbb{R} \vdash \sigma p \hat{\sim} \sigma q : \mathbb{P}$ for a bijection $\sigma : s' \to A'$. This implies $s'; ; x' : \delta\mathbb{R} \vdash new\alpha.p[x'[\alpha]/x] \hat{\sim} new\alpha.q[x'[\alpha]/x] : \delta\mathbb{P}$, and also $s' \dot{\cup} \{a\}; ; x : \mathbb{R} \vdash p[a/\alpha] \hat{\sim} q[a/\alpha] : \mathbb{P}$. Since $s \dot{\cup} \{a\} \vdash t[a] \hat{\sim}_c u : \mathbb{P}$ there exists a term v such that $s \vdash t \hat{\sim} v : \delta\mathbb{P}$ and $s \dot{\cup} \{a\} \vdash v[a] \sim u : \mathbb{P}$. By the induction hypothesis we have $s' \vdash v \xrightarrow{new\alpha.q[x'[\alpha]/x]} v'$ with $s' \vdash t' \hat{\sim}_c v' : \delta\mathbb{R}$. As $\hat{\sim}$ is operator respecting (Lemma 5.3.10.2), we obtain $s' \dot{\cup} \{a\} \vdash t'[a] \hat{\sim}_c v'[a] : \mathbb{P}$. We get a transition $s' \dot{\cup} \{a\} \vdash v[a] \xrightarrow{q[a/\alpha]} v'[a]$ by the operational rules.

Pattern matching. Suppose $s \vdash [t_1 > p(x) \Rightarrow t_2] \hat{\sim}_c u : \mathbb{Q}$ and that $s' \vdash [t_1 > p(x) \Rightarrow t_2] \xrightarrow{p'(x')} t_3$ is derived from $s' \vdash t_1 \xrightarrow{p(x)} t_1'$ and $s' \vdash t_2[t_1'/x] \xrightarrow{p'(x')} t_3$, for some $s' \supseteq s$. Let q and q' be actions such that $s'; ; x{:}\mathbb{R} \vdash p \hat{\sim} q : \mathbb{P}$ and $s'; ; x'{:}\mathbb{R}' \vdash p' \hat{\sim}_c q' : \mathbb{Q}$. Since $s \vdash [t_1 > p(x) \Rightarrow t_2] \hat{\sim}_c u : \mathbb{Q}$, there exist two terms v_1 and v_2 such that $s_1 \vdash t_1 \hat{\sim}_c v_1 : \mathbb{P}$ and $A; x{:}\mathbb{R}; A, d \vdash \sigma t_2 \hat{\sim}_c \sigma v_2 : \mathbb{Q}$ for a bijection $\sigma : s_2 \to A$, where $s_1 \cup s_2 = s$ and $s' \cap \sigma(\{\alpha \mid (\alpha, x) \in d\}) = \emptyset$. It also holds $s \vdash [v_1 > p(x) \Rightarrow v_2] \sim u : \mathbb{Q}$. By the induction hypothesis we have $s' \vdash v_1 \xrightarrow{q(x)} v_1'$ with $s' \vdash t_1' \hat{\sim}_c v_1' : \mathbb{R}$. By weakening and by Lemma 5.3.9.1 we get $s' \vdash t_2[t_1'/x] \hat{\sim}_c v_2[v_1'/x] : \mathbb{Q}$. Applying the induction hypothesis in this case yields $s' \vdash v_2[v_1'/x] \xrightarrow{q'(x')} v_3$ with $s' \vdash t_3 \hat{\sim}_c v_3$, and hence $s' \vdash [v_1 > p(x) \Rightarrow v_2] \xrightarrow{q'(x')} v_3$.

The induction is complete. □

Theorem 5.3.14 *Bisimilarity $\sim$ is a congruence.*

Proof As shown in Lemma 5.3.13, $\hat{\sim}_c$ is a simulation. Then $\hat{\sim}_c{}^*$ is a bisimulation by Property 5.3.12, and so $\hat{\sim}_c{}^* \subseteq \sim$. In particular $\hat{\sim}_c \subseteq \sim$. By Lemma 5.3.10.1 and Lemma 5.3.9, it follows that $\hat{\sim} \subseteq \sim^\circ$, and so by Lemma 5.3.10.3, $\hat{\sim} = \sim^\circ$. Hence, $\sim$ is a congruence because it is an equivalence relation and by Lemma 5.3.10.2 is operator respecting. □

A remark on the definition of the precongruence candidate: the more standard rule

$$\frac{A, \alpha; \Gamma; d \vdash t \hat{\sim} t' : \mathbb{P} \qquad A; \Gamma; (d \setminus \alpha) \vdash new\alpha.t' \sim^\circ w : \delta\mathbb{P}}{A; \Gamma; (d \setminus \alpha) \vdash new\alpha.t \hat{\sim} w : \delta\mathbb{P}}$$

does not seem to capture the essence of new-name abstraction. In fact, this rule does not allow to prove Lemma 5.3.13 (at least not in a handy way).

5.3.2 Algebraic theory

We list of some results about bisimilarity. In an equation $s \vdash t \sim u : \mathbb{P}$, we write *lhs* for the left hand side of the equation (that is, t), and *rhs* for the right hand side (q). We also write $\mathsf{n}(t,u)$ to denote the set $\mathsf{n}(t) \cup \mathsf{n}(u)$.

Proposition 5.3.15 *For closed, well-formed, terms we have*

$$\begin{array}{rr}
s \vdash (\lambda x.t)u \sim t[u/x] : \mathbb{P} & s \vdash (\lambda \alpha.t)a \sim t[a/\alpha] : \mathbb{P} \\
s \vdash \lambda x.(tx) \sim t : \mathbb{P} \to \mathbb{Q} & s \vdash \lambda \alpha.(t\alpha) \sim t : \mathbb{N} \to \mathbb{P} \\
s \vdash \lambda x.(\Sigma_{i\in I} t_i) \sim \Sigma_{i\in I}(\lambda x.t_i) : \mathbb{P} \to \mathbb{Q} & s \vdash \lambda \alpha.(\Sigma_{i\in I} t_i) \sim \Sigma_{i\in I}(\lambda \alpha.t_i) : \mathbb{N} \to \mathbb{P} \\
s \vdash (\Sigma_{i\in I} t_i)u \sim \Sigma_{i\in I}(t_i u) : \mathbb{P} & s \vdash (\Sigma_{i\in I} t_i)a \sim \Sigma_{i\in I}(t_i a) : \mathbb{P}
\end{array}$$

$$\begin{array}{rr}
s \vdash \pi_\beta(\beta \cdot t) \sim t : \mathbb{P} & s \vdash \pi_\beta(\alpha \cdot t) \sim \mathbf{0} : \mathbb{P} \\
s \vdash t \sim \Sigma_{\alpha\in\mathbb{N}} \alpha \cdot (\pi_\alpha t) : \mathbb{N} \otimes \mathbb{P} & \\
s \vdash \beta \cdot (\Sigma_{i\in I} t_i) \sim \Sigma_{i\in I} \beta \cdot t_i : \mathbb{P} & s \vdash \pi_\beta(\Sigma_{i\in I} t_i) \sim \Sigma_{i\in I} \pi_\beta t_i : \mathbb{P}
\end{array}$$

$$s \vdash [!u > !x \Rightarrow t] \sim t[u/x] : \mathbb{P} \qquad s \vdash [\Sigma_{i\in I} u_i > !x \Rightarrow t] \sim \Sigma_{i\in I}[u_i > !x \Rightarrow t] : \mathbb{P}$$

Proof Let $\mathcal{I}$ be the identical type respecting relation over closed terms. In each postulated case $s \vdash \mathit{lhs} \sim \mathit{rhs} : \mathbb{P}$, the relation $\mathcal{S} = \{ (s' \vdash \mathit{lhs} : \mathbb{P}, s' \vdash \mathit{rhs} : \mathbb{P}) \mid s' \supseteq s \} \cup \mathcal{I}$ is a bisimulation. □

Proposition 5.3.16 *If $s \dot\cup \{a\} \vdash t[a/\alpha] \sim u[a/\alpha] : \mathbb{P}$ and $a \notin \mathsf{n}(t,u)$, then $s \vdash \mathit{new}\alpha.t \sim \mathit{new}\alpha.u : \delta\mathbb{P}$.*

Proof Let

$$\mathcal{R} = \{ (s \vdash \mathit{new}\alpha.t : \delta\mathbb{P}, s \vdash \mathit{new}\alpha.u : \delta\mathbb{P}) \mid s \dot\cup \{a\} \vdash t[a/\alpha] \sim u[a/\alpha] : \mathbb{P} \} .$$

We show that $\mathcal{R}$ is a bisimulation. Suppose $s' \vdash \mathit{new}\alpha.t \xrightarrow{\mathit{new}\alpha.p} \mathit{new}\alpha.t'$ for some $s' \supseteq s$. This must have been derived from $s' \dot\cup \{a\} \vdash t[a/\alpha] \xrightarrow{p[a/\alpha]} t'[a/\alpha]$ for some a. By bisimulation, we have $s' \dot\cup \{a\} \vdash u[a/\alpha] \xrightarrow{p[a/\alpha]} u'[a/\alpha]$ with $s' \dot\cup \{a\} \vdash t'[a/\alpha] \sim u'[a/\alpha] : \mathbb{P}$. By the operational rules, we have $s' \vdash \mathit{new}\alpha.u \xrightarrow{\mathit{new}\alpha.p} \mathit{new}\alpha.u'$, and by definition of $\mathcal{R}$ we conclude $s \vdash \mathit{new}\alpha.t' \sim \mathit{new}\alpha.u' : \delta\mathbb{P}$. □

Proposition 5.3.17 (β-equivalence on new names) *Let t be a term with α free, that is $A, \alpha; \emptyset; A, d \vdash t : \mathbb{P}$. Let $\sigma : s \to A$ be a bijection. Then*

$$s \dot\cup \{a\} \vdash (\mathit{new}\alpha.t)[a] \sim t[a/\alpha] : \mathbb{P} .$$

Proof Let

$$\mathcal{R} = \{ (s \dot\cup \{a\} \vdash (\mathit{new}\alpha.t)[a] : \mathbb{P}, s \dot\cup \{a\} \vdash t[a/\alpha] : \mathbb{P}) \mid A, \alpha; \emptyset; A, d \vdash t : \mathbb{P} \text{ for } \sigma : s \to_{\mathrm{bij}} A \} .$$

We show that $\mathcal{R}$ is a bisimulation. Consider $s \dot\cup \{a\} \vdash (\mathit{new}\alpha.t)[a] \; \mathcal{R} \; t[a/\alpha] : \mathbb{P}$.

Suppose that $s' \dot\cup \{a\} \vdash (new\alpha.t)[a] \xrightarrow{p[a/\alpha](x)} (new\alpha.t')[a]$ for some $s' \supseteq s$. This must have been derived from $s' \vdash new\alpha.t \xrightarrow{new\alpha.\alpha \mapsto p[x'[\alpha]/x](x')} new\alpha.t'$. In turn, this must have derived from $s' \dot\cup \{a\} \vdash t[a/\alpha] \xrightarrow{p[a/\alpha](x)} t'[a/\alpha]$. So we have a matching transition, and $s' \dot\cup \{a\} \vdash (new\alpha.t')[a] \; \mathcal{R} \; t'[a/\alpha] : \mathbb{R}$ follows from the construction of $\mathcal{R}$, where $\mathbb{R}$ is the type of the resumption variable in p.

Suppose now that $s' \dot\cup \{a\} \vdash t[a/\alpha] \xrightarrow{p[a/\alpha](x)} t'[a/\alpha]$ for some $s' \supseteq s$. By the operational rules we get $s' \vdash new\alpha.t \xrightarrow{new\alpha.\alpha \mapsto p[x'[\alpha]/x](x')} new\alpha.t'$. In turn, by the operational rules we get $s \dot\cup \{a\} \vdash (new\alpha.t)[a] \xrightarrow{p[a/\alpha](y)} (new\alpha.t')[a]$. So we have a matching transition, and $s' \dot\cup \{a\} \vdash (new\alpha.t')[a] \; \mathcal{R} \; t'[a/\alpha] : \mathbb{R}$ follows from the construction of $\mathcal{R}$. □

Corollary 5.3.18 *If $s \vdash new\alpha.t \sim new\alpha.u : \delta\mathbb{P}$, then $s \dot\cup \{a\} \vdash t[a/\alpha] \sim u[a/\alpha] : \mathbb{P}$.*

Proof By weakening, $s \dot\cup \{a\} \vdash new\alpha.t \sim new\alpha.u : \delta\mathbb{P}$. By congruence, $s \dot\cup \{a\} \vdash (new\alpha.t)[a] \sim new(\alpha.u)[a] : \mathbb{P}$. By Proposition 5.3.17, we have $s \dot\cup \{a\} \vdash (new\alpha.t)[a/\alpha] \sim t[a/\alpha] : \mathbb{P}$ and $s \dot\cup \{a\} \vdash (new\alpha.u)[a/\alpha] \sim u[a/\alpha] : \mathbb{P}$. The result follows from transitivity of $\sim$. □

Corollary 5.3.19 *Let t and u be terms such that $A, \alpha{:}\mathbb{N}; \emptyset; A, d \vdash t : \mathbb{P}$, let $\sigma : s \to A$ be a bijection, and let a, a' be names not in s. If $s \dot\cup \{a\} \vdash t[a/\alpha] \sim u[a/\alpha] : \mathbb{P}$ then $s \dot\cup \{a'\} \vdash t[a'/\alpha] \sim u[a'/\alpha] : \mathbb{P}$.*

Proof By Proposition 5.3.16 $s \vdash new\alpha.t \sim new\alpha.u : \delta\mathbb{P}$. The result follows by Corollary 5.3.18. □

We conclude this section by introducing a basic up-to proof technique.

Definition 5.3.20 (Bisimulation up to bisimilarity) *A symmetric type respecting relation on closed terms, $\mathcal{R}$, is a* bisimulation up to bisimilarity *if $s \vdash t \; \mathcal{R} \; u : \mathbb{P}$ and $s' \vdash t \xrightarrow{p} t'$ for $s' \supseteq s$ imply that there exists a term u' such that $s' \vdash u \xrightarrow{p} u'$ and $s' \vdash t' \sim\mathcal{R}\sim u' : \mathbb{P}$.*

Proposition 5.3.21 *If $\mathcal{R}$ is a bisimulation up to bisimilarity, then $\mathcal{R} \subseteq \sim$.*

Proof Let $\mathcal{S} = \{(s \vdash t : \mathbb{P}, s \vdash u : \mathbb{P}) \mid s \vdash t \sim\mathcal{R}\sim u : \mathbb{P}\}$. The relation $\mathcal{S}$ is a bisimulation (simple diagram chasing argument). □

5.4 Examples

In this section, we illustrate how new-HOPLA can be used to give semantics to well-known process algebras. We initially focus on π-calculus, more precisely on its late semantics. We define an encoding that preserves and reflects both the reduction relation and strong bisimilarity. A similar encoding is proposed for the early semantics. Then we consider the polyadic extension of π-calculus: polyadicity turns out to be an excellent testing ground for the expressivity of the operators that manipulate names. Again, the encoding preserves and reflects the reduction relation: we conjecture that results on preservation of strong bisimulation can be proved along the lines of our development of monadic π-calculus. Finally,

Actions: $\ell \;::=\; \overline{n}m \;\mid\; \overline{n}(\alpha) \;\mid\; n\alpha \;\mid\; \tau$

$$\frac{}{\overline{n}m.P \xrightarrow{\overline{n}m}_l P} \qquad \frac{}{n(\alpha) \xrightarrow{n(\alpha)}_l P} \qquad \frac{P \xrightarrow{\ell}_l P' \quad \alpha \notin \mathrm{fv}(\ell)}{(\boldsymbol{\nu}\alpha)P \xrightarrow{\ell}_l (\boldsymbol{\nu}\alpha)P'} \qquad \frac{P \xrightarrow{\overline{n}\alpha}_l P'}{(\boldsymbol{\nu}\alpha)P \xrightarrow{\overline{n}(\alpha)}_l P'}$$

$$\frac{P \xrightarrow{\ell}_l P'}{P \mid Q \xrightarrow{\ell}_l P' \mid Q} \qquad \frac{P \xrightarrow{\overline{n}m}_l P' \quad Q \xrightarrow{n\alpha}_l Q'}{P \mid Q \xrightarrow{\tau}_l P' \mid Q'[m/\alpha]} \qquad \frac{P \xrightarrow{\overline{n}(\beta)}_l P' \quad Q \xrightarrow{n\alpha}_l Q'}{P \mid Q \xrightarrow{\tau}_l (\boldsymbol{\nu}\beta)(P' \mid Q'[\beta/\alpha])}$$

Figure 5.2: π-calculus: the *late* labelled transition system

we provide an encoding of Mobile Ambients into new-HOPLA that preserves and reflects the reduction semantics of Mobile Ambients.

We introduce an useful product type $\mathbb{P}\,\&\,\mathbb{Q}$, which is not primitive in new-HOPLA. It is definable as $1{:}\mathbb{P} + 2{:}\mathbb{Q}$. The projections are given by $\mathit{fst}(t) = \pi_1(t)$ and $\mathit{snd}(t) = \pi_2(t)$, while pairing is defined as $(t, u) = 1{:}t + 2{:}u$. For actions $(p, -) = 1{:}p$, $(-, q) = 2{:}q$. It is then easy to verify that $s \vdash \mathit{fst}(t, u) \sim t : \mathbb{P}$, that $s \vdash \mathit{snd}(t, u) \sim u : \mathbb{Q}$, and that $s \vdash (\mathit{fst}(t, u), \mathit{snd}(t, u)) \sim (t, u) : \mathbb{P}\,\&\,\mathbb{Q}$, for all $s \supseteq \mathsf{n}(t) \cup \mathsf{n}(u)$.

5.4.1 π-calculus: the late semantics

We denote *name constants* with $a, b, \ldots$, and *name variables* with $\alpha, \beta, \ldots$; the letters $n, m, \ldots$ range over both name constants and name variables. The terms of the language are constructed according the following grammar:

$$P \;::=\; \mathbf{0} \;\mid\; P \mid P \;\mid\; (\boldsymbol{\nu}\alpha)P \;\mid\; \overline{n}m.P \;\mid\; n(\alpha).P\ .$$

The late labelled transition system is reported in Figure 5.2 (we omit the symmetric rules).

Definition 5.4.1 (Late strong bisimilarity) Late strong bisimilarity *is the largest symmetric relation, $\sim_l$, such that whenever $P \sim_l Q$,*

1. *$P \xrightarrow{n\beta}_l P'$ implies there is Q' such that $Q \xrightarrow{n\beta}_l Q'$ and $P'[m/\alpha] \sim_l Q'[m/\alpha]$ for every m;*

2. *if ℓ is not an input action then $P \xrightarrow{\ell}_l P'$ implies $Q \xrightarrow{\ell}_l \sim_l Q'$.*

It is well-known that late strong bisimilarity is preserved by all operators except input prefix. In particular, both transitions and late strong bisimilarity are preserved by injective renaming.

The encoding Our encoding of the late semantics of π-calculus into new-HOPLA is reported in Table 5.4. The terms of π-calculus are translated into new-HOPLA by the function $[\![-]\!]$ defined by structural induction. The operation of parallel composition $|| : \mathbb{P}\&\mathbb{P} \to \mathbb{P}$ (we use infix notation for convenience) is an 'implementation' of the *(late) expansion law* of π-calculus. The restriction $Res : \delta\mathbb{P} \to \mathbb{P}$ informally pushes restrictions inside processes as far as possible. The five summands corresponds to the five equations below:

$$\begin{aligned}
(\boldsymbol{\nu}\alpha)\tau.P &\sim_l \tau.(\boldsymbol{\nu}\alpha)P \\
(\boldsymbol{\nu}\alpha)\overline{m}n.P &\sim_l \overline{m}n.(\boldsymbol{\nu}\alpha)P \qquad \text{if } \alpha \neq m, n \\
(\boldsymbol{\nu}\alpha)\overline{m}\alpha.P &\sim_l \overline{m}(\alpha).P \qquad \text{if } \alpha \neq m \\
(\boldsymbol{\nu}\alpha)\overline{m}(\beta).P &\sim_l \overline{m}(\beta).(\boldsymbol{\nu}\alpha)P \qquad \text{if } \alpha \neq m \\
(\boldsymbol{\nu}\alpha)m\beta.P &\sim_l m\beta.(\boldsymbol{\nu}\alpha)P \qquad \text{if } \alpha \neq m
\end{aligned}$$

where $\overline{m}(\alpha)$ is an abbreviation to express bound-output, that is, $(\boldsymbol{\nu}\alpha)\overline{m}\alpha$. The map *Res* implicitly also implements the equation $(\boldsymbol{\nu}\alpha)P \sim \mathbf{0}$ if none of the above cases applies.

Lemma 5.4.2 (Basic properties of $[\![-]\!]$)

1. $\mathrm{fv}(P) = \mathrm{fv}([\![P]\!])$;
2. $\mathsf{n}(P) = \mathsf{n}([\![P]\!])$;
3. $[\![P]\!][a/\alpha] = [\![P[a/\alpha]]\!]$.

In the sequel we use these facts without mention.

We concentrate on closed terms. There is a strong correspondence between actions performed by a closed π-calculus process and the actions emitted by its encoding.

Theorem 5.4.3 *Let P a closed π-calculus process. Then,*

1. $P \xrightarrow{\overline{\alpha}\beta}_l P'$ *if and only if* $\mathsf{n}(P) \vdash [\![P]\!] \xrightarrow{\mathsf{out}:a\cdot b\cdot !} [\![P']\!]$;

2a. $P \xrightarrow{\alpha\beta}_l P'$ *implies* $\mathsf{n}(P) \vdash [\![P]\!] \xrightarrow{\mathsf{inp}:a!} \sim (\lambda\beta.[\![P']\!])$;

2b. $\mathsf{n}(P) \vdash [\![P]\!] \xrightarrow{\mathsf{inp}:a!} t$ *implies* $P \xrightarrow{\alpha\beta}_l P'$ *and* $t \sim (\lambda\beta.[\![P']\!])$;

3a. $P \xrightarrow{\overline{\alpha}(\beta)}_l P'$ *implies* $\mathsf{n}(P) \vdash [\![P]\!] \xrightarrow{\mathsf{bout}:a!} \sim (new\beta.[\![P']\!])$;

3b. $\mathsf{n}(P) \vdash [\![P]\!] \xrightarrow{\mathsf{bout}:a!} t$ *implies* $P \xrightarrow{\overline{\alpha}(\beta)}_l P'$ *and* $t \sim (new\beta.[\![P']\!])$;

4a. $P \xrightarrow{\tau}_l P'$ *implies* $\mathsf{n}(P) \vdash [\![P]\!] \xrightarrow{\tau:!} \sim [\![P']\!]$;

4b. $\mathsf{n}(P) \vdash [\![P]\!] \xrightarrow{\tau:!} t$ *implies* $P \xrightarrow{\tau}_l P'$ *and* $t \sim [\![P']\!]$

Parts 4*a* and 4*b* of the previous lemma show that the encoding preserves and reflects τ-transitions. This shows that the encoding agrees with the reduction semantics.

More than that, as we will see shortly, the encoding preserves and reflects late strong bisimulation. We introduce some notations useful in the proofs of the next two theorems:

$$\mathbb{P} \;=\; \tau{:}!\mathbb{P} \;+\; \mathsf{out}{:}\mathbb{N} \otimes \mathbb{N} \otimes\, !\mathbb{P} \;+\; \mathsf{bout}{:}\mathbb{N} \otimes\, !(\delta\mathbb{P}) \;+\; \mathsf{inp}{:}\mathbb{N} \otimes\, !(\mathbb{N} \to \mathbb{P})$$

$$\begin{array}{rcl}
[\![\, \mathbf{0} \,]\!] & = & \mathbf{0} \\
[\![\, \overline{\alpha}\beta.p \,]\!] & = & \mathsf{out}{:}\alpha \cdot \beta \cdot\, !\,[\![\, p \,]\!] \\
[\![\, \alpha(\beta).p \,]\!] & = & \mathsf{inp}{:}\alpha \cdot\, !(\lambda\beta.[\![\, p \,]\!]) \\
[\![\, (\boldsymbol{\nu}\alpha)p \,]\!] & = & \mathit{Res}\,(\mathit{new}\alpha.[\![\, p \,]\!]) \\
[\![\, p \mid q \,]\!] & = & [\![\, p \,]\!] \;\;||\;\; [\![\, q \,]\!]
\end{array}$$

$$\begin{array}{rcl}
\mathit{Res} & : & \delta\mathbb{P} \to \mathbb{P} \\
\mathit{Res}\;\; t & = & [t > \mathit{new}\alpha.\tau{:}!(x[\alpha])) \Rightarrow \tau{:}!\mathit{Res}\, x] \\
& + & \Sigma_{\beta\in\mathbb{N}}\Sigma_{\gamma\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{out}{:}\beta \cdot \gamma \cdot\, !(x[\alpha]) \Rightarrow \mathsf{out}{:}\beta \cdot \gamma \cdot\, !\mathit{Res}\, x] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{out}{:}\beta \cdot \alpha \cdot\, !(x[\alpha]) \Rightarrow \mathsf{bout}{:}\beta \cdot\, !x] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{bout}{:}\beta \cdot\, !(x[\alpha]) \Rightarrow \mathsf{bout}{:}\beta \cdot\, !\mathit{new}\gamma \cdot \mathit{Res}\,(\mathit{new}\eta.x[\eta][\gamma])] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{inp}{:}\beta \cdot\, !(x[\alpha]) \Rightarrow \mathsf{inp}{:}\beta \cdot\, !\lambda\gamma.\mathit{Res}\,(\mathit{new}\eta.x[\eta](\gamma))]
\end{array}$$

$$\begin{array}{rcl}
|| & : & \mathbb{P}\&\mathbb{P} \to \mathbb{P} \\
t \;||\; u & = & [t > \tau{:}!x \Rightarrow \tau{:}!(x \;||\; u)] \\
& + & \Sigma_{\beta\in\mathbb{N}}\Sigma_{\gamma\in\mathbb{N}}[t > \mathsf{out}{:}(\beta \cdot \gamma \cdot\, !x) \Rightarrow [u > \mathsf{inp}{:}(\beta \cdot\, !y) \Rightarrow \tau{:}!(x \;||\; y\gamma)]] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{bout}{:}(\beta \cdot\, !x) \Rightarrow [u > \mathsf{inp}{:}(\beta \cdot\, !y) \Rightarrow \tau{:}!\mathit{Res}\,(\mathit{new}\eta.(x[\eta] \;||\; y\eta))]] \\
& + & \Sigma_{\beta\in\mathbb{N}}\Sigma_{\gamma\in\mathbb{N}}[t > \mathsf{out}{:}\beta \cdot \gamma \cdot\, !x \Rightarrow \mathsf{out}{:}\beta \cdot \gamma \cdot\, !(x \;||\; u)] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{bout}{:}\beta \cdot\, !x \Rightarrow \mathsf{bout}{:}\beta \cdot\, !\mathit{new}\eta.(x[\eta] \;||\; u)] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{inp}{:}\beta \cdot\, !x \Rightarrow \mathsf{inp}{:}\beta \cdot\, !\lambda\eta.(x(\eta) \;||\; u)] \\
& + & \text{same cases with } t \text{ and } u \text{ swapped.}
\end{array}$$

where η is chosen to avoid clashes with the free name variables of u.

Table 5.4: π-calculus: encoding of the late semantics

- we write $\vec{\alpha}_n$ or simply $\vec{\alpha}$ for the set $\{\alpha_1, \ldots, \alpha_n\}$ where the α_i are all distinct. We write $new\vec{\alpha}_n.t$ for $new\alpha_1.\ldots.new\alpha_n.t$. Also $\delta^n\mathbb{P}$ stands for $\delta \ldots \delta\mathbb{P}$ where the δ is replicated n times;
- most of the substitutions we use in the next two theorems are bijections involving fresh name constants. So, whenever a is fresh for P, we write $P[a/\alpha] \xrightarrow{\tau} P'[a/\alpha]$ as a shorthand for $P[a/\alpha] \xrightarrow{\tau} P'_1$ and $P' = P'_1[\alpha/a]$. Same with terms and transitions of new-HOPLA;
- we write $\rightarrow$ instead of $\rightarrow_l$.

Theorem 5.4.4 *Let P and Q be two closed π-calculus processes. If $P \sim_l Q$ then $\mathsf{n}(P,Q) \vdash [\![P]\!] \sim [\![Q]\!] : \mathbb{P}$.*

Proof We actually prove a stronger theorem:

Let P and Q be two π-calculus processes such that $\mathrm{fv}(P) = \mathrm{fv}(Q) = \vec{\alpha}_n$.

1. If $P \sim_l Q$, then $\mathsf{n}(lhs,rhs) \vdash new\vec{\alpha}_n.[\![P]\!] \sim new\vec{\alpha}_n.[\![Q]\!] : \delta^n\mathbb{P}$, and
2. if $\gamma \in \vec{\alpha}$ and for all m it holds $P[m/\gamma] \sim_l Q[m/\gamma]$, then $\mathsf{n}(lhs,rhs) \vdash new(\vec{\alpha}_n \setminus \gamma).\lambda\gamma.[\![P]\!] \sim new(\vec{\alpha}_n \setminus \gamma).\lambda\gamma.[\![Q]\!] : \delta^{n-1}(\mathbb{N} \rightarrow \mathbb{P})$.

Let

$$\begin{aligned}\mathcal{R} \;=\; & \{\; (s \vdash new\vec{\alpha}_n.[\![P]\!] : \delta^n\mathbb{P}\,,\; s \vdash new\vec{\alpha}_n.[\![Q]\!] : \delta^n\mathbb{P}) \;\mid\; P \sim_l Q \text{ and } \mathrm{fv}(P) = \mathrm{fv}(Q) = \vec{\alpha} \;\} \\ & \cup \{\; (s \vdash new\vec{\alpha}_n.\lambda\gamma.[\![P]\!] : \delta^n(\mathbb{N} \rightarrow \mathbb{P})\,,\; s \vdash new\vec{\alpha}_n.\lambda\gamma.[\![Q]\!] : \delta^n(\mathbb{N} \rightarrow \mathbb{P})) \;\mid \\ & \qquad \forall m.P[m/\gamma] \sim_l Q[m/\gamma] \text{ and } \mathrm{fv}(P) = \mathrm{fv}(Q) = \vec{\alpha} \cup \{\gamma\} \;\}\end{aligned}$$

where $s \supseteq \mathsf{n}(lhs,rhs)$. We prove that $\mathcal{R}$ is a bisimulation up to bisimulation; the result will follows from the soundness of the up-to bisimulation proof technique (Proposition 5.3.21).

First consider

$$s \vdash new\vec{\alpha}_n.[\![P]\!] \;\mathcal{R}\; new\vec{\alpha}_n.[\![Q]\!] : \delta^n\mathbb{P}$$

with $P \sim_l Q$ and $\mathrm{fv}(P) = \mathrm{fv}(Q) = \vec{\alpha}$. We perform a case analysis on the actions performed by $new\vec{\alpha}_n.[\![P]\!]$.

- Suppose that $s' \vdash new\vec{\alpha}_n.[\![P]\!] \xrightarrow{new\vec{\alpha}_n.\tau:!} new\vec{\alpha}_n.t$ for some $s' \supseteq s$. This must have been derived from $s' \dot\cup \vec{a} \vdash [\![P]\!][\vec{a}/\vec{\alpha}] \xrightarrow{\tau:!} t[\vec{a}/\vec{\alpha}]$. As $[\![P]\!][\vec{a}/\vec{\alpha}] = [\![P[\vec{a}/\vec{\alpha}]]\!]$, by Lemma 5.4.3, there is a term $P'[\vec{a}/\vec{\alpha}]$ such that $P[\vec{a}/\vec{\alpha}] \xrightarrow{\tau} P'[\vec{a}/\vec{\alpha}]$ and $s' \dot\cup \vec{a} \vdash t[\vec{a}/\vec{\alpha}] \sim [\![P']\!][\vec{a}/\vec{\alpha}] : \mathbb{P}$. Late strong bisimulation is preserved by injective substitution, so $P[\vec{a}/\vec{\alpha}] \sim_l Q[\vec{a}/\vec{\alpha}]$. Then, by bisimulation there is $Q'[\vec{a}/\vec{\alpha}]$ such that $Q[\vec{a}/\vec{\alpha}] \xrightarrow{\tau} Q'[\vec{a}/\vec{\alpha}]$ and $P'[\vec{a}/\vec{\alpha}] \sim_l Q'[\vec{a}/\vec{\alpha}]$. By Lemma 5.4.3, $s' \dot\cup \vec{a} \vdash [\![Q]\!][\vec{a}/\vec{\alpha}] \xrightarrow{\tau:!} u[\vec{a}/\vec{\alpha}]$ and $s' \dot\cup \vec{a} \vdash u[\vec{a}/\vec{\alpha}] \sim [\![Q']\!][\vec{a}/\vec{\alpha}] : \mathbb{P}$. By the operational rules we have $s' \vdash new\vec{\alpha}_n.[\![Q]\!] \xrightarrow{new\vec{\alpha}_n.\tau:!} new\vec{\alpha}_n.u$. We must still show $s' \vdash new\vec{\alpha}_n.t \sim \mathcal{R} \sim new\vec{\alpha}_n.u : \delta^n\mathbb{P}$. By multiple applications of Lemma 5.3.16 we derive $s' \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.[\![P']\!] : \delta^n\mathbb{P}$ from $s' \dot\cup \vec{a} \vdash t[\vec{a}/\vec{\alpha}] \sim [\![P']\!][\vec{a}/\vec{\alpha}] : \mathbb{P}$. We also derive $s' \vdash new\vec{\alpha}.u \sim new\vec{\alpha}.[\![Q']\!] : \delta^n\mathbb{P}$ from $s' \dot\cup \vec{a} \vdash u[\vec{a}/\vec{\alpha}] \sim [\![Q']\!][\vec{a}/\vec{\alpha}] : \mathbb{P}$. Again,

late strong bisimulation is preserved by injective substitution, so $P'[\vec{a}/\vec{\alpha}] \sim_l Q'[\vec{a}/\vec{\alpha}]$ implies $P' \sim_l Q'$. By construction of $\mathcal{R}$, we finally have

$$s' \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.[\![P']\!] \; \mathcal{R} \; new\vec{\alpha}.[\![Q']\!] \sim new\vec{\alpha}.u : \delta^n\mathbb{P} \; .$$

as wanted.

- The case $new\vec{\alpha}.\mathsf{out}{:}a \cdot b \cdot !$ is similar to the previous one.

- Suppose that for some $s' \supseteq s$, $s' \vdash new\vec{\alpha}_n.[\![P]\!] \xrightarrow{new\vec{\alpha}_n.\mathsf{bout}:b\cdot!} new\vec{\alpha}_n.t$. This must have been derived from $s' \dot\cup \vec{a} \vdash [\![P]\!][\vec{a}/\vec{\alpha}] \xrightarrow{\mathsf{bout}:i\cdot!} t[\vec{a}/\vec{\alpha}]$, where $i = a_i$ if $b = \alpha_i$, and $i = b$ otherwise. By Lemma 5.4.3, there is a term $P'[\vec{a}/\vec{\alpha}]$ such that $P[\vec{a}/\vec{\alpha}] \xrightarrow{\bar{i}(\gamma)} P'[\vec{a}/\vec{\alpha}]$ and $s' \dot\cup \vec{a} \vdash t[\vec{a}/\vec{\alpha}] \sim (new\gamma.[\![P']\!])[\vec{a}/\vec{\alpha}] : \delta\mathbb{P}$. Observe that $\gamma \notin \vec{\alpha}$. Now, by late strong bisimulation, there is $Q'[\vec{a}/\vec{\alpha}]$ such that $Q[\vec{a}/\vec{\alpha}] \xrightarrow{\bar{i}(\gamma)} Q'[\vec{a}/\vec{\alpha}]$ and $P'[\vec{a}/\vec{\alpha}] \sim_l Q'[\vec{a}/\vec{\alpha}]$. By Lemma 5.4.3, $s' \dot\cup \vec{a} \vdash [\![Q]\!][\vec{a}/\vec{\alpha}] \xrightarrow{\mathsf{bout}:i\cdot!} u[\vec{a}/\vec{\alpha}]$ and $s' \dot\cup \vec{a} \vdash u[\vec{a}/\vec{\alpha}] \sim (new\gamma.[\![Q']\!])[\vec{a}/\vec{\alpha}] : \delta\mathbb{P}$. By the operational rules we have $s' \vdash new\vec{\alpha}_n.[\![Q]\!] \xrightarrow{new\vec{\alpha}_n.\mathsf{bout}:b\cdot!} new\vec{\alpha}_n.u$. We must conclude $s' \vdash new\vec{\alpha}_n.t \sim\mathcal{R}\sim new\vec{\alpha}_n.u : \delta^n\delta\mathbb{P}$. Multiple applications of Lemma 5.3.16 allow us to derive $s' \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.new\gamma.[\![P']\!] : \delta^n\delta\mathbb{P}$ from $s' \dot\cup \vec{a} \vdash t[\vec{a}/\vec{\alpha}] \sim (new\gamma.[\![P']\!])[\vec{a}/\vec{\alpha}] : \delta\mathbb{P}$. We also derive $s' \vdash new\vec{\alpha}.u \sim new\vec{\alpha}.new\gamma.[\![Q']\!] : \delta^n\delta\mathbb{P}$ from $s' \dot\cup \vec{a} \vdash u[\vec{a}/\vec{\alpha}] \sim (new\gamma.[\![Q']\!])[\vec{a}/\vec{\alpha}] : \delta\mathbb{P}$. Since $P' \sim_l Q'$, by construction of $\mathcal{R}$, we have

$$s' \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.new\gamma.[\![P']\!] \; \mathcal{R} \; new\vec{\alpha}.new\gamma.[\![Q']\!] \sim new\vec{\alpha}.u : \delta^n\delta\mathbb{P} \; .$$

- Suppose that for some $s' \supseteq s$, $s' \vdash new\vec{\alpha}_n.[\![P]\!] \xrightarrow{new\vec{\alpha}_n.\mathsf{inp}:b\cdot!} new\vec{\alpha}_n.t$. This must have been derived from $s' \dot\cup \vec{a} \vdash [\![P]\!][\vec{a}/\vec{\alpha}] \xrightarrow{\mathsf{inp}:i\cdot!} t[\vec{a}/\vec{\alpha}]$, where $i = a_i$ if $b = \alpha_i$, and $i = b$ otherwise. By Lemma 5.4.3, there is a term $P'[\vec{a}/\vec{\alpha}]$ such that $P[\vec{a}/\vec{\alpha}] \xrightarrow{i\gamma} P'[\vec{a}/\vec{\alpha}]$ and $s' \dot\cup \vec{a} \vdash t[\vec{a}/\vec{\alpha}] \sim (\lambda\gamma.[\![P']\!])[\vec{a}/\vec{\alpha}] : \mathbb{N} \to \mathbb{P}$. Now, by late strong bisimulation, there is $Q'[\vec{a}/\vec{\alpha}]$ such that $Q[\vec{a}/\vec{\alpha}] \xrightarrow{i\gamma} Q'[\vec{a}/\vec{\alpha}]$ and for all m, $P'[\vec{a}/\vec{\alpha}][m/\gamma] \sim_l Q'[\vec{a}/\vec{\alpha}][m/\gamma]$. By Lemma 5.4.3, $s' \dot\cup \vec{a} \vdash [\![Q]\!][\vec{a}/\vec{\alpha}] \xrightarrow{\mathsf{inp}:i\cdot!} u[\vec{a}/\vec{\alpha}]$ and $s' \dot\cup \vec{a} \vdash u[\vec{a}/\vec{\alpha}] \sim (\lambda\gamma.[\![Q']\!])[\vec{a}/\vec{\alpha}] : \mathbb{N} \to \mathbb{P}$. By the operational rules we have the transition $s' \vdash new\vec{\alpha}_n.[\![Q]\!] \xrightarrow{new\vec{\alpha}_n.\mathsf{inp}:b\cdot!} new\vec{\alpha}_n.u$. We must conclude $s' \vdash new\vec{\alpha}_n.t \sim\mathcal{R}\sim new\vec{\alpha}_n.u : \delta^n\mathbb{N} \to \mathbb{P}$. By multiple applications of Lemma 5.3.16 we derive $s' \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.\lambda\gamma.[\![P']\!] : \delta^n(\mathbb{N} \to \mathbb{P})$ from $s' \dot\cup \vec{a} \vdash t[\vec{a}/\vec{\alpha}] \sim (\lambda\gamma.[\![P']\!])[\vec{a}/\vec{\alpha}] : \mathbb{N} \to \mathbb{P}$. We also derive $s' \vdash new\vec{\alpha}.u \sim new\vec{\alpha}.\lambda\gamma.[\![Q']\!] : \delta^n(\mathbb{N} \to \mathbb{P})$ from $s' \dot\cup \vec{a} \vdash u[\vec{a}/\vec{\alpha}] \sim (\lambda\gamma.[\![Q']\!])[\vec{a}/\vec{\alpha}] : \mathbb{N} \to \mathbb{P}$. Since $P' \sim_l Q'$, we have

$$s' \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.\lambda\gamma.[\![P']\!] \; \mathcal{R} \; new\vec{\alpha}.\lambda\gamma.[\![Q']\!] \sim new\vec{\alpha}.u : \delta^n(\mathbb{N} \to \mathbb{P}) \; .$$

Consider now

$$s \vdash new\vec{\alpha}_n.\lambda\gamma.[\![P]\!] \;\; \mathcal{R} \;\; new\vec{\alpha}_n.\lambda\gamma.[\![Q]\!] : \delta^n(\mathbb{N} \to \mathbb{P})$$

because for all m, $P[m/\gamma] \sim_l Q[m/\gamma]$. We perform a case analysis on the actions performed by $new\vec{\alpha}_n.\lambda\gamma.[\![P]\!]$.

- Suppose that $s' \vdash new\vec{\alpha}_n.\lambda\gamma.[\![P]\!] \xrightarrow{new\vec{\alpha}_n.n \mapsto \tau:!} new\vec{\alpha}_n.t$ for some $s' \supseteq s$. This must have been derived from $s' \dot\cup \vec{a} \vdash [\![P]\!][\vec{a}/\vec{\alpha}] \xrightarrow{e \mapsto \tau:!} t[\vec{a}/\vec{\alpha}]$, that in turn must have been derived from $s' \dot\cup \vec{a} \vdash [\![P]\!][\vec{a}/\vec{\alpha}][e/\gamma] \xrightarrow{\tau:!} t[\vec{a}/\vec{\alpha}]$ for $e \in s' \dot\cup \vec{a}$. Observe that $e = a_i$ if $n = \alpha_i$ for some i, and $e = n$ otherwise. By Lemma 5.4.3, there is a term $P'[\vec{a}/\vec{\alpha}]$ such that $P[\vec{a}/\vec{\alpha}][e/\gamma] \xrightarrow{\tau} P'[\vec{a}/\vec{\alpha}]$ and $s' \dot\cup \vec{a} \vdash t[\vec{a}/\vec{\alpha}] \sim [\![P']\!][\vec{a}/\vec{\alpha}] : \mathbb{P}$. As for all m it holds $P[m/\gamma] \sim_l Q[m/\gamma]$, by bisimulation there is $Q'[\vec{a}/\vec{\alpha}]$ such that $Q[\vec{a}/\vec{\alpha}][e/\gamma] \xrightarrow{\tau} Q'[\vec{a}/\vec{\alpha}]$ and $P'[\vec{a}/\vec{\alpha}] \sim_l Q'[\vec{a}/\vec{\alpha}]$. By Lemma 5.4.3, $s' \dot\cup \vec{a} \vdash [\![Q]\!][\vec{a}/\vec{\alpha}][e/\gamma] \xrightarrow{\tau:!} u[\vec{a}/\vec{\alpha}]$ and $s' \dot\cup \vec{a} \vdash u[\vec{a}/\vec{\alpha}] \sim [\![Q']\!][\vec{a}/\vec{\alpha}] : \mathbb{P}$. By the operational rules we have $s' \vdash new\vec{\alpha}_n.\lambda\gamma.[\![Q]\!] \xrightarrow{new\vec{\alpha}_n.n \mapsto \tau:!} new\vec{\alpha}_n.u$. We must conclude $s' \vdash new\vec{\alpha}_n t \sim \mathcal{R} \sim new\vec{\alpha}_n u : \delta^n\mathbb{P}$. By multiple applications of Lemma 5.3.16 we derive $s' \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.[\![P']\!] : \delta^n\mathbb{P}$ from $s' \dot\cup \vec{a} \vdash t[\vec{a}/\vec{\alpha}] \sim [\![P']\!][\vec{a}/\vec{\alpha}] : \mathbb{P}$. We also derive $s' \vdash new\vec{\alpha}.u \sim new\vec{\alpha}.[\![Q']\!] : \delta^n\mathbb{P}$ from$s' \dot\cup \vec{a} \vdash u[\vec{a}/\vec{\alpha}] \sim [\![Q']\!][\vec{a}/\vec{\alpha}] : \mathbb{P}$. Late strong bisimulation is preserved by injective substitution, so $P'[\vec{a}/\vec{\alpha}] \sim_l Q'[\vec{a}/\vec{\alpha}]$ implies $P' \sim_l Q'$. By construction of $\mathcal{R}$, we finally have

$$s' \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.[\![P']\!] \; \mathcal{R} \; new\vec{\alpha}.[\![Q']\!] \sim new\vec{\alpha}.u : \delta^n\mathbb{P} \; .$$

  as wanted.

- The case $new\vec{\alpha}.\mathsf{out}{:}a \cdot b \cdot !$ is similar to the previous one.

- Suppose that $s' \vdash new\vec{\alpha}_n.\lambda\gamma.[\![P]\!] \xrightarrow{new\vec{\alpha}_n.n \mapsto \mathsf{bout}:b\cdot!} new\alpha_n.t$ for some $s' \supseteq s$. This must have been derived from $s' \dot\cup \vec{a} \vdash [\![P]\!][\vec{a}/\vec{\alpha}] \xrightarrow{e \mapsto \mathsf{bout}:i\cdot!} t[\vec{a}/\vec{\alpha}]$, that in turn must have been derived from $s' \dot\cup \vec{a} \vdash [\![P]\!][\vec{a}/\vec{\alpha}][e/\gamma] \xrightarrow{\mathsf{bout}:i\cdot!} t[\vec{a}/\vec{\alpha}]$ for $e \in s' \dot\cup \vec{a}$. Observe that $e = a_i$ if $n = \alpha_i$ and $e = n$ otherwise. Observe also that $i = a_j$ if $b = \alpha_j$ and $i = b$ otherwise. By Lemma 5.4.3, there is a term $P'[\vec{a}/\vec{\alpha}]$ such that $P[\vec{a}/\vec{\alpha}][e/\gamma] \xrightarrow{\bar{i}(\zeta)} P'[\vec{a}/\vec{\alpha}]$ and $s' \dot\cup \vec{a} \vdash t[\vec{a}/\vec{\alpha}] \sim (new\zeta.[\![P']\!])[\vec{a}/\vec{\alpha}] : \delta\mathbb{P}$. As for all m it holds $P[m/\gamma] \sim_l Q[m/\gamma]$, by bisimulation there is $Q'[\vec{a}/\vec{\alpha}]$ such that $Q[\vec{a}/\vec{\alpha}][e/\gamma] \xrightarrow{\bar{i}(\zeta)} Q'[\vec{a}/\vec{\alpha}]$ and $P'[\vec{a}/\vec{\alpha}] \sim_l Q'[\vec{a}/\vec{\alpha}]$. By Lemma 5.4.3, $s' \dot\cup \vec{a} \vdash [\![Q]\!][\vec{a}/\vec{\alpha}][e/\gamma] \xrightarrow{\mathsf{bout}:i\cdot!} u[\vec{a}/\vec{\alpha}]$ and $s' \dot\cup \vec{a} \vdash u[\vec{a}/\vec{\alpha}] \sim (new\zeta.[\![Q']\!])[\vec{a}/\vec{\alpha}] : \delta\mathbb{P}$. By the operational rules we have $s' \vdash new\vec{\alpha}_n.\lambda\gamma.[\![Q]\!] \xrightarrow{new\vec{\alpha}_n.n \mapsto \mathsf{bout}:b\cdot!} new\vec{\alpha}_n.u$. We must conclude $s' \vdash new\vec{\alpha}_n t \sim \mathcal{R} \sim new\vec{\alpha}_n u : \delta^n\delta\mathbb{P}$. By multiple applications of Lemma 5.3.16 we derive $s' \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.new\zeta.[\![P']\!] : \delta^n\delta\mathbb{P}$ from $s' \dot\cup \vec{a} \vdash t[\vec{a}/\vec{\alpha}] \sim (new\zeta.[\![P']\!])[\vec{a}/\vec{\alpha}] : \delta\mathbb{P}$. We also derive $s' \vdash new\vec{\alpha}.u \sim new\vec{\alpha}.new\zeta.[\![Q']\!] : \delta^n\delta\mathbb{P}$ from $s' \dot\cup \vec{a} \vdash u[\vec{a}/\vec{\alpha}] \sim (new\zeta.[\![Q']\!])[\vec{a}/\vec{\alpha}] : \delta\mathbb{P}$. Since $P' \sim_l Q'$, by construction of $\mathcal{R}$, we finally have

$$s' \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.new\zeta.[\![P']\!] \; \mathcal{R} \; new\vec{\alpha}.new\zeta.[\![Q']\!] \sim new\vec{\alpha}.u : \delta^n\delta\mathbb{P} \; .$$

- Suppose that $s' \vdash new\vec{\alpha}_n.\lambda\gamma.[\![P]\!] \xrightarrow{new\vec{\alpha}_n.n \mapsto \mathsf{inp}:b\cdot!} new\alpha_n.t$ for some $s' \supseteq s$. This must have been derived from $s' \dot\cup \vec{a} \vdash [\![P]\!][\vec{a}/\vec{\alpha}] \xrightarrow{e \mapsto \mathsf{inp}:i\cdot!} t[\vec{a}/\vec{\alpha}]$, that in turn must have been derived from $s' \dot\cup \vec{a} \vdash [\![P]\!][\vec{a}/\vec{\alpha}][e/\gamma] \xrightarrow{\mathsf{inp}:i\cdot!} t[\vec{a}/\vec{\alpha}]$ for $e \in s' \dot\cup \vec{a}$. Observe that

$e = a_i$ if $n = \alpha_i$ and $e = n$ otherwise. Observe also that $i = a_j$ if $b = \alpha_j$ and $i = b$ otherwise. By Lemma 5.4.3, there is a term $P'[\vec{a}/\vec{\alpha}]$ such that $P[\vec{a}/\vec{\alpha}][e/\gamma] \xrightarrow{i\zeta} P'[\vec{a}/\vec{\alpha}]$ and $s' \dot\cup \vec{a} \vdash t[\vec{a}/\vec{\alpha}] \sim (\lambda\zeta.[\![P']\!])[\vec{a}/\vec{\alpha}] : \mathbb{N} \to \mathbb{P}$. As for all m it holds $P[m/\gamma] \sim_l Q[m/\gamma]$, by bisimulation there is $Q'[\vec{a}/\vec{\alpha}]$ such that $Q[\vec{a}/\vec{\alpha}][e/\gamma] \xrightarrow{i\zeta} Q'[\vec{a}/\vec{\alpha}]$ and $P'[\vec{a}/\vec{\alpha}] \sim_l Q'[\vec{a}/\vec{\alpha}]$. By Lemma 5.4.3, $s' \dot\cup \vec{a} \vdash [\![Q]\!][\vec{a}/\vec{\alpha}][e/\gamma] \xrightarrow{\mathsf{inp}:i\cdot!} u[\vec{a}/\vec{\alpha}]$ and $s' \dot\cup \vec{a} \vdash u[\vec{a}/\vec{\alpha}] \sim (\lambda\zeta.[\![Q']\!])[\vec{a}/\vec{\alpha}] : \mathbb{N} \to \mathbb{P}$. By the operational rules we have $s' \vdash new\vec{\alpha}_n.\lambda\gamma.[\![Q]\!] \xrightarrow{new\vec{\alpha}_n.n\mapsto\mathsf{inp}:b\cdot!} new\vec{\alpha}_n.u$. We must conclude $s' \vdash new\vec{\alpha}_n.t \sim\mathcal{R}\sim new\vec{\alpha}_n.u : \delta^n\mathbb{N} \to \mathbb{P}$. By multiple applications of Lemma 5.3.16 we derive $s' \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.\lambda\zeta.[\![P']\!] : \delta^n(\mathbb{N} \to \mathbb{P})$ from $s' \dot\cup \vec{a} \vdash t[\vec{a}/\vec{\alpha}] \sim (\lambda\zeta.[\![P']\!])[\vec{a}/\vec{\alpha}] : \mathbb{N} \to \mathbb{P}$. We also derive $s' \vdash new\vec{\alpha}.u \sim new\vec{\alpha}.\lambda\zeta.[\![Q']\!] : \delta^n(\mathbb{N} \to \mathbb{P})$ from $s' \dot\cup \vec{a} \vdash u[\vec{a}/\vec{\alpha}] \sim (\lambda\zeta.[\![Q']\!])[\vec{a}/\vec{\alpha}] : \mathbb{N} \to \mathbb{P}$. Since $P' \sim_l Q'$, by construction of $\mathcal{R}$, we finally have

$$s' \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.\lambda\zeta.[\![P']\!] \;\mathcal{R}\; new\vec{\alpha}.\lambda\zeta[\![Q']\!] \sim new\vec{\alpha}.u : \delta^n(\mathbb{N} \to \mathbb{P}) \ .$$

This concludes the analysis. □

Theorem 5.4.5 *Let P and Q two closed π-calculus processes. If $\mathsf{n}(P, Q) \vdash [\![P]\!] \sim [\![Q]\!] : \mathbb{P}$, then $P \sim_l Q$.*

Proof We prove a stronger theorem:

> Let P and Q two π-calculus processes such that $\mathrm{fv}(P) = \mathrm{fv}(Q) = \vec{\alpha}_n$. If $\mathsf{n}(lhs,rhs) \vdash new\vec{\alpha}_n.[\![P]\!] \sim new\vec{\alpha}_n.[\![Q]\!] : \delta^n\mathbb{P}$, then $P \sim_l Q$.

Let

$$\mathcal{R} \;=\; \{(P, Q) \mid \mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}_n.[\![P]\!] \sim new\vec{\alpha}_n.[\![Q]\!] : \delta^n\mathbb{P} \text{ and } \mathrm{fv}(P) = \mathrm{fv}(Q) = \alpha_n\} \ .$$

We prove that $\mathcal{R}$ is a strong late bisimulation. Suppose $P \,\mathcal{R}\, Q$ and $P \xrightarrow{\ell} P'$. We perform a case analysis on ℓ.

- Suppose that $P \xrightarrow{\tau} P'$. Since transition are preserved by bijective substitution, we have $P[\vec{a}/\vec{\alpha}] \xrightarrow{\tau} P'[\vec{a}/\vec{\alpha}]$, for a set $\vec{a}$ of names fresh for both for P and Q. By Lemma 5.4.3, $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash [\![P]\!][\vec{a}/\vec{\alpha}] \xrightarrow{\tau:!} t[\vec{a}/\vec{\alpha}]$, with $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash t[\vec{a}/\vec{\alpha}] \sim [\![P']\!][\vec{a}/\vec{\alpha}] : \mathbb{P}$. By the operational rules we get the transition $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.[\![P]\!] \xrightarrow{new\vec{\alpha}.\tau:!} new\vec{\alpha}.t$, and by multiple applications of Lemma 5.3.16 we have $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.[\![P']\!] : \delta^n\mathbb{P}$. By bisimulation, there is a transition $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.[\![Q]\!] \xrightarrow{new\vec{\alpha}.\tau:!} new\vec{\alpha}.u$, with $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.u : \delta\mathbb{P}$. This must have been derived from $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash [\![Q]\!][\vec{a}/\vec{\alpha}] \xrightarrow{\tau:!} u[\vec{a}/\vec{\alpha}]$, and by Corollary 5.3.18 we have $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash t[\vec{a}/\vec{\alpha}] \sim u[\vec{a}/\vec{\alpha}] : \mathbb{P}$. By Lemma 5.4.3, $Q[\vec{a}/\vec{\alpha}] \xrightarrow{\tau} Q'[\vec{a}/\vec{\alpha}]$, with $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash u[\vec{a}/\vec{\alpha}] \sim [\![Q']\!][\vec{a}/\vec{\alpha}] : \mathbb{P}$. In turn $Q \xrightarrow{\tau} Q'$, and by multiple applications of Lemma 5.3.16 we have $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.u \sim new\vec{\alpha}.[\![Q']\!] : \delta^n\mathbb{P}$. As $\sim$ is transitive, by construction of $\mathcal{R}$ we conclude $P' \,\mathcal{R}\, Q'$.

- The case $P \xrightarrow{\overline{n}m}$ is similar to the previous one.

- Suppose $P \xrightarrow{\overline{n}(\zeta)} P'$. Since transition are preserved by bijective substitution, we have $P[\vec{a}/\vec{\alpha}] \xrightarrow{\overline{e}(\zeta)} P'[\vec{a}/\vec{\alpha}]$, for $\alpha = \mathrm{fn}(P)$, $\vec{a}$ a set of names fresh for both for P and Q, and $e = a_i$ if $n = \alpha_i$ and $e = n$ otherwise. Observe that $\zeta \notin \vec{\alpha}$. By Lemma 5.4.3, $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash [\![P]\!][\vec{a}/\vec{\alpha}] \xrightarrow{\mathsf{bout}:e\cdot!} t[\vec{a}/\vec{\alpha}]$, with $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash t[\vec{a}/\vec{\alpha}] \sim (new\zeta.[\![P']\!])[\vec{a}/\vec{\alpha}] : \delta\mathbb{P}$. By the operational rules we get $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.[\![P]\!] \xrightarrow{new\vec{\alpha}.\mathsf{bout}:n\cdot!} new\vec{\alpha}.t$, and by multiple applications of Lemma 5.3.16 we have $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.new\zeta.[\![P']\!] : \delta^n\delta\mathbb{P}$. By bisimulation, $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.[\![Q]\!] \xrightarrow{new\vec{\alpha}.\mathsf{bout}:n\cdot!} new\vec{\alpha}.u$, with $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.u : \delta^n\delta\mathbb{P}$. This must have been derived from $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash [\![Q]\!][\vec{a}/\vec{\alpha}] \xrightarrow{\mathsf{bout}:e\cdot!} u[\vec{a}/\vec{\alpha}]$, and by Corollary 5.3.18 we have $\mathsf{n}(lhs, rhs) \vdash t[\vec{a}/\vec{\alpha}] \sim u[\vec{a}/\vec{\alpha}] : \delta\mathbb{P}$. By Lemma 5.4.3, $Q[\vec{a}/\vec{\alpha}] \xrightarrow{\overline{e}(\zeta)} Q'[\vec{a}/\vec{\alpha}]$, with $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash u[\vec{a}/\vec{\alpha}] \sim (new\zeta.[\![Q']\!])[\vec{a}/\vec{\alpha}] : \delta\mathbb{P}$. In turn $Q \xrightarrow{\overline{n}(\zeta)} Q'$, and by multiple applications of Lemma 5.3.16 we have $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.u \sim new\vec{\alpha}.new\zeta.[\![Q']\!] : \delta^n\delta\mathbb{P}$. As $\sim$ is transitive, by construction of $\mathcal{R}$ we conclude $P' \mathcal{R} Q'$.

- Suppose $P \xrightarrow{n\zeta} P'$. Since transition are preserved by bijective substitution, we have $P[\vec{a}/\vec{\alpha}] \xrightarrow{e\zeta} P'[\vec{a}/\vec{\alpha}]$, for $\alpha = \mathrm{fn}(P)$, $\vec{a}$ a set of names fresh for both for P and Q, and $e = a_i$ if $n = \alpha_i$ and $e = n$ otherwise. Observe that $\zeta \notin \vec{\alpha}$. By Lemma 5.4.3, $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash [\![P]\!][\vec{a}/\vec{\alpha}] \xrightarrow{\mathsf{inp}:e\cdot!} t[\vec{a}/\vec{\alpha}]$, with $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash t[\vec{a}/\vec{\alpha}] \sim (\lambda\zeta.[\![P']\!])[\vec{a}/\vec{\alpha}] : \mathbb{N} \to \mathbb{P}$. By the operational rules we get $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.[\![P]\!] \xrightarrow{new\vec{\alpha}.\mathsf{inp}:n\cdot!} new\vec{\alpha}.t$, and by multiple applications of Lemma 5.3.16 we have $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.\lambda\zeta.[\![P']\!] : \delta^n\mathbb{N} \to \mathbb{P}$. By bisimulation, $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.[\![Q]\!] \xrightarrow{new\vec{\alpha}.\mathsf{inp}:n\cdot!} new\vec{\alpha}.u$, with $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.u : \delta^n\mathbb{N} \to \mathbb{P}$. This must have been derived from $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash [\![Q]\!][\vec{a}/\vec{\alpha}] \xrightarrow{\mathsf{inp}:e\cdot!} u[\vec{a}/\vec{\alpha}]$, and by Corollary 5.3.18 we have $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash new\vec{\alpha}.t \sim new\vec{\alpha}.u : \delta^n\mathbb{N} \to \mathbb{P}$. By Lemma 5.4.3, $Q[\vec{a}/\vec{\alpha}] \xrightarrow{e\zeta} Q'[\vec{a}/\vec{\alpha}]$, with $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash u[\vec{a}/\vec{\alpha}] \sim (\lambda\zeta.[\![Q']\!])[\vec{a}/\vec{\alpha}] : \mathbb{N} \to \mathbb{P}$. In turn $Q \xrightarrow{n\zeta} Q'$, and by multiple applications of Lemma 5.3.16 we have $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.\lambda\zeta.P' \sim new\vec{\alpha}.\lambda\zeta.[\![Q']\!] : \delta^n(\mathbb{N} \to \mathbb{P})$. It remains to prove that for all m it holds $P'[m/\zeta] \ \mathcal{R} \ Q'[m/\zeta]$. By transitivity of $\sim$ we have $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.\lambda\zeta.[\![P']\!] \sim new\vec{\alpha}.\lambda\zeta.[\![Q']\!] : \delta^n\mathbb{N} \to \mathbb{P}$ and by multiple applications of Corollary 5.3.18 we derive $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash (\lambda\zeta.[\![P']\!])[\vec{a}/\vec{\alpha}] \sim (\lambda\zeta.[\![Q']\!])[\vec{a}/\vec{\alpha}] : \mathbb{N} \to \mathbb{P}$. We perform a case analysis on m.

  - If m is a name constant, then by Lemma 5.3.2, (repeated applications of) Corollary 5.3.19, and Proposition 5.3.15, there is a $\vec{a}'$ such that $m \cap a' = \emptyset$ and $(\mathsf{n}(lhs, rhs) \cup \{m\}) \dot\cup \{\vec{a}'\} \vdash [\![P']\!][\vec{a}'/\vec{\alpha}][m/\zeta] \sim [\![Q']\!][\vec{a}'/\vec{\alpha}][m/\zeta] : \mathbb{P}$. By multiple applications of Lemma 5.3.16 we obtain $\mathsf{n}(lhs, rhs) \cup \{m\} \vdash new\vec{\alpha}.[\![P'[m/\zeta]]\!] \sim new\vec{\alpha}.[\![Q'[m/\zeta]]\!] : \delta^n\mathbb{P}$ and by construction of $\mathcal{R}$ we get $P'[m/\zeta] \sim_l Q'[m/\zeta]$.
  - Suppose that m is a name variable and $m \notin \vec{\alpha}$. By congruence, the equation $(\mathsf{n}(lhs, rhs)) \dot\cup \{\vec{a}'\} \vdash newm.((\lambda\zeta.[\![P']\!])m)[\vec{a}'/\vec{\alpha}] \sim newm.((\lambda\zeta.[\![Q']\!])m)[\vec{a}'/\vec{\alpha}] :$

Actions: $\ell \ ::= \ \overline{n}m \ \mid \ \overline{n}(\alpha) \ \mid \ n\alpha \ \mid \ \tau$

$$\frac{}{\overline{n}m.P \xrightarrow{\overline{n}m}_e P} \qquad \frac{}{n(\alpha).P \xrightarrow{nm}_e P[m/\alpha]} \qquad \frac{P \xrightarrow{\ell}_e P' \quad \alpha \notin \mathrm{fv}(\ell)}{(\boldsymbol{\nu}\alpha)P \xrightarrow{\ell}_e (\boldsymbol{\nu}\alpha)P'} \qquad \frac{P \xrightarrow{\overline{n}\alpha}_e P'}{(\boldsymbol{\nu}\alpha)P \xrightarrow{\overline{n}(\alpha)}_e P'}$$

$$\frac{P \xrightarrow{\ell}_e P'}{P \mid Q \xrightarrow{\ell}_e P' \mid Q} \qquad \frac{P \xrightarrow{\overline{n}m}_e P' \quad Q \xrightarrow{nm}_e Q'}{P \mid Q \xrightarrow{\tau}_e P' \mid Q'} \qquad \frac{P \xrightarrow{\overline{n}(\alpha)}_e P' \quad Q \xrightarrow{n\alpha}_e Q'}{P \mid Q \xrightarrow{\tau}_e (\boldsymbol{\nu}\beta)(P' \mid Q')}$$

Table 5.5: Pi-calculus: the early semantics

$\delta\mathbb{P}$ holds. By Proposition 5.3.15 and by by multiple applications of Lemma 5.3.16 we obtain $\mathsf{n}(lhs, rhs) \vdash new\vec{\alpha}.newm.[\![P'[m/\zeta]]\!] \sim new\vec{\alpha}.newm.[\![Q'[m/\zeta]]\!] : \delta^n\delta\mathbb{P}$ and by construction of $\mathcal{R}$ we get $P'[m/\zeta] \sim_l Q'[m/\zeta]$.

– Finally, suppose that m is a name variable and $m = \alpha_i$. By Proposition 5.3.15 we have $\mathsf{n}(lhs, rhs) \dot\cup \{\vec{a}\} \vdash [\![P']\!][\vec{a}'/\vec{\alpha}][a_i/\zeta] \sim [\![Q']\!][\vec{a}'/\vec{\alpha}][a_i/\zeta] : \mathbb{P}$. By multiple applications of Lemma 5.3.16 we obtain $\mathsf{n}(lhs, rhs) \cup \{m\} \vdash new\vec{\alpha}.[\![P'[\alpha_i/\zeta]]\!] \sim new\vec{\alpha}.[\![Q'[\alpha_i/\zeta]]\!] : \delta^n\mathbb{P}$ and by construction of $\mathcal{R}$ we get $P'[m/\zeta] \sim_l Q'[m/\zeta]$.

This concludes the analysis. □

5.4.2 π-calculus: the early semantics

The metalanguage can represent the early semantics of π-calculus along the same lines of the late semantics. We report in Table 5.5 the early labelled transition system for π-calculus.

In the early semantics, when an action of a process $n(\alpha).P$ is inferred, the actual name received via n is recorded in the free input action and substituted in P. Using the late rules, the variable α is instantiated only when the communication is inferred. The type of the input action assigned to π-calculus terms captures this difference. In the late semantics a process performing an input action has type $\mathsf{inp}{:}\mathbb{N} \otimes !(\mathbb{N} \to \mathbb{P})$: the type of the continuation $(\mathbb{N} \to \mathbb{P})$ ensures that the continuation is actually an *abstraction* that will be instantiated with the received name when interaction takes place. In the early semantics, the type of a process performing an input action is changed into $\mathsf{inp}{:}\mathbb{N} \otimes \mathbb{N} \to !\mathbb{P}$. Performing an input action now involves picking up a name before executing the prototypical action, and in the continuation (whose type is $\mathbb{P}$) the formal variable α has been instantiated with the received name. The complete encoding of the early semantics is reported in Table 5.6.

A development analogous to that of the late semantics shows that the early encoding preserves and reflects both internal reductions and early bisimulation.

$$\mathbb{P} \;=\; \tau{:}!\mathbb{P} \;+\; \mathsf{out}{:}\mathbb{N}\otimes\mathbb{N}\otimes!\mathbb{P} \;+\; \mathsf{bout}{:}\mathbb{N}\otimes!(\delta\mathbb{P}) \;+\; \mathsf{inp}{:}\mathbb{N}\otimes\mathbb{N}\to!\mathbb{P})$$

$$\begin{array}{rcl}
[\![\,\mathbf{0}\,]\!] & = & \mathbf{0} \\
[\![\,\overline{\alpha}\beta.p\,]\!] & = & \mathsf{out}{:}\alpha\cdot\beta\cdot!\,[\![\,p\,]\!] \\
[\![\,\alpha(\beta).p\,]\!] & = & \mathsf{inp}{:}\alpha\cdot\lambda\beta.!\,[\![\,p\,]\!]) \\
[\![\,(\boldsymbol{\nu}\alpha)p\,]\!] & = & \mathit{Res}\,(\mathit{new}\alpha.[\![\,p\,]\!]) \\
[\![\,p\mid q\,]\!] & = & [\![\,p\,]\!] \;||\; [\![\,q\,]\!]
\end{array}$$

$$\begin{array}{rcl}
\mathit{Res} & : & \delta\mathbb{P}\to\mathbb{P} \\
\mathit{Res}\;t & = & [t > \mathit{new}\alpha.\tau{:}!(x[\alpha]))\Rightarrow\tau{:}!\mathit{Res}\,x] \\
& + & \Sigma_{\beta\in\mathbb{N}}\Sigma_{\gamma\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{out}{:}\beta\cdot\gamma\cdot!(x[\alpha])\Rightarrow\mathsf{out}{:}\beta\cdot\gamma\cdot!\mathit{Res}\,x] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{out}{:}\beta\cdot\alpha\cdot!(x[\alpha])\Rightarrow\mathsf{bout}{:}\beta\cdot!x] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{bout}{:}\beta\cdot!(x[\alpha])\Rightarrow\mathsf{bout}{:}\beta\cdot!\mathit{new}\gamma\cdot\mathit{Res}\,(\mathit{new}\eta.x[\eta][\gamma])] \\
& + & \Sigma_{\beta\in\mathbb{N}}\Sigma_{\gamma\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{inp}{:}\beta\cdot\gamma\mapsto!(x[\alpha])\Rightarrow\mathsf{inp}{:}\beta\cdot\lambda\gamma.!\mathit{Res}\,(\mathit{new}\eta.x[\eta])]
\end{array}$$

$$\begin{array}{rcl}
|| & : & \mathbb{P}\&\mathbb{P}\to\mathbb{P} \\
t \;||\; u & = & [t > \tau{:}!x\Rightarrow\tau{:}!(x \;||\; u)] \\
& + & \Sigma_{\beta\in\mathbb{N}}\Sigma_{\gamma\in\mathbb{N}}[t > \mathsf{out}{:}(\beta\cdot\gamma\cdot!x)\Rightarrow[u > \mathsf{inp}{:}(\beta\cdot\gamma\mapsto!y)\Rightarrow\tau{:}!(x \;||\; y)]] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{bout}{:}(\beta\cdot!x)\Rightarrow \\
& & \qquad [u > \mathsf{inp}{:}(\beta\cdot\gamma\mapsto!y)\Rightarrow\tau{:}!\mathit{Res}\,(\mathit{new}\eta.(x[\eta] \;||\; (\lambda\gamma.y)\eta))]] \\
& + & \Sigma_{\beta\in\mathbb{N}}\Sigma_{\gamma\in\mathbb{N}}[t > \mathsf{out}{:}\beta\cdot\gamma\cdot!x\Rightarrow\mathsf{out}{:}\beta\cdot\gamma\cdot!(x \;||\; u)] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{bout}{:}\beta\cdot!x\Rightarrow\mathsf{bout}{:}\beta\cdot!\mathit{new}\eta.(x[\eta] \;||\; u)] \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{inp}{:}\beta\cdot\gamma\mapsto!x\Rightarrow\mathsf{inp}{:}\beta\cdot\lambda\eta!((\lambda\gamma.x)(\eta) \;||\; u)] \\
& + & \text{same cases with } t \text{ and } u \text{ swapped.}
\end{array}$$

where η is chosen to avoid clashes with the free name variables of u.

Table 5.6: π-calculus: encoding of the early semantics

Actions: $\ell \ ::= \ \overline{n} \ \mid \ n \ \mid \ \tau$ **Abstractions:** $\lambda\vec{\alpha}.P$ **Concretions:** $(\boldsymbol{\nu}\vec{\beta})\langle\vec{m}\rangle P$

$$\frac{}{\overline{n}\vec{m}.P \xrightarrow{\overline{n}} \langle\vec{m}\rangle P} \qquad \frac{}{n(\vec{\alpha}).P \xrightarrow{n} \lambda\vec{\alpha}.P} \qquad \frac{P \xrightarrow{\ell} P' \quad \beta \notin \mathrm{fv}(\ell)}{(\boldsymbol{\nu}\beta)P \xrightarrow{\ell} (\boldsymbol{\nu}\beta)P'}$$

$$\frac{P \xrightarrow{\ell} P'}{P \mid Q \xrightarrow{\ell} P' \mid Q} \qquad \frac{P \xrightarrow{\overline{n}} C \quad Q \xrightarrow{n} F}{P \mid Q \xrightarrow{\tau} C \bullet F}$$

where $(\boldsymbol{\nu}\vec{\alpha})\langle\vec{m}\rangle P \bullet \lambda\vec{\beta}.Q \ = \ (\boldsymbol{\nu}\vec{\alpha})(P \mid Q[\vec{m}/\vec{\beta}])$.

Table 5.7: Polyadic π-calculus: labelled transition system

5.4.3 Polyadic π-calculus

A natural and convenient extension to π-calculus is to admit processes that pass tuples of names. We present an encoding of the enriched language, the polyadic π-calculus, into new-HOPLA: polyadicity is a good testing ground for the expressivity of our language.

Writing $\vec{m}$ to denote tuples of names, the terms of polyadic π-calculus are defined by the grammar below:

$$P \ ::= \ \mathbf{0} \ \mid \ P \mid P \ \mid \ (\boldsymbol{\nu}\alpha)P \ \mid \ \overline{n}\vec{m}.P \ \mid \ n(\vec{\alpha}).P \ .$$

We report its semantics in Table 5.7. Standard conventions about abstractions and concretions apply (we refer to [Mil91] for a detailed presentation of polyadic π-calculus). We also omit symmetric rules.

The type of a polyadic π-calculus process can be defined as:

$$\begin{array}{rcl} \mathbb{P} & = & \tau{:}!\mathbb{P} \ + \ \mathsf{out}{:}\mathbb{N} \otimes \mathbb{C} \ + \ \mathsf{inp}{:}\mathbb{N} \otimes !\mathbb{F} \\ \mathbb{C} & = & 0{:}\mathbb{N} \otimes \mathbb{C} \ + \ 1{:}\delta\mathbb{C} \ + \ 2{:}!\mathbb{P} \\ \mathbb{F} & = & 3{:}\mathbb{N} \to \mathbb{F} \ + \ 4{:}\mathbb{P} \end{array}$$

Recursive types are used to encode tuples of (possibly new) names in concretions, and sequences of name abstractions in abstractions.

The function $[\![-]\!]$ translates the terms of polyadic π-calculus into new-HOPLA by structural induction, with the help of two simple auxiliary functions, `concr` and `abst`, that transform a sequence of names and a continuation process into the corresponding concretion or

abstraction. We use a Caml-like notation to specify these functions.

$$\begin{array}{rcl}
[\![\,\mathbf{0}\,]\!] & = & \mathbf{0} \\
[\![\,\overline{n}\vec{m}.P\,]\!] & = & \mathsf{out}{:}n \cdot \texttt{concr}\ \vec{m}\ P \\
[\![\,n\vec{\alpha}.P\,]\!] & = & \mathsf{inp}{:}n \cdot \texttt{abst}\ \vec{\alpha}\ P \\
[\![\,\nu\alpha P\,]\!] & = & \mathit{Res}\,(\mathit{new}\alpha.[\![\,P\,]\!]) \\
[\![\,P \mid Q\,]\!] & = & [\![\,P\,]\!]\ ||\ [\![\,Q\,]\!]
\end{array}$$

```
let rec concr n P =
  match m with
    [] -> 2 : ![[ P ]]
  | h::t -> 0 : h·(concr t P)
```

```
let rec abst a P =
  match a with
    [] -> 4 : [[ P ]]
  | h::t -> 3 : λh.(abst t P)
```

An example is useful to illustrate the use of recursive types: the concretion $\langle m_1 \ldots m_k \rangle P$ is translated into the term $0{:}m_1 \cdot 0{:}m_2 \cdot 0{:}\ldots 0{:}m_k 2{:}![\![\,P\,]\!]$. Bound names are translated into new name abstractions. For instance, the concretion $(\boldsymbol{\nu}\alpha)\langle m_1 \ldots \alpha \cdots_k \rangle P$ is represented as a term that behaves as $0{:}m_1 \cdot 0{:}\cdots\cdot 1{:}\mathit{new}\alpha.0{:}\ldots 0{:}m_k 2{:}![\![\,P\,]\!]$. The restriction $\mathit{Res} : \delta\mathbb{P} \to \mathbb{P}$ and the auxiliary maps, $\mathit{ResConc} : \delta\mathbb{C} \to \mathbb{C}$ and $\mathit{ResAbs} : \delta\mathbb{F} \to \mathbb{F}$, are responsible to push restrictions inside processes (respectively concretions and abstractions), along the lines of the restriction map of π-calculus encoding.

- Restriction applied to a process. The first summand corresponds to the equation $(\boldsymbol{\nu}\alpha)\tau.P \sim \tau.(\boldsymbol{\nu}\alpha)P$. The second summands checks that the subject of the output action is different from the name being restricted, and subsequently invokes the map *ResConc* to push the restriction inside the concretion. The third summand is similar, but for input actions.

$$\begin{array}{rcl}
\mathit{Res} & : & \delta\mathbb{P} \to \mathbb{P} \\
\mathit{Res}\ t & = & [t > \mathit{new}\alpha.\tau{:}!(x[\alpha]) \Rightarrow \tau{:}!\mathit{Res}\,x] \\
& + & \Sigma_{\beta\in\mathbb{N}}\mathsf{out}{:}\beta \cdot \mathit{ResConc}\,(\mathit{new}\alpha.\pi_\beta\pi_{\mathsf{out}}(t[\alpha])) \\
& + & \Sigma_{\beta\in\mathbb{N}}[t > \mathit{new}\alpha.\mathsf{inp}{:}\beta \cdot !(x[\alpha]) \Rightarrow \mathsf{inp}{:}\beta \cdot !\mathit{ResAbs}\,x]
\end{array}$$

- Restriction applied to a concretion. This map pushes restrictions inside concretions, thus 'implementing' the convention $(\boldsymbol{\nu}\alpha)(\boldsymbol{\nu}\vec{\beta})\langle\vec{m}\rangle P = (\boldsymbol{\nu}\vec{\beta})\langle\vec{m}\rangle(\boldsymbol{\nu}\alpha)P$ whenever $\alpha \notin \vec{m} \setminus \vec{\beta}$. Moreover, if $\alpha = m_i$ for some m_i free in the concretion, then it converts it into a a new name abstraction (that is, a 'bound output').

  A concretion is translated intro a list of (possibly fresh) names and a continuation. For the sole purpose of illustrating the map *ResConc*, we enclose the bound outputs of a concretion by parentheses. The map *ResConc* is defined by recursion on the list of names $m_1 \ldots m_k$. The first summand corresponds to a concretion $m \ldots .P$ where $m \neq \alpha$: the concretion $0{:}m\mathit{ResConc}\,(\ldots .P)$ is returned. The second summand corresponds to a concretion $\alpha \ldots .P$: the name α is then converted into a fresh name. The third summands checks the case $(\beta) \ldots .P$: the concretion $1{:}:\mathit{new}\beta.\mathit{ResConc}\,(\ldots .P)$ is then

returned. At last, if the concretion is composed by the sole continuation, then the last rule calls back the *Res* map to continue pushing the restriction inside the term.

$$\begin{array}{rcl} \mathit{ResConc} & : & \delta\mathbb{C} \to \mathbb{C} \\ \mathit{ResConc}\ t & = & \Sigma_{\gamma\in\mathbb{N}} 0{:}\gamma \cdot \mathit{ResConc}\, new\alpha.\pi_\gamma\pi_0(t[\alpha]) \\ & + & 1{:}new\alpha.\pi_\alpha\pi_0(t[\alpha]) \\ & + & 1{:}new\gamma.\mathit{ResConc}\,(new\alpha.((\pi_1(t[\alpha]))[\gamma])) \\ & + & [t > new\alpha.2{:}!(x[\alpha]) \Rightarrow 2{:}!\mathit{Res}\, x] \end{array}$$

- Restriction applied to an abstraction. The *ResAbs* map is defined by recursion on the list of the abstracted names. It simply pushes the restriction under the λ-abstractions, and if the list of abstractions is empty then calls back the *Res* map. In other words, it 'implements' the convention $(\boldsymbol{\nu}\alpha)\lambda\vec{\beta}.P = \lambda\vec{\beta}.(\boldsymbol{\nu}\alpha)P$.

$$\begin{array}{rcl} \mathit{ResAbs} & : & \delta\mathbb{F} \to \mathbb{F} \\ \mathit{ResAbs}\, t & = & 3{:}\lambda\beta.\mathit{ResAbs}\, new\alpha.((\pi_3(t[\alpha]))\beta) \\ & + & \mathit{Res}\, new\alpha.\pi_4(t[\alpha]) \end{array}$$

Parallel composition is a family of operations, defined in a simultaneous recursive definition below. For convenience we use infix notation.

- Processes in parallel with processes.

$$\begin{array}{rcl} || & : & \mathbb{P}\&\mathbb{P} \to \mathbb{P} \\ t\ ||\ u & = & \Sigma_{\beta\in\mathbb{N}}[u > \mathsf{inp}{:}\beta \cdot !y \Rightarrow \pi_\beta\pi_{\mathsf{out}}(t)\ \ ||_i\ \ y] \\ & + & \Sigma_{\beta\in\mathbb{N}}[u > \mathsf{inp}{:}\beta.!y \Rightarrow \mathsf{inp}{:}\beta \cdot !(t\ \ ||_a\ \ y)] \\ & + & \Sigma_{\beta\in\mathbb{N}}\mathsf{out}{:}\beta \cdot (t\ \ ||_c\ \ \pi_\beta\pi_{\mathsf{out}}(u)) \\ & + & [u > \tau{:}!y \Rightarrow \tau{:}!(t\ \ ||\ \ y)] \\ & + & \text{symmetric cases} \end{array}$$

- Concretions in parallel with abstractions. This map implements the application of a concretion to an abstraction, and corresponds to

$$(\boldsymbol{\nu}\vec{\alpha})\langle\vec{m}\rangle P \bullet \lambda\vec{\beta}.Q = (\boldsymbol{\nu}\tilde{\alpha})(P \mid Q[\vec{m}/\vec{\beta}])\ .$$

This map is defined by recursion on the length of the list of names $\vec{m}$.

$$\begin{array}{rcl} ||_i & : & \mathbb{C}\&\mathbb{F} \to \mathbb{P} \\ t\ ||_i\ u & = & \pi_2 x\ \ ||\ \ \pi_4 y \\ & + & \Sigma_{\beta\in\mathbb{N}}(\pi_\beta\pi_0(x)\ \ ||_i\ \ (\pi_3(y))\beta) \\ & + & \mathit{Res}\, new\alpha.((\pi_1(x))[\alpha]\ \ ||_i\ \ (\pi_3(y))\alpha) \end{array}$$

- Concretions in parallel with processes. This map corresponds to the convention

$$(\boldsymbol{\nu}\vec{\alpha})\langle\vec{m}\rangle P \mid Q = (\boldsymbol{\nu}\vec{\alpha})\langle\vec{m}\rangle(P \mid Q) \text{ for } \mathrm{fn}(Q) \cap \vec{\alpha} = \emptyset \ .$$

$$\begin{array}{rcl} ||_c & : & \mathbb{C}\&\mathbb{P} \to \mathbb{C} \\ t \ ||_c \ u & = & \Sigma_{\beta\in\mathbb{N}} 0{:}\beta.((\pi_\beta\pi_0(t)) \ ||_c \ u) \\ & + & 1{:}new\alpha.(((\pi_1 t)[\alpha]) \ ||_c \ u) \\ & + & [t > 2{:}!x \Rightarrow 2{:}!(x \ || \ u)] \end{array}$$

  where α is chosen to avoid clashes with the free name variables of u.

- Abstractions in parallel with processes. This map corresponds to the convention

$$\lambda\vec{\alpha}.P \mid Q = \lambda\vec{\alpha}.(P \mid Q) \text{ for } \mathrm{fn}(Q) \cap \vec{\alpha} = \emptyset \ .$$

$$\begin{array}{rcl} ||_a & : & \mathbb{F}\&\mathbb{P} \to \mathbb{F} \\ t \ ||_a \ u & = & 3{:}\lambda\beta.((\pi_3 t)\beta \ ||_a \ u) \\ & + & 4{:}((\pi_4 t) \ || \ u) \end{array}$$

  where β is chosen to avoid clashes with the free name variables of u.

The encoding preserves and reflects internal reductions of closed terms:

Proposition 5.4.6 *Let P be a closed polyadic π-calculus term. If $P \xrightarrow{\tau} P'$, then $\mathsf{n}(P) \vdash [\![P]\!] \xrightarrow{\tau:!} \sim [\![P']\!]$. If $\mathsf{n}(P) \vdash [\![P]\!] \xrightarrow{\tau:!} t$, then $P \xrightarrow{\tau} P'$ and $\mathsf{n}(P) \vdash t \sim [\![P']\!] : \mathbb{P}$.*

We conjecture that late strong bisimilarity is also preserved and reflected by the encoding.

5.4.4 Mobile Ambients

In this section we present an encoding of the mobility core of the Ambient Calculus. Instead of following the labelled transition system given in Chapter 4, we base our encoding on the semantics given in [CG96], extending the encoding of Mobile Ambients with public names into HOPLA given in [NW02]. As a side effect, we will point out the effects of restriction on names on the transition of Mobile Ambients.

Again, we denote *name constants* with $a, b, \ldots$, *name variables* with $\alpha, \beta, \ldots$, and we let $n, m, \ldots$ range over both name constants and name variables. The terms of the language are constructed according the following grammar:

$$P \ ::= \ \mathbf{0} \ \mid \ n[P] \ \mid \ P \mid P \ \mid \ (\boldsymbol{\nu}\alpha)P \ \mid \ \mathtt{in}\_n.P \ \mid \ \mathtt{out}\_n.P \ \mid \ \mathtt{open}\_n.P \ .$$

With respect to the HOPLA encoding, the syntax has been enriched with the restriction operator.

Types for ambients are given recursively by:

$$\begin{aligned}
\mathbb{P} &= \tau{:}!\mathbb{P} + \mathsf{in}{:}\mathbb{N} \otimes !\mathbb{P} + \mathsf{out}{:}\mathbb{N} \otimes !\mathbb{P} + \mathsf{open}{:}\mathbb{N} \otimes !\mathbb{P} + \mathsf{mvin}{:}\mathbb{N} \otimes !\mathbb{C} \\
&\quad + \mathsf{mvout}{:}\mathbb{N} \otimes !\mathbb{C} + \overline{\mathsf{open}}{:}\mathbb{N} \otimes !\mathbb{P} + \overline{\mathsf{mvin}}{:}\mathbb{N} \otimes !\mathbb{F} \\
\mathbb{C} &= 0{:}\mathbb{P}\&\mathbb{P} + 1{:}\delta\mathbb{C} \\
\mathbb{F} &= \mathbb{P} \to \mathbb{P}
\end{aligned}$$

In the HOPLA encoding, the type of concretions was $\mathbb{C} ::= \mathbb{P}\&\mathbb{P}$: it has been changed to reflect that mobility can cause extrusion of names, along the lines of what we did in the last section for polyadic π-calculus. The injections in, out, and open correspond to the basic capabilities a process can exercise, while their action on the enclosing ambients is registered by the components mvin and mvout. The injections $\overline{\mathsf{open}}$ and $\overline{\mathsf{mvin}}$ record the receptive interactions that an ambient can (implicitly) have with the environment. Again, recursive types are used in concretions to record the sequence of names that must be extruded. [6]

The translation of terms is inherited from the HOPLA paper, with the addition of the rule for the restriction operator.[7]

$$\begin{aligned}
[\![\, \mathbf{0} \,]\!] &= \mathbf{0} & [\![\, \mathtt{in}_n.P \,]\!] &= \mathsf{in}\ n \cdot ![\![\, P \,]\!] \\
[\![\, n[P] \,]\!] &= Amb\,(n, [\![\, P \,]\!]) & [\![\, \mathtt{out}_n.P \,]\!] &= \mathsf{out}\ n \cdot ![\![\, P \,]\!] \\
[\![\, P \mid Q \,]\!] &= [\![\, P \,]\!] \ ||\ [\![\, Q \,]\!] & [\![\, \mathtt{open}_n.P \,]\!] &= \mathsf{open}\ n \cdot ![\![\, P \,]\!] \\
[\![\, (\boldsymbol{\nu}\alpha)P \,]\!] &= Res\,(new\alpha.[\![\, P \,]\!])
\end{aligned}$$

The restriction map $Res : \delta\mathbb{P} \to \mathbb{P}$ filters the actions that a process emit, and blocks actions that refer to the name that is restricted. In fact, in Pure Mobile Ambients, the only scope extrusions are caused by mobility, and not by pre-actions.

$$\begin{aligned}
Res &: \delta\mathbb{P} \to \mathbb{P} \\
Res\ t &= [t > new\alpha.\tau{:}x[\alpha] \Rightarrow \tau{:}Res\ x)] \\
&+ \Sigma_{\beta\in\mathbb{N}}[t > new\alpha.\mathsf{in}{:}(\beta \cdot !x[\alpha]) \Rightarrow \mathsf{in}{:}(\beta \cdot !Res\ x)] \\
&+ \Sigma_{\beta\in\mathbb{N}}[t > new\alpha.\mathsf{out}{:}(\beta \cdot !x[\alpha]) \Rightarrow \mathsf{out}{:}(\beta \cdot !Res\ x)] \\
&+ \Sigma_{\beta\in\mathbb{N}}[t > new\alpha.\mathsf{open}{:}(\beta \cdot !x[\alpha]) \Rightarrow \mathsf{open}{:}(\beta \cdot !Res\ x)] \\
&+ \Sigma_{\beta\in\mathbb{N}}[t > new\alpha.\mathsf{mvin}{:}(\beta \cdot !x[\alpha]) \Rightarrow \mathsf{mvin}{:}(\beta \cdot !1{:}x) \\
&+ \Sigma_{\beta\in\mathbb{N}}[t > new\alpha.\mathsf{mvout}{:}(\beta \cdot !x[\alpha]) \Rightarrow \mathsf{mvout}{:}(\beta \cdot !1{:}x) \\
&+ \Sigma_{\beta\in\mathbb{N}}[t > new\alpha.\overline{\mathsf{open}}{:}(\beta \cdot !x[\alpha]) \Rightarrow \overline{\mathsf{open}}{:}(\beta \cdot !Res\ x)] \\
&+ \Sigma_{\beta\in\mathbb{N}}[t > new\alpha.\overline{\mathsf{mvin}}{:}(\beta \cdot !x[\alpha]) \Rightarrow \overline{\mathsf{mvin}}{:}(\beta \cdot !\lambda y.Res\,(new\gamma.x[\gamma](y)))]
\end{aligned}$$

Parallel composition is a family of operations, one of which is a binary operation between processes, $||_{\mathbb{P}\&\mathbb{P}}$: $\mathbb{P}\&\mathbb{P} \to \mathbb{P}$. The family is defined in a simultaneous recursive definition below.

[6] As a minor notational change, the meaning of actions-coactions $\mathsf{mvin}, \overline{\mathsf{mvin}}$ and $\mathsf{open}, \overline{\mathsf{open}}$ has been swapped with respect to the encoding of MA into HOPLA.

[7] Since the publication of [NW02], the language HOPLA has evolved. In particular the type $\Sigma_{\alpha\in A}\alpha.\mathbb{P}_\alpha$ has been decomposed into $\Sigma_{\alpha\in A}\mathbb{P}_\alpha$ and $!\mathbb{P}$, and it is definable as $\Sigma_{\alpha\in A}!\mathbb{P}_\alpha$.

- Processes in parallel with processes:

$$\begin{array}{rcl} t \;||\; u & = & \Sigma_{\beta\in\mathbb{N}}[t > \overline{\mathsf{open}}{:}\ \beta\cdot !x \Rightarrow [u > \mathsf{open}{:}\ \beta\cdot !y \Rightarrow \tau{:}\cdot !(x \;||\; y)]] \\ & + & \Sigma_{\beta\in\mathbb{N}}[t > \overline{\mathsf{mvin}}{:}\ \beta\cdot !f \Rightarrow [u > \mathsf{mvin}{:}\ \beta\cdot !c \Rightarrow \tau{:}\cdot !(c \;||\; f)]] \\ & + & [t > \tau{:}!x \Rightarrow \tau{:}!(x \;||\; u)] \\ & + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{in}{:}\beta\cdot !x \Rightarrow \mathsf{in}{:}\beta\cdot !(x \;||\; u)] \\ & + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{out}{:}\beta\cdot !x \Rightarrow \mathsf{out}{:}\beta\cdot !(x \;||\; u)] \\ & + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{open}{:}\beta\cdot !x \Rightarrow \mathsf{open}{:}\beta\cdot !(x \;||\; u)] \\ & + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{mvin}{:}\beta\cdot !x \Rightarrow \mathsf{mvin}{:}\beta\cdot !(x \;||\; u)] \\ & + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{mvout}{:}\beta\cdot !x \Rightarrow \mathsf{mvout}{:}\beta\cdot !(x \;||\; u)] \\ & + & \Sigma_{\beta\in\mathbb{N}}[t > \overline{\mathsf{open}}{:}\beta\cdot !x \Rightarrow \overline{\mathsf{open}}{:}\beta\cdot !(x \;||\; u)] \\ & + & \Sigma_{\beta\in\mathbb{N}}[t > \overline{\mathsf{mvin}}{:}\beta\cdot !x \Rightarrow \overline{\mathsf{mvin}}{:}\beta\cdot !(x \;||\; u)] \\ & + & \text{symmetric cases.} \end{array}$$

  All summands except the first two correspond to congruence rules.

- Concretions in parallel with abstractions:

$$c \;||_i\; f = snd(\pi_0 c) \;||\; f(fst(\pi_0 c)) + Res\,(new\alpha.(((\pi_1 c)[\alpha]) \;||_i\; f))$$

- Concretions in parallel with processes:

$$c \;||_c\; t = 0{:}(fst(\pi_0 c), snd(\pi_0 c) \;||\; t) + 1{:}(new\alpha.((\pi_1 c)[\alpha] \;||_c\; t))$$

- Abstractions in parallel with processes:

$$f \;||_a\; t = \lambda x.((fx) \;||\; u)$$

Remaining cases are given symmetrically.

Finally, ambient creation can be defined recursively in new-HOPLA as an operation $Amb : \mathbb{N}\&\mathbb{P} \to \mathbb{P}$:

$$\begin{array}{rcl} Amb\,(m,t) & = & [t > \tau{:}!x \Rightarrow \tau{:}!Amb\,(m,x)] \\ & + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{in}{:}\beta\cdot !x \Rightarrow \mathsf{mvin}{:}\beta\cdot !(Amb\,(m,x),\mathbf{0}) \\ & + & \Sigma_{\beta\in\mathbb{N}}[t > \mathsf{out}{:}\beta\cdot !x \Rightarrow \mathsf{mvout}{:}\beta\cdot !(Amb\,(m,x),\mathbf{0}) \\ & + & [t > \mathsf{mvout}{:}m\cdot !c \Rightarrow \tau{:}!Extr\,(m,c)] \\ & + & \overline{\mathsf{open}}{:}m\cdot !t + \overline{\mathsf{mvin}}{:}m\cdot !\lambda y.Amb\,(m,t \;||\; y) \end{array}$$

where the map $Extr : \mathbb{N}\&\mathbb{C} \to \mathbb{P}$ extrudes names across ambient's boundary after an mvout action:

$$Extr(m,c) \;=\; fst(\pi_0 c) \;||\; Amb\,(m, snd(\pi_0 c)) + Res\,(new\alpha.(Extr\,(m,(\pi_1 c)[\alpha]))) \;.$$

The denotations of ambients are determined by their capabilities: an ambient $m[t]$ can perform the internal (τ) actions of t, enter a brother ambient ($\mathsf{mvin}\ n$) if called upon to do so by an $\mathsf{in}\ n$ action of t, exit its parent ambient ($\mathsf{mvout}\ n$) if called upon to do so by an $\mathsf{out}\ n$ action of t, be exited if t so requests through an $\mathsf{mvout}\ m$ action, be opened ($\overline{\mathsf{open}}\ m$), or be entered by an ambient ($\overline{\mathsf{mvin}}\ m$); other pre-actions are restricted away. The tree-containment structure of ambients is captured in the chain of $\overline{\mathsf{open}}\ m$'s that they can perform. The only differences with the rules given in Section 4.2 is that the $\mathtt{amb}\_m$ transition has been split in two actions $\overline{\mathsf{open}}\ m$ and $\overline{\mathsf{mvin}}\ m$ to simplify the types of the continuations.

The semantics given above agrees with the reduction semantics of Mobile Ambients, possibly up to structural congruence:

Proposition 5.4.7

1. *If* $P \xrightarrow{\tau} Q$ *then* $\mathsf{n}(P) \vdash [\![P]\!] \xrightarrow{\tau:!} \sim [\![Q]\!]$;
2. *if* $\mathsf{n}(P) \vdash [\![P]\!] \xrightarrow{\tau:!} t$ *then* $P \xrightarrow{\tau} Q$ *for some* Q *such that* $\mathsf{n}(P) \vdash t \sim [\![Q]\!] : \mathbb{P}$.

This propositions relies on the fact that if $P \equiv Q$, then $\mathsf{n}(P, Q) \vdash [\![P]\!] \sim [\![Q]\!] : \mathbb{P}$. Contrary to what happens in π-calculus and polyadic π-calculus encodings, more name extrusions than strictly needed may be performed by τ:! actions. This is because the proposed encoding makes very difficult to test if given a concretion $\langle P \rangle Q$ a name α is free in the process P.

It is easy to prove that if the encodings of two processes are bisimilar in new-HOPLA, then they are *strong* reduction barbed congruent. We do not know if the converse is true (it is false if we add recursive definition of processes to the fragment of MA we consider). Yet, in MA, strong reduction barbed congruence is an extremely discriminating relation and its interest is very limited.

Notes and References

HOPLA and new-HOPLA at a glance The language new-HOPLA extends HOPLA by adding a type of names $\mathbb{N}$, a function space $\mathbb{N} \to \mathbb{P}$ as well as a type $\delta\mathbb{P}$ supporting new name generation through the abstraction $new\alpha.t$. Its operational semantics is like that of HOPLA but given at stages indexed by the current set of names. These extensions make new-HOPLA much more complex than HOPLA. If a risky parallel can be done, then we feel that the gap between the two metalanguages is analogous to the gap between CCS and π-calculus.

Towards a domain theory for concurrency The potentially complicated structure of computation paths suggests building a domain theory directly on computation paths.

This line has been followed also in what seemed originally a different direction: that of Hennessy's semantics for higher-order processes, in which process equivalence coincides with testing theories [Hen94].

In Section 5.1 we presented a domain theory based on path sets. Originally, the target denotational model of metalanguages such as HOPLA, AL, and new-HOPLA was a more informative domain theory than that based on path sets. As remarked earlier, the domain of path sets $\widehat{\mathbb{P}}$, of a path order $\mathbb{P}$, is isomorphic to characteristic functions $[\mathbb{P}^{\mathrm{op}}, \mathbf{2}]$ ordered pointwise. In modelling a process as a path set we are in effect representing a process by a characteristic function from paths to truth values $0 < 1$. If instead of these simple truth values we take sets of realisers, replacing $\mathbf{2}$ by the category of sets $\mathbf{Set}$, we obtain a functor category $[\mathbb{P}^{\mathrm{op}}, \mathbf{Set}]$, whose objects, traditionally called *presheaves*, provide an alternative 'domain' of meanings. Viewed as a process, the presheaf $X : \mathbb{P}^{\mathrm{op}} \to \mathbf{Set}$ associates to a path $p \in \mathbb{P}$ the set Xp whose elements stand for the ways in which the path can be realised. As an example, consider the two CCS processes $t = a.\mathbf{0} + a.a.\mathbf{0}$ and $u = a.a.\mathbf{0}$, These processes will be represented as the same path set, and are thus identified by the path semantics.[8] On the other hand, the process t has two different ways of realising the path $a\mathbf{0}$ while u has only one, and in this way, a presheaf representation will distinguish between the two processes. A presheaf captures the nondeterministic branching of a process and a presheaf semantics can support branching time equivalences. Presheaf models have been proposed for a wide range of process calculi, including CCS [CW96], higher-order CCS [Cat99], and π-calculus [CSW97].

More than that, presheaf categories provides a promising framework to develop a domain theory for concurrency that accounts for independence models, name generation, higher-order processes, and possesses an operational interpretation.

Name generation Behavioural properties of name generation have much in common, at least intuitively, with that of local variables in Algol-like languages and references in ML-like

[8] The full abstraction result for HOPLA links path equivalence with simulation equivalence, see [Nyg03].

languages.

Our denotational model for new-HOPLA is built out of a functor category indexed by the category $\mathcal{I}$ of finite sets and injections. This construction is based on Moggi's model of dynamic allocation [Mog90]; it is related to the 'possible worlds' models of Reynolds, Tennent and O'Hearn [Rey81, OT91], and also to Mitchell and Moggi's Kripke-style models [MM91]. Crucial ingredients of the dynamic allocation monad used here are the 'object of names', given by the inclusion functor $\mathcal{I} \hookrightarrow \mathbf{Lin}$, and the shift functor $\delta : \mathbf{Lin}^{\mathcal{I}} \to \mathbf{Lin}^{\mathcal{I}}$, given by $\delta\mathbb{P}(s) = \mathbb{P}(s+1)$. This approach has been used in [Sta96] and [FMS96] to define denotational models for π-calculus. In these works, the denotational semantics is proved fully abstract with respect to bisimulation based equivalences. It has been further carried on in [Hen01], where the behavioural preorders are the standard versions of *may* and *must* testing adapted to the π-calculus. Recent work on modelling variable-binding abstract syntax [Hof99, FPT99] is also worth mentioning.

Along a different line, notions of 'name-abstraction' and 'fresh name' are also supported by the permutation model of set theory with atoms (FM-sets), devised by Fraenkel and Mostowsky in the 1930s [GP01]. FM-sets guided the design of an ML-style metalanguage called FreshML [PG00] for programming with bound names modulo renaming.[9]

Motivations and targets of FreshML and new-HOPLA are different. Their denotational models use unrelated 'ingredients' for the allocation monad. Yet, they offer similar syntactically constructions to the programmer. In particular FreshML's *atom abstraction* and *concretion* play roles comparable to those of new-HOPLA's new name abstraction and new name application. At the same time, FreshML local declaration `new` a `in` e `end`, whose role is to create a new name and bind it to a in e has no counterpart in new-HOPLA. In fact, the term above has the same type as the expression e, while if we mimic it with new name abstraction, $new\alpha.t$, the resulting type is δ of the type of t. For people familiar with FreshML, this is why in new-HOPLA patterns cannot be decomposed into atomic patterns matching only the prototypical action.

[9] Also visit `http://www.freshml.org`.

6 Conclusion

In this dissertation we have investigated the semantics of higher-order processes, with the aim to elaborate mathematical tools to express and reason about mobile systems. Behaviour is the unifying concept guiding us all along the different developments.

We focused on two process languages, the Seal Calculus and Mobile Ambients: although they had received a lot of attention from the scientific community, their behavioural theories were largely unexplored. We proposed labelled transition semantics, and on top of them we defined bisimulation based equivalences that are sound proof techniques for reduction barbed congruence. In MA, the proposed bisimilarity is also complete: this important result required novel techniques to deal with asynchronous mobility, 'stuttering' phenomena, and unobservable movements of secret ambients.

A contribution of this thesis is to present a clear picture of the Seal Calculus. We believe that there is not much point in discussing the relative merits of the various collections of combinators, as different process languages will be more or less effective depending on the area of application. Yet, a lesson we learnt is that even if Mobile Ambients behavioural theory is much more complicated than Seal's, its syntax and reduction semantics are simpler: this greatly simplified its theoretical study.

Some reflections on the LTSs. Technically both Seal bisimilarity and MA bisimilarity are contextual bisimulations in the sense of [San94]: they solve the contextuality problem by a universal quantification over all the possible interacting contexts. In Seal we closely follow the approach of [San94]: the universal quantification is part of the definition of bisimulation. As a consequence, the bisimulation must be formulated as a delay bisimulation. In MA, on the contrary, the (essence of the) universal quantification is taken into account by the env-actions, and is integrated into the LTS. The essence of the interaction with the context is then captured by the LTS: we believe this presentation favours the emergence of a pattern in LTSs for higher-order process calculi. As a side note, a technical contribution of Seal's LTS is to extend the techniques of [Sew00] to a higher-order LTS. This approach simplifies the definition of extrusions of names, though at some cost (in particular it requires working with indexed bisimulations).

In the context of MA, we have developed powerful up-to proof methods. In particular bisimulation up to context bisimulation revealed almost indispensable when working with (variants of) context bisimulation. In fact, complex contexts may be introduced in the bisimulation game, and without appropriate tools it is awkward to prove even simple properties. The proof of the perfect firewall equation is a startling example.

We have presented a simple domain theory for nondeterministic processes within the framework of Scott and Strachey. An expressive language suitable to describe higher-order processes and name generation is based on canonical constructions on these domains. We gave an exhaustive presentation of its operational theory. The language has then been used to give semantics to rich process algebras like π-calculus, polyadic π-calculus, and Mobile

Ambients. It is probably too early to summarise the new-HOPLA experience. For instance, we are currently programming an interpreter of new-HOPLA. But the results of this thesis make, we believe, a strong case that the development of the theories underpinning higher-order computation and name generation is truly fundamental.

Further work

This thesis opens several research directions. We briefly outline some of the possibilities.

Our results on the foundations of the Seal Calculus and of Mobile Ambients are stepping stones not only to investigate more in depth their intrinsic theories, but also in the programming practise, to verify that an implementation respects its specification, or that an abstract machine is correct. One way forward is to refine our techniques to be more suitable for application specific domains. In particular, we believe that interesting labelled characterisations of typed reduction barbed congruence for MA can be derived along the lines of [HMR03].

At the same time, these results are just a contribution towards a reasonably complete understanding of higher-order process calculi. Contextual bisimilarity proved a robust tool to define sound bisimulations for higher-order process calculi. The next step is to develop techniques to eliminate, or at least to limit, the universal quantifications over all the interacting contexts. The up-to context proof technique goes in this direction, and reveals surprisingly effective. Nevertheless, it would be very satisfying if the LTS we proposed for MA could be extended along the lines of Jeffrey and Rathke approach for HOπ.

A major project is to provide axiomatisations of behavioural equivalences in higher-order process calculi. Even if equational reasoning is a central technique in process calculi, this is a completely unexplored area. Relying on the techniques introduced in Chapter 4, it seems possible to provide a complete axiomatisation of reduction barbed congruence in Ambient-like calculi. Some steps in this direction have already be undertaken [MZN03a].

The natural continuation of our research on new-HOPLA is to work out the exact relationships between the language and the denotational model $\mathbf{Lin}^{\mathcal{I}}$. We expect to obtain a full abstraction result with a natural, linear-time, contextual equivalence characterised as equality of path set denotations. Relating the operational semantics to the more informative denotational model based on presheaves presents several additional difficulties. In particular, at the time of writing, we do not know whether the function space $\mathbb{P} \to \mathbb{Q}$ exists in the natural model based on presheaves.

On the operational side, it is unclear how to provide encodings of higher-order process calculi into new-HOPLA (but the same problem arises with HOPLA) that are fully abstract with respect to the natural contextual equivalence. In particular, it is unclear how to reconcile the ideas behind Sangiorgi's contextual bisimulation with the type-driven bisimulation of new-HOPLA. Also, a major inadequacy of the work presented here is the lack of suitable weak equivalences for new-HOPLA. The language new-HOPLA does not have an analogous to internal reductions, so there is the issue of how to support more abstract operational equivalences. The paper [FCW99] provides a mathematical framework for weak bisimilarity in the context of presheaf models, and might be a good starting point.

To conclude this dissertation, we would like to focus on its general scientific project. Our studies of the operational theories of higher-order process languages aimed at a better understanding of the concurrent computation. They were also meant to work towards a domain theory for concurrency which handles higher-order and name generation. Even if providing a global mathematical framework for concurrency seems a tall order, we hope that, on the long term, this line of research will reveal as fruitful as the works of Scott and Strachey for sequential computation.

Bibliography

[ACS98] R. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195:291–324, 1998.

[AGR88] E. Astesiano, A. Giovini, and G. Reggio. Generalized bisimulation in relational specifications. In *Proc. STACS '88*, volume 294 of *LNCS*, pages 207–226. Springer Verlag, 1988.

[AKH92] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica*, 29:737–760, 1992.

[BB92] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96, 1992.

[BCC01a] M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *Proc. 4th TACS*, volume 2215 of *LNCS*. Springer Verlag, 2001.

[BCC01b] M. Bugliesi, G. Castagna, and S. Crafa. Reasoning about security in mobile ambients. In *Proc. of 12th CONCUR 2001*, LNCS. Springer Verlag, 2001.

[BCMS] M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication interference in mobile boxed ambients. Forthcoming Technical Report. An extended abstract appeared in Proc. FSTTCS'02, LNCS, Springer Verlag.

[Bou89] G. Boudol. Towards a lambda calculus for concurrent and communicating systems. In *Proc. TAPSOFT '89*, volume 351 of *LNCS*, pages 149–161. Springer Verlag, 1989.

[Bou92] G. Boudol. Asynchrony and the π-calculus. Technical Report RR-1702, INRIA-Sophia Antipolis, 1992.

[BS98] M. Boreale and D. Sangiorgi. Bisimulation in name-passing calculi without matching. In *Proc. 13th LICS*. IEEE Computer Society Press, 1998.

[BV01] C. Bryce and J. Vitek. The JavaSeal mobile agent kernel. *Autonomous Agents and Multi-Agent Systems*, 4(4):359–384, 2001.

[Cas85] I. Castellani. Bisimulation and abstraction homomorphisms. In *Proc. CAAP'85*, LNCS. Springer Verlag, 1985.

[Cas01] I. Castellani. Process algebras with localities. In J. Bergstra, A. Ponse, and S. Smolka, editors, *Handbook of Process Algebra*, pages 945–1045. North-Holland, Amsterdam, 2001.

[Cat99] G. L. Cattani. *Presheaf models for concurrency*. PhD thesis, BRICS, 1999.

[CBC02] S. Crafa, M. Bugliesi, and G. Castagna. Information flow security for Boxed Ambients. In *Foundations of Wide Area Network Computing (F-WAN)*, volume 66(3) of *ENTCS*. Elsevier Science B.V., 2002.

[CG96] L. Cardelli and A. Gordon. A commitment relation for the ambient calculus. Unpublished notes, 1996.

[CG00a] L. Cardelli and A. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proc. of the 27th POPL*, pages 365–377. ACM Press, 2000.

[CG00b] L. Cardelli and A. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000. An extended abstract appeared in *Proc. of FoSSaCS '98*.

[CGZN01] G. Castagna, G. Ghelli, and F. Zappa Nardelli. Typing mobility in the seal calculus. In *Proc. of 12th CONCUR 2001*, LNCS. Springer Verlag, 2001.

[CSW97] G. L. Cattani, I. Stark, and G. Winskel. Presheaf models for the π-calculus. In *Proc. CTCS'97*, volume 1290 of *LNCS*. Springer Verlag, 1997.

[CVZN03] G. Castagna, J. Vitek, and F. Zappa Nardelli. The seal calculus. Manuscript. Available from `www.di.ens.fr/~castagna`, 2003.

[CW96] G. L. Cattani and G. Winskel. Presheaf models for concurrency. In *Proc. CSL'96*, volume 1258 of *LNCS*. Springer Verlag, 1996.

[CW03] G. L. Cattani and G. Winskel. Profunctors, open maps and bisimulation. Manuscript, 2003.

[CZN02] G. Castagna and F. Zappa Nardelli. The seal calculus revisited: Contextual equivalence and bisimilarity. In *Proc. 22nd FSTTCS '02*, volume 2556 of *LNCS*. Springer Verlag, 2002.

[Day70] B. J. Day. On closed categories of functors. In *Reports of the Midwest Category Seminar IV*, volume 137 of *Lecture Notes in Mathematics*, pages 1–38. Springer Verlag, 1970.

[DH84] R. De Nicola and M. Hennessy. Testing equivalences for processes. *Theoretical Computer Science*, 34:83–133, 1984.

[FCW99] M. Fiore, G. L. Cattani, and G Winskel. Weak bisimulation and open maps. In *Proc. LICS '99*. IEEE Computer Society Press, 1999.

[FG98] C. Fournet and G. Gonthier. A hierarchy of equivalences for asynchronous calculi. In *Proc. 25th ICALP*, pages 844–855. Springer Verlag, 1998.

[FGL+96] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile processes. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: Concurrency Theory, 7th International Conference*, volume 1119 of *LNCS*, pages 406–421. Springer Verlag, 1996.

[FMS96] M. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the π-calculus. In *Proc. 11th LICS*. IEEE Computer Society Press, 1996.

[FMT01] G. Ferrari, U. Montanari, and E. Tuosto. A LTS semantics of ambients via graph synchronization with mobility. In *Proc. ICTCS*, volume 2202 of *LNCS*, 2001.

[Fou98] C. Fournet. *The Join-Calculus: a calculus for distributed mobile programming*. PhD thesis, École Polytechnique, Paris, 1998.

[FPT99] M. P. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In *Proc. 14th LICS*, pages 193–202. IEEE Computer Society Press, 1999.

[GC02] A. D. Gordon and L. Cardelli. Equational properties of mobile ambients. *Journal of Mathematical Structures in Computer Science*, 12:1–38, 2002. An extended abstract appeared in *Proc. FoSSaCs '99*.

[GHS02] J.C. Godskesen, T. Hildebrandt, and V. Sassone. A calculus of mobile resources. In *Proc. 10th CONCUR '02*, volume 2421 of *LNCS*, 2002.

[Gor95] A. D. Gordon. Bisimilarity as a theory of functional programming: mini-course. Notes Series BRICS-NS-95-3, BRICS, Department of Computer Science, University of Aarhus, July 1995.

[GP01] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*, 2001. A preliminary version appeared in Proc. 14th LICS.

[Hen88] M. Hennessy. *Algebraic Theory of Processes*. The MIT Press, Cambridge, Mass., 1988.

[Hen94] M. Hennessy. A fully abstract denotational model for higher-order processes. *Information and Computation*, 112(1):55–95, 1994.

[Hen01] M. Hennessy. A fully abstract denotational model for the π-calculus. *Theoretical Computer Science*, pages 53–89, 2001.

[HMR03] M. Hennessy, M. Merro, and J. Rathke. Towards a behavioural theory of access and mobility control in distributed system. In *Proc. 5th FoSSaCS '03*, LNCS. Springer Verlag, 2003.

[Hof99] M. Hofmann. Semantical analysis of higher-order abstract syntax. In *Proc. 14th LICS*, pages 204–213. IEEE Computer Society Press, 1999.

[How96] D. J. Howe. Proving congruence of bisimulation in functional programming languages. *Information and Computation*, 124(2):103–112, 1996.

[HP79] M. Hennessy and G. D. Plotkin. Full abstraction for a simple parallel programming language. In *Proc. MFCS'79*, volume 74 of *LNCS*. Springer Verlag, 1979.

[HR98] M. Hennessy and J. Riely. A typed language for distributed mobile processes. In *Proc. 25th POPL*. ACM Press, 1998.

[HRY03] M. Hennessy, J. Rathke, and N. Yoshida. Safedpi: a language for controlling mobile code. Computer Science Report 2:03, University of Sussex, 2003.

[HT91] K. Honda and M. Tokoro. An Object Calculus for Asynchronous Communications. In *Proc. ECOOP '91*, volume 512 of *LNCS*. Springer Verlag, 1991.

[HY94] K. Honda and N. Yoshida. Replication in Concurrent Combinators. In *Proc. TACS'94*, volume 789 of *LNCS*. Springer Verlag, 1994.

[HY95] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[JNW94] A. Joyal, M. Nielsen, and G. Winskel. Bisimulation from open maps. Technical Report RS-94-7, BRICS, 1994. An extract in *Proc. LICS'93*, IEEE Computer Society Press.

[JR03] A. Jeffrey and J. Rathke. Contextual equivalence for higher-order π-calculus revisited. In *Proc. MFPS XIX*, volume 83 of *ENTCS*. Elsevier, 2003. Also available as University of Sussex Computer Science Report 2002:04.

[LS00a] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proc. 27th POPL*. ACM Press, 2000.

[LS00b] F. Levi and D. Sangiorgi. Controlling interference in ambients. An extended abstract appeared in *Proc. 27th Symposium on Principles of Programming Languages*, ACM Press, 2000.

[LW84] K. G. Larsen and G. Winskel. Using informations systems to solve recursive domain equations effectively. In *Proc. Semantics of Data Types*, volume 173 of *LNCS*. Springer Verlag, 1984.

[MH01] M. Merro and M. Hennessy. Bisimulation congruences in safe ambients. Computer Science Report 5/01, University of Sussex, 2001.

[MH02] M. Merro and M. Hennessy. Bisimulation congruences in safe ambients. In *Proc. 29th POPL '02*. ACM Press, 2002.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Mil91] R. Milner. The polyadic π-calculus: a tutorial. Technical Report ECS–LFCS–91–180, LFCS, Dept. of Comp. Sci., Edinburgh Univ., October 1991. Also in *Logic and Algebra of Specification*, ed. F.L. Bauer, W. Brauer and H. Schwichtenberg, Springer Verlag, 1993.

[MM91] J. C. Mitchell and E. Moggi. Kripke-style models for typed lambda calculus. *Annals of Pure and Applied Logic*, 51:99–124, 1991.

[Mog90] E. Moggi. An abstract view of programming languages. Technical Report ECS-LFCS-90-113, LFCS, Dept. of Comp. Sci., Edinburgh Univ., 1990.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.

[MS92] R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. 19th ICALP*, volume 623 of *LNCS*, pages 685–695. Springer Verlag, 1992.

[MZN03a] M. Merro and F. Zappa Nardelli. Axioms for safe ambients. Manuscript, 2003.

[MZN03b] M. Merro and F. Zappa Nardelli. Bisimulation proof methods for mobile ambients. In *Proc. ICALP 2003*. Springer Verlag, 2003. An extended version is available as Computer Science Report 2003:01, University of Sussex, `http://cogslib.cogs.susx.ac.uk/csr_abs.php?cs`.

[NPW81] M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13(1):85–108, 1981.

[NW] M. Nygaard and G. Winskel. Domain theory for concurrency. To appear in *Theoretical Computer Science*, special issue on domain theory.

[NW02] M. Nygaard and G. Winskel. Hopla—a higher-order process language. In *Proc. CONCUR'02*, volume 2421 of *LNCS*. Springer Verlag, 2002.

[NW03] M. Nygaard and G. Winskel. Full abstraction for HOPLA. In *Proc. CONCUR'03*, LNCS. Springer Verlag, 2003.

[Nyg03] M. Nygaard. *Domain theory for concurrency*. PhD thesis, BRICS, 2003.

[OT91] P. W. O'Hearn and R. D. Tennent. Semantics of local variables. In *Applications of categories in computer science*, number 177 in London Mathematical Society Lecture Note Series, pages 217–238. CUP, 1991.

[Par81] D.M. Park. Concurrency on automata and infinite sequences. In P. Deussen, editor, *Conf. on Theoretical Computer Science*, volume 104 of *LNCS*. Springer Verlag, 1981.

[PG00] A. M. Pitts and M. J. Gabbay. A metalanguage for programming with bound names modulo renaming. In *Proc. MPC 2000*, volume 1837 of *LNCS*. Springer Verlag, 2000.

[Pit97] A. M. Pitts. Operationally-based theories of program equivalence. In P. Dybjer and A. M. Pitts, editors, *Semantics and Logics of Computation*, Publications of the Newton Institute, pages 241–298. Cambridge University Press, 1997.

[PJ03] Sewell P. and Vitek J. Composition of untrusted code: Wrappers and causality types. *Journal of Computer Security*, 2003.

[PS00] B. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. *Journal of the ACM*, 47(3):531–584, 2000.

[Rey81] J. C. Reynolds. The essence of Algol. In *Proc. 1981 International Symposyoum on Algorithmic Languages*, pages 345–372. North Holland, 1981.

[RH97] J. Riely and M. Hennessy. Distributed processes and location failures. In *Proc. ICALP '97*, volume 1256 of *LNCS*. Springer Verlag, 1997.

[San92] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, Dept. of Comp. Sci., Edinburgh Univ., 1992.

[San94] D. Sangiorgi. Bisimulation in higher-order calculi. In *Proc. IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET'94)*, pages 207–224. North-Holland, 1994.

[San96a] D. Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Information and Computation*, 131(2):141–178, 1996.

[San96b] D. Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. *Theoretical Computer Science*, 155:39–83, 1996.

[San98] D. Sangiorgi. On the bisimulation proof method. *Journal of Mathematical Structures in Computer Science*, 8:447–479, 1998.

[San01] D. Sangiorgi. Extensionality and intensionality of the ambient logic. In *Proc. 28th POPL*. ACM Press, 2001.

[Sco82] D. S. Scott. Domains for denotational semantics. In *Proc. ICALP'82*, volume 140 of *LNCS*. Springer Verlag, 1982.

[Sew00] P. Sewell. Applied π – a brief tutorial. Technical Report 498, Computer Laboratory, University of Cambridge, 2000.

[Sew03] P. Sewell. From rewrite rules to bisimulation congruences. *TCS*, 2003. To appear.

[SM92] D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". In *Proc. CONCUR '92*, volume 630 of *LNCS*, pages 32–46. Springer Verlag, 1992.

[Sta96] I. Stark. A fully-abstract domain model for the π-calculus. In *Proc. 11th LICS*, pages 36–42, Washington, DC, 1996. IEE Computer Society Press.

[SV99] P. Sewell and J. Vitek. Secure composition of insecure components. In *12th IEEE Computer Security Foundations Workshop*, 1999.

[SW01] D. Sangiorgi and D. Walker. *The π-calculus: a Theory of Mobile Processes*. Cambridge University Press, 2001.

[Tho90] B. Thomsen. *Calculi for Higher Order Communicating Systems*. PhD thesis, Department of Computing, Imperial College, 1990.

[VC99] J. Vitek and G. Castagna. Seal: A framework for secure mobile computations. In *Internet Programming Languages*, number 1686 in LNCS, pages 47–77. Springer Verlag, 1999.

[Vig99] M. G. Vigliotti. Transition systems for the ambient calculus. Master thesis, Imperial College of Science, Technology and Medicine (University of London), September 1999.

[WN95] G. Winskel and M. Nielsen. Models for concurrency. In S. Abramsky et al., editors, *Handbook of Logic in Computer Science. Volume 4. Semantic modelling*. Oxford University Press, 1995.

[WZN03] G. Winskel and F. Zappa Nardelli. Name generation and higher-order processes. Manuscript, 2003.

[ZN00] F. Zappa Nardelli. Types for the seal calculus. Master's thesis, Università di Pisa, 2000.

[ZN03] F. Zappa Nardelli. Notes on 'very early' transitions in mobile ambients. Manuscript. Available from `www.di.ens.fr/~zappa/EarlyPtma.ps.gz`, 2003.

Résumé : La présente thèse étudie les théories à la base de la mobilité, dans le but de développer des outils mathématiques de formalisation et de preuve des propriétés des systèmes mobiles.
Afin de comprendre l'impact de la mobilité sur la théorie équationelle des langages, nous nous sommes intéressé à deux calculs de processus présentant deux modèles de mobilité opposés : le Seal Calcul et les Ambients Mobiles. Dans les deux cas, nous avons donné une bisimulation étiquetée qui caractérise l'équivalence observationnelle, ce qui a permis non seulement de développer des techniques puissantes de preuves de l'équivalence de deux systèmes, mais a aussi mis en relief certaines spécificités des deux modèles de mobilité. Dans le cas des Ambients Mobiles, la caractérisation est complète : ce résultat important a nécessité l'élaboration de techniques spécifiques permettant de manipuler la mobilité asynchrone et le *stuttering*. De plus, pour la première fois dans le cas de calculs d'ordre supérieur, on a démontré que des techniques de preuve dites *up-to* sont correctes.
En parallèle, nous avons pris les catégories de pré-faisceaux comme modèle de la concurrence, et nous avons donné une lecture opérationnelle d'une catégorie de pré-faisceaux appropriée pour modéliser la création de noms et les processus d'ordre supérieur. Cela nous a conduit à la définition d'un langage concis mais expressif, nommé new-HOPLA, à même de représenter une grande variété de langages de processus. Le langage est typé, et le type d'un terme décrit la forme des chemins d'exécution qu'il peut réaliser. Il a donc fallu élaborer le typage du langage, ce qui a été compliqué par la nécessité de distinguer les noms inédits des autres, ainsi que sa théorie opérationnelle. Nous avons en outre montré comment new-HOPLA peut être utilisé pour donner une sémantique à des calculs de processus comme le π-calcul et les Ambients Mobiles.

Summary: This PhD dissertation addresses the theories underlying the mobile computation, with the aim to develop mathematical tools to express and reason about mobile systems.
We studied the impact that mobility of active computations has on the observational congruence, and in general on the equational theory of a process language. In particular, we focused on two process calculi, namely the Seal Calculus and the Ambient Calculus, that offer two opposite models of mobile computation. In both cases, we found a coinductive characterisation of observational congruence: these results not only originate powerful proof methods to prove the equivalence of two systems, but also highlight peculiarities of the two models of mobility. In the case of the Ambient Calculus, the characterisation is *complete*: this important result required the development of techniques to deal with asynchronous mobility and "stuttering" phenomena. Also, we showed the soundness of *up-to proof techniques* in the presence of higher-order computation.
We also studied presheaf categories as a general model of concurrency. We gave an operational reading of a presheaf category suitable to describe higher-order process and name generation. This led us to a compact but expressive language, called new-HOPLA, that directly encodes a rich variety of process languages. The language is typed, and the type of a term describes the shape of the computation paths it can perform. The operational theory of the metalanguage and the resulting operational equivalence have been investigated. The metalanguage has been used to give semantics to rich process algebras like π-calculus and Mobile Ambients.

Discipline : Informatique

Mots-clés : Informatique théorique, sémantique de la concurrence, processus d'ordre supérieur, équivalence comportementale, bisimulation, typage

Adresse : Département d'informatique, École normale supérieure, 45 rue d'Ulm, 75005 Paris