Année : 2002

N° attribué par la bibliothèque

THÈSE

présentée à

ÉCOLE POLYTECHNIQUE

pour obtenir le titre de

DOCTEUR EN SCIENCES

Spécialité
Informatique

soutenue par

Alan SCHMITT

le 16 septembre 2002

Titre

Conception et Implémentation de Calculs d'Agents Mobiles

Directeur de thèse : Jean-Jacques Lévy

Jury

Président	Matthew Hennessy
Directeur	Jean-Jacques Lévy
Rapporteurs	Vincent Danos Davide Sangiorgi
Examinateurs	Cédric Fournet

À mes parents, mon épouse, et mon fils.

Résumé

Nous nous intéressons aux calculs d'agents mobiles, c'est à dire à la formalisation de programmes s'exécutant en parallèle sur plusieurs machines et pouvant migrer d'une machine à l'autre, et plus précisément à des calculs remplissant trois conditions : être suffisamment riche pour être proche d'un langage de programmation ; être conçu de telle sorte qu'une implémentation distribuée soit aisée ; se prêter facilement à l'écriture de preuves ou de systèmes de types.

Nous étudions dans une première partie une traduction distribuée du calcul des ambients dans le join calcul distribué, ce dernier étant implémenté. Nous utilisons le cadre formel de cette traduction pour en prouver la correction. Ce premier travail montre certaines difficultés associées à une implémentation distribuée des ambients. Par contre, certains concepts importants de la programmation distribuée sont présents dans le calcul des ambients et absents du join calcul, comme par exemple une certaine forme de liaison dynamique liée à la localité, ou une notion de filtrage permettant de simuler des pare-feux.

C'est pourquoi dans une seconde partie nous étendons le join calcul distribué avec une notion de liaison dynamique associée à la localité. Nous développons pour ce calcul un système de types garantissant la présence d'une liaison pour tout canal dynamique utilisé. Ce calcul ne permet cependant pas le filtrage des messages envoyés d'une location à une autre.

Nous concevons enfin dans une troisième partie un calcul, le M-calcul, s'inspirant très fortement du join calcul auquel des fonctions sont ajoutées. Le M-calcul associe à toute localité, appelée domaine, un contrôleur, qui est un processus gérant l'interaction du contenu du domaine avec l'extérieur. Les messages entre domaines sont routés de proche en proche de manière transparente (il n'est pas nécessaire de spécifier le chemin menant à la destination du message) tout en laissant la possibilité à chaque domaine dont le message traverse la frontière de l'intercepter. De plus, nous nous donnons la possibilité de geler un processus pour pouvoir l'inclure dans un message. Ainsi la migration n'est qu'un cas particulier de communication et est soumise aux mêmes règles de filtrage. Afin d'obtenir une implémentation distribuée du calcul, nous présentons un système de types garantissant l'unicité du nom de chaque domaine. Ce système de types s'inspire beaucoup de celui du join calcul dynamique.

Mots Clés

Concurrence, Distribution, Agents mobiles, Langages de programmation, Calculs de processus, Typage

Addresse du Laboratoire

Projet Moscova
INRIA Rocquencourt
Domaine de Voluceau
B.P. 105
78153 Le Chesnay Cedex

Remerciements

Jean-Jacques Lévy a su à travers son enseignement éveiller en moi l'intérêt pour la recherche en informatique. Il a ensuite été un directeur de thèse amical et disponible, qui a su m'encadrer tout en me laissant une grande liberté. Je lui en suis très reconnaissant.

Je tiens à remercier l'ensemble des membres du jury, et tout particulièrement Matthew Hennessy, qui a accepté d'en être le président, ainsi que Vincent Danos et Davide Sangiorgi, qui ont courageusement lu cette thèse en détail et ont accepté d'en être les rapporteurs.

Je remercie très chaleureusement Cédric Fournet, sa collobaration dès le début de la thèse et son invitation au laboratoire de recherche de Microsoft à Cambridge pendant le printemps 2000 ont contribué de manière significative à cette thèse.

Le groupe de travail Marvel, et tout particulièrement Jean-Bernard Stefani, ont participé à la conception du M-Calcul, je les en remercie. L'accueil de Jean-Bernard lors de mon séjour à l'Inria Rhône-Alpes, son amitié et sa collaboration ont été très appréciés, je tiens à lui exprimer toute ma gratitude.

Je remercie également l'équipe Moscova, qui m'a accueilli, ainsi que tous les membres du bâtiment 8 virtuel pour les conditions de travail excellentes, les discussions animées et l'ambience chaleureuse. Partager un bureau avec James Leifer et pouvoir disserter de questions techniques, politiques, ou d'exceptions linguistiques a été extrêmement enrichissant. Je suis tout particulièrement reconnaissant envers Sylvie Loubressac et Josy Baron pour leur amitié, bonne humeur, et compétence.

Je remercie enfin mon épouse Christelle qui a su m'accompagner lors des moments difficiles, ainsi que mon fils Augustin pour avoir rapidement fait ses nuits.

Sommaire

Table des figures

Chapitre 1

Introduction

La programmation mobile et répartie est un sujet d'étude récent qui a déjà donné naissance à une foison de calculs. Ces calculs d'agents mobiles, c'est à dire ces formalisations de programmes s'exécutant en parallèle sur plusieurs machines et pouvant migrer d'une machine à l'autre, ont chacuns leurs objectifs. Certains cherchent à modéliser des domaines administratifs, comme par exemple un réseau local auquel seuls quelques agents ont accès. Le calcul des ambients [CG98] en est un exemple. D'autres, comme le join calcul distribué [Fou98], se préoccupent d'une programmation répartie bien plus transparente, où la distribution n'interfère pas ou peu avec le comportement des agents.

La notion même de calcul pour la mobilité et la répartition implique immédiatement une notion de localité. Ces localités peuvent représenter les agents, elles sont alors mobiles, ou des machines sur lesquelles les agents migrent et s'exécutent. Cette distinction n'est plus si claire si l'on souhaite également modéliser des machines pouvant se déplacer, que ce soit physiquement, comme des assistants personnels ou des ordinateurs portables, ou conceptuellement, pour donner par exemple de nouveaux droits à une machine. Ceci nous amène à considérer une seule sorte de localité et à structurer les localités de manière hiérarchique, sous forme d'un arbre. De nombreux calculs ont en effet fait ce choix. La mobilité d'un agent correspond alors à la migration d'un sous-arbre de localités.

Une deuxième notion cruciale pour les calculs distribués est la notion de communication et de synchronisation, permettant de modéliser les interactions entre agents. Cette notion se traduit habituellement soit par des rendez-vous, comme dans CCS [Mil80] ou le π-calcul [MPW92], soit par des messages asynchrones comme dans le join calcul. L'envoi de messages entre agents peut soit se faire directement, soit utiliser un troisième agent qui transporte le message et le libère à sa destination, comme dans le calcul des ambients.

Dans tous les cas, un des aspects principaux de la conception d'un calcul d'agents mobiles est la spécification de l'interaction entre localité d'une part et migration et communication de l'autre. Cette interaction peut être inexistante, le calcul est dit transparent et on obtient alors un calcul similaire au join calcul distribué, elle peut aussi être relativement locale, comme dans le cas du calcul des ambients.

Nous présentons ainsi la conception d'un tel calcul d'agents mobiles, en ne perdant pas de vue trois exigences. Tout d'abord, nous voulons que le calcul soit proche d'un langage de programmation, c'est à dire qu'il soit suffisamment riche pour exprimer facilement les nombreux concepts liés aux interactions entre agents mobiles : communication, migration, pannes, contrôle, etc. Ensuite, nous voulons que le calcul soit directement implémentable dans un cadre distribué. En particulier, il est nécessaire que toute synchronisation présente dans le calcul soit locale. Nous interdisons par exemple qu'un rendez-vous puisse avoir lieu entre deux agents pouvant être sur deux sites distincts. Enfin, ce calcul doit toujours se positionner dans un cadre formel, afin de prouver par exemple la correction de certains programmes ou de pouvoir définir des systèmes de types ayant des propriétés intéressantes.

Notre démarche est la suivante. Nous considérons tout d'abord deux calculs d'agents mobiles semblant *a priori* s'approcher des critères que nous nous donnons, bien que très différents dans l'importance qu'ils attachent à la localité : le calcul des ambients et le join calcul. Ce dernier possédant une implémentation distribuée, JoCaml [Le 98, CL99], nous présentons une traduction du calcul des ambients dans le join calcul et nous basons sur cette traduction pour obtenir une implémentation distribuée du calcul des ambients en JoCaml. Cette première étape, décrite dans le chapitre 3, nous apporte plusieurs enseignement sur ces deux calculs.

Tout d'abord, cette traduction permet de mettre à jour certaines synchronisations cachées

dans les réductions entre ambients. Nous proposons ainsi un algorithme permettant de résoudre ces synchronisations, celles-ci étant raisonnablement locales.

D'un point de vue programmation, le calcul des ambients présente plusieurs difficultés. Ainsi, tout envoi de message à un ambient distant se fait en encapsulant le message dans un ambient qui migre ensuite jusqu'à sa destination. Cette migration est rendue complexe par la nécessité de spécifier le chemin que l'ambient doit suivre et l'on risque de perdre le message si le chemin change avant que le message n'arrive à destination. De plus, la possibilité pour plusieurs ambients de porter le même nom peut rendre équivoque la destination finale associée à un chemin. Le join calcul résout ce problème en proposant une communication transparente et directe, quelles que soient les positions respectives du message et de sa destination. Cependant, une telle solution rend la notion de localité accessoire si l'on ne considère pas les pannes, puisque le comportement d'un processus ne dépend plus de l'endroit où il est.

Le join calcul distribué satisfaisant déjà la contrainte d'être facilement implémentable, nous présentons dans une deuxième partie, le chapitre 4, une extension du join calcul distribué avec une notion de liaison dynamique dépendant de la localité. Nous gardons bien entendu à l'esprit les contraintes posées par la nécessité d'implémentabilité pour les constructions que nous ajoutons. Afin de simplifier la tâche du programmeur et pour illustrer la possibilité de raisonner sur ce calcul, nous développons un système de types garantissant la présence de ressources dynamiques.

Le join calcul dynamique n'est cependant qu'une première étape. Il permet effectivement de donner du sens à l'endroit où s'exécute un processus, mais ne donne que peu de contrôle sur les interactions entre agents. En particulier, une location n'a aucun contrôle sur les messages qu'elle reçoit ou que ses sous-locations envoient. Au lieu de modifier et d'étendre la sémantique du join calcul distribué, nous concevons dans une troisième partie un nouveau calcul, le M-calcul, décrit dans le chapitre 5. Ce calcul s'inspire très fortement du join calcul mais utilise une notion de contrôleur programmable pour chaque localité. Ce contrôleur implémente la politique d'interaction du contenu de la localité avec l'extérieur. Nous unifions également migration et communication, représentant la migration comme la communication d'une localité gelée. Nous obtenons ainsi un calcul combinant certains traits intéressants des ambients, comme la localité et le contrôle des interactions, avec des notions propres au join calcul, comme les filtres de messages pour la synchronisation locale, ou la communication facilité à distance. La syntaxe du calcul ne nous permettant pas de garantir l'efficacité d'une implémentation comme dans le join calcul, où chaque canal ne peut être défini que dans une unique location, nous développons un système de type nous donnant une propriété d'unicité adaptée au calcul : il s'agit de l'unicité des noms des localités en contexte d'évaluation. La correction de ce système n'est pas immédiate, les localités pouvant être dupliquées, mais la gestion rigoureuse des noms pouvant apparaître dans les types nous permet de simplifier la preuve de correction du système.

Chapitre 2

Deux notions de calculs mobiles

Nous décrivons dans ce premier chapitre deux modèles de programmation mobile et distribuée : le join calcul distribué [FG96, FGL+96] et le calcul des ambients[CG98]. Ces deux modèles sont des calculs de processus utilisant la communication de noms, dans l'esprit du π calcul [MPW92]. La structure spatiale des calculs est rendue explicite grâce à la distribution des processus dans un arbre de locations (ou ambients) emboîtés, qui représentent à la fois des sites et des agents. Certaines étapes de calcul modifient la structure de l'arbre, modélisant ainsi la migration d'agents. Cependant, ces deux modèles traitent d'aspects différents des calculs répartis et utilisent des notions distinctes de localité.

2.1 Transparence des localités

Dans le join calcul, *la localisation est transparente* : elle peut modifier les performances, voire le comportement dans le cas de pannes, mais n'entrave en rien la capacité à communiquer ou migrer. Comme dans un système d'objets distribués, les locations communiquent entre elles dès qu'elles connaissent un nom de leurs interfaces, et ceci indépendemment de leurs positions respectives dans l'arbre des locations. Afin de conserver un modèle de programmation réaliste, les interactions entre locations sont limitées à l'envoi asynchrone de messages ou de sous-locations. En pratique, la transparence des locations est implémentée par un mécanisme de routage silencieux vers le site accueillant la location cible. Par conséquent les interactions sont locales, puisque les synchronisations ne se produisent que dans une location donnée, et ce indépendemment de ses sous-locations. Dans l'ensemble, les locations du join calcul sont adaptées à une programmation de haut niveau utilisant des messages et des agents asynchrones. Plusieurs implémentation du join calcul existent, comme par exemple JoCaml [Le 98, CL99].

Dans le calcul des ambients, *la localisation et le contrôle sont intimement mêlés* : chaque ambient correspond à une boîte opaque, et les interactions n'impliquent que des processus dans des ambients voisins. Le routage d'un ambient à l'autre est explicite et les ambients migrant doivent avoir connaissance du chemin à parcourir dans l'arbre des ambients. Si un ambient présent sur ce chemin choisi de bloquer la migration, ou si le chemin évolue au cours de la migration, il est possible que l'ambient en cours de migration se perde ou reste bloqué. Les interactions sont donc locales, ce qui n'empêche pas des synchronisations complexes entre les processus puisqu'un ambient migrant effectue un rendez-vous avec un ambient qui lui est extérieur. Dans l'ensemble, les ambients excellent à la modélisation de domaines administratifs, d'environnement très dynamiques, et de migration contrôlée dans de grands réseaux [CG98]. Une ancienne version du calcul des ambients a été implémentée par Cardelli en Java [Car97a]. Dans cette implémentation centralisée, chaque processus ambient correspond à un thread Java qui essaye continuellement de prendre un verrou sur chacun des ambients impliqués dans la réduction (dans la plupart des cas trois verrous sont nécessaires) [Car97b]. Cependant cette séquentialisation n'est pas efficace dans un cadre distribué.

Nous commençons par définir les syntaxe et sémantique du calcul des ambients mobiles, décrivons plus en détail ce calcul, puis procédons de même pour le join calcul distribué.

P	::=		processus ambient
		$n[\,P\,]$	ambient
		$P \mid P'$	composition parallèle
		$C.P$	processus gardé
		$\nu n.P$	restriction de nom
		$\langle n \rangle$	message asynchrone
		$(x).P$	réception de message
		$!P$	réplication
		$\mathbf{0}$	processus inerte
C	::=		capacité
		$\texttt{in}\ n$	migration entrante
		$\texttt{out}\ n$	migration sortante
		$\texttt{open}\ n$	dissolution d'ambient

FIG. 2.1 – Syntaxe du calcul des ambients

Les *contextes d'évaluation* $E(\cdot)$ sont définis par la grammaire :

$$E(\cdot) ::= \cdot \quad \Big| \quad P \mid E(\cdot) \quad \Big| \quad E(\cdot) \mid P \quad \Big| \quad n[\,E(\cdot)\,] \quad \Big| \quad \nu n.E(\cdot)$$

L'*équivalence structurelle* $\equiv$ est la plus petite relation d'équivalence stable par application des contextes d'évaluation, par α-conversion, et telle que :

$$\begin{array}{ll} \text{P0} & P \mid \mathbf{0} \equiv P \\ \text{P1} & P \mid P' \equiv P' \mid P \\ \text{P2} & (P \mid P') \mid P'' \equiv P \mid (P' \mid P'') \end{array} \qquad \begin{array}{ll} \text{R1} & \dfrac{n \text{ n'est pas libre dans } P}{P \mid \nu n.Q \equiv \nu n.(P \mid Q)} \\ \text{R2} & \dfrac{m \neq n}{m[\nu n.P] \equiv \nu n.m[P]} \end{array}$$

La *réduction d'ambients* $\rightarrow$ est la plus petite relation stable par équivalence structurelle, par l'application de contextes d'évaluation, et telle que :

$$\text{IN} \quad \begin{array}{ll} & m[\,P\,] \mid n[\,\texttt{in}\ m.Q \mid R\,] \\ \rightarrow & m\big[\,P \mid n[\,Q \mid R\,]\,\big] \end{array} \qquad \text{OUT} \quad \begin{array}{ll} & m\big[\,P \mid n[\,\texttt{out}\ m.Q \mid R\,]\,\big] \\ \rightarrow & m[\,P\,] \mid n[\,Q \mid R\,] \end{array}$$

$$\text{OPEN} \quad \texttt{open}\ n.Q \mid n[\,R\,] \rightarrow Q \mid R$$

$$\text{RECV} \quad \langle n \rangle \mid (x).P \rightarrow P\{^{n}/_{x}\} \qquad\qquad \text{REPL} \quad !P \rightarrow P \mid !P$$

FIG. 2.2 – Sémantique opérationnelle pour le calcul des ambients

2.2 Le calcul des ambients

2.2.1 Sémantique opérationnelle

La syntaxe est la sémantique que nous considérons pour le calcul des ambients sont données dans les figures 2.1 et 2.2.

Notre présentation est légèrement différente de celle de Cardelli et Gordon [CG98]. Elle est en fait plus proche de l'esprit de la sémantique *harness* de [GC99]. Notre équivalence structurelle est plus restrictive : elle ne permet ni d'ajouter ni de supprimer des lieurs νx, et n'est utilisable que dans un contexte d'évaluation. Notre sémantique opérationnelle décrit la création de nouvelles copies d'un processus répliqué comme étant une étape de réduction au lieu d'une étape d'équivalence structurelle. De plus, seuls les noms d'ambients sont autorisés à être envoyés dans des messages, au lieu d'autoriser également les chaînes de capacités (ces dernières sont supportées dans l'implémentation en JoCaml de la traduction du chapitre 3, mais leur encodage est assez lourd).

2.2.2 Présentation

Le calcul des ambients [CG98] décrit la migration de processus résidant dans des domaines administratifs. Un processus ambient P est soit le processus inerte $\mathbf{0}$, soit une composition parallèle $P \mid P'$, soit un processus répliqué $!P$, soit un ambient $n[\,P\,]$ nommé n, soit un processus $\nu a.P$ avec un nom local a, soit un processus $C.P$ gardé par la capacité C, soit une réception de message $(x).P$, soit un envoi de message asynchrone $\langle a \rangle$ communiquant le nom a. Les capacités C représentent les actions In/Out/Open.

Un message $\langle a \rangle$ ne peut être communiqué qu'à un récepteur $(x).P$ présent à l'intérieur de l'ambient courant. Pour envoyer un message à un récepteur situé dans un ambient distant, le message doit être encapsulé dans un ambient qui migre jusqu'à la destination du message où il est dissout afin de libérer le message, qui pourra alors interagir avec un récepteur local. Comme dans le join calcul, les ambients sont organisés en un arbre. La racine (invisible) de l'arbre représente le réseau. Les sous-ambients, plus ou moins profonds dans l'arbre, sont des entités mobiles logiques ou physiques. Les migrations sont locales et dépendent de la structure locale de l'arbre. Un ambient peut migrer hors de son père, ou entrer dans un de ses frères. Un ambient peut aussi dissoudre un ambient fils. Pour qu'un ambient atteigne un ambient distant, il doit connaître la structure de l'arbre afin d'effectuer des migrations élémentaires le rapprochant de sa destination.

Prenons comme exemple un processus $Q \stackrel{\text{def}}{=} \langle 4 \rangle$ qui envoie la valeur 4 à une imprimante sécurisée $P \stackrel{\text{def}}{=} !(x).Print_x$ (nous supposons que $Print_x$ est une primitive qui permet d'imprimer la valeur associée à x). Nous utilisons pour cette exemple un protocole inspiré de l'exemple du pare-feu présenté en [CG98] :

$$(\nu w.\, w[\; k[\,\texttt{out}\; w.\texttt{in}\; k'.\texttt{in}\; w\,] \;\mid \texttt{open}\; k'.P]) \;\mid\; k'[\,\texttt{open}\; k.Q\,]$$

Le pare-feu envoie le sous-ambient portant le nom k à k' pour vérifier sa clé. Cette clé correspond au nom k, et la vérification de la clé à l'ouverture de l'ambient k dans k'. L'ambient k contient une capacité $\texttt{in}\; w$ correspondant au nom du pare-feu. Ceci permet à k' de franchir le pare-feu pour y apporter Q qui pourra ainsi interagir avec P.

Comme dans le π calcul, l'équivalence structurelle autorise l'extrusion de portée pour les noms d'ambients liés. Cette équivalence permet aussi d'utiliser les lois de commutation et de monoïde associées à la composition parallèle.

$$\begin{aligned}
& \nu w.\, w[\; k[\,\texttt{out}\; w.\texttt{in}\; k'.\texttt{in}\; w\,] \;\mid \texttt{open}\; k'.P] \;\mid\; k'[\,\texttt{open}\; k.Q\,] && (2.1)\\
\rightarrow\;& \nu w.\, w[\,\texttt{open}\; k'.P\,] \;\mid\; k[\,\texttt{in}\; k'.\texttt{in}\; w\,] \;\mid\; k'[\,\texttt{open}\; k.Q\,] && (2.2)\\
\rightarrow\;& \nu w.\, w[\,\texttt{open}\; k'.P\,] \;\mid\; k'[\; k[\,\texttt{in}\; w\,] \;\mid \texttt{open}\; k.Q] && (2.3)\\
\rightarrow\;& \nu w.\, w[\,\texttt{open}\; k'.P\,] \;\mid\; k'[\,\texttt{in}\; w \mid Q\,] && (2.4)\\
\rightarrow\;& \nu w.\, w[\,\texttt{open}\; k'.P \mid k'[\,Q\,]\,] &&\\
\rightarrow\;& \nu w.\, w[\,P \mid Q\,] &&\\
\rightarrow\;& \nu w.\, w[\,P \mid (x).Print_x \;\mid\; \langle 4 \rangle\,] && (2.5)\\
\rightarrow\;& \nu w.\, w[\,P \mid Print_4\,] && (2.6)
\end{aligned}$$

Le processus de l'exemple est équivalent à (2.1), après extrusion de la portée de w. Le sous-ambient k commence par sortir du pare-feu, en utilisant la capacité Out, pour donner (2.2). L'ambient k

peut alors entrer dans l'ambient k' en utilisant sa capacité In, donnant (2.3). Ensuite, l'ambient k est dissout par la capacité Open de k', donnant (2.4). Le calcul se poursuit par des réductions In/Open/Repl jusqu'à (2.5). Enfin, les processus P et Q sont voisins dans l'ambient w et peuvent communiquer (2.6).

Ainsi, les mouvements d'ambients sont des séquences de réductions In/Out/Open qui modifient la structure arborescente des ambients. Comme dans le join calcul, un ambient se déplace avec tous ses sous-ambients. Chaque réduction In ou Out déplace un sous-arbre dans l'arbre des ambients, alors qu'une réduction Open supprime un nœud de l'arbre, et attache les fils du nœud dissout à leur grand-père. Les communications par la règle Recv sont des communications par un canal anonyme dans l'ambient courant.

2.3 Le join calcul

2.3.1 Sémantique opérationnelle

La syntaxe et la sémantique que nous utilisons pour le join calcul sont données en figures 2.3 et 2.4.

Un *chemin* α est une chaîne de noms de locations a, b, etc. Les *locations actives* sont des locations qui ne sont pas sous un `def` ; elles peuvent être emboîtées. Le chemin d'une location active $a\,[D : P]$ est αa si le chemin de la location englobante est α. Une *configuration* est une conjonction de locations dépliées telle que chaque location active possède un nom unique, telle que l'ensemble des chemins de toutes les locations actives est fermé par préfixe (en d'autres termes, les locations actives forment un arbre dont les nœuds sont les noms des locations), et telle que chaque canal est défini dans au plus une location active. Dans une configuration, une location a est dite *repliée* si aucun chemin d'une location dépliée ne mentionne a.

Les noms du join calcul peuvent être liés en tant que paramètres $\widetilde{y}$ d'un message requis $x\langle\widetilde{y}\rangle$ dans le filtre d'une règle de réaction, ou en tant que noms définis par D dans `def` D `in` P. La définition $a\,[D : P]$ définit a et les noms définis par D. Une définition contenant le filtre $x\langle\widetilde{y}\rangle$ dans une règle définit x. Nous disons qu'une définition D définit *localement* x si D définit x sans inspecter les locations repliées contenues dans D.

Nous remarquons que nous utilisons des variantes du join calcul dans les chapitres 3 et 4. Dans le chapitre 3, nous ajoutons des constructions primitives pour éviter de trop nombreux codages. Dans le chapitre 4, nous transformons la règle d'équivalence structurelle Scope en une règle de réduction (permettant de dissoudre un lieur `def`). Nous détaillerons et justifierons ces modifications mineures au moment opportun.

2.3.2 Présentation

Le join calcul [FG96] est une variante du π calcul asynchrone [MPW92, Bou92] possédant une implémentation distribuée [Le 98] et un traitement uniforme des canaux de communications locaux et distants. Ce travail ne considère que le join calcul distribué sans pannes (simplement appelé join calcul par la suite) [FGL$^+$96, Fou98].

Les termes du join calcul sont soit des processus, soit des définitions, soit des configurations. Un processus P peut être soit le processus inerte $\mathbf{0}$, soit une composition parallèle de processus $P_1 \mid P_2$, soit l'envoi asynchrone du message $\widetilde{y}$ sur le canal x, noté $x\langle\widetilde{y}\rangle$, soit une requête de migration `go` $a; Q$ de la location courante dans la location a suivie de la continuation Q, soit un processus Q avec une définition locale D, noté `def` D `in` Q.

Les définitions de canaux expriment le comportement associé à la réception de messages sur ces canaux. Une définition D peut être soit la définition vide $\top$, soit une composition de définitions D_1, D_2, soit une règle de réaction $J \triangleright P$ consommant les messages correspondant au filtre J pour exécuter P, soit une location repliée $a\,[D : P]$ ayant pour nom a, pour définition D et pour processus P. Les filtres J modélisent la réception et la synchronisation de messages : un filtre de message $x\langle\widetilde{y}\rangle$ attend qu'un message $\widetilde{y}$ sur le canal x soit présent ; le filtre $J \mid J'$ attend que les filtres J et J' soient satisfaits pour être satisfait.

Les configurations sont les termes en cours d'exécution du join calcul. Une configuration peut être soit la configuration vide Ω, soit une conjonction de configurations $\mathcal{S} \parallel \mathcal{S}'$, soit une location dépliée $\alpha[D : P]$ ayant pour chemin α, pour définition active D et pour processus actif P.

Par exemple, un générateur de compteurs est modélisé par le processus suivant :

```
def counter⟨x, κ⟩ ▷
    def count⟨n⟩ | inc⟨κ'⟩ ▷ count⟨n+1⟩ | κ'⟨⟩
```

P	::=		processus du join calcul
		$\mathbf{0}$	processus inerte
	$\mid$	$P \mid P'$	composition parallèle
	$\mid$	$x\langle\widetilde{y}\rangle$	message asynchrone
	$\mid$	$\texttt{go}\ a; P$	requête de migration
	$\mid$	$\texttt{def}\ D\ \texttt{in}\ P$	définition locale
D	::=		définition du join calcul
		$\top$	définition vide
	$\mid$	D, D'	composition
	$\mid$	$J \triangleright P$	règle de réaction
	$\mid$	$a\,[D : P]$	location repliée (appelée a, exécutant D et P)
J	::=		filtre
		$x\langle\widetilde{y}\rangle$	message requis
	$\mid$	$J \mid J'$	synchronisation
$\mathcal{S}$	::=		configuration
		Ω	configuration vide
	$\mid$	$\mathcal{S} \parallel \mathcal{S}'$	composition de configurations
	$\mid$	$\alpha\,[D : P]$	location dépliée (avec chemin α, exécutant D et P)

FIG. 2.3 – Syntaxe du join calcul distribué

L'*équivalence structurelle* $\equiv$ (pour les processus et les définitions) est la plus petite relation d'équivalence stable par application des contextes $(\cdot,\cdot)$; $(\cdot \mid \cdot)$; $\alpha[\ \cdot : \cdot\ ]$ et $(\cdot \parallel \cdot)$, stable par α-conversion sur les noms liés, et telle que :

$$\begin{array}{llllll}
P0 & P \mid \mathbf{0} \equiv P & P1 & P \mid P' \equiv P' \mid P & P2 & (P \mid P') \mid P'' \equiv P \mid (P' \mid P'') \\
D0 & D, \top \equiv D & D1 & D, D' \equiv D', D & D2 & (D, D'), D'' \equiv D, (D', D'') \\
S0 & \mathcal{S} \parallel \Omega \equiv \mathcal{S} & S1 & \mathcal{S} \parallel \mathcal{S}' \equiv \mathcal{S}' \parallel \mathcal{S} & S2 & (\mathcal{S} \parallel \mathcal{S}') \parallel \mathcal{S}'' \equiv \mathcal{S} \parallel (\mathcal{S}' \parallel \mathcal{S}'')
\end{array}$$

$$\textsc{Tree}\ \frac{a \text{ replié}}{\begin{array}{rl} & \alpha\,[a\,[D' : P'], D : P] \\ \equiv & \alpha a\,[D' : P'] \parallel \alpha\,[D : P] \end{array}} \qquad \textsc{Scope}\ \frac{\begin{array}{l}\text{les noms définis dans } D' \text{ ne sont pas libres} \\ \text{dans toute la configuration}\end{array}}{\begin{array}{rl} & \alpha\,[D : P \mid \texttt{def}\ D'\ \texttt{in}\ P'] \\ \equiv & \alpha\,[D, D' : P \mid P'] \end{array}}$$

La *réduction du join calcul* $\rightarrow$ est la plus petite relation sur les configurations qui est stable par équivalence structurelle et telle que :

$$\textsc{Comm}\ \frac{x \text{ est localement défini dans } D'}{\begin{array}{rl} & \alpha\,[D : x\langle\widetilde{v}\rangle \mid P] \parallel \beta\,[D' : P'] \parallel \mathcal{S} \\ \rightarrow & \alpha\,[D : P] \parallel \beta\,[D' : x\langle\widetilde{v}\rangle \mid P'] \parallel \mathcal{S} \end{array}} \qquad \textsc{Join}\ \frac{\begin{array}{l}\sigma \text{ opère sur les arguments} \\ \text{formels des messages de } J\end{array}}{\begin{array}{rl} & \alpha\,[D, J \triangleright Q : J\sigma \mid P] \parallel \mathcal{S} \\ \rightarrow & \alpha\,[D, J \triangleright Q : Q\sigma \mid P] \parallel \mathcal{S} \end{array}}$$

$$\textsc{Go}\ \frac{}{\begin{array}{rl} & \alpha\,[a\,[D : P \mid \texttt{go}\ b; Q], D_\alpha : P_\alpha] \parallel \beta b\,[D' : P'] \parallel \mathcal{S} \\ \rightarrow & \alpha\,[D_\alpha : P_\alpha] \parallel \beta b\,[a\,[D : P \mid Q], D' : P'] \parallel \mathcal{S} \end{array}}$$

FIG. 2.4 – Sémantique opérationnelle pour le join calcul distribué

```
      ,  count⟨n⟩ | get⟨κ′⟩ ▷ count⟨n⟩ | κ′⟨n⟩
    in count⟨x⟩ | κ⟨get, inc⟩
in P
```

La définition des canaux `get`, `inc`, et `count` est locale à la définition de `counter`. L'accès à `counter` est autorisé dans P, ses arguments sont sa valeur initiale `x` et une continuation κ. Considérons par exemple le processus P :

```
def κ1⟨g, i⟩ ▷
    def κ2⟨⟩ ▷ 0 in i⟨κ2⟩ | i⟨κ2⟩ | g⟨print⟩
in counter⟨3, κ1⟩ | counter⟨10, κ1⟩
```

Nous supposons ici que `print` est un canal permettant d'imprimer son argument sur un terminal. Deux compteurs sont créés avec les valeurs initiales 3 et 10, et deux valeurs sont imprimées (dans un ordre non spécifié) : la première est une valeur de $\{3, 4, 5\}$, la deuxième une valeur de $\{10, 11, 12\}$. Il est possible de définir des abréviations pour ne pas avoir à utiliser autant de continuations, mais nous ne le ferons pas ici.

Notre compteur peut être localisé et mobile. Par exemple, considérons le processus :

```
def counter⟨user, x, κ⟩ ▷
    def a [    count⟨n⟩ | inc⟨κ′⟩ ▷ count⟨n+1⟩ | κ′⟨⟩
             , count⟨n⟩ | get⟨κ′⟩ ▷ count⟨n⟩ | κ′⟨n⟩
             : go user; (count⟨x⟩ | κ⟨get, inc⟩) ] in 0
in Q
```

Les canaux locaux `get`, `inc`, et `count` sont désormais attachés à une location `a` et se déplaceront avec elle. La destination de cette location, `user`, est un nouvel argument de `counter`.

Les locations sont structurées en arbre. De manière intuitive, la location à la racine de l'arbre est le réseau, les sous-locations immédiates sont les adresses IP, les sous-location plus profondes sont des entités logiques mobiles. Dans notre exemple, la location `a` est une sous-location de la location courante de `counter`, mais elle deviendra une sous-location de `user` après avoir effectué la migration `go user`, et ce avant l'exécution de la continuation $\texttt{count⟨x⟩} \mid \kappa\texttt{⟨get, inc⟩}$.

Afin d'examiner l'évolution dynamique de notre exemple, nous notons le compteur localisé et mobile `def` D_c `in` Q, où D_c est de la forme :

```
counter⟨user, x, κ⟩ ▷
def a [D_{i,g} : go user; (count⟨x⟩ | κ⟨get, inc⟩) ] in 0
```

et où $D_{i,g}$ est composé des deux règles définissant `count`, `inc`, et `get` comme auparavant.

Supposons que notre exemple est exécuté dans la location racine anonyme, et que Q est en fait une location `u` exécutant le processus P défini plus tôt (auquel on a ajouté le nom de location `u` aux messages sur `counter`).

```
[⊤ : def D_c in def u[⊤ : P] in 0]
```

En effaçant le lieur `def` (par la règle Scope), cette définition est équivalente à :

```
[D_c : def u[⊤ : P] in 0]
```

En effaçant maintenant le lieur `def` de `u`, puis celui de la définition de κ_1 dans P (que nous appelons ici D_{κ_1}), nous obtenons la définition équivalente :

```
[D_c , u[D_{κ1} : counter⟨u,3,κ1⟩ | counter⟨u,10,κ1⟩ ] : 0]
```

En utilisant la structure en arbre des locations (règle Tree), cette définition est équivalente à :

```
[D_c : 0]  ‖  u[D_{κ1} : counter⟨u,3,κ1⟩ | counter⟨u,10,κ1⟩ ]
```

Ces différentes étapes font partie de l'équivalence structurelle définie sur les termes du join calcul. Celle-ci demande de prendre quelques précautions dans le traitement des noms liés par la construction `def`, qui lie tous les noms de canaux et de locations qui y sont définis, afin d'éviter des captures. L'équivalence structurelle prend aussi en compte la structure de monoïde associée à l'opérateur | dans les processus et à l'opérateur "," dans les définitions, ainsi que la structure d'arbre des locations.

En outre, les termes du join calcul peuvent évoluer par les règles de réduction. La première, Comm, envoie un message émis sur un certain canal dans la location (unique) définissant ce message. Dans notre exemple, la définition se réduit ainsi en :

$[D_c$: counter⟨u,3,κ_1⟩$]$ ‖ u$[D_{\kappa_1}$: counter⟨u,10,κ_1⟩$]$

La deuxième règle, JOIN, remplace les messages satisfaisant un filtre d'une règle de réaction par le processus associé de cette même règle de réaction, en substituant aux paramètres formels du filtre les arguments des messages. Notre exemple se réduit maintenant en :

$[D_c$: def a$[D_{i,g}$: go u; (count⟨3⟩ | κ_1⟨get, inc⟩)] in **0**$]$
‖ u$[D_{\kappa_1}$: counter⟨u,10,κ_1⟩$]$

Cette définition est structurellement équivalente, en utilisant la règle SCOPE, à :

$[D_c$, a$[D_{i,g}$: go u; (count⟨3⟩ | κ_1⟨get, inc⟩)] : **0**$]$
‖ u$[D_{\kappa_1}$: counter⟨u,10,κ_1⟩$]$

La troisième règle de réduction, GO, provoque la migration subjective de la location repliée contenant la requête de migration go vers la location portant le nom correspondant à la requête. Nous obtenons donc après réduction :

$[D_c$: **0**$]$
‖ u$[D_{\kappa_1}$, a$[D_{i,g}$: count⟨3⟩ | κ_1⟨get, inc⟩] : counter⟨u,10,κ_1⟩$]$

Cette dernière définition est équivalente à :

$[D_c$: **0**$]$
‖ u.a$[D_{i,g}$: count⟨3⟩ | κ_1⟨get, inc⟩$]$
‖ u$[D_{\kappa_1}$: counter⟨u,10, κ_1⟩$]$

Nous remarquons que à ce point de la réduction, la définition de notre générateur de compteur D_c reste présente au niveau du réseau alors qu'une instance d'un compteur est dans la sous-location a de la location u. Le calcul peut alors continuer en générant le deuxième compteur ou en exécutant la continuation κ_1.

Les trois règles de réduction ainsi décrites montrent les différentes évolutions d'un terme du join calcul. Il est crucial de remarquer que la mobilité influe sur la localisation des termes mais pas sur leur fonctionnalité, puisque les liaisons sont conservées par les règles de réduction. Un autre aspect important concerne la linéarité des noms de canaux et de locations : tout canal est défini dans une location unique, le nom de la location où un canal est défini reste inchangé au cours des réductions, et chaque location porte un nom unique. Ce dernier aspect est essentiel à une implémentation distribuée, puisqu'il permet de déterminer de manière locale où envoyer un message ou une location.

Plusieurs implémentations distribuées du join calcul existent, dont JoCaml [Le 98].

Chapitre 3

Implémentation des ambients dans le join calcul

Ce chapitre présente une implémentation distribuée et très parallèle du calcul des ambients mobiles de Luca Cardelli et Andrew Gordon [CG98] dans le langage de programmation JoCaml [Le 98, CL99]. Le calcul des ambients modélise simplement et esthétiquement la notion de calcul distribué et mobile. Cependant, jusqu'à ce jour, aucune implémentation distribuée de ce calcul n'était disponible. Il pourrait sembler *a priori* qu'une telle implémentation soit facile à réaliser, en particulier en utilisant un langage autorisant la distribution et la migration forte comme JoCaml. Néanmoins, nous avons rencontré certaines difficultés intéressantes, et nous avons dû considérer des choix de conceptions inattendus.

Nous rappelons brièvement ce qu'est le calcul des ambients, décrit plus en détail en section 2.2. Les ambients mobiles sont des processus actifs emboîtés. Leur structure peut être modifiée de trois manières élémentaires : un ambient peut entrer dans un ambient voisin (étape In), un ambient peut sortir de son ambient père (étape Out), un ambient peut ouvrir (ou dissoudre) un de ses ambients fils (étape Open). Certaines des ces étapes élémentaires impliquent donc plusieurs ambients. Par exemple, lors d'une étape In, l'ambient migrant et l'ambient destination participent à la réduction. De la même manière, lors d'une étape Out, l'ambient migrant et l'ambient père sont impliqués. De plus, plusieurs étapes élémentaires peuvent s'appliquer à une configuration donnée, et certaines de ces étapes sont mutuellement exclusives.

Dans un cadre distribué, il est possible que chaque ambient soit exécuté sur un site (une machine) différent. Par conséquent, certaines étapes élémentaires impliquent plusieurs sites et peuvent avoir pour conséquence la migration d'un ambient d'un site à un autre. On peut considérer que chaque étape élémentaire se décompose en fait en deux micro-étapes. La première micro-étape consiste à vérifier que la structure de la configuration autorise bien l'étape à avoir lieu. On se donne alors la garantie qu'elle sera complétée : on s'astreint à la réaliser. La seconde micro-étape est la migration elle-même. Dans un cadre distribué, la première micro-étape requiert donc une forme de synchronisation distribuée. On pourrait pour ce faire utiliser une primitive globale de synchronisation, présente par exemple dans le système d'exploitation ou fournie par le réseau, mais supposer l'existence d'une telle solution n'est pas réaliste pour les grands réseaux.

Afin de résoudre ce problème, une première grande catégorie de solutions consiste à utiliser des verrous et des sections critiques pour rendre séquentielle l'exécution de l'implémentation de chaque étape élémentaire. Par exemple, tous les ambients prenant part à une étape élémentaire pourraient être gelés temporairement. Cependant une telle solution ne peut être symétrique, pour la même raison qu'il n'y a pas de solution symétrique au problème du dîner des philosophes. Pour une configuration donnée, il est donc nécessaire de distinguer certains ambients qui mettraient le verrou en place. Par exemple, un ambient unique pourrait s'occuper de la synchronisation de toutes les étapes de tous les ambients. Une autre solution, plus naturelle, utilise la structure arborescente des configurations d'ambients. On pourrait ainsi décider qu'un ambient père contrôle la synchronisation des étapes impliquant ses fils immédiats. Cette solution est semblable à celle utilisée dans l'implémentation de Luca Cardelli, implémentation centralisée d'une ancienne version du calcul des ambients en Java [Car97a, Car97b]. Dans cette implémentation, chaque processus correspond à un thread Java qui essaye sans arrêt de prendre un verrou sur chaque ambient impliqué dans la migration (dans la plupart des cas, trois verrous sont nécessaires). Un avantage d'une telle solution séquentielle est qu'elle facilite la preuve de correction de l'implémentation. Cependant il faut s'assurer qu'un phénomène de deadlock ne peut pas arriver. Par exemple, dans

l'implémentation de Luca Cardelli, les verrous sont toujours pris dans un ordre dicté par la structure arborescente, ce qui permet d'éviter les deadlocks au prix d'une séquentialité accrue. L'inconvénient majeur d'une utilisation de verrous est bien entendu la perte de parallélisme. De plus, si des verrous doivent être pris sur des sites distants, le délai avant qu'ils ne soient rendus peut être long. Ceci introduit donc des goulets d'étranglement au niveau les plus hauts de la hiérarchie (les plus proches de la racine), aggravés par la nécessité pour un ambient qui migre d'un point à un autre d'effectuer une série d'étapes Out afin de se trouver dans un ambient qui englobe à la fois la source et la destination de la migration.

Un deuxième ensemble de solutions offre la possibilité d'utiliser uniquement des mécanismes distribués asynchrones. Les étapes élémentaires sont toujours décomposées en micro-étapes, cependant ces micro-étapes n'utilisent que des synchronisations locales. L'exécution d'une étape correspond donc à l'exécution d'un protocole utilisant plusieurs messages et minimisant le gel d'autres actions. Par exemple, notre solution ne bloque jamais l'exécution d'étapes impliquant le père de l'ambient migrant, alors que c'était le cas dans l'implémentation centralisée décrite précédemment [Car97a, Car97b]. Par contre, ce parallélisme accru rend plus complexe la preuve de correction de notre implémentation. La migration proprement dite (la deuxième micro-étape dans la description ci-dessus) peut aussi poser des problèmes de synchronisation qui sont typiquement résolus par des mécanismes qui font suivre les messages. Dans notre cas, nous utilisons directement la migration forte de JoCaml et ne détaillons pas ces aspects.

Ce chapitre présente donc un algorithme distribué permettant d'implémenter le calcul des ambients, sous la forme d'une traduction vers le join calcul [FG96, FGL$^+$96], qui est le calcul de processus modélisant JoCaml. Cet algorithme permet de rendre explicites les synchronisations cachées dans les étapes élémentaires du calcul des ambients, et nous renseigne ainsi sur les difficultés potentielles que d'autres implémentations distribuées pourraient rencontrer. La traduction est ensuite directement implémentée en JoCaml, nous donnant ainsi une implémentation distribuée du calcul des ambients. Nous détaillons également une preuve que la traduction est correcte. Cette preuve de correction procède en deux étapes : nous prouvons tout d'abord la correction de l'algorithme lui-même en utilisant des simulations couplées, puis nous prouvons la correction de la traduction dans le join calcul en utilisant une bisimulation hybride. D'un point de vue technique, la première partie de la preuve est une première application des simulations couplées [PS92] dans le cadre d'un calcul asynchrone avec réductions. Cette partie introduit un calcul étendu des ambients dans lequel des états transitoires sont représentés, et ne dépend pas du calcul cible de la traduction. La deuxième partie est une application non triviale de la technique des diagrammes décroissants [Oos94]. Mis ensemble, ces deux résultats de correction montrent que notre traduction reflète et préserve un large spectre de prédicats standards d'observations.

Ce chapitre est organisé comme suit. Dans la section 3.1, nous présentons l'algorithme asynchrone et une traduction formelle des processus du calcul des ambients en processus du join calcul. Dans la section 3.2, nous spécifions la correction en terme d'observables. Dans la section 3.3, nous nous intéressons à la correspondance opérationnelle entre un processus et sa traduction. À cette fin, nous introduisons un calcul étendu reflétant les états transitoires dus à la traduction. Dans la section 3.4, nous donnons nos principaux résultats techniques, ainsi qu'une intuition sur leurs preuves. Dans la section 3.5, nous décrivons les aspects pratiques de l'implémentation en JoCaml. Les preuves de correction des résultats de la section 3.4 sont présentées en sections A.1 et A.2. Nous concluons en section 3.6.

Nous rappelons que nous avons présenté dans le chapitre 2 une sémantique opérationnelle pour le join calcul et pour le calcul des ambients. Des informations supplémentaires sur ces calculs peuvent être trouvées en [FGL$^+$96] pour le join calcul et en [CG98] pour le calcul des ambients.

3.1 Des ambients au join calcul

Nous commençons par décrire l'algorithme asynchrone, puis nous spécifions une traduction des ambients dans le join calcul. Pour simplifier la présentation, nous commençons par le fragment du calcul des ambients donné par la grammaire :

$$P ::= \; a[\,P\,] \;\mid\; \texttt{in}\; a.P \;\mid\; \texttt{out}\; a.P \;\mid\; P \mid P' \;\mid\; \mathbf{0}$$

Nous introduirons ensuite l'étape Open ainsi que les autres primitives du calcul des ambients.

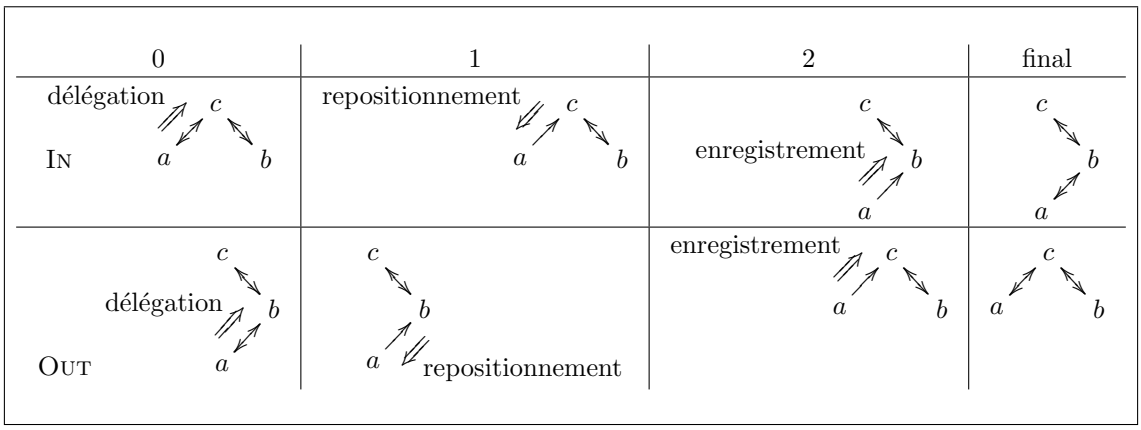


Fig. 3.1 – L'algorithme sans Open

3.1.1 Un algorithme asynchrone

L'arbre des ambients est représenté par un arbre doublement lié, où chaque père pointe vers ses fils et chaque fils vers son père. Chaque nœud de l'arbre implémente un ambient et comporte :

- un nom d'ambient ;
- des processus en parallèle qui ne sont pas des ambients, comme par exemple $\texttt{in}\, b.P$ ou $\texttt{out}\, a.c[Q]$;
- un *contrôleur* qui supervise les étapes ayant lieu dans cet ambient et les sous-ambients immédiats.

Comme chaque nœud ne s'exécute pas forcément sur le même site physique, nous restreignons la communication entre contrôleurs à des messages asynchrones. De plus, un même nom pouvant être associé à plusieurs ambients, nous donnons à chaque nœud un identificateur unique. Malgré cela, nous utiliserons souvent de manière informelle le nom d'un ambient pour désigner un nœud, au lieu de son identificateur unique.

Chaque ambient possède un pointeur vers chacun de ses sous-ambients immédiats, ainsi qu'un pointeur vers son père, conservant ainsi suffisamment d'information pour pouvoir communiquer avec eux. Les pointeurs vers les sous-ambients sont utilisés pour les contrôler ; le pointeur vers le père est utilisé pour communiquer des demandes d'action pour l'ambient courant. Par définition des étapes de réduction des ambients, le père d'un ambient voulant effectuer une étape In connaît l'ambient cible, qui est nécessairement un de ses sous-ambients. Le père d'un ambient voulant effectuer une action Out connaît aussi l'ambient cible : c'est son propre père. Enfin, un ambient est toujours ouvert dans son père lors de l'exécution d'une étape Open. Par conséquent, la décision d'exécuter une étape sera toujours prise par le père de l'ambient migrant ou étant dissous.

Chaque migration d'un ambient a pour entrer ou sortir d'un ambient b correspond à trois micro-étapes décrites ci-après et illustrées dans la figure 3.1. Les flèches représentent les pointeurs et les doubles flèches représentent les messages asynchrones envoyés d'un nœud à l'autre.

Nous détaillons une étape In de la forme $c[\, a[\, \texttt{in}\, b.Q\,]\ \mid\ b[\, \mathbf{0}\,]\,] \to c[\, b[\, a[\, Q\,]\,]\,]$:

étape 0 L'ambient a commence par déléguer la requête In b à son père courant, ici c. Pour se faire, il utilise le pointeur vers son père pour envoyer un message à c indiquant que a désire aller dans un ambient portant le nom b.

étape 1 L'ambient c compare la requête avec la présence de pointeurs vers les sous-ambients a et b, tous deux présents dans notre cas. La requête pouvant être satisfaite, elle est effacée en même temps que le pointeur vers a, et un message de repositionnement est envoyé à a. Ce message contient le pointeur vers b, pour que a puisse y migrer, ainsi qu'une valeur décrivant la requête de a venant d'être satisfaite.

étape 2 L'ambient a reçoit le message de repositionnement de c, migre vers b, puis positionne son pointeur ascendant vers son nouveau père b. Le processus gardé par la requête correspondant à la valeur reçue dans le message est activé. L'ambient a envoie également un message à b pour lui signaler sa présence en tant que sous-ambient. Lorsque b recevra ce message, il ajoutera un pointeur vers a aux pointeurs vers ses propres sous-ambients.

Comme il est possible qu'un ambient envoie plusieurs requêtes en parallèle, une étape 1 peut rendre caduque d'anciennes requêtes de a envoyées à son précédent père c. Ces requêtes non satisfaites doivent être envoyées de nouveau au nouveau père b. Dans ce but, le contrôleur de a conserve un journal des requêtes qui on été déléguées (lors d'une étape 0). Lors de la complétion

d'une action par une étape 2, toutes les autres requêtes enregistrées et par conséquent non satisfaites sont déléguées au nouveau père. Nous remarquons que ce journal ne peut pas être tenu par le père, puisque des requêtes peuvent lui parvenir bien après le départ du fils, à cause de la nature asynchrone de l'envoi de messages. Nous devons par contre prendre en compte la possibilité qu'un fils prodigue retourne vers un ancien père. Dans ce cas, de vieilles requêtes parvenant au père ne doivent pas être confondues avec les nouvelles requêtes que le fils peut déléguer. Ces vieilles requêtes doivent donc être détruites. Ceci n'est pas possible directement dans un monde asynchrone, mais le même résultat peut être obtenu en modifiant après chaque migration l'identificateur unique de l'ambient, en annotant chaque requête par l'identificateur unique courant, et en jetant les requêtes d'un fils dont l'identificateur est différent de celui contenu dans la requête.

Une action Out d'un ambient a qui sort d'un ambient b correspond à la même série de trois étapes. La différence principale est l'étape 1, où l'ambient b compare la requête avec la présence d'un pointeur vers a et son propre nom b, et envoie dans le message de repositionnement destiné à a un pointeur vers son propre père.

Nous remarquons que ces deux actions utilisent principalement la présence d'un pointeur d'un ambient vers son fils pour décider d'effectuer une action. Une étape 1 supprimant le pointeur vers l'ambient migrant, celui-ci ne peut plus participer à d'autres actions. Par contre, la cible d'un ambient effectuant une étape In est immédiatement disponible pour d'autres actions, puisque le pointeur de son père vers celle-ci ne disparaît pas. L'algorithme présenté ici essaie ainsi de minimiser le nombre d'ambients qui ne peuvent plus participer à d'autres actions ayant lieu en parallèle.

3.1.2 Une traduction simple

Afin de simplifier la traduction, nous ajoutons au join calcul décrit en section 2.3 quelques extensions utiles, qui peuvent facilement être encodées dans le join calcul classique. Nous autorisons les filtres non linéaires : les noms définis et les noms liés à des paramètres formels peuvent être présents plusieurs fois dans un filtre (ceci permet entre autres un léger filtrage sur le contenu des messages). Nous ajoutons aux définitions les construction `uid` i et `fresh` a qui lient les noms i et a, et interdisons l'envoi de message sur ces noms. Nous utilisons ces constructions pour créer des identificateurs uniques ou des nouveaux noms d'ambients (nous pourrions les remplacer par des règles inutiles comme $i() \triangleright \mathbf{0}$ ou $a() \triangleright \mathbf{0}$). Nous utilisons une notation de type enregistrement $e = \{l_1 = x_1; \ldots l_n = x_n\}$ comme un raccourci pour désigner un nuplet de noms $x_1, \ldots, x_n$ que nous transmettrions toujours dans le même ordre. Nous notons $e.l_i$ au lieu de x_i. Nous utilisons les notations algébriques In b κ, Out b κ pour désigner des entrées dans le journal contenant les étiquettes In, Out, ainsi que les noms b et κ. Nous utilisons un ensemble fini d'entrées dans un journal, dans le sens mathématique classique. Au lieu d'utiliser un encodage standard des itérateurs sur les ensembles pour implémenter le processus $Flush(l, in, out, \kappa)$, nous étendons la sémantique opérationnelle du join calcul avec une règle qui émet les messages présents dans le journal :

$$\textsc{Flush}\quad Flush(l, in, out, \kappa) \rightarrow \prod_{\textsc{In}\ d\ \kappa' \in l \mid \kappa \neq \kappa'} in(d, \kappa') \quad \mid \prod_{\textsc{Out}\ d\ \kappa' \in l \mid \kappa \neq \kappa'} out(d, \kappa')$$

Grâce aux extensions précédemment décrites, nous présentons la traduction compositionnelle $[\![\cdot]\!]_e$ en figure 3.2. Globalement, l'arbre formé par les ambients emboîtés est traduit en un arbre isomorphe de locations emboîtées. [1] Chaque ambient correspond à une location du join calcul contenant la définition D correspondant au contrôleur, ainsi que les processus représentant l'état de l'ambient. La définition D se décompose en trois groupes de règles D_0, D_1 et D_2 qui implémentent respectivement les étapes 0, 1 et 2 de l'algorithme.

Les pointeurs de l'algorithme de la section 3.1.1 n'étant utilisés que pour envoyer des messages ou pour désigner une cible de migration, nous n'implémentons pas ces pointeurs mais les moyens directs d'interagir avec les ambients. Pour se faire, nous associons à chaque contrôleur d'ambient son interface e qui est un enregistrement contenant les champs suivants : *here*, *amb*, sub_{in}, sub_{out}, *reloc*, *in* et *out*. Le champ *here* contient le nom de la location associée à l'ambient. Les autres champs contiennent les noms des canaux utilisés pour interagir avec cet ambient. Ces canaux sont définis par le contrôleur de l'ambient. La traduction est paramétrée par l'interface e de l'ambient englobant courant. Dans chaque ambient, la présence d'un sous-ambient de nom b se manifeste par un message $amb(j, b, e_b)$. Ce message contient l'interface e_b du sous-ambient, qui correspond donc au pointeur vers ce sous-ambient, ainsi que l'identificateur unique (uid) j de b. Le pointeur vers le père courant est représenté par l'interface e du père contenu dans le message d'état $s(a, i, e, l)$. Ce message comprend également certaines informations relatives à l'ambient courant : son nom

[1] L'environnement initial de traduction est décrit dans la section 3.2.

$$
\begin{aligned}
[\![a[\,P\,]]\!]_e &= \texttt{def}\ AM_{a,e}(P)\ \texttt{in}\ \mathbf{0} \\
[\![\texttt{in}\ a.P]\!]_e &= \texttt{def}\ \kappa() \triangleright [\![P]\!]_e\ \texttt{in}\ e.in(a,\kappa) \\
[\![\texttt{out}\ a.P]\!]_e &= \texttt{def}\ \kappa() \triangleright [\![P]\!]_e\ \texttt{in}\ e.out(a,\kappa) \\
[\![P \mid Q]\!]_e &= [\![P]\!]_e \mid [\![Q]\!]_e \\
[\![\mathbf{0}]\!]_e &= \mathbf{0}
\end{aligned}
$$

où le contrôleur $AM_{a,e}(P)$ est défini de la manière suivante :

$$
\begin{aligned}
D_0 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid in(b,\kappa) \triangleright s(a,i,e,l \cup \{\textsc{In}\ b\ \kappa\}) \mid e.sub_{in}(i,b,\kappa) \\
&\wedge\ s(a,i,e,l) \mid out(b,\kappa) \triangleright s(a,i,e,l \cup \{\textsc{Out}\ b\ \kappa\}) \mid e.sub_{out}(i,b,\kappa) \\
D_1 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid amb(j,b,e_b) \mid amb(k,c,e_c) \mid sub_{in}(k,b,\kappa) \triangleright \\
&\qquad s(a,i,e,l) \mid amb(j,b,e_b) \mid e_c.reloc(e_b,\kappa) \\
&\wedge\ s(a,i,e,l) \mid amb(j,b,e_b) \mid sub_{out}(j,a,\kappa) \triangleright s(a,i,e,l) \mid e_b.reloc(e,\kappa) \\
D_2 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid reloc(e',\kappa) \triangleright \texttt{go}\ e'.here; (I_{a,e_h,e'} \mid \kappa() \mid Flush(l,in,out,\kappa)) \\
D &\stackrel{\text{def}}{=} D_0, D_1, D_2 \\
\\
I_{a,e_h,e} &\stackrel{\text{def}}{=} \texttt{def uid}\ i\ \texttt{in}\ s(a,i,e,\emptyset) \mid e.amb(i,a,e_h) \\
AM_{a,e}(P) &\stackrel{\text{def}}{=} here\left[D\ :\ I_{a,e_h,e} \mid [\![P]\!]_{e_h}\right]
\end{aligned}
$$

avec (en utilisant une notation d'enregistrement) :

$$
e_h \stackrel{\text{def}}{=} \left\{ \begin{array}{l} here = here, amb = amb, sub_{in} = sub_{in}, \\ sub_{out} = sub_{out}, reloc = reloc, in = in, out = out \end{array} \right\}
$$

FIG. 3.2 – Traduction des processus ambients IN/OUT dans le join calcul

a, son uid i, ainsi que le journal l des requêtes IN et OUT qui ont été déléguées au père courant en utilisant e. Nous rappelons que le nom de chaque location étant unique dans le join calcul, à la différence des noms d'ambients, le nom d'une location représentant un ambient et le nom de l'ambient correspondant sont différents.

Reprenons l'exemple d'une action IN, en étudiant le rôle de chaque message dans la traduction de $c[\, a[\, \texttt{in}\ b.Q\ ]\ \mid\ b[\,\mathbf{0}\,]\,]$. La traduction commence par définir une continuation κ correspondant à Q, et envoie un message $in(b,\kappa)$ dans a. Ce message correspond à une requête (non encore déléguée) de migration dans un ambient appelé b.

L'étape 0 se résume à déléguer cette requête au père courant. La première règle de D_0 du contrôleur de a est activée, ce qui consomme les messages $s(a,i,e,l)$ et $in(b.\kappa)$, ajoute au journal l de a l'entrée IN b κ, et délègue la requête à son père c en envoyant un message $e.sub_{in}(i,b,\kappa)$. Ce message contient l'uid i de l'ambient a. L'ambient a reste actif, avec le nouveau message d'état $s(a,i,e,l \cup \{\textsc{In}\ b\ \kappa\})$ contenant le journal de requêtes déléguées mis à jour.

L'étape 1 est exécutée par le contrôleur de l'ambient c. La requête déléguée $sub_{in}(i,b,\kappa)$ ne peut être consommée que par la première règle de D_1. Cette règle requiert la présence du sous-ambient ayant émis la requête (identifié par son uid) et d'un sous-ambient portant le nom de la destination souhaitée (ici b). Cette étape supprime le pointeur vers le sous-ambient désirant migrer (le message $amb(i,a,e_a)$), empêchant ainsi toute autre action requérant ce message. Par contre, le message indiquant la présence de b est immédiatement réémis, ce qui signifie qu'il est disponible pour d'autres actions ayant lieu en parallèle. Un message de repositionnement $e_a.reloc(e_b,\kappa)$ est envoyé à a, lui signalant qu'il doit migrer vers b (représenté par l'interface de son contrôleur) et exécuter la continuation κ.

L'étape 2 est exécutée par le contrôleur de a, en utilisant la règle de D_2. Cette étape consomme le message *reloc* ainsi que le message d'état, effectue une migration du join calcul dans la location de l'ambient destination, et réactive l'ambient a avec comme nouveau père b. Pour ce faire, le processus I_{a,e_a,e_b} est activé. Ce processus génère un nouvel uid frais i', émet un message d'état $s(a,i',e_b,\emptyset)$ contenant le nouvel uid, l'interface du nouveau père, ainsi qu'un journal de requêtes déléguées à ce père vide, et signale au nouveau père sa présence en envoyant un message $e_b.amb(i',a,e_a)$, qui représentera le pointeur du père vers a lorsque ce message arrivera à destination. Nous remarquons que puisqu'aucun pointeur vers un sous-ambient ne contient l'ancien uid i, aucune ancienne requête déléguée contenant i ne pourra être consommée par une règle de D_1. Dans notre implémentation, ces vieilles requêtes sont en fait supprimées lorsqu'elle sont reçues. En parallèle avec la réactivation

$$
\begin{aligned}
[\![\texttt{open}\ a.P]\!]_e &= \texttt{def}\ \kappa() \triangleright [\![P]\!]_e\ \texttt{in}\ e.open(a,\kappa)\\
[\![\langle n\rangle]\!]_e &= e.send(n)\\
[\![(n).P]\!]_e &= \texttt{def}\ \kappa(n) \triangleright [\![P]\!]_e\ \texttt{in}\ e.recv(\kappa)\\
[\![!P]\!]_e &= \texttt{def}\ \kappa() \triangleright [\![P]\!]_e \mid \kappa()\ \texttt{in}\ \kappa()\\
[\![\nu a.P]\!]_e &= \texttt{def fresh}\ a\ \texttt{in}\ [\![P]\!]_e
\end{aligned}
$$

avec comme règles supplémentaires dans la définition de $AM_{a,e}(P)$:

$$
\begin{aligned}
D'_1 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid amb(j,b,e_b) \mid open(b,\kappa) \triangleright s(a,i,e,l) \mid e_b.opening(\kappa)\\
D'_2 &\stackrel{\text{def}}{=} s(a,i,e,l) \mid opening(\kappa) \triangleright f(e) \mid \kappa() \mid Flush(l, e.in, e.out, \kappa)\\
D_C &\stackrel{\text{def}}{=} s(a,i,e,l) \mid recv(\kappa) \mid send(n) \triangleright s(a,i,e,l) \mid \kappa(n)\\
D_F &\stackrel{\text{def}}{=} f(e) \mid in(b,\kappa) \triangleright f(e) \mid e.in(b,\kappa)\\
&\quad \wedge\ f(e) \mid out(b,\kappa) \triangleright f(e) \mid e.out(b,\kappa)\\
&\quad \wedge\ f(e) \mid open(b,\kappa) \triangleright f(e) \mid e.open(b,\kappa)\\
&\quad \wedge\ f(e) \mid amb(j,b,e_b) \triangleright f(e) \mid e.amb(j,b,e_b)\\
&\quad \wedge\ f(e) \mid sub_{in}(k,b,\kappa) \triangleright f(e) \mid e.sub_{in}(k,b,\kappa)\\
&\quad \wedge\ f(e) \mid sub_{out}(k,b,\kappa) \triangleright f(e) \mid e.sub_{out}(k,b,\kappa)\\
&\quad \wedge\ f(e) \mid recv(\kappa) \triangleright f(e) \mid e.recv(\kappa)\\
&\quad \wedge\ f(e) \mid send(b) \triangleright f(e) \mid e.send(b)\\
\\
D &\stackrel{\text{def}}{=} D_0, D_1, D'_1, D_2, D'_2, D_C, D_F
\end{aligned}
$$

avec (toujours en utilisant une notation d'enregistrement) :

$$
e_h \stackrel{\text{def}}{=} \left\{ \begin{array}{l} here = here, amb = amb, sub_{in} = sub_{in}, sub_{out} = sub_{out}, open = open,\\ reloc = reloc, in = in, out = out, opening = opening, recv = recv, send = send \end{array} \right\}
$$

FIG. 3.3 – Cas supplémentaires pour la traduction complète

de l'ambient, l'étape 2 émet le message κ pour exécuter la continuation de l'action, et exécute le processus $Flush(l \cup \{\text{IN } b\ \kappa\}, in, out, \kappa)$ qui émet à nouveau les requêtes présentes de le journal qui ont été préemptées. Ce processus, défini ci-dessus, émet une requête $in(d,\kappa')$ ou $out(d,\kappa')$ pour chaque entrée IN $d\ \kappa'$ ou OUT $d\ \kappa'$ apparaissant dans le journal, si la continuation κ' est différente de la continuation κ. Ces entrées correspondent aux requêtes déléguées qui n'ont pas été satisfaites avant la migration. Ces nouvelles requêtes seront déléguées au nouveau père en appliquant de nouveau l'étape 0.

Par un mécanisme similaire, une action OUT s'exécute en suivant l'algorithme, c'est à dire en utilisant la deuxième règle de D_0 de l'ambient migrant, la deuxième règle de D_1 du père de cet ambient, puis la règle de D_2 de l'ambiant migrant.

Nous remarquons que la structure arborescente des ambients est représentée de deux manières : par l'imbrication des locations et par les messages sur *amb*. Ces deux représentations ne sont pas tout le temps en phase. En effet, un message sur *amb* représente pour le père d'un ambient le fait que son fils est près à participer à une action. Lorsque cette action est entreprise (étape 1), ce message peut disparaître, alors que la location est toujours là. Le message ne sera régénéré qu'après la complétion de l'action (étape 2), qui a lieu après la migration de la location. Nous soulignons le fait que la structure arborescente des locations ne participe pas au calcul, le join calcul étant un calcul transparent. Ceci rend nécessaire la représentation de l'arbre des ambients par les messages *amb*.

3.1.3 Les autres constructions du calcul

La traduction présentée en figure 3.3 étend la précédente traduction pour prendre en compte le calcul des ambients complet. Pour chaque nouvelle construction, nous ajoutons un cas à l'opérateur de traduction compositionnel $[\![\ \cdot\ ]\!]_e$. Nous étendons également le contrôleur $AM_{a,e}(\ \cdot\ )$ et utilisons une interface e étendue, contenant les nouveaux champs *open*, *opening*, *recv*, et *send*.

Valeurs et Portées. Les noms des ambients ne sont pas modifiés lors de la traduction dans le join calcul. Les deux calculs utilisent la même notion de portée lexicale, l'extrusion d'une portée étant réalisée par équivalence structurelle (règle SCOPE dans le join calcul, règles R1

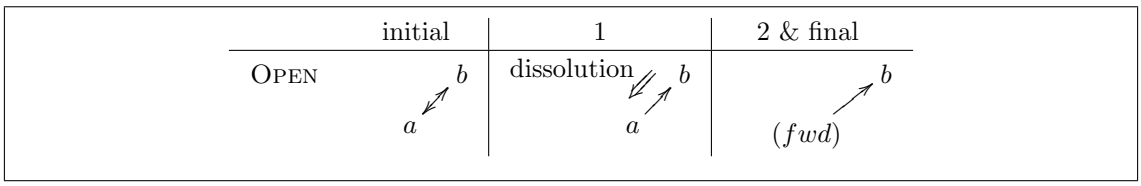


Fig. 3.4 – L'algorithme pour la capacité Open

et R2 dans le calcul des ambients). Ainsi, il suffit de traduire la création de noms locaux d'ambients $\nu a.P$ par des liaisons du join calcul `fresh` a avec la même portée.

Communication. La communication des ambients est implémentée en ajoutant une règle D_C à chaque contrôleur, qui définit deux nouveaux canaux *send* et *recv*. Ces canaux synchronisent l'émission et la réception de messages ambients. Ce codage est très similaire au codage des canaux du π calcul dans le join calcul (voir [FG96]).

Réplication. Chaque processus répliqué $!P$ est traduit en utilisant le codage récursif classique des boucles infinies dans le join calcul.

Dissolution. Des processus ambients peuvent dissoudre la frontière d'un ambient en utilisant la construction `open` , exposant ainsi leur contenu à l'ambient englobant. Dans le join calcul, les noms sont statiquement attachés à la location qui les définit et les frontières de locations ne peuvent jamais disparaître. Il est donc impossible de conserver l'isomorphisme de l'arbre des ambients dans l'arbre des locations, et nous devons donc distinguer deux états pour chaque location : soit l'ambient correspondant est actif et le message d'état s est présent, soit l'ambient correspondant a été ouvert et tout message envoyé à l'interface de cet ambient doit être propagé à l'ambient père. Cette indirection est réalisée en utilisant un message d'état sur le canal f, défini dans le contrôleur par D_F. Cette solution, illustrée en figure 3.4, rend les preuves de correction plus complexes puisqu'il est nécessaire de prouver que ces locations ouvertes n'interfèrent pas avec le reste de la traduction. Nous remarquons que, une fois de plus, la disponibilité d'un ambient pour participer à d'autres actions est représentée par la présence d'un pointeur de son père vers lui. Un ambient ouvert ne peut ainsi plus être la cible d'une étape In, par exemple.

3.2 Correction de la traduction

L'algorithme de synchronisation distribuée semble être assez éloigné de la sémantique opérationnelle des ambients. Cependant, à partir du moment où nous considérons qu'il est possible que deux ambients présents sur deux sites distincts interagissent, il nous semble nécessaire d'introduire des états intermédiaires comme nous l'avons fait dans la définition de l'algorithme puisque de telles synchronisations distribuées ne sont pas supposées comme primitives dans le calcul cible. De plus, la traduction d'une configuration d'ambients emboîtés produit un terme du join calcul pouvant être particulièrement gros et contenant de nombreuses instances de l'algorithme s'exécutant en parallèle. Ces deux aspects, complexité de l'algorithme et taille des termes manipulés, participent à la complexité de la preuve de correction de l'implémentation distribuée. D'un point de vue technique, les deux calculs ont une sémantique à réduction, auxquelles on peut associer des notions classiques d'observables. Cette propriété nous permet d'exprimer directement la correction de la traduction dans un cadre précis, sans passer par une abstraction de l'algorithme. Bien entendu, de légères différences existent entre la traduction et l'implémentation en JoCaml, elles sont abordées dans la section 3.5.

Nous commençons donc par définir une notion syntaxique d'observable. Pour chacun des calculs, nous utilisons une famille de prédicats d'observation immédiate sur les processus P indexés par les noms b, notés $P \downarrow_b$.

- Dans le calcul des ambients, nous observons les ambients à la racine dont le nom est libre : nous avons $P \downarrow_b$ si $P \equiv \nu\widetilde{v}.(\, b[\, Q\,] \mid R)$ avec $b \notin \widetilde{v}$.
- Dans le join calcul, nous observons les messages envoyés sur des canaux dont le nom est libre et non défini dans une location active : nous avons $P \downarrow_b$ si $P \equiv \alpha[D' : b(\widetilde{v}) \mid P'] \parallel \mathcal{S}$ avec b défini ni en D', ni en $\mathcal{S}$.

Ensuite, nous exprimons la correction de la traduction en fonction des prédicats suivants, à la fois pour les processus des ambients et du join calcul :

- Un processus P exhibe une *barbe faible* sur b (noté $P \Downarrow_b$) si $P \rightarrow^* P' \downarrow_b$.

– Un processus P *diverge* (noté $P \Uparrow$) si P peut se réduire un nombre infini de fois.
– Un processus P exhibe une *barbe fair-must* sur b (notée $\Box P \Downarrow_b$) si pour tous les P' tels que $P \to^* P'$, nous avons $P' \Downarrow_b$.

Mis ensemble, ces prédicats donnent un sens à la notion informelle de correction : "la traduction ne devrait ni supprimer des comportements existants, ni introduire de nouveaux comportements." La notion minimale de correction pour une implémentation est la préservation des barbes faibles. Dans un sens (du processus traduit vers le processus ambient initial), cette préservation garantie qu'aucun nouveau comportement n'est introduit. Dans l'autre sens, cette préservation garantie qu'aucun comportement n'est supprimé, et élimine par exemple la traduction vide. D'un point de vue pragmatique, il est intéressant de conserver la propriété de convergence. En particulier, ceci indique que la traduction n'introduit pas de divergence n'existant pas dans le processus initial. La correction par rapport à l'observation des barbes fair-must permet de comparer des calculs infinis [Fou98] et élimine les implémentations qui ont des deadlocks ou un ordonnancement restrictif des calculs. En effet, la traduction d'un processus n'exhibe une barbe fair-must uniquement si cette barbe est également présente dans le processus initial.

Comme les ambients présents à la racine ne sont pas traduits en des messages sur des noms libres, l'observation d'ambients traduits demande un soin particulier. A cette fin, nous étendons la traduction ayant lieu à la racine en ajoutant une définition D_t qui exhibe la présence d'une certaine barbe $\downarrow_b$ dans le processus source. De plus, il est nécessaire d'empêcher que l'ambient racine délègue des messages envoyés sur *in* ou *out*, cet ambient n'ayant personne à qui déléguer ces messages. Pour ce faire, nous modifions la définition D_0 pour que ses règles requièrent la présence d'un message *lock* qui ne sera jamais présent. Nous définissons la traduction à la racine permettant d'observer le nom b notée $[\![\cdot]\!]^b$ (en utilisant les mêmes définitions D_1, D'_1, D_2, D'_2, D_C, D_F, et la même interface e_h que celles introduites dans la figure 3.3, pour une interface donnée e) :

$$
\begin{array}{rcl}
[\![P]\!]^b & \stackrel{\text{def}}{=} & \mathit{here}[D_l, D_t, \mathtt{uid}\ i : s(a,i,e,\emptyset) \mid t(b) \mid [\![P]\!]_{e_h}] \\
D'_0 & \stackrel{\text{def}}{=} & s(a,i,e,l) \mid \mathit{in}(b,\kappa) \mid \mathit{lock}() \ \triangleright s(a,i,e,l \cup \{\textsc{In}\ b\ \kappa\}) \mid e.\mathit{sub}_{in}(i,b,\kappa) \\
 & \wedge & s(a,i,e,l) \mid \mathit{out}(b,\kappa) \mid \mathit{lock}() \triangleright s(a,i,e,l \cup \{\textsc{Out}\ b\ \kappa\}) \mid e.\mathit{sub}_{out}(i,b,\kappa) \\
D_l & \stackrel{\text{def}}{=} & D'_0, D_1, D'_1, D_2, D'_2, D_C, D_F \\
D_t & \stackrel{\text{def}}{=} & s(a,i,e,l) \mid \mathit{amb}(j,b,e_b) \mid t(b) \triangleright s(a,i,e,l) \mid \mathit{amb}(j,b,e_b) \mid \mathit{yes}()
\end{array}
$$

Sans perte de généralité, nous supposons par la suite que les noms de a, i, t, e, *yes* et *lock* sont tous différents des noms libres de P. Nous supposons aussi que tous les noms introduits par la traduction sont différents des noms libres de P.

Nous pouvons maintenant exprimer le théorème de correction de la traduction des processus ambients en processus du join calcul, qui correspond au fait que notre traduction préserve et reflète toutes les observables décrites ci-avant :

Théorème 1 *Pour tout processus ambient P et nom b, nous avons :*

1. *$P \Downarrow_b$ si et seulement si $[\![P]\!]^b \Downarrow_{yes}$;*
2. *$P \Uparrow$ si et seulement si $[\![P]\!]^b \Uparrow$;*
3. *$\Box P \Downarrow_b$ si et seulement si $\Box [\![P]\!]^b \Downarrow_{yes}$.*

Bien que la correction s'exprime simplement en terme d'observables sur les réductions des processus, sa preuve est complexe. En particulier, une approche directe conduit rapidement à des inductions à la fois sur la syntaxe du processus source et sur des séries de réductions difficiles à gérer. Les deux prochaines sections présentent notre stratégie de preuve. Une preuve du théorème 1 est présentée à la fin de la section A.2.

3.3 Un calcul des ambients étendu par des états transitoires

Afin de prouver le théorème 1, nous introduisons maintenant un calcul des ambients étendus possédant des constructions reflétant les états transitoires les plus importants de l'algorithme de la section 3.1.1. Ce calcul possède une sémantique à réduction en correspondance directe avec l'algorithme. Par exemple, dans ce calcul, les actions In sont formées de deux étapes : une étape 1 et une étape 2 (les étapes 0 ne sont pas représentées parce que toutes les requêtes finissent par être déléguées au père courant). Dans la prochaine section, nous nous basons sur le calcul étendu pour établir la correction en tant que composition de deux équivalences. Tout d'abord nous établissons une paire de *simulations couplées* [PS92] qui mettent en relation les deux sémantiques du calcul

des ambients (le calcul source et le calcul étendu), puis nous utilisons une bisimulation pour mettre en relation les ambients étendus et leur traduction dans le join calcul.

La grammaire du calcul étendu est présentée dans la figure 3.5. Ce calcul possède de nouveaux processus représentant des ambients qui doivent migrer ou être dissous, suite à une étape 1 de leur père. Nous appelons ces ambients transitoires *souches*. Ce calcul introduit aussi des nouveaux processus donnant la position future des ambients en cours de migration, nous appelons ces processus *scions*. Les paires de souches et de scions sont syntaxiquement associés par une marque i. La sémantique opérationnelle du calcul étendu est décrite dans la figure 3.6. Cette sémantique est une sémantique à réduction, avec des étiquettes additionnelles pour les souches et les scions. Chacune des étapes de réduction In, Out et Open se décompose en deux étapes, décrites ci-après. Les étapes initiales $\rightarrow_1$ créent des souches et des scions ; les étapes finales $\rightarrow_2$ les consomment, comme illustré en figure 3.7.

Les règles de réduction pour les étapes Recv et Repl sont celles de la sémantique originelle, avec la seule exception que nous notons ces étapes $\rightarrow_C$ au lieu de $\rightarrow$. Nous obtenons donc un système de réduction pour les processus ambients étendus contenant les étapes $\rightarrow_{12C} \stackrel{\text{def}}{=} \rightarrow_1 \cup \rightarrow_2 \cup \rightarrow_C$. Afin de pouvoir comparer le calcul originel et le calcul étendu, nous étendons aussi la sémantique originelle et les prédicats d'observation pour les adapter aux ambients étendus. La définition des observables n'est pas modifié, dans le sens où le prédicat $\downarrow_b$ ne s'applique pas aux ambients étendus $Xb[\,Q\,]$.

Une souche et un scion sont voisins lorsqu'ils sont créés, mais ils peuvent s'éloigner l'un de l'autre lors de l'exécution d'autres étapes ayant lieu avant leur étape finale. C'est pourquoi un système de transitions étiquetées auxiliaire est utilisé pour apparier souches et scions. La finalisation d'une migration correspond donc à une étape de communication non locale entre deux processus pouvant être dans des contextes distincts mais syntaxiquement liés par la marque i. Ceci pourrait sembler difficile à implémenter, mais en fait les scions n'ont aucun contenu opérationnel, ils représentent juste la destination passive d'une migration forte (en d'autres termes, la pointe de la flèche d'un pointeur). Nous remarquons que par définition des réductions étiquetées, toutes les extrusions de portée doivent être accomplies avant que l'étape Move 2 ne puisse avoir lieu.

Pour illustrer le fonctionnement de la sémantique étendue, nous comparons dans la figure 3.8 les réductions d'un processus P contenant une paire critique. Dans ce processus, soit a est ouvert en premier, ce qui empêche b d'exécuter son action `out` a (c'est le cas $P \rightarrow P_1$), soit b sort de a, puis a est ouvert, laissant b seul et vide (cas $P \rightarrow\!\!\rightarrow P_3$). Le premier diagramme montre les étapes pour la sémantique originelle ; le second diagramme représente les étapes pour la sémantique étendue. Les processus P, P_1, P_2 et P_3 sont des processus originels ; les processus Q, Q_1 et Q_2 sont des processus transitoires, correspondant aux états intermédiaires de l'algorithme. Nous remarquons que la création de tels états intermédiaires est inévitable dans un cadre distribué, puisque les actions initiales Open et Out sont présentes dans des ambients distincts qui peuvent par conséquent être exécutés sur des sites distincts.

P	::=		processus d'ambient étendu
		$\ldots$	*toutes les constructions de la figure 2.1*
		$\mid \quad Xn[\,P\,]$	souche
		$\mid \quad i$	scion
		$\mid \quad \nu i.P$	restriction de marque
X	::=		extension d'état
		$\overline{\imath}\{P\}$–	la souche doit aller dans i
		$\mid \quad \mathtt{o}\{P\}$–	la souche est en cours de dissolution

Conditions de bonne formation : les souches et les scions ne peuvent être présents que dans des contextes d'évaluation étendus ; les marques restreintes i doivent être utilisées de manière linéaire (exactement une souche et un scion). Par la suite, nous notons $X^{=}n[P]$ pour n'importe quel ambient étendu : soit $Xn[P]$, soit $n[P]$.

Fig. 3.5 – Syntaxe du calcul d'ambients étendu avec des états transitoires

Les *contextes d'évaluation étendus* $E(\cdot)$ sont définis par la grammaire :

$$E(\cdot) ::= \quad \cdot \quad \mid \quad P \mid E(\cdot) \quad \mid \quad E(\cdot) \mid P \quad \mid \quad X^{=} n[\, E(\cdot)\,] \quad \mid \quad \nu n.E(\cdot) \quad \mid \quad \nu i.E(\cdot)$$

L'*équivalence structurelle* $\equiv$ est la plus petite relation d'équivalence sur les processus qui est stable par application des contextes d'évaluation étendus et par α-conversion, qui satisfait les axiomes P0, P1, P2, C0, C1, R1, R2 de la figure 2.2 et qui satisfait l'axiome :

$$\text{R2X} \ \frac{m \neq n \quad m \text{ n'est pas libre dans } X}{Xn[\nu m.P] \equiv \nu m.Xn[P]}$$

Les *transitions étiquetées* $\xrightarrow{\alpha}$ sont les plus petites familles de relations stables par application de contextes d'évaluation étendus (en omettant la restriction de nom et de marque), et telles que :

$$\textsc{Stub} \quad \bar{\imath}\{Q\}\text{–}n[R] \xrightarrow{\bar{\imath}.n[Q|R]} \mathbf{0} \qquad \textsc{Scion} \quad i \xrightarrow{i.P} P$$

Les *étapes de réduction originelles* $\rightarrow$ sont définies dans la figure 2.2, en utilisant les contextes d'évaluation étendus. Les *étapes initiales* $\rightarrow_1$*, étapes finales* $\rightarrow_2$*, et étapes diverses* $\rightarrow_C$ sont les plus petites relations stables par équivalence structurelle, par application de contextes d'évaluation étendus, et telles que :

$$\begin{array}{rlcrl} \textsc{In 1} & m[P] \mid n[\mathtt{in}\ m.Q \mid R] & & \textsc{Out 1} & X^{=} m\big[P \mid n[\mathtt{out}\ m.Q \mid R]\big] \\ \rightarrow_1 & \nu i.\ m[i \mid P] \mid \bar{\imath}\{Q\}\text{–}n[R] & & \rightarrow_1 & \nu i.\ i \mid X^{=} m\big[P \mid \bar{\imath}\{Q\}\text{–}n[R]\big] \end{array}$$

$$\textsc{Move 2} \ \frac{P \xrightarrow{\bar{\imath}.S} P' \quad Q \xrightarrow{i.S} Q'}{\nu i.(P \mid Q) \rightarrow_2 P' \mid Q'}$$

$$\textsc{Open 1} \quad \mathtt{open}\ n.Q \mid n[R] \rightarrow_1 \mathtt{o}\{Q\}\text{–}n[R] \qquad \textsc{Open 2} \quad \mathtt{o}\{Q\}\text{–}n[R] \rightarrow_2 Q \mid R$$

$$\textsc{Recv} \quad \langle n \rangle \mid (x).P \rightarrow_C P\{^{n}/_{x}\} \qquad \textsc{Repl} \quad !P \rightarrow_C P \mid !P$$

FIG. 3.6 – Sémantique du calcul des ambients étendu avec des états transitoires

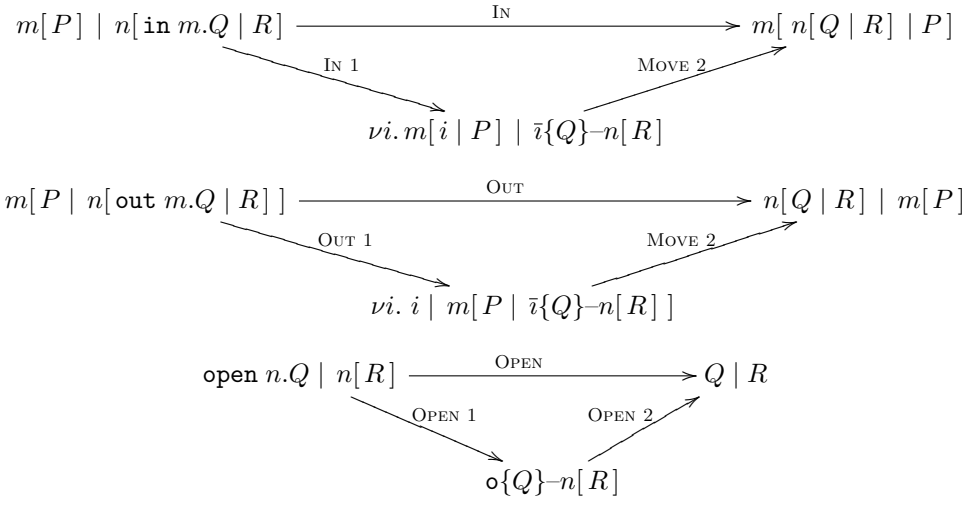


FIG. 3.7 – Traduction des capacités dans le calcul étendu

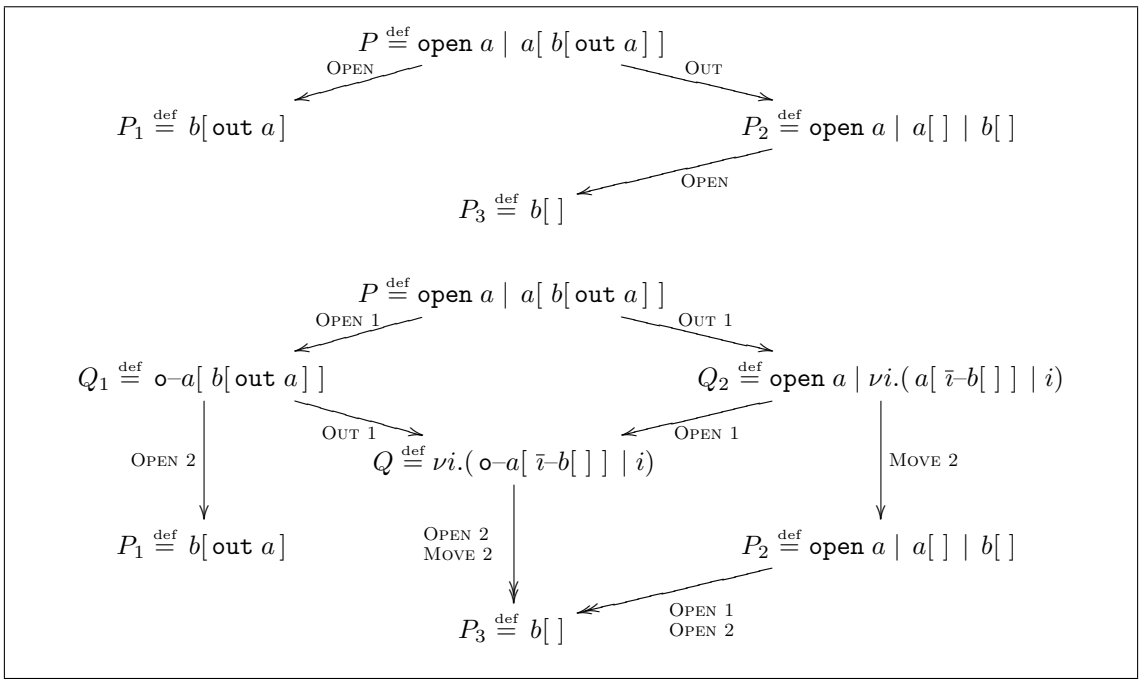


FIG. 3.8 – Réductions comparées pour un processus contenant une paire critique.

3.4 Simulations couplées et correspondance opérationnelle

Nous poursuivons notre étude de la correction de la traduction en terme d'équivalences basées sur les barbes faibles. Ces équivalences sont essentielles pour obtenir une preuve modulaire. De plus, elles nous donnent une version plus fine de la correction. (Se référer à [Fou98] pour une discussion des équivalences et encodages dans les calculs de processus.) A la place d'équivalences, nous utilisons en fait souvent des relations qui possèdent des domaines différents ($\mathcal{P}_a$ et $\mathcal{P}_b$). Ces domaines sont équipés de systèmes de réduction ($\rightarrow_a$ et $\rightarrow_b$) et de familles d'observables ($\Downarrow_{a,x}$ et $\Downarrow_{b,x}$).

Définition 3.4.1 (Bisimulations barbues) *Une relation $\mathcal{R} \in \mathcal{P}_a \times \mathcal{P}_b$ est une simulation barbue (faible) lorsque, pour tout $P\ \mathcal{R}\ Q$, nous avons :*

1. *si $P \rightarrow_a^* P'$, alors il existe Q' tel que $Q \rightarrow_b^* Q'$ et $P'\ \mathcal{R}\ Q'$;*
2. *pour tous x, si $P \Downarrow_{a,x}$, alors nous avons $Q \Downarrow_{b,x}$.*

$\mathcal{R}$ est une bisimulation barbue quand $\mathcal{R}$ et la relation inverse $\mathcal{R}^{-1}$ sont des simulations barbues.

Le grand avantage des techniques de preuve basées sur les bisimulations consiste à ne considérer que quelques étapes de réduction à la fois, au lieu de s'intéresser à des traces complètes d'exécution. Malheureusement, la bisimilarité barbue $\approx$ (c'est à dire la plus grande bisimulation barbue stable par application de contextes d'évaluation) est trop précise pour notre protocole. En effet, les processus transitoires comme Q_1 dans l'exemple précédent représentent un choix interne partiel : Q_1 peut se réduire en P_1 ou P_3 mais pas en P_2. Puisque P_1, P_2 et P_3 sont distincts selon $\approx$ ($P_2 \not\approx P_3$ se prouve facilement en considérant le contexte $(\cdot) \mid a[\, c[\,]\,]$), le processus Q_1 n'est bisimilaire à *aucun* processus P ou P_i de l'exemple. Nous remarquons par contre que nous avons $Q_2 \approx P_2$ et $Q \approx P_3$, mais pour d'autres processus exhibant des réductions à l'intérieur de souches, cela ne serait plus nécessairement le cas.

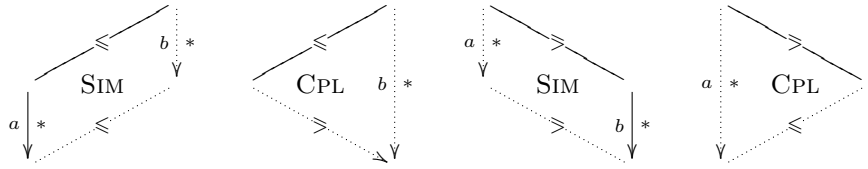
Pour résoudre ce problème de choix partiel, Parrow and Sjödin ont proposé des relations moins précises appelées *simulations couplées* [PS92, NP96]. Nous adaptons leur définition à notre cas :

Définition 3.4.2 (Simulations couplées) *Les relations $\leqslant \in \mathcal{P}_a \times \mathcal{P}_b$ et $\leqslant \in \mathcal{P}_b \times \mathcal{P}_a$ forment une paire de* simulations couplées barbues *si $\leqslant$ et $\leqslant$ sont des simulations barbues qui satisfont les conditions de couplage :*

1. *si $P \leqslant Q$, alors il existe Q' tel que $Q \rightarrow_b^* Q'$ et $Q' \leqslant P$;*
2. *si $Q \leqslant P$, alors il existe P' tel que $P \rightarrow_a^* P'$ et $P' \leqslant Q$.*

Dans la définition originale des simulations couplées de [PS92], les conditions de couplages ne sont requises que pour les paires de processus stables. Comme dans [Fou98], notre définition est en fait la contrepartie barbue des *simulations faiblement couplées* présentées dans [PS94].

Dans les preuves du chapitre A, nous utilisons des diagrammes pour représenter les propriétés de processus dans une relation. Nous adoptons l'iconographie habituelle : les arêtes des diagrammes représentent les relations entre les processus, les arêtes solides représentent les relations universellement quantifiées (les hypothèses), les arêtes en pointillés représentent les relations existentiellement quantifiées (les conclusions). Par exemple, la précédente définition peut être graphiquement représentée par les quatre diagrammes :



Les différences entre $\leqslant$ et $\geqslant$ sont très utiles pour s'accommoder de processus transitoires tel Q_1. Les conditions de couplage garantissent que tout état transitoire peut à la fois être associé à un état moins avancé et à un état plus avancé. Dans notre exemple, nous avons $Q_1 \leqslant P$ et $P_i \leqslant Q_1$ pour $i = 1, 3$.

3.4.1 Correction de l'algorithme asynchrone

La première partie de notre preuve de correction se traduit en une paire de simulations couplées entre les processus du calcul des ambients utilisant la sémantique originelle et les processus étendus utilisant la sémantique étendue. Dans le théorème ci-après, les processus en relation ont la même syntaxe mais sont considérés dans deux calculs différents équipés de sémantiques de réduction différentes.

Théorème 2 *Soit $\lesseqgtr$ l'union de $\leqslant \cap \geqslant$ pour toutes les simulations couplées barbues entre ambients et ambients étendus qui soient stables par l'application de contextes d'évaluation étendus. Pour tous les processus ambient P, nous avons $P \lesseqgtr P$.*

La preuve est détaillé en section A.1. Cette preuve fait apparaître des points subtils de notre algorithme qui sont dus à son parallélisme. Après quelques résultats sur des propriétés de commutation partielle pour les étapes de réduction étendues, les lemmes clés établissent la propriété que, pour n'importe quel processus ambient P et processus ambient étendu Q, si $P \rightarrow^*_{12C} Q$ alors :

1. il existe un processus ambient P' tel que $Q \rightarrow^*_{12C} P'$;
2. quelque soit P' satisfaisant (1), nous avons $P \rightarrow^* P'$ dans la sémantique originelle.

La preuve de la propriété (2) est schématiquement comme suit : par induction sur le nombre d'étapes étendues, nous utilisons les commutations partielles des étapes 1 et 2 pour amener en tête de la série de réductions une paire d'étapes appariées $\rightarrow_1\rightarrow_2$, qui peuvent donc être remplacées par une étape $\rightarrow$. Nous utilisons une procédure itérative pour choisir une bonne paire d'étapes. Quand la série de réductions commence par une étape 1, nous partitionnons les étapes suivantes en étapes externes (ce sont les étapes qui concernent les ambients extérieurs à la souche créée par la première étape), les étapes pénétrantes (les étapes MOVE 2 qui entrent dans la souche), et les étapes internes. Si des étapes internes 1 sont présentes, nous montrons que la première d'entre elles commute avec n'importe quelle étape précédente et peut donc être amenée au début de la série de réductions. On répète alors la procédure avec cette nouvelle première étape. Après un nombre fini de répétitions, nous obtenons une série de réductions où la première étape (qui est une étape 1) n'est suivie que d'étapes externes, puis de l'étape 2 appariée. Nous concluons en montrant que cette étape 2 peut commuter avec les autres étapes jusqu'à parvenir en seconde position, et remplaçons les deux étapes par une étape originelle.

3.4.2 Correspondance opérationnelle

La deuxième partie de la preuve met en relation les ambients utilisant la sémantique étendue avec leur traduction dans le join calcul. Elle est plus simple en principe que la première partie, mais est rendue plus complexe à cause des nombreux détails explicites de l'implémentation qui n'étaient pas essentiels pour l'algorithme.

Afin d'exprimer la correspondance des observations malgré la traduction, nous augmentons la traduction à la racine $[\![\cdot]\!]^b$ du théorème 1 pour permettre le choix externe de la barbe qui est testée. En utilisant les mêmes notations, nous notons $[\![\cdot]\!]^t$ la traduction qui associe à chaque processus P le processus $h^0[D_l^0, D_t^0, \texttt{uid}\ i^0 : s(a, i^0, e, \emptyset) \mid p(t) \mid [\![P]\!]_{e_0}]$, en renommant e_h en e_0 dans D_l et D_t. Comme auparavant, nous supposons que les noms de h^0, a, i^0, e, p, t, *yes* et *lock* sont tous distincts des noms libres de P.

À n'importe quel moment lors de l'exécution, nous pouvons utiliser le contexte d'évaluation $T_b(\cdot) \stackrel{\text{def}}{=} [p(t) \triangleright t(b) : \mathbf{0}] \parallel (\cdot)$ pour tester la présence de la traduction d'une barbe ambient b en observant la présence d'une barbe du join calcul $T_b(\cdot) \Downarrow_{yes}$, en nous restreignant aux noms b non définis dans le processus join. Nous avons la propriété suivante : $[\![P]\!]^b \approx T_b([\![P]\!]^t)$ dans le join calcul (nous pouvons toujours appliquer le contexte $T_b(\,\cdot\,)$ dans ce cas puisque les seuls noms définis dans $[\![P]\!]^t$ sont des noms de la traduction qui sont distincts des noms d'ambients, donc de b).

Théorème 3 (Correction de la traduction) *Soit $\approx^{aj}$ la plus grande bisimulation entre les processus ambients étendus munis des réductions $\rightarrow_{12C}$ et les processus du join calcul telle que si $Q \approx^{aj} R$, nous avons $Q \Downarrow_b$ si et seulement si $b \notin dn(R)$ et $T_b(R) \Downarrow_{yes}$. Pour tous les processus du calcul des ambients P nous avons $P \approx^{aj} [\![P]\!]^t$.*

Le théorème 3 est un corollaire du théorème 8 de la section A.2. Ce dernier théorème est une *bisimulation forte up to bookkeeping* entre les ambients étendus atteignables par réduction à partir d'un ambient et leur traduction. Puisque les états transitoires créés par la traduction ont été incorporés dans le calcul étendu, cette preuve consiste principalement à montrer la correspondance opérationnelle entre les deux calculs. Pour ce faire, nous partitionnons les réductions du join calcul en fonction de la règle join utilisée. Par exemple, les réductions du join calcul qui utilisent les règles des définitions D_1, D_1' de la figure 3.3 sont dénotées $\rightarrow_1$. Ces réductions créent soit un message sur le canal *reloc*, soit un message sur le canal *opening*, et correspondent aux règles $\rightarrow_1$ du calcul des ambients étendus. Nous obtenons ainsi deux grandes classes de réductions du join calcul : les étapes de réduction qui correspondent à une réduction $\rightarrow_{12C}$ du calcul des ambients étendus, et les étapes que nous appelons de "bookkeeping", notées $\rightarrow_B$, qui sont les étapes non essentielles de la traduction utilisées pour démarrer l'exécution des continuations, faire migrer les locations, gérer les journaux d'étapes déléguées, et activer les nouveaux contrôleurs d'ambients.

Les lemmes principaux décrivent les simplifications que l'on applique aux dérivations de processus issus de la traductions. Ces simplifications sont nécessaires pour obtenir des processus qui soient eux aussi des traductions d'un ambient étendu. Ces lemmes correspondent à des diagrammes de commutation élémentaires entre les relations de simplifications et les étapes de réductions. Par exemple, un de ces lemmes exprime le fait que les "vieilles requêtes" peuvent être supprimées ; un autre, plus complexe, montre que les locations représentant des ambients qui ont été dissous peuvent être éliminées en insérant le contenu de ces ambients dissous dans leur ancien père. Afin de conclure, nous exhibons une relation de bisimulation entre les ambients étendus utilisant les réductions $\rightarrow_{12C}$ et les traductions à la racine de ces ambients utilisant les réductions $\rightarrow_B^* \rightarrow_{12C} \rightarrow_B^*$. Cette preuve utilise la technique des diagrammes décroissants de [Oos94], qui donne la garantie lorsque certaines hypothèses sont satisfaites pour chaque diagramme que les diagrammes peuvent être assemblés de manière finie pour donner le résultat désiré.

3.5 Implémentation distribuée

Cette section décrit brièvement l'implémentation réalisée en JoCaml et aborde le problème du lancement de l'exécution distribuée de programmes ambients. Le lecteur peut se référer à [FS99] pour avoir le code source, le manuel d'installation, ainsi que des exemples de programmes ambients.

3.5.1 L'implémentation en JoCaml

Notre implémentation suit de très près la traduction donnée dans les figures 3.2 et 3.3. Le système JoCaml intégrant déjà des primitives pour la mobilité, la synchronisation locale et la distribution pendant exécution, le code est très compact : moins de 400 lignes de code source, et moins de 40ko de bytecode pour les fichiers objets (fichiers `.cmo`). Les principales différences entre la traduction et notre code sont les suivantes :

– Comme dans le calcul des ambients [CG98], les messages peuvent communiquer des noms d'ambients, mais aussi des suites d'actions arbitraires, comme par exemple dans le processus ambient $\langle \texttt{in}\ a.\texttt{out}\ b \rangle \mid\ !(x).x.\langle x \rangle$. Nous communiquons ainsi des valeurs représentant la syntaxe abstraite de ces suites d'actions, qui sont ensuite interprétées.

- Notre implémentation n'est pas une traduction globale (*i.e.* une compilation), mais un interpréteur. Par conséquent, les processus gardés sont traduits au vol en maintenant un environnement conservant la valeur des variables locales. L'interpréteur effectue aussi une forme de typage dynamique pour s'assurer de l'utilisation correcte d'une valeur en tant que nom ou action.
- La traduction utilise des filtres non linéaires, qui n'existent pas dans JoCaml pour des raisons d'efficacité. Pour pallier cette absence, l'implémentation utilise un cache. Lorsqu'un message arrive sur un canal sub_{in}, sub_{out}, ou *open*, l'interprète examine les messages *amb* qui sont dans le cache pour exécuter immédiatement la requête. Si les messages nécessaires ne sont pas présents, le message est mis dans le cache. De façon identique, lorsqu'un message arrive sur le canal *amb*, le cache est parcouru pour identifier la possibilité d'exécuter certaines requêtes, et le message est ajouté au cache si aucune requête ne peut être satisfaite.
- Comme dans le calcul source, la réplication provoque une exécution qui diverge. Pour limiter la consommation de ressources, l'interprète réplique des processus à la demande : puisque chaque réduction ambient utilise au plus deux copies d'un processus répliqué, il suffit de commencer par créer deux copies du processus répliqué, puis de générer une nouvelle copie lorsqu'une copie existante est utilisée ou modifiée. Par conséquent, le processus $!\,a[\,]$ ne diverge pas ; par contre le processus $!\,a[\,\texttt{in}\,a\,]$ diverge encore.
- Les règles de réaction du contrôleur des figure 3.2 et 3.3 requièrent toutes la présence d'un message sur s (ou f dans le cas d'un ambient ouvert). Ce message représente un verrou local, et il est intéressant de voir que ce verrou est toujours immédiatement réémis dans le processus gardé des règles de réaction excepté dans un cas : la complétion d'une action. Dans ce cas, le verrou n'est réémis que lorsque la migration est terminée. De plus, comme nous l'avons décrit auparavant, la présence d'un message *amb* dans le père d'un ambient représente la disponibilité du fils pour une action. Le contrôleur est conçu de telle sorte que ces messages soient le plus souvent réémis immédiatement lorsqu'ils sont utilisés. Les seules exceptions sont les cas correspondant aux ambients effectuant une action : l'ambient migrant pour les actions In et Out (le message sur *amb* ne sera réémis que lorsque l'ambient parviendra à destination), et un ambient étant dissous pour une action Open (le message ne sera jamais réémis, l'ambient n'existant plus). Ainsi, les cibles d'une migration (pour les cas In et Out) et les sous-ambients de l'ambient migrant ne sont pas bloqués pendant la réduction. Ces contrôleurs offrent ainsi un fort niveau de parallélisme et n'utilisent que des formes locales de synchronisation, permettant une implémentation distribuée.

3.5.2 Contrôle de la distribution à l'exécution

Bien que les ambients et les locations possèdent une structure hiérarchique similaire, les interprétations de cette structure en tant que distribution physique sont très différentes. Nous décrivons tout d'abord ces deux interprétations, puis nous indiquons les choix que nous avons faits pour l'implémentation.

Le join calcul présente un modèle "horizontal" de distribution à l'exécution : à cause de la transparence du calcul, chaque paire de location peut interagir comme si elles étaient exécutées en parallèle, indépendemment de leur localisation physique. De plus, on peut considérer que chaque site exécutant un interpréteur JoCaml est en fait un certaine location accueillant un arbre de sous-locations. Enfin, on peut garantir que après une migration, la location ayant migré et son nouveau père sont sur le même site, et ce jusqu'à la prochaine migration. Par conséquent, un programme JoCaml distribué se compose de sites sur lesquels résident des locations, et la distribution est donc "horizontale" puisque basée sur un modèle plat de sites.

De son côté, le calcul des ambients ne prend pas explicitement en compte la notion de distribution. Cependant, les ambients représentent intuitivement des pare-feux, des domaines administratifs, voire des routeurs. Ces ambients s'exécutent donc sur différents sites, et leur contenu s'exécute aussi potentiellement sur différents sites. De plus, le modèle de synchronisation des ambients se base sur l'arbre des ambients et des interactions verticales, c'est à dire entre un ambient et son père (cas Out et Open) ou entre deux frères (cas In). Par conséquent, le modèle de distribution est plutôt "vertical", puisque un ambient et son père ne s'exécutent pas nécessairement sur le même site mais peuvent interagir.

Afin de décrire la distribution associée à une exécution d'ambients, on peut naturellement annoter chaque processus avec une étiquette de "site". Cependant, lors de l'évolution dynamique de l'exécution et des diverses migrations, plusieurs sites peuvent logiquement être associés à un ambient impliqué dans une étape de réduction. Le choix du site est donc un choix de conception. Par exemple, considérons le processus $a[\,P\,] \mid b[\,\texttt{in}\,a.Q \mid R\,]$, et supposons que les ambients a et

b s'exécutent sur deux sites distincts appelés s_a et s_b. Après l'étape In, b peut soit continuer à s'exécuter sur le site s_b, soit s'exécuter sur le site s_a. Si on décide que b doit s'exécuter sur s_a, la même question se pose pour chacun de ses sous-ambients. Considérons maintenant le processus $c[\, a[\, P \mid b[\, \texttt{out}\ a.Q \mid R\,]\,]\,]$, où a, b et c s'exécutent respectivement sur s_a, s_b et s_c. Après l'étape Out, b peut s'exécuter sur s_b, s_a ou s_c. De la même manière, après la réduction d'un processus $b[\, \texttt{open}\ a.P \mid a[\, Q\,]\,]$, le processus Q peut s'exécuter soit sur s_a, soit sur s_b. Ces choix conditionnent la dynamique de la distribution et sont visibles dans l'implémentation, sous la forme d'une migration du join calcul, bien qu'ils n'apparaissent pas dans le langage source. Puisque la distribution des locations est transparente dans le join calcul, et comme notre traduction utilise l'envoi de messages asynchrones pour représenter la structure de l'arbre des ambients, tous les différents choix possibles sont implémentables, simplement en utilisant la primitive `go` du join calcul aux moments appropriés. Ceci ne justifie par contre pas le manque de contrôle de la distribution au niveau du calcul des ambients.

Nous avons donc choisi le modèle suivant pour la distribution. L'arbre des ambients est partitionné en plusieurs arbres de locations s'exécutant sur différents sites. Chaque location à la racine d'un site est virtuellement associée à une location représentant un ambient tournant sur un autre site. Ces locations racines ne font que faire suivre les messages vers l'ambient auxquelles elles sont associées, et permettent ainsi la partition vertical de l'arbre dans le modèle horizontal du join calcul. Bien entendu, la location racine du site où s'exécute l'ambient racine n'est associée à aucune autre location. L'exécution des étapes de réduction peut utiliser des messages venant de plusieurs site, et peut avoir pour conséquence la migration de la location représentant un ambient impliqué dans la réduction vers un autre site, en accord avec la partition choisie. Indépendemment des mécanismes précédemment décrits, une configuration distribuée peut être étendue en lançant un nouvel interpréteur sur un nouveau site. Pour se faire, chaque interpréteur reçoit en argument un processus ambient P qu'il doit exécuter, ainsi qu'un pointeur vers un ambient existant auquel P sera virtuellement associé. En d'autres termes, le nouvel interpréteur exécute P et la location racine de cet interpréteur est associée à l'ambient indiqué.

Par exemple, en notant `$jam code` la commande qui démarre un nouvel interpréteur exécutant le processus ambient `code`, et utilisant des entiers pour identifier les sites, considérons les effets des commandes suivantes :

```
$jam "public a[0]"                    sur le site 2
$jam "b[0]" -amb a                    sur le site 1
$jam "c[ m[out c.in b] ]" -amb a      sur le site 3
```

Dans la première commande, le mot clé `public` enregistre l'ambient a comme étant un ambient auquel d'autres interpréteurs peuvent s'attacher. Dans les autres commandes, l'option `-amb a` spécifie que le processus ambient s'exécute justement dans cet ambient. Par soucis de simplicité, nous ne décrivons pas le mécanisme utilisant le serveur de noms de JoCaml qui permet de mettre en place les communications distantes. Le calcul distribué procède comme suit :

$$
\begin{array}{rl}
 & \qquad\qquad \text{site 1} \qquad\qquad\quad \text{site 2} \qquad\quad \text{site 3} \\
 & a\Big[\ b[\,^{Db}\ \mathbf{0}\,] \quad\mid\quad {}^{Da}\ \mathbf{0} \quad\mid\quad c[\,^{Dc}\ m[\,^{Dm}\ \texttt{out}\ c.\texttt{in}\ b\,]\,]\ \Big] \\
\to & a\Big[\ b[\,^{Db}\ \mathbf{0}\,] \quad\mid\quad {}^{Da}\ \mathbf{0} \quad\mid\quad m[\,^{Dm}\ \texttt{in}\ b\,] \mid c[\,^{Dc}\ \mathbf{0}\,]\ \Big] \\
\to & a\Big[\ b[\,^{Db}\ m[\,^{Dm}\ \texttt{in}\ b\,]\,] \quad\mid\quad {}^{Da}\ \mathbf{0} \quad\mid\quad c[\,^{Dc}\ \mathbf{0}\,]\ \Big]
\end{array}
$$

en explicitant par D_a, D_b, D_c, et D_m les contrôleurs de chaque ambient traduit a, b, c et m .

Au cours de la phase d'initialisation, les sites 1 et 3 reçoivent l'interface de a correspondant à leur père. Puisque a est distribué, chaque site remplace dans cette interface l'étiquette *here* correspondant à la location pour lui associer la location racine de leur propre site. Par contre, les noms des canaux du contrôleur ne sont pas modifiés. Ainsi, les contrôleurs de b et c envoient un message *amb* signalant leur existence à a, en utilisant le canal ad hoc de D_a présent dans l'interface, mais l'exécution de b et c reste locale aux sites 1 et 3. Par exemple, l'ambient m présent dans c exécute son étape Out sans devoir se synchroniser avec a. La conséquence de cette étape est l'envoi de deux messages à D_a : un message *amb* signalant la présence de m en tant que sous-ambient de a, et une requête sub_{in}. La synchronisation de cette requête est exécutée sur le site 1, utilisant D_a, et a pour résultat l'envoi de l'interface de b à D_m. La migration de m du site 3 au site 1 ne nécessitera plus l'intervention du site 2.

3.6 Conclusions et enseignements

3.6.1 Traduction et implémentation

Nous avons présenté la traduction du calcul des ambients mobiles dans le join calcul, et son implémentation en JoCaml, avec un haut niveau de parallélisme. A notre connaissance, ceci est la première implémentation distribuée du calcul des ambients.

D'autres travaux se sont penchés sur la question de l'implémentation distribuée des ambients. Par exemple, une machine abstraite distribuée a été proposée [SV01] pour implémenter les Safe Ambients [LS00]. Cette implémentation utilise un protocole bien plus simple que celui décrit ici, en se basant principalement sur le fait que les Safe Ambients éliminent de nombreuses interactions néfastes entre ambients. En effet, les Safe Ambients se basent sur deux mécanismes pour simplifier les interactions : une action ne peut avoir lieu que si la co-action réciproque est présente dans l'ambient ciblé, et un seul processus local (qui n'est pas un sous-ambient) ne peut être actif à la fois. Nous remarquons que ces deux conditions sont nécessaires pour garantir l'absence de processus de la forme $m[\,\texttt{open}\ n.P \mid n[\,\texttt{out}\ m.Q\,]\,]$. Il semble par contre complexe de traduire un processus ambient en un processus safe ambient à thread unique, et l'élimination de certaines interférences (qui consiste ainsi principalement à spécifier l'ordre dans lequel les actions doivent se dérouler) semble se faire au coût de l'expressivité, tout en donnant une spécification plus claire du comportement attendu.

Notre traduction de la sémantique des ambients n'utilisant que des synchronisations locales a été difficile à concevoir et à prouver correcte. Cette expérience souligne les adaptations à apporter au calcul des ambients pour en faire un modèle de programmation. Elle montre aussi comment JoCaml et son modèle formel peuvent être utilisés pour aborder des implémentations de calculs mobiles et distribués. Par exemple, la traduction se base en grande partie sur les filtres pour décrire les synchronisations locales complexes utilisées par l'algorithme, alors qu'une traduction dans un autre langage devrait certainement décomposer ces étapes en une série de lectures et écritures protégées par des verrous.

La correction de la traduction est basée sur l'invariance d'observables, des deadlocks, de l'équité et de la divergence. La traduction est presque aussi détaillée que l'implémentation, ce qui rend plus complexe son analyse formelle. En fait, les preuves ont été plus difficile à écrire que l'implémentation, et nous aurions aimé avoir des outils pour automatiser une partie de ces preuves. Afin d'en réduire la complexité, un calcul intermédiaire a été conçu pour décrire le fonctionnement de l'algorithme dans un cadre plus formel, ce qui nous a permis de découper la preuve en deux parties. La première partie correspond à la preuve de l'algorithme et utilise des simulations couplées (section A.1). La deuxième partie représente le passage d'un calcul à l'autre et se base sur l'utilisation de la technique des diagrammes décroissants pour avoir une preuve plus modulaire (section A.2).

Afin d'avoir une implémentation plus sûre et plus efficace, il serait utile de prendre en compte des informations de type pour les valeurs communiquées à l'intérieur des ambients ainsi que pour les actions de mobilité [CG99, CGG99]. Notre implémentation réalise un typage dynamique des valeurs, alors qu'il serait préférable d'utiliser le système de typage statique de JoCaml. De plus, une analyse statique permettant d'inférer quelles actions ne peuvent avoir lieu dans un ambient donné permettrait d'obtenir des contrôleurs spécialisés plus efficaces.

Enfin, peu de travaux traitent de programmation basée sur les ambients, ou des abstractions nécessaires pour bâtir un langage de haut niveau sur le calcul des ambients. Bien que nous n'ayons pas essayé de modifier le langage source, nous pensons que notre implémentation fournit un système utile pour réaliser des expériences sur des langages basés sur les ambients. Par exemple, nous pouvons facilement ajouter à notre traduction les co-capacités de [LS00].

3.6.2 Les ambients et le join calcul

Nous détaillons maintenant les enseignements que nous ont apportés ce travail, motivant les différentes directions que nous allons poursuivre dans la suite de ce document.

Du point de vue interaction entre localité et communication ou migration, les deux calculs sont aux deux extrêmes : le join calcul est complètement transparent, alors que les ambients sont très locaux. La localité des interactions dans le calcul des ambients rend complexe la migration à distance. En effet, il est nécessaire dans ce cas de spécifier intégralement le chemin que devra suivre l'ambient. De plus, aucune garantie n'est donnée que l'ambient arrivera à bon port. En effet, non seulement d'autres migrations peuvent modifier le chemin, mais celui-ci peut être équivoque : le calcul ne garantit pas l'unicité des noms d'ambients. Ainsi, la programmation distribuée distante

repose intégralement sur le programmeur qui doit s'assurer que certains invariants sont satisfaits. C'est la raison pour laquelle nous estimons que le calcul des ambients ne remplit pas le rôle d'un langage de programmation.

Il est également intéressant de remarquer que la localité des interactions entre ambients n'est pas suffisante pour garantir une implémentation distribuée aisée : deux ambients sur deux machines distinctes peuvent avoir besoin de se synchroniser pour décider si une réduction peut avoir lieu. Cette synchronisation distribuée est la raison pour laquelle notre implémentation n'est pas immédiate, le join calcul ne fournissant aucune primitive de synchronisation distribuée.

Cependant, le calcul des ambients motive l'importance de donner une réelle signification à la localité. En effet, si l'on ne considère pas les pannes, tout terme du join calcul distribué s'exécute indépendemment de l'endroit où il est. Il est ainsi complexe dans le join calcul distribué de coder un pare-feu, c'est à dire une partition contrôlé entre une partie de l'arbre des locations et une autre partie, ou de tout simplement simuler des ressources locales, comme par exemple le nom de la machine courante. C'est pourquoi dans le chapitre suivant nous étendons le join calcul avec une notion de canaux dépendant de la localité.

Remerciements. Ce travail a été réalisé avec Cédric Fournet et Jean-Jacques Lévy. Les commentaires de Luca Cardelli, Fabrice Le Fessant et Luc Maranget nous ont été très utiles.

Chapitre 4

Le join calcul dynamique

Dans le join calcul distribué, chaque canal est défini dans une unique location. Un message sur un canal donné est envoyé dans la location où le canal est défini, quelle que soit la position courante du message. Nous pouvons ainsi considérer que la liaison entre messages et définitions est statique. Nous introduisons dans ce chapitre une extension du join calcul distribué possédant des canaux dynamique, afin de modéliser des comportements dépendant de la position des messages. Contrairement aux canaux statiques, un canal dynamique peut être défini dans plusieurs locations, et un message sur un canal dynamique est envoyé à la location englobante la plus proche qui définit ce canal. Ainsi, lors de la migration de locations, la liaison entre messages dynamiques et leurs définitions peut changer. De nouvelles définitions pour des canaux dynamiques existant, ainsi que de nouveaux canaux dynamiques peuvent être créés lors de l'exécution d'un programme. Afin de rendre l'implémentation du join calcul dynamique plus simple, les synchronisations supplémentaires introduites pour modéliser cette liaison dynamique sont purement locales. Nous étendons également le système de types polymorphe du join calcul distribué pour garantir une forme de réceptivité : tout message dynamique est lié à une définition. Ce système de types associe à toute location l'ensemble des canaux dynamiques qu'elle fournit, et à chaque canal l'ensemble des canaux dont il a besoin. La preuve de correction de ce système de types est assez complexe puisque des noms de canaux apparaissent dans les types.

4.1 Vers le join calcul dynamique

4.1.1 Le cahier des charges

Dans un monde distribué asynchrone, la communication entre processus utilise en général des émetteurs et des récepteurs qui peuvent être dans des locations distinctes. Un récepteur est une *définition*, c'est à dire une association entre un nom de canal et un processus qui sera exécuté à la réception d'un message sur ce canal. Un message émis sur un canal est envoyé à une location contenant une définition pour ce canal. Ainsi, la liaison entre un message et une définition dépend de comment est choisie la location où le message sera envoyé. Un choix possible consiste en une liaison *statique*, qui ne dépend que du nom du message et pas de l'endroit où il réside. Ceci simplifie beaucoup la communication distante puisqu'il n'est pas nécessaire de se soucier de l'endroit où l'on est pour envoyer un message, et c'est le choix qui a été retenu pour le join calcul distribué [FGL+96]. Pour ce faire, le join calcul distribué impose que tout nom de canal ne soit défini que dans une location unique, et c'est à cette location qu'un message sur ce canal est envoyé. Un autre choix possible est de faire dépendre la liaison de l'endroit où le message réside. Cette liaison *dynamique* permet de modéliser des comportements locaux. Ce chapitre présente une extension du join calcul distribué avec de tels canaux dynamiques.

Le join calcul distribué est un calcul de processus possédant une notion de lieu, sous la forme de *locations*, qui sont structurées en un arbre. Un sous-arbre de locations migre lorsque la racine de ce sous-arbre migre. Les noms de canaux dans le join calcul ont une portée globale et statique : tout message sur un nom donné est envoyé de manière transparente dans l'unique location définissant ce nom, et ce indépendemment des positions respectives dans l'arbre des locations. Nous étendons dans ce chapitre le join calcul présenté en section 2.3, à quelques détails près qui seront discutés au moment opportun.

Une implémentation distribuée du join calcul, appelée JoCaml [Le 98], dispose déjà d'une notion de liaison dynamique pour les fonctions définies dans l'environnement d'exécution, comme la bibliothèque standard. Modéliser ce comportement est un de nos objectifs.

Nous désirons ajouter à notre cahier des charges pour le join calcul dynamique la possibilité de créer de nouvelles définitions pour des noms dynamiques existant, ainsi que la possibilité d'introduire de nouveaux noms dynamiques. Par exemple, un programmeur peut écrire une définition pour un canal `print` possédant un bug. Lorsqu'il se rend compte du problème, le programmeur crée une nouvelle définition de `print` plus correcte, et il ne peut bien sûr s'empêcher d'y ajouter de nouvelles fonctionnalités, comme par exemple un moyen de spécifier la couleur d'impression. Cette fonctionnalité se traduit par un nouveau nom de canal dynamique `setcolor`, associé à une définition pour ce canal. Le programmeur envisage bien entendu de continuer à utiliser ses anciens programmes avec la nouvelle définition de `print`, tout en en créant de nouveaux utilisant la nouvelle fonctionnalité.

Cependant, un tel système où les noms de canaux dynamiques sont des valeurs de première classe, qui peuvent être envoyés dans des messages et redéfinis, rend complexe la vérification manuelle qu'à tout message est bien associée une définition. Cette propriété de *réceptivité* devrait interdire l'utilisation de `setcolor` alors que l'on utilise encore la vieille version de `print` ne fournissant pas ce canal. Nous introduisons donc un système de types statique nous garantissant que l'utilisation locale de ressources est toujours sûre.

4.1.2 Objectivité et Subjectivité

La plupart des actions possibles dans un calcul de processus concurrents peuvent se partitionner en deux grandes catégories : les actions objectives et les actions subjectives [CG98]. Dans notre cas, la question se pose pour la reliaison dynamique : est-elle objective, c'est à dire provoqué par un processus qui est une garde du processus subissant la reliaison (étant sous une garde, celuici n'est pas actif), ou est-elle subjective, c'est à dire provoqué par un processus qui modifie son environnement, le processus relié étant alors en cours d'exécution. Par exemple, les systèmes ayant des variables à portée dynamique, comme les paramètres implicites de [LSML00], ont une forme objective de liaison dynamique, puisque la valeur associée à une variable n'est modifiée que dans la continuation de la reliaison, et nulle part ailleurs. Par contre, le système JoCaml est subjectif puisque la reliaison des fonctions de la bibliothèque standard s'effectue lors de la migration, qui est une opération subjective dans le join calcul, puisqu'elle affecte toute la location contenant la requête de migration ainsi que ses sous-locations qui s'exécutent en parallèle de la requête de migration. Le calcul des Ambients [CG98] est aussi un calcul subjectif puisqu'une capacité mentionnant un nom b fait référence à un ambient voisin portant ce nom, et que toute migration de cet ambient modifie son environnement local, donc les ambients auxquels ce nom fait référence.

Un calcul objectif simplifie l'analyse d'un programme, puisque seule la continuation de l'action effectuant la reliaison est affectée. A contrario, un calcul subjectif doit prendre en compte le fait que l'environnement peut être modifié, alors qu'il n'est pas nécessairement connu, lors du typage par exemple. Ceci rend la preuve de réceptivité plus complexe, mais donne la possibilité de modifier l'environnement de processus en cours d'exécution, par exemple pour modifier de manière transparente une fonction de bibliothèque.

4.1.3 Le join calcul dynamique

Notre extension consiste en une nouvelle catégorie de canaux, les canaux dynamiques. La définition liée à un canal donné en une location donnée est celle présente dans la location englobante la plus proche définissant ce canal. Nous disons qu'un canal dynamique est *disponible* à un endroit donné lorsqu'il existe une définition pour ce canal dans une location englobante. La migration dans le join calcul étant subjective, et la migration étant l'action provoquant une reliaison des noms dynamiques, notre calcul possède une liaison dynamique subjective. Comme il est possible dans notre calcul de redéfinir un nom déjà défini dans une location englobante, ce calcul est une extension du modèle de JoCaml où seuls les *runtimes*, c'est à dire les locations racines ne migrant pas, peuvent fournir des définitions pour les canaux dynamiques.

Le join calcul dynamique est conçu pour être implémenté dans un cadre distribué, comme le join calcul distribué. Pour ce faire, ce dernier rend déterministe et local le choix de la location où envoyer un message en n'autorisant les définitions pour un nom donné à n'être présentes que dans une seule location, qui ne peut changer au cours des réductions. Ainsi on peut considérer que chaque nom de canal contient également le nom de la location où il est défini. Notre calcul autorisant les définitions pour un nom dans plusieurs locations, il pourrait sembler que son implémentation distribuée soit compromise. Il n'en est rien puisque le choix de la location où envoyer un message reste univoque : il s'agit de la location englobante définissant ce canal, qui est nécessairement unique. Notre calcul est aussi conçu de telle sorte que la résolution de la destination d'un message est effectuée en

considérant uniquement la location courante, sans regarder les locations englobantes. Ceci nous garantit l'absence de synchronisations distribuées.

Un choix difficile lorsque l'on considère un système possédant une forme de liaison dynamique est la spécification du comportement lorsque l'on utilise un nom qui n'est pas lié. Nous choisissons de ne rien faire lorsqu'un message est présent sur un canal dynamique non disponible, et présentons un système de types garantissant que cette situation ne peut se produire, ce malgré l'introduction de nouvelles définitions et de nouveaux noms dynamiques lors de l'exécution.

Une partie de ce travail aurait pu être réalisée dans un π calcul distribué. Cependant, la notion de liaison dynamique associée à la localité serait plus subtile dans le π calcul puisque l'association entre récepteurs et émetteurs est déjà dynamique. De plus, les récepteurs peuvent disparaître dans le π calcul, ce qui rend difficile la distinction entre absence de deadlock [ABL99, KSS00] et présence de récepteur.

Nous présentons dans la section 4.2 la syntaxe et la sémantique du join calcul distribué. Puis dans la section 4.3 nous discutons des choix faits pour notre système et donnons des exemples. Le système de types est présenté en section 4.4 et des exemples sont donnés dans la section 4.5. Nous établissons la correction du système de types en section 4.6. Les preuves elles-mêmes sont présentées en appendice B.1. Nous concluons en section 4.7.

4.2 Syntaxe et sémantique

Les syntaxe, sémantique structurelle et règles de réduction sont données respectivement dans les figures 4.1, 4.4, et 4.5. Celles-ci sont très semblables à celles du join calcul distribué décrit en section 2.3.1. La différence principale, hormis les nouvelles constructions dues aux canaux dynamiques, est le remplacement de la règle d'équivalence structurelle Scope par une règle de réduction Def pour dissoudre les définitions locales.

Nous détaillons maintenant le join calcul dynamique. Nous supposons donné un ensemble infini dénombrable de noms notés $m, n, x.$ les noms de locations sont notés a, b, c, les noms de canaux dynamiques sont notés $\mathbf{m}, \mathbf{n}, \mathbf{x}$, les noms de canaux statiques sont notés $\mathtt{m}, \mathtt{n}, \mathtt{x}$, et les variables sont notées u, y. Nous notons $\widetilde{n}$ pour un nuplet potentiellement vide de noms. Nous notons P (respectivement D) pour les processus (respectivement les définitions) qui ne contiennent aucune occurrence de messages résolus (de la forme $a.n\langle\widetilde{n_i}\rangle$). A contrario, nous notons $\mathcal{P}$ (respectivement $\mathcal{D}$) pour les processus (respectivement les définitions) pouvant contenir des messages résolus, puisque ceux-ci peuvent apparaître dans les processus actifs ou les définitions actives après plusieurs étapes de réduction.

Les noms libres, reçus, et définis sont définis de la manière habituelle. Intuitivement, une définition locale $\mathtt{def}\ D\ \mathtt{in}\ P$ lie dans D et P les noms de canaux statiques et les noms de locations définis par D. Cependant, une telle définition ne lie pas les noms de canaux dynamiques définis par D. Une restriction $\nu\mathbf{n}.P$ lie le nom dynamique $\mathbf{n}$ dans P. Une règle de réaction $J \triangleright P$ lie les noms reçus du filtre J (les paramètres formels) dans P. Nous introduisons également les notions de *nom définis localement* noté *dln* comme l'ensemble des noms définis dans une location donnée, de *noms définis statiques* noté *dsn* (respectivement de *noms définis dynamiques* noté *ddn*) comme l'ensemble des noms définis qui sont statiques (respectivement, l'ensemble des noms définis qui sont dynamiques).

Nous donnons une définition formelle de ces ensembles de noms.

Définition 4.2.1 (Noms du join calcul dynamique) *Nous définissons en figures 4.2 et 4.3 l'ensemble des noms libres fn, l'ensemble des noms reçus rn, l'ensemble des noms définis dn, l'ensemble des noms définis statiques dsn, l'ensemble des noms définis dynamiques ddn, l'ensemble des noms définis localement dln et l'ensemble des noms liés bn. L'ensemble des noms dynamiques locaux dyn et l'ensemble des noms dynamiques importés imp sont respectivement définis pour une location active* $\alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,F}$ *ou a* $[\mathcal{D} : \mathcal{P}]^{\Delta,I}$ *comme étant* $dyn(a) = \Delta$ *et* $imp(a) = I$*. L'ensemble des noms d'une configuration* $names(\mathcal{S})$ *est tout simplement l'ensemble* $fn(\mathcal{S}) \cup bn(\mathcal{S})$*.*

Nous remarquons que les noms reçus ne sont définis que pour des filtres. Le cas $dsn(a\langle\widetilde{y_i}\rangle)$ n'est pas défini parce qu'un tel cas ne peut arriver en contexte d'évaluation si le processus est bien typé ; le théorème 5 garantissant en effet que tout message en contexte d'évaluation dans une configuration bien typée est envoyé sur un nom de canal, donc ne peut être envoyé sur un nom de location. Nous remarquons que pour une définition donnée, les ensembles *dsn*, *ddn* et *dln* sont des sous-ensembles de *dn*. Les noms définis localement sont simplement les noms définis sans inspecter les sous-locations présentes dans la définition donnée.

$\mathcal{P}$	::=		processus
		$\mathbf{0}$	processus inerte
	|	$\mathcal{P} \mid \mathcal{P}'$	composition parallèle
	|	$n\langle \widetilde{n}_i \rangle ; P$	message
	|	$\texttt{go}\ n ; P$	requête de migration
	|	$\texttt{def}\ D\ \texttt{in}\ P$	définition locale
	|	$\nu\mathbf{n}.P$	création de canal dynamique
	|	$a.n\langle \widetilde{n}_i \rangle$	message résolu
$\mathcal{D}$	::=		définition
		$\top$	définition vide
	|	$\mathcal{D}, \mathcal{D}'$	composition
	|	$J \triangleright P$	règle de réaction
	|	$a\,[\mathcal{D} : \mathcal{P}]^{\Delta, I}$	location repliée
J	::=		filtre
		$n\langle \widetilde{y} \rangle$	message requis
	|	$J \mid J'$	synchronisation
n	::=		nom
		$\mathbf{n}$	nom de canal dynamique
	|	$\mathtt{n}$	nom de canal statique
	|	a	nom de location
	|	y	variable
Δ, I	::=		ensemble de noms dynamiques
		$\emptyset$	ensemble vide
	|	$\{\mathbf{n}\}$	singleton de nom dynamique
	|	$\{y\}$	singleton de variable
	|	$\Delta \cup \Delta$	union
α	::=		chaîne de locations
			chaîne vide
	|	$\alpha' a$	a sous-location de α'
$\mathcal{S}$	::=		configuration
		Ω	configuration vide
	|	$\mathcal{S} \parallel \mathcal{S}'$	composition de soupes
	|	$\alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta, I, F}$	location dépliée

FIG. 4.1 – Syntaxe du join calcul dynamique

Par la suite nous nous donnons la possibilité de noter $dln(a)$ pour $dln(\mathcal{D})$ pour toute location active dépliée $\alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta, I, F}$ ou repliée $a\,[\mathcal{D} : \mathcal{P}]^{\Delta, I}$.

Une configuration est composée de plusieurs locations dépliées (ou *soupes*) s'exécutant en parallèle. Chaque location contient un multi-ensemble de définitions actives $\mathcal{D}$ ainsi qu'un multi-ensemble de processus actifs $\mathcal{P}$.

Comme dans le join calcul distribué, les locations sont structurées en un arbre et chaque location possède un nom statique unique. La sémantique chimique utilisant une représentation plate des locations dépliées, la structure d'arbre est contenue dans le noms des locations : une location dépliée a le nom αa si son nom est a et si le chemin de la racine de l'arbre jusqu'à cette location est α.

Pour prendre en compte la différence de comportement de messages sur des noms statiques et dynamiques, nous séparons l'envoi de message en deux étapes (d'une manière très similaire à [FMLR00]). La première étape, appelée *étape de résolution*, détermine la location où envoyer le message et préfixe le message avec le nom de cette location. Cette location est bien entendu celle contenant la définition à laquelle le message est lié. Dans le cas d'un message sur un canal statique, il s'agit de l'unique location active contenant une définition pour ce canal ; dans le cas d'un message sur un canal dynamique, il s'agit de la location englobante la plus proche contenant une définition

$$
\begin{array}{rl}
fn(\mathbf{0})=\emptyset & fn(\mathcal{P} \mid \mathcal{P}')=fn(\mathcal{P}) \cup fn(\mathcal{P}') \\
fn(n\langle \widetilde{m} \rangle; P)=\{n\} \cup \widetilde{m} \cup fn(P) & fn(\mathtt{go}\ n; P)=\{n\} \cup fn(P) \\
fn(\mathtt{def}\ D\ \mathtt{in}\ P)=(fn(D) \cup fn(P)) \setminus dsn(D) & fn(\nu \mathbf{n}.P)=fn(P) \setminus \{\mathbf{n}\} \\
fn(a.n\langle \widetilde{m} \rangle)=\{a\} \cup \{n\} \cup \widetilde{m} & fn(\top)=\emptyset \\
fn(\mathcal{D}, \mathcal{D}')=fn(\mathcal{D}) \cup fn(\mathcal{D}') & fn(J \triangleright P)=(fn(J) \cup fn(P)) \setminus rn(J) \\
fn(a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I})=\{a\} \cup \Delta \cup fn(\mathcal{D}) \cup fn(\mathcal{P}) \cup I & fn(n\langle \widetilde{y_i} \rangle)=\{n\} \cup \bigcup_i \{y_i\} \\
fn(J \mid J')=fn(J) \cup fn(J') & fn(\Omega)=\emptyset \\
fn(\mathcal{S} \parallel \mathcal{S}')=fn(\mathcal{S}) \cup fn(\mathcal{S}') & fn(\alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,F})=fn(\mathcal{D}) \cup \Delta \cup I \cup \{a\} \cup fn(\mathcal{P})
\end{array}
$$

$$
\begin{array}{rl}
rn(n\langle \widetilde{y_i} \rangle)=\bigcup_i \{y_i\} i & rn(J \mid J')=rn(J) \cup rn(J')
\end{array}
$$

$$
\begin{array}{rl}
dn(\top)=\emptyset & dn(\mathcal{D}, \mathcal{D}')=dn(\mathcal{D}) \cup dn(\mathcal{D}') \\
dn(J \triangleright Q)=dn(J) & dn(a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I})=\{a\} \cup dn(\mathcal{D}) \\
dn(n\langle \widetilde{y_i} \rangle)=\{n\} & dn(J \mid J')=dn(J) \cup dn(J') \\
dn(\Omega)=\emptyset & dn(\mathcal{S} \parallel \mathcal{S}')=dn(\mathcal{S}) \cup dn(\mathcal{S}') \\
dn(\alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,F})=dn(\mathcal{D}) \cup \{a\} &
\end{array}
$$

$$
\begin{array}{rl}
dsn(\top)=\emptyset & dsn(\mathcal{D}, \mathcal{D}')=dsn(\mathcal{D}) \cup dsn(\mathcal{D}') \\
dsn(J \triangleright Q)=dsn(J) & dsn(a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I})=\{a\} \cup dsn(\mathcal{D}) \\
dsn(\mathbf{n}\langle \widetilde{y_i} \rangle)=\{\mathbf{n}\} & dsn(\mathbf{n}\langle \widetilde{y_i} \rangle)=\emptyset \\
dsn(y\langle \widetilde{y_i} \rangle)=\emptyset & dsn(J \mid J')=dsn(J) \cup dsn(J') \\
dsn(\Omega)=\emptyset & dsn(\mathcal{S} \parallel \mathcal{S}')=dsn(\mathcal{S}) \cup dsn(\mathcal{S}') \\
dsn(\alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,F})=dsn(\mathcal{D}) \cup \{a\} &
\end{array}
$$

FIG. 4.2 – Noms du join calcul dynamique, partie 1

$$
\begin{array}{rl}
ddn(\top)=\emptyset & ddn(\mathcal{D}, \mathcal{D}')=ddn(\mathcal{D}) \cup ddn(\mathcal{D}') \\
ddn(J \triangleright Q)=ddn(J) & ddn(a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I})=ddn(\mathcal{D}) \\
ddn(\mathbf{n}\langle \widetilde{y_i} \rangle)=\emptyset & ddn(\mathbf{n}\langle \widetilde{y_i} \rangle)=\{\mathbf{n}\} \\
ddn(y\langle \widetilde{y_i} \rangle)=\{y\} & ddn(J \mid J')=ddn(J) \cup ddn(J') \\
ddn(\Omega)=\emptyset & ddn(\mathcal{S} \parallel \mathcal{S}')=ddn(\mathcal{S}) \cup ddn(\mathcal{S}') \\
ddn(\alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,F})=ddn(\mathcal{D}) &
\end{array}
$$

$$
\begin{array}{rl}
dln(\top)=\emptyset & dln(\mathcal{D}, \mathcal{D}')=dln(\mathcal{D}) \cup dln(\mathcal{D}') \\
dln(J \triangleright P)=dn(J) & dln(a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I})=\emptyset
\end{array}
$$

$$
\begin{array}{rl}
bn(\mathbf{0})=\emptyset & bn(\mathcal{P} \mid \mathcal{P}')=bn(\mathcal{P}) \cup bn(\mathcal{P}') \\
bn(n\langle \widetilde{m} \rangle; P)=bn(P) & bn(\mathtt{go}\ n; P)=bn(P) \\
bn(\mathtt{def}\ D\ \mathtt{in}\ P)=dsn(D) \cup bn(D) \cup bn(P) & bn(\nu \mathbf{n}.P)=\{\mathbf{n}\} \cup bn(P) \\
bn(a.n\langle \widetilde{m} \rangle)=\emptyset & bn(\top)=\emptyset \\
bn(\mathcal{D}, \mathcal{D}')=bn(\mathcal{D}) \cup bn(\mathcal{D}') & bn(J \triangleright P)=rn(J) \cup bn(P) \\
bn(a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I})=bn(\mathcal{D}) \cup bn(\mathcal{P}) & bn(n\langle \widetilde{y_i} \rangle)=\emptyset \\
bn(J \mid J')=\emptyset & bn(\Omega)=\emptyset \\
bn(\mathcal{S} \parallel \mathcal{S}')=bn(\mathcal{S}) \cup bn(\mathcal{S}') & bn(\alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,F})=bn(\mathcal{D}) \cup bn(P)
\end{array}
$$

FIG. 4.3 – Noms du join calcul dynamique, partie 2

$$\frac{\mathcal{S} =_\alpha \mathcal{S}'}{\mathcal{S} \equiv \mathcal{S}'}\ [\textsc{Str}\text{-}\alpha] \qquad \frac{\forall \beta \in loc(\mathcal{S}), a \notin \beta \qquad G = Lookup(F, I, \Delta, a)}{\alpha\left[a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I}, \mathcal{D}' : \mathcal{P}'\right]^{\Delta',I',F} \equiv \alpha\,[\mathcal{D}' : \mathcal{P}']^{\Delta',I',F} \parallel \alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,G}}\ [\textsc{Tree}]$$

FIG. 4.4 – Sémantique : règles structurelles

pour ce canal. La deuxième étape est l'*étape de communication*, qui transporte un message résolu jusqu'à sa destination (c'est la règle COMM). Nous remarquons que nous ne nous intéressons qu'à la résolution de la destination d'un message et non pas au routage des messages (règle COMM) ou des locations (règle GO), comme par exemple dans [US01]. À ce point de vue, notre sémantique reste très proche du join calcul distribué.

Afin de simplifier la syntaxe et la sémantique, et pour nous assurer que l'étape de résolution pour les messages dynamiques peut bien avoir lieu localement, nous associons à chaque location dépliée une fonction de résolution F qui prend en argument un nom de canal dynamique et retourne le nom de la location englobante la plus proche définissant ce canal. Cette fonction est utilisée dans la règle de résolution des messages dynamiques (NL-DYN), en notant $\perp$ lorsque la fonction n'est pas définie. La règle de résolution des messages statiques (NL-STAT) préfixe le message avec l'unique location définissant le canal. Une étape de résolution étant locale à la location contenant le message, nous autorisons la spécification d'une continuation aux messages non résolus. Cette continuation est activée au moment de la résolution du message (règles NL-STAT et NL-DYN). Nous donnons en section 4.3 un exemple motivant cette fonctionnalité. Nous remarquons que, malgré cette continuation, l'envoi et l'utilisation d'un message sont toujours asynchrones. Par la suite, nous abrégeons $n\langle\widetilde{m}\rangle;\mathbf{0}$ en $n\langle\widetilde{m}\rangle$.

Une règle de réaction a la forme $n_1\langle\widetilde{y_1}\rangle \mid \ldots \mid n_k\langle\widetilde{y_k}\rangle \triangleright P$ où les n_i sont des noms de canaux ou des variables, où les $\widetilde{y_i}$ sont les paramètres formels (aussi appelés *noms reçus*), et où P est le processus gardé par la règle de réaction. Un nom de canal dans le filtre d'une règle de réaction peut être soit de la forme $\mathtt{n}_i$ si le canal est statique, soit de la forme $\mathbf{n}_i$ si le canal est dynamique, soit de la forme y_i si c'est une variable (ce cas ne se produit que si la règle de réaction est dans le processus gardé d'une autre règle de réaction, le système de types garantissant que cette variable ne peut être substituée que par un nom dynamique et que la présence d'une variable en tant que nom défini par un filtre ne peut se produire en contexte d'évaluation) . Une règle de réaction est activée lorsque des messages sur chacun des n_i sont présents parmi les processus actifs de la location. Ces messages sont consommés et le processus gardé est activé après avoir substitué les paramètres formels par les noms contenus dans les messages consommés (grâce à la substitution σ_{rn}). Cette étape correspond à la règle JOIN. En fait, la règle que nous utilisons est légèrement différente, puisque seuls des messages résolus peuvent être consommés. Nous notons $a.J$ pour la composition parallèle de messages résolus où chaque message possède le préfixe a (*i.e.* $a.(J \mid J') = a.J \mid a.J'$). De nouvelles définitions peuvent être introduites grâce à la construction $\mathtt{def}\ D\ \mathtt{in}\ P$, où les noms de canaux statiques définis dans D ont pour porté D et P. Les nouveaux noms de canaux dynamiques sont introduits par la construction $\nu\mathbf{n}.P$.

Les locations, qu'elles soient dépliées ou repliées, spécifient grâce à l'ensemble Δ les noms dynamiques qu'elles *définissent*, et grâce à l'ensemble I les noms dynamiques qu'elles *importent* (*i.e.* les noms dynamiques qui doivent être définis dans les locations englobantes).

L'évaluation d'une requête de migration de la forme $\mathtt{go}\ b; P$ provoque la migration de la location courante ainsi que de toutes ses sous-locations (règle GO).

Une location dépliée αa peut être repliée à l'intérieur de sa location père α (avant de migrer, par exemple) grâce à la règle TREE. La première condition de cette règle s'assure que toutes les sous-locations de a ont été repliées, afin de conserver la structure d'arbre des locations dépliées (dans la présentation originale du join calcul distribué, la location a était dite *gelée*). Inversement, lors du dépliage d'une location, il est nécessaire de calculer sa fonction de résolution. Ce calcul utilise l'opérateur *Lookup* qui prend en argument la fonction de résolution de la location père et la modifie de telle sorte qu'elle soit adaptée à la location étant dépliée.

Définition 4.2.2 (opérateur *Lookup***)** *L'opérateur Lookup prend en arguments la fonction de résolution F de la location père, les noms dynamiques importés I, les noms dynamiques localement définis Δ, ainsi que le nom de la location courante a, et retourne une fonction de résolution $G = Lookup(F, I, \Delta, a)$ qui associe aux noms localement définis la location courante, et aux noms importés le résultat de la fonction de résolution de la location père. Nous écrivons $\perp$ lorsque la fonction n'est pas définie.*

$$\frac{dsn(D) \cap (bn(\mathcal{S}) \cup bn(\mathcal{D}, D) \cup bn(\mathcal{P} \mid P)) = \emptyset \qquad dln(D) \cap ddn(D) = \emptyset}{\mathcal{S} \parallel \alpha a\,[\mathcal{D} : \mathcal{P} \mid \texttt{def}\ D\ \texttt{in}\ P]^{\Delta,I,F} \longrightarrow \mathcal{S} \parallel \alpha a\,[\mathcal{D}, D : \mathcal{P} \mid P]^{\Delta,I,F}}\ [\textsc{Def}]$$

$$\frac{\{\mathbf{n}\} \cap (bn(\mathcal{S}) \cup bn(\mathcal{D}) \cup bn(\mathcal{P} \mid P)) = \emptyset}{\mathcal{S} \parallel \alpha a\,[\mathcal{D} : \mathcal{P} \mid \nu\mathbf{n}.P]^{\Delta,I,F} \longrightarrow \mathcal{S} \parallel \alpha a\,[\mathcal{D} : \mathcal{P} \mid P]^{\Delta,I,F}}\ [\textsc{Nu}]$$

$$\frac{\mathtt{n} \in dln(b)}{\alpha a\,[\mathcal{D} : \mathcal{P} \mid \mathtt{n}\langle\widetilde{v}\rangle; P]^{\Delta,I,F} \longrightarrow \alpha a\,[\mathcal{D} : \mathcal{P} \mid b.\mathtt{n}\langle\widetilde{v}\rangle \mid P]^{\Delta,I,F}}\ [\text{NL-}\textsc{Stat}]$$

$$\frac{F(\mathbf{n}) = b \wedge b \neq \bot}{\alpha a\,[\mathcal{D} : \mathcal{P} \mid \mathbf{n}\langle\widetilde{v}\rangle; P]^{\Delta,I,F} \longrightarrow \alpha a\,[\mathcal{D} : \mathcal{P} \mid b.\mathbf{n}\langle\widetilde{v}\rangle \mid P]^{\Delta,I,F}}\ [\text{NL-}\textsc{Dyn}]$$

$$\frac{dom(\sigma_{rn}) = rn(J)}{\alpha a\,[\mathcal{D}, J \triangleright P : \mathcal{P} \mid a.J\sigma_{rn}]^{\Delta,I,F} \longrightarrow \alpha a\,[\mathcal{D}, J \triangleright P : \mathcal{P} \mid P\sigma_{rn}]^{\Delta,I,F}}\ [\textsc{Join}]$$

$$\frac{}{\begin{array}{l}\alpha a\,[\mathcal{D}_a : \mathcal{P}_a \mid b.n\langle\widetilde{v}\rangle]^{\Delta_a,I_a,F_a} \parallel \beta b\,[\mathcal{D}_b : \mathcal{P}_b]^{\Delta_b,I_b,F_b} \\ \longrightarrow \alpha a\,[\mathcal{D}_a : \mathcal{P}_a]^{\Delta_a,I_a,F_a} \parallel \beta b\,[\mathcal{D}_b : \mathcal{P}_b \mid b.n\langle\widetilde{v}\rangle]^{\Delta_b,I_b,F_b}\end{array}}\ [\textsc{Comm}]$$

$$\frac{}{\begin{array}{l}\alpha\left[a\,[\mathcal{D}_a : \mathcal{P}_a \mid \texttt{go}\ b; Q]^{\Delta_a,I_a}, \mathcal{D} : \mathcal{P}\right]^{\Delta,I,F} \parallel \beta b\,[\mathcal{D}_b : \mathcal{P}_b]^{\Delta_b,I_b,F_b} \\ \longrightarrow \alpha\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,F} \parallel \beta b\left[a\,[\mathcal{D}_a : \mathcal{P}_a \mid Q]^{\Delta_a,I_a}, \mathcal{D}_b : \mathcal{P}_b\right]^{\Delta_b,I_b,F_b}\end{array}}\ [\textsc{Go}]$$

FIG. 4.5 – Sémantique : règles de réduction

Nous avons :

$$G(n) = \begin{cases} a & n \in \Delta \\ F(n) & n \notin \Delta \wedge n \in I \\ \bot & n \notin \Delta \wedge n \notin I \end{cases}$$

Nous définissons maintenant l'α-conversion, en ne considérant que l'α-conversion vers des noms frais.

Définition 4.2.3 (Contextes) *Un contexte, défini en figure 4.6 est un processus, une définition, une soupe ou une configuration qui contient un unique trou. Ce trou peut être rempli par un processus, une définition ou une location dépliée tant que le résultat est syntaxiquement correct.*

Définition 4.2.4 (α-conversion) *Soient $\mathcal{S}$ et $\mathcal{S}'$ des configurations. Nous avons $\mathcal{S} =_\alpha \mathcal{S}'$ si et seulement si l'une des propriétés suivantes est satisfaite :*

$$\begin{array}{rlll} & \mathcal{S} = C(J \triangleright Q) & et\ \mathcal{S}' = C(J\theta_{rn} \triangleright Q\theta_{rn}) & pour\ tout\ C,\ J,\ Q \\ or & \mathcal{S} = C(\texttt{def}\ D\ \texttt{in}\ Q) & et\ \mathcal{S}' = C(\texttt{def}\ D\theta_{dsn}\ \texttt{in}\ Q\theta_{dsn}) & pour\ tout\ C,\ D,\ Q \\ or & \mathcal{S} = C(\nu\mathbf{n}.Q) & et\ \mathcal{S}' = C(\nu\mathbf{n}'.Q\{^{\mathbf{n}'}/_{\mathbf{n}}\}) & pour\ tout\ C,\ \mathbf{n},\ Q \end{array}$$

avec

- *les noms renommés par θ_{rn} sont des noms reçus de J ($dom(\theta_{rn}) \subseteq rn(J)$) ;*
- *les noms renommés par θ_{dsn} sont des noms statiques définis de D ($dom(\theta_{dsn}) \subseteq dsn(D)$) ;*
- *le renommage est un renommage vers des noms frais ($(ran(\theta_{dsn}) \cup ran(\theta_{rn}) \cup \{\mathbf{n}'\}) \cap (bn(\mathcal{S}) \cup fn(\mathcal{S})) = \emptyset$) ;*
- *les substitutions θ_{dsn} et θ_{rn} sont injectives.*

Nous rappelons que, comme en section 2.3, nous disons qu'une location est *active* si elle est dépliée ou peut être dépliée par équivalence structurelle. Nous appelons contexte d'évaluation un contexte dont le trou se trouve dans la partie processus d'une location active et n'est pas gardé.

Dans la suite, nous ne considérons qu'une classe restreinte de configurations : toute location possède un nom unique ; tout nom statique défini n'est défini que dans une unique location ; les filtres sont linéaires pour les noms reçus, *i.e.* aucun nom reçu ne peut être présent plus d'une fois dans un filtre ; les noms libres et liés sont distincts ($fn(\mathcal{S}) \cap bn(\mathcal{S}) = \emptyset$). Nous appelons cette dernière condition la *condition d'hygiène*. Nous remarquons que nous n'imposons pas la linéarité des filtres pour les noms définis, celle-ci n'étant pas nécessairement préservée par réduction (comme par exemple pour le processus $\texttt{def}\ \texttt{create}_2\langle x_1, x_2\rangle \triangleright a\,[x_1 \mid x_2 \triangleright \mathbf{0} : \mathbf{0}]^{\{x_1,x_2\},\emptyset}\ \texttt{in}\ \texttt{create}_2\langle \mathbf{d}, \mathbf{d}\rangle$). L'extension du join calcul avec des filtres non linéaires pour les noms définis n'a un impact que sur

C	::=		contexte
		$(\,\cdot\,)$	trou
	$\mid$	$n\langle\widetilde{m}\rangle; C$	message
	$\mid$	$C \mid P$	composition parallèle gauche
	$\mid$	$P \mid C$	composition parallèle droite
	$\mid$	`def` C `in` P	définition d'un `def`
	$\mid$	`def` D `in` C	processus gardé d'un `def`
	$\mid$	$\nu\mathbf{n}.C$	restriction
	$\mid$	$J \triangleright C$	règle de réaction
	$\mid$	$C, \mathcal{D}$	composition de définition gauche
	$\mid$	$\mathcal{D}, C$	composition de définition droite
	$\mid$	$a\,[C : \mathcal{P}]^{\Delta,I}$	définition d'une location repliée
	$\mid$	$a\,[\mathcal{D} : C]^{\Delta,I}$	processus d'une location repliée
	$\mid$	$\mathcal{S} \parallel C \parallel \mathcal{S}'$	configuration, avec C une location dépliée
	$\mid$	$\alpha a\,[C : \mathcal{P}]^{\Delta,I,F}$	définition d'une location dépliée
	$\mid$	$\alpha a\,[\mathcal{D} : C]^{\Delta,I,F}$	processus d'une location dépliée

FIG. 4.6 – Contexte

l'implémentation de la compilation des filtres de messages, qui ne peut plus utiliser un vecteur de bits pour détecter la possible activation d'un filtre.

La première condition de la règle DEF garantit la préservation de la condition d'hygiène. Puisque cette condition est vraie avant la réduction, les noms statiques définis de D qui deviennent libres étaient liés, donc ne peuvent être libres dans la configuration initiale. La définition ainsi dépliée ne capture donc aucun nom de la configuration. La première condition de la règle DEF vérifie simplement que les noms statiques définis de D qui deviennent libres ne sont pas liés dans la configuration finale. La deuxième condition vérifie que aucun nom localement défini de D n'est un nom dynamique (les définitions bien typées satisfont cette propriété). La condition d'hygiène est aussi préservée par la règle NU.

Les figures 4.4 et 4.5 ne mentionnent la configuration globale que pour les règles DEF et NU. Les autres règles laissent implicite le contexte composé du reste de la configuration (les autres locations dépliées).

L'équivalence structurelle $\equiv$ est la plus petite relation réflexive, symétrique et transitive engendrée par les règles de la figure 4.4 telle que l'opérateur de composition parallèle "|" est associatif, commutatif et possède $\mathbf{0}$ comme élément neutre, telle que l'opérateur de composition de définitions "," est associatif, commutatif et possède $\top$ comme élément neutre, et telle que l'opérateur de composition des soupes " $\parallel$ " est associatif, commutatif et possède Ω comme élément neutre. La relation de réduction $\rightarrow$ est la plus petite relation engendrée par les règles de la figure 4.5 telle que $\equiv\!\rightarrow\!\equiv \subseteq \rightarrow$.

4.3 Discussion et exemples

Comme nous l'avons suggéré en section 2.3.1, nous justifions maintenant l'utilisation d'une règle de réduction pour la dissolution d'une définition locale, ou en d'autre termes l'interdiction de son repliage. La motivation généralement avancée pour autoriser le repliage d'une définition locale est de permettre le renommage par α-conversion des noms définis qui n'est possible que si la définition est repliée. Ce choix n'est plus évident dans le join calcul distribué puisqu'il est parfois impossible de replier une définition, si par exemple un des noms définis a été envoyé à une autre location. De plus, autoriser le repliage et dépliage de définitions dans l'équivalence structurelle rend difficile la description des termes manipulés si l'on désire faire abstraction de cette équivalence. Ainsi, les preuves de la traduction du calcul des ambients dans le join calcul (présentées dans l'appendice A) orientent l'équivalence structurelle pour clarifier la forme des termes. Le repliage des définitions ne nous semblant pas utile, nous avons préféré simplifier le calcul en considérant la dissolution d'une définition locale comme étape de réduction.

Les règles NL-STAT et NL-DYN peuvent laisser penser que nous nous éloignons de manière significative de la nature asynchrone des messages dans le join calcul distribué, puisqu'elles activent une continuation d'un message. En fait, ces messages sont toujours asynchrones puisqu'il est impossible de détecter quand ils atteignent leur destination. Il est cependant très utile de pouvoir

$$v_1\left[\mathbf{n}\langle\rangle \triangleright V_1,\ c\left[\mathtt{nv}\langle b\rangle \triangleright \mathtt{go}\ b,\ a\,[\mathcal{D}:\mathcal{P}]^{\emptyset,\{\mathbf{n}\}}:\mathbf{0}\right]^{\emptyset,\{\mathbf{n}\}}:\mathbf{0}\right]^{\{\mathbf{n}\},\emptyset},\ v_2\left[\mathbf{n}\langle\rangle \triangleright V_2 : \underline{\mathtt{nv}\langle v_2\rangle}\right]^{\{\mathbf{n}\},\emptyset} \quad (4.1)$$

$$\textsc{Nl-Stat} \quad \rightarrow \quad v_1\left[\mathbf{n}\langle\rangle \triangleright V_1,\ c\left[\mathtt{nv}\langle b\rangle \triangleright \mathtt{go}\ b,\ a\,[\mathcal{D}:\mathcal{P}]^{\emptyset,\{\mathbf{n}\}}:\mathbf{0}\right]^{\emptyset,\{\mathbf{n}\}}:\mathbf{0}\right]^{\{\mathbf{n}\},\emptyset},\ v_2\left[\mathbf{n}\langle\rangle \triangleright V_2 : \underline{c.\mathtt{nv}\langle v_2\rangle}\right]^{\{\mathbf{n}\},\emptyset} \quad (4.2)$$

$$\textsc{Comm} \quad \rightarrow \quad v_1\left[\mathbf{n}\langle\rangle \triangleright V_1,\ c\left[\underline{\mathtt{nv}\langle b\rangle \triangleright \mathtt{go}\ b},\ a\,[\mathcal{D}:\mathcal{P}]^{\emptyset,\{\mathbf{n}\}} : \underline{c.\mathtt{nv}\langle v_2\rangle}\right]^{\emptyset,\{\mathbf{n}\}}:\mathbf{0}\right]^{\{\mathbf{n}\},\emptyset},\ v_2\,[\mathbf{n}\langle\rangle \triangleright V_2 : \mathbf{0}]^{\{\mathbf{n}\},\emptyset} \quad (4.3)$$

$$\textsc{Join} \quad \rightarrow \quad v_1\left[\mathbf{n}\langle\rangle \triangleright V_1,\ c\left[\mathtt{nv}\langle b\rangle \triangleright \mathtt{go}\ b,\ a\,[\mathcal{D}:\mathcal{P}]^{\emptyset,\{\mathbf{n}\}} : \underline{\mathtt{go}\ v_2}\right]^{\emptyset,\{\mathbf{n}\}}:\mathbf{0}\right]^{\{\mathbf{n}\},\emptyset},\ v_2\,[\mathbf{n}\langle\rangle \triangleright V_2 : \mathbf{0}]^{\{\mathbf{n}\},\emptyset} \quad (4.4)$$

$$\textsc{Go} \quad \rightarrow \quad v_1\,[\mathbf{n}\langle\rangle \triangleright V_1 : \mathbf{0}]^{\{\mathbf{n}\},\emptyset},\ v_2\left[\mathbf{n}\langle\rangle \triangleright V_2,\ c\left[\mathtt{nv}\langle b\rangle \triangleright \mathtt{go}\ b,\ a\,[\mathcal{D}:\mathcal{P}]^{\emptyset,\{\mathbf{n}\}}:\mathbf{0}\right]^{\emptyset,\{\mathbf{n}\}}:\mathbf{0}\right]^{\{\mathbf{n}\},\emptyset} \quad (4.5)$$

FIG. 4.7 – Mise à jour dynamique

détecter à quel moment la résolution du message à lieu, par exemple pour ne migrer qu'après la résolution du message (la migration modifiant la liaison d'un message dynamique).

Par exemple, un processus imprimant sur deux locations a et b, pouvant représenter par exemple deux machines, en utilisant un canal dynamique **print** pourrait être codé de la manière suivante :

$$\mathtt{go}\ a; \mathbf{print}\langle\text{“Hello”}\rangle; \mathtt{go}\ b; \mathbf{print}\langle\text{“World”}\rangle$$

En supposant qu'aucun autre processus de la location courante ne la fait migrer, ce processus créera successivement les messages résolus $a.\mathbf{print}\langle\text{“Hello”}\rangle$ et $b.\mathbf{print}\langle\text{“World”}\rangle$. Ces messages ne transportant pas de continuation explicite, il est impossible de détecter le moment où ils seront consommés. La seule garantie que nous ayons est que le premier message sera envoyé à la location a et le second à la location b.

Un critère important de notre système est de nous restreindre à des synchronisations locales. La fonction de résolution n'étant modifiée que par la règle TREE, cette fonction n'est calculée que pour des locations gelées. Ces locations n'ayant aucune sous-location depliée, le calcul est bien local à la location étant dépliée. Ses sous-locations hériteront de sa nouvelle fonction de résolution lorsqu'elles seront elles-mêmes dépliées.

Un autre critère important est la manière dont sont introduites des nouvelles définitions pour des noms dynamiques existant. La seconde condition de la règle DEF de la figure 4.5 impose que tous les noms définis introduits par la définition locale soient statiques. Ceci n'empêche pas l'introduction de définitions pour des noms dynamiques à partir du moment où ils sont à l'intérieur d'une location $a\,[D:P]^{\Delta,I}$ créée par la définition. Ainsi, le seul moyen d'introduire de nouvelles définitions pour des noms dynamiques est d'introduire également de nouvelles locations. Ce choix est justifié par le fait que la reliaison des noms dynamiques ne devrait être provoquée que par des migrations. Par conséquent, cette liaison ne devrait pas être modifiée lorsqu'aucune migration n'a lieu. La définition liée à un nom dynamique étant celle présente dans la plus proche location englobante définissant ce nom, nous devons donc interdire l'introduction de définitions dynamiques dans les locations dépliées. Nous remarquons de plus que si nous l'autorisions, il faudrait synchroniser toutes les sous-locations dépliées de la location active pour modifier leur fonction de résolution. Nous décidons ainsi d'encapsuler la portée initiale d'une nouvelle définition dynamique aux sous-locations de la nouvelle location la contenant.

Un autre choix de conception est la spécification explicite des noms qui sont importés par une location. Supposons que la migration vers une location a ne soit possible que lorsque les noms importés par la location migrant sont fournis par a (soit parce qu'ils sont définis par a, soit parce qu'ils sont importés par a). Une telle restriction pourrait être mise en place par une phase de typage statique, ou par une vérification dynamique au moment de la migration. Un premier point de vue consiste à choisir un ensemble de noms importés le plus grand possible, afin de les utiliser dans la location et de les rendre disponibles pour ses sous-locations : c'est le cas d'un *serveur*. Un autre point de vue, opposé au premier, consiste à minimiser cet ensemble afin d'être capable de migrer dans de nombreuses locations, puisque moins de noms doivent être disponibles. C'est le cas d'un *client* ou d'un *agent*. Ces deux points de vue étant en opposition, nous laissons la possibilité de définir cet ensemble de noms importés selon l'usage souhaité pour la location.

Notre premier exemple souligne l'interaction entre les définitions et les locations, et montre comment mettre à jour une définition utilisée par un client de manière transparente pour ce client. Plus précisément, nous modélisons un agent a qui utilise un canal dynamique $\mathbf{n}$. La version de $\mathbf{n}$ que a utilise est contrôlée par un gestionnaire de versions c. Nous supposons que a et c importent

seulement **n** et ne définissent aucun nom dynamique. Initialement, l'agent a utilise la version V_1 de la bibliothèque qui est définie dans la location v_1. Cette dernière location ne définit que le canal **n** et n'importe rien :

$$v_1 \left[\mathbf{n}\langle\rangle \triangleright V_1,\ c \left[\mathtt{nv}\langle b\rangle \triangleright \mathtt{go}\ b,\ a\,[\mathcal{D} : \mathcal{P}]^{\emptyset,\{\mathbf{n}\}} : \mathbf{0} \right]^{\emptyset,\{\mathbf{n}\}} : \mathbf{0} \right]^{\{\mathbf{n}\},\emptyset}$$

Si un bug est découvert dans V_1, il peut être corrigé en créant une nouvelle version de la bibliothèque V_2 dans la location v_2, et en demandant au gestionnaire de versions c d'utiliser la nouvelle version :

$$v_2\,[\mathbf{n}\langle\rangle \triangleright V_2 : \mathtt{nv}\langle v_2\rangle]^{\{\mathbf{n}\},\emptyset}$$

La figure 4.7 détaille la série d'étapes de réduction implémentant le changement de version. Nous ne détaillons pas les dépliages et repliages de locations (règle Tree). La configuration initiale est (4.1). Tout d'abord, le message statique $\mathtt{nv}\langle v_2\rangle$ est résolu grâce à la règle NL-Stat, donnant la configuration (4.2). Puis le message résolu $c.\mathtt{nv}\langle v_2\rangle$ est envoyé en c par la règle Comm, donnant la configuration (4.3). Ce message active alors le filtre de l'unique règle de réaction de c par la règle Join, ce qui crée un processus `go` v_2 dans la configuration (4.4). Enfin, la migration de c dans v_2 a lieu par la règle Go pour donner l'état final (4.5). L'agent a utilise désormais la nouvelle version de la bibliothèque, sans qu'il n'ait été nécessaire de l'interrompre ou de demander sa collaboration.

Un deuxième exemple montre qu'il est possible d'avoir des canaux statiques et dynamiques dans un même filtre. Nous modélisons ici l'exemple du canal **print** de l'introduction. Voici une première version pour les imprimantes noir et blanc (le canal `p` prend comme arguments une couleur et une chaîne de caractères, et imprime cette dernière avec la couleur spécifiée sur l'imprimante) :

$$s_1\,[\mathbf{print}\langle s\rangle \triangleright \mathtt{p}\langle \text{“black”}, s\rangle : \mathbf{0}]^{\{\mathbf{print}\},\emptyset}$$

Une deuxième version plus évoluée pour les imprimantes couleur peut autoriser le changement de la couleur d'impression. Nous implémentons ceci en utilisant une simple cellule de référence `color` :

$$s_2 \left[\begin{array}{l} \mathbf{print}\langle s\rangle \mid \mathtt{color}\langle c\rangle \triangleright \mathtt{p}\langle c, s\rangle \mid \mathtt{color}\langle c\rangle, \\ \mathbf{setcolor}\langle c', \kappa\rangle \mid \mathtt{color}\langle c\rangle \triangleright \mathtt{color}\langle c'\rangle \mid \kappa\langle\rangle \end{array} : \mathtt{color}\langle \text{“black”}\rangle \right]^{\{\mathbf{print},\mathbf{setcolor}\},\emptyset}$$

Nous remarquons qu'il est intéressant que le canal `color` soit statique, puisque ainsi il n'est pas exposé en tant que canal dynamique défini dans s_2.

Un agent utilisant la deuxième version peut être écrit de la manière suivante :

$$agent \left[\begin{array}{ll} \mathtt{hi}\langle a\rangle \triangleright & \mathtt{def}\ \kappa\langle\rangle \triangleright \mathbf{print}\langle \text{“hello”}\rangle\ \mathtt{in} \\ & \mathtt{go}\ a; \mathbf{setcolor}\langle \text{“red”}, \kappa\rangle \end{array} : \mathbf{0} \right]^{\emptyset,\{\mathbf{print},\mathbf{setcolor}\}}$$

L'envoi du message $\mathtt{hi}\langle s_2\rangle$ donnerait bien le comportement attendu. Par contre, si par erreur (la malédiction du copier-coller par exemple) le message envoyé est $\mathtt{hi}\langle s_1\rangle$, le message généré ensuite sur le canal **setcolor** bloquerait le reste de l'exécution, puisque ce canal dynamique n'est pas disponible dans s_1. Nous prolongerons cet exemple dans la section 4.5, où nous présentons un système de types permettant de garantir dans ce cas particulier la présence d'une définition pour **setcolor**.

Un troisième exemple montre l'utilité de la création de nouveaux noms dynamiques lors de l'exécution, de l'introduction de nouvelles définitions dynamiques, ainsi que des noms dynamiques en tant que valeurs de première classe. Soient $\mathtt{reg}_1, \ldots, \mathtt{reg}_n$ des canaux pouvant être utilisés pour lier un nom de canal dynamique passé en argument à des services $P_1, \ldots, P_n$:

$$\mathtt{reg}_i\langle n, \kappa\rangle \triangleright \mathtt{def}\ a\,[n\langle\rangle \triangleright P_i : \mathbf{0}]^{\{n\},\emptyset}\ \mathtt{in}\ \kappa\langle a\rangle$$

De tels canaux pourraient par exemple être utilisés pour lier des processus de facturation à un numéro privé donné. La facturation pouvant être décentralisée dans un réseau distribué, chaque location représentant un nœud du réseau peut posséder son propre processus de facturation associé au numéro privé.

Le processus décrit ci-après crée un nouveau numéro sous la forme d'un canal dynamique **n**, et associe ce numéro aux services de facturation locaux en utilisant les canaux $\mathtt{reg}_1, \ldots, \mathtt{reg}_n$. Les serveurs de facturation créés portent les noms $a_1, \ldots, a_n$.

$$\begin{array}{ll} \nu\mathbf{n}. & \mathtt{def}\ \kappa_1\langle a_1\rangle \triangleright \ldots\ \triangleright \mathtt{def}\ \kappa_n\langle a_n\rangle \triangleright Q \\ & \mathtt{in}\ \mathtt{reg}_n\langle \mathbf{n}, \kappa_n\rangle \ldots\ \mathtt{in}\ \mathtt{reg}_1\langle \mathbf{n}, \kappa_1\rangle \end{array}$$

τ	::=		type monomorphe
		$\widetilde{\tau}$	nuplet
	|	$\langle\tau\rangle_w^+$	canal redéfinissable
	|	$\langle\tau\rangle_{\mathbf{\Delta}}$	canal
	|	$loc(\mathbf{\Delta})$	location
	|	α	variable de type
w	::=		type de nom dynamique
		$\mathbf{n}$	nom de canal dynamique
	|	δ	variable de type de nom
σ	::=		type polymorphe
		$\forall\widetilde{\alpha}\widetilde{\delta}.\tau$	schéma de type
$\mathbf{\Delta}, \mathbf{I}$	::=		ensemble de types de noms dynamiques
		$\emptyset$	ensemble vide
	|	$\{w\}$	singleton
	|	$\mathbf{\Delta} \cup \mathbf{\Delta}$	union

FIG. 4.8 – Types

Le processus Q peut ensuite librement se déplacer d'un serveur à l'autre et utiliser son numéro privé $\mathbf{n}$, ce indépendemment du processus de facturation localement utilisé.

Nous soulignons que dans cet exemple les canaux dynamiques sont des valeurs de première classe qui peuvent être redéfinis et qui peuvent être créés lors de l'exécution du programme grâce à l'opérateur $\nu\mathbf{n}$.

Comme dans l'exemple précédent, si le processus Q migre par erreur dans une location où $\mathbf{n}$ n'est pas disponible, tout envoi de message sur ce canal sera bloquant jusqu'à ce que Q migre à nouveau vers un des a_i. Notre système de types nous garantit que cette situation ne peut survenir.

4.4 Liaison dynamique sûre

4.4.1 Le système de types du join calcul dynamique

Nous introduisons un système de types statique qui n'autorise que les configurations dont les messages dynamiques sont tous liés à une définition, présente par conséquent dans une location englobante. Les types sont définis dans la figure 4.8.

Ce système de types est très proche de celui du join calcul distribué. Il utilise le même critère de généralisation que celui implémenté dans JoCaml et formalisé dans [FMLR00].

Types et sous-typage (fig. 4.8 et 4.9)

De manière intuitive, les locations rendent disponibles des noms dynamiques, soit en les définissant directement, soit en demandant aux locations englobantes de les fournir (en les important). Le type d'une location est $loc(\mathbf{\Delta})$, où $\mathbf{\Delta}$ est l'ensemble des noms dynamiques qui sont disponibles dans cette location. Par conséquent, tout message envoyé dans cette location sur un des noms de $\mathbf{\Delta}$ est correct, alors que les messages envoyés sur d'autres noms dynamiques ne doivent pas être considérés comme correctement typés. Cet ensemble de noms dynamiques peut également contenir des variables de types de noms δ. On trouve ces variables de types de noms dans les types des processus gardés par un filtre. Ces variables représentent un nom de canal dynamique qui n'est pas connu, puisqu'il dépend des arguments des messages déclenchant le filtre, de la même manière que les variables y représentent des noms de canaux ou de locations.

Le type d'un canal dynamique reflète les noms dynamiques dont le canal a besoin. Par exemple, un message sur un canal de type $\langle\tau\rangle_{\mathbf{\Delta}}$ transporte un argument de type τ et requiert que des définitions pour les noms de $\mathbf{\Delta}$ soient disponibles. Plus précisément, un canal dynamique $\mathbf{n}$ qui ne transporte aucun argument à pour type $\langle\rangle_{\{\mathbf{n}\}}$, puisque qu'un message sur un tel canal a besoin qu'une définition de $\mathbf{n}$ soit disponible. Les canaux statiques n'ayant besoin d'aucune définition dynamique, leur type est de la forme $\langle\tau\rangle_\emptyset$ qui sera par la suite noté $\langle\tau\rangle$.

Les canaux étant des valeurs de première classe, il est tout à fait possible d'écrire la règle de réaction $\texttt{send}\langle x\rangle \triangleright x\langle\rangle$, qui reçoit un nom et envoie un message sur ce nom. Toute utilisation de `send` avec un nom statique est correcte. Par contre, il n'est possible d'utiliser `send` avec un nom dynamique que si celui-ci est défini dans une location englobante. Cette contrainte est représentée en donnant à l'argument de `send` un type $\langle\rangle_{\mathbf{\Delta}}$ si $\mathbf{\Delta}$ est l'ensemble des noms dynamiques disponibles

$$\frac{}{\langle\tau\rangle_w^+ \leq \langle\tau\rangle_{\{w\}}}\ [\textsc{Plus}] \qquad \frac{\tau' \leq \tau \qquad \mathbf{\Delta} \subseteq \mathbf{\Delta'}}{\langle\tau\rangle_{\mathbf{\Delta}} \leq \langle\tau'\rangle_{\mathbf{\Delta'}}}\ [\text{N-}\textsc{Sub}] \qquad \frac{\mathbf{\Delta'} \subseteq \mathbf{\Delta}}{loc(\mathbf{\Delta}) \leq loc(\mathbf{\Delta'})}\ [\text{L-}\textsc{Sub}]$$

$$\frac{\tau_i \leq \tau_i' \text{ pour } i \in [1..n]}{\tau_1, \ldots, \tau_n \leq \tau_1' \ldots \tau_n'}\ [\text{T-}\textsc{Sub}]$$

FIG. 4.9 – Règles de sous-typage

dans la location courante. Par conséquent le canal `send` a pour type $\langle\langle\rangle_{\mathbf{\Delta}}\rangle$. L'ensemble $\mathbf{\Delta}$ étant une borne supérieure des noms dynamiques pouvant être utilisés, nous avons une notion de sous-typage (notée $\leq$) sur les types des canaux : $\langle\tau\rangle_{\mathbf{\Delta'}} \leq \langle\tau\rangle_{\mathbf{\Delta}}$ si $\mathbf{\Delta'} \subseteq \mathbf{\Delta}$ (un canal qui utilise potentiellement moins de définitions dynamiques qu'un autre est un sous-type de ce deuxième canal). Ainsi un canal statique a un type qui est un sous-type de tous les canaux dynamiques transportant des arguments de même type. La règle de sous-typage N-SUB de la figure 4.9 autorise également le sous-typage contravariant des types des arguments du canal. La règle de sous-typage des nuplets de types est bien entendu covariante (règle T-SUB).

Une location fournissant plus de définitions dynamiques peut être utilisée à la place d'une location en fournissant moins. Nous formalisons cette notion par la règle de sous-typage sur les types des locations L-SUB.

Un critère crucial pour garantir la correction de notre système de types est d'interdire le sous-typage pour les canaux dynamiques qui sont redéfinis. Nous notons $\langle\tau\rangle_w^+$ le type de ces canaux avec w représentant le nom dynamique du canal. Ces types n'ont aucun sous-type. Un canal redéfinissable pouvant être utilisé comme un canal dynamique pour envoyer des messages, nous introduisons la règle de sous-typage PLUS. Par la suite, nous notons $\leq$ comme étant la plus petite relation réflexive et transitive engendrée par les règles de la figure 4.9.

Un schéma de type $\forall\widetilde{\alpha}\widetilde{\delta}.\tau$ comporte un ensemble de variables de types généralisées $\widetilde{\alpha}$, un ensemble de variables de types de noms généralisées $\widetilde{\delta}$ et un type monomorphe τ. Une instanciation d'un tel schéma, notée $Inst(\forall\widetilde{\alpha}\widetilde{\delta}.\tau)$, est un type $\tau\theta$ où θ est une substitution des variables de types $\widetilde{\alpha}$ vers les types monomorphes et des variables de types de noms $\widetilde{\delta}$ vers les noms de canaux dynamiques.

Pour garantir la correction du système de types, les types de canaux redéfinissables polymorphes sont interdits. Nous interdisons aussi les types de locations polymorphes. Tous les autres types sont dits *bien formés*.

Un *environnement de typage monomorphe* B est une liste d'associations entre des noms et des types monomorphes, où chaque nom est associé au plus une fois. Un *environnement de typage* A ou Γ est une liste d'associations entre des noms et des schémas de types, où chaque nom est associé au plus une fois.

Dans la suite, nous notons $ftv(\tau)$ les variables de types libres de τ, $fnv(\tau)$ les variables de types de noms libres de τ définis comme suit :

$$\begin{array}{rcl@{\quad\text{et}\quad}rcl}
ftv(\tau_1,\ldots,\tau_n) & = & ftv(\tau_1)\cup\ldots\cup ftv(\tau_n) & fnv(\tau_1,\ldots,\tau_n) & = & fnv(\tau_1)\cup\ldots\cup fnv(\tau_n)\\
ftv(\langle\tau\rangle_w^+) & = & ftv(\tau) & fnv(\langle\tau\rangle_w^+) & = & fnv(\tau)\cup fnv(\{w\})\\
ftv(\langle\tau\rangle_{\mathbf{\Delta}}) & = & ftv(\tau) & fnv(\langle\tau\rangle_{\mathbf{\Delta}}) & = & fnv(\tau)\cup fnv(\mathbf{\Delta})\\
ftv(loc(\mathbf{\Delta})) & = & \emptyset & fnv(loc(\mathbf{\Delta})) & = & fnv(\mathbf{\Delta})\\
ftv(\alpha) & = & \{\alpha\} & fnv(\alpha) & = & \emptyset\\
ftv(\emptyset) & = & \emptyset & fnv(\emptyset) & = & \emptyset\\
ftv(\{\mathbf{n}\}) & = & \emptyset & fnv(\{\mathbf{n}\}) & = & \emptyset\\
ftv(\{\delta\}) & = & \emptyset & fnv(\{\delta\}) & = & \{\delta\}\\
ftv(\mathbf{\Delta}\cup\mathbf{\Delta'}) & = & \emptyset & fnv(\mathbf{\Delta}\cup\mathbf{\Delta'}) & = & fnv(\mathbf{\Delta})\cup fnv(\mathbf{\Delta'})
\end{array}$$

Nous notons $fv(\tau)$ pour $ftv(\tau)\cup fnv(\tau)$.

Nous étendons ftv et fnv aux environnements de typage monomorphes et aux environnements de typage de la manière suivante :

$$ftv(B) \stackrel{\text{def}}{=} \bigcup_{m:\tau\in B} ftv(\tau) \qquad ftv(A) \stackrel{\text{def}}{=} \bigcup_{m:\forall\widetilde{\alpha}\widetilde{\delta}.\tau\in A} ftv(\tau)\setminus\widetilde{\alpha}$$

$$fnv(B) \stackrel{\text{def}}{=} \bigcup_{m:\tau\in B} fnv(\tau) \qquad fnv(A) \stackrel{\text{def}}{=} \bigcup_{m:\forall\widetilde{\alpha}\widetilde{\delta}.\tau\in A} fnv(\tau)\setminus\widetilde{\delta}$$

$$\frac{m : \forall \widetilde{\alpha}\widetilde{\delta}.\tau' \in \Gamma \qquad \tau = Inst(\forall \widetilde{\alpha}\widetilde{\delta}.\tau')}{\Gamma \vdash m : \tau} \text{ [Name]} \qquad \frac{\Gamma \vdash m : \tau \qquad \tau \leq \tau'}{\Gamma \vdash m : \tau'} \text{ [Sub]}$$

$$\frac{\Gamma \vdash m_i : \tau_i \text{ pour } i \in [1..n]}{\Gamma \vdash m_1, \ldots, m_n : \tau_1, \ldots, \tau_n} \text{ [Tuple]}$$

Fig. 4.10 – Règles de typage pour les noms

Généralisation Plusieurs règles de typage utilisent un opérateur de *généralisation* $Gen(B, \Lambda, \Theta)$ défini comme suit :

$$Gen(B, \Lambda, \Theta) = \bigcup_{m:\tau \in B} \{m : \forall \widetilde{\alpha}\widetilde{\delta}.\tau\}$$

avec $\widetilde{\alpha} = ftv(\tau) \setminus (\Lambda \cup \Theta)$ et $\widetilde{\delta} = fnv(\tau) \setminus (\Lambda \cup \Theta)$.

L'ensemble Λ contient les variables de types et les variables de types de noms libres dans l'environnement de typage (dans la règle de typage Def, Λ est $fv(\Gamma)$) ; l'ensemble Θ comprend les variables de types et les variables de types de noms qui ne peuvent être généralisées parce qu'elles sont partagées dans un filtre (comme dans [FMLR00]).

Jugements de typage Un jugement de typage a l'une des formes suivantes :

$\Gamma \vdash \widetilde{n} : \widetilde{\tau}$	nuplet de noms
$\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash P$	processus
$\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D} :: B; \Delta_1; \Theta$	définition
$\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D} : \Delta_1$	définition d'une location dépliée
$\Gamma \vdash \mathcal{S}$	configuration

où :

- Γ est l'environnement de typage ;
- $\mathbf{\Delta}$ est l'ensemble de types de noms dynamiques correspondant aux noms dynamiques définis dans la location courante ;
- $\mathbf{I}$ est l'ensemble de types de noms dynamiques correspondant aux noms dynamiques importés par la location courante ;
- B est l'environnement de typage monomorphe rassemblant les types des noms statiques (canaux et locations) définis dans $\mathcal{D}$;
- Δ_1 est l'ensemble de noms dynamiques ou de variables, rassemblant les canaux dynamiques définis localement par $\mathcal{D}$ (sans considérer les sous-locations de $\mathcal{D}$) ;
- Θ est un ensemble de variables de types et de variables de types de noms qui ne peuvent être généralisées parce qu'elles sont partagées.

Les quatre règles de typage principales qui garantissent la présence d'une définition dynamique pour tout message dynamique sont les règles Msg, Loc, Soup-Loc et Go. La règle Msg vérifie que les noms potentiellement utilisés par le canal sur lequel le message est envoyé sont inclus dans l'ensemble des noms localement disponibles (qu'ils soient définis localement ($\mathbf{\Delta}$) ou qu'ils soient importés ($\mathbf{I}$)). Les règles Loc et Soup-Loc se ressemblent beaucoup, la première étant un peu plus complexe puisqu'elle peut être utilisée lors du typage du processus gardé d'une règle de réaction. Ces règles vérifient que les noms dynamiques qui sont déclarés comme étant définis localement le sont bien, et que les noms importés sont disponibles dans la location englobante (dans le cas de la règle Soup-Loc, cette vérification a lieu dans la règle Configuration). Le contenu de la location est typé dans un environnement où les noms définis et les noms importés correspondent à ceux de la location (en effet, ces règles modifient $\mathbf{\Delta}$ et $\mathbf{I}$ dans la partie gauche du jugement de typage). La règle Go vérifie que la location ciblée par la migration fournit au moins les canaux dynamiques importés par la location migrant.

Règles de typage pour les noms (fig. 4.10) Ces règles sont les règles de typage habituelles pour les noms.

Règles de typage pour les processus (fig. 4.11) Les deux règles de typage pour les messages Msg et R-Msg sont très semblables. Elles correspondent aux deux états qu'un message peut prendre avant ou après l'étape de résolution. Dans ces règles, le nom sur lequel le message est envoyé doit satisfaire le jugement de typage suivant : $\Gamma \vdash n : \langle \widetilde{\tau} \rangle_{\mathbf{\Delta} \cup \mathbf{I}}$. Par définition du sous-typage

$$\frac{\Gamma \vdash n : \langle \widetilde{\tau} \rangle_{\mathbf{\Delta} \cup \mathbf{I}} \qquad \Gamma \vdash \widetilde{m} : \widetilde{\tau} \qquad \mathbf{\Delta}; \mathbf{I}; \Gamma \vdash P}{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash n\langle \widetilde{m} \rangle; P} \text{ [Msg]}$$

$$\frac{\Gamma \vdash n : \langle \widetilde{\tau} \rangle_{\mathbf{\Delta} \cup \mathbf{I}} \qquad \Gamma \vdash \widetilde{m} : \widetilde{\tau} \qquad \Gamma \vdash a : \mathit{loc}(\emptyset) \qquad n \in \mathit{dln}(a)}{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash a.n\langle \widetilde{m} \rangle} \text{ [R-Msg]}$$

$$\frac{\mathbf{\Delta}; \mathbf{I}; \Gamma + A \vdash D :: B; \emptyset; \Theta \qquad A = \mathit{Gen}(B, \mathit{fv}(\Gamma), \Theta) \qquad \mathbf{\Delta}; \mathbf{I}; \Gamma + A \vdash P \qquad \mathit{dom}(A) \cap \mathit{dom}(\Gamma) = \emptyset}{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathtt{def}\ D\ \mathtt{in}\ P} \text{ [Def]}$$

$$\frac{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{P}_1 \qquad \mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{P}_2}{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{P}_1 \mid \mathcal{P}_2} \text{ [Par]} \qquad \frac{\Gamma \vdash n : \mathit{loc}(\mathbf{I}) \qquad \mathbf{\Delta}; \mathbf{I}; \Gamma \vdash P}{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathtt{go}\ n; P} \text{ [Go]}$$

$$\frac{\mathbf{d} \notin \mathit{fn}(\Gamma) \qquad \mathbf{\Delta}; \mathbf{I}; \Gamma + \mathbf{d} : \langle \tau \rangle^{+}_{\mathbf{d}} \vdash P}{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \nu \mathbf{d}.P} \text{ [Nu]} \qquad \frac{}{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathbf{0}} \text{ [Nil]}$$

Fig. 4.11 – Règles de typage pour les processus

sur les noms de canaux, ce jugement donne une borne supérieure à l'ensemble des canaux que le message peut utiliser, donc à l'ensemble des définitions dynamiques utilisées. Cette borne supérieure est tout simplement l'ensemble des définitions dynamiques disponibles dans la location courante. Par conséquent, ces règles s'assurent que tout message dynamique est bien lié à une définition. La règle R-Msg vérifie également que la location spécifiée dans le préfixe du message est présente dans l'environnement de typage Γ avec un type de location (nous remarquons que par définition du sous-typage, nous avons $\mathit{loc}(\mathbf{\Delta}) \leq \mathit{loc}(\emptyset)$ pour tout $\mathbf{\Delta}$), et que le canal sur lequel le message est envoyé est bien défini dans cette location. Nous remarquons que pour un message résolu il n'est plus nécessaire de vérifier qu'il est bien lié à une définition disponible localement, puisqu'il a été résolu. Nous pourrions alors typer le canal n par un jugement de la forme $\Gamma \vdash n : \langle \widetilde{\tau} \rangle_{\top}$, en disant que tout ensemble de types de noms dynamiques est inclus dans $\top$. Ceci n'est pas nécessaire puisque les messages résolus ne sont présents qu'en contexte d'évaluation (c'est à dire dans la partie processus d'une location dépliée ou dépliable par équivalence structurelle), ce qui permet de toujours typer ce message avec le jugement plus restrictif $\Gamma \vdash n : \langle \widetilde{\tau} \rangle_{\mathbf{\Delta} \cup \mathbf{I}}$ (voir les cas NL-Dyn, NL-Stat et Comm dans la preuve du théorème 4, page 148).

La règle de typage Def vérifie qu'aucun nom dynamique n'est localement défini dans D ($\Delta_1 = \emptyset$). Elle vérifie aussi que les noms statiques définis de D, qui sont également le domaine des environnements B et A, ne sont pas présent dans l'environnement Γ (par définition des types, les seuls noms statiques qui sont présents dans Γ sont des noms du domaine de Γ). Nous remarquons que nous utilisons la récursion polymorphe dans cette règle de typage, ce qui simplifie grandement la preuve de la stabilité de la typabilité par réduction (nous détaillons cette remarque en section 4.4.2).

La règle de typage Go vérifie que la location destination peut avoir pour type $\mathit{loc}(\mathbf{I})$, où $\mathbf{I}$ est l'ensemble des noms dynamiques importés par la location courante. Par définition du sous-typage pour les types des locations, cet ensemble est en fait une borne inférieure. Toute location fournissant plus de définitions dynamiques peut donc être la cible de cette migration.

La règle Nu introduit un nouveau nom de canal dynamique. Ce canal est monomorphe, redéfinissable, et a le type d'un canal envoyant des messages sur son propre nom. Les types des canaux redéfinissables ne possédant pas de sous-types, toute définition ultérieure de ce canal doit avoir exactement ce type.

Règles de typage pour les définitions (fig. 4.12) La règle de typage Join est utilisée pour typer une règle de réaction. Les noms définis du filtre sont partitionnés en deux ensembles : les noms statiques et les noms dynamiques. Les noms statiques $\mathtt{x_i}$ ont pour type $\langle \widetilde{\tau_i} \rangle$, sont inclus dans l'environnement B, et doivent être présent dans Γ pour des raisons techniques. Les noms dynamiques doivent aussi être présent dans Γ avec un type redéfinissable $\langle \widetilde{\tau_j} \rangle^{+}_{w_j}$, puisqu'ils sont redéfinis. Nous ne notons pas ces noms dynamiques $\mathbf{m}_j$ puisqu'ils peuvent également être des variables liées par un filtre (si la règle de réaction en cours de typage fait partie du processus gardé d'une autre règle de réaction). Par contre, en contexte d'évaluation, nous aurons toujours $m_j = w_j = \mathbf{m}_j$. Tous les noms dynamiques (ou variables) sont inclus dans l'ensemble des noms dynamiques locaux. Cette règle s'assure également que les noms reçus ne sont pas présents dans Γ. Enfin, il est vérifié que toute variable de type ou variable de type de nom partagée dans le type de deux noms définis du filtre est bien présente dans Θ, puisque de telles variables ne peuvent être

$$\frac{\begin{array}{c}\mathbf{\Delta};\mathbf{I};\Gamma+(\widetilde{u_i}:\widetilde{\tau_i})^i+(\widetilde{y_j}:\widetilde{\tau_j})^j\vdash P \qquad \Gamma\vdash \mathtt{x_i}:\langle\widetilde{\tau_i}\rangle \qquad \Gamma\vdash m_j:\langle\widetilde{\tau_j}\rangle^+_{w_j} \qquad \left(\bigcup_i \widetilde{u_i}\cup\bigcup_j \widetilde{y_j}\right)\cap dom(\Gamma)=\emptyset \\ \forall(n:\tau),(n':\tau')\in\left(\{\mathtt{x_i}:\langle\widetilde{\tau_i}\rangle\}\cup\{m_j:\langle\widetilde{\tau_j}\rangle^+_{w_j}\}\right)\ .\ n\neq n' \implies fv(\tau)\cap fv(\tau')\subseteq\Theta\end{array}}{\mathbf{\Delta};\mathbf{I};\Gamma\vdash(\mathtt{x_i}\langle\widetilde{u_i}\rangle)^i\mid(m_j\langle\widetilde{y_j}\rangle)^j\triangleright P::(\mathtt{x_i}:\langle\widetilde{\tau_i}\rangle)^i;\bigcup_j\{m_j\};\Theta}\ [\textsc{Join}]$$

$$\frac{}{\mathbf{\Delta};\mathbf{I};\Gamma\vdash\top::\emptyset;\emptyset;\Theta}\ [\textsc{Top}] \qquad \frac{\mathbf{\Delta};\mathbf{I};\Gamma\vdash\mathcal{D}_1::B_1;\Delta_1;\Theta \qquad \mathbf{\Delta};\mathbf{I};\Gamma\vdash\mathcal{D}_2::B_2;\Delta_2;\Theta}{\mathbf{\Delta};\mathbf{I};\Gamma\vdash\mathcal{D}_1,\mathcal{D}_2::B_1\oplus B_2;\Delta_1\cup\Delta_2;\Theta}\ [\textsc{And}]$$

$$\frac{\begin{array}{c}\forall m_i\in\Delta\ .\ m_i:\langle\tau_i\rangle^+_{w_i}\in\Gamma \qquad \forall n_j\in I\ .\ n_j:\langle\tau_j\rangle^+_{w'_j}\in\Gamma \qquad \mathbf{\Delta}'=\bigcup_i w_i \quad \mathbf{I}'=\bigcup_j w'_j \\ \mathbf{\Delta}';\mathbf{I}';\Gamma\vdash\mathcal{D}::B;\Delta;\Theta \qquad \mathbf{\Delta}';\mathbf{I}';\Gamma\vdash\mathcal{P} \qquad \Gamma\vdash a:loc(\mathbf{\Delta}'\cup\mathbf{I}') \qquad \mathbf{I}'\subseteq(\mathbf{\Delta}''\cup\mathbf{I}'')\end{array}}{\mathbf{\Delta}'';\mathbf{I}'';\Gamma\vdash a\,[\mathcal{D}:\mathcal{P}]^{\Delta,I}::B+a:loc(\mathbf{\Delta}'\cup\mathbf{I}');\emptyset;\Theta}\ [\textsc{Loc}]$$

FIG. 4.12 – Règles de typage pour les définitions

$$\frac{\begin{array}{c}\{\alpha_i\}\text{ forme un arbre ayant comme racine }b \qquad \forall\beta a\in\{\alpha_i\}.I_{\beta a}\subseteq\Delta_\beta\cup I_\beta \\ I_b=\emptyset \qquad \forall\beta a\in\{\alpha_i\}.F_{\beta a}=Lookup(F_\beta,I_{\beta a},\Delta_{\beta a},a) \qquad (\Gamma\vdash\alpha_i\,[\mathcal{D}_i:\mathcal{P}_i]^{\Delta_{\alpha_i},I_{\alpha_i},F_{\alpha_i}})^i\end{array}}{\Gamma\vdash\prod_i(\alpha_i\,[\mathcal{D}_i:\mathcal{P}_i]^{\Delta_{\alpha_i},I_{\alpha_i},F_{\alpha_i}})^i}\ [\textsc{Configuration}]$$

$$\frac{n\in\Delta\implies n=\mathbf{n}\ \wedge\ \mathbf{n}:\langle\tau\rangle^+_{\mathbf{n}}\in\Gamma \qquad a:loc(\Delta\cup I)\in\Gamma \qquad \Delta;I;\Gamma\vdash\mathcal{P} \qquad \Delta;I;\Gamma\vdash\mathcal{D}:\Delta}{\Gamma\vdash\alpha a\,[\mathcal{D}:\mathcal{P}]^{\Delta,I,F}}\ [\textsc{Soup-Loc}]$$

$$\frac{\mathbf{\Delta};\mathbf{I};\Gamma\vdash\mathcal{D}::B;\Delta_\mathcal{D};\Theta \qquad A=Gen(B,fv(\Gamma),\Theta) \qquad A\subseteq\Gamma}{\mathbf{\Delta};\mathbf{I};\Gamma\vdash\mathcal{D}:\Delta_\mathcal{D}}\ [\textsc{Chem-Def}]$$

FIG. 4.13 – Règles de typage pour les configurations

généralisées (comme dans [FMLR00]).

La règle de typage AND utilise l'opérateur $\oplus$ dans $B_1\oplus B_2$, qui impose à tous les noms étant à la fois dans le domaine de B_1 et le domaine de B_2 d'avoir le même type associé dans les deux environnements.

La règle de typage LOC extrait de l'environnement de typage Γ les types associés aux noms spécifiés par Δ et I, afin de construire l'ensemble des types de noms dynamiques correspondant. Comme pour la règle JOIN, en contexte d'évaluation les ensembles Δ et $\mathbf{\Delta}'$ sont identiques (ainsi que I et $\mathbf{I}'$). Par contre, si la location typée fait partie du processus gardé d'une règle de réaction englobante, ces ensembles peuvent être différents si la règle de réaction reçoit en argument un nom dynamique (représenté dans le processus gardé par une variable) qui est redéfini dans la location en cours de typage. Le typage de la définition $\mathcal{D}$ doit posséder comme ensemble de noms dynamiques localement définis Δ le même ensemble que celui déclaré par la location. Le typage de $\mathcal{D}$ et $\mathcal{P}$ se fait dans un environnement où l'on peut utiliser les canaux définis et importés par la location. La règle de typage vérifie également que les noms importés sont bien disponibles dans la location englobante. Dans la conclusion de la règle, l'ensemble des noms dynamiques définis localement est vide puisque tous les noms dynamiques définis de la définition sont définis dans une sous-location.

Règles de typage pour les configurations (fig. 4.13) La règle de typage CONFIGURATION vérifie que les locations dépliées forment un arbre, qu'elles n'importent que des noms qui sont disponibles dans la location englobante (la location racine n'important aucun nom), que toutes les fonctions de résolution sont correctes, et que toutes les locations dépliées sont bien typées.

La règle SOUP-LOC est utilisée pour typer une location dépliée. Elle vérifie tout d'abord que tous les noms spécifiés dans Δ et I sont des canaux dynamiques (et non pas des variables), et que le type de chacun de ces canaux est le type d'un canal redéfinissable envoyant des messages sur son propre nom. La définition et le processus actif de la location sont typés dans un environnement où les canaux dynamiques définis et importés sont disponibles. Cette règle est plus simple que la règle LOC puisque les locations dépliées sont nécessairement en contexte d'évaluation. Comme pour la règle LOC, le typage de la définition doit posséder comme ensemble de noms dynamiques définis localement le même ensemble que les noms spécifiés par Δ, et la location a doit être présente dans Γ avec le type d'une location fournissant $\Delta\cup I$.

La règle de typage CHEM-DEF est utilisée pour typer la définition d'une location dépliée. Elle type la définition et vérifie que la généralisation de B est incluse dans l'environnement Γ.

Dans la suite, nous nous restreignons aux jugements de typage tels que si le jugement a la forme $\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \ldots$ alors nous avons nécessairement $fnv(\mathbf{\Delta} \cup \mathbf{I}) \subseteq fnv(\Gamma)$ et $fn(\mathbf{\Delta} \cup \mathbf{I}) \subseteq fn(\Gamma)$. On démontre facilement que pour toute dérivation de typage finie, si la propriété est vraie pour le jugement final, alors cette propriété est vraie pour tous les jugements de la dérivation (les seuls cas à vérifier sont les règles LOC et SOUP-LOC en utilisant le lemme B.1.4).

4.4.2 Le typage du join calcul distribué

Nous motivons dans cette section le choix de l'utilisation de la récursion polymorphe dans notre système de types.

Un aspect subtil du typage du join calcul distribué est l'interaction entre les définitions mutuellement récursives et le polymorphisme lorsque l'on cherche à la fois à satisfaire la stabilité de la typabilité par réduction et l'inférence des types. Supposons que nous typons une définition `def` D `in` P. Les définitions mutuellement récursives étant autorisées dans D, on type D dans un environnement étendu de manière monomorphe (afin d'éviter le problème de la récursion polymorphe lors de l'inférence) :

$$\Gamma + B \vdash D :: B$$

Le processus gardé P est lui par-contre typé dans l'environnement étendu de manière polymorphe :

$$\Gamma + A \vdash P$$

avec A étant la généralisation de B.

Par définition des réductions, deux définitions non mutuellement récursives peuvent être associées après réduction :

$$\alpha\,[\top : \texttt{def}\ D_1\ \texttt{in}\ \texttt{def}\ D_2\ \texttt{in}\ P] \rightarrow^* \alpha\,[D_1, D_2 : P]$$

Dans le cas où D_2 utilise D_1 de manière polymorphe, ce qui est tout à fait correct, la configuration finale n'est plus typable si le système de type considère que D_1 et D_2 sont mutuellement récursives. La solution proposé en [Fou98] pour le join calcul non distribué consiste à conserver la structure des définitions récursives, notées $D_1 \wedge D_2$, afin de ne pas les confondre avec des définitions ayant été assemblées par dissolution de `def`.

Cette solution ne peut pas s'appliquer au cas distribué, puisque les locations sont des définitions qui peuvent être séparées alors qu'elles sont toujours mutuellement récursives :

$$\texttt{def}\ a\,[n\langle\rangle \triangleright m\langle\rangle : \texttt{go}\ far] \wedge b\,[m\langle\rangle \triangleright n\langle\rangle : \texttt{go}\ away]\ \texttt{in}\ P$$

Deux classes de solutions sont alors envisageables. La première consiste à conserver la structure récursive des définitions, par exemple en utilisant des annotations. Cette structure spécifie alors l'ordre dans lequel il est nécessaire de typer les définitions. L'inconvénient de cette approche est que le typage n'est plus dirigé par la syntaxe, mais par ces annotations.

Une deuxième solution consiste à prouver la propriété de stabilité de la typabilité par réduction dans un système plus riche qui autorise la récursion polymorphe, alors que l'inférence de types n'autorise que la récursion monomorphe. La propriété de correction du typage pour les configuration bien typées par inférence est préservée, puisque ce système autorise moins de configurations. Cette approche utilise la stabilité de typabilité comme un outil permettant de prouver que les termes bien typés n'échouent pas, et non pas comme une fin en soit.

4.5 Exemples de typage

Un premier exemple simple consiste en un canal qui émet les messages qu'il reçoit :

$$\texttt{def}\ \texttt{fwd}\langle n, args\rangle \triangleright n\langle args\rangle\ \texttt{in}\ P$$

Si le canal `fwd` est défini dans une locations où les noms $\mathbf{d}$ et $\mathbf{d}'$ sont disponibles, lors du typage de P le canal `fwd` a pour type (par les règles de typage JOIN et DEF) :

$$\texttt{fwd} : \forall\alpha.\langle\langle\alpha\rangle_{\{\mathbf{d},\mathbf{d}'\}}, \alpha\rangle$$

Ainsi le premier argument de `fwd` peut être n'importe quel canal statique, ou un canal dynamique lié à $\mathbf{d}$ ou $\mathbf{d}'$, qui sont les seuls canaux dynamiques disponibles localement. Si l'on essaie d'utiliser

`fwd` de la manière suivante : $\texttt{fwd}\langle \mathbf{d}'', m\rangle$, avec m étant un argument ayant un type compatible avec $\mathbf{d}''$ mais $\mathbf{d}''$ étant différent de $\mathbf{d}$ ou $\mathbf{d}'$, le message ne sera pas typable, puisque $\langle\tau\rangle_{\{\mathbf{d}''\}}$ n'est pas un sous-type de $\langle\tau\rangle_{\{\mathbf{d},\mathbf{d}'\}}$.

Nous continuons maintenant l'exemple de l'introduction de ce chapitre (voir page 38). Dans la première version du canal **print**, le serveur s_1 a pour type $loc(\{\mathbf{print}\})$. Dans la nouvelle version, le serveur s_2 a pour type $loc(\{\mathbf{print}, \mathbf{setcolor}\})$. Lors du typage de l'agent, le canal `hi` a pour type $\langle loc(\{\mathbf{print}, \mathbf{setcolor}\})\rangle$. Ainsi, le message $\texttt{hi}\langle s_2\rangle$ est bien typé, alors que le message $\texttt{hi}\langle s_1\rangle$ n'est pas typable puisque $loc(\{\mathbf{print}\})$ n'est pas un sous-type de $loc(\{\mathbf{print}, \mathbf{setcolor}\})$. Cette utilisation incorrecte de `hi` est donc détectée.

Nous présentons enfin un exemple de typage où un canal dynamique est redéfini. Nous utilisons le dernier exemple de la section 4.3 :

$$\texttt{def reg}\langle n, \kappa\rangle \triangleright \texttt{def}\ a[n\langle\rangle \triangleright P : \mathbf{0}]^{\{n\},\emptyset}\ \texttt{in}\ \kappa\langle a\rangle\ \texttt{in}\ P$$

Durant le typage de P, le canal `reg` a pour type (nous détaillons la dérivation donnant ce type un peu plus loin) :

$$\texttt{reg} : \forall\delta.\langle\langle\rangle^+_\delta, \langle loc(\{\delta\})\rangle\rangle$$

Ainsi, le canal `reg` peut recevoir comme argument n'importe quel canal redéfinissable accompagné d'une continuation permettant de retourner le nom de la location définissant ce canal.

L'exemple précédent ne dépend pas des canaux dynamiques disponibles localement, puisqu'il se contente de redéfinir un canal que l'on passe en argument. Par contre, ce n'est plus le cas si le processus gardé utilise également le canal passé en argument, comme pour le canal $\texttt{r}'$:

$$\texttt{def r}'\langle n, \kappa\rangle \triangleright \texttt{def}\ a\,[n\langle\rangle \triangleright P : \mathbf{0}]^{\{n\},\emptyset}\ \texttt{in}\ \kappa\langle a\rangle \mid n\langle\rangle\ \texttt{in}\ P$$

Si le seul canal disponible localement est $\mathbf{d}$, le type de $\texttt{r}'$ est alors : $\langle\langle\rangle^+_{\mathbf{d}}, \langle loc(\{\mathbf{d}\})\rangle\rangle$. Par contre, si à la fois $\mathbf{d}$ et $\mathbf{d}'$ sont disponible, le canal $\texttt{r}'$ peut prendre soit le type $\langle\langle\rangle^+_{\mathbf{d}}, \langle loc(\{\mathbf{d}\})\rangle\rangle$, soit le type $\langle\langle\rangle^+_{\mathbf{d}'}, \langle loc(\{\mathbf{d}'\})\rangle\rangle$, mais il n'existe pas de type principal. Pour exprimer un tel type, un système de typage plus complexe et plus riche serait nécessaire, comme par exemple un système possédant le polymorphisme borné.

Nous présentons en figure 4.14 une dérivation de typage pour le canal `reg` avec :

$$\begin{array}{rcl}
\tau_a & \stackrel{\text{def}}{=} & loc(\{\delta\}) \\
\sigma_a & \stackrel{\text{def}}{=} & \tau_a \\
\tau_r & \stackrel{\text{def}}{=} & \langle\langle\rangle^+_\delta, \langle loc(\{\delta\})\rangle\rangle \\
\sigma_r & \stackrel{\text{def}}{=} & \forall\delta.\tau_r \\
\Gamma' & \stackrel{\text{def}}{=} & \texttt{reg} : \sigma_r, n : \langle\rangle^+_\delta, \kappa : \langle loc(\{\delta\})\rangle \\
\Gamma_A & \stackrel{\text{def}}{=} & \Gamma', a : \sigma_a
\end{array}$$

et en supposant que nous avons une dérivation de typage $\{\delta\}; \emptyset; \Gamma_A \vdash P_a$.

Nous appelons D la définition de `reg` et P le processus :

$$\nu\mathbf{d}.\texttt{def}\ tmp\left[\mathbf{d}\langle\rangle \triangleright \mathbf{0} \wedge a\left[\texttt{come}\langle b\rangle \triangleright \texttt{go}\ b; \mathbf{d}\langle\rangle : \texttt{reg}\langle\mathbf{d}, \texttt{come}\rangle\right]^{\emptyset,\{\mathbf{d}\}} : \mathbf{0}\right]^{\{\mathbf{d}\},\emptyset}\ \texttt{in}\ \mathbf{0}$$

et nous typons le processus $\texttt{def}\ D\ \texttt{in def create}\langle\rangle \triangleright P\ \texttt{in create}\langle\rangle$ en figures 4.15 et 4.16 avec :

$$\begin{array}{rcl}
\Gamma_C & \stackrel{\text{def}}{=} & \texttt{reg} : \sigma_r, \texttt{create} : \langle\rangle \\
\Gamma_D & \stackrel{\text{def}}{=} & \Gamma_C + \mathbf{d} : \langle\rangle^+_{\mathbf{d}} \\
B & \stackrel{\text{def}}{=} & a : loc(\{\mathbf{d}\}), \texttt{come} : \langle loc(\{\mathbf{d}\})\rangle, tmp : loc(\{\mathbf{d}\}) \\
B' & \stackrel{\text{def}}{=} & a : loc(\{\mathbf{d}\}), \texttt{come} : \langle loc(\{\mathbf{d}\})\rangle \\
B'' & \stackrel{\text{def}}{=} & \texttt{come} : \langle loc(\{\mathbf{d}\})\rangle \\
A & \stackrel{\text{def}}{=} & a : loc(\{\mathbf{d}\}), \texttt{come} : \langle loc(\{\mathbf{d}\})\rangle, tmp : loc(\{\mathbf{d}\}) \\
\Gamma_T & \stackrel{\text{def}}{=} & \Gamma_D + A \\
\Gamma^b_T & \stackrel{\text{def}}{=} & \Gamma_T + b : loc(\{\mathbf{d}\})
\end{array}$$

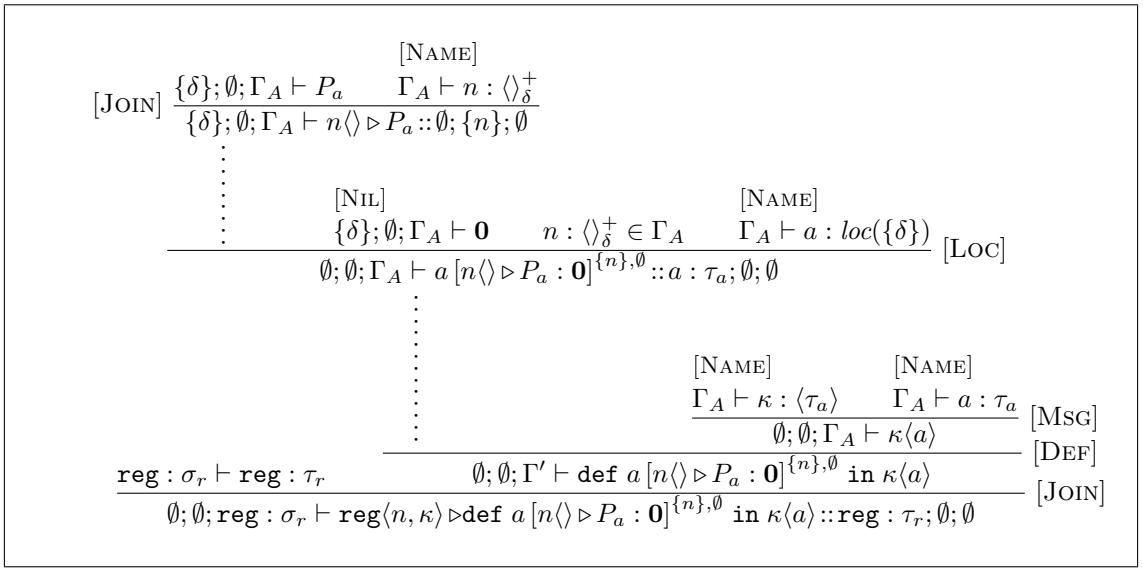


Fig. 4.14 – Typage du canal `reg`

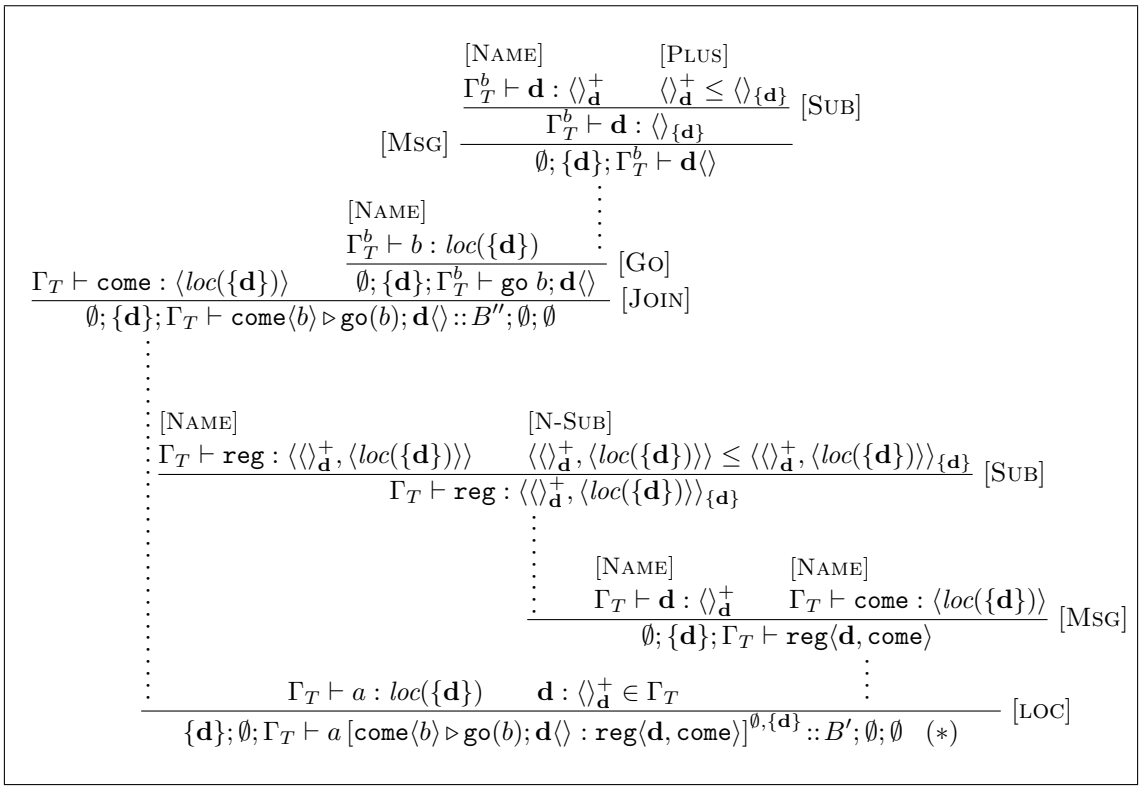


Fig. 4.15 – Typage d'un processus utilisant `reg`, partie 1

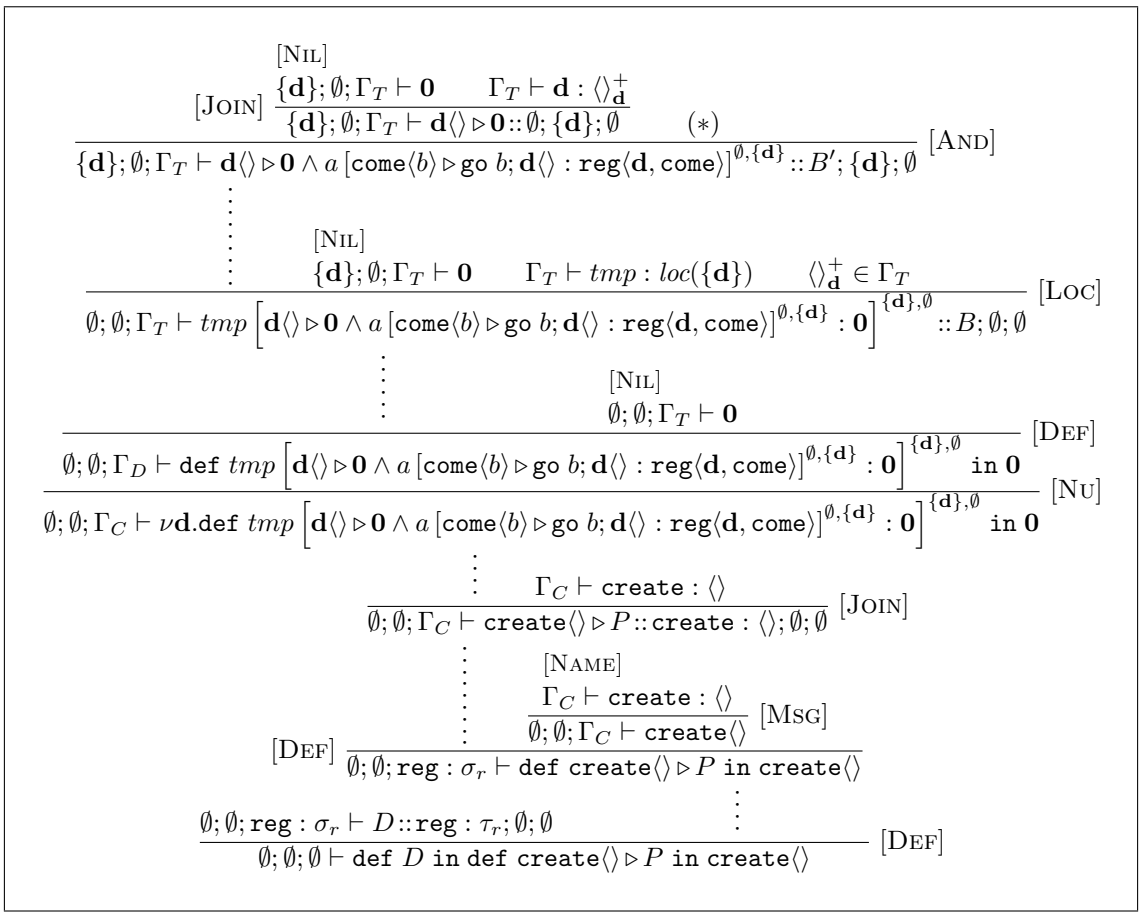


FIG. 4.16 – Typage d'un processus utilisant `reg`, partie 2

4.6 Sûreté du typage

Pour prouver la sûreté de notre système de typage, nous prouvons tout d'abord que la typabilité est stable par réduction, puis nous prouvons qu'une configuration typable ne peut échouer, en spécifiant ce que nous entendons par échec.

Nous commençons par nous restreindre à des environnements de typage *bien formés*.

Définition 4.6.1 *Un environnement de typage* Γ *est* bien formé *si et seulement si ses types sont bien formés et si toute association entre nom et schéma de type est de la forme :* $\mathbf{n} : \langle\tau\rangle^+_{\mathbf{n}}$, $\mathtt{n} : \forall\widetilde{\alpha}\widetilde{\delta}.\langle\tau\rangle$, *ou* $a : loc(\mathbf{\Delta})$ *avec* $\mathbf{\Delta}$ *sans variable de type de nom.*

Nous établissons tout d'abord que l'équivalence structurelle et la réduction préservent la typabilité.

Lemme 4.6.2 *Soient* $\mathcal{S}$ *une configuration et* $\Gamma \vdash \mathcal{S}$ *une dérivation de typage de cette configuration avec* Γ *bien formé. Pour toute configuration* $\mathcal{S}'$ *telle que* $\mathcal{S} \equiv \mathcal{S}'$, *il existe une dérivation de typage* $\Gamma' \vdash \mathcal{S}'$ *avec* Γ' *bien formé.*

Théorème 4 (Stabilité de la typabilité par réduction) *Soient* $\mathcal{S}$ *une configuration et* $\Gamma \vdash \mathcal{S}$ *une dérivation de typage de cette configuration avec* Γ *bien formé. Pour tout configuration* $\mathcal{S}'$ *telle que* $\mathcal{S} \to \mathcal{S}'$, *il existe une dérivation de typage* $\Gamma' \vdash \mathcal{S}'$ *avec* Γ' *bien formé.*

Nous montrons maintenant que l'étape NL-DYN résout bien les messages dynamiques de telle sorte qu'ils soient liés à la définition la plus proche dans les locations englobantes. Nous notons $dyn(b)$ l'ensemble des noms dynamique localement définis dans b. Ce lemme nous garantit que le calcul de la fonction de résolution correspond à notre spécification.

Lemme 4.6.3 *Soit* $\Gamma \vdash \mathcal{S}$ *une dérivation de typage avec* Γ *bien formé. Pour toute location dépliée* αa *de* $\mathcal{S}$ *et pour tout nom* n, *si nous avons* $n \in dyn(a) \cup imp(a)$, *alors* $F_{\alpha a}(n) = b \neq \bot$, *avec* $n \in dyn(b)$ *et* $\alpha a = \beta b \beta'$ *avec* $\forall c \in \beta'.n \notin dln(c)$.

Nous définissons une notion de *configuration bloquée*, qui correspond à un échec.

Définition 4.6.4 *Une configuration $\mathcal{S}$ est dite* bloquée *lorsque l'une des propriétés suivantes est vraie :*

1. *Un message de la forme $n\langle\widetilde{m}\rangle$ ou $a.n\langle\widetilde{m}\rangle$ est présent en contexte d'évaluation avec n n'étant pas un nom de canal ;*
2. *un nom $n \in dn(\mathcal{S})$ n'est pas un nom de canal ou un nom de location ;*
3. *un processus $\texttt{go}\ n; P$ est présent en contexte d'évaluation avec n n'étant pas un nom de location ;*
4. *un message de la forme $n\langle\widetilde{m}\rangle$ ou $a.n\langle\widetilde{m}\rangle$ est présent en contexte d'évaluation, et une règle de réaction de la forme $\ldots \mid n\langle\widetilde{y}\rangle \mid \ldots \triangleright P$ est également présente dans une location active avec les arités de $\widetilde{m}$ et $\widetilde{y}$ différentes ;*
5. *un message $\mathbf{n}\langle\widetilde{m}\rangle$ qui ne peut être réduit par une réduction* NL-Dyn *est présent en contexte d'évaluation ;*
6. *un message $a.n\langle\widetilde{m}\rangle$ est présent en contexte d'évaluation avec $n \notin dln(a)$.*

Nous pouvons maintenant établir que tout configuration typable ne peut être bloquée.

Théorème 5 (Sûreté du typage) *Soient $\mathcal{S}$ une configuration et $\Gamma \vdash \mathcal{S}$ une dérivation de typage avec Γ bien formé. La configuration $\mathcal{S}$ n'est pas bloquée.*

Nous obtenons donc en combinant les lemmes 4.6.2, 4.6.3 et les théorèmes 4 et 5 que toute configuration typable ne peut devenir bloquée, donc les messages dynamiques sont toujours liés à une définition qui est la plus proche définition dans les locations englobantes.

4.7 Conclusion

4.7.1 Travaux futurs sur le join calcul dynamique

Une de nos premières priorités consiste en l'intégration des canaux dynamiques dans JoCaml. Une forme de liaison dynamique existe déjà au niveau des modules, mais son utilisation est complexe. Une implémentation des canaux dynamiques permettrait une programmation plus simple des ressources locales. Notre système ayant été conçu en envisageant son implémentation, ceci ne devrait présenter aucune difficulté. Plus précisément, la maintenance de la fonction de résolution F peut facilement être ajoutée au niveau des runtimes, en utilisant le code qui déplie les locations après migration.

D'un point de vue plus théorique, il est dommage que les messages résolus $a.n\langle\widetilde{m}\rangle$, ressemblant fortement aux messages de [US01], ne soient qu'un état interne et ne puissent être utilisés par le programmeur. Pour qu'une telle construction ne conduise pas à des configurations bloquées, il est nécessaire de s'assurer que le canal n est bien défini dans la location a, où a peut être une variable liée par un filtre si le message est présent dans le processus gardé par le filtre. Nous avons conçu un tel système, et un tel système de types, et nous sommes en train d'en prouver la correction.

Nous n'avons pas présenté d'algorithme d'inférence de types pour notre système, pour deux raisons. La première est que afin d'obtenir la propriété de stabilité de la typabilité par réduction, il est beaucoup plus simple d'utiliser la récursion polymorphe pour typer les définitions. Bien entendu, un algorithme d'inférence ne peut utiliser la récursion polymorphe, et un système de types différent est nécessaire, comme c'est expliqué en 4.4.2. Ce problème n'a pas pour cause l'introduction des canaux dynamiques, puisqu'il est déjà présent dans le join calcul distribué. Un deuxième écueil à franchir est l'absence de types principaux. Nous sommes en train d'adapter le système B(T) [CP01] à notre calcul afin d'introduire du polymorphisme borné, nécessaire à l'obtention de ces types principaux.

Une autre extension de notre système tient plus de l'analyse statique que du typage. Lorsqu'un processus dans une location utilise un nom dynamique, celui-ci doit être en permanence disponible, même si son utilisation est temporaire. Il serait intéressant d'utiliser une information de séquentialité, comme par exemple l'utilisation d'un nom dynamique uniquement après une migration donnée, pour pouvoir modifier l'ensemble des noms importés lors d'une migration. Il serait aussi intéressant de pouvoir spécifier les noms exportés, afin de rendre plus fin le contrôle des ressources qu'une location rend disponibles.

Enfin, par de nombreux aspect le join calcul dynamique ressemble à un calcul à objets où les locations sont des objets et les noms dynamiques des noms de méthodes. Il serait donc intéressant d'étudier comment notre système de types s'adapte à un calcul à objets possédant des noms de méthodes comme valeurs de première classe pouvant être créées lors de l'exécution.

4.7.2 État de l'art

Notre stratégie en ce qui concerne les messages dynamiques est de procéder en deux temps : nous résolvons tout d'abord la destination du message, puis nous l'y envoyons. Une autre stratégie pourrait être de simplement envoyer un message dynamique qui n'est pas défini localement dans la location englobante, et ainsi de suite jusqu'à ce que le message arrive à la racine de l'arbre des locations ou parvienne à une location définissant le canal dynamique. Nous présentons d'ailleurs en section 5.4.5 une simulation du join calcul dynamique dans le M-calcul utilisant cette sémantique. Cette deuxième approche est semblable à [CG98] et [SV99]. Ces deux stratégies donnant deux comportements différents, il est intéressant de remarquer que le système de types présenté dans ce chapitre est également correct pour la deuxième stratégie et garantit que tout message parviendra à une location le définissant.

L'objectif principal de [ABL99] est de garantir que tout canal est réceptif, donc l'absence de deadlocks, dans un π calcul avec localités. Cette propriété de réceptivité est plus complexe à démontrer dans le π calcul puisque des récepteurs peuvent disparaître. Notre système est par conséquent plus simple puisque nous ne garantissons pas l'absence de deadlocks qui peuvent toujours avoir lieu à cause des filtres. Par contre, le système de types de [ABL99] impose que les noms passés en argument sur des canaux ne peuvent pas être ensuite utilisés en tant que récepteurs, à l'opposé des canaux dynamiques comme montré dans le dernier exemple de la section 4.3.

De nombreux travaux s'intéressent au contrôle de l'usage de ressources (à l'opposé de la disponibilité des ressources) dans des π calculs avec localité et reliaison objective. Dans [HRes], les localités ont une structure plate et les processus peuvent migrer objectivement d'une localité à l'autre. Un système de types garantit qu'aucun agent (c'est à dire aucun processus ayant migré) ne puisse accéder à une ressource s'il n'a pas reçu l'autorisation de le faire. Dans le *local area calculus* [CS01], les localités sont structurées en une hiérarchie fixe de niveaux qui ne migrent pas. Les canaux possèdent eux aussi un niveau qui interdit toute communication si le chemin entre les localités essayant de communiquer passe par une localité de niveau supérieur. Le *box-*π calcul [SV99] introduit lui aussi des localités structurées en une hiérarchie fixe. Dans ce calcul une communication ne peut traverser qu'une frontière de localité à la fois, ce qui permet de contrôler le flot d'informations entre les localités.

Le π calcul d'ordre supérieur de [YH00] s'intéresse aussi à la question du contrôle d'accès, en spécifiant explicitement pour chaque réception quelles ressources sont accessibles par le processus reçu, en opérant une distinction entre lecture et écriture sur un canal. Ce calcul est objectif et ne garantit pas la disponibilité des ressources. Un autre π calcul d'ordre supérieur [YH99] garantit la localité des ressources (et non leur disponibilité) en utilisant un système de types très semblable au nôtre. Cependant, leur utilisation de types dépendants au lieu de polymorphisme accompagné de variables de types de noms semble rendre leur système de types plus complexe. Le contrôle d'accès de Klaim [NFPV00] est lui aussi basé sur un calcul où la migration de processus est objective.

Le travail sur les groupes permettant de garder secrets certains noms [CGG00b] a lui aussi comme objectif le contrôle d'accès à ces noms en les associant à des groupes. De nouveaux groupes peuvent être créés durant l'exécution du processus. Cependant, les objectifs différents conduisent à des systèmes de types différents. Dans notre cas, le but est de s'assurer que la ressource est disponible, et nous utilisons le polymorphisme pour permettre aux noms dynamiques nouvellement introduits de sortir de la portée de la restriction les ayant introduits. Dans [CGG00b], le secret des noms est justement garanti en montrant que les groupes ne peuvent s'échapper de la portée de la restriction les introduisant.

Un calcul simple possédant une propriété de liaison dynamique associée à la localité est le calcul des ambients [CG98]. En effet, plusieurs ambients peuvent avoir le même nom, et l'ambient désigné par une capacité (une requête de migration par exemple) est choisi parmi les voisins de l'ambient exerçant la capacité. De récents travaux sur les ambients spécifient des systèmes de types pour analyser ou restreindre leur comportement. Dans [CGG00a], une notion de groupe est introduite pour contrôler l'échappement de certaines capacités. Dans [BC01], des ambients *sûrs* sont typés par rapport à une spécification de sécurité. Le système de types prend alors en compte les capacités qu'un ambient peut acquérir pendant l'exécution. Les *boxed ambients* [BCC01] remplacent la capacité *open* de dissolution d'un ambient par la possibilité de communiquer entre père et fils. Un système de types permet d'analyser le comportement de chaque ambient, en différenciant les communications locales des communications avec le contexte. Ces calculs s'intéressent donc principalement au contrôle des ressources. De plus, ils ne possèdent pas de notion de noms statiques.

4.7.3 Conclusion et limites du join calcul dynamique

Nous avons présenté dans ce chapitre une extension du join calcul distribué avec une forme de liaison dynamique associée à la localité. De nouveaux canaux dynamiques peuvent être créés pendant l'exécution et de nouvelles définitions peuvent être associées à des canaux existant. Un système de types garantissant la présence d'une définition pour tout message a également été présenté.

Bien que le join calcul dynamique nous permette de donner du sens à la notion de localité dans le join calcul distribué, cette extension reste limitée à plusieurs points de vue. Tout d'abord, la liaison dynamique ne concerne que la communication. Il est par exemple impossible d'écrire directement un processus de la forme `go` *up* pour indiquer que l'on souhaite sortir de la location englobante (comme une capacité Out des ambients). Ensuite, nous ne nous donnons que la possibilité de paramétrer la destination finale d'un message par rapport à la position courante. Il n'est par contre pas possible d'intercepter des messages ou des locations venant d'une sous-location ou de l'extérieur, comme le ferait un pare-feu.

Suite à ces observations, nous avons prolongé notre réflexion en concevant un nouveau calcul de processus, le M-calcul, très inspiré du join calcul mais essayant de répondre à ces limitations.

Remerciements Nous voudrions remercier Sylvain Conchon, Cédric Fournet, James Leifer, Jean-Jacques Lévy, François Pottier, et Didier Rémy pour leur précieux commentaires.

Chapitre 5

Le M-calcul

Ce chapitre présente un nouveau calcul de processus répartis, appelé M-calcul. Ce calcul constitue un modèle formel de programmation répartie avec mobilité de processus, dont les motivations sont similaires à celles avancées par L. Cardelli pour son calcul des ambients. Le calcul des ambients et d'autres calculs de processus répartis comme le join calcul ou le Dπ-calcul, introduisent des notions de localités, qui correspondent à une partition spatiale des exécutions réparties et qui sont censées capturer différents aspects de ces exécutions (par exemple, aspects liés aux défaillances, au contrôle d'accès, à la migration de processus, etc). Ces différents calculs de processus demeurent cependant insatisfaisants car ils associent tous un comportement unique et prédéfini à la notion de localité qu'ils introduisent. Dans un système réparti de grande taille, on peut au contraire s'attendre à trouver des localités de différentes sortes, avec des comportements très variés. Le M-calcul essaie de pallier cette limitation en fournissant un modèle de programmation réparti qui autorise la programmation explicite du comportement des localités. Plus précisément, le M-calcul peut être compris comme une généralisation du join calcul qui comprend : des localités au comportement programmable (domaines), des processus d'ordre supérieur, de la mobilité de processus et une forme de liaison dynamique.

5.1 Introduction

Notre objectif dans ce chapitre est de développer un calcul de processus inspiré à la fois du calcul des ambients et du join calcul, en prenant en compte les difficultés associées à des deux calculs et décrites en section 3.6.2, et en intégrant les enseignements du join calcul dynamique.

Du join calcul, nous désirons conserver la facilité à communiquer à distance, en nous affranchissant de la nécessité de spécifier le chemin que le message doit parcourir pour parvenir à destination, comme c'est le cas par exemple dans le calcul des ambients. Nous voulons également conserver la facilité d'implémentation, en évitant toute forme de synchronisation distribuée. Enfin, nous considérons que la notion de définitions sous forme de filtre de messages associée à un processus est une construction riche très utile à la programmation.

Du join calcul dynamique, nous gardons la possibilité de définir un canal à plusieurs endroits, afin d'obtenir une notion de liaison dynamique associée à la localité. Nous soulignons également la notion de déterminisme de la résolution de nom, cruciale à la facilité d'implémentation. Nous devons en effet être capable de déterminer localement ce qu'il faut faire avec tout message, en particulier résoudre sa destination s'il doit être envoyé ailleurs. Par contre, nous désirons ne plus dépendre de la structure arborescente des localités pour résoudre la liaison dynamique. Cette structure correspond intuitivement à une notion d'héritage simple (non multiple), rendant complexe le codage de la mise à jour de plusieurs bibliothèques indépendantes.

Du calcul des ambients, nous retenons les aspects très locaux de la migration et de la communication, permettant de contrôler les messages et agents franchissant une frontière de localité.

Nous souhaitons également pouvoir exprimer dans le M-calcul plusieurs protocoles d'interactions entre localités, plusieurs modes d'accès à des ressources, plusieurs notions de pannes, sans pour autant devoir modifier la sémantique opérationnelle du calcul pour chaque variante. Pour ce faire, nous dotons chaque localité du M-calcul, désormais appelées domaines, d'un *contrôleur*, c'est à dire un processus du M-calcul contrôlant l'interaction du contenu du domaine avec le monde extérieur au domaine.

Nous combinons les objectifs précédemment décrits de la manière suivante. Du point de vue de l'envoi de messages, nous distinguons deux types de messages : des messages locaux, qui restent

dans le domaine courant, et des messages adressés, qui peuvent atteindre des domaines distants. Les premiers ressemblent aux messages du calcul des ambients, à l'exception près qu'il sont envoyés sur des canaux nommés. Les messages adressés sont par contre très similaires aux messages résolus du join calcul dynamique. De plus, à partir du moment où le nom de tout domaine est unique, la destination des messages adressés est univoque. Le contrôle des messages franchissant une frontière de domaine est effectué en propageant les messages adressés de proche en proche et en laissant l'occasion à chaque contrôleur de domaine traversé d'intercepter le message. Plus précisément, tout message extérieur dont la destination est le domaine ou un sous-domaine est passé au port spécial **i** défini dans le contrôleur. De la même manière, tout message du contenu destiné à l'extérieur du contenu est passé au port spécial **o**. Les définitions associées à ces ports déterminent la transparence du domaine donné. Les messages présents dans un contrôleur se déplacent par contre librement et automatiquement vers le domaine cible. Ainsi, la communication distante est transparente si les ports spéciaux réémettent les messages interceptés, et il n'est bien sûr pas nécessaire de spécifier le chemin jusqu'à la destination.

Les récepteurs associés aux messages sont tout simplement des définitions du join calcul, c'est à dire des récepteurs répliqués associant un processus à un filtre de messages. Par contre, à la différence du join calcul et de manière similaire au join calcul dynamique, la définition d'un canal ne lie pas ce canal. Il est ainsi possible de définir le même canal dans plusieurs domaines distincts.

Les mécanismes de contrôle liés à la communication distante sont très proches de ceux liés à la migration. C'est pourquoi nous unifions ces deux notions dans le M-calcul en considérant la migration comme étant la communication d'un processus gelé : le domaine migrant. Nous représentons simplement un processus gelé par une valeur, une λ-abstraction $\lambda.P$, en nous interdisant d'évaluer un processus sous un λ. Comme nous désirons conserver la notion de migration subjective présente dans le join calcul comme dans le calcul des ambients, nous avons besoin d'un opérateur subjectif permettant de transformer un domaine en un processus gelé. Nous choisissons donc un opérateur de *passivation* `pass` réalisant ceci. Le M-calcul est donc un join calcul d'ordre supérieur.

Enfin, comme nous l'avons vu précédemment, il est très intéressant pour obtenir une implémentation efficace du M-calcul de pouvoir garantir que tout domaine pouvant être la cible d'un message distant possède un nom unique. Notre calcul étant trop riche pour garantir cette propriété syntaxiquement, comme c'est le cas pour le join calcul, nous développons un système de types nous donnant cette garantie.

La structure de ce chapitre est la suivante. Nous définissons le M-calcul en section 5.2, puis nous définissons le système de types associé en section 5.3. Nous présentons ensuite de nombreux exemples en section 5.4 et concluons en section 5.5

5.2 Le M-calcul : syntaxe et sémantique opérationnelle

Nous définissons dans cette section la syntaxe et sémantique du M-calcul.

5.2.1 Syntaxe

La syntaxe du M-calcul est une extension à la fois du λ-calcul avec appel par valeur et du join calcul. Du premier nous conservons les notions de λ-abstraction et d'application. Du deuxième nous retenons la notion de définition de canaux (appelés ici *noms de ressources*) par des filtres de messages et la composition parallèle. Nous remplaçons la notion de localité du join calcul par la notion de domaine $a(P)[Q]$ pour un domaine nommé a, ayant P comme contrôleur et Q comme contenu. Nous ajoutons la notion de passivation subjective par l'opérateur `pass`, la notion de test de noms avec l'opérateur $[pat = V]P, Q$, et la restriction de nom du π-calcul $\nu n.P$.

La syntaxe du M-calcul est décrite en figures 5.1 et 5.2. Nous présupposons l'existence d'un ensemble infini dénombrable de *noms de domaines* DN (contenant un élément particulier que nous notons ϵ), l'existence d'un ensemble infini dénombrable de noms de ressources Ref (contenant deux éléments particuliers notés **i** et **o**), et l'existence d'un ensemble infini dénombrable de variables Var. L'ensemble des noms N est l'union disjointe des ensembles DN, Var, Ref, et des noms adressés $((\mathsf{DN} \cup \mathsf{Var}) \times (\mathsf{Ref} \cup \mathsf{Var}))$. Les notations suivantes comprennent également les variantes annotées (par exemple r' pour r). Nous notons r les éléments de Ref ; x, y les éléments de Var ; a, b les éléments de DN ; u, v les éléments de N. Nous notons n les noms simples résolus (qui ne sont pas des variables), c'est à dire l'union disjointe de DN et Ref. Nous notons $\mathbf{r}$ les noms de ressources variables, c'est à dire l'union disjointe de Ref et de Var ; $\mathbf{a}$ les noms de domaines variables, c'est à dire l'union disjointe de DN et de Var ; d les noms de destination, c'est à dire des noms de domaines variables surlignés lorsque la destination est le contrôleur, et soulignés lorsque c'est le contenu ;

$\mathcal{S}$	::=		configuration
		$\epsilon[P]$	domaine racine
	|	$\nu n.\mathcal{S}$	restriction
P	::=		processus
		$\mathbf{0}$	processus inerte
	|	V	valeur
	|	$\mathbf{a}(P)[P]$	domaine
	|	$(P \mid P)$	composition parallèle
	|	(PP)	application fonctionnelle
	|	$\nu n.P$	restriction
	|	$[pat = V]P, P$	conditionnelle
	|	$\langle D \rangle$	définition
	|	$\mathtt{pass}_{\mathbf{a}}\ V$	passivation
	|	$P_1, \ldots, P_q$	nuplet
V	::=		valeur
		$()$	valeur vide
	|	u	nom
	|	d	destination
	|	$\lambda x.P$	λ-abstraction
	|	$V_1, \ldots, V_q$	nuplet de valeurs
D	::=		définition
		$\top$	définition vide
	|	$J \triangleright P$	règle de réaction
	|	$D; D$	composition
J	::=		filtre
		$r\widetilde{x}$	message requis
	|	$J \mid J$	synchronisation

FIG. 5.1 – Syntaxe du M-calcul

nous notons enfin s les noms de services, c'est à dire l'union disjointe des noms de ressources variables et des noms adressés (composés d'un nom de domaine variable et d'un nom de ressource variable).

Nous notons θ les substitutions finies des noms vers les termes, telles que les variables sont substitués par des termes, les noms de domaines par des noms de domaines et les noms de ressources par des noms de ressources. Le système de types garantit que toutes les substitutions utilisées sont correctes. Nous notons $\{^{t_1}/_{u_1}, \ldots, ^{t_q}/_{u_q}\}$ une substitution finie où chaque nom u_i est simultanément substitué par le terme t_i. Nous notons $P\theta$ ou $P\{^{t_1}/_{u_1}, \ldots, ^{t_q}/_{u_q}\}$ l'image de P par la substitution. Nous notons $\widetilde{t}$ un nuplet de termes $(t_1, \ldots, t_q)$. Lorsque la taille des nuplets $\widetilde{t}$ et $\widetilde{u}$ est identique, nous notons $\{^{\widetilde{t}}/_{\widetilde{u}}\}$ la substitution $\{^{t_1}/_{u_1}, \ldots, ^{t_p}/_{u_p}\}$.

Nous remarquons que dans le cas d'un test, une substitution s'applique de la manière suivante : $[pat = V](P, Q)\theta = [pat = (V\theta)](P\theta, Q\theta)$: on ne substitue pas dans le filtre de test.

Nous notons P, Q, R, S les processus du M-calcul. Comme dans le λ-calcul, certains processus du M-calcul sont considérés comme étant des *valeurs*. Nous notons ces dernières V, et elles sont définies en figure 5.1. Les processus de la forme $\langle D \rangle$ sont appelés des définitions. Ils sont similaires aux ressources dupliquées du calcul bleu, ou aux définitions de join calcul. Comme dans le join calcul, une définition est composé de règles de réactions $J \triangleright Q$ comprenant un filtre J ainsi qu'un processus gardé Q.

Nous adoptons la convention usuelle en ce qui concerne la portée des lieurs $\lambda x.P$ et $\nu n.P$: cette portée s'étend autant que possible à droite. Nous notons $PQ_1 \ldots Q_n$ pour $(\ldots(PQ_1)\ldots Q_n)$, et $\lambda u_1 \ldots u_q.P$ pour $\lambda u_1.\ldots.\lambda u_q.P$. Similairement, nous notons $\nu u_1 \ldots u_q.P$ pour $\nu u_1.\ldots.\nu u_q.P$. Nous notons enfin $\lambda.P$ pour $\lambda x.P$ où x n'est pas libre dans P.

La signification informelle des différentes constructions du M-calcul est la suivante :

– Un programme complet du M-calcul prend la forme d'un domaine racine $\epsilon[P]$, avec ϵ étant le nom du domaine racine, et P le processus correspondant au programme. Nous remarquons

n	::=		noms simples résolus
		r	nom de ressource
	|	a	nom de domaine
$\mathbf{r}$	::=		nom de ressource variable
		r	nom de ressource
	|	x	variable
$\mathbf{a}$	::=		nom de domaine variable
		a	nom de domaine
	|	x	variable
d	::=		destination
		$\overline{\mathbf{a}}$	pour le contrôleur
	|	$\underline{\mathbf{a}}$	pour le contenu
s	::=		nom de service
		$\mathbf{r}$	nom de ressource variable
	|	$d.\mathbf{r}$	ressource adressée
u	::=		nom
		a	nom de domaine
	|	s	nom de service
pat	::=		filtre de nom
		u	nom
	|	$\_$	tout
	|	$d.\_$	ressource pour d
	|	$\_.\mathbf{r}$	ressource $\mathbf{r}$ adressée
	|	$\_.\_$	ressource adressée

FIG. 5.2 – Noms

que du point de vue du comportement, ce domaine racine est équivalent à un domaine de la forme $\epsilon(\mathbf{0})[P]$, c'est à dire un domaine avec un contrôleur inexistant.
- Le processus $\mathbf{0}$ est le processus inactif. Ce processus ne peut se réduire et est élément neutre de l'opérateur de composition parallèle | dans l'équivalence structurelle.
- Un processus de la forme $a(P)[Q]$ est un domaine. Chaque domaine comprend un *nom de domaine* a, un processus *contrôleur* P et un processus *contenu* Q. Bien entendu, le contrôleur ainsi que le contenu peuvent inclure des domaines, donnant ainsi aux processus du M-calcul une structure d'arbre.
- Un processus peut être une valeur V, c'est à dire soit la valeur vide (), soit un nom u, soit une λ-abstraction (c'est à dire un processus de la forme $\lambda x.P$). Un nom (défini dans la figure 5.2) est soit une variable, soit un nom de ressource (c'est à dire un nom défini dans le filtre d'une définition), soit un nom de domaine, soit un nom de ressource adressé (c'est à dire la concaténation d'un nom de domaine ou d'une variable et d'un nom de ressource ou d'une variable). Une λ-abstraction est considérée comme étant une valeur puisque nous interdisons l'exécution de processus gardé par un λ. Nous remarquons qu'une λ-abstraction ne peut lier que des variables, alors que les noms de ressources d'une définition ne peuvent être des variables. Ainsi, il est impossible d'écrire un processus de la forme $\lambda x.\langle x = P\rangle$ dans le M-calcul. Cette contrainte est très similaire à celle décrite dans le calcul bleu [Bou98].
- Un processus de la forme $P \mid Q$ est la composition parallèle des deux processus P et Q. L'opérateur de composition parallèle est commutatif, associatif, et a $\mathbf{0}$ pour élément neutre dans l'équivalence structurelle.
- Un processus de la forme PQ est une application fonctionnelle : le processus P est une fonction qui est appliquée à l'argument Q. Notre sémantique étant en *appel par valeur*, le processus Q doit tout d'abord être évalué en une valeur V avant que l'application ne soit réduite.
- Un processus de la forme $\nu n.P$ correspond à la restriction classique du π-calcul [Mil99]. Le nom n est considéré comme nouveau dans le processus P.
- Un processus $[pat = V]P, Q$ est un choix conditionnel. Si le filtre pat accepte la valeur V, alors P est exécuté, sinon Q est exécuté.
- Un processus $\langle J_1 \triangleright P_1; \ldots J_q \triangleright P_q\rangle$ correspond à la définition de ressources. Chaque J_i est un filtre de la forme $r_1\widetilde{x_1} \mid \ldots \mid r_p\widetilde{x_p}$, où les r_i sont les noms de ressources *définis* par le filtre, et où les variables $\widetilde{x_i}$ sont les variables *reçues*, correspondant aux paramètres formels et liées dans le processus gardé. Comme dans le join calcul, les filtres sont linéaires : dans un filtre donné, chaque nom de ressource est défini au plus une fois, et chaque variable reçue est présente au plus une fois. Intuitivement, lorsque des messages sont présents sur chaque nom définis par un filtre J_i, ce filtre est activé et les messages sont remplacés par le processus gardé P_i, après substitution des paramètres formels par les arguments des messages. Un message dans le M-calcul est tout simplement l'application d'un nom de ressource r à un nuplet de valeurs $(V_1, \ldots, V_p)$.
- Enfin, un processus $\texttt{pass}_a\ V$ présent dans le contrôleur d'un domaine a entraîne la passivation du domaine lorsqu'il est évalué. La passivation du domaine est tout simplement la transformation de son contrôleur et de son contenu en formes gelées (de la forme $\lambda.P$). Ces deux valeurs sont ensuite passées en argument à la fonction V qui décrit par conséquent le comportement à adopter après passivation.

5.2.2 Sémantique opérationnelle

Un M-contexte est un terme utilisant les mêmes constructions que les termes du M-calcul ainsi qu'une constante supplémentaire $\cdot$, appelée *trou*. Nous notons $C(\cdot)$ un M-contexte. Il est possible de remplir le trou d'un M-contexte avec un processus à condition que le processus obtenu soit syntaxiquement correct. Nous notons $C(Q)$ le processus correspondant au M-contexte $C(\cdot)$ dont le trou a été rempli par Q. Les contextes d'évaluation $\mathbf{E}$ sont des M-contextes dont la grammaire est donnée par la figure 5.3.

Nous notons $fn(P)$ l'ensemble des noms libres de P. Cet ensemble est défini récursivement en figure 5.4. Nous notons $dn(J)$ l'ensemble des noms de ressources définis par le filtre J, c'est à dire :

$$dn(r_1\widetilde{x}_1 \mid \ldots \mid r_q\widetilde{x}_q) = \{r_1, \ldots, r_q\}$$

Nous notons $dln(P)$ l'ensemble des noms défini localement dans P, c'est à dire l'ensemble des noms définis dans les filtres des définitions de P en contexte d'évaluation, sans inspecter les sous domaines de P. L'ensemble $dln(P)$ est défini récursivement en figure 5.5. Nous notons $doms(P)$ le multi-ensemble des domaines actifs de P, c'est à dire le multi-ensemble des domaines présents en

$\mathbf{E}$	::=		contexte d'évaluation
		$\cdot$	trou
	$\mid$	$\mathbf{E}V$	fonction
	$\mid$	$P\mathbf{E}$	argument
	$\mid$	$\nu n.\mathbf{E}$	restriction
	$\mid$	$\mathbf{E} \mid P$	composition parallèle gauche
	$\mid$	$P \mid \mathbf{E}$	composition parallèle droite
	$\mid$	$a(P)[\mathbf{E}]$	contenu
	$\mid$	$a(\mathbf{E})[P]$	contrôleur
	$\mid$	$P_1, \ldots, \mathbf{E}, \ldots, P_q$	nuplet
	$\mid$	$\epsilon[\mathbf{E}]$	racine

FIG. 5.3 – Contextes d'évaluation

$$fn(\mathbf{a}) = fn(\overline{\mathbf{a}}) = fn(\underline{\mathbf{a}}) = \{\mathbf{a}\} \quad fn(\mathbf{r}) = \{\mathbf{r}\} \quad fn(d.\mathbf{r}) = fn(d) \cup \{\mathbf{r}\}$$
$$fn(\_) = \emptyset \quad fn(d.\_) = fn(d) \quad fn(\_.\{\mathbf{r}\}) = \{\mathbf{r}\}$$
$$fn(()) = fn(\mathbf{0}) = \emptyset \qquad fn(\lambda x.P) = fn(P) \setminus \{x\}$$
$$fn(\nu n.P) = fn(P) \setminus \{n\} \qquad fn(PQ) = fn(P) \cup fn(Q)$$
$$fn(\langle D \rangle) = fn(D) \qquad fn(D; D') = fn(D) \cup fn(D')$$
$$fn(\top) = \emptyset \qquad fn(J \triangleright P) = (fn(P) \setminus fn(J)) \cup dn(J)$$
$$fn(r_1\widetilde{x}_1 \mid \ldots \mid r_q\widetilde{x}_q) = fn(r_1\widetilde{x}_1) \cup \ldots \cup fn(r_q\widetilde{x}_q) \qquad fn(r\widetilde{x}) = \{r, x_1, \ldots, x_p\} \quad \widetilde{x} = (x_j)^{j\in[1\ldots p]}$$
$$fn(a(P)[Q]) = \{a\} \cup fn(P) \cup fn(Q) \qquad fn(P \mid Q) = fn(P) \cup fn(Q)$$
$$fn([pat = V]P, Q) = fn(pat) \cup fn(P) \cup fn(Q) \cup fn(V) \qquad fn(\mathtt{pass}_a\ V) = \{a\} \cup fn(V)$$
$$fn(\nu n.\mathcal{S}) = fn(\mathcal{S}) \setminus \{n\} \qquad fn(\epsilon[P]) = fn(P)$$

FIG. 5.4 – Noms libres

contexte d'évaluation dans P. Cet ensemble est défini récursivement en figure 5.6. Nous remarquons que nous ne considérons pas les définitions (pour $dln(P)$) ni les domaines (pour $doms(P)$) présents dans la fonction P ou dans l'argument Q lors d'une application fonctionnelle ou présents dans le P_i d'un nuplet $(P_1, \ldots, P_q)$, le typage nous garantissant que aucune définition ou aucun domaine ne peut être présent en contexte d'évaluation dans P, Q ni dans un P_i (cf. la preuve du théorème 7 page 175).

Nous notons $P =_\alpha Q$ lorsque deux processus P et Q sont α-équivalents. Nous rappelons que dans $\nu n.P$, $\lambda x.P$ et $r_1\widetilde{x_1} \mid \ldots \mid r_q\widetilde{x_q} = P$, les noms et variables n, x, et chaque x_i sont liés dans P, et peuvent donc être α-convertis.

La sémantique opérationnelle du M-calcul est défini dans le style d'une machine chimique abstraite [BB92], par une notion d'équivalence structurelle $\equiv$ et une notion de relation de réduction $\rightarrow$.

La relation d'équivalence structurelle $\equiv$ est la plus petite relation réflexive, symétrique et transitive satisfaisant les règles de la figure 5.7, telle que l'opérateur "|" soit commutatif et associatif avec $\mathbf{0}$ pour élément neutre, et telle que l'opérateur ";" soit commutative et associatif avec $\top$ pour élément neutre.

La signification intuitive de ces règles est la suivante :

- Les règles STRUCT.NU.PAR, STRUCT.NU.TOP, STRUCT.NU.BOSS et STRUCT.NU.CONT sont les règles d'extrusion de portée. Elles indique que l'opérateur de restriction crée des noms qui sont uniques dans toute la configuration.
- La règle STRUCT.α indique que les processus ou configurations α-équivalents sont structurellement équivalents.
- La règle STRUCT.CONTEXT indique que l'équivalence $\equiv$ est une congruence pour les contextes d'évaluation.

La relation de réduction $\rightarrow$ pour le M-calcul est définie comme la plus petite relation qui satisfait les règles des figures 5.8, 5.9 et 5.10. Dans les règles de la figure 5.10, nous notons $\underline{b}$ pour $\overline{b}$ ou $\underline{b}$.

Nous définissons un opérateur de filtrage sur les valeurs *match* qui est satisfait dans les cas suivant :

$$match(\_, V) \qquad match(u, u) \qquad match(d.\_, d.\mathbf{r}) \qquad match(\_.\mathbf{r}, d.\mathbf{r}) \qquad match(\_.\_, d.\mathbf{r})$$

Nous notons $\neg match(pat, V)$ si $match(pat, V)$ n'est pas satisfait.

$$
\begin{array}{cc}
dln(()) = \emptyset & dln(\mathbf{0}) = \emptyset \\
dln(u) = \emptyset & dln(\lambda x.P) = \emptyset \\
dln(\nu n.P) = dln(P) \setminus \{n\} & dln(PQ) = \emptyset \\
dln(\langle D \rangle) = dln(D) & dln(D; D') = dln(D) \cup dln(D') \\
dln(\top) = \emptyset & dln(J \triangleright P) = dln(J) \\
dln(r_1 \widetilde{x}_1 \mid \ldots \mid r_q \widetilde{x}_q) = \{r_1, \ldots, r_q\} & dln(a(P)[Q]) = \emptyset \\
dln(P \mid Q) = dln(P) \cup dln(Q) & dln([pat = V]P, Q) = \emptyset \\
dln(\mathtt{pass}_a\ V) = \emptyset & dln(\mathcal{S}) = \emptyset
\end{array}
$$

FIG. 5.5 – Noms définis locaux

$$
\begin{array}{cc}
doms(()) = \emptyset & doms(\mathbf{0}) = \emptyset \\
doms(u) = \emptyset & doms(\lambda x.P) = \emptyset \\
doms(\nu n.P) = doms(P) \setminus \{n\} & doms(PQ) = \emptyset \\
doms(\langle D \rangle) = \emptyset & doms(a(P)[Q]) = \{a\} \cup doms(P) \cup doms(Q) \\
doms(P \mid Q) = doms(P) \cup doms(Q) & doms([pat = V]P, Q) = \emptyset \\
doms(\mathtt{pass}_a\ V) = \emptyset & doms(\nu n.\mathcal{S}) = doms(\mathcal{S}) \setminus \{n\} \\
doms(\epsilon[P]) = doms(P) & doms(P_1, \ldots, P_q) = \emptyset
\end{array}
$$

FIG. 5.6 – Domaines actifs

$$\frac{n \notin fn(Q)}{(\nu n.P) \mid Q \equiv \nu n.P \mid Q}\ [\textsc{Struct.Nu.Par}] \qquad \frac{}{\epsilon[\nu n.P] \equiv \nu n.\epsilon[P]}\ [\textsc{Struct.Nu.Top}]$$

$$\frac{n \notin fn(Q) \ \wedge\ n \neq a}{a(\nu n.P)[Q] \equiv \nu n.a(P)[Q]}\ [\textsc{Struct.Nu.Boss}] \qquad \frac{n \notin fn(P) \ \wedge\ n \neq a}{a(P)[\nu n.Q] \equiv \nu n.a(P)[Q]}\ [\textsc{Struct.Nu.Cont}]$$

$$\frac{P =_\alpha Q}{P \equiv Q}\ [\textsc{Struct.}\alpha] \qquad \frac{P \equiv Q}{\mathbf{E}\{P\} \equiv \mathbf{E}\{Q\}}\ [\textsc{Struct.Context}]$$

FIG. 5.7 – Équivalence structurelle

$$\frac{}{(\lambda x.P)V \to P\{^V/_x\}}\ [\textsc{Red.Beta}] \qquad \frac{match(pat, V)}{[pat = V]P, Q \to P}\ [\textsc{Red.If.Then}]$$

$$\frac{\neg match(pat, V)}{[pat = V]P, Q \to Q}\ [\textsc{Red.If.Else}] \qquad \frac{}{a(\mathtt{pass}_a\ V \mid P)[Q] \to V(\lambda.P)(\lambda.Q)}\ [\textsc{Red.Passiv}]$$

$$\frac{\langle D \rangle = \langle D_0\ ;\ r_1\widetilde{x_1} \mid \ldots \mid r_n\widetilde{x_n} \triangleright P \rangle}{\langle D \rangle \mid r_1\widetilde{V_1} \mid \ldots \mid r_n\widetilde{V_n} \to \langle D \rangle \mid P\{^{\widetilde{V_i}}/_{\widetilde{x_i}}\}}\ [\textsc{Red.Res}] \qquad \frac{P \to Q}{\mathbf{E}\{P\} \to \mathbf{E}\{Q\}}\ [\textsc{Red.Context}]$$

$$\frac{P \equiv P' \qquad P' \to Q' \qquad Q' \equiv Q}{P \to Q}\ [\textsc{Red.Proc.Equiv}]$$

$$\frac{\mathcal{S}_1 \equiv \mathcal{S}_1' \qquad \mathcal{S}_1' \to \mathcal{S}_2' \qquad \mathcal{S}_2' \equiv \mathcal{S}_2}{\mathcal{S}_1 \to \mathcal{S}_2}\ [\textsc{Red.Top.Equiv}]$$

FIG. 5.8 – Réduction : règles de calcul

$$\frac{r \notin dln(P) \qquad r \in dln(Q)}{a(P \mid r\widetilde{V})[Q] \to a(P)[Q \mid r\widetilde{V}]} \text{ [Red.Mess.Boss.To.Cont]}$$

$$\frac{r \in dln(P) \qquad r \notin dln(Q)}{a(P)[Q \mid r\widetilde{V}] \to a(P \mid r\widetilde{V})[Q]} \text{ [Red.Mess.Cont.To.Boss]}$$

Fig. 5.9 – Réduction : règles de routages pour les messages locaux

$$\frac{r \in dln(P)}{a(P \mid \overline{a}.r\widetilde{V})[Q] \to a(P \mid r\widetilde{V})[Q]} \text{ [Red.Addr.Boss.Final]}$$

$$\frac{r \in dln(Q)}{a(P)[Q \mid \underline{a}.r\widetilde{V}] \to a(P)[Q \mid r\widetilde{V}]} \text{ [Red.Addr.Cont.Final]}$$

$$\frac{b \in doms(P) \cup doms(Q) \cup \{a\}}{a(P)[Q] \mid \overline{\underline{b}}.r\widetilde{V} \to a(P \mid \mathbf{i}(\overline{\underline{b}}, r, \widetilde{V}))[Q]} \text{ [Red.Addr.In]}$$

$$\frac{b \notin doms(P) \qquad (b \in doms(Q) \vee \overline{\underline{b}} = \underline{a}) \qquad \overline{\underline{b}} \neq \overline{a}}{a(P \mid \overline{\underline{b}}.r\widetilde{V})[Q] \to a(P)[Q \mid \overline{\underline{b}}.r\widetilde{V}]} \text{ [Red.Addr.Boss.To.Cont]}$$

$$\frac{b \notin doms(P) \cup doms(Q) \qquad b \neq a}{a(P \mid \overline{\underline{b}}.r\widetilde{V})[Q] \to a(P)[Q] \mid \overline{\underline{b}}.r\widetilde{V}} \text{ [Red.Addr.Boss.Out]}$$

$$\frac{b \notin doms(Q) \qquad \overline{\underline{b}} \neq \underline{a}}{a(P)[Q \mid \overline{\underline{b}}.r\widetilde{V}] \to a(P \mid \mathbf{o}(\overline{\underline{b}}.r, \widetilde{V}))[Q]} \text{ [Red.Addr.Cont.Out]}$$

Fig. 5.10 – Réduction : règles de routage pour les messages adressés

L'intuition sous-jacente dans la spécification des règles de réduction est la suivante :
- La règle Red.Beta est la règle de β-réduction classique du λ-calcul.
- Les règles Red.If.Then et Red.If.Else sont les règles classiques pour un branchement conditionnel. Elles utilisent une forme de filtrage très simple.
- La règle Red.Passiv décrit le comportement d'une passivation. Si le nom du domaine courant est identique au nom indiqué par l'opérateur de passivation, le contrôleur et le contenu du domaine sont gelés et passés en argument à la fonction de passivation.
- La règle Red.Res correspond à la règle Join du join calcul. Lorsque les messages correspondant à un filtre d'une définition locale sont présents, le processus gardé associé remplace ces messages après substitution des paramètres formels du filtre par les arguments des messages. La définition reste disponible, comme dans le join calcul.
- La règle Red.Context indique que les réductions sont possibles dans un contexte d'évaluation, c'est à dire à l'intérieur du contrôleur ou du contenu d'un domaine actif (lui-même en contexte d'évaluation), en parallèle avec d'autres processus, dans l'argument d'une application, dans la fonction d'une application si l'argument est une valeur, dans un nuplet, et sous une restriction. Par contre, la réduction est interdite sous une λ-abstraction.
- Les règles Red.Proc.Equiv et Red.Top.Equiv indique que la réduction est stable par équivalence structurelle.
- Les règles de la figure 5.9 spécifient le routage pour un message sur un nom de ressource local. Ces règles permettent au contenu et au contrôleur d'un domaine de communiquer. Aucun message local ne peut franchir la frontière d'un domaine.
- Les règles de la figure 5.10 spécifient comment un message sur un nom adressé $d.r$ est routé. Ces règles sont les seules permettant d'envoyer des valeurs hors d'un domaine. Intuitivement, un message est envoyé au contrôleur ou au contenu spécifié par d. S'il doit entrer dans un domaine ou sortir du contenu du domaine, le contrôleur de ce domaine a possibilité de filtrer le message grâce aux ressources **i** et **o** (règles Red.Addr.In et Red.Addr.Cont.Out). Dans tout les cas, tant que le message n'est pas intercepté il remonte dans l'arbre des domaines jusqu'à être dans un domaine englobant le domaine cible ; il redescend alors vers celui-ci.

Nous décrivons ensuite le système de types garantissant l'unicité des noms de domaines actifs du M-calcul avant de donner des exemples de processus du calcul en section 5.4, afin de décrire les types des ressources et processus de ces exemples.

5.3 Système de types du M-calcul

5.3.1 Présentation du système de types

Le système de types présenté dans cette section garantit l'unicité des noms des domaines actifs d'une configuration. La grammaire des types est donnée en figure 5.11. L'intuition fondamentale sous-jacente aux types est de rassembler dans le type Δ d'un processus le multi-ensemble des noms de domaines actifs présents dans ce processus. A la racine, on exige que ce multi-ensemble soit en fait un ensemble : tout nom du multi-ensemble est présent au plus une fois.

Nous notons $\widetilde{\sigma}$ pour un nuplet de types $\sigma_1, \ldots, \sigma_q$. Les *variables de types* α sont des variables qui peuvent être remplacées par des types de valeurs. Les *variables de types de noms* δ sont des variables représentant des noms de domaines non encore déterminés. Ces dernières sont très semblables aux variables de types de noms du chapitre 4. Les *variables de multi-ensembles* ρ sont des variables qui peuvent être remplacées par des multi-ensembles. Nous notons $\forall\widetilde{\alpha}\widetilde{\delta}\widetilde{\rho}.\sigma$ le schéma de type où les variables de type $\widetilde{\alpha}$, les variables de nom de type $\widetilde{\delta}$ et les variables de multi-ensembles $\widetilde{\rho}$ sont généralisées. Dans la suite nous notons β une variable de type, de type de nom ou de multi-ensemble.

Dans la suite, nous considérons une syntaxe étendue pour le M-calcul où les restrictions pour les noms de ressources possèdent une annotation indiquant le type polymorphe de la ressource : $\nu r : s.P$.

Un nom de domaine a pour type $\mathtt{dom}(w)$. Si le domaine est a, nous avons alors $w = a$. Si le nom du domaine n'est pas déterminé (parce que ce nom est une variable liée par exemple par un filtre dans la définition d'une ressource), nous avons alors $w = \delta$. Enfin, pour des raisons techniques (le lemme C.1.11), nous autorisons les types de la forme $\mathtt{dom}(\emptyset)$ pour un nom de domaine qui ne sera jamais instancié.

Les multi-ensembles de noms de domaines et de variables de multi-ensembles sont notés Δ. L'opérateur d'union sur ces multi-ensemble "," est classique : le nombre d'occurrences de chaque nom de Δ_1, Δ_2 est la somme du nombre d'occurrences de ce nom dans Δ_1 et dans Δ_2. L'intersection sur les multi-ensembles est défini de la manière suivante : le nombre d'occurrences de chaque nom

τ	::=		type
		Δ	processus
	$\mid$	σ	valeur
s	::=		schéma de type
		$\forall\widetilde{\alpha}\widetilde{\delta}\widetilde{\rho}.\sigma$	schéma de type
σ	::=		valeur
		$\texttt{unit}$	type unité
	$\mid$	α	variable de type
	$\mid$	$\texttt{dom}(w)$	type d'un nom de domaine
	$\mid$	$\texttt{dest}$	destination
	$\mid$	$\sigma_1, \ldots, \sigma_q$	nuplet
	$\mid$	$\sigma \to \tau$	fonction
	$\mid$	$\langle\sigma\rangle_\Delta$	ressource
w	::=		nom de domaine dans les types
		a	nom de domaine
	$\mid$	δ	variable de type de nom
	$\mid$	$\emptyset$	domaine inexistant
Δ	::=		multi-ensemble de noms de domaines
		$\emptyset$	multi-ensemble vide
	$\mid$	ρ	variable de multi-ensemble
	$\mid$	δ	variable de type de nom
	$\mid$	a	nom de domaine
	$\mid$	Δ, Δ	union

FIG. 5.11 – Types : Syntaxe

dans $\Delta_1 \cap \Delta_2$ est le minimum du nombre d'occurrences de ce nom dans Δ_1 et Δ_2. Un multi-ensemble est inclus dans un autre $\Delta_1 \subseteq \Delta_2$ si pour tout nom le nombre d'occurrences dans Δ_1 est inférieur ou égal au nombre d'occurrences de ce nom dans Δ_2. Nous notons $\Delta_1 - \Delta_2$ le multi-ensemble dont le nombre d'occurrence de chaque nom est égal au nombre d'occurrences dans Δ_1 moins celui dans Δ_2 (ce nombre est nul si Δ_2 contient plus d'occurrences du nom que Δ_1). Par exemple, nous avons $\rho, \rho, a, b, b - \rho, a, a, b = \rho, b$.

Nous définissons une notion de sous-typage $\leq$ qui est une extension de la notion d'inclusion des multi-ensembles sur les types :

$$
\begin{array}{rcl}
\Delta \leq \Delta' & \Leftarrow & \Delta \subseteq \Delta' \\
\texttt{unit} \leq \texttt{unit} & & \\
\texttt{dest} \leq \texttt{dest} & & \\
\texttt{dom}(w) \leq \texttt{dom}(w) & & \\
\alpha \leq \alpha & & \\
\widetilde{\sigma_i}^{i\in[1..n]} \leq \widetilde{\sigma'_i}^{i\in[1..n]} & \Leftarrow & (\sigma_i \leq \sigma'_i)^{i\in[1..n]} \\
\sigma \to \tau \leq \sigma' \to \tau' & \Leftarrow & \sigma' \leq \sigma \text{ et } \tau \leq \tau' \\
\langle\sigma\rangle_\Delta \leq \langle\sigma'\rangle_{\Delta'} & \Leftarrow & \sigma' \leq \sigma \text{ et } \Delta \leq \Delta' \\
\langle\sigma\rangle_\Delta \leq \sigma' \to \Delta' & \Leftarrow & \sigma' \leq \sigma \text{ et } \Delta \leq \Delta'
\end{array}
$$

Intuitivement, cette relation de sous-typage indique qu'un processus possédant moins de domaines actifs peut remplacer un processus possédant plus de domaines actif, et qu'un nom de ressource (ayant pour type $\langle\sigma\rangle_\Delta$) peut être utilisé quand une fonction (ayant pour type $\sigma \to \Delta$) est attendue. Nous détaillons cette discussion en section 5.3.3.

Nous définissons l'opérateur symétrique $\sqcup$ sur les types comme suit :

$$\begin{array}{rcl}
a, \Delta \sqcup a, \Delta' & = & a, (\Delta \sqcup \Delta') \\
\rho, \Delta \sqcup \rho, \Delta' & = & \rho, (\Delta \sqcup \Delta') \\
\Delta \sqcup \Delta' & = & \Delta, \Delta' \text{ si } \Delta \cap \Delta' = \emptyset \\
\mathtt{unit} \sqcup \mathtt{unit} & = & \mathtt{unit} \\
\mathtt{dest} \sqcup \mathtt{dest} & = & \mathtt{dest} \\
\mathtt{dom}(w) \sqcup \mathtt{dom}(w) & = & \mathtt{dom}(w) \\
\alpha \sqcup \alpha & = & \alpha \\
\widetilde{\sigma} \sqcup \widetilde{\sigma}' & = & \widetilde{(\sigma \sqcup \sigma')} \text{ pour des nuplets de même taille} \\
\sigma \to \tau \sqcup \sigma \to \tau' & = & (\sigma \sqcap \sigma') \to (\tau \sqcup \tau') \\
\langle \sigma \rangle_\Delta \sqcup \langle \sigma' \rangle_{\Delta'} & = & \langle \sigma \sqcap \sigma' \rangle_{\Delta \sqcup \Delta'} \\
\langle \sigma \rangle_\Delta \sqcup \sigma' \to \Delta' & = & (\sigma \sqcap \sigma') \to (\Delta \sqcup \Delta')
\end{array}$$

Tous les autres cas sont indéfinis. En ce qui concerne les multi-ensembles, $\Delta_1 \sqcup \Delta_2$ est en fait le multi-ensemble dont le nombre d'occurrence de chaque nom est le maximum du nombre d'occurrences de ce nom dans Δ_1 et dans Δ_2.

Nous définissons l'opérateur symétrique $\sqcap$ sur les types comme suit :

$$\begin{array}{rcl}
\Delta \sqcap \Delta' & = & \Delta \cap \Delta' \\
\mathtt{unit} \sqcap \mathtt{unit} & = & \mathtt{unit} \\
\mathtt{dest} \sqcap \mathtt{dest} & = & \mathtt{dest} \\
\mathtt{dom}(w) \sqcap \mathtt{dom}(w) & = & \mathtt{dom}(w) \\
\alpha \sqcap \alpha & = & \alpha \\
\widetilde{\sigma} \sqcap \widetilde{\sigma}' & = & \widetilde{(\sigma \sqcap \sigma')} \text{ pour des nuplets de même taille} \\
\sigma \to \tau \sqcap \sigma \to \tau' & = & (\sigma \sqcup \sigma') \to (\tau \sqcap \tau') \\
\langle \sigma \rangle_\Delta \sqcap \langle \sigma' \rangle_{\Delta'} & = & \langle \sigma \sqcup \sigma \rangle_{\Delta \sqcap \Delta'} \\
\langle \sigma \rangle_\Delta \sqcap \sigma' \to \Delta' & = & \langle \sigma \sqcup \sigma' \rangle_{\Delta \sqcap \Delta'}
\end{array}$$

Tous les autres cas sont indéfinis. En ce qui concerne les multi-ensembles, $\Delta_1 \sqcap \Delta_2$ est en fait le multi-ensemble dont le nombre d'occurrence de chaque nom est le minimum du nombre d'occurrences de ce nom dans Δ_1 et dans Δ_2.

Nous notons Γ pour un environnement de typage, c'est à dire une association finie entre des noms et des types ou des schémas de types, telle que tout nom est associé au plus une fois.

Nous définissons l'ensemble des variables libres de multi-ensembles *fsv* comme étant :

$$\begin{array}{rcl}
\mathit{fsv}(\emptyset) & = & \emptyset \\
\mathit{fsv}(\rho) & = & \{\rho\} \\
\mathit{fsv}(a) & = & \emptyset \\
\mathit{fsv}(\Delta, \Delta') & = & \mathit{fsv}(\Delta) \cup \mathit{fsv}(\Delta')
\end{array}$$

Nous définissons également l'ensemble des variables de types de noms libres *fwv* comme étant :

$$\begin{array}{rcl}
\mathit{fwv}(\emptyset) & = & \emptyset \\
\mathit{fwv}(\delta) & = & \{\delta\} \\
\mathit{fsv}(a) & = & \emptyset \\
\mathit{fsv}(\Delta, \Delta') & = & \mathit{fsv}(\Delta) \cup \mathit{fsv}(\Delta')
\end{array}$$

Nous étendons de manière immédiate *fsv* et *fwv* aux types et schémas de types. Nous définissons également l'ensemble *ftv* pour les types et schémas de types, correspondant aux variables de types qui ne sont pas liées par un $\forall$. Les ensembles *fsv*, *ftv* et *fwv* sont étendus de manière immédiate aux environnements de typage. Nous notons *fv* pour l'union de *fsv*, *ftv* et *fwv*.

C	::=		contexte
		$\cdot : \tau$	trou pour un processus
	$\mid$	$\cdot$	trou pour une définition
	$\mid$	$\cdot : \Delta$	trou pour une configuration
	$\mid$	$\epsilon[C]$	domaine racine
	$\mid$	$\nu r : s.C$	restriction de ressource
	$\mid$	$\nu a.C$	restriction de domaine
	$\mid$	$\lambda x.C$	fonction
	$\mid$	$a(C)[Q]$	contrôleur
	$\mid$	$a(P)[C]$	contenu
	$\mid$	$(C \mid P)$	processus parallèle gauche
	$\mid$	$(P \mid C)$	processus parallèle droit
	$\mid$	$P_1, \ldots, P_q$	nuplet
	$\mid$	$\mathtt{pass}_a\ C$	passivation
	$\mid$	(CQ)	application gauche
	$\mid$	(PC)	application droite
	$\mid$	$[pat = C]P, Q$	valeur testée
	$\mid$	$[pat = V]C, P$	test vrai
	$\mid$	$[pat = V]P, C$	test faux
	$\mid$	$\langle C \rangle$	définition
	$\mid$	C, D	composition de définition gauche
	$\mid$	D, C	composition de définition droite
	$\mid$	$J \triangleright C$	processus d'une règle de réaction

FIG. 5.12 – Contextes typés

Le système de types utilise des jugements de la forme suivante :

$$\begin{aligned} \Gamma &\vdash u : \sigma \\ \Gamma &\vdash C : \tau \\ \Gamma &\vdash P : \tau \\ \Gamma &\vdash D \\ \Gamma &\vdash \mathcal{S} : \Delta \end{aligned}$$

où C est un M-contexte étendu avec un trou typé. Ces contextes sont définis en figure 5.12.

Définition 5.3.1 (Environnement de typage minimal) *Afin de pouvoir correctement typer les ressources définissant les noms* **i** *et* **o**, *nous ne considérons que les environnements de typages contenant les associations suivantes :*

$$\begin{aligned} \mathbf{i} &: \forall \alpha \rho. \langle \mathtt{dest}, \langle \alpha \rangle_\rho, \alpha \rangle_\rho \\ \mathbf{o} &: \forall \alpha \rho. \langle \mathtt{dest}, \langle \alpha \rangle_\rho, \alpha \rangle_\rho \end{aligned}$$

Le système de types est défini par les règles des figures 5.13, 5.14, 5.15 et 5.16. La règle NAME utilise l'opérateur *Inst* qui prend un schéma de type $\forall \widetilde{\alpha} \widetilde{\delta} \widetilde{\rho}.\sigma$ et renvoi un type σ' qui est σ avec les variables de types généralisées $\widetilde{\alpha}$ instanciées par des types de valeurs, les variables de types de noms $\widetilde{\delta}$ instanciés par des w et les variables de multi-ensembles $\widetilde{\rho}$ instanciées par des multi-ensembles. La règle TOP utilise le prédicat *set* qui est un prédicat sur les multi-ensembles Δ qui est vrai lorsque Δ est tel qu'il ne contient aucune variable de multi-ensembles ni de types de noms, et tel que tout nom de domaine présent dans Δ l'est au plus une fois.

Nous commentons à présent les règles de typage. Une première remarque générale sur le système de types est que nous autorisons la présence de plusieurs domaines ayant le même nom tant que ceux-ci ne pourront pas être en contexte d'évaluation. Le type d'un sous-terme nous donne une approximation de la possibilité qu'un domaine portant un certain nom se retrouve en contexte d'évaluation.

La règle de typage NAME vérifie que les noms de domaines présents dans la substitution instanciant le schéma de type sont bien déclarés dans l'environnement de typage.

La règle de typage ADDR indique que le type d'une ressource adressée est un type fonctionnel. Intuitivement, la motivation pour ce choix (qui est discuté plus en détails en section 5.3.3) est d'interdire les ressources adressées de la forme $a.b.r$.

$$\frac{u : s = \forall\widetilde{\beta}.\sigma \in \Gamma \qquad \sigma\theta = Inst(s) \qquad fn(ran(\theta)) \subseteq dom(\Gamma)}{\Gamma \vdash u : \sigma\theta}\ [\textsc{Name}] \qquad \frac{}{\Gamma \vdash \mathbf{0} : \emptyset}\ [\textsc{Nil}]$$

$$\frac{}{\Gamma \vdash () : \mathtt{unit}}\ [\textsc{Void}] \qquad \frac{\Gamma \vdash \mathbf{a} : \mathtt{dom}(w)}{\Gamma \vdash \overline{\mathbf{a}} : \mathtt{dest}}\ [\textsc{Dest.Boss}] \qquad \frac{\Gamma \vdash \mathbf{a} : \mathtt{dom}(w)}{\Gamma \vdash \underline{\mathbf{a}} : \mathtt{dest}}\ [\textsc{Dest.Cont}]$$

$$\frac{\Gamma \vdash d : \mathtt{dest} \qquad \Gamma \vdash \mathbf{r} : \langle\sigma\rangle_\Delta}{\Gamma \vdash d.\mathbf{r} : \sigma \to \Delta}\ [\textsc{Addr}]$$

$$\frac{\Gamma + x : \sigma \vdash P : \tau \qquad x \notin dom(\Gamma) \qquad fn(\sigma) \subseteq dom(\Gamma)}{\Gamma \vdash \lambda x.P : \sigma \to \tau}\ [\textsc{Fun}]$$

$$\frac{(\Gamma \vdash P_i : \sigma_i)^{i\in[1..q]}}{\Gamma \vdash P_1, \ldots, P_q : (\sigma_1, \ldots, \sigma_q)}\ [\textsc{Tuple}] \qquad \frac{\Gamma \vdash \mathbf{a} : \mathtt{dom}(w) \qquad \Gamma \vdash P : \Delta_1 \qquad \Gamma \vdash Q : \Delta_2}{\Gamma \vdash \mathbf{a}(P)[Q] : w, \Delta_1, \Delta_2}\ [\textsc{Dom}]$$

$$\frac{\Gamma \vdash P : \Delta_1 \qquad \Gamma \vdash Q : \Delta_2}{\Gamma \vdash P \mid Q : \Delta_1, \Delta_2}\ [\textsc{Par}]$$

$$\frac{\begin{array}{c}\Gamma + r : \forall\widetilde{\beta}.\langle\sigma\rangle_\Delta \vdash P : \Delta_1 \\ r \notin dom(\Gamma) \qquad fv(\forall\widetilde{\beta}.\langle\sigma\rangle_\Delta) = \emptyset \qquad fn(\forall\widetilde{\beta}.\langle\sigma\rangle_\Delta) \subseteq dom(\Gamma)\end{array}}{\Gamma \vdash \nu r : \forall\widetilde{\beta}.\langle\sigma\rangle_\Delta.P : \Delta_1}\ [\textsc{Nu.Res}]$$

$$\frac{\Gamma + a : \mathtt{dom}(a) \vdash P : \Delta \qquad a \notin fn(\Gamma) \qquad a \notin (\Delta - a)}{\Gamma \vdash \nu a.P : \Delta - a}\ [\textsc{Nu.Dom}]$$

$$\frac{\begin{array}{c}\Gamma \vdash V : \sigma_V \quad \begin{array}{rl}\text{avec} & \sigma_V = (\mathtt{unit} \to \Delta_1) \to (\mathtt{unit} \to \Delta_2) \to \Delta \\ \text{ou} & \sigma_v = (\mathtt{unit} \to \Delta_1) \to \langle \mathtt{unit} \to \Delta_2\rangle_\Delta\end{array} \\ \Gamma \vdash \mathbf{a} : \mathtt{dom}(w) \qquad \rho_1 \neq \rho_2 \qquad \rho_1 \in \Delta_1 \;\; \rho_2 \in \Delta_2 \qquad \rho_1, \rho_2 \notin fsv(\Gamma) \cup (\Delta - \rho_1, \rho_2)\end{array}}{\Gamma \vdash \mathtt{pass}_{\mathbf{a}}\, V : \Delta - (w, \rho_1, \rho_2)}\ [\textsc{Pass}]$$

$$\frac{\Gamma \vdash P : \sigma_P \qquad \sigma_P = \langle\sigma\rangle_\Delta \text{ avec } \Delta = \tau \text{ ou } \sigma_P = \sigma \to \tau \qquad \sigma' \leq \sigma \qquad \Gamma \vdash Q : \sigma'}{\Gamma \vdash PQ : \tau}\ [\textsc{App}]$$

$$\frac{\Gamma \vdash V : \tau \qquad \Gamma \vdash P : \tau_1 \qquad \Gamma \vdash Q : \tau_2}{\Gamma \vdash [pat = V]P, Q : \tau_1 \sqcup \tau_2}\ [\textsc{Test}] \qquad \frac{\Gamma \vdash D}{\Gamma \vdash \langle D\rangle : \emptyset}\ [\textsc{Def}]$$

FIG. 5.13 – Règles de typage pour les processus

$$\frac{\Gamma \vdash D_1 \qquad \Gamma \vdash D_2}{\Gamma \vdash D_1, D_2}\ [\textsc{And}] \qquad\qquad \frac{}{\Gamma \vdash \top}\ [\textsc{Def.}\top]$$

$$\frac{\begin{array}{c}(r_i : s_i = \forall\widetilde{\beta_i}.\langle\widetilde{\sigma_i}\rangle_{\Delta_i} \in \Gamma)^{i\in[1..n]} \\ \Delta' \leq \Delta_1, \ldots, \Delta_n \qquad (\widetilde{x_i})^i \cap dom(\Gamma) = \emptyset \qquad \Gamma + \widetilde{x_1} : \widetilde{\sigma_1} + \ldots + \widetilde{x_n} : \widetilde{\sigma_n} \vdash P : \Delta' \\ \forall i \in [1..n].\widetilde{\beta_i} \cap fv(\Gamma) = \emptyset \qquad \forall i, j \in [1..n]^2.i \neq j \implies \widetilde{\beta_i} \cap \widetilde{\beta_j} = \emptyset\end{array}}{\Gamma \vdash r_1\widetilde{x_1} \mid \ldots \mid r_n\widetilde{x_n} \triangleright P}\ [\textsc{Join}]$$

FIG. 5.14 – Règles de typage pour les définitions

$$\frac{\Gamma \vdash P : \Delta \qquad set(\Delta)}{\Gamma \vdash \epsilon[P] : \Delta}\ [\textsc{Top}]$$

$$\frac{\begin{array}{c}\Gamma + r : \forall\widetilde{\beta}.\langle\sigma\rangle_\Delta \vdash \mathcal{S} : \Delta' \\ r \notin dom(\Gamma) \qquad fv(\forall\widetilde{\beta}.\langle\sigma\rangle_\Delta) = \emptyset \qquad fn(\forall\widetilde{\beta}.\langle\sigma\rangle_\Delta) \subseteq dom(\Gamma)\end{array}}{\Gamma \vdash \nu r : \forall\widetilde{\beta}.\langle\sigma\rangle_\Delta.\mathcal{S} : \Delta'}\ [\textsc{Top.Nu.Res}]$$

$$\frac{\Gamma + a : \mathtt{dom}(a) \vdash \mathcal{S} : \Delta \qquad a \notin fn(\Gamma)}{\Gamma \vdash \nu a.\mathcal{S} : \Delta - a}\ [\textsc{Top.Nu.Dom}]$$

FIG. 5.15 – Règles de typage pour les configurations

La règle FUN est classique, à l'exception de la condition sur les noms libres du type de l'argument. Cette condition technique nous permet de nous assurer que les noms de domaines présents dans le type de l'argument sont des noms connus de Γ.

Les règles NU.RES et TOP.NU.RES (figure 5.15) précisent le schéma de type associé à un nom de ressource. Ce schéma est en adéquation avec celui indiqué dans la règle JOIN. De plus, ces règles vérifient que les noms libres du schéma de type sont bien déclarés dans Γ. La règle de typage NU.DOM vérifie que le nom a est présent au plus une fois dans Δ. Cette vérification est nécessaire puisque l'évaluation est possible sous une restriction. Ceci nous garantit également que le nom a n'est pas libre dans la conclusion de la règle de typage.

La règle de typage PASS requiert que les variables ρ_1 et ρ_2 soient fraîches. Intuitivement, nous justifions ce choix par la forme de la règle de typage PASS qui suggère fortement que ρ_1 et ρ_2 sont d'une certaine manière généralisées (comme c'est suggéré par la preuve du cas RED.PASSIV du théorème de stabilité de la typabilité par réduction, page 173). Par conséquent, nous voulons que V soit complètement spécifié au moment du typage de la passivation, et nous interdisons donc le typage de termes de la forme $\lambda f.\mathtt{pass}_a\ f$. La règle de typage PASS requiert également que ρ_1 et ρ_2 soient présents au plus une fois dans Δ, ce qui est justifié par le fait que ρ_1 sera substitué par le multi-ensemble représentant le contrôleur de a, et ρ_2 par celui représentant le contenu de a lors du typage suivant la passivation de a. Pour conserver la propriété d'unicité des noms des domaines actifs, il faut donc que la fonction V soit linéaire en ρ_1 et ρ_2. La condition sur la forme du type de la fonction de passivation V est due au fait que nous restreignons notre utilisation du sous-typage, et devons donc prendre en compte les différents types acceptables que V peut avoir.

De manière similaire à la règle PASS, la règle APP doit prendre en compte les différentes formes que peuvent prendre le type de la fonction (ou du nom de ressource) appliqué. Cette règle est une des deux règles où l'utilisation du sous-typage est autorisée.

Nous remarquons que la règle de typage TEST ne s'applique que si le type $\tau_1 \sqcup \tau_2$ existe. Nous remarquons également que nous ne cherchons pas à typer l'adéquation du filtre de valeurs et de la valeur, puisque ce filtrage simpliste ne lie aucune variable.

Afin de conserver une propriété de typage complètement dirigé par la syntaxe, la relation de sous-typage $\leq$ n'est utilisée que dans la règle d'application fonctionnelle APP et dans la règle JOIN. En ce qui concerne cette dernière, nous justifions ce choix par le fait que le type des ressources est explicitement spécifié et correspond ainsi à une borne supérieur sur les domaines actifs que la ressource crée. Le choix d'un système de types dirigé par la syntaxe simplifie les preuves des propriétés de ce système, et facilite la construction d'un algorithme correspondant au système.

$$\frac{}{\Gamma \vdash (\cdot : \tau) : \tau}\ [\textsc{Proc.Hole}] \qquad \frac{}{\Gamma \vdash \cdot}\ [\textsc{Def.Hole}] \qquad \frac{set(\Delta)}{\Gamma \vdash (\cdot : \Delta) : \Delta}\ [\textsc{Top.Hole}]$$

FIG. 5.16 – Règles de typage pour les contextes typés

La règle JOIN peut *a priori* sembler complexe mais elle est très similaire à la règle de typage équivalente du join calcul. En effet, le processus gardé est typé dans un environnement étendu par les paramètres formels associés aux types correspondant aux types des arguments des ressources, et le type obtenu doit être un sous-type de la borne supérieure donnée par les types des résultats des ressources. Les deux dernières conditions vérifient que la règle de réaction satisfait la généralisation spécifiée par le schéma de type des ressources :

- les variables généralisées ne sont pas libres dans l'environnement de typage ;
- les variables généralisées sont présentes au plus dans un type de ressource (cette condition est similaire à celle de [FMLR00]).

La règle TOP est une règle clé du système, puisqu'elle vérifie que le type du processus du domaine racine est tel que tout nom de domaine soit présent au plus une fois. C'est donc cette règle, avec la règle NU.DOM qui prend en compte les noms cachés par une restriction, qui garantit l'unicité des noms de domaines actifs.

5.3.2 Sûreté du typage

Nous décrivons maintenant les garanties que nous donne le système de types du M-calcul. Les preuves de ces résultats se trouvent en appendice C.

Nous procédons de manière très classique. Nous définissons tout d'abord une notion d'environnement bien formé, qui établit la forme des types autorisés dans l'environnement de typage, et requiert que tout nom de domaine utilisé dans les types soit associé dans l'environnement de typage.

Définition 5.3.2 (Environnement de typage bien formé) *Un environnement de typage est dit* bien formé *s'il ne contient que des associations de la forme* $r : \forall\widetilde{\beta}.\langle\sigma\rangle_\Delta$ *et* $a : \mathtt{dom}(a)$*, et si* $fn(\Gamma) = dom(\Gamma)$*.*

Nous remarquons que les associations pour les canaux spéciaux **i** et **o** satisfont la condition de bonne formation.

Nous établissons ensuite la propriété de stabilité du typage par équivalence structurelle et par réduction.

Lemme 5.3.3 (Stabilité du typage pour $\equiv$) *Soit* $\Gamma \vdash \mathcal{S} : \Delta$ *une dérivation de typage avec* Γ *bien formé. Si nous avons* $\mathcal{S} \equiv \mathcal{S}'$*, nous avons alors une dérivation* $\Gamma \vdash \mathcal{S}' : \Delta$*. Cette propriété est également satisfaite pour les processus.*

Théorème 6 (Stabilité du typage) *Soit* $\Gamma \vdash \mathcal{S}$ *une dérivation de typage avec* Γ *bien formé. Si* $\mathcal{S} \to \mathcal{S}'$*, il existe alors un* Δ' *tel que* $\Delta' \leq \Delta$ *et* $\Gamma \vdash \mathcal{S}' : \Delta'$*. La propriété similaire est également vrai pour les processus.*

La propriété d'être typable est ainsi un invariant du calcul par rapport à la relation de réduction. Nous établissons maintenant une situation d'échec, correspondant à la présence de plusieurs domaines actifs portant le même nom non restreint en contexte d'évaluation.

Définition 5.3.4 (Echec) *Nous disons qu'un domaine* a *est libre et actif dans* P *si il est présent en contexte d'évaluation dans* P *et qu'il n'est pas sous une restriction de* a*.*

Nous disons qu'une configuration $\mathcal{S}$ *a échoué quand* $\mathcal{S}$ *s'écrit* $\mathbf{E}\{P\}$ *avec* $\mathbf{E}$ *étant un contexte d'évaluation et* P *ayant deux domaines libres et actifs portant le même nom.*

Nous pouvons maintenant établir que toute configuration bien typée n'a pas échoué.

Théorème 7 (Sûreté du typage) *Si nous avons une dérivation de typage* $\Gamma \vdash \mathcal{S} : \Delta$ *avec* Γ *bien formé, alors la configuration* $\mathcal{S}$ *n'a pas échoué.*

Nous réunissons les théorèmes 6 et 7 pour en déduire que toute configuration bien typée ne pourra jamais échouer : quelques soient les réductions ayant lieu, dans tout contexte d'évaluation, tout domaine libre et actif porte un nom unique.

5.3.3 Discussion du système de types

Nous discutons à présent quelques caractéristiques de notre système de types et décrivons ses forces et faiblesses.

Linéarité

L'objectif premier de notre système est de garantir l'unicité des noms des domaines actifs alors que nous autorisons des fonctions et définitions de ressources pouvant prendre des processus gelés en argument. Une approche immédiate est d'exiger que toutes les fonctions et définitions utilisent de manière linéaire leurs arguments. Malheureusement, nos premières tentatives dans cette direction nous ont suggéré qu'un tel système serait bien trop restrictif et complexe. En fait, notre système ne garantit l'unicité que pour les domaines en *contexte d'évaluation* (comme cela est décrit dans le théorème 7). Ceci nous permet d'obtenir un système de types plus simple, acceptant il est vrai des fonctions de la forme $\lambda.a(\mathbf{0})[\mathbf{0}] \mid a(\mathbf{0})[\mathbf{0}]$ tant que celles-ci ne sont pas appliquées. Intuitivement, le type d'un processus est une approximation conservative de l'ensemble des domaines actifs qu'il contient et pourra contenir, ou de la valeur qu'il pourra devenir. Nous pouvons ainsi nous contenter de vérifier la linéarité des noms de domaines seulement à la racine (règle Top) et pour les restrictions de noms de domaines (règle Nu.Dom). En ce qui concerne ce deuxième point, cette vérification est nécessaire puisque l'on peut calculer sous une restriction qui pourrait cacher plusieurs domaines actifs portant le même nom. Dans les exemples ci-après, nous ne considérons que des processus en contexte d'évaluation.

Prenons par exemple la fonction $\lambda x.x() \mid x()$. Cette fonction prend nécessairement un processus gelé et le relance deux fois, en parallèle. Nous autorisons bien entendu de telles fonctions, qui peuvent être appliquées à tout processus gelé ne contenant pas de domaine actif libre, comme :

$$(\lambda x.x() \mid x())\lambda.\mathbf{0}$$

ou

$$(\lambda x.x() \mid x())\lambda.\nu a.a(\mathbf{0})[\mathbf{0}]$$

Dans ces cas, la fonction a le type $(\texttt{unit} \to \emptyset) \to \emptyset$. Par contre, le système de types n'accepterait pas que l'on applique cette fonction au processus gelé $\lambda.a(\mathbf{0})[\mathbf{0}]$.

Considérons maintenant la fonction qui lance simplement le processus gelé qu'on lui passe en argument : $\lambda x.x()$. Nous pouvons donner à cette fonction le type $(\texttt{unit} \to \Delta) \to \Delta$. Les fonctions ne pouvant avoir de type polymorphe, considérons plutôt la définition d'une ressource *run* identique : $\langle run(x) \triangleright x() \rangle$. Cette ressource peut avoir le type $\forall \rho.\langle \texttt{unit} \to \rho \rangle_\rho$: elle ne fait que relancer les domaines actifs passés en argument. La vérification que ces domaines n'interfèrent pas avec d'autres domaines de la configuration se fait lors du typage des messages envoyés sur *run*.

Sous-typage

Notre relation de sous-typage $\leq$ se base sur l'observation qu'il est correct de remplacer un processus par un autre contenant moins de domaines actifs sans casser la propriété d'unicité des noms des domaines actifs. Nous étendons bien entendu cette notion à tous les types de manière traditionnelle. Un autre aspect de la relation de sous-typage est l'observation qu'un nom de canal peut être utilisé comme une fonction alors que l'inverse n'est pas vrai. En effet, l'envoi de message consiste tout simplement à appliquer le nom du canal à ses arguments. Ainsi, on peut utiliser un canal à la place d'une fonction, tant que leurs types sont cohérents (règle de typage Fun). Par contre, un canal peut être également adressé : on peut lui adjoindre une destination (règle de typage Addr), ce qui est impossible avec une fonction. Les ressources sont donc des sous-types des fonctions. Comme on ne peut que appliquer un canal adressé pour envoyer un message, les canaux adressés ont un type fonctionnel.

Nous illustrons ceci en reprenant l'exemple de la ressource *run* décrite ci-avant. Cette ressource peut bien entendu recevoir un processus gelé en argument (un processus gelé est une fonction), comme $run(\lambda.a.(\mathbf{0})[\mathbf{0}])$. On peut également lui passer un nom de ressource, comme dans $run(create_a)$ en prenant $\langle create_a() \triangleright a(\mathbf{0})[\mathbf{0}] \rangle$.

Les règles de typage utilisent la relation de sous-typage de manière algorithmique : elle n'intervient que dans les règle App et Join. Le cas de la règle Join mérite d'être discuté. Chaque nom de ressource possède un type décrivant entre autres les domaines actifs qu'elle peut créer. La règle Join vérifie que l'ensemble des domaines actifs ainsi spécifiés dans les ressources définies par le filtre est plus grand que l'ensemble des domaines effectivement créés par le processus gardé. Chaque message

porte ainsi une responsabilité partielle dans la comptabilité des domaines actifs. De plus, l'approximation supplémentaire introduite en ne demandant qu'une inclusion des noms des domaines créés dans les noms déclarés nous donne une certaine flexibilité. Ainsi, le port spécial **i** possède le type $\forall\alpha\rho.\langle \mathtt{dest}, \langle\alpha\rangle_\rho, \alpha\rangle_\rho$ qui indique que ce canal crée *au plus* les canaux créés par le canal passé en argument. On peut ainsi donner l'implémentation $\langle \mathbf{i}(dest, chan, args) \triangleright dest.chan\ args\rangle$ à ce canal, lorsque l'on considère une propagation transparente des messages, ou bien l'implémentation $\langle \mathbf{i}(dest, chan, args) \triangleright \mathbf{0}\rangle$, lorsque l'on désire intercepter tous les messages. Cette deuxième implémentation est bien typée puisque les domaines créés (ici aucun) sont bien inclus dans les domaines déclarés.

Polymorphisme et types dépendants

Bien que notre système de types n'utilise pas des types dépendants, il les simule par le polymorphisme et les *variables de types de noms* (les variables de types représentant des noms de domaines), les noms de domaines pouvant être présents dans les types. Par exemple la ressource *new* dans la définition suivante : $\langle new(x) \triangleright x(\mathbf{0})[\mathbf{0}]\rangle$ peut avoir le type $\forall\delta.\langle \mathtt{dom}(\delta)\rangle_\delta$. Ce type indique que le canal *new* attend un nom de domaine et crée un domaine portant ce nom. Il existe ainsi une dépendance entre le nom du domaine passé en argument et le type du processus obtenu. Cependant, notre système est moins puissant que s'il utilisait les types dépendants, puisque nous utilisons un polymorphisme à la ML, c'est à dire prénexe et avec les arguments des fonctions et canaux nécessairement monomorphes. Ainsi, nous ne pouvons pas typer le processus $(\lambda f.f\ a \mid f\ b)(\lambda x.x(\mathbf{0})[\mathbf{0}])$.

Nous remarquons par contre que le polymorphisme associé aux variables de types de noms et aux variables de multi-ensembles est suffisamment puissant pour typer des processus de la forme :

$$\langle create(x, boss, cont) \triangleright x(boss())[cont()]\rangle$$

avec create : $\forall\delta\rho_1\rho_2.\langle \mathtt{dom}(\delta), \mathtt{unit} \to \rho_1, \mathtt{unit} \to \rho_2\rangle_{\delta,\rho_1,\rho_2}$.

Passivation subjective

L'opérateur de passivation subjective `pass` est très puissant, puisqu'il permet le contrôle de processus en contexte d'évaluation. Cependant, le typage de cet opérateur montre quelques limites du système. Considérons par exemple le processus :

$$a\left(\mathtt{pass}_a\ \lambda pq.a\big(\mathbf{0}\big)\big[b(\mathbf{0})[\mathbf{0}]\big]\right)\big[b(\mathbf{0})[\mathbf{0}]\big]$$

L'opérateur $\mathtt{pass}_a$ gèle le domaine a et le relance. Le contrôleur et le contenu du domaine ne sont pas réactivés (ils sont représentés par les variables p et q dans la fonction en argument de $\mathtt{pass}_a$), mais un nouveau domaine b est créé dans le contenu final. Puisque ce domaine est présent dans le contenu initial, on pourrait s'attendre à ce que ce processus soit bien typé (l'ensemble des domaines actifs après réduction est le même qu'avant). Ce n'est cependant pas le cas puisque le système de types ne détecte pas que le contenu qui n'est pas réactivé contient un domaine b. En effet, ce domaine pourrait être envoyé ailleurs (voir par exemple 5.4.4), donc on ne peut avoir la garantie qu'il sera bien détruit. C'est pourquoi le processus $\mathtt{pass}_a\ \lambda pq.a(\mathbf{0})[b(\mathbf{0})[\mathbf{0}]]$ a pour type b, le typage ne pouvant détecter que la réutilisation du contrôleur ou du contenu, et non pas les domaines actifs qu'ils contiennent. Ce processus crée donc nécessairement un nouveau domaine b. Par contre, le processus $\mathtt{pass}_a\ \lambda pq.a(p())[q()]$ a pour type $\emptyset$ puisqu'il ne recrée que des domaines qui ont été passivés. Le processus suivant est par conséquent bien typé :

$$a\left(\mathtt{pass}_a\ \lambda pq.a(p())[q()]\right)[b(\mathbf{0})[\mathbf{0}]]$$

5.4 Discussion et exemples

Cette section donne des motivations pour les différentes constructions du M-calcul, et présente plusieurs exemples accompagnés de leurs types. Nous donnons en particulier des exemples de traduction d'autres calculs distribués dans le M-calcul. Nous soulignons également les limites et extensions possibles du calcul.

5.4.1 Communications transparentes

Un des objectifs du M-calcul est d'adapter les communications complètement transparentes du join calcul distribué de telle sorte qu'un certain contrôle soit possible lors du routage des messages.

En effet, un message adressé à un domaine n'y est pas immédiatement envoyé en une seule étape, mais y parvient petit à petit : chaque étape rapproche le message de sa destination finale tant que celle-ci ne migre pas. Les messages locaux ne peuvent être utilisés que pour les communications à l'intérieur du contrôleur ou du contenu, ou pour communiquer entre le contrôleur et le contenu, donc ne sont pas impliqués par le routage. Ce routage à base d'étapes élémentaires est réalisé par les règles de réduction de la figure 5.10, et donne la possibilité aux contrôleurs d'intercepter un message venant de l'extérieur, ou un message sortant de leur contenu.

Cependant, il est toujours possible de spécifier un contrôleur qui n'intercepte pas les messages, ce qui permet d'obtenir des configurations où le routage reste transparent, c'est à dire où il n'est pas nécessaire de spécifier explicitement le chemin que doivent prendre les messages pour atteindre leur destination. Tout contrôleur contenant le processus *Fwd* suivant réalise ce routage transparent :

$$Fwd = \langle \mathbf{i}(dest, chan, args) \triangleright dest.chan\ args\ ;\ \ \mathbf{o}(dest, chan, args) \triangleright dest.chan\ args \rangle$$

Nous remarquons que ce processus est bien typé dans un environnement contenant les associations $\mathbf{i} : \forall \alpha\rho.\mathtt{dest}, \langle\langle\alpha\rangle_\rho, \alpha\rangle_\rho, \mathbf{o} : \forall \alpha\rho.\mathtt{dest}, \langle\langle\alpha\rangle_\rho, \alpha\rangle_\rho$ Les messages entrants (règle Red.Addr.In) et les messages sortants du contenu (règle Red.Addr.Cont.Out), sont tout simplement réémis dans le contrôleur par une étape Red.Res, et seront ensuite routés selon les règles de réduction Red.Addr.Boss.To.Cont ou Red.Addr.Boss.Out si le message doit encore voyager, ou par les règles Red.Addr.Boss.Final ou Red.Addr.Cont.Final si le message a atteint sa destination.

Nous soulignons le fait que ce contrôleur élémentaire ne conserve pas d'état et n'utilise aucune information de routage : il se base entièrement sur les règles de réduction pour effectuer le routage. Nous remarquons également que ce mécanisme de routage continue à fonctionner dans le cas de processus mobiles, c'est à dire même si la destination initiale s'est déplacée. Ainsi, due à la nature asynchrone du M-calcul, il est par exemple possible que après une étape Red.Addr.In pour un message ayant pour destination un sous-domaine du contenu, et avant que le message ne soit effectivement envoyé dans le contenu, ce dernier ait changé et que la condition pour utiliser la règle Red.Addr.Boss.To.Cont ne soit plus satisfaite (c'est le cas si par exemple la destination a disparu ou est en train de migrer, voir la section 5.4.4). Le contrôleur n'étant pas au courant de ce fait, il libère quand même le message. Dans ce cas, l'étape suivante de réduction pour le message est une règle Red.Addr.Boss.Out et le message remonte dans l'arbre des domaines, comme attendu. La réduction suivante illustre ce cas :

$$\begin{array}{lcll}
\overline{b}.r\widetilde{V} \mid a(Fwd)[Q] & \to & a(\mathbf{i}(\overline{b}, r, \widetilde{V}) \mid Fwd)[Q] & \text{par Red.Addr.In avec } b \in doms(Q) \\
& \to^* & a(\mathbf{i}(\overline{b}, r, \widetilde{V}) \mid Fwd)[R] & Q \text{ évolue en } R \text{ avec } b \notin doms(R) \\
& \to^* & a(\overline{b}.r\widetilde{V} \mid Fwd)[R] & \text{par définition de } Fwd \\
& \to & \overline{b}.r\widetilde{V} \mid a(Fwd)[R] & \text{par Red.Addr.Boss.Out}
\end{array}$$

Nous remarquons que la phase finale du routage a lieu lorsque le message local correspondant au message adressé est libéré (règles Red.Addr.Boss.Final et Red.Addr.Cont.Final). Ces règle vérifient la présence de la ressource et ne libèrent le message que dans ce cas. Dans le cas où la ressource n'est pas présente, le message reste tout simplement en attente à l'intérieur du contrôleur ou du contenu du domaine destination. La seule règle pouvant supprimer une ressource étant la règle Red.Passiv, nous regardons maintenant son influence sur ces messages parvenus à destination. Les seules possibilités qu'a la fonction V du processus $\mathtt{pass}_a\ V$ sont soit de réémettre le contrôleur et le contenu, soit de supprimer l'un des deux, soit de les supprimer tous les deux. Dans tous les cas, le destin du message local et de la ressource associée sont le même : la ressource et le message sont soit tous les deux présents après la réduction, soit ils ont tous les deux disparu. Ainsi, on ne peut subrepticement envoyer un message au contenu sans que le contrôleur n'ait la chance de l'intercepter : si on essaie de l'envoyer au contrôleur, le message ne sera libéré que si une ressource est présente pour ce nom dans le contrôleur, et la règle Red.Mess.Boss.To.Cont garantit que ce message local ne sera envoyé dans le contenu que si la ressource correspondante du contrôleur disparaît, ce qui implique la disparition du message.

La première version de ce calcul considérait des message adressés aux domaines, et non pas aux contrôleurs et contenus directement. La motivation derrière ce changement est la suivante : avec des messages adressés aux domaines, il était difficile de contrôler que tout message franchissant une frontière de domaine était filtré et qu'un message parvenu à destination était effectivement lié à une ressource locale. De plus, une telle distinction entre envoi d'un message au contrôleur ou au contenu permet de simplifier certains codages de calculs, comme le codage du join calcul dynamique par exemple.

5.4.2 Ressources et domaines de première classe, fonctions et choix conditionnel

Nous soulignons maintenant quelques choix qui ont été réalisés lors de la conception du calcul.

Contrairement au join calcul dynamique du chapitre 4, les noms de ressources ne sont pas complètement des valeurs de première classe : il est impossible de recevoir un nom de ressource puis de le définir. Nous rappelons que le prix à payer pour que les ressources du join calcul dynamique soient complètement de première classe est le polymorphisme : toute ressource redéfinissable est nécessairement monomorphe. Ce choix peut être acceptable dans le join calcul dynamique puisque des canaux statiques, donc non redéfinissables, sont également présents et peuvent être polymorphes. Le M-calcul ne possédant qu'une seule famille de ressources, le polymorphisme de ces ressources a été considéré plus important. Il serait cependant possible d'intégrer cette fonctionnalité, au prix de quelques changements du calcul. Tout d'abord, il serait nécessaire de distinguer les ressources que l'on peut redéfinir bien qu'elles soient passées en argument d'une fonction des ressources actuelles du calcul. Un simple type de ressources redéfinissables ne suffit pas à cela, il est nécessaire de considérer une nouvelle classe de noms, comme dans le join calcul dynamique. Ces noms possèdent un type de ressource redéfinissable n'ayant aucun sous-type, par exemple noté $\langle\sigma\rangle^{+}_{\Delta}$, et la règle de typage Join doit les prendre explicitement en compte. Il nous faut également modifier le typage de la restriction des ressources pour considérer les deux cas possibles. Les règles de typage App et Pass doivent être adaptées pour considérer ce nouveau type, à cause de notre traitement algorithmique du sous-typage. Par contre, les modifications à apporter aux preuves de correction sont mineurs. Nous avons cependant choisi de ne pas présenter cette extension, celle-ci compliquant significativement le calcul, et la réservons pour des travaux futurs.

Le traitement des noms de domaines comme valeurs de première classe a par contre nécessité moins de modifications au niveau du calcul, et s'appuie sur un système de types très proche de celui du join calcul dynamique. C'est pourquoi nous l'avons intégré au calcul.

La raison motivant l'ajout de fonctions et du choix conditionnel dans le M-calcul, alors que ces deux constructions sont parfaitement codable dans le join calcul ou le π-calcul, est la possibilité d'exprimer facilement un processus gelé et de réaliser des analyses plus fines. Ce deuxième point concerne principalement le choix conditionnel. Par exemple, le système de types de la section 5.3.1 présente une règle de typage pour le choix conditionnel Test qui prend explicitement en compte le fait qu'une seule des branches ne pourra être prise. Une analyse bien plus complexe aurait été nécessaire pour dériver cette propriété d'un codage de la conditionnelle.

5.4.3 Noms de ressources et liaison dynamique

Comme dans le join calcul dynamique du chapitre 4, plusieurs ressources portant le même nom peuvent être définies dans différents domaines. Il existe donc une notion de liaison dynamique dépendant du choix de la ressource associée à un message. A la différence du join calcul dynamique, les messages non adressés sont purement locaux : ils ne peuvent faire référence uniquement à des ressources définies dans le domaine où ils sont. Cependant, un message local dans un domaine ne possédant aucune ressource pour ce nom n'est pas perdu, puisque de nouvelles ressources peuvent apparaître lors de l'exécution.

Dans un soucis d'implémentabilité, nous nous assurons que le routage des messages adressés est déterministe. Dans la tradition du join calcul distribué, nous supposons que tous les sous-domaines d'un domaine sont exécutés par la même machine, donc que nous pouvons savoir de manière locale quel sont les sous-domaines actifs dans un domaine donné. De plus, notre système de types nous garantissant que tout domaine actif possède un nom unique (voir la section C.1), il est donc possible d'envoyer des messages à un domaine particulier. La destination des messages étant explicitement spécifiable, la *résolution de nom* est donc programmable : il est ainsi possible de traduire le join calcul distribué dans le M-calcul en préfixant tout nom de ressource dans un message par le nom du domaine définissant cette ressource ; il est également possible de traduire une notion de liaison dynamique similaire au join calcul dynamique en conservant dans chaque domaine la liste des ressources localement définies et le nom du domaine père courant (voir la section 5.4.5).

Nous illustrons à présent la liaison dynamique en montrant comment mettre à jour une fonction de bibliothèque. Nous donnons deux solutions, la première se basant sur l'utilisation d'une cellule de référence, la deuxième sur la structure des domaines.

L'utilisation d'une cellule de référence pour représenter la fonction associée à une ressource correspond au codage classique des ressources dynamiques dans le join calcul : le nom de la ressource ne sert qu'à propager le message vers la définition courante. Cependant, à la différence du join calcul, nous pouvons directement stocker dans la cellule de référence la fonction correspondant à

la ressource courante, au lieu de devoir introduire un nouveau nom de canal à chaque fois que la ressource est mise à jour :

$$a(\langle \mathsf{l}\ \widetilde{x} \triangleright L_1 \rangle \mid P(\mathsf{l}))^*[Q] \mid (\overline{a}.\mathsf{update}\ (\mathsf{l}, \lambda\widetilde{x}.L_2)) \rightarrow^* a(\langle \mathsf{l}\ \widetilde{x} \triangleright L_2 \rangle \mid P(\mathsf{l}))^*[Q]$$

en notant $(\langle \mathsf{l}\ \widetilde{x} \triangleright L_i \rangle \mid P(\mathsf{l}))^*$ pour un codage de $(\langle \mathsf{l}\ \widetilde{x} \triangleright L_i \rangle \mid P(\mathsf{l}))$. Ce codage utilise une cellule de référence locale r contenant la fonction courante associée à la ressource l :

$$(\langle \mathsf{l}\ \widetilde{x} \triangleright L \rangle \mid P(\mathsf{l}))^* \quad = \quad \nu r.((r\ \lambda\widetilde{x}.L) \mid \langle r\ f \mid \mathsf{l}\ \widetilde{y} \triangleright f\ \widetilde{y} \mid r\ f \rangle \mid \langle r\ f' \mid (\mathsf{update}\ f'') \triangleright r f'' \rangle)$$

avec $\widetilde{x}$ et $\widetilde{y}$ ayant la même taille (ce qui est garanti par le système de types). Ainsi, un message sur l accède au contenu de la cellule de référence r pour en extraire la fonction courante f et l'appliquer aux arguments de l. La mise à jour de l correspond à la modification du contenu de r.

Si nous supposons que la fonction de bibliothèque a pour type $\mathsf{l} : \widetilde{\sigma} \rightarrow \emptyset$, les types de r et de update devraient être :

$$\begin{array}{rcl} r & : & \langle \widetilde{\sigma} \rightarrow \emptyset \rangle_\emptyset \\ \mathsf{update} & : & \langle \widetilde{\sigma} \rightarrow \emptyset \rangle_\emptyset \end{array}$$

Nous remarquons que la bibliothèque ne peut être polymorphe, puisque la fonction est contenu dans une cellule de référence qui est mise à jour. D'un point de vue technique, c'est le partage des variables de types de r et l, et de r et update, qui empêche la généralisation par hypothèse de la règle de typage Join.

Nous présentons maintenant un autre codage de la même fonctionnalité.

$$\begin{array}{rcl} (\langle \mathsf{l}\ \widetilde{x} \triangleright L \rangle \mid P(\mathsf{l}))^* & = & \nu b.\nu l'. \\ & & \quad \langle \mathsf{update}\ f \triangleright \overline{b}.\mathsf{update}\ f\ ;\ \mathsf{l}\ \widetilde{x} \triangleright \underline{b}.l'\ \widetilde{x} \rangle \\ & & \mid\ b(Fwd \mid \langle \mathsf{update}\ f \triangleright \mathtt{pass}_b\ (\lambda pq.b(p())[\langle l'\ \widetilde{x} \triangleright f\ \widetilde{x} \rangle])\rangle)[\langle l'\ \widetilde{y} \triangleright (\lambda\widetilde{x}.L)\ \widetilde{y} \rangle] \end{array}$$

Ce codage fonctionne en associant à la ressource un nouveau domaine qui n'intercepte pas les messages (grâce au processus *Fwd* décrit en section 5.4.1). Le contrôleur de ce domaine contient la ressource update de mise à jour, et le contenu contient la ressource courante associée à la fonction de bibliothèque. La mise à jour consiste à passiver le domaine et à remplacer le contenu par la nouvelle ressource. Nous remarquons que ce codage ne fonctionne pas si la bibliothèque est supposée fonctionner localement, puisque les ressources de a ne sont pas accessibles localement dans b. Il est aisé de remédier à ce problème en modifiant la définition de la nouvelle ressource en $\langle l'\ \widetilde{x} \triangleright \overline{a}.\mathsf{spawn}\ \lambda.f\ \widetilde{x} \rangle$ et en ajoutant une définition dans le contrôleur de a : $\langle \mathsf{spawn}\ y \triangleright y\ () \rangle$.

Comme dans le précédent codage, les types de l' et update doivent être (en supposant que l ait le type $\widetilde{\sigma} \rightarrow \emptyset$) :

$$\begin{array}{rcl} l' & : & \langle \widetilde{\sigma} \rangle_\emptyset \\ \mathsf{update} & : & \langle \widetilde{\sigma} \rightarrow \emptyset \rangle_\emptyset \end{array}$$

Une fois de plus, il n'est pas possible d'obtenir une ressource polymorphe puisque lors du typage de $\langle l'\ \widetilde{x} \triangleright f\ \widetilde{x} \rangle$, les variables de types sont présentes dans l'environnement par l'association pour $f : \widetilde{\sigma} \rightarrow \emptyset$.

Nous remarquons enfin que ces formes de modification de bibliothèque dynamique sont plus puissantes que dans le join calcul dynamique, puisque nous pouvons facilement utiliser plusieurs bibliothèques différentes (dans le join calcul dynamique, ceci exigeait de définir une bibliothèque par location englobante), et nous ne produisons pas de locations ou définitions qui doivent être glanées par le ramasse-miettes. Par contre, nous ne garantissons pas la présence des ressources.

5.4.4 Exemples de reconfiguration dynamique

Nous décrivons dans cette section d'autres formes de reconfiguration dynamique qui peuvent être modélisées dans le M-calcul.

Création d'un nouveau domaine

Nous considérons ici la possibilité de créer un domaine voisin depuis le contrôleur d'un domaine existant. Nous cherchons ainsi à modéliser le comportement suivant :

$$\overline{a}.n\ (b) \mid a(\langle n\ (x) \triangleright New(x) \rangle \mid P_1)[Q_1] \quad \rightarrow^* \quad b(P_2)[Q_2] \mid a(\langle n\ (x) \triangleright New(x) \rangle \mid P_1)[Q_1]$$

La création d'un nouveau domaine est triviale, la seule difficulté dans le comportement recherché est de créer ce domaine à l'extérieur du domaine effectuant la création. Pour se faire, nous utilisons la passivation :

$$New(x) = \mathtt{pass}_a\ \lambda pq.(x(P_2)[Q_2] \mid a(p())[q()])$$

Le processus $New(x)$ a pour type δ, Δ_2 si Δ_2 est le multi-ensemble des noms des domaines actifs de P_2 et Q_2. Nous pouvons donner à la ressource n le schéma de type $\forall\delta.\langle \mathtt{dom}(\delta)\rangle_{\delta,\Delta_2}$. Pour être bien typé, la configuration doit utiliser n au plus une fois si Δ_2 n'est pas vide ou à chaque fois avec un nouveau nom si Δ_2 est vide, Δ_2 doit être un ensemble et aucun nom de Δ_2 ne devrait être le nom d'un domaine actif dans le reste de la configuration.

Introduction d'un processus dans le contenu d'un domaine

Nous considérons la possibilité d'ajouter un processus dans le contenu d'un domaine. Plus précisément, nous désirons implémenter une ressource $\langle \mathsf{add}\ \ f \triangleright \ldots\rangle$ fournissant le comportement suivant :

$$\overline{a}.\mathsf{add}\,(\lambda.P) \mid a(Fwd \mid \langle \mathsf{add}\ \ f \triangleright Add(a,f)\rangle)[Q] \quad \rightarrow^* \quad a(Fwd \mid \langle \mathsf{add}\ \ f \triangleright Add(a,f)\rangle)[P \mid Q]$$

Le processus $Add(a,f) = \mathtt{pass}_a\ \lambda pq.a(p())[f() \mid q()]$ a le comportement désiré. En effet, nous avons la série de réductions :

$$\begin{array}{rl}
 & \overline{a}.\mathsf{add}\,(\lambda.P) \mid a(Fwd \mid \langle \mathsf{add}\ \ f \triangleright Add(a,f)\rangle)[Q] \\
\rightarrow & a(\mathbf{i}(\overline{a}, \mathsf{add}\,, \lambda.P) \mid Fwd \mid \langle \mathsf{add}\ \ f \triangleright Add(a,f)\rangle)[Q] \\
\rightarrow^* & a(\mathsf{add}\,(\lambda.P) \mid Fwd \mid \langle \mathsf{add}\ \ f \triangleright Add(a,f)\rangle)[Q] \\
\rightarrow & a(Fwd \mid \langle \mathsf{add}\ \ f \triangleright Add(a,f)\rangle \mid \mathtt{pass}_a\ \lambda pq.a(p())[(\lambda.P)() \mid q()])[Q] \\
\rightarrow & (\lambda pq.a(p())[(\lambda.P)() \mid q()])\ \ (\lambda.Fwd \mid \langle \mathsf{add}\ \ f \triangleright Add(a,f)\rangle)\ \ (\lambda.Q) \\
\rightarrow^* & a(Fwd \mid \langle \mathsf{add}\ \ f \triangleright Add(a,f)\rangle)[P \mid Q]
\end{array}$$

Nous utilisons dans l'exemple précédent le processus Fwd de la section 5.4.1. Si nous désirons contrôler plus finement les communications avec l'extérieur, nous pouvons tout simplement décider d'accepter les messages sur $\overline{a}.\mathsf{add}$ et faire autre chose pour les autres messages, en définissant $\mathbf{i}$ de la manière suivante : $\langle \mathbf{i}(dest, chan, args) \triangleright [\overline{a}.\mathsf{add}\ = dest.chan](dest.chan\ args, \ldots)\rangle$.

Il est également possible d'éviter la passivation du domaine en utilisant le processus $Add'(a,f) = (\mathsf{ins}\ f)$ comme définition pour add, en supposant que Q est de la forme $\langle \mathsf{ins}\ f \triangleright f()\rangle \mid Q'$. Par contre, cette deuxième solution demande la collaboration du contenu pour fonctionner.

Dans tous les cas, le type de add ou de ins est $\forall\rho.\langle \mathtt{unit} \rightarrow \rho\rangle_\rho$.

Déplacement d'un processus dans un autre domaine

Le déplacement d'un processus vers un autre domaine est assez simple, puisque nous avons vu comment ajouter un processus à un domaine. Il est simplement nécessaire de commencer par passiver ce processus, puis de l'envoyer sous forme gelée à un domaine définissant la ressource add vue précédemment. Nous prenons comme exemple une passivation subjective, similaire à la primitive go du join calcul distribué, pour déplacer le domaine courant vers un domaine cible. Nous cherchons donc à implémenter le comportement suivant :

$$a(P)[(\mathsf{go}\ u) \mid Q] \rightarrow^* \overline{u}.\mathsf{add}\,(a(P)[Q]^*)$$

en notant $a(P)[Q]^*$ une forme gelée du domaine. Il suffit de définir P comme étant :

$$\begin{array}{rcl}
P & = & (Fwd \mid \langle \mathsf{go}\ u \triangleright Go(a,u)\rangle) \\
Go(a,u) & = & \mathtt{pass}_a\ \lambda pq.\overline{u}.\mathsf{add}\,(\lambda.a(p())[q()])
\end{array}$$

De manière assez surprenante, le processus $Go(a,u)$ a pour type $\emptyset$, puisque ce processus se contente de recréer un domaine qui a été passivé, et n'introduit donc pas de nouveau domaine. La ressource go a donc le type $\forall\delta.\langle \mathtt{dom}(\delta)\rangle_\emptyset$.

Nous remarquons que si la destination u de la migration est un sous-domaine de a (voire a lui-même), alors le message sur $\overline{u}.\mathsf{add}$ ne pourra jamais parvenir à u, puisque celui-ci est gelé dans l'argument du message.

Supprimer un processus du contenu d'un domaine

La ressource go codée ci-dessus implémente une migration subjective, c'est à dire la migration d'un domaine initiée à l'intérieur du domaine (voir la section 4.1.2 pour une discussion entre *subjectif* et *objectif*). Nous considérons maintenant une migration démarrée de l'extérieur du domaine, qui correspond au comportement suivant :

$$\overline{a}.\mathsf{move}\ (u,v) \mid a(P)[Q(u) \mid R] \ \rightarrow^*\ \overline{v}.\mathsf{add}\ (Q^*(u)) \mid a(P)[R]$$

avec $Q(u)$ représentant un processus portant le nom u, c'est à dire un *composant*, et avec $Q^*(u)$ représentant la forme gelée de ce composant. Nous supposons que v n'est pas un sous-domaine de P, ni Q, ni a. Si ce n'est pas le cas, seule l'étape de routage finale est différente.

Nous représentons bien évidemment un composant nommé dans le M-calcul par un domaine, et notons $Q(b)$ pour $b(C_b)[Q]$, avec :

$$C_b = (Fwd \mid \langle \mathsf{go}\ v \triangleright Go(b,v)\rangle)$$

Nous obtenons le comportement désiré en définissant P comme étant :

$$P = (Fwd \mid \langle \mathsf{move}\ (x,y) \triangleright (\overline{x}.\mathsf{go}\ y)\rangle)$$

Nous avons la série de réductions :

$$\begin{array}{rcl}
\overline{a}.\mathsf{move}\ (b,v) \mid a(P)[Q(b) \mid R] & \rightarrow^* & a(\mathsf{move}\ (b,v) \mid P)[Q(b) \mid R] \\
& \rightarrow^* & a(P)[\overline{b}.\mathsf{go}\ v \mid Q(b) \mid R] \\
& \rightarrow^* & a(P)[\overline{v}.\mathsf{add}\ (\lambda.b((\lambda.C_b)())[(\lambda.Q)()]) \mid R] \\
& \rightarrow^* & a(\overline{v}.\mathsf{add}\ (\lambda.b((\lambda.C_b)())[(\lambda.Q)()]) \mid P)[R] \\
& \rightarrow & \overline{v}.\mathsf{add}\ (\lambda.b((\lambda.C_b)())[(\lambda.Q)()]) \mid a(P)[R])
\end{array}$$

Nous avons donc $Q^*(b) = \lambda.b((\lambda.C_b)())[(\lambda.Q)()])$, qui correspond à la forme gelée du domaine $b(C_b)[Q]$. Nous remarquons que dans l'exemple précédent nous utilisons beaucoup le filtrage transparent donné par le processus Fwd. Comme dans les exemples précédents, nous pourrions définir une ressource de filtrage plus fine.

Dans cet exemple, le domaine $a(P)[\cdot]$ peut être considéré comme étant un *component container*, c'est à dire une boîte à composants, comprise dans le sens donné dans [Gro99, Mic98].

De manière très similaire à l'exemple de go, le type de move est $\forall\delta\delta'.\langle \mathtt{dom}(\delta), \mathtt{dom}(\delta')\rangle_\emptyset$.

Composants contrôlables

La notion de contrôle sur un composant de la section précédente peut être déclinée sous la forme de nombreux contrôles différents sur des domaines du M-calcul. Nous donnons l'exemple d'un composant que l'on peut suspendre $Q^i(b)$ portant le nom b et ayant pour comportement Q :

$$\begin{array}{rcl}
Q^i(b) & = & \nu r\ \mathsf{on}.b(Fwd \mid \langle \mathsf{suspend} \mid \mathsf{on} \triangleright Suspend_b; \mathsf{resume} \mid r\ x \triangleright Add(b,x) \mid \mathsf{on}\rangle \mid \mathsf{on})[Q] \\
Suspend_b & = & \mathtt{pass}_b\ \ \lambda pq.b(p() \mid (r\ q))[\mathbf{0}]
\end{array}$$

Nous utilisons la notation on, suspend et resume pour les messages on(), suspend() et resume() qui ne transportent pas d'argument.

Dans cette exemple, le processus Q est gelé dans son état courant à la réception d'un message suspend par son contrôleur, et est stocké dans l'argument du message sur r. Il est réactivé lorsqu'un message resume parvient au contrôleur. Le message sur on permet de ne pas suspendre le processus s'il est déjà suspendu. Nous remarquons également que les messages à destination du contenu y sont envoyés, et attendent que celui-ci soit réactivé pour être consommés par la ressource ciblée, si celle-ci est présente.

On peut donner à la ressource de stockage r le type $\forall\rho.\langle \mathtt{unit} \rightarrow \rho\rangle_\rho$. Le processus $Suspend_b$ a pour type $\emptyset$, puisqu'il recrée le domaine qui est passivé, la réactivation du contenu étant retardée.

Il est également possible d'utiliser une des constructions données à la section 5.4.3 pour définir un composant pouvant évoluer dans le temps $Q^e(b)$ ayant le nom b et le comportement Q, et dont le contrôleur contient une fonction de bibliothèque L qui peut être mise à jour. Une autre possibilité consiste à donner la capacité au composant à changer intégralement son contrôleur, en lui communiquant un nouveau contrôleur par la ressource update :

$$\begin{array}{rcl}
Q^e(b) & = & b(P \mid \langle \mathsf{update}\ f \triangleright Update_b(f)\rangle)[Q] \\
Update_b(f) & = & \mathtt{pass}_b\ \ \lambda pq.b(f())[q()]
\end{array}$$

Dans cet exemple la ressource update peut avoir $\forall\rho.\langle \texttt{unit} \to \rho\rangle_\rho$ pour type. Ce type ne reflète pas le fait que le contrôleur précédent P disparaît lors de son remplacement. Si l'ancien contrôleur contient un domaine actif a, et que le nouveau contrôleur contient également un domaine actif a, la configuration ne sera pas bien typée. Un système de types bien plus complexe serait nécessaire pour détecter ces situations. Cependant, aucun des exemples précédents de composants contrôlables n'utilise de sous-domaine actif dans le contrôleur, excepté la deuxième implémentation des bibliothèques que l'on peut mettre à jour. Dans ce cas, le nom du sous-domaine utilisé est créé frais et n'interagit pas avec le nouveau contrôleur introduit. Nous remarquons également que la ressource update est perdue lors de la mise à jour qui remplace intégralement le contrôleur. Le nouveau contrôleur doit donc contenir une telle ressource si une future mise à jour est envisagée.

Ces différentes formes de composants programmables peuvent être assemblées, pour donner par exemple un composant nommé que l'on peut déplacer, interrompre et mettre à jour $Q^{mie}(b)$.

La possibilité de facilement contrôler de l'extérieur des processus individuels est un des aspects les plus intéressants du M-calcul.

5.4.5 Simulations de calculs de processus distribués

Nous présentons dans cette sections des exemples capturant l'essence d'autres calculs de processus distribués. Nous considérons le π_{1l} calcul [Ama97], le join calcul distribué [FG96, Fou98] et le join calcul dynamique (voir le chapitre 4).

Simulation du π_{1l} calcul

Le π_{1l} calcul possède trois notions principales que nous cherchons à simuler ici : des localités nommées où les processus s'exécutent, une primitive spawn permettant de lancer un processus dans une localité, et un détecteur de pannes sous la forme d'une primitive ping pouvant détecter les localités en panne.

Nous représentons une localité du π_{1l} calcul $a[\cdot]$ par un domaine de la forme $a(PP(a))[\cdot]$ où le contrôleur $PP(a)$ est défini comme étant :

$$\begin{array}{rcl} PP(a) & = & \nu \mathsf{on}\ \mathsf{off}.(\langle \quad \mathsf{on} \mid \mathbf{i}\ (d, m, \mathit{args}) \triangleright (\mathsf{on} \mid d.m\ \mathit{args}); \\ & & \qquad\quad \mathsf{on} \mid \mathbf{o}(d, m, \mathit{args}) \triangleright (\mathsf{on} \mid d.m\ \mathit{args}); \\ & & \qquad\quad \mathsf{off} \mid \mathbf{i}\ (d, m, \mathit{args}) \triangleright [\overline{a}.\mathsf{ping} = d.m](m\ \mathit{args}, \mathbf{0}); \\ & & \qquad\quad \mathsf{off} \mid \mathbf{o}\ (d, m, \mathit{args}) \triangleright \mathbf{0}; \\ & & \qquad\quad \mathsf{on} \mid \mathsf{add}\ \ f \triangleright \mathit{Augment}(a, f); \\ & & \qquad\quad \mathsf{on} \mid \mathsf{stop} \triangleright \mathsf{off}; \\ & & \qquad\quad \mathsf{on} \mid \mathsf{ping}\ (y, n) \triangleright \mathsf{on} \mid y(); \\ & & \qquad\quad \mathsf{off} \mid \mathsf{ping}\ (y, n) \triangleright \mathsf{off} \mid n()\rangle) \\ \mathit{Augment}(a, f) & = & \mathtt{pass}_a\ \lambda p\ q.a(\mathsf{on} \mid p())[q() \mid f()] \end{array}$$

La construction $\mathsf{spawn}(a, P)$ du π_{1l} calcul correspond à l'envoi du processus P au domaine (ou localité) a. On peut simplement la coder par $\overline{a}.\mathsf{add}\ \ \lambda.P$. La construction $\mathsf{stop}(a)$ est codée par $\overline{a}.\mathsf{stop}$, et la construction $\mathsf{ping}(a, y, n)$ par $\overline{a}.\mathsf{ping}\ (y, n)$. Lorsqu'une localité est en panne, le seul message accepté de l'extérieur est un message sur ping, qui est réémis localement et sera consommé pour indiquer que la localité est effectivement en panne. Cette réduction se passant dans le contrôleur, le message indiquant que la localité est en panne peut bien sortir du domaine. Par contre, tout message du contenu pour l'extérieur est intercepté. Nous ne présentons pas le codage des canaux du π calcul, qui est identique à celui décrit dans [Fou98].

Comme dans les exemples précédents, la ressource add peut avoir le type $\forall\rho.\langle \texttt{unit} \to \rho\rangle_\rho$, et le processus ping le type $\langle \texttt{unit} \to \emptyset, \texttt{unit} \to \emptyset\rangle_\emptyset$.

Simulation du join calcul distribué

Le M-calcul s'inspirant très fortement du join calcul distribué, les principales caractéristiques de ce dernier sont directement reproductibles. Nous décrivons une traduction en trois étapes d'un processus join sans nom libre. La première étape consiste à typer le processus join à traduire, afin de déterminer le type à donner aux ressources correspondant aux noms de canaux. Le processus obtenu est identique au processus initial, excepté pour les définitions. Nous notons $\texttt{def}\ (D; n_1 : s_1, n_q : s_q, a_1, \ldots, a_r)\ \texttt{in}\ P$ la traduction après cette première étape de $\texttt{def}\ D\ \texttt{in}\ P$ avec $dn(D) = \{n_1, \ldots, n_q\} \cup \{a_1, \ldots a_r\}$, les n_i étant des noms de canaux possédant le schéma de type s_i, et les a_j étant des noms de locations. Nous avons par définition des types du join calcul distribué les schémas s_i de la forme $\forall\widetilde{\alpha}.\langle\widetilde{\tau}\rangle$. Nous supposons que ce type n'a aucune variable de type libre. Nous

transformons de manière immédiate ces schémas de type en schémas de type du M-calcul, notant $\langle\rangle$ pour $\langle\rangle_\emptyset$.

Une deuxième étape de la traduction consiste à annoter tous les noms de canaux avec la location les définissant. Nous nous basons ici sur le fait que dans le join calcul, la location dans laquelle une définition sera dépliée peut être connue statiquement : il suffit de regarder la location syntaxiquement englobante de la définition. Ceci est lié au fait que la seule migration d'ordre supérieur dans le join calcul distribué est la migration des locations, et non pas des autres processus. Nous préfixons ainsi tout nom de canal (qui n'est donc pas une variable) par la location englobant la définition de ce nom.

La troisième étape est la traduction proprement dite dans le M-calcul. Nous représentons une location du join calcul distribué $a[\,\cdot\,]$ par un domaine $a(PJ(a))[\,\cdot\,]$ avec $PJ(a)$ étant le contrôleur suivant :

$$\begin{aligned} PJ(a) &= Fwd \mid \langle \texttt{add}\ f \triangleright Add(a,f)\rangle \mid \langle \texttt{go}\ (b,\kappa) \triangleright Send(a,b,\kappa)\rangle \\ Add(a,f) &= \texttt{pass}_a\ \lambda pq.a(p())[q() \mid f()] \\ Send(a,b,\kappa) &= \texttt{pass}_a\ \lambda pq.(\overline{b}.\texttt{add}\ \lambda.a(p())[q() \mid \kappa()]) \end{aligned}$$

Nous notons $[\![\,\cdot\,]\!]^a$ l'opérateur de traduction paramétré par la location courante a. Nous avons :

$$\begin{aligned} [\![b.n]\!]^a &= \underline{b}.n \\ [\![b]\!]^a &= b \\ [\![x]\!]^a &= x \\ [\![\mathbf{0}]\!]^a &= \mathbf{0} \\ [\![P \mid Q]\!]^a &= [\![P]\!]^a \mid [\![Q]\!]^a \\ [\![m\langle n_1,\ldots,n_q\rangle]\!]^a &= [\![m]\!]^a([\![n_1]\!]^a,\ldots,[\![n_q]\!]^a) \\ [\![\texttt{def}\ (D; n_1:s_1,\ldots,n_q:s_q,a_1,\ldots,a_r)\ \texttt{in}\ P]\!]^a &= \nu n_1:s_1,\ldots,n_q:s_q.\nu a_1,\ldots,a_r.[\![D]\!]^a \mid [\![P]\!]^a \\ [\![\texttt{go}\ b; P]\!]^a &= \overline{a}.\texttt{go}(b,\lambda.[\![P]\!]^a) \\ [\![\top]\!]^a &= \mathbf{0} \\ [\![n_1\langle\widetilde{x_1}\rangle \mid \ldots \mid n_q\langle\widetilde{x_q}\rangle \triangleright P]\!]^a &= \langle n_1\langle\widetilde{x_1}\rangle \ldots n_q\langle\widetilde{x_q}\rangle \triangleright [\![P]\!]^a\rangle \end{aligned}$$

$$\begin{aligned} [\![D, D']\!]^a &= [\![D]\!]^a \mid [\![D']\!]^a \\ [\![b\,[D:P]\,]\!]^a &= b(PJ(b))[[\![D]\!]^b \mid [\![P]\!]^b] \end{aligned}$$

La traduction se contente simplement de remplacer les définitions du join calcul par les définitions équivalentes du M-calcul. Nous remarquons que nous prenons bien garde de n'utiliser que des messages adressés, pour ne pas confondre les messages nécessaires au codage (destinés aux contrôleurs) des messages du join calcul (destinés au contenu).

Comme auparavant, `add` a le type $\forall\rho.\langle \texttt{unit} \to \rho\rangle_\rho$. Le canal `go` prenant une continuation sous la forme d'un processus gelé, il a pour type $\forall\delta\rho.\langle \texttt{dom}(\delta), \texttt{unit} \to \rho\rangle_\rho$.

Nous remarquons que notre codage n'est pas fidèle en ce qui concerne la migration. En effet, dans le join calcul distribué, la migration n'est déclenchée que si la location destination n'est pas une sous-location de la location migrant, et si la destination est prête à recevoir la location migrant. En ce qui concerne le premier point, il nécessite une synchronisation avec toutes les sous-locations. Par exemple, l'implémentation de JoCaml procède de la manière suivante : si la destination est sur le runtime courant, l'exécution de tous les processus locaux est interrompue, et le runtime parcourt les locations vers la racine (chaque location possédant un pointeur vers son père) en partant de la location destination. Si la racine du runtime est atteinte sans rencontrer la location migrant, alors le `go` peut s'exécuter, sinon il est bloqué jusqu'à ce que la location cible migre, et l'exécution des processus reprend. Un codage plus fidèle de la migration peut donc utiliser un processus centralisé maintenant une représentation abstraite de l'arbre des locations. Une solution plus satisfaisante consisterait à étendre le M-calcul pour lui ajouter de la réflexivité, comme par exemple une primitive permettant de déterminer si un sous-domaine donné est présent. En ce qui concerne le deuxième point (la location destination est prête à recevoir la location migrant), nous remarquons que l'implémentation de JoCaml implémente très simplement la migration si la destination est dans le runtime courant (il s'agit tout simplement de modifier le pointeur vers la location père). Dans le cas d'une destination dans un runtime distant, la location est envoyée de runtime en runtime jusqu'à ce qu'elle rattrape la location destination. De ce point de vue, le M-calcul est bien plus proche de l'implémentation que le join calcul.

Simulation du join calcul dynamique

Nous étendons la traduction précédente avec une forme de liaison dynamique s'inspirant du chapitre 4. Cependant, au lieu de résoudre un message directement vers la location englobante le définissant, nous l'envoyons de proche en proche vers la location père si la location courante ne le définit pas. Nous traduisons ainsi un nom statique en nom adressé au domaine le définissant, et un nom dynamique en un nom local. Tout nom dynamique importé possède une définition l'envoyant dans le domaine englobant. Nous supposons que les noms statiques et les noms dynamiques sont disjoints, et nous supposons également que les noms importés et les noms définis sont également disjoints. Nous supposons enfin que les filtres ne définissent aucune variable (nous interdisons la redéfinition d'un nom dynamique reçu en argument).

Comme dans le cas de la simulation du join calcul distribué, nous procédons en trois temps : typage, résolution de noms statiques et traduction. La phase de typage est identique au cas précédent, en associant à tout définition `def` D `in` P l'ensemble des noms de canaux statiques qu'elle définit avec leurs schémas de types ainsi que l'ensemble des locations qu'elle crée. La deuxième étape est également similaire, associant à tout nom de canal statique la location où ce canal est défini.

En ce qui concerne la troisième étape, nous ne détaillons que les cas qui sont modifiés ou ajoutés :

$$\begin{aligned} [\![\mathbf{d}]\!]^a &= \mathbf{d} \\ [\![b\,[D : P]^{\Delta,I}\,]\!]^a &= b(PJD(b,a,I))[[\![D]\!]^b \mid [\![P]\!]^b] \\ [\![\nu\mathbf{d} : \langle\widetilde{\tau}\rangle^+_\Delta.P]\!]^a &= \nu\mathbf{d} : \langle\widetilde{\tau}\rangle_\emptyset.[\![P]\!]^a \end{aligned}$$

Afin de d'envoyer les messages dynamiques sur des noms importés, nous définissons le processus :

$$Dyn(\{\mathbf{d}_1,\ldots,\mathbf{d}_q\}) = \langle \mathbf{d_1} x \triangleright \mathsf{up}(\mathbf{d}_1, x)\rangle, \ldots \langle \mathbf{d_q} x \triangleright \mathsf{up}(\mathbf{d}_q, x)\rangle$$

Nous traduisons ainsi une location $a[\,\cdot\,]^{\Delta,I}$ par le domaine $a(PJD(a,b,I))[\,\cdot\,]$ si b est la location englobante, avec $PJD(a,b,I)$ étant le contrôleur :

$$\begin{aligned} PJD(a,b,I) &= Fwd \mid \langle \mathsf{add}\ f \triangleright Add(a,f)\rangle \mid \langle \mathsf{father}\ (c) \mid \mathsf{go}\ (b,\kappa) \triangleright Send(a,b,\kappa)\rangle \\ &\quad \mid \langle \mathsf{dyn}(m, args) \triangleright m\ args\rangle \\ &\quad \mid \langle \mathsf{father}(c) \mid \mathsf{up}(m, args) \triangleright \mathsf{father}(c) \mid \overline{c}.\mathsf{dyn}(m, args)\rangle \\ &\quad \mid Dyn(I) \mid \mathsf{father}\ (b) \\ Add(a,f) &= \mathtt{pass}_a\ \lambda pq.a(p())[q() \mid f()] \\ Send(a,b,\kappa) &= \mathtt{pass}_a\ \lambda pq.(\overline{b}.\mathsf{add}\ \lambda.a(p() \mid \mathtt{father}(b))[q() \mid \kappa()]) \end{aligned}$$

Étudions le devenir d'un message sur un nom dynamique $\mathbf{d}$. Si $\mathbf{d}$ est défini localement (il n'est pas dans I), il existe une définition pour ce nom localement, et le message pourra être consommé lorsque les autres messages nécessaires au filtre seront présents. Sinon, si le nom est importé, il possède une définition dans Dyn qui réémet le message sur le canal `up`. Ce message se synchronise alors avec le message contenant le nom du père courant $\mathsf{father}(b)$, et un message adressé à $\overline{b}.\mathsf{dyn}$ est émis. Lorsque le message parvient au contrôleur de b, il est réémis par le canal `dyn` et le cycle peut recommencer. Un nom dynamique est tout simplement un nom de ressource local qui est résolu de proche en proche.

Nous pouvons donner aux canaux `up` et `dyn` les schémas de types $\forall\alpha\rho.\langle\langle\alpha\rangle_\rho, \alpha\rangle_\rho$.

Nous remarquons que cette traduction n'est pas fidèle sur trois points. Tout d'abord, comme dans le cas du join calcul distribué, nous ne détectons pas les migrations circulaires. Ensuite, nous propageons les messages dynamiques de proche en proche, et non pas directement vers leur location cible. Ceci est dû à la définition du join calcul dynamique qui se base de manière cruciale sur la notion de *location gelée* (règle Str-Loc, figure 4.4) pour calculer la fonction de résolution de noms. Il serait possible de simuler ce comportement si la réactivation de domaines pouvait se faire de proche en proche, alors que pour le moment un processus gelé est réactivé en une seule étape. Enfin, nous n'autorisons pas la redéfinition de noms dynamiques reçus en argument, puisque le M-calcul ne permet pas la redéfinition de noms de ressources reçus en argument.

Simulation du join calcul distribué avec pannes

Nous étendons l'exemple de la simulation du join calcul distribué en y ajoutant les pannes.

Nous traduisons désormais une location $a[\cdot]$ par le domaine de la forme $a(PJF(a))[\cdot]$, où $PJF(a)$ est défini de la manière suivante :

$$
\begin{array}{rcl}
PJF(a) & = & \quad Fwd \\
 & | & \langle \mathsf{add}\ f \triangleright Add(a,f) \rangle \\
 & | & \langle \mathsf{go}\ (b,\kappa) \triangleright Send(a,b,\kappa) \rangle \\
 & | & \langle \mathsf{halt} \triangleright Halt(a) \rangle \\
 & | & \langle \mathsf{ping}\ (y,n) \triangleright y\,() \rangle \\
Add(a,f) & = & \mathtt{pass}_a\ \lambda pq.a(p())[q() \mid f()] \\
Send(a,b,\kappa) & = & \mathtt{pass}_a\ \lambda pq.(\bar{b}.\mathsf{add}\ \lambda.a(p())[q() \mid \kappa()]) \\
Halt(a) & = & \mathtt{pass}_a\ \lambda pq.\ A(a,p,q) \\
A(a,p,q) & = & a \left(\begin{array}{ll}
 & \langle \mathsf{ping}\ (y,n) \triangleright n\,() \rangle \\
| & \langle \mathsf{add}\ f \triangleright Add(a,f) \rangle \\
| & \langle \mathbf{i}\ (d,m,args) \triangleright [\mathsf{ping} = m](m\ args, [\mathsf{add} = m](m\ args, \mathbf{0})) \rangle \\
| & \langle \mathbf{o}\ (d,m,args) \triangleright \mathbf{0} \rangle
\end{array} \right) [q()]
\end{array}
$$

Ainsi, lorsqu'une location a échoué, elle ne laisse plus sortir aucun message de son contenu (excepté les messages locaux, qui ne peuvent sortir du domaine), et elle n'accepte de l'extérieur que les ping, qu'elle réémet en tant que messages locaux et auxquels elle répondra négativement quelle que soit leur destination (qui est nécessairement un sous-domaine de a ou a), et les locations migrant dans a ou une sous-location de a. Ces dernières deviennent alors inaccessibles.

Nous remarquons que les sous-locations de a sont toujours actives. Elles ne peuvent cependant plus communiquer avec l'extérieur. Notre codage s'appuie crucialement sur le fait que les messages sont routés de proche en proche, donnant ainsi l'occasion au contrôleur de la location en panne de les intercepter. C'est ainsi que nous interceptons les sous-locations essayant de migrer à l'extérieur de la location en panne pour les recréer localement, les forçant ainsi à rester en panne et être la destination de messages ping. Un autre codage possible, présenté dans [BSS01], consiste pour le contrôleur à maintenir la liste de toutes ses sous-locations, et à les avertir lorsque la location tombe en panne. Ce codage est bien entendu bien plus complexe que celui présenté précédemment.

5.4.6 Réflexivité

Le M-calcul possède deux constructions lui donnant une forme de réflexivité : les règles d'interception Red.Addr.In et Red.Addr.Cont.Out qui passent aux canaux **i** et **o** la destination, le canal et les arguments du message, et la règle de passivation Red.Passiv qui permet de séparer le contrôleur du contenu. Cependant, dans l'état actuel du calcul, ces constructions restent très limitées dans le sens où il n'est pas possible d'accéder au contenu des arguments d'un message, au contenu du contrôleur ou du contenu d'un domaine passivé.

En ce qui concerne le premier point, une extension intéressante peut être envisagée au niveau des tests, autorisant des filtres de tests plus complexes, et en utilisant les informations de types en fonction de la satisfaction du test pour typer le processus gardé.

En ce qui concerne le second point, celui-ci nécessite une étude sur les primitives utiles pour observer une structure de domaines. Ainsi, comme nous l'avons vu dans l'exemple de la simulation du join calcul distribué, il serait intéressant de pouvoir garantir qu'un certain domaine n'est pas présent.

Enfin, et ceci est lié au second point, un contrôleur a peu de contrôle sur les échanges ayant lieu à l'intérieur de son contenu, rendant impossible le codage d'observateurs comme décrit dans [SGN00]. Une telle possibilité serait nécessaire pour assimiler les domaines à des composants EJB [Mic98] par exemple, mais la simple définition du contrôle recherché semble très complexe.

5.5 Conclusion

Nous avons présenté dans ce chapitre le M-calcul, un calcul de processus d'ordre supérieur, ainsi qu'un système de types garantissant l'unicité des noms des domaines actifs, critère crucial pour le déterminisme du routage et l'implémentation. Le M-calcul consiste en une extension non triviale du join calcul [Fou98, Lev97] par des fonctions d'ordre supérieur en appel par valeur, des localités programmable (les domaines), des processus mobiles (au lieu de simples locations mobiles), et une forme de liaison dynamique par définition du même nom de ressources dans plusieurs domaines.

Le mariage entre le join calcul et le λ-calcul s'inspire du calcul Bleu de Boudol [Bou98]. Comme nous l'avons montré dans les exemples de la section 5.4, les localités programmables nous permettent de capturer dans un unique calcul de nombreuses notions qui ont été décrites dans plu-

sieurs calculs de processus distribués, dont les calculs d'ambients [CG98, LS00, MH02, BCC01]. En ce qui concerne ces derniers, nous remarquons que nous n'avons présenté que des formes de migration asynchrones, c'est à dire ne requérant pas de synchronisation entre ambients pour que la migration ait lieu. Afin de capturer les capacités classiques des ambients, il serait nécessaire d'utiliser l'algorithme décrit dans le chapitre 3, ou simplement de traduire la version join calcul de l'algorithme en M-calcul. Nous remarquons que le M-calcul nous donne la possibilité de dissoudre des domaines (alors que cela n'est pas possible avec le join calcul) ; nous pouvons ainsi fidèlement traduire la capacité Open.

Du point de vue d'un langage de programmation, le M-calcul possède des caractéristiques intéressantes :

1. Il est possible de définir plusieurs sémantiques liées à l'interaction entre localités sans modifier le calcul. Ceci nous a ainsi permis le codage de nombreux autres calculs.
2. La notion de fonction est disponible, se rapprochant ainsi de JoCaml. Ainsi l'envoi d'une fonction dans un message correspond à une migration de code, alors que l'envoi d'un nom de ressource correspond à l'envoi d'une référence distante.
3. Le corps des fonctions pouvant contenir des processus, nous unifions migration et communication, en considérant la migration comme étant la communication du processus migrant. Ceci permet de ne proposer qu'un seul mécanisme de contrôle d'interaction entre localités.
4. La communication est automatique mais contrôlée : il n'est pas nécessaire de spécifier comment un message peut atteindre un domaine donné, les règles de routage s'en chargent. Par contre, chaque domaine dont la frontière est traversée possède l'opportunité d'intercepter le message.
5. Enfin, nous retenons du join calcul dynamique du chapitre 4 la possibilité de créer une définition pour une ressource donnée en plusieurs endroits, permettant ainsi de donner un sens local à un nom.

Cette conception des localités s'approche de celle du calcul des Boxed Ambients [BCC01], dont les primitives de communication s'inspirent du Seal Calcul [VC98]. Les différences les plus importantes avec le M-calcul résident dans la communication, la migration, et le routage. La communication dans le M-calcul est complètement asynchrone et d'ordre supérieur (on peut envoyer des processus gelés dans les messages), alors que la communication dans le calcul des Boxed Ambients est synchrone et du premier ordre. La migration dans le M-calcul est, comme nous l'avons précédemment souligné, la communication du processus migrant gelé, et n'est pas limitée aux localités. Par contre, la migration du calcul des Boxed Ambients ne concerne que les ambients, et utilise les primitives (ou capacités) in et out. Enfin, le routage dans le calcul des Boxed Ambients est explicite : il est nécessaire de spécifier le chemin pour atteindre un ambient distant, alors que ce n'est pas le cas dans le M-calcul. La possibilité d'avoir un routage automatique mais contrôlé nous semble cruciale à un langage de programmation distribué. En effet, la communication à distance est un aspect récurrent des programmes distribués, et obliger le programmeur à spécifier le routage ne nous semble pas réaliste. De plus, écrire une bibliothèque permettant d'abstraire la communication à distance pour un calcul d'ambients nous semble être une tâche très complexe, puisqu'il est nécessaire de prendre en compte le fait que plusieurs ambients peuvent porter le même nom. La solution la plus simple à ce problème serait d'ajouter une primitive de routage automatique, comme dans le join calcul. Par contre, un deuxième aspect de la communication qui est bien pris en compte dans le calcul des Boxed Ambient est le contrôle de la communication, c'est à dire la notion qu'un ambient doit autoriser que l'on franchisse sa frontière. C'est cet aspect, absent du join calcul, que nous avons voulu reprendre dans nos primitives de routage qui se font étape par étape, en laissant à tout domaine traversé la possibilité d'intercepter le message.

D'un point de vue théorique, nous illustrons la possibilité de raisonner sur des processus du M-calcul par un système de types garantissant l'unicité des noms des domaines actifs. Ce résultat est également utile du point de vue de l'implémentation, puisqu'il nous garantit le déterminisme du routage : à tout instant, il existe au plus un domaine actif portant un nom donné. Ce travail est très proche de celui de Yoshida et Hennessi sur le D$\pi\lambda$ calcul, un calcul de processus d'ordre supérieur inspiré du π calcul et du λ calcul [YH00, YH99]. Le D$\pi\lambda$ calcul autorise en effet la communication de processus sous forme gelée. Nous soulignons tout d'abord quelques différences entre le D$\pi\lambda$ calcul et le M-calcul. Du point de vue de la communication, le D$\pi\lambda$ calcul adopte une vision très proche de celle du join calcul : la communication se passe dans un modèle plat, où toute localité est voisine de toute autre, et la destination d'un message dépend du nom du canal sur lequel ce message est envoyé. Le M-calcul adopte lui une approche basée sur les noms de domaines : la destination d'un message est explicitement spécifiée. Ainsi, pour assurer une implémentation aisée,

les deux systèmes ont besoin de propriétés différentes : le D$\pi\lambda$ requiert la *localité des noms*, c'est à dire la garantie que tout canal soit défini dans au plus une localité, alors que le M-calcul requiert l'unicité des noms de domaines actifs. Ce choix permet au M-calcul de définir un même nom de ressource à plusieurs endroit, lui donnant un sens dépendant de la localité. De plus, la garantie de l'unicité des noms de domaines actifs est plus complexe à établir que la localité des noms du D$\pi\lambda$, à cause de l'opérateur de passivation. Ce dernier est en effet un opérateur subjectif : il peut geler des processus en contexte d'évaluation. Le D$\pi\lambda$ ne propose qu'un envoi de processus gelés objectifs, puisque ceux-ci sont soit présents explicitement dans le terme du calcul, soit reçus en argument avant d'être envoyés. Malgré ces différences, les deux calculs considèrent que le type d'un processus correspond à une approximation conservative des évolutions du processus : son interface pour le D$\pi\lambda$, ses domaines actifs pour le M-calcul. Ainsi, les contraintes de localité ou d'unicité de nom ne sont appliquées qu'aux processus qui sont ou seront en contexte d'évaluation. En ce qui concerne le système de types proprement dit, les deux calculs ont besoin de noms dans les types. L'approche est cependant différente en ce qui concerne les noms non encore déterminés, parce qu'ils correspondent aux paramètre d'une fonction ou d'une ressource. Le D$\pi\lambda$ calcul utilise des types dépendants, en autorisant des variables de noms dans les types et la liaison de ces variables dans les types fonctionnels. Le typage d'une application d'un processus à un nom substitue la variable de nom à gauche de la flèche par le nom appliqué pour obtenir le type final. A contrario, le M-calcul n'autorise pas de variables de noms dans les types, mais des variables de types de noms, qui sont une forme de variable de types et qui ne peuvent être présentes dans les termes. Ces variables peuvent être généralisées et c'est le mécanisme classique d'instanciation qui simule la dépendance entre arguments et type final. Cette utilisation du polymorphisme est moins puissante que les types dépendants, puisque le type d'une ressource devient monomorphe après instanciation, empêchant par exemple de la passer à une fonction qui l'utiliserait avec deux noms de domaines différents. Par contre, nous désirions avoir du polymorphisme à la mode du join calcul dans notre système, et l'extension que nous avons présentée pour représenter les types dépendants ne complique pas beaucoup le système de types. De plus, ce choix semble être suffisant dans la plupart des cas. Enfin, une dernière différence entre le D$\pi\lambda$ et le M-calcul est la possibilité dans le D$\pi\lambda$ de créer un récepteur avec un nom passé en argument. Nous n'avons pas décrit cette possibilité dans le M-calcul pour simplifier la présentation, mais ceci ne présente pas d'obstacle technique majeur. Pour ce faire, il est nécessaire d'introduire un nouveau type de *ressources redéfinissable* très similaire au type homonyme du chapitre 4, ainsi qu'une nouvelle catégorie de noms.

De nombreuses extensions au M-calcul sont envisagées. Nous soulignons particulièrement deux d'entre elles. Tout d'abord, nous désirons augmenter la réflexivité du calcul pour affiner le filtrage. Nous comptons pour ce faire adapter le travail sur le typage dynamique de Duggan [Dug99] au M-calcul. En parallèle, nous travaillons sur les aspects théoriques du calcul, pour par exemple définir des notions d'observables et des équivalences, et sur l'implémentation d'une machine virtuelle.

Remerciement Ce travail a été réalisé avec Jean-Bernard Stefani. Nous remercions Gérard Boudol ainsi que tout le groupe de travail MARVEL pour leur nombreuses suggestions.

Chapitre 6

Conclusions

Nous avons décrit dans ce travail le cheminement suivi dans la conception d'un calcul d'agents mobiles répondant à certains critères, ainsi que les calculs intermédiaires que nous avons développés. Nos trois critères principaux sont la richesse du calcul, lui permettant d'être proche d'un langage de programmation, la facilité d'implémentation, en particulier la volonté d'éviter toute synchronisation distribuée, et le cadre formel laissant la possibilité de prouver la correction de programmes ou de concevoir des systèmes de types.

Notre travail s'est ainsi basé sur deux calculs d'agents assez différents : le calcul des ambients [CG98] et le join calcul [Fou98]. Nous avons tout d'abord étudié les différences et similitudes de ces calculs dans une première partie, en traduisant le calcul des ambients dans le join calcul. Cette traduction a permis de mettre en évidence certaines synchronisations dans le calcul des ambients qui se traduisent par un algorithme complexe dans le join calcul. Le cadre formel des deux calculs nous a permis de montrer la correction de cette traduction, et l'existence d'une implémentation du join calcul, JoCaml [Le 98], d'obtenir une implémentation distribuée du calcul des ambients [FS99]. Nous avons également souligné les forces et faiblesses des deux calculs en tant que langages de programmation. En particulier, le calcul des ambients ne facilite pas la communication à distance, alors que le join calcul ne reflète pas la notion de localité, que ce soit au niveau de l'accès à des ressources locales qu'au niveau du contrôle de la communication ou de la migration.

Dans une deuxième partie, nous avons montré comment le join calcul distribué peut être étendu pour prendre en partie en compte la notion de localité. Nous avons ainsi défini une notion de canaux dynamiques, dont l'implémentation dépend de l'endroit où le message est émis. Nous avons bien pris garde de réaliser cette extension de telle sorte que la résolution des noms, c'est à dire la détermination de la destination d'un message, soit une étape purement locale. Nous allons profiter de la réimplémentation actuellement en cours de JoCaml pour intégrer les canaux dynamiques. D'un point de vue théorique, nous avons étendu le système de types polymorphe du join calcul distribué pour représenter les canaux dynamiques qu'implémente ou importe une location, ainsi que les canaux dynamiques sur lesquels un canal donné peut émettre. Notre système de types, prouvé correct, nous permet ainsi de garantir que tout message sur un nom dynamique possède bien une location englobante définissant ce nom. Cependant, une telle extension ne permet pas un contrôle simple de la communication ni de la migration, elle se contente de donner un sens à la localité.

C'est pourquoi dans une troisième partie nous avons conçu un calcul ayant pour objectif d'intégrer les propriétés de contrôle local du calcul des ambients tout en gardant la facilité de communication à distance du join calcul et la notion de ressource localisée du join calcul dynamique. Nous désirions également être capable de simuler de nombreuses formes d'interactions entre localités sans pour autant devoir changer la sémantique du calcul à chaque fois. La notion de *domaine* est donc apparue naturellement, composée d'un *contrôleur*, un processus du M-calcul, qui gère l'interaction du *contenu* avec l'extérieur. Nous avons unifié communication et migration en considérant la migration comme l'envoi d'un message contenant un processus gelé. La communication à distance se fait de proche en proche, en ne franchissant qu'une frontière de domaine à chaque étape, et en laissant l'opportunité au contrôleur d'un domaine traversé d'intercepter le message. Par contre, afin de simplifier la communication à distance, le routage des messages est automatique. La destination du message étant explicitement spécifiée pour les messages distants, il est nécessaire pour une implémentation plus efficace de garantir l'unicité des noms des domaines actifs. Nous avons développé et prouvé correct un système de types nous donnant cette garantie.

La conception du M-calcul nous ouvre des perspectives intéressantes et nous comptons prolonger ce travail de plusieurs manières. Tout d'abord, nous allons réaliser une implémentation du

calcul, en nous basant sur les techniques utilisées par JoCaml. Nous pourrons ainsi comparer la facilité d'écriture de programmes entre le join calcul et le M-calcul. D'un point de vue théorique, deux chemins s'offrent à nous. Nous envisageons tout d'abord d'étendre le calcul pour lui donner plus de réflexivité, afin de permettre un contrôle plus fin. Nous voulons également définir des notions d'observables, en déduire des équivalences qui nous permettraient d'établir la correction des traductions proposées dans le chapitre 5. Ces observables sont bien sûr très proches des capacités réflexives du calcul, et nous comptons développer ces deux aspects en parallèle.

Annexe A

Preuves du chapitre 3

A.1 Correction de l'algorithme de synchronisation

Dans ce chapitre, nous désignons par $\mathcal{J}$ l'ensemble des processus du join calcul, par $\mathcal{A}$ l'ensemble des processus du calcul des ambients et par $\mathcal{E}$ l'ensemble des processus du calcul des ambients étendus. Nous avons donc $\mathcal{A} \subseteq \mathcal{E}$. Nous disons qu'un processus $P \in \mathcal{E}$ est *non restreint* lorsqu'il ne contient aucune restriction νx en contexte d'évaluation. Nous utilisons les notations habituelles pour les relations : $\mathcal{R}^{=}$, $\mathcal{R}^{+}$ et $\mathcal{R}^{*}$ sont respectivement les fermetures réflexives, transitives et reflexives-transitives de la relation $\mathcal{R}$; $\mathcal{R}^{-1}$ est l'inverse de la relation $\mathcal{R}$; $\mathcal{R}\mathcal{R}'$ est la composition des relations $\mathcal{R}$ et $\mathcal{R}'$.

Dans toute cette section nous utilisons la convention de désigner par P et P' des processus ambients et par Q, Q', R et R' des processus ambients étendus. Soit $i \in \widetilde{x}$, nous notons $\widetilde{x} \setminus i$ le nuplet obtenu en supprimant i de $\widetilde{x}$.

Nous décrivons tout d'abord la technique de preuve que nous utilisons afin de prouver le théorème 2, puis nous prouvons chacun des lemmes nécessaire dans un deuxième temps. La formulation du lemme A.1.1 est adaptée aux ambients, mais la technique de preuve utilisée peut s'appliquer à toute preuve de simulations couplées entre des processus identiques équipés de sémantiques de réduction différentes.

Lemme A.1.1 *Afin de prouver le théorème 2 : $(P, \longrightarrow) \lesssim (P, \rightarrow_{12C})$ pour tout $P \in \mathcal{A}$, il suffit de montrer que :*

1. *Pour tout $P \in \mathcal{A}$ et $Q \in \mathcal{E}$, si $P \rightarrow_{12C}^{*} Q$, alors il existe $P' \in \mathcal{A}$ tel que $Q \rightarrow_{2}^{*} P'$.*
2. *Pour tous $P, P' \in \mathcal{A}$, nous avons $P \longrightarrow^{*} P'$ si et seulement si $P \rightarrow_{12C}^{*} P'$.*
3. *Pour tous $Q, Q' \in \mathcal{E}$, si $Q \downarrow_a$ et si $Q \rightarrow_2 Q'$, alors $Q' \downarrow_a$.*

Preuve: Nous définissons les relations entre processus et processus étendus comme suit : $\geqslant = \{(P, Q) \mid P \rightarrow_{12C}^{*} Q\}$ et $\leqslant = \{(P, Q) \mid Q \rightarrow_{2}^{*} P\}$. Nous prouvons maintenant que $(\geqslant, \leqslant)$ sont des simulations couplées qui préservent les barbes faibles et sont fermées par application de contextes d'évaluation. Nous avons donc $\geqslant \cap \leqslant \subseteq \lesssim$, et nous concluons en remarquant que $\{(P, P)\} \subseteq \geqslant \cap \leqslant$.

Puisque les relations de réductions $\rightarrow_1$, $\rightarrow_2$ et $\rightarrow_C$ sont fermées par application de contextes d'évaluation, c'est aussi le cas pour $\geqslant$ et $\leqslant$ par définition.

Nous montrons que $\geqslant$ et $\leqslant$ satisfont les quatre diagrammes définissant une paire de simulations couplées (ces diagrammes sont présentés après la définition 3.4.2). En ce qui concerne la relation $\geqslant$, le diagramme de simulation Sim est satisfait avec aucune réduction dans sa partie gauche, et avec la composition des deux séries d'étapes universellement quantifiées $\rightarrow_{12C}^{*}$ dans sa partie basse. Le diagramme de couplage Cpl pour $\geqslant$ correspond à une combinaison des hypothèses 1 et 2 : l'hypothèse 1 est utilisée pour obtenir un P' convenable, et l'hypothèse 2 pour fermer la partie gauche du diagramme. En ce qui concerne la relation $\leqslant$, le diagramme de couplage est satisfait en utilisant l'identité à la place de $\geqslant$ en bas du diagramme, et en reportant les étapes $\rightarrow_2$ correspondant à la relation $\leqslant$ initiale à droite du diagramme, qui est donc trivialement fermé. Enfin, le diagramme de simulation de $\leqslant$ est fermé en assemblant à droite toutes les étapes $\rightarrow_2$ correspondant à la relation $\leqslant$ initiale, suivies par les étapes $\rightarrow_{12C}^{*}$ correspondant aux étapes $\longrightarrow^{*}$ présentes à gauche du diagramme, en appliquant l'hypothèse 1. Le diagramme est fermé par l'identité en bas.

Nous montrons enfin que les relations $\leqslant$ et $\geqslant$ préservent tous les barbes en utilisant entre autres les diagrammes juste démontrés. Supposons que nous ayons $P \geqslant Q$ et $Q \Downarrow_a$, alors par définition des barbes faibles nous avons $Q \rightarrow_{12C}^{*} Q' \downarrow_a$, donc par simulation il existe un P' tel

que $P \longrightarrow^* P'$ et $P' \geqslant Q'$. Par couplage, il existe un P'' tel que $P' \longrightarrow^* P''$ et $P'' \leqslant Q'$. Par définition de $\leqslant$, nous avons $Q' \rightarrow_2^* P''$. Puisque $Q' \downarrow_a$, nous avons par l'hypothèse 3 $P'' \downarrow_a$. Donc nous avons $P \longrightarrow^* P' \longrightarrow^* P'' \downarrow_a$, et donc $P \Downarrow_a$. Nous montrons la préservation des barbes pour $\leqslant$. Supposons que nous ayons $P \leqslant Q$ et $P \Downarrow_a$, alors par définition de la barbe faible il existe P' tel que $P \longrightarrow^* P' \downarrow_a$. Par simulation de $\leqslant$, il existe Q' tel que $Q \rightarrow_{12C}^* Q'$ et $P' \leqslant Q'$. Par définition de $\leqslant$, nous avons $Q' \rightarrow_2^* P'$. Donc nous avons $Q \rightarrow_{12C}^* Q' \rightarrow_2^* P' \downarrow_a$, et donc $Q \Downarrow_a$.

□

Informellement, les ambients étendus se réduisent de la manière suivante. Chaque étape $\rightarrow_1$ transforme un ambient présent dans un contexte d'évaluation en une souche et, dans le cas des étapes In 1 ou Out 1, introduit un scion correspondant. Chaque étape Open 2 transforme une souche en un ambient dissous. Chaque étape Move 2 supprime une souche et substitue l'ambient correspondant au scion qui lui est associé. Les autres étapes ne font intervenir ni les souches, ni les scions. Par définition, aucune souche ne peut être présente sous une garde, comme c'est indiqué en figure 3.5. De plus, certains processus étendus, comme $\bar{\imath}\{Q\}\text{–}n[i]$ (pour lequel la règle Move 2 ne peut pas s'appliquer), ne sont pas atteignables par réduction de processus ambients. Par la suite, nous appelons les souches de la forme $\bar{\imath}\{P\}\text{–}[Q]$ *souches migrantes*, et les souches de la forme $\circ\{P\}\text{–}[Q]$ *souches ouvertes*.

Le lemme suivant formalise ces propriétés, sous la forme d'un invariant, et décrit les résidus des étapes 2 au cours des réductions.

Lemme A.1.2 *Soient $(P, Q) \in \mathcal{A} \times \mathcal{E}$ des processus tels que $P \rightarrow_{12C}^* Q$. Il existe des noms $\widetilde{x}$ et un processus $R \in \mathcal{E}$ tel que nous ayons :*

1. *$Q \equiv \nu\widetilde{x}.R$ et R est non restreint.*
2. *Pour chaque souche présente dans R, soit il s'agit d'une souche migrante, alors R est de la forme $E(i)(\,\bar{\imath}\{P_i\}\text{–}n[\,R_i\,]\,)$ pour un certain nom i de $\widetilde{x}$ et pour un certain contexte d'évaluation étendu à deux trous $E(\,\cdot\,)(\,\cdot\,)$, soit c'est une souche ouverte, alors le processus R est de la forme $E(\,\circ\{P_o\}\text{–}n[\,R_o\,]\,)$ pour un certain contexte d'évaluation $E(\,\cdot\,)$.*
3. *Soit $\prec$ la relation sur les ambients étendus définie de la manière suivante : nous avons $j \prec i$ si et seulement si il existe des contextes d'évaluation étendus $E(\,\cdot\,)$ et $F(\,\cdot\,)$ tels que $R = E(\,\bar{\imath}\{P_i\}\text{–}n[\,F(j)\,]\,)$. Intuitivement, nous avons $j \prec i$ lorsque le scion j est à l'intérieur de la souche i.*

   La relation $\prec$ ne possède pas de cycles, c'est à dire qu'il n'existe aucun i tel que $i \prec^+ i$.
4. *Pour toute étape de réduction $Q \rightarrow_{12C} Q'$ nous considérons chaque souche de R.*
   S'il s'agit d'une souche migrante, de la forme $R = E(i)(\,\bar{\imath}\{P_i\}\text{–}n[\,R_i\,]\,)$, un des cas suivant est satisfait :
   - *(a) $Q' \equiv \nu\widetilde{x} \setminus i.E(\,n[\,P_i \mid R_i\,]\,)(\mathbf{0})$ (étape finale).*
   - *(b) $Q' \equiv \nu\widetilde{y}.E'(i)(\,\bar{\imath}\{P_i\}\text{–}n[\,R_i\,]\,)$ pour des noms $\widetilde{y}$ et un contexte d'évaluation étendu dont les trous sont non restreints $E'(\,\cdot\,)(\,\cdot\,)$ et tel que (étape externe) :*
     - *i. $\widetilde{y} = \widetilde{x}$ et $E(\,\cdot\,)(\,\cdot\,) \rightarrow_{12C} E'(\,\cdot\,)(\,\cdot\,)$, ou*
     - *ii. $\widetilde{y} = \widetilde{x}, j$ et $E(\,\cdot\,)(\,\cdot\,) \rightarrow_1 \nu j.E'(\,\cdot\,)(\,\cdot\,)$ pour un nom frais j, ou*
     - *iii. $\widetilde{y} = \widetilde{x} \setminus j$ et $\nu j.E(\,\cdot\,)(\,\cdot\,) \rightarrow_2 E'(\,\cdot\,)(\,\cdot\,)$ pour un nom $j \in \widetilde{x} \setminus i$.*
   - *(c) $Q' \equiv \nu\widetilde{y}.E(i)(\,\bar{\imath}\{P_i\}\text{–}n[\,R'_i\,]\,)$ pour des noms $\widetilde{y}$ et un processus $R'_i \in \mathcal{E}$ tels que (étape interne) :*
     - *i. $\widetilde{y} = \widetilde{x}$ et $R_i \rightarrow_{12C} R'_i$, ou*
     - *ii. $\widetilde{y} = \widetilde{x} \setminus j$ et $\nu j.R_i \rightarrow_2 R'_i$ pour un nom $j \in \widetilde{x} \setminus i$.*
   - *(d) $Q' \equiv \nu\widetilde{x}, j.E(i)(j \mid \bar{\imath}\{P_i\}\text{–}n[\,R'_i\,]\,)$ pour un processus $R'_i \in \mathcal{E}$ et un nom frais j tels que $\bar{\imath}\{P_i\}\text{–}n[\,R_i\,] \rightarrow_1 \nu j.j \mid \bar{\imath}\{P_i\}\text{–}n[\,R'_i\,]$ (étape sortante Out 1).*
   - *(e) $Q' \equiv \nu\widetilde{x} \setminus j.F(i)(\,\bar{\imath}\{P_i\}\text{–}n[\,G(\mathbf{0})\,]\,)(\,m[\,P_j \mid R_j\,]\,)$ pour un nom j, des contextes d'évaluation F (à trois trous) et G (à un trou), et un processus $m[\,P_j \mid R_j\,] \in \mathcal{E}$ tels que $j \in \widetilde{x} \setminus i$ et $R = F(i)(\,\bar{\imath}\{P_i\}\text{–}n[\,G(\,\bar{\jmath}\{P_j\}\text{–}m[\,R_j\,]\,)\,]\,)(j)$ (étape sortante Move 2).*
   - *(f) $Q' \equiv \nu\widetilde{x} \setminus j.F(i)(\,\bar{\imath}\{P_i\}\text{–}n[\,G(\,m[\,P_j \mid R_j\,]\,)\,]\,)(\mathbf{0})$ pour un nom j, des contextes d'évaluation F (à trois trous) et G (à un trou), et un processus $m[\,P_j \mid R_j\,] \in \mathcal{E}$ tels que $j \in \widetilde{x} \setminus i$ et $R = F(i)(\,\bar{\imath}\{P_i\}\text{–}n[\,G(j)\,]\,)(\,\bar{\jmath}\{P_j\}\text{–}m[R_j])$ (étape entrante Move 2).*

   S'il s'agit d'une souche ouverte, de la forme $R = E(\,\circ\{P_o\}\text{–}n[\,R_o\,]\,)$, un des cas suivants est satisfait :

(a) $Q' \equiv \nu\widetilde{x}.E(P_o \mid R_o)$ *(étape finale).*

(b) $Q' \equiv \nu\widetilde{y}.E'(\,\mathtt{o}\{P_o\}\text{–}n[\,R_o\,]\,)$ *pour des noms* $\widetilde{y}$ *et un contexte d'évaluation* $E'(\,\cdot\,)$ *dont le trou est nom restreint et tel que (étape externe) :*

i. $\widetilde{y} = \widetilde{x}$ *et* $E(\,\cdot\,) \rightarrow_{12C} E'(\,\cdot\,)$*, ou*

ii. $\widetilde{y} = \widetilde{x}, j$ *et* $E(\,\cdot\,) \rightarrow_1 \nu j.E'(\,\cdot\,)$ *pour un nom frais* j*, ou*

iii. $\widetilde{y} = \widetilde{x} \setminus j$ *et* $\nu j.E(\,\cdot\,) \rightarrow_2 E'(\,\cdot\,)$ *pour un nom* $j \in \widetilde{x}$*.*

(c) $Q' \equiv \nu\widetilde{y}.E(\,\mathtt{o}\{P_o\}\text{–}n[\,R'_o\,]\,)$ *pour des noms* $\widetilde{y}$ *et un processus* $R'_o \in \mathcal{E}$ *tels que (étape interne) :*

i. $\widetilde{y} = \widetilde{x}$ *et* $R_o \rightarrow_{12C} R'_o$*, ou*

ii. $\widetilde{y} = \widetilde{x} \setminus j$ *et* $\nu j.R_o \rightarrow_2 R'_o$ *pour un nom* $j \in \widetilde{x}$*.*

(d) $Q' \equiv \nu\widetilde{x}, j.E(j \mid \mathtt{o}\{P_o\}\text{–}n[\,R'_o\,]\,)$ *pour un processus* $R'_o \in \mathcal{E}$ *et un nom frais* j *tels que* $\mathtt{o}\{P_o\}\text{–}n[\,R_o\,] \rightarrow_1 \nu j.j \mid \mathtt{o}\{P_o\}\text{–}n[\,R'_o\,]$ *(étape sortante* Out 1*).*

(e) $Q' \equiv \nu\widetilde{x} \setminus j.F(\,\mathtt{o}\{P_o\}\text{–}n[\,G(\mathbf{0})\,]\,)(\,m[\,P_j \mid R_j\,]\,)$ *pour un nom* j*, des contextes d'évaluation* F *(à deux trous) et* G *(à un trou), et un processus* $m[\,P_j \mid R_j\,] \in \mathcal{E}$ *tels que* $j \in \widetilde{x}$ *et* $R = F(\mathtt{o}\{P_o\}\text{–}n[G(\overline{\jmath}\{P_j\}\text{–}m[R_j])])(j)$ *(étape sortante* Move 2*).*

(f) $Q' \equiv \nu\widetilde{x} \setminus j.F(\,\mathtt{o}\{P_o\}\text{–}n[\,G(\,m[\,P_j \mid R_j\,]\,)\,]\,)(\mathbf{0})$ *pour un nom* j*, des contextes d'évaluation* F *(à deux trous) et* G *(à un trou), et un processus* $m[\,P_j \mid R_j\,] \in \mathcal{E}$ *tels que* $j \in \widetilde{x}$ *et* $R = F(\,\mathtt{o}\{P_o\}\text{–}n[\,G(j)\,]\,)(\overline{\jmath}\{P_j\}\text{–}m[R_j])$ *(étape entrante* Move 2*).*

Preuve: Nous prouvons la conservation de l'invariant par induction globale sur la longueur de la dérivation $P \rightarrow_{12C}^{*} Q$ pour les trois premières propriétés du lemme, et nous vérifions que la quatrième est satisfaite par l'étape considérée.

Les propriétés 1 à 3 sont immédiates si $P = Q$: l'ambient de la forme $\nu\widetilde{x}.R$ est obtenu en extrudant toutes les restrictions jusqu'à la racine après α-conversion, et par définition de $\mathcal{A}$, R ne contient aucune souche.

Supposons maintenant que les propriétés soient satisfaites pour la dérivation $P \rightarrow_{12C}^{n} Q$, et considérons l'étape $Q \rightarrow_{12C} Q'$. Nous prouvons les propriétés 1 à 3 au rang $n+1$ et la propriété 4 au rang n (c'est à dire pour l'étape $Q \rightarrow_{12C} Q'$ elle-même), ce par cas selon l'étape $\rightarrow_{12C}$ ayant lieu. Dans chaque cas nous introduisons un processus étendu R' dérivé de R et tel que $Q' \equiv \nu\widetilde{y}.R'$, puis nous étudions l'effet de l'étape sur chaque souche de R et sur la relation $\prec$.

In 1 **:** Dans ce cas une souche migrante est créée. Soit i un nom frais. Par définition de l'étape In 1 il existe un contexte d'évaluation $E(\,\cdot\,)(\,\cdot\,)$ dont les deux trous sont en parallèle (donc de la forme $E(\,\cdot \mid \cdot\,)$, quitte à réarranger la composition parallèle), tel que :

$$\begin{aligned} R &= E(\,m[\,Q_i\,]\,)(\,n[\,\mathtt{in}\; m.P_i \mid R_i\,]\,) \\ Q' &\equiv \nu\widetilde{x}.\nu i.E(\,m[\,i \mid Q_i\,]\,)(\,\overline{\imath}\{P_i\}\text{–}n[\,R_i\,]\,) \end{aligned}$$

Pour toutes les souches contenant les trous de $E(\,\cdot\,)(\,\cdot\,)$, cette étape est une étape interne *(4c,i)* suivie de l'extrusion de la portée de i. Pour toutes les autres souches de R, cette étape est une étape externe *(4b,ii)*.

En ce qui concerne la relation $\prec$, soit J l'ensemble des noms associés aux souches migrantes contenant les trous de $E(\,\cdot\,)(\,\cdot\,)$, et soit K l'ensemble des scions de R_i. Les noms en relation avant l'étape le sont toujours après l'étape. La relation est par contre étendue par les couples $k \prec i$ pour tout $k \in K$ et par les couples $i \prec j$ pour tout $j \in J$. Nous remarquons que cette extension ne peut faire apparaître de nouveau cycle, puisqu'un tel nouveau cycle utiliserait nécessairement i, donc comprendrait la chaîne $k \prec i \prec j$ pour un certain $k, j \in K \times J$, alors que nous avons déjà $k \prec j$ pour R, donc un cycle existerait avant l'étape, ce qui contredit l'hypothèse d'induction *3*.

Out 1 **:** Comme dans le cas In 1, une souche migrante est créée. Soit i un nom frais. Quitte à réarranger la composition parallèle des termes à l'intérieur de l'ambient père, il existe un contexte d'évaluation $E(\,\cdot\,)$ et des processus étendus R_i et R'_i tels que :

$$\begin{aligned} R &= E(\,X^{=}m[\,n[\,\mathtt{out}\; m.P_i \mid R_i\,] \mid R'_i\,]\,) \\ Q' &\equiv \nu\widetilde{x}.\nu i.E(i \mid X^{=}m[\,\overline{\imath}\{P_i\}\text{–}n[\,R_i\,] \mid R'_i\,]\,) \end{aligned}$$

En fonction de la forme de $X^{=}$, l'ambient père est soit un ambient classique, soit une souche migrante avec une marque que nous nommerons l, soit une souche ouverte.

Si l'ambient est une souche, la réduction est une étape sortant OUT 1 pour cette souche *(4d)*. Pour toutes les autres souches contenant le processus impliqué, l'étape est une étape interne *(4c,i)* suivie par l'extrusion de la portée introduite. Pour toutes les autres souches, cette étape est une étape externe *(4b,ii)*.

En ce qui concerne la relation $\prec$, la situation est semblable au cas précédent. Nous appelons toujours J l'ensemble des noms associés aux souches migrantes contenant le trou de $E(\,\cdot\,)$, et K l'ensemble des scions de R_i. La relation est étendue par les couples $k \prec i$ pour tout $k \in K$ et par $i \prec j$ pour tout $j \in J$. Comme auparavant, cette extension n'introduit pas de cycles puisque pour tout $k, j \in K \times J$, nous avons $k \prec j$ avant la réduction. Nous remarquons aussi que si le père est une souche, aucun nouveau couple utilisant sa marque l n'est ajouté à la relation.

MOVE 2 **:** La souche qui est consommée par cette étape est une souche migrante de R. Soit i le nom du scion aussi consommé par cette étape. Par l'hypothèse d'induction *3*, ce scion est en dehors de la souche consommée. Nous avons donc des contextes $A_0(\,\cdot\,)$, $A_1(\,\cdot\,)$, et $A_2(\,\cdot\,)$ dont les trous ne sont en parallèle avec aucun processus, et des processus étendus R_0, R_1, R_2, et R_i, tels que :

$$\begin{aligned} R &= A_0\big(R_0 \mid A_1(R_1 \mid i) \mid A_2(R_2 \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,])\big) \\ Q' &\equiv \nu\widetilde{x} \setminus i.A_0\big(R_0 \mid A_1(R_1 \mid n[P_i \mid R_i]) \mid A_2(R_2)\big) \end{aligned}$$

En ce qui concerne la souche migrante consommée, cette étape est une étape de complétion *(4a)*. Pour toutes les souches contenant le trou de $A_0(\,\cdot\,)$, cette étape est une étape interne *(4c,ii)*. Pour toutes les souches de A_1 contenant le trou de $A_1(\,\cdot\,)$, cette étape est une étape entrante MOVE 2 *(4f)*. Pour toutes les souches de A_2 contenant le trou de $A_2(\,\cdot\,)$, cette étape est une étape sortant MOVE 2 *(4e)*. Pour toutes les autres souches, cette étape est une étape externe *(4b,iii)*.

Soit K l'ensemble des scions de R_i, soient J_1 et J_2 les ensembles des noms associés aux souches migrantes de A_1 et A_2 contenant les trous de $A_1(\,\cdot\,)$ et $A_2(\,\cdot\,)$, respectivement. La relation $\prec$ est modifiée comme suit. Tous les couples mentionnant i sont supprimés. Tous les couples $k \prec j$ avec $k, j \in K \times J_2$ sont aussi supprimés. Par contre, les couples $k \prec j$ sont ajoutés pour tout $k, j \in K \times J_1$. Cet ajout ne peut introduire de cycle puisque avant la réduction nous avions $k \prec i \prec j$ et par hypothèse d'induction *3*.

OPEN 1 **:** Une souche ouverte est créée par cet étape. Pour toutes les autres souches, cette étape est soit interne *(4c,i)*, soit externe *(4b,i)*. La relation $\prec$ n'est pas modifiée.

OPEN 2 **:** Une souche ouverte est dissoute par cette étape. En ce qui concerne cette souche, l'étape est une étape finale. Pour toutes les autres souches, c'est soit une étape interne *(4c,i)*, soit une étape externe *(4b,i)*. La relation $\prec$ n'est pas modifiée.

REPL**,** RECV **:** Ces étapes peuvent libérer de nouveaux processus en contexte d'évaluation, mais ces processus ne font pas partie du fragment étendu du calcul. Pour toutes les souches, cette étape est soit interne, soit externe.

Dans tous les cas, nous obtenons un processus R' pouvant contenir des restrictions en contexte d'évaluation. Il suffit donc d'extruder ces restrictions jusqu'à la racine après les α-conversions nécessaires pour obtenir un processus satisfaisant la propriété 1. Nous remarquons que les propriétés 2 et 3 ne dépendent pas de cette extrusion. □

Chaque souche décrite dans le lemme A.1.2 a la possibilité d'effectuer une étape finale *(4a)*, donc chaque souche peut être éliminée en une étape 2. En itérant ces étapes, nous obtenons le corollaire :

Lemme A.1.3 (Complétion) *Pour tous $(P, Q) \in \mathcal{A} \times \mathcal{E}$ tels que $P \rightarrow_{12C}^{*} Q$, il existe un $P' \in \mathcal{A}$ tel que $Q \rightarrow_{2}^{*} P'$.*

Nous comparons maintenant deux étapes successives $\rightarrow_1$ et $\rightarrow_2$ impliquant la même extension d'un ambient avec une réduction $\longrightarrow$ du calcul originel :

Lemme A.1.4 (Raffinement) *Pour tout $P, P' \in \mathcal{A}$, nous avons $P \longrightarrow P'$ si et seulement si $P \rightarrow_1 \rightarrow_2 P'$ ou $P \rightarrow_C P'$.*

Preuve: Ceci est une conséquence directe du lemme A.1.2 pour une suite de deux étapes $P \rightarrow_1 \rightarrow_2 P'$. La première étape est possible si et seulement si elle est possible dans le calcul originel. Puisque P' ne comporte qu'un seul ambient étendu, la seconde étape correspond nécessairement à l'étape finale associée à la première étape. Nous obtenons ainsi le diagramme de commutation présenté à la section 3.3 en figure 3.7. □

Grâce au lemme A.1.2, nous utilisons les marques i, j et k créées par des étapes 1 pour suivre l'évolution des souches et des scions ainsi que pour associer ces étapes 1 avec les étapes 2 éliminant ces souches et ces scions. Nous étendons les souches ouvertes pour leur associer aussi une marque unique, afin d'être capable d'apparier chaque étape Open 1 avec l'étape Open 2 idoine. Plus formellement, nous pourrions étendre la sémantique étendue de telle sorte qu'un lieur νi soit créé à chaque étape Open 1, qu'une souche ouverte ait la forme $\mathtt{o}(i)\{P\}$– et que le lieur soit consommé lors de l'étape Open 2. Nous considérons cependant par la suite que nous pouvons associer de manière unique une étape Open 1 et l'étape Open 2 correspondant sans recourir à cette extension. Nous écrivons donc $\rightarrow_1^i$ pour une étape créant une souche avec la marque i, et $\rightarrow_2^i$ pour l'étape consommant cette souche.

Nous avons besoin des propriétés de commutation suivantes pour la sémantique étendue :

Lemme A.1.5 *Pour tous $(P, Q) \in \mathcal{A} \times \mathcal{E}$ tels que $P \rightarrow_{12C}^* Q$, nous avons les propriétés de commutation suivantes :*

- *$Q \rightarrow_1^i \rightarrow_1^j Q'$ implique $Q \rightarrow_1^j \rightarrow_1^i Q'$ si l'étape $\rightarrow_1^i$ n'est pas une étape* In 1 *dans un ambient qui devient une souche à cause de l'étape $\rightarrow_1^j$;*
- *$Q \rightarrow_1^i \rightarrow_2^j Q'$ implique $Q \rightarrow_2^j \rightarrow_1^i Q'$ si $i \neq j$ et si l'étape $\rightarrow_1^i$ n'est pas une étape* Out-1 *hors de la souche disparaissant à cause de l'étape $\rightarrow_2^j$ (par une étape* Move 2 *ou* Open 2*) ;*
- *$Q \rightarrow_2^i \rightarrow_1^j Q'$ implique $Q \rightarrow_1^j \rightarrow_2^i Q'$ si l'étape $\rightarrow_1^j$ n'utilise pas un processus activé par l'étape $\rightarrow_2^i$ ni l'ambient de la souche disparaissant à l'étape $\rightarrow_2^i$;*
- *$Q \rightarrow_2 \rightarrow_C Q'$ implique $Q \rightarrow_C \rightarrow_2 Q'$ si l'étape $\rightarrow_C$ n'utilise pas un processus activé par l'étape $\rightarrow_2$;*
- *$Q \rightarrow_C \rightarrow_1 Q'$ implique $Q \rightarrow_1 \rightarrow_C Q'$ et $Q \rightarrow_C \rightarrow'_C Q'$ implique $Q \rightarrow'_C \rightarrow_C Q'$ si la seconde étape n'utilise pas un processus activé par la première étape $\rightarrow_C$.*
- *$Q \rightarrow_C \rightarrow_2 Q'$ implique $Q \rightarrow_2 \rightarrow_C Q'$;*
  $Q \rightarrow_1 \rightarrow_C Q'$ implique $Q \rightarrow_C \rightarrow_1 Q'$;
  $Q \rightarrow_2^i \rightarrow_2^j Q'$ implique $Q \rightarrow_2^j \rightarrow_2^i Q'$.

Entre autres, nous avons toujours $Q \rightarrow_1^i \rightarrow_1^j Q'$ implique $Q \rightarrow_1^j \rightarrow_1^i Q'$ si la deuxième étape est interne par rapport à i, et nous avons $Q \rightarrow_1^i \rightarrow_2^j Q'$ implique $Q \rightarrow_2^j \rightarrow_1^i Q'$ si $i \neq j$ et la première étape n'est pas une étape sortante Out 1 de j.

Preuve: Pour chaque propriété de commutation, nous appliquons le lemme A.1.2 avec la suite de réductions $P \rightarrow_{12C}^* Q$, afin d'utiliser la propriété *(4)* pour vérifier que les deux étapes commutent.

Nous commençons par le cas simple $\rightarrow_C \rightarrow'_C$. Ce cas est immédiat puisque la deuxième réduction $\rightarrow'_C$ n'utilise pas un processus activé par la première réduction. S'il s'agit d'une réplication, le processus répliqué est par conséquent déjà présent en contexte d'évaluation avant la réduction, et la réplication n'invalide pas l'exécution subséquente de la première étape. S'il s'agit d'une communication, l'émetteur et le récepteur sont présents avant la première réduction et la communication n'invalide pas la possibilité d'effectuer cette première réduction.

Nous étudions le cas $\rightarrow_C \rightarrow_1$. Pour se faire, nous appliquons le lemme A.1.2 à l'issue des deux réductions pour identifier par la propriété *(2)* la souche créée par l'étape $\rightarrow_1$. Cette seconde étape est de l'une des formes suivantes :

$$\begin{aligned}
\nu\widetilde{x}.F(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) &\rightarrow_1^i \nu\widetilde{x}, i.F(m[Q_i \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]) \\
\nu\widetilde{x}.F(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) &\rightarrow_1^i \nu\widetilde{x}, i.F(i \mid X^{=}m[Q_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]]) \\
\nu\widetilde{x}.F(\mathtt{open}\ n.P_o \mid n[R_i]) &\rightarrow_1^i \nu\widetilde{x}.F(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,])
\end{aligned}$$

La première étape peut donc être une étape externe ayant lieu dans F ou dans Q_i *(4b,i)*, ou une étape interne ayant lieu dans R_i *(4c,i)*. Cette première étape est donc de la forme $F'(\cdot)(\cdot) \rightarrow_C F(\cdot)(\cdot)$, ou $F'(\cdot) \rightarrow_C F(\cdot)$, ou $Q'_i \rightarrow_C Q_i$, ou $R'_i \rightarrow_C R_i$. Puisque l'étape $\rightarrow_1$ n'implique pas de processus activé par l'étape $\rightarrow_C$, le processus initial a l'une des formes suivantes :

$$\begin{gathered}
\nu\widetilde{x}.F'(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) \\
\nu\widetilde{x}.F(m[Q'_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) \\
\nu\widetilde{x}.F(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R'_i]) \\
\nu\widetilde{x}.F'(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) \\
\nu\widetilde{x}.F(X^{=}m[Q'_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) \\
\nu\widetilde{x}.F(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R'_i]]) \\
\nu\widetilde{x}.F'(\mathtt{open}\ n.P_o \mid n[R_i]) \\
\nu\widetilde{x}.F(\mathtt{open}\ n.P_o \mid n[R'_i])
\end{gathered}$$

Nous avons donc les réductions :

$$\begin{array}{rcl} \nu\widetilde{x}.F'(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) & \to_1 & \nu\widetilde{x}, i.F'(m[Q_i \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]) \\ & \to_C & \nu\widetilde{x}, i.F(m[Q_i \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(m[Q'_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) & \to_1 & \nu\widetilde{x}, i.F(m[Q'_i \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]) \\ & \to_C & \nu\widetilde{x}, i.F(m[Q_i \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R'_i]) & \to_1 & \nu\widetilde{x}, i.F(m[Q_i \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R'_i\,]) \\ & \to_C & \nu\widetilde{x}, i.F(m[Q_i \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F'(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) & \to_1 & \nu\widetilde{x}, i.F'(i \mid X^{=}m[Q_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]]) \\ & \to_C & \nu\widetilde{x}, i.F(i \mid X^{=}m[Q_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(X^{=}m[Q'_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) & \to_1 & \nu\widetilde{x}, i.F(i \mid X^{=}m[Q'_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]]) \\ & \to_C & \nu\widetilde{x}, i.F(i \mid X^{=}m[Q_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R'_i]]) & \to_1 & \nu\widetilde{x}, i.F(i \mid X^{=}m[Q_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R'_i\,]]) \\ & \to_C & \nu\widetilde{x}, i.F(i \mid X^{=}m[Q_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F'(\mathtt{open}\ n.P_o \mid n[R_i]) & \to_1 & \nu\widetilde{x}.F'(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,]) \\ & \to_C & \nu\widetilde{x}.F(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(\mathtt{open}\ n.P_o \mid n[R'_i]) & \to_1 & \nu\widetilde{x}.F(\,\mathtt{o}\{P_o\}\text{–}n[\,R'_i\,]) \\ & \to_C & \nu\widetilde{x}.F(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,]) \end{array}$$

Nous étudions les cas $\to_1^i\to_1^j$ et $\to_1^i\to_C$ où $\to_1^i$ est une étape In 1, Out 1 ou Open 1. Nous pouvons donc écrire cette première étape :

$$\begin{array}{rcl} \nu\widetilde{x}.F(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) & \to_1^i & \nu\widetilde{x}, i.F(m[Q_i \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]) \\ \nu\widetilde{x}.F(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) & \to_1^i & \nu\widetilde{x}, i.F(i \mid X^{=}m[Q_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]]) \\ \nu\widetilde{x}.F(\mathtt{open}\ n.P_o \mid n[R_i]) & \to_1^i & \nu\widetilde{x}.F(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,]) \end{array}$$

avec $F(\,m[\,Q_i \mid (\,\cdot\,)\,] \mid (\,\cdot\,))$ ou $F((\,\cdot\,) \mid X^{=}m[\,Q_i \mid (\,\cdot\,)\,])$ étant le contexte d'évaluation $E(\,\cdot\,)(\,\cdot\,)$ du lemme A.1.2*(2)* pour la réduction In 1 et Out 1, et avec $F(\,\cdot\,)$ étant le contexte $E(\,\cdot\,)$ pour la réduction Open 1.

La deuxième étape est soit une étape externe *(4b,i)* ou *(4b,ii)*, soit une étape interne *(4c,i)*, soit une étape sortante Out 1 *(4d)* (les autres cas sont pour des étapes 2). Nous étudions tout d'abord les cas *(4b,i,ii)*, où la deuxième étape est de la forme $E(\,\cdot\,)(\,\cdot\,) \to_C E'(\,\cdot\,)(\,\cdot\,)$ ou $E(\,\cdot\,)(\,\cdot\,) \to_1^j \nu j.E'(\,\cdot\,)(\,\cdot\,)$ (ou les même étapes avec un contexte à un trou). Par la suite nous écrivons $\widetilde{y}$ pour $\widetilde{x}$ si la deuxième étape ne crée pas de souche migrante, et pour $\widetilde{x}, j$ si elle crée une souche migrante. Nous avons trois sous-cas selon la position du processus impliqué par la deuxième étape. Ce processus peut appartenir à $F((\,\cdot\,) \mid (\,\cdot\,))$ (ou $F(\,\cdot\,)$), à Q_i pour les étapes In 1 ou Out 1, ou être l'ambient $m[\,Q_i\,]$ pour les étape In 1 et Out 1 si l'ambient n'est pas une souche.

Pour le premier sous-cas, nous remarquons que les étapes $\to_1$ et $\to_C$ ne modifient pas la structure de l'arbre des ambients ni le contenu des trous, et nous pouvons écrire cette étape :

$$\begin{array}{rcl} \nu\widetilde{x}.F(\,m[\,Q_i \mid (\,\cdot\,)\,] \mid (\,\cdot\,)) & \to_{1C} & \nu\widetilde{y}.F'(\,m[\,Q_i \mid (\,\cdot\,)\,] \mid (\,\cdot\,)) \\ \nu\widetilde{x}.F((\,\cdot\,) \mid X^{=}m[\,Q_i \mid (\,\cdot\,)\,]) & \to_{1C} & \nu\widetilde{y}.F'((\,\cdot\,) \mid X^{=}m[\,Q_i \mid (\,\cdot\,)\,]) \\ \nu\widetilde{x}.F(\,\cdot\,) & \to_{1C} & \nu\widetilde{y}.F'(\,\cdot\,) \end{array}$$

Nous avons donc :

$$\begin{array}{rcl} \nu\widetilde{x}.F(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) & \rightarrow_{1C} & \nu\widetilde{y}.F'(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) \\ & \rightarrow_1^i & \nu\widetilde{y}.\nu i.F'(m[Q_i \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) & \rightarrow_{1C} & \nu\widetilde{y}.F'(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) \\ & \rightarrow_1^i & \nu\widetilde{y}.\nu i.F'(i \mid m[Q_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(\mathtt{open}\ n.P_o \mid n[R_i]) & \rightarrow_{1C} & \nu\widetilde{y}.F'(\mathtt{open}\ n.P_o \mid n[R_i]) \\ & \rightarrow_1^i & \nu\widetilde{y}.F'(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,]) \end{array}$$

Pour le deuxième sous-cas, la deuxième étape peut s'écrire (ce cas ne s'applique pas pour une étape Open 1) :

$$\begin{array}{rcl} \nu\widetilde{x}.F(\,m[\,Q_i \mid (\,\cdot\,)\,])(\,\cdot\,) & \rightarrow_{1C} & \nu\widetilde{y}.F(Q_i' \mid m[\,Q_i'' \mid (\,\cdot\,)\,])(\,\cdot\,) \\ \nu\widetilde{x}.F((\,\cdot\,) \mid X^{=}m[\,Q_i \mid (\,\cdot\,)\,]) & \rightarrow_{1C} & \nu\widetilde{y}.F((\,\cdot\,) \mid Q_i' \mid X^{=}m[\,Q_i'' \mid (\,\cdot\,)\,]) \end{array}$$

avec Q_i étant soit $\mathbf{0}$ si l'étape est interne à m, soit j s'il s'agit d'une étape sortante Out 1 de m. Nous avons donc :

$$\begin{array}{rcl} \nu\widetilde{x}.F(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) & \rightarrow_{1C} & \nu\widetilde{y}.F(Q_i' \mid m[Q_i''] \mid n[\mathtt{in}\ m.P_i \mid R_i]) \\ & \rightarrow_1^i & \nu\widetilde{y}.\nu i.F(Q_i' \mid m[Q_i'' \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) & \rightarrow_{1C} & \nu\widetilde{y}.F(Q_i' \mid X^{=}m[Q_i'' \mid n[\mathtt{out}\ m.P_i \mid R_i]]) \\ & \rightarrow_1^i & \nu\widetilde{y}.\nu i.F(Q_i' \mid i \mid X^{=}m[Q_i'' \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]]) \end{array}$$

En ce qui concerne le troisième sous-cas, la seconde étape ne peut pas être une étape $\rightarrow_C$, puisqu'une telle étape n'implique pas un ambient. C'est donc une étape $\rightarrow_1$. Si la première étape est une étape In 1, les réductions ne commutent pas, ce qui est en accord avec le lemme, puisque l'ambient cible de la première étape devient une souche à cause de la deuxième étape, ce qui empêche l'étape In 1 d'avoir lieu en seconde position (la définition de cette étape exige que l'ambient cible ne soit pas une souche). Nous étudions donc le cas où la première étape est une étape Out 1. La deuxième étape s'écrit donc :

$$\nu\widetilde{x}.F((\,\cdot\,) \mid m[\,Q_i \mid (\,\cdot\,)\,]) \rightarrow_1 \nu\widetilde{y}.F'((\,\cdot\,) \mid Xm[\,Q_i' \mid (\,\cdot\,)\,])$$

Le processus Q_i est transformé en Q_i' si la deuxième étape est une étape In 1 ou Out 1, qui est initiée par un processus dans l'ambient m. Dans le cas d'une étape Open 1, nous avons $Q_i = Q_i'$. Nous avons donc :

$$\begin{array}{rcl} \nu\widetilde{x}.F(m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) & \rightarrow_{1C} & \nu\widetilde{y}.F'(Xm[Q_i' \mid n[\mathtt{out}\ m.P_i \mid R_i]]) \\ & \rightarrow_1^i & \nu\widetilde{y}.\nu i.F'(i \mid Xm[Q_i' \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]]) \end{array}$$

Nous étudions maintenant le cas où la deuxième étape est une étape interne *(4c,i)*. Dans ce cas, la seule possibilité est que cette étape implique le processus R_i. Cette deuxième étape est de la forme $\nu\widetilde{x}.\nu i.E(i)(\,\bar{\imath}\{P_i\}\text{–}n[\,R_i\,]) \rightarrow_{1C} \nu\widetilde{x}.\nu i.E(i)(\,\bar{\imath}\{P_i\}\text{–}n[\,R_i'\,])$ (si la première étape crée une souche migrante, c'est à dire In 1 et Out 1), ou de la forme $\nu\widetilde{x}.E(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,]) \rightarrow_{1C} \nu\widetilde{x}.E(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i'\,])$ si la première étape est une étape Open. Dans tous les cas nous avons $R_i \rightarrow_{1C} R_i'$. Par conséquent, nous avons donc les réductions :

$$\begin{array}{rcl} \nu\widetilde{x}.F(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) & \rightarrow_{1C} & \nu\widetilde{x}.F(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i']) \\ & \rightarrow_1^i & \nu\widetilde{x}.\nu i.F(m[Q_i \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i'\,]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) & \rightarrow_{1C} & \nu\widetilde{x}.F(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i']]) \\ & \rightarrow_1^i & \nu\widetilde{x}.\nu i.F(i \mid X^{=}m[Q_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i'\,]]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(\texttt{open}\ n.P_o \mid n[R_i]) & \to_{1C} & \nu\widetilde{x}.F(\texttt{open}\ n.P_o \mid n[R'_i]) \\ & \to_1^i & \nu\widetilde{x}.F(\ \texttt{o}\{P_o\}\text{–}n[\,R'_i\,]\ ) \end{array}$$

Nous étudions maintenant le cas où la deuxième étape est une étape sortante Out 1 *(4d)*. Dans ce cas, la deuxième étape est soit de la forme :

$$\nu\widetilde{x}.\nu i.E(i)(\ \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]\ ) \to_1^j \nu\widetilde{x},\nu i,j.E(i)(j \mid \bar{\imath}\{P_i\}\text{–}n[\,R'_i\,]\ )$$

soit de la forme :

$$\nu\widetilde{x}.E(\ \texttt{o}\{P_o\}\text{–}n[\,R_i\,]\ ) \to_1^j \nu\widetilde{x},\nu j.E(j \mid \texttt{o}\{P_o\}\text{–}n[\,R'_i\,]\ )$$

Nous remarquons que nécessairement R_i est de la forme (à réarrangement de la composition parallèle près) $n'[\texttt{out}\ n.S_i \mid S'_i] \mid S''_i$.

Nous avons donc, selon les trois cas correspondant aux trois premières réductions possibles, en commutant les réductions :

$$\begin{array}{rcl} \nu\widetilde{x}.F(m[Q_i] \mid n[\texttt{in}\ m.P_i \mid R_i]) & \to_1^j & \nu\widetilde{x}.\nu j.F(m[Q_i] \mid j \mid n[\texttt{in}\ m.P_i \mid R'_i]) \\ & \to_1^i & \nu\widetilde{x}.\nu i,j.F(m[Q_i \mid i] \mid j \mid \bar{\imath}\{P_i\}\text{–}n[\,R'_i\,]\ ) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(X^{=}m[Q_i \mid n[\texttt{out}\ m.P_i \mid R_i]]) & \to_1^j & \nu\widetilde{x}.\nu j.F(X^{=}m[Q_i \mid j \mid n[\texttt{out}\ m.P_i \mid R'_i]]) \\ & \to_1^i & \nu\widetilde{x}.\nu i,j.F(i \mid j \mid X^{=}m[Q_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R'_i\,]\,]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(\texttt{open}\ n.P_o \mid n[R_i]) & \to_1^j & \nu\widetilde{x}.\nu j.F(\texttt{open}\ n.P_o \mid j \mid n[R'_i]) \\ & \to_1^i & \nu\widetilde{x}.\nu j.F(j \mid \texttt{o}\{P_o\}\text{–}n[\,R'_i\,]\ ) \end{array}$$

Nous étudions les cas $\to_1^i\to_2^j$ avec $i \neq j$ et l'étape $\to_1^i$ n'étant pas une étape Out 1 hors d'une souche disparaissant par l'étape $\to_2^j$ (par une étape Move 2 ou Open 2).

Comme pour le cas $\to_1^i\to_1^j$, nous précisons la forme de la première réduction créant une souche :

$$\begin{array}{rcl} \nu\widetilde{x}.F(m[Q_i] \mid n[\texttt{in}\ m.P_i \mid R_i]) & \to_1^i & \nu\widetilde{x},i.F(m[Q_i \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]\ ) \\ \nu\widetilde{x}.F(X^{=}m[Q_i \mid n[\texttt{out}\ m.P_i \mid R_i]]) & \to_1^i & \nu\widetilde{x},i.F(i \mid X^{=}m[Q_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]\,]) \\ \nu\widetilde{x}.F(\texttt{open}\ n.P_o \mid n[R_i]) & \to_1^i & \nu\widetilde{x}.F(\ \texttt{o}\{P_o\}\text{–}n[\,R_i\,]\ ) \end{array}$$

avec $F(\ m[\,Q_i \mid (\,\cdot\,)\,] \mid (\,\cdot\,))$ ou $F((\,\cdot\,) \mid X^{=}m[\,Q_i \mid (\,\cdot\,)\,]\ )$ étant le contexte d'évaluation $E(\,\cdot\,)(\,\cdot\,)$ du lemme A.1.2*(2)* pour les réduction In 1 et Out 1, et avec $F(\,\cdot\,)$ étant le contexte $E(\,\cdot\,)$ pour la réduction Open 1.

Grâce au lemme A.1.2*(4)*, nous savons que la deuxième étape est soit une étape finale *(4a)*, soit une étape externe *(4b,i)* ou *(4b,iii)*, soit une étape interne *(4c,i)* ou *(4c,ii)*, soit une étape sortante Move 2 *(4e)*, soit une étape entrante Move 2 *(4f)*.

Dans la suite, nous écrivons $\widetilde{y}$ pour $\widetilde{x}$ si l'étape $\to_2^j$ correspond aux cas *(4b,i)* ou *(4c,i)*, et nous écrivons $\widetilde{y}$ pour $\widetilde{x} \setminus j$ avec $j \in \widetilde{x}$ pour les autres cas.

Le cas de l'étape finale ne peut arriver ici puisque nous avons $i \neq j$. Nous étudions donc d'abord le cas des étapes externes. Comme auparavant, nous avons trois sous-cas selon le processus qui est modifié par l'étape externe. Il peut s'agir d'un processus de $F(\,\cdot\,)(\,\cdot\,)$ (ou $F(\,\cdot\,)$), de Q_i (si la première étape n'est pas une étape Open 1), ou de l'ambient m (dans le cas où la première étape est une étape Out 1 et si cet ambient est une souche).

Nous étudions le premier sous-cas. Nous remarquons que les trous de $F(\,\cdot\,)(\,\cdot\,)$ étant en parallèle, ils sont soit tous les deux à l'intérieur de la souche réduite par l'étape 2, soit tous les deux à l'extérieur. Dans tous les cas ces trous sont toujours en parallèle à l'issue de la seconde étape et ne sont pas impliqués par la réduction. Nous pouvons écrire cette étape :

$$\begin{array}{rcl} \nu\widetilde{x}.F(\ m[\,Q_i \mid (\,\cdot\,)\,]\ \mid (\,\cdot\,)) & \to_2^j & \nu\widetilde{y}.F'(\ m[\,Q_i \mid (\,\cdot\,)\,]\ \mid (\,\cdot\,)) \\ \nu\widetilde{x}.F((\,\cdot\,) \mid X^{=}m[\,Q_i \mid (\,\cdot\,)\,]\ ) & \to_2^j & \nu\widetilde{y}.F'((\,\cdot\,) \mid X^{=}m[\,Q_i \mid (\,\cdot\,)\,]\ ) \\ \nu\widetilde{x}.F(\,\cdot\,) & \to_2^j & \nu\widetilde{y}.F'(\,\cdot\,) \end{array}$$

Nous avons donc :

$$\begin{array}{rcl} \nu\widetilde{x}.F(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) & \to_2^j & \nu\widetilde{y}.F'(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) \\ & \to_1^i & \nu\widetilde{y}.\nu i.F'(m[Q_i \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) & \to_2^j & \nu\widetilde{y}.F'(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) \\ & \to_1^i & \nu\widetilde{y}.\nu i.F'(i \mid m[Q_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(\mathtt{open}\ n.P_o \mid n[R_i]) & \to_2^j & \nu\widetilde{y}.F'(\mathtt{open}\ n.P_o \mid n[R_i]) \\ & \to_1^i & \nu\widetilde{y}.F'(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,]) \end{array}$$

Nous considérons maintenant le deuxième sous-cas, où l'étape externe consomme une souche de Q_1. Dans ce cas, par le lemme *(4b,i,iii)* nous avons la réduction (ce cas ne s'applique pas pour une première étape Open 1) :

$$\begin{array}{rcl} \nu\widetilde{x}.F(\,m[\,Q_i \mid (\,\cdot\,)\,]\,)(\,\cdot\,) & \to_2^j & \nu\widetilde{y}.F'(\,m[\,Q_i' \mid (\,\cdot\,)\,]\,)(\,\cdot\,) \\ \nu\widetilde{x}.F((\,\cdot\,) \mid X^{=}m[\,Q_i \mid (\,\cdot\,)\,]\,) & \to_2^j & \nu\widetilde{y}.F'((\,\cdot\,) \mid X^{=}m[\,Q_i' \mid (\,\cdot\,)\,]\,) \end{array}$$

Intuitivement, le contexte F est modifié s'il s'agit d'une étape Move 2 dont le scion est dans ce contexte. Nous avons donc :

$$\begin{array}{rcl} \nu\widetilde{x}.F(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) & \to_2^j & \nu\widetilde{y}.F'(m[Q_i'] \mid n[\mathtt{in}\ m.P_i \mid R_i]) \\ & \to_1^i & \nu\widetilde{y}.\nu i.F'(m[Q_i' \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) & \to_2 & \nu\widetilde{y}.F'(X^{=}m[Q_i' \mid n[\mathtt{out}\ m.P_i \mid R_i]]) \\ & \to_1^i & \nu\widetilde{y}.\nu i.F'(i \mid X^{=}m[Q_i' \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i\,]]) \end{array}$$

Le troisième sous-cas correspond à la consommation de la souche de l'ambient m. Ce cas n'est pas possible pour les actions In 1 ou Open 1 qui exigent que l'ambient cible ne soit pas une souche. Le seul cas est donc une action Out 1 hors d'une souche consommée par l'étape $\to_2^j$. Ce cas est explicitement indiqué comme étant un cas où la commutation n'est pas garantie, puisque la destination finale de l'ambient sortant de m dépend de la position de m au moment où l'étape Out 1 a lieu : c'est à ce moment que le scion marquant la destination est créé.

Nous étudions maintenant le cas où l'étape $\to_2^j$ est une étape interne *(4c,i,ii)*. Dans ce cas, la seule possibilité est que cette étape implique le processus R_i. Cette deuxième étape est de la forme $\nu\widetilde{x}.\nu i.E(i)(\,\bar{\imath}\{P_i\}\text{–}n[\,R_i\,]) \to_2^j \nu\widetilde{y}.\nu i.E(i)(\,\bar{\imath}\{P_i\}\text{–}n[\,R_i'\,])$ (si la première étape crée une souche migrante, c'est à dire In 1 et Out 1), ou de la forme $\nu\widetilde{x}.E(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,]) \to_2^j \nu\widetilde{y}.E(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i'\,])$ si la première étape est une étape Open. Dans tous les cas nous avons $R_i \to_2^j R_i'$. Par conséquent, nous avons donc les réductions :

$$\begin{array}{rcl} \nu\widetilde{x}.F(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i]) & \to_2^j & \nu\widetilde{y}.F(m[Q_i] \mid n[\mathtt{in}\ m.P_i \mid R_i']) \\ & \to_1^i & \nu\widetilde{y}.\nu i.F(m[Q_i \mid i] \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i'\,]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i]]) & \to_2^j & \nu\widetilde{y}.F(X^{=}m[Q_i \mid n[\mathtt{out}\ m.P_i \mid R_i']]) \\ & \to_1^i & \nu\widetilde{y}.\nu i.F(i \mid X^{=}m[Q_i \mid \bar{\imath}\{P_i\}\text{–}n[\,R_i'\,]]) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(\mathtt{open}\ n.P_o \mid n[R_i]) & \to_2^j & \nu\widetilde{y}.F(\mathtt{open}\ n.P_o \mid n[R_i']) \\ & \to_1^i & \nu\widetilde{y}.F(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i'\,]) \end{array}$$

Nous étudions maintenant le cas des étapes sortantes et entrantes Move 2 *(4e,4f)*. Deux sous-cas sont à distinguer lorsque la première étape est une étape In 1 ou Out 1, en fonction de la

position du scion (respectivement de la souche) pour une étape sortante (respectivement entrante). Dans un premier cas il est dans le contexte $F(\cdot)(\cdot)$, dans le second cas il est dans le processus Q_i. Lorsque la première étape est une étape Open 1, seul le premier sous-cas s'applique.

Dans tous les cas, le processus R_i s'écrit sous la forme $G(\overline{\jmath}\{P_j\}\text{–}n'[R_j])$ dans le cas d'une étape sortante (et $G(j)$ dans le cas d'une étape entrante) puisque le contexte $G(\cdot)$ des propriétés *(4e,4f)* est un contexte d'évaluation.

En ce qui concerne le premier sous-cas, pour une étape sortante, nous réécrivons le contexte $F(\cdot)(\cdot)$ en $F'(\cdot)(\cdot)(j)$ et le contexte $F(\cdot)$ en un contexte $F'(\cdot)(j)$. Pour les étapes entrantes, nous remplaçons simplement le j du dernier trou par une souche $\overline{\jmath}\{P_j\}\text{–}n'[R_j]$.

Nous avons donc les réductions $\rightarrow_2^j$ suivantes pour une étape sortante :

$$\begin{array}{rl}
 & \nu\widetilde{x}.\nu i.F'(\,m[\,Q_i \mid i\,] \mid \overline{\imath}\{P_i\}\text{–}n[\,G(\overline{\jmath}\{P_j\}\text{–}n'[\,R_j\,])\,]\,)(j) \\
\rightarrow_2^j & \nu\widetilde{y}.\nu i.F'(\,m[\,Q_i \mid i\,] \mid \overline{\imath}\{P_i\}\text{–}n[\,G(\mathbf{0})\,]\,)(\,n'[\,P_j \mid R_j\,]\,) \\
 & \nu\widetilde{x}.\nu i.F'(\,i \mid m[\,Q_i \mid \overline{\imath}\{P_i\}\text{–}n[\,G(\overline{\jmath}\{P_j\}\text{–}n'[\,R_j\,])\,]\,]\,)(j) \\
\rightarrow_2^j & \nu\widetilde{y}.\nu i.F'(\,i \mid m[\,Q_i \mid \overline{\imath}\{P_i\}\text{–}n[\,G(\mathbf{0})\,]\,]\,)(\,n'[\,P_j \mid R_j\,]\,) \\
 & \nu\widetilde{x}.F'(\,\mathtt{o}\{P_o\}\text{–}n[\,G(\overline{\jmath}\{P_j\}\text{–}n'[\,R_j\,])\,]\,)(j) \\
\rightarrow_2^j & \nu\widetilde{y}.F'(\,\mathtt{o}\{P_o\}\text{–}n[\,G(\mathbf{0})\,]\,)(\,n'[\,P_j \mid R_j\,]\,)
\end{array}$$

Les étapes entrantes sont identiques, en inversant le rôle de la souche et du scion j, et donc en inversant la position finale de l'ambient $n'[\,P_j \mid R_j\,]$. Nous ne les répéterons pas ici.

Puisque les étapes Stub et Scion sont stables par application de contexte d'évaluation étendu non restreint, nous avons donc les réductions suivantes :

$$\begin{array}{rl}
 & \nu\widetilde{x}.F'(\,m[\,Q_i\,] \mid n[\,\mathtt{in}\ m.P_i \mid G(\overline{\jmath}\{P_j\}\text{–}n'[\,R_j\,])\,]\,)(j) \\
\rightarrow_2^j & \nu\widetilde{y}.F'(\,m[\,Q_i\,] \mid n[\,\mathtt{in}\ n.P_i \mid G(\mathbf{0})\,]\,)(\,n'[\,P_j \mid R_j\,]\,) \\
\rightarrow_1^i & \nu\widetilde{y}.\nu i.F'(\,m[\,Q_i \mid i\,] \mid \overline{\imath}\{P_i\}\text{–}n[\,G(\mathbf{0})\,]\,)(\,n'[\,P_j \mid R_j\,]\,)
\end{array}$$

ou

$$\begin{array}{rl}
 & \nu\widetilde{x}.F'(\,m[\,Q_i \mid \,n[\,\mathtt{out}\ m.P_i \mid G(\overline{\jmath}\{P_j\}\text{–}n'[\,R_j\,])\,]\,]\,)(j) \\
\rightarrow_2^j & \nu\widetilde{y}.F'(\,m[\,Q_i \mid \,n[\,\mathtt{out}\ n.P_i \mid G(\mathbf{0})\,]\,]\,)(\,n'[\,P_j \mid R_j\,]\,) \\
\rightarrow_1^i & \nu\widetilde{y}.\nu i.F'(\,i \mid m[\,Q_i \mid \overline{\imath}\{P_i\}\text{–}n[\,G(\mathbf{0})\,]\,]\,)(\,n'[\,P_j \mid R_j\,]\,)
\end{array}$$

ou

$$\begin{array}{rl}
 & \nu\widetilde{x}.F'(\mathtt{open}\ n.P_o \mid \,n[\,G(\overline{\jmath}\{P_j\}\text{–}n'[\,R_j\,])\,]\,)(j) \\
\rightarrow_2^j & \nu\widetilde{y}.F'(\mathtt{open}\ n.P_o \mid \,n[\,G(\mathbf{0})\,]\,)(\,n'[\,P_j \mid R_j\,]\,) \\
\rightarrow_1^i & \nu\widetilde{y}.F'(\,\mathtt{o}\{P_o\}\text{–}n[\,G(\mathbf{0})\,]\,])(\,n'[\,P_j \mid R_j\,]\,)
\end{array}$$

Les étapes entrantes sont identiques, en inversant la souche et le scion j ainsi que la position de l'ambient n' résultant après l'étape $\rightarrow_2^j$; nous les omettons ici.

Nous étudions maintenant le deuxième sous-cas, lorsque le scion (respectivement la souche) est dans Q_i pour l'étant sortante (respectivement entrante). Le processus Q_i s'écrit donc sous la forme $F'(j)$ (respectivement $F'(\overline{\jmath}\{P_j\}\text{–}n'[\,R'j\,])$). Nous avons donc la réduction :

$$\begin{array}{rl}
 & \nu\widetilde{x}.\nu i.F(\,m[\,F'(j) \mid i\,] \mid \overline{\imath}\{P_i\}\text{–}n[\,G(\overline{\jmath}\{P_j\}\text{–}n'[\,R_j\,])\,]\,) \\
\rightarrow_2^j & \nu\widetilde{y}.\nu i.F(\,m[\,F'(\,n'[\,P_j \mid R_j\,]\,) \mid i\,] \mid \overline{\imath}\{P_i\}\text{–}n[\,G(\mathbf{0})\,]\,) \\
 & \nu\widetilde{x}.\nu i.F(\,i \mid m[\,F'(j) \mid \overline{\imath}\{P_i\}\text{–}n[\,G(\overline{\jmath}\{P_j\}\text{–}n'[\,R_j\,])\,]\,]\,) \\
\rightarrow_2^j & \nu\widetilde{y}.\nu i.F(\,i \mid m[\,F'(\,n'[\,P_j \mid R_j\,]\,) \mid \overline{\imath}\{P_i\}\text{–}n[\,G(\mathbf{0})\,]\,])
\end{array}$$

Nous avons donc les réductions commutées suivantes :

$$\begin{array}{rl}
 & \nu\widetilde{x}.F(\,m[\,F'(j)\,] \mid n[\,\mathtt{in}\ m.P_i \mid G(\overline{\jmath}\{P_j\}\text{–}n'[\,R_j\,])\,]\,) \\
\rightarrow_2^j & \nu\widetilde{y}.F(\,m[\,F'(\,n'[\,P_j \mid R_j\,]\,)\,] \mid n[\,\mathtt{in}\ n.P_i \mid G(\mathbf{0})\,]\,) \\
\rightarrow_1^i & \nu\widetilde{y}.\nu i.F(\,m[\,F'(\,n'[\,P_j \mid R_j\,]\,) \mid i\,] \mid \overline{\imath}\{P_i\}\text{–}n[\,G(\mathbf{0})\,]\,)
\end{array}$$

ou

$$\begin{array}{rl} & \nu\widetilde{x}.F(\,m[\,F'(j) \mid n[\,\mathtt{out}\ m.P_i \mid G(\,\overline{\jmath}\{P_j\}\text{–}n'[\,R_j\,]\,)\,]\,]\,) \\ \rightarrow_2^j & \nu\widetilde{y}.F(\,m[\,F'(\,n'[\,P_j \mid R_j\,]\,) \mid n[\,\mathtt{out}\ n.P_i \mid G(\mathbf{0})\,]\,]) \\ \rightarrow_1^i & \nu\widetilde{y}.\nu i.F(\,i \mid m[\,F'(\,n[\,P_j \mid R_j\,]\,) \mid \overline{\imath}\{P_i\}\text{–}n[\,G(\mathbf{0})\,]\,]) \end{array}$$

Les étapes entrantes sont identiques en inversant la souche et le scion j, ainsi que la position de l'ambient n' après l'étape $\rightarrow_2^j$.

Nous étudions maintenant la commutation des étapes $\rightarrow_2^i\rightarrow_1^j$ et $\rightarrow_2^i\rightarrow_C$. Nous considérons pour ce faire la souche disparaissant lors de l'étape $\rightarrow_2^i$. S'il s'agit d'une souche migrante, par le lemme A.1.2*(2)*, il existe un contexte non restreint E tel que $Q \equiv \nu\widetilde{x}.E(i)(\,\overline{\imath}\{P_i\}\text{–}n[\,R_i\,]\,)$ avec $i \in \widetilde{x}$. La première étape $\rightarrow_2^i$ correspond à une étape finale :

$$\nu\widetilde{x}.E(i)(\,\overline{\imath}\{P_i\}\text{–}n[\,R_i\,]\,) \rightarrow_2^i \nu\widetilde{x} \setminus i.E(\,n[\,P_i \mid R_i\,]\,)(\mathbf{0})$$

De manière semblable, dans le cas d'une souche ouverte, la première étape est :

$$\nu\widetilde{x}.E(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,]\,) \rightarrow_2^i \nu\widetilde{x}.E(P_o \mid R_o)$$

Nous considérons maintenant la position du processus impliqué dans la deuxième étape $\rightarrow_{1C}$. Cela ne peut être ni P_i, ni P_o, puisque ces processus ont été activés par l'étape $\rightarrow_2^i$. Cela ne peut être une étape entrant dans, sortant de ou dissolvant l'ambient n puisque le lemme le précise. Il s'agit donc soit d'une étape purement interne à R_i, soit d'une étape du contexte E.

Dans le premier sous-cas, nous avons donc une étape $R_i \rightarrow_{1C} R_i'$, donc les étapes :

$$\begin{array}{rcl} \nu\widetilde{x} \setminus i.E(\,n[\,P_i \mid R_i\,]\,)(\mathbf{0}) & \rightarrow_{1C} & \nu\widetilde{x} \setminus i.E(\,n[\,P_i \mid R_i'\,]\,)(\mathbf{0}) \\ \nu\widetilde{x}.E(P_o \mid R_i) & \rightarrow_{1C} & \nu\widetilde{x}.E(P_o \mid R_i') \end{array}$$

Nous avons donc, par la propriété *(4c,i)*, la réduction :

$$\begin{array}{rcl} \nu\widetilde{x}.E(i)(\,\overline{\imath}\{P_i\}\text{–}n[\,R_i\,]\,) & \rightarrow_{1C} & \nu\widetilde{x}.E(i)(\,\overline{\imath}\{P_i\}\text{–}n[\,R_i'\,]\,) \\ & \rightarrow_2^i & \nu\widetilde{x} \setminus i.E(\,n[\,P_i \mid R_i'\,]\,)(\mathbf{0}) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.E(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,]\,) & \rightarrow_{1C} & \nu\widetilde{x}.E(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i'\,]\,) \\ & \rightarrow_2^i & \nu\widetilde{x}.E(P_o \mid R_i') \end{array}$$

Nous considérons maintenant le sous-cas où la réduction $\rightarrow_{1C}$ a lieu dans le contexte E. Cette réduction est de la forme (avec $\nu\widetilde{y}$ pouvant être égal à $\nu\widetilde{x}$ ou $\nu\widetilde{x}.\nu j$) :

$$\begin{array}{rcl} \nu\widetilde{x} \setminus i.E(\,\cdot\,)(\,\cdot\,) & \rightarrow_{1C} & \nu\widetilde{y} \setminus i.E'(\,\cdot\,)(\,\cdot\,) \\ \nu\widetilde{x}.E(\,\cdot\,) & \rightarrow_{1C} & \nu\widetilde{y}.E'(\,\cdot\,) \end{array}$$

Nous avons donc (par les propriétés *(4b,i,ii)*) :

$$\begin{array}{rcl} \nu\widetilde{x}.E(i)(\,\overline{\imath}\{P_i\}\text{–}n[\,R_i\,]\,) & \rightarrow_{1C} & \nu\widetilde{y}.E'(i)(\,\overline{\imath}\{P_i\}\text{–}n[\,R_i\,]\,) \\ & \rightarrow_2^i & \nu\widetilde{y} \setminus i.E'(\,n[\,P_i \mid R_i\,]\,)(\mathbf{0}) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.E(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,]\,) & \rightarrow_{1C} & \nu\widetilde{y}.E'(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,]\,) \\ & \rightarrow_2^i & \nu\widetilde{y}.E'(P_o \mid R_i) \end{array}$$

Nous étudions maintenant les derniers cas de commutation : $\rightarrow_C\rightarrow_2^i$ et $\rightarrow_2^j\rightarrow_2^i$. Nous appliquons le lemme A.1.2 pour la souche consommée par la deuxième étape $\rightarrow_2^i$ avant que la première étape n'ait lieu (la première étape ne créant pas cette souche, elle est déjà présente). Par la propriété *(2)*, le processus initial est de la forme $\nu\widetilde{x}.E(i)(\,\overline{\imath}\{P_i\}\text{–}n[\,R_i\,]\,)$ ou $\nu\widetilde{x}.E(\,\mathtt{o}\{P_o\}\text{–}n[\,R_i\,]\,)$. Nous procédons par cas suivant la propriété *(4)* pour la première réduction. Dans la suite nous notons $\widetilde{y}$ pour $\widetilde{x}$ dans les cas *(4b,i)* et *(4c,i)*; nous notons $\widetilde{y}$ pour $\widetilde{x} \setminus j$ dans les autres cas. Nous remarquons que nous avons toujours $j \neq i$.

Nous commençons par le cas d'une étape externe *(4b,i,iii)*. Nous avons donc comme première réduction :

$$\begin{array}{rcl} \nu\widetilde{x}.E(\,\cdot\,)(\,\cdot\,) & \rightarrow_{2C} & \nu\widetilde{y}.E'(\,\cdot\,)(\,\cdot\,) \\ \nu\widetilde{x}.E(\,\cdot\,) & \rightarrow_{2C} & \nu\widetilde{y}.E'(\,\cdot\,) \end{array}$$

Donc nous avons :

$$\begin{array}{rcl} \nu\widetilde{x}.E(i)(\,\overline{\imath}\{P_i\}\text{–}n[\,R_i\,]\,) & \rightarrow_2^i & \nu\widetilde{x}\setminus i.E(\,n[\,P_i \mid R_i\,]\,)(\mathbf{0}) \\ & \rightarrow_{2C} & \nu\widetilde{y}\setminus i.E'(\,n[\,P_i \mid R_i\,]\,)(\mathbf{0}) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.E(\,\circ\{P_o\}\text{–}n[\,R_i\,]\,) & \rightarrow_2^i & \nu\widetilde{x}.E(P_o \mid R_i) \\ & \rightarrow_{2C} & \nu\widetilde{y}.E'(P_o \mid R_i) \end{array}$$

Nous étudions maintenant le cas d'une étape interne *(4c,i,ii)*. Dans ce cas nous avons $\nu\widetilde{x}.R_i \rightarrow_{2C} \nu\widetilde{y}.R'_i$, et nous en déduisons :

$$\begin{array}{rcl} \nu\widetilde{x}.E(i)(\,\overline{\imath}\{P_i\}\text{–}n[\,R_i\,]\,) & \rightarrow_2^i & \nu\widetilde{x}\setminus i.E(\,n[\,P_i \mid R_i\,]\,)(\mathbf{0}) \\ & \rightarrow_{2C} & \nu\widetilde{y}\setminus i.E(\,n[\,P_i \mid R'_i\,]\,)(\mathbf{0}) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.E(\,\circ\{P_o\}\text{–}n[\,R_i\,]\,) & \rightarrow_2^i & \nu\widetilde{x}.E(P_o \mid R_i) \\ & \rightarrow_{2C} & \nu\widetilde{y}.E(P_o \mid R'_i) \end{array}$$

Nous étudions maintenant les deux cas plus complexes d'une étape sortante ou entrante Move 2 *(4e, 4f)*. Nous détaillons le cas de l'étape sortant, l'étape entrante étant identique en inversant souche et scion de j ainsi que la position finale de l'ambient m.

Le processus initial s'écrit sous la forme $\nu\widetilde{x}.F(i)(\,\overline{\imath}\{P_i\}\text{–}n[\,G(\,\overline{\jmath}\{P_j\}\text{–}m[\,R_j\,]\,)\,]\,)(j)$ ou sous la forme $\nu\widetilde{x}.F(\,\circ\{P_o\}\text{–}n[\,G(\,\overline{\jmath}\{P_j\}\text{–}m[\,R_j\,]\,)\,]\,)(j)$.

Nous avons comme première étape :

$$\begin{array}{rcl} \nu\widetilde{x}.F(i)(\,\overline{\imath}\{P_i\}\text{–}n[\,G(\,\overline{\jmath}\{P_j\}\text{–}m[\,R_j\,]\,)\,]\,)(j) & \rightarrow_2^j & \nu\widetilde{y}.F(i)(\,\overline{\imath}\{P_i\}\text{–}n[\,G(\mathbf{0})\,]\,)(\,m[\,P_j \mid R_j\,]\,) \\ \nu\widetilde{x}.F(\,\circ\{P_o\}\text{–}n[\,G(\,\overline{\jmath}\{P_j\}\text{–}m[\,R_j\,]\,)\,]\,)(j) & \rightarrow_2^j & \nu\widetilde{y}.F(\,\circ\{P_o\}\text{–}n[\,G(\mathbf{0})\,]\,)(\,m[\,P_j \mid R_j\,]\,) \end{array}$$

Le contexte F étant un contexte d'évaluation, nous avons donc les réductions commutées :

$$\begin{array}{rcl} \nu\widetilde{x}.F(i)(\,\overline{\imath}\{P_i\}\text{–}n[\,G(\,\overline{\jmath}\{P_j\}\text{–}m[\,R_j\,]\,)\,]\,)(j) & \rightarrow_2^i & \nu\widetilde{x}\setminus i.F(\,n[\,P_i \mid G(\,\overline{\jmath}\{P_j\}\text{–}m[\,R_j\,]\,)\,])(\mathbf{0})(j) \\ & \rightarrow_2^j & \nu\widetilde{y}\setminus i.F(\,n[\,P_i \mid G(\mathbf{0})\,]\,)(\mathbf{0})(\,m[\,P_j \mid R_j\,]\,) \end{array}$$

ou

$$\begin{array}{rcl} \nu\widetilde{x}.F(\,\circ\{P_o\}\text{–}n[\,G(\,\overline{\jmath}\{P_j\}\text{–}m[\,R_j\,]\,)\,]\,)(j) & \rightarrow_2^i & \nu\widetilde{x}.F(P_o \mid G(\,\overline{\jmath}\{P_j\}\text{–}m[\,R_j\,]\,))(j) \\ & \rightarrow_2^j & \nu\widetilde{y}.F(P_o \mid G(\mathbf{0}))(\,m[\,P_j \mid R_j\,]\,) \end{array}$$

□

Nous prouvons maintenant le lemme fondamental qui permet d'associer à une série de réductions étendues entre deux processus classiques une série de réductions classiques entre ces deux mêmes processus.

Lemme A.1.6 (Correction) *Pour tout $P, P' \in \mathcal{A}$, si $P \rightarrow_{12C}^* P'$, alors $P \rightarrow^* P'$.*

Preuve: Nous prouvons la propriété par induction sur le nombre n d'étapes de réduction de $P \rightarrow_{12C}^{n} P'$. Pour une paire donnée de processus (P, P'), les commutations effectuées dans la preuve ne modifient pas le nombre d'étapes de P à P'. Par conséquent, ce nombre n'est pas explicité lors des commutations.

La propriété est immédiate pour aucune réduction. Sinon, la première étape est soit une étape $\rightarrow_C$, soit une étape $\rightarrow_1$. En effet, P étant un ambient classique, il ne contient aucune souche, donc aucune étape $\rightarrow_2$ ne peut avoir lieu.

Si nous avons $P \rightarrow_C Q \rightarrow_{12C}^{*} P'$, alors Q est un processus classique, et nous concluons par induction pour la dérivation $Q \rightarrow_{12C}^{*} P'$.

Sinon nous avons une série de réductions de la forme $P \rightarrow_1^i Q \rightarrow_{12C}^{*} P'$. Puisque Q contient une souche (créée par la première étape), et que P' n'en contient pas, il existe dans la série de réductions une étape $\rightarrow_2^i$ consommant la souche. Nous pouvons ainsi grâce au lemme A.1.2 partitionner toutes les étapes entre $\rightarrow_1^i$ et $\rightarrow_2^i$ en étapes internes, externes, sortantes Out 1, sortantes Move 2 et entrantes Move 2.

Nous appelons par la suite la *profondeur* d'un ambient ou d'une souche comme étant le nombre d'ambients ou de souches l'englobant. Nous appelons *profondeur maximale* d'un processus étendu le maximum des profondeurs des ambients ou souches en contexte d'évaluation étendu présents dans ce processus.

Nous procédons maintenant par induction sur la profondeur maximale de la souche créée par l'étape $\rightarrow_1^i$, c'est à dire sur le maximum de la profondeur de ses sous-ambients par rapport à la souche.

Si la profondeur maximale est nulle, la souche ne possède initialement aucun sous-ambient. Nous remarquons que puisque cet ambient est une souche, un autre ambient ne peut pas le prendre pour cible d'une étape In 1. De plus, comme c'est la première souche créée par la réduction, elle ne contient aucun scion et aucune étape ne peut créer de scion à l'intérieur de cette souche (les seules possibilités serait une étape In 1 dans cette souche, ce qui est impossible, ou une étape Move 2, ce qui exigerait la présence préalable d'un scion). Par conséquent, il est impossible que cette souche ait un sous-ambient jusqu'à sa disparition, par l'étape $\rightarrow_2^i$. En particulier, une étape sortante Out 1 ne peut avoir lieu entre l'étape $\rightarrow_1^i$ et l'étape $\rightarrow_2^i$.

Par conséquent, le lemme A.1.5 nous indique que l'étape $\rightarrow_2^i$ commute avec toutes les étapes précédentes jusqu'à être l'étape immédiatement après $\rightarrow_1^i$. Nous avons donc une série de réductions de la forme :

$$P \rightarrow_1^i \rightarrow_2^i R \rightarrow_{12C}^{*} P'$$

pour un certain processus étendu R. Nous utilisons le lemme A.1.4 pour prouver que R est un processus classique tel que $P \rightarrow R$. Nous concluons par induction sur le reste de la série de réductions $R \rightarrow_{12C}^{*} P'$.

Nous prouvons maintenant le cas général de l'induction sur la profondeur maximale. Si aucune étape sortante Out 1 n'est présente entre l'étape $\rightarrow_1^i$ et l'étape $\rightarrow_2^i$, comme dans le cas précédent l'étape $\rightarrow_2^i$ commute avec toutes les étapes précédentes pour se retrouver en seconde position, et nous concluons par induction sur la longueur de la série de réductions.

Si une étape Out 1 est présente entre les étapes $\rightarrow_1^i$ et $\rightarrow_2^i$, la commutation ne peut plus avoir lieu. Nous partitionnons alors la série de réductions de la manière suivante :

$$P \rightarrow_1^i Q \, (\rightarrow_{12C}^{ext})^{*} \rightarrow^j R \rightarrow_{12C}^{*} P'$$

où $\rightarrow_{12C}^{ext}$ est une étape externe, et où $\rightarrow^j$ est la première étape qui ne soit pas une étape externe. Nous remarquons que, comme auparavant, il est impossible qu'un ambient initie une étape In 1 ayant pour cible la souche i. Puisque c'est la première souche créée, il est impossible qu'un processus extérieur à la souche y introduise un scion, puisque pour ce faire les deux seules possibilités sont soit une étape In 1, soit une étape entrante Move 2 si un scion a déjà été introduit par un processus extérieur. De manière symétrique, une étape sortant Move 2 ne peut avoir lieu que si un ambient interne a exporté un scion à l'extérieur de la souche i. Ceci peut avoir lieu par une autre étape Move 2 si un scion a déjà été exporté, ou par une étape sortante Out 1. Par conséquent la première étape qui n'est pas une étape externe est soit une étape interne, soit une étape sortante Out 1. De plus, si c'est une étape interne cela ne peut être une étape $\rightarrow_2$ puisque initialement aucune souche n'existe à l'intérieur de la souche i, et puisqu'aucune souche n'y est créée (l'étape $\rightarrow^j$ est la première étape interne) ni importée (aucune étape entrante Move 2 n'a lieu avant l'étape $\rightarrow^j$). Par conséquent deux cas sont possibles pour cet étape $\rightarrow^j$:

1. Il s'agit d'une étape $\rightarrow_C$. Toutes les étapes précédentes étant externes alors que cette étape est interne, elle n'utilise donc pas un processus activé par une étape précédente. Nous pouvons

donc commuter cette étape avec toutes les étapes précédentes par le lemme A.1.5 jusqu'au tout début de la série de réductions, qui a donc désormais la forme :

$$P \to_C R \to^*_{12C} P'$$

avec R étant un processus classique. Nous concluons par induction sur la longueur de la série de réductions.

2. Il s'agit d'une étape interne $\to_1$ ou d'une étape sortante Out 1. Toutes les étapes précédentes étant externes, cette étape n'utilise pas un processus activé par une étape précédente, ni ne transforme en souche un ambient qui est la cible d'une étape In 1 précédente (puisque l'étape considérée est la première étape qui n'est pas externe). Par conséquent elle peut commuter avec toutes les étapes précédentes par le lemme A.1.5, et être mise en première position. Comme cette étape $\to^j_1$ crée une souche qui est à l'intérieur de la souche créée par $\to^i_1$, sa profondeur maximale est nécessairement moindre. Nous concluons donc par l'hypothèse d'induction sur la profondeur maximale.

□

La preuve du théorème de correction est maintenant immédiate :

Preuve du théorème 2: Nous vérifions que toutes les conditions du lemme A.1.1 sont satisfaites : (1) est le lemme A.1.3 ; (2 $\Rightarrow$) est le lemme A.1.4 ; (2 $\Leftarrow$) est le lemme A.1.6 ; (3) est impliqué par notre définition des barbes : les étapes 2 n'effacent que des souches qui, par définition, n'ont pas de barbes. □

A.2 Correction de la traduction étendue

Une configuration du join calcul obtenue par traduction d'un processus ambient peut se réduire de nombreuses manières pour donner des configurations qui ne sont pas nécessairement dans l'image de la traduction. La factorisation de la preuve de correction nous permet d'aborder ce problème dans cette section en ignorant les complications dues à l'algorithme de synchronisation.

Afin de simplifier l'analyse par cas des réductions possibles d'une configuration issue de la traduction, nous explicitons la structure d'une approximation $\mathcal{T} \subset \mathcal{J}$ de l'image de la traduction, qui possède des propriétés de fermeture plus simples par rapport à la réduction (lemme A.2.7) et qui contient l'image de la traduction (lemme A.2.6). Afin d'obtenir une correspondance opérationnelle entre les réductions du calcul des ambients étendus et les réductions des configurations obtenues par traduction, nous introduisons des relations de simplification opérant sur $\mathcal{T}$ qui permettent d'obtenir des configurations qui sont dans l'image de la traduction après réduction. Nous étudions les propriétés de commutation de ces simplifications. Les diagrammes de commutation et de correspondance opérationnelle étant nombreux et devant être assemblés pour obtenir la relation recherchée, nous utilisons la technique des diagrammes décroissants de Van Oostrom [Oos94] pour cet assemblage. Le résultat est une une bisimulation hybride barbue entre les ambients étendus et leur traduction dans le join calcul. Nous déduisons enfin de ces résultats nos théorèmes principaux.

Dans cette section nous notons $\bigwedge_{i\in[1..n]} D_i$ pour $D_1, \ldots, D_n$ et $\prod_{i\in[1..n]} \mathcal{S}_i$ pour $\mathcal{S}_1 \parallel \ldots \parallel \mathcal{S}_n$.

A.2.1 Traduction des ambients étendus dans le join calcul

Nous commençons par définir en figure A.1 une nouvelle traduction $\to_{tr}$ qui étend la traduction de la figure 3.3 aux constructions du calcul étendu, qui satisfait des propriétés globales dans le choix des noms, et qui réalise tous les dépliages des locations et des `def` en contexte d'évaluation. Par la suite, nous utilisons les abréviations r, o, et h pour les noms *reloc*, *opening*, et *here* respectivement Nous rappelons que la traduction d'un processus ambient contenant de nombreux ambients consiste en une configuration du join calcul possédant de multiples instances de l'algorithme s'exécutant en parallèle, donc de nombreuses instances de la définition D de la figure 3.3. Nous associons un indice à chaque ambient et ambient étendu présent en contexte d'évaluation. Nous notons ces indices n, m et p et notons D^n la définition D de la figure 3.3 dans laquelle on a substitué h^n pour *here*, r^n pour *reloc*, o^n pour *opening*, et x^n pour chaque nom x défini par D. Comme dans le cas de la traduction, nous supposons que les noms introduits par la traduction étendue (c'est à dire les noms définis par les contrôleurs, les noms des locations et les noms définis pour les continuations) sont différents des noms libres du processus traduit. De plus, nous supposons que le processus est

$$\frac{e,\alpha,P \to_{tr} S_1 \qquad e,\alpha,Q \to_{tr} S_2}{e,\alpha,P \mid Q \to_{tr} S_1 \oplus S_2} \qquad \frac{e,\alpha,P \to_{tr} S}{e,\alpha,\nu a.P \to_{tr} S \oplus (\emptyset;\emptyset;\{a\};\top;\top;\mathbf{0})} \qquad \frac{e,\alpha,P \to_{tr} S}{e,\alpha,\nu i.P \to_{tr} S}$$

$$\frac{}{e,\alpha,\mathbf{0} \to_{tr} (\emptyset;\emptyset;\emptyset;\top;\top;\mathbf{0})} \qquad \frac{}{e,\alpha,i \to_{tr} (\emptyset;\emptyset;\emptyset;\top;\top;\mathbf{0})}$$

$$\frac{}{e,\alpha,\mathtt{in}\ a.P \to_{tr} (\emptyset;\emptyset;\emptyset;\top;\kappa() \triangleright [\![P]\!]_e;e.in(a,\kappa))}$$

$$\frac{}{e,\alpha,\mathtt{out}\ a.P \to_{tr} (\emptyset;\emptyset;\emptyset;\top;\kappa() \triangleright [\![P]\!]_e;e.out(a,\kappa))}$$

$$\frac{}{e,\alpha,\mathtt{open}\ a.P \to_{tr} (\emptyset;\emptyset;\emptyset;\top;\kappa() \triangleright [\![P]\!]_e;e.open(a,\kappa))}$$

$$\frac{}{e,\alpha,(n).P \to_{tr} (\emptyset;\emptyset;\emptyset;\top;\kappa(n) \triangleright [\![P]\!]_e;e.recv(\kappa))} \qquad \frac{}{e,\alpha,!P \to_{tr} (\emptyset;\emptyset;\emptyset;\top;\kappa() \triangleright [\![P]\!]_e \mid \kappa();\kappa())}$$

$$\frac{}{e,\alpha,\langle n\rangle \to_{tr} (\emptyset;\emptyset;\emptyset;\top;\top;send(n))}$$

$$\frac{e^n,\alpha h^n,P \to_{tr} (N;I;L;C;E;M) \qquad dn(D^n),i^n \text{ sont frais}}{e,\alpha,\ a[\,P\,] \to_{tr} (N \uplus \{n\};I \uplus \{i^n\};L;C \parallel H_a^n(E)(M);\top;e.amb(i^n,a,e^n))}$$

$$\frac{e^n,\alpha h^n,P \to_{tr} (N;I;L;C;E;M) \qquad dn(D^n),i^n,\kappa \text{ sont frais}}{e,\alpha,\ \overline{\imath}\{Q\}\text{–}a[\,P\,] \to_{tr} (N \uplus \{n\};I \uplus \{i^n\};L;C \parallel H_{a,\kappa}^n(E,\kappa() \triangleright [\![Q]\!]_{e^n})(r^n(e^m,\kappa) \mid M);\top;\mathbf{0})}$$

$$\frac{e^n,\alpha h^n,P \to_{tr} (N;I;L;C;E;M) \qquad dn(D^n),i^n,\kappa \text{ sont frais}}{e,\alpha,\ \mathtt{o}\{Q\}\text{–}a[\,P\,] \to_{tr} (N \uplus \{n\};I \uplus \{i^n\};L;C \parallel H_a^n(E,\kappa() \triangleright [\![Q]\!]_{e^n})(o^n(\kappa) \mid M);\top;\mathbf{0})}$$

FIG. A.1 – Définition de la traduction étendue

α-renommé de telle sorte que chaque nom restreint soit lié au plus une fois, distinct des noms libres du processus et distinct des noms introduits par la traduction.

La traduction associe à tout processus ambient étendu P accompagné d'une interface e et d'une chaîne de locations non vide α un sextuplet S composé de :

- l'ensemble N des indices des ambients de P ;
- un ensemble I correspondant aux uids de ces ambients ;
- un ensemble L des noms d'ambients restreints de P ;
- une configuration C dont toute location est une sous-location de α ;
- une définition E ;
- un processus M.

Nous supposons dans la suite que les ensembles N, I, L, $dn(C)$ et $dn(E)$ sont deux à deux disjoints. Ceci ne restreint pas la traduction puisque les indices peuvent être choisis distincts de tous les noms, les noms de continuation de $dn(E)$ peuvent eux aussi être choisis, et les noms de L ont été renommés.

Nous nous donnons un opérateur binaire $\oplus$ sur les traductions défini de la manière suivante :

$$\begin{aligned}&(N_1;I_1;L_1;C_1;E_1;M_1) \oplus (N_2;I_2;L_2;C_2;E_2;M_2)\\ \stackrel{\text{def}}{=}\ &(N_1 \uplus N_2;I_1 \uplus I_2;L_1 \uplus L_2;C_1 \parallel C_2;E_1,E_2;M_1 \mid M_2)\end{aligned}$$

avec $N_1 \cap N_2 = \emptyset$, $I_1 \cap I_2 = \emptyset$, $L_1 \cap L_2 = \emptyset$, $dn(C_1) \cap dn(C_2) = \emptyset$ et $dn(E_1) \cap dn(E_2) = \emptyset$.

Dans la définition suivante, nous appelons m l'indice de l'ambient contenant le scion i, et appelons H_a^n le contexte à deux trous de la forme :

$$H_a^n(\,\cdot\,)(\,\cdot\,) = \alpha h^n[D^n,(\,\cdot\,) : s^n(a,i^n,e,\emptyset) \mid (\,\cdot\,)]$$

et $H_{a,\kappa}^n$ le contexte à deux trous de la forme :

$$H_a^n(\,\cdot\,)(\,\cdot\,) = \alpha h^n[D^n,(\,\cdot\,) : s^n(a,i^n,e,\{\mathrm{L}\ b\ \kappa\}) \mid (\,\cdot\,)]$$

pour L étant soit In, soit Out, et b étant un nom.

Nous définissons maintenant la traduction étendue à la racine d'un processus ambient étendu P. Nous appelons 0 l'indice de la racine, et notons D_l^0 et D_t^0 pour D_l et D_t introduits en section 3.4 après renommage de e en e^0. Si $e^0, h^0, P \rightarrow_{tr} N; I; L; C; E; M$, et si les noms de h^0, a, e, p, t, i^0, *yes*, et *lock* n'apparaissent pas dans P, nous définissons $[\![P]\!]^{t\sharp}$ comme étant

$$h^0[D_l^0, E, D_t^0, \bigwedge_{a\in L}\texttt{fresh}\ a, \bigwedge_{i\in I\cup\{i^0\}}\texttt{uid}\ i : p(t) \mid s^0(a, i^0, e, \emptyset) \mid M] \parallel C$$

La définition de $[\![\ \cdot\ ]\!]^{t\sharp}$ suppose que tous les scions sont restreints, ce qui est le cas pour tout processus ambient étendu issu d'une série de réductions partant d'un ambient classique. Nous remarquons que la traduction étendue efface les scions i et les restriction νi, confirmant le fait que les scions représentent des cibles passives de migration, remplacées par les indices dans la traduction étendue.

La définition suivante rend plus explicite la forme générale des traductions. Intuitivement, l'ensemble N est l'ensemble des indices des ambients, les chaînes $(\alpha^n)_{n\in N^+}$ représentent la structure arborescente des ambients ; les ensembles F et L contiennent respectivement les noms libres et restreints des ambients ; l'ensemble I contient l'ensemble des uids des ambients ; pour tout ambient d'indice n, le processus S^n correspond à son état, les messages M^n sont les processus s'exécutant dans n, la définition D^n est le contrôleur d'ambient de n, et la définition E^n correspond aux continuations qui ont été créés par n.

Définition A.2.1 (Etat d'approximation) *Une configuration du join calcul est un état d'approximation, notée $P \in \mathcal{T}$, quand il existe un ensemble N d'indices contenant l'indice 0 tel que :*

1. **Structure** *:*

$$\begin{aligned} P &= h^0[D_l^0, E^0, D_t^0, \bigwedge_{a\in L}\texttt{fresh}\ a, \bigwedge_{i\in I}\texttt{uid}\ i : p(t) \mid s^0(a, i^0, e, \emptyset) \mid M^0] \\ &\parallel \prod_{n\in N^+} \alpha^n h^n[D^n, E^n : S^n \mid M^n] \end{aligned}$$

$$\begin{aligned} E^n &= \bigwedge_{\kappa\in K_A^n} \kappa() \triangleright [\![Q^\kappa]\!]_{e^n} \\ &, \bigwedge_{\kappa\in K_B^n} \kappa(a) \triangleright [\![Q^\kappa]\!]_{e^n} \\ &, \bigwedge_{\kappa\in K_C^n} \kappa() \triangleright [\![Q^\kappa]\!]_{e^n} \mid \kappa() \end{aligned}$$

   pour des processus ambients classiques $(Q^\kappa)_\kappa$, et pour des ensembles de noms deux à deux disjoints : F, L, I, $\{a\}$, les noms de e, et pour chaque $n \in N$ les ensembles K_A^n, K_B^n, K_C^n, $\{h^n\}$, et les noms définis dans D^n.

   L'ensemble I doit être indexé par $n \in N$ et les noms i^n de I doivent être deux à deux distincts. Nous notons N^+ pour $N \setminus \{0\}$. Les chaînes de noms α^n sont composées de noms de $\{h^n \mid n \in N^+\}$ et sont indexées par $n \in N^+$.

   Nous notons : $K_{AB}^n \stackrel{def}{=} K_A^n \cup K_B^n$, $K_{AB} \stackrel{def}{=} \bigcup_{n\in N} K_{AB}^n$, $K_A \stackrel{def}{=} \bigcup_{n\in N} K_A^n$, $K_B \stackrel{def}{=} \bigcup_{n\in N} K_B^n$, $K_C \stackrel{def}{=} \bigcup_{n\in N} K_C^n$, et $K \stackrel{def}{=} K_{AB} \cup K_C$.

2. **Messages** *: Pour tout $n \in N$, le processus M^n est une composition parallèle de messages envoyés sur des noms de l'ensemble $\{amb^n, in^n, out^n, open^n, recv^n, send^n, sub_{in}^n, sub_{out}^n, o^n, r^n\} \cup K^n$. De plus, le processus M^0 ne contient aucun message envoyé sur les noms de l'ensemble $\{o^0, r^0\}$. Chaque message de la forme $send^n(a)$ dans M^n est tel que $a \in F \cup L$.*

3. **Journal** *: Pour tout $n \in N^+$, l^n est un ensemble d'entrées de la forme* In b κ *ou* Out b κ *avec $b \in F \cup L$ et $\kappa \in K_A$.*

4. **Continuations de migration** *: Nous disons qu'un message $sub_{in}^n(i, c, \kappa)$ ou $sub_{out}^n(i, c, \kappa)$ est* vieux *lorsqu'il n'existe aucun $m \in N$ tel que S^m ait la forme $s^m(\_, i, \_, \_)$.*

   Tout nom $\kappa \in K_a$ n'est présent que dans sa définition E^n, dans de vieux messages, et au plus dans un seul des cas suivants (où nous avons $m \in N+$ et $p \in N$) :

   (a) *$in^p(b, \kappa)$ une fois dans M^p (ou $out^p(b, \kappa)$ une fois dans M^p), avec $b \in F \cup L$.*

   (b) *$sub_{in}^p(i^m, b, \kappa)$ une fois dans M^p et* In b κ *une fois dans l^m (ou $sub_{out}^p(i^m, b, \kappa)$ une fois dans M^p et* Out b κ *une fois dans l^m) avec $b \in F \cup L$.*

   (c) *$r^m(e^p, \kappa)$ une fois dans M^m et au plus une entrée* In b κ *(ou* Out b κ*) dans l^m, avec $b \in F \cup L$.*

   (d) *dans $S^m = \texttt{go}\ h^p; I_{b,e^m,e^p} \mid \kappa() \mid Flush(l^m, in^m, out^m, \kappa)$, avec au plus une entrée* In b κ *dans l^m (ou* Out b κ *dans l^m).*

   (e) $open^p(b, \kappa)$ *une fois dans* M^p*, avec* $b \in F \cup L$*.*
   (f) $o^m(\kappa)$ *une fois dans* M^m*.*
   (g) $\kappa()$ *une fois dans* M^p*.*
5. **Continuations de communication** *: Tout nom* $\kappa \in K_b$ *n'est présent que dans sa définition* E^n *et au plus une fois dans un message* $\kappa(a)$ *(alors* $a \in F \cup L$*) ou dans un message* $recv^p(\kappa)$ *pour* $p \in N$*.*
6. **Réplication** *: Tout nom* $\kappa \in K_c$ *n'est présent que dans sa définition* E^n *et dans un message unique* $\kappa()$ *de* M^n*.*
7. **État de l'ambient** *: Pour tout* $n \in N^+$*, l'une des propriétés suivantes est satisfaite :*
   (a) *soit* $S^n = s^n(b, i^n, e^m, l^n)$ *pour un* $m \in N$ *et un* $b \in F \cup L$*; nous disons alors que l'ambient est* vivant*. Nous avons aussi* $\alpha^n = \alpha^m h^m$ *et la configuration* P *contient un et un seul des messages suivants :*
      i. $amb^p(i^n, b, e^n)$ *pour* $p \in N$*. Dans ce cas nous avons* $\alpha^p h^p \beta = \alpha^n$ *pour une chaîne* β *de noms de locations* h^s *avec* s *ouvert (voir ci-dessous).*
      ii. $r^n(e^p, \kappa)$ *pour* $\kappa \in K_a^n$ *et* $p \in N$*.*
      iii. $o^n(\kappa)$ *pour* $\kappa \in K_a^n$*.*
   (b) *soit* P *ne contient aucun message de la forme* $r^n(\_, \_)$*,* $o^n(\_)$*, ou* $amb^p(i^n, \_, \_)$ *pour tout* $p \in N$*, alors*
      i. *soit* $S^n = f^n(e^m)$ *pour un* $m \in N$*; nous disons alors que* n *est* ouvert*, et nous avons nécessairement* $\alpha^n = \alpha^m h^m$*.*
      ii. *soit* $S^n = \texttt{go}\ h^p; I_{b,e^n,e^p} \mid \kappa() \mid Flush(l^n, in^n, out^n, \kappa)$ *pour un* $p \in N$*, un* $b \in F \cup L$*, et un* $\kappa \in K_a^n$*. Nous disons alors que* n *est* transitoire*.*

Par la suite, nous appelons les conditions de la définition A.2.1 les *conditions d'approximation*. Nous introduisons aussi des notations pour rendre plus précises les notions d'équivalence structurelle et d'étape de réduction, définies ci-après pour deux processus ou configurations $P, Q \in \mathcal{J}$. Par la suite nous utiliserons souvent la même notation pour processus et configurations, excepté lorsque la distinction est nécessaire.

- Nous notons $P \equiv_{AC0} Q$ quand P et Q sont structurellement équivalents en utilisant uniquement les règles d'associativité et de commutativité P0–P2 et D0–D2 et la fermeture par contexte d'évaluation. Par la suite nous identifierons les processus qui sont $\equiv_{AC0}$-équivalents.
- Nous notons $P \Rrightarrow Q$ quand P et Q sont structurellement équivalents en utilisant $\equiv_{AC0}$, la règle Tree uniquement pour déplier des locations de P, la règle Scope seulement pour enlever des lieurs `def` de P, et la fermeture par contexte d'évaluation.
- Nous notons $P \Rrightarrow\!\!\!\!\!\!\Rightarrow Q$ quand $P \Rrightarrow Q$ et si Q ne contient aucun lieur `def` en contexte d'évaluation ni aucune location repliée en contexte d'évaluation.
- Soient $\mathcal{S}, \mathcal{S}'$ deux configurations du join calcul. Nous avons $\mathcal{S} \sim_\alpha \mathcal{S}'$ si et seulement si il existe une substitution injective σ des noms définis de $\mathcal{S}$ vers des noms non libres dans $\mathcal{S}$ telle que $\mathcal{S}' = \mathcal{S}\sigma$. Nous appelons ce renommage α*-conversion globale*. Par la suite, nous considérons la fermeture réflexive transitive de $\sim_\alpha$ contenant l'α-conversion, que nous notons toujours $\sim_\alpha$. Nous remarquons que cette relation nous permet d'effectuer une α-conversion sur les noms définis qui ne sont pas sous un `def`.
- Étant donnée une famille d'étapes de réduction $\rightarrow_z \subseteq \rightarrow$, nous notons $P \mapsto_z Q$ si nous avons $P \rightarrow_z Q$ et si l'étape $P \rightarrow_z Q$ n'utilise que l'équivalence $\equiv_{AC0}$ avant et après la réduction au lieu de l'équivalence structurelle à une exception près : dans le cas d'une étape Go, il est autorisé de replier la location migrant (et bien sûr ses sous-locations) avant l'étape. Par conséquent aucune α-conversion ne peut avoir lieu dans l'étape $P \rightarrow_z Q$.
- Nous notons $P \rightarrow_d P'$ si $P \rightarrow P'$ et cette étape est soit une étape Comm, soit une étape Flush.
- Nous notons $P =_d Q$ si Q est identique à P excepté pour des définitions `fresh` a et `uid` i qui ont été déplacées d'une location dépliée à une autre.
- Nous définissons $\Rightarrow_d \stackrel{\text{def}}{=} =_d \cap\, \mathcal{J} \times \mathcal{T}$ et $\twoheadrightarrow_d \stackrel{\text{def}}{=} =_\alpha \Rrightarrow \mapsto_d^* \Rightarrow_d$.
- Nous définissons $\twoheadrightarrow_z \stackrel{\text{def}}{=} \sim_\alpha \mapsto_z \twoheadrightarrow_d \cap\, \mathcal{T} \times \mathcal{T}$, pour une famille donnée d'étapes de réduction $\rightarrow_z \subseteq \rightarrow$.

Les relations et notations précédentes ont été introduites dans le but de pouvoir contrôler la structure des processus manipulés, sans avoir à considérer tous les processus leur étant structurellement équivalents.

Nous prouvons maintenant des propriétés de ces relations.

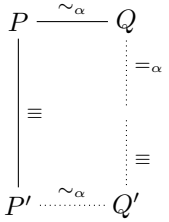
Lemme A.2.2 *Nous avons les propriétés suivantes :*

1. *La relation $\sim_\alpha$ est une bisimulation forte préservant les barbes pour les réductions $\rightarrow$ et les réductions $\mapsto$.*
2. *La notion d'état d'approximation est stable par la relation $\sim_\alpha$.*
3. *Soient $P, P', Q, Q'' \in \mathcal{J}$ tels que $P \sim_\alpha P'$, $P' \Rrightarrow Q''$, et $P \equiv Q$, alors il existe Q' tel que $Q \sim_\alpha Q'$ et $P' \equiv Q'$ sans utiliser d'α-conversion.*
4. *Soient $P, P', Q \in \mathcal{J}$ tels que $P \sim_\alpha \Rrightarrow P'$ et $P \equiv Q$, alors nous avons $Q \sim_\alpha \Rrightarrow P'$.*
5. *Soient $P, P', Q \in \mathcal{J}$ tels que $P \Lleftarrow P'$ et $P \sim_\alpha Q$, alors il existe $Q' \in \mathcal{J}$ tel que $P' \sim_\alpha Q'$ et $Q \Lleftarrow Q'$.*
6. *Soient $P, P', Q \in \mathcal{J}$ tels que $P \mapsto P'$ et $P \Rrightarrow Q$ alors il existe Q' tel que $Q \mapsto =_\alpha \Rrightarrow Q'$ et $P' =_\alpha \Rrightarrow Q'$.*
7. *Soient $P, P', Q \in \mathcal{J}$ tels que $P \mapsto_d P'$ et $P \Rrightarrow Q$ alors il existe Q' tel que $Q \mapsto_d Q'$ et $P' \Rrightarrow Q'$.*
8. *Soient P, P', Q tels que $P \mapsto_d P'$ et $P \mapsto =_\alpha \Rrightarrow Q$, alors soit $P' =_\alpha \Rrightarrow Q$, soit il existe $Q' \in \mathcal{J}$ tels que $P' \mapsto =_\alpha \Rrightarrow Q'$ et $Q \mapsto_d Q'$. De plus, $\mapsto_d$ est fortement normalisante.*
9. *Soient P, P', Q tels que $P \mapsto_d P'$ et $P \mapsto_d Q$, alors soit $P' = Q$, soit il existe un Q' tel que $P' \mapsto_d Q'$ et $Q \mapsto_d Q'$.*
10. *Si $P \in \mathcal{T}$, alors $P \not\rightarrow_d$.*
11. *$=_d \mapsto = \mapsto =_d$, $=_\alpha =_d = =_d =_\alpha$, et $\Lleftarrow =_d \subseteq =_d \Lleftarrow$.*
12. *$=_d \twoheadrightarrow_d \subseteq \twoheadrightarrow_d$, $\Rrightarrow \twoheadrightarrow_d \subseteq \twoheadrightarrow_d$, et $\mapsto_d \twoheadrightarrow_d \subseteq \twoheadrightarrow_d$.*

Preuve: Nous prouvons les propriétés dans l'ordre. Puisque la relation $\equiv_{AC0}$ est réversible, nous ne détaillons pas son utilisation pour les propriétés l'utilisant.

Propriété (1) Le fait que $\sim_\alpha$ soit une bisimulation forte est immédiat puisque le renommage n'empêche aucune réduction d'avoir lieu : la structure des termes est préservée, ainsi que le lien entre noms définis et leur utilisation (règles Comm, Flush, Join et Go). En ce qui concerne l'équivalence structurelle, la relation $\sim_\alpha$ commute immédiatement avec $\equiv_{AC0}$, avec la règle Tree, et avec $=_\alpha$ (nous rappelons que nous avons $=_\alpha \subseteq \sim_\alpha$). Nous étudions maintenant la commutation de $\sim_\alpha$ et de la règle Scope. Le résultat est immédiat dans le cas d'un repliage, puisque les noms renommés par $\sim_\alpha$ de la définition qui est repliée peuvent être renommés après repliage par $=_\alpha$. Dans le cas du dépliage, nous considérons les noms de la définition qui est dépliée et qui sont renommés par $\sim_\alpha$. Ceux-ci sont en fait renommés par $=_\alpha$ et ce renommage peut empêcher le dépliage. Nous procédons alors en deux étapes : nous les renommons tout d'abord en des noms frais par $=_\alpha$, nous effectuons le dépliage, et nous fermons le diagramme par $\sim_\alpha$.

Nous avons donc le diagramme :



En ce qui concerne les barbes, celles-ci étant définies comme les noms sur lesquels un message est présent tel que ce nom ne soit pas défini dans la configuration, leur observation est aussi préservée.

Propriété (2) Soient $P \in \mathcal{T}$ et $P' \in \mathcal{J}$ tel que $P \sim_\alpha P'$. Nous voulons montrer que $P' \in \mathcal{T}$. Ceci est immédiat puisque le renommage $\sim_\alpha$ ne capture aucun nom libre et est injectif en ce qui concerne l'α-conversion globale.

Propriété (3) Nous procédons par cas selon l'étape élémentaire d'équivalence structurelle utilisée $P \equiv Q$. Le résultat s'étend immédiatement à la fermeture transitive réflexive.

α-conversion Nous avons $P =_\alpha Q$. Nous prenons $Q' = P'$ et nous avons $Q =_\alpha P =_\alpha \sim_\alpha Q'$. Puisque $\sim_\alpha$ contient l'α-conversion, nous avons $Q \sim_\alpha Q' = P'$.

TREE Nous avons l'équivalence suivante :

$$\beta\big[a[D' : P'], D : P\big] \;\|\; \mathcal{S} \equiv \beta.a[D' : P'] \;\|\; \beta[D : P] \;\|\; \mathcal{S}$$

Nous considérons maintenant le renommage provoqué par $\sim_\alpha$. Ce renommage ne modifie pas la structure du terme, et nous avons :

$$\beta\big[a[D' : P'], D : P\big] \;\|\; \mathcal{S} \sim_\alpha \beta_\alpha\big[a_\alpha[D'_\alpha : P'_\alpha], D_\alpha : P_\alpha\big] \;\|\; \mathcal{S}_\alpha$$

et

$$\beta_\alpha.a_\alpha[D'_\alpha : P'_\alpha] \;\|\; \beta_\alpha[D_\alpha : P_\alpha] \;\|\; \mathcal{S}_\alpha$$

Nous concluons en remarquant que nous avons (en utilisant seulement la règle TREE) :

$$\beta_\alpha\big[a_\alpha[D'_\alpha : P'_\alpha], D_\alpha : P_\alpha\big] \;\|\; \mathcal{S}_\alpha \equiv \beta_\alpha.a_\alpha[D'_\alpha : P'_\alpha] \;\|\; \beta_\alpha[D_\alpha : P_\alpha] \;\|\; \mathcal{S}_\alpha$$

SCOPE Ce cas est la raison pour laquelle nous utilisons $\sim_\alpha$ au lieu de l'α-conversion. Nous avons l'équivalence suivante :

$$\alpha[D : P \mid \texttt{def}\ D'\ \texttt{in}\ P'] \;\|\; \mathcal{S} \equiv \alpha[D, D' : P \mid P'] \;\|\; \mathcal{S}$$

Nous considérons maintenant le renommage provoqué par $\sim_\alpha$. Comme auparavant, ce renommage ne modifie pas la structure du terme. Nous remarquons que le renommage des noms définis de D' se fait par α-conversion dans $\texttt{def}\ D'\ \texttt{in}\ P'$ et par $\sim_\alpha$ lorsque le `def` est déplié. Nous avons :

$$\beta[D : P \mid \texttt{def}\ D'\ \texttt{in}\ P'] \;\|\; \mathcal{S} \sim_\alpha \beta_\alpha[D_\alpha : P_\alpha \mid \texttt{def}\ D'_\alpha\ \texttt{in}\ P'_\alpha] \;\|\; \mathcal{S}_\alpha$$

et

$$\beta[D, D' : P \mid P'] \;\|\; \mathcal{S} \sim_\alpha \beta_\alpha[D_\alpha, D'_\alpha : P_\alpha \mid P'_\alpha] \;\|\; \mathcal{S}_\alpha$$

Puisque la règle SCOPE ne s'applique que si les noms définis de D' ne sont pas libre dans toute la configuration, et puisque $\sim_\alpha$ renomme injectivement les noms libres vers des noms qui ne sont pas libres eux non plus dans la configuration, nous pouvons appliquer la règle SCOPE pour replier la configuration. En ce qui concerne l'autre direction pour cette règle, nous nous basons sur l'hypothèse additionnelle de la propriété, qui suppose l'existence d'un R tel que $\beta_\alpha[D_\alpha : P_\alpha \mid \texttt{def}\ D'_\alpha\ \texttt{in}\ P'_\alpha] \;\|\; \mathcal{S}_\alpha \twoheadrightarrow R$. Puisque la relation $\twoheadrightarrow$ déplie toutes les locations et tous les `def` sans utiliser d'α-conversion, nous avons bien :

$$\beta_\alpha[D_\alpha : P_\alpha \mid \texttt{def}\ D'_\alpha\ \texttt{in}\ P'_\alpha] \;\|\; \mathcal{S}_\alpha \equiv \beta_\alpha[D_\alpha, D'_\alpha : P_\alpha \mid P'_\alpha] \;\|\; \mathcal{S}_\alpha$$

en utilisant seulement la règle SCOPE. Nous remarquons que cette preuve est différente de celle de (1) puisque nous nous interdisons d'utiliser l'α-conversion.

Propriété (4) Pour prouver cette propriété, nous utilisons la propriété précédente. Par hypothèse nous avons $P \sim_\alpha Q' \twoheadrightarrow P'$ et $P \equiv Q$. Par la propriété (3), il existe un Q'' tel que $Q \sim_\alpha Q''$ et $Q' \equiv Q''$ sans utiliser d'α-conversion. Nous prouvons maintenant que $Q'' \twoheadrightarrow P'$, par cas sur l'étape d'équivalence structurelle élémentaire $Q' \equiv Q''$. Nous remarquons que l'ordre dans lequel s'effectue le dépliage n'a aucune influence sur le terme final, à équivalence $\equiv_{AC0}$ près.

TREE Si la règle replie une location, alors il suffit de considérer le dépliage de cette location suivi par la série de dépliages $\twoheadrightarrow$. Si la règle déplie une location, il suffit de considérer la série de dépliages $\twoheadrightarrow$ dans laquelle ce premier dépliage n'a pas lieu.

SCOPE Ce cas est identique au précédent, sachant que tous les dépliages pouvant avoir lieu, il n'y a aucun conflit concernant les noms liés par le `def`.

Propriété (5) Nous procédons par cas sur la règle $\Leftarrow$ utilisée.

TREE Le résultat est immédiate, puisque le même renommage et le même repliage peuvent avoir lieu.

SCOPE Le résultat est immédiat, excepté pour les noms définis par la définition dépliée qui sont renommés par $\sim_\alpha$. Nous avons :

$$\begin{array}{rcl} \mathcal{S} \;\|\; \beta[D' : P' \mid \texttt{def}\ D\ \texttt{in}\ P] & \twoheadrightarrow & \mathcal{S} \;\|\; \beta[D', D : P' \mid P] \\ \mathcal{S} \;\|\; \beta[D', D : P' \mid P] & \sim_\alpha & \mathcal{S}_\alpha \;\|\; \beta_\alpha[D'_\alpha, D_\alpha : P'_\alpha \mid P_\alpha] \end{array}$$

Par hypothèse de la règle Scope, nous savons que les noms définis de D ne sont pas libres dans $\mathcal{S} \parallel \beta[D' : P']$. Le renommage $\sim_\alpha$ ne capturant pas les noms libres et étant injectif, les noms définis de D'_α ne sont pas libres dans $\mathcal{S}_\alpha \parallel \beta_\alpha[D'_\alpha : P'_\alpha]$. Nous avons donc :

$$\mathcal{S}_\alpha \parallel \beta_\alpha[D'_\alpha : P'_\alpha \mid \texttt{def } D_\alpha \texttt{ in } P_\alpha] \Rrightarrow \mathcal{S}_\alpha \parallel \beta_\alpha[D'_\alpha, D_\alpha : P'_\alpha \mid P_\alpha]$$

Nous remplaçons les étapes de renommage global des noms définis de D par des étapes d'α-conversion, et nous obtenons :

$$\mathcal{S} \parallel \beta[D' : P' \mid \texttt{def } D \texttt{ in } P] \sim_\alpha \mathcal{S}_\alpha \parallel \beta_\alpha[D'_\alpha : P'_\alpha \mid \texttt{def } D_\alpha \texttt{ in } P_\alpha]$$

Propriété (6) Nous procédons par analyse de cas sur la réduction ayant lieu.

Comm,Flush Le résultat est immédiat puisque le dépliage n'invalide ni l'étape Comm, ni l'étape Flush, et commute avec celles-ci. Nous avons donc $Q \mapsto Q'$ et $P' \Rrightarrow Q'$.

Join Le résultat n'est pas immédiat puisque l'étape Join peut créer un processus qui doit être déplié (et éventuellement α-renommé pour ce faire). Par contre, nous remarquons que le dépliage n'empêche pas l'étape Join d'avoir lieu, puisque le dépliage ne modifie ni la présence de la règle de réduction, ni la présence des messages dans la même location. Suite à l'étape $P \mapsto P'$, nous α-renommons le processus nouvellement créé pour que ses noms définis soient frais, et nous déplions tout P' pour obtenir Q'. A partir de Q, nous effectuons la même étape Join, nous α-renommons comme avant le processus généré, et nous déplions ce processus pour obtenir Q'.

Go Nous rappelons qu'une étape $\mapsto_{Go}$ peut replier la location migrant avant l'étape, ce qui rend possible cette étape après dépliage du processus initial. Comme cette étape génère elle aussi un nouveau processus en contexte d'évaluation, la preuve est identique au cas Join, a l'exception qu'il est aussi nécessaire de déplier les sous-locations de la location ayant migré.

Propriété (7) Cette propriété est une conséquence immédiate du premier cas de la propriété (6)

Propriété (8) Nous prouvons la partie commutation de cette propriété par cas sur la réduction $\mapsto$ ayant lieu.

Comm,Flush Si cette étape est la même que l'étape $\mapsto_d$, le résultat est identique. Sinon, les deux étapes sont indépendantes et commutent immédiatement. Dans les deux cas, nous appliquons au résultat de l'étape $\mapsto_d$ suivie de l'étape $\mapsto$ (si ces étapes sont distinctes) la même α-conversion et le même dépliage $\Rrightarrow$ que celui suivant l'étape $\mapsto$.

Join Si l'étape $\mapsto_d$ est une étape Flush, la commutation est immédiate. Sinon, nous remarquons qu'une étape Comm ne peut pas faire disparaître un message participant à l'étape Join, puisque chaque canal est défini dans une unique location. Les deux étapes commutent donc. Nous appliquons ensuite la même α-conversion et le même dépliage $\Rrightarrow$ que dans la réduction initiale.

Go Nous remarquons que l'étape Go pouvant replier la location migrant, cela peut invalider l'étape $\mapsto_d$. C'est pourquoi la propriété explicitement mentionne le dépliage. Par contre, après α-conversion et dépliage, l'étape $\mapsto_d$ peut bien avoir lieu (l'α-conversion ne modifie que des noms liés, qui sont donc distincts des noms impliqués par l'étape $\mapsto_d$), pour donner Q'. Pour aller de P' à Q', nous effectuons l'étape Go, qui est toujours possible après une étape $\mapsto_d$, puis nous effectuons le même α-renommage et dépliage que dans la réduction initiale.

Nous prouvons maintenant que la réduction $\mapsto_d$ est fortement normalisante. Soit P un processus. Nous considérons l'ordre lexicographique sur la paire d'entiers composée du nombre de processus de la forme $Flush(\ldots)$ et du nombre de messages qui ne sont pas dans la location où le canal sur lequel le message est émis est défini. Nous écrivons cette paire (F, C) et montrons que toute étape $\mapsto_d$ la fait décroître, par cas selon l'étape.

Flush Cette réduction faisant disparaître un processus de la forme $Flush(\ldots)$, la paire décroît nécessairement, même si de nouveaux messages sont générés.

Comm Cette réduction ne crée pas de processus $Flush$ et transporte un message dans la location où son canal est défini. Nous avons comme paire finale $(F, C-1)$, qui a bien strictement décru.

Propriété (9) Cette propriété est immédiate si c'est la même étape dans les deux cas $P \mapsto_d P'$ et $P \mapsto_d Q$. Si les étapes sont différentes, on prouve immédiatement par cas sur chaque étape qu'elles commutent. Nous pouvons déduire de cette propriété que la réduction $\mapsto_d$ est confluente.

Propriété (10) Cette propriété consiste à vérifier qu'il n'existe aucun processus de la forme $Flush(\ldots)$ dans un état d'approximation, et que tous les messages sont dans la location où leur canal est défini (ce qui signifie qu'aucune règle Comm ne peut avoir lieu). En ce qui concerne la première vérification, c'est une conséquence immédiate des conditions d'approximation (1, 2, 7) qui décrivent les processus présents. La deuxième vérification (pas de règle Comm possible) est conséquence de la condition d'approximation (1), de la condition (2) associée à la définition de D^n, et de la condition (7) une fois encore associée à la définition de D^n. Nous remarquons que le nom p sur lequel un message est présent à la racine n'est pas défini dans la configuration, donc aucune règle Comm ne peut s'appliquer pour ce message.

Propriété (11) La propriété $=_d \mapsto = \mapsto =_d$ est immédiate, puisqu'aucun message ne peut être présent sur un nom lié par un `fresh` ou un `uid`, la position de ces lieurs est donc complètement indépendante des réductions. La propriété $=_\alpha =_d = =_d =_\alpha$ est elle aussi immédiate, puisque l'α-conversion implique des noms liés alors que la relation $=_d$ ne modifie pas la présence des définitions globales qu'elle déplace. Nous prouvons maintenant la propriété $\Lleftarrow =_d \subseteq =_d \Lleftarrow$. Cette propriété est aussi immédiate puisque si un déplacement de définitions `fresh` et `uid` est possible avant un dépliage, il est toujours possible après. Nous remarquons que l'inclusion réciproque n'est pas vraie puisque le déplacement considéré pourrait exiger le dépliage initial d'un `def` contenant la définition à déplacer.

Propriété (12) La propriété $=_d \twoheadrightarrow_d \subseteq \twoheadrightarrow_d$ est une conséquence immédiate de la propriété (11). En effet nous avons (par définition de $\twoheadrightarrow_d$) :

$$\begin{array}{rl} & (=_d =_\alpha \Rrightarrow \mapsto_d^* \Rightarrow_d) = (=_\alpha =_d \Rrightarrow \mapsto_d^* \Rightarrow_d) \\ \subseteq & (=_\alpha \Rrightarrow =_d \mapsto_d^* \Rightarrow_d) = (=_\alpha \Rrightarrow \mapsto_d^* =_d \Rightarrow_d) = (=_\alpha \Rrightarrow \mapsto_d^* \Rightarrow_d) \end{array}$$

La propriété $\Rrightarrow \twoheadrightarrow_d \subseteq \twoheadrightarrow_d$ est aussi une conséquence immédiate de la définition de $\twoheadrightarrow_d$, ainsi que de la propriété $\Rrightarrow =_\alpha \subseteq =_\alpha \Rrightarrow$ (si une étape d'α-conversion peut être accomplie après une étape de dépliage, elle peut aussi être accomplie avant, puisqu'elle ne concerne pas la définition étant éventuellement dépliée).

La propriété $\mapsto_d \twoheadrightarrow_d \subseteq \twoheadrightarrow_d$ est une conséquence immédiate du fait que l'α-conversion et les étapes $\mapsto_d$ commutent, de l'inclusion $\mapsto_d \Rrightarrow \subseteq \Rrightarrow \mapsto_d$ (si une étape $\mapsto_d$ est possible avant dépliage, la même étape est possible après dépliage), et de la définition de $\twoheadrightarrow_d$.

□

Lemme A.2.3 *Pour tout $P \in \mathcal{A}$ tel que ses noms libres soient distincts des noms introduits par les traductions, nous avons :*

1. *$fn(\llbracket P \rrbracket_e) = fn(e) \cup fn(P)$;*
2. *pour tout nom d'ambient a et b, $\llbracket P \rrbracket_e \{^b/_a\} = \llbracket P\{^b/_a\} \rrbracket_e$;*
3. *soit $P' \in \mathcal{A}$ tel que $P =_\alpha P'$, alors $\llbracket P \rrbracket_e =_\alpha \llbracket P' \rrbracket_e$.*

Preuve: La propriété (1) est immédiate par induction sur la structure de P, puisque tout nom introduit par la traduction est introduit lié.

Nous prouvons la propriété (2) par induction sur la structure de P.

Tous les cas sont immédiats en se basant sur le fait que les noms libres apparaissant dans le processus ambient sont distincts des noms introduits par la traduction (en particulier les noms du contrôleur D, donc de l'interface e_h, ou les noms de l'interface e). Par exemple nous avons :

$$\begin{array}{rcl} \llbracket a[P]\{^b/_a\} \rrbracket_e & = & \llbracket b[P\{^b/_a\}] \rrbracket_e \\ & = & \texttt{def } here[D : \llbracket P\{^b/_a\} \rrbracket_{e_h} \mid \texttt{def uid } i \texttt{ in } s(b,i,e,\emptyset) \mid e.amb(i,b,e_h)] \texttt{ in } \mathbf{0} \\ & = & \texttt{def } here[D : \llbracket P \rrbracket_{e_h}\{^b/_a\} \mid \texttt{def uid } i \texttt{ in } s(b,i,e,\emptyset) \mid e.amb(i,b,e_h)] \texttt{ in } \mathbf{0} \\ & = & (\texttt{def } here[D : \llbracket P \rrbracket_{e^n} \mid \texttt{def uid } i \texttt{ in } s(a,i,e,\emptyset) \mid e.amb(i,a,e_h)] \texttt{ in } \mathbf{0})\{^b/_a\} \\ & = & \llbracket a[P] \rrbracket_e \{^b/_a\} \end{array}$$

La propriété (3) est prouvée par cas sur l'α-conversion ayant lieu.

Restriction Nous avons $\nu a.P =_\alpha \nu b.P\{^b/_a\}$. Donc nous avons $\llbracket \nu a.P \rrbracket_e = \texttt{def fresh } a \texttt{ in } \llbracket P \rrbracket_e =_\alpha \texttt{def fresh } b \texttt{ in } \llbracket P \rrbracket_e \{^b/_a\} = \texttt{def fresh } b \texttt{ in } \llbracket P\{^b/_a\} \rrbracket_e = \llbracket \nu b.P\{^b/_a\} \rrbracket_e$. Le passage de la substitution à l'intérieur de la traduction est une conséquence de la propriété (2).

Communication Nous avons $(x).P =_\alpha (y).P\{^y/_x\}$. Nous avons par conséquent $[\![(x).P]\!]_e = \texttt{def}\ \kappa(x) \triangleright [\![P]\!]_e\ \texttt{in}\ e.recv(\kappa) =_\alpha \texttt{def}\ \kappa(y) \triangleright [\![P\{^y/_x\}]\!]_e\ \texttt{in}\ e.recv(\kappa) = [\![(y).P\{^y/_x\}]\!]_e$. Le passage de la substitution sous la traduction est justifiée comme dans le cas précédent.

□

Nous partitionnons les étapes $\mapsto$ qui sont possibles à partir d'un état d'approximation. Cette partition se base sur une analyse par cas de la règle de réduction utilisée. Pour certaines de ces étapes, nous donnons une correspondance opérationnelle informelle avec des étapes des ambients étendus. Cette correspondance est formellement prouvée par le lemme A.2.15.

Comm, Flush Ces étapes ne peuvent avoir lieu, comme l'indique le lemme A.2.2(10).

Go Nous appelons ces étapes des *étapes Go*, que nous notons $\mapsto_{Go}$. Nous rappelons que ces étapes peuvent replier la location migrant afin d'accomplir la migration.

Join Selon la définition A.2.1, cette étape utilise soit une règle de réduction de D^n (plus précisément une règle de $D^n_0, D^n_1, D^n_2, D^n_C$, ou D^n_F) pour un $n \in N$, soit une règle de E^n pour un $n \in N$, soit une règle de D^0_t. Nous rappelons que les règles D^n sont définies dans la figure 3.3.

- D^n_0 Nous appelons ces étapes *étapes 0*, notées $\mapsto_0$. Ces étapes correspondent aux étapes préliminaires de délégation de l'algorithme de la section 3.1.1. Elles ne sont pas présentes dans le calcul étendu des ambients.
- D^n_1 Nous appelons ces étapes *étapes 1*, et les partitionnons en $\mapsto_{\text{In } 1}$, $\mapsto_{\text{Out } 1}$, et $\mapsto_{\text{Open } 1}$ selon la règle utilisée (la première consomme un message sur sub^n_{in}, la deuxième un message sur sub^n_{out} et la troisième un message sur $open^n$). Ces étapes correspondent respectivement aux étapes In 1, Out 1 et Open 1 du calcul étendu des ambients.
- D^n_2 Nous appelons ces étapes *étapes 2*, et les partitionnons en $\mapsto_{\text{Move } 2}$ et $\mapsto_{\text{Open } 2}$ selon la règle utilisée (la première consomme un message sur r^n et la deuxième un message sur o^n). Ces étapes correspondent respectivement aux étapes Move 2 et Open 2 du calcul des ambients.
- D^n_C Nous appelons ces étapes *étapes de communication*, notées $\mapsto_{\text{Recv}}$. Elles correspondent aux étapes Recv.
- D^n_F Nous appelons ces étapes *étapes de propagation*, notées $\mapsto_{Fwd}$. Ces étapes propagent les messages présents dans une location ouverte et ne correspondent pas à des étapes du calcul des ambients étendus.
- E^n Soit κ le nom défini par la règle utilisée. Si nous avons $\kappa \in K^n_{ab}$, cette étape est une *étape de continuation*, notée $\mapsto_\kappa$. Cette étape ne correspond pas à une étape ambient. Si nous avons $\kappa \in K^n_c$, cette étape est une *étape de réplication*, notée $\mapsto_{\text{Repl}}$, qui correspond à une étape Repl.
- D^0_t Par définition A.2.1, aucun message n'est présent sur le nom t, cette règle ne peut donc être utilisée.

Utilisant cette partition, nous classons les réductions $\twoheadrightarrow$ en deux familles de relations : la famille X comporte toutes les étapes en correspondance avec une étape ambient, la famille B comporte toutes les autres étapes, que nous appelons *étapes de bookkeeping*.

$$X \stackrel{\text{def}}{=} \{\twoheadrightarrow_{\text{In } 1}, \twoheadrightarrow_{\text{Out } 1}, \twoheadrightarrow_{\text{Open } 1}, \twoheadrightarrow_{\text{Move } 2}, \twoheadrightarrow_{\text{Open } 2}, \twoheadrightarrow_{\text{Recv}}, \twoheadrightarrow_{\text{Repl}}\}$$
$$B \stackrel{\text{def}}{=} \{\twoheadrightarrow_0, \twoheadrightarrow_{Fwd}, \twoheadrightarrow_\kappa, \twoheadrightarrow_{Go}\}$$

Nous montrons maintenant que la traduction étendue d'un processus ambient étendu se réduit en un état d'approximation (respectivement, nous montrons que la traduction d'un processus ambient classique se réduit aussi en un état d'approximation). De plus, nous montrons que les états d'approximation sont stables par $\twoheadrightarrow$.

Pour ce faire, nous avons besoin d'une notion d'*extension* d'un état d'approximation.

Définition A.2.4 (Extension d'état) *Soient $P, Q \in \mathcal{T}$ avec des ensembles d'indices respectifs N_P et N_Q. Soit m un indice de N_P. Nous disons que Q* étend *P en m par $F; N; I; L; C; E; M$ quand toutes les conditions suivantes sont satisfaites :*

1. *$F_P \uplus F = F_Q$; $N_P \uplus N = N_Q$; $L_P \uplus L = L_Q$; et $I_P \uplus I = I_Q$;*
2. *pour tout $n \in N^+_P$, nous avons $S^n_P = S^n_Q$ et $\alpha^n_P = \alpha^n_Q$;*
3. *pour tout $n \in N_P \setminus \{m\}$, nous avons $M^n_P = M^n_Q$ et $E^n_P = E^n_Q$;*

4. *nous avons* $M_P^m \mid M \equiv M_Q^m$ *;* $K_P^m \subseteq K_Q^m$ *avec* $E_P^m, E \equiv E_Q^m$ *;*
5. $q \in N$ *si et seulement si une location* $\alpha^q h^q$ *est présente dans* C*; pour tout* $q \in N$*, la chaîne* $\alpha^m h^m$ *est un préfixe de* $\alpha^q h^q$ *;*
6. $(dn(C) \cup dn(E)) \cap fn(P) = \emptyset$.

Lemme A.2.5 (Traduction d'ambient) *Soient* $Q \in \mathcal{E}$*,* $P \in \mathcal{T}$*, et* m *un indice de* N_P*, tels que les noms libres de* Q *soient distincts des noms introduits par la traduction, et tels que les noms liés de* Q *soient deux à deux distincts et distincts des noms de* P*. Il existe alors un ensemble de noms* F *et un état d'approximation* P' *tels que* $e^m, \alpha^m h^m, Q \rightarrow_{tr} (N; I; L; C; E; M)$*, et* P *peut être étendu en* m *par* $F; N; I; L; C; E; M$ *pour donner le processus* P'*. De plus, si* Q *est un ambient classique, en notant* $C(\,\cdot\,)$ *le contexte obtenu en remplaçant* M^m *par* $M^m \mid (\,\cdot\,)$ *dans* P*, nous avons* $C([\![Q]\!]_{e^m}) \twoheadrightarrow_d P'$.

Preuve: Nous prouvons ce lemme par induction sur la structure de Q, ce quel que soit $P \in \mathcal{T}$. Nous rappelons que par définition des traductions, les noms introduits par les traductions sont distincts des noms libres de Q.

$Q = \mathbf{0}$**.** Ce cas est immédiat puisque nous avons $e^m, \alpha^m h^m, \mathbf{0} \rightarrow_{tr} (\emptyset; \emptyset; \emptyset; \top; \top; \mathbf{0})$ et $C(\mathbf{0}) = P$ à équivalence $\equiv_{AC0}$ près.

$Q = \nu i.Q'$**.** Ce cas est immédiat par induction puisque $e^m, \alpha^m h^m, \nu i.Q' \rightarrow_{tr} S$ si $e^m, \alpha^m h^m, Q' \rightarrow_{tr} S$.

$Q = \nu a.Q'$**.** Le nom a étant distinct des noms de L_P et des noms introduits par la traduction, la configuration P_0 à laquelle on a ajouté a à L_P est un état d'approximation. Par hypothèse d'induction, nous avons $e^m, \alpha^m h^m, Q' \rightarrow_{tr} S$, et il existe un ensemble F et un état d'approximation P' tels que P_0 puisse être étendu en m par $F; S$ en P'.

Puisque a est distinct des noms liés dans Q' (donc des noms restreints en contexte d'évaluation) et des noms de la traduction, nous avons $e^m, \alpha^m h^m, \nu a.Q' \rightarrow_{tr} S \oplus (\emptyset; \emptyset; \{a\}; \top; \top; \mathbf{0})$.

Par conséquent, P est étendu en m par $F; (S \oplus (\emptyset; \emptyset; \{a\}; \top; \top; \mathbf{0}))$ en l'état d'approximation P'.

Dans le cas où Q est un processus classique, nous avons, puisque $C(\,\cdot\,)$ est un contexte d'évaluation :

$$C([\![\nu a.Q']\!]_{e^m}) \stackrel{\text{def}}{=} C(\texttt{def fresh } a \texttt{ in } [\![Q']\!]_{e^m}) \Rrightarrow C'_m([\![Q']\!]_{e^m}) =_d C'_0([\![Q']\!]_{e^m})$$

avec $C'_m(\,\cdot\,)$ étant $C(\,\cdot\,)$ en ajoutant la définition $\texttt{fresh } a$ à h^m, et $C'_0(\,\cdot\,)$ étant $C(\,\cdot\,)$ en ajoutant la définition $\texttt{fresh } a$ à h^0.

Puisque le processus $P'_0 = C'_0(\mathbf{0})$ est un état d'approximation (en prenant $L \uplus \{a\}$ à la place de L), nous utilisons l'hypothèse d'induction pour obtenir :

$$C'_0([\![Q'']\!]_{e^m}) \twoheadrightarrow_d P''$$

Nous avons par le lemme A.2.2(12) :

$$(\Rrightarrow =_d \twoheadrightarrow_d) \subseteq \twoheadrightarrow_d$$

Nous avons donc $C([\![\nu a.Q']\!]_{e^m}) \twoheadrightarrow_d P''$.

$Q = Q' \mid Q''$**.** et Nous commençons par utiliser l'hypothèse d'induction avec P et Q'. Nous avons donc $e^m, \alpha^m h^m, Q' \rightarrow_{tr} S_1$, et il existe un F_1 tel que P puisse être étendu par $F_1; S_1$ en m en un état d'approximation P_1. Si Q' est un processus classique, nous avons $C([\![Q']\!]_{e^m}) \twoheadrightarrow_d P_1$.

Nous appliquons de nouveau l'hypothèse d'induction avec P_1 et Q''. Nous avons par conséquent $e^m, \alpha^m h^m, Q'' \rightarrow_{tr} S_2$ et nous obtenons un F_2 tel que P_1 puisse être étendu par $F_2; S_2$ en m en un état d'approximation P_2.

Nous remarquons que par définition de l'extension d'état, les noms de S_1 apparaissent dans P_1 et les noms définis de S_2 ne peuvent apparaître définis dans P_1. La traduction $S_1 \oplus S_2$ est donc bien définie et nous avons :

$$e^m, \alpha^m h^m, Q' \mid Q'' \rightarrow_{tr} S_1 \oplus S_2$$

La première partie du lemme est donc vérifiée en prenant $F = F_1 \cup F_2$ et en remarquant que l'extension d'état se fait en m par $F; S_1 \oplus S_2$ (l'extension d'état est transitive sur le même indice).

Nous considérons maintenant le cas où Q'' est aussi un ambient classique. Soit $C_1(\,\cdot\,)$ le contexte généré par P_1 avec le même m. En appliquant la même série de réductions et d'équivalence au processus $C([\![Q]\!]_{e^m})$, nous obtenons :

$$C([\![Q]\!]_{e^m}) \stackrel{\text{def}}{=} C([\![Q']\!]_{e^m} \mid [\![Q'']\!]_{e^m}) =_\alpha \Rrightarrow \mapsto_d^* =_d C_1([\![Q'']\!]_{e^m})$$

Nous remarquons que l'étape d'α-conversion ne peut modifier $[\![Q'']\!]_{e^m}$ puisque $C(\,\cdot\,)$ est un contexte d'évaluation (qui ne lie donc aucun nom dans $[\![Q'']\!]_{e^m}$).

Puisque P_1 est un état d'approximation, nous pouvons appliquer l'hypothèse d'induction avec P_1 et Q'' en m. Nous avons donc :

$$C_1([\![Q'']\!]_{e^m}) \twoheadrightarrow_d P_2$$

Nous appliquons le lemme A.2.2(12) pour obtenir :

$$C([\![Q]\!]_{e^m}) \twoheadrightarrow_d P''$$

$Q = !Q'$**.** L'état d'approximation P peut immédiatement être étendu par $\emptyset;\emptyset;\emptyset;\emptyset;\top;\kappa() \triangleright [\![Q']\!]_e \mid \kappa();\kappa()$ en P', en choisissant un nom κ frais. Nous vérifions que les conditions d'approximation pour P' sont bien satisfaites. Le processus P' satisfait la condition d'approximation 1 avec le nouvel E^m (le reste du processus n'est pas modifié). Ce processus satisfait aussi la condition 2 avec l'ajout du message $\kappa()$. La condition 6 est aussi satisfaite. Les autres conditions ne sont pas modifiées.

Le processus Q étant nécessairement classique, nous construisons la dérivation ($C(\,\cdot\,)$ étant un contexte d'évaluation) :

$$C([\![!Q']\!]_{e^m}) \stackrel{\text{def}}{=} C(\texttt{def}\ \kappa() \triangleright [\![Q']\!]_{e^m} \mid \kappa()\ \texttt{in}\ \kappa()) \Rrightarrow P'$$

où P' est P auquel on a ajouté la définition $\kappa() \triangleright [\![Q']\!]_{e^m} \mid \kappa()$ à E^m (avec $\kappa \in K_c^m$ et $Q^\kappa = Q'$) et le message $\kappa()$ à M^m.

$Q = (x).Q'$**,** $\texttt{open}\ a.Q'$**,** $\texttt{out}\ a.Q'$**, or** $\texttt{in}\ a.Q'$**.** Ce cas est identique au cas précédent. Comme dans le cas précédent, la condition d'approximation 1 est satisfaite puisque seul E^m est étendu avec la définition de la continuation, la condition 2 est aussi satisfaite avec l'ajout d'un message $recv^m$, ou $open^m$, ou out^m, ou in^m. La condition 5 est satisfaite dans le cas $(x).Q'$ avec κ présent uniquement dans E^m et dans le message $recv(\kappa)$. La condition 4 est satisfaite dans les autres cas puisque le nom κ est présent uniquement dans sa définition dans E^m et dans un message $open^m(b,\kappa)$ (sous-cas (4e)), ou $out^m(b,\kappa)$ ou $in^m(b,\kappa)$ (sous-cas (4a)). Dans ces trois cas, si a n'est ni dans L_P ni dans F_P, nous prenons $F = \{a\}$. Les autres conditions d'approximation sont inchangées.

$Q = \langle a \rangle$**.** Nous prenons $F = \{a\}$ si a n'est ni dans L_P, ni dans F_P. Le processus P' étant le processus P étendu avec le message $send^m(a)$ en M^m, la condition d'approximation (2) est immédiatement satisfaite.

Par définition de la traduction, nous avons $C([\![\langle a \rangle]\!]_{e^m}) \stackrel{\text{def}}{=} C(send^m(a)) = P'$.

$Q = a[\,Q'\,]$**.** Soit n un nom frais.

Nous notons α^n pour $\alpha^m h^m$ et S^n pour $s^n(a, i^n, e^m, \emptyset)$.

Nous considérons le processus P_0 étant égal à $P_0' \parallel \alpha^n h^n[D^n : S^n]$, avec P_0' étant P avec une définition $\texttt{uid}\ i^n$ supplémentaire en h^0 et un message $amb^m(i^n, a, e^n)$ supplémentaire en h^m.

Nous prouvons maintenant que P_0 est un état d'approximation, en prenant $F_{P_0} = F_P \cup \{a\}$ si $a \notin L_P$, ou $F_{P_0} = F_P$ sinon.

La condition (1) est satisfaite avec $E^n = \top$. La condition (2) est immédiatement satisfaite. La condition (7) est satisfaite par le sous-cas (7(a)i), puisque $a \in F_{P_0} \cup L$, $\alpha^n = \alpha^m h^m$, M^m comporte bien un message $amb^m(i^n, a, e^n)$, et aucun message n'est présent sur les noms r^n, ni o^n, ni sur un amb^p avec l'identificateur i^n, puisque cet identificateur est nouveau.

Nous pouvons donc appliquer l'hypothèse d'induction avec P_0 et Q' en n. Nous avons : $e^n, \alpha^m h^m h^n, P \rightarrow_{tr} (N; I; L; C; E; M)$, et il existe un F tel que la configuration P_0 peut être étendue en n par $F; N; I; L; C; E; M$ en un état d'approximation P'. En particulier, les indices de N sont en bijection avec les locations de C, et le nom de chacune de ces locations est strictement préfixé par $\alpha^n h^n$.

Nous construisons la traduction (puisque les noms $dn(D^n)$ et i^n sont frais) :

$$e^m, \alpha^m h^m,\ a[\,P\,] \rightarrow_{tr} (N \uplus \{n\}; I \uplus \{i^n\}; L; C \ \parallel\ H^n_a(E)(M); \top; e^m.amb(i^n, a, e^n))$$

avec $H^n_a(\,\cdot\,)(\,\cdot\,) = \alpha^m h^m h^n[D^n, (\,\cdot\,) : s^n(a, i^n, e, \emptyset) \mid (\,\cdot\,)]$

Par conséquent, P peut être étendu en m par $F'; N \uplus \{n\}; I \uplus \{i^n\}; L; C \ \parallel\ \alpha^n h^n[D^n, E : S^n \mid M]; \top; e^m.amb(i^n, a, e^n)$ en le même P', avec $F' = F \cup \{a\}$ si $a \notin F_P \cup L_P$. En effet, toutes les conditions pour être une extension d'état sont satisfaites, en particulier le fait qu'à l'indice n corresponde une location, et que chaque location soit strictement préfixée par $\alpha^m h^m = \alpha^n$. De plus, les noms définis dans l'extension sont bien distincts des noms libres de P, par hypothèse d'induction et parce que n est frais.

Nous considérons le cas où Q' est un ambient classique. Nous avons :

$$\begin{aligned}
& C(\llbracket\, a[\,Q'\,] \,\rrbracket_{e^m}) \\
\stackrel{\text{def}}{=}\ & C(\texttt{def}\ here[D : (\texttt{def uid}\ i\ \texttt{in}\ s(a, i, e, \emptyset) \mid e^m.amb(i, a, e_h)) \mid \llbracket Q' \rrbracket_{e_h}]\ \texttt{in}\ \mathbf{0}) \\
=_\alpha\ & C(\texttt{def}\ h^n[D^n : (\texttt{def uid}\ i^n\ \texttt{in}\ s^n(a, i^n, e^n, \emptyset) \mid e^m.amb(i^n, a, e^n)) \mid \llbracket Q' \rrbracket_{e^n}]\ \texttt{in}\ \mathbf{0}) && \text{(A.1)} \\
\Rrightarrow\ & C(\mathbf{0}) \ \parallel\ \alpha^n h^n[D^n, \texttt{uid}\ i^n : S^n \mid \llbracket Q' \rrbracket_{e^n} \mid amb^m(i^n, a, e^n)] && \text{(A.2)} \\
=_d\ & C'(\mathbf{0}) \ \parallel\ \alpha^n h^n[D^n : S^n \mid \llbracket Q' \rrbracket_{e^n} \mid amb^m(i^n, a, e^n)] && \text{(A.3)} \\
\mapsto_d\ & C'(amb^m(i^n, a, e^n)) \ \parallel\ \alpha^n h^n[D^n : S^n \mid \llbracket Q' \rrbracket^\sharp_{e^n}] && \text{(A.4)} \\
\stackrel{\text{def}}{=}\ & C''(\llbracket Q' \rrbracket^\sharp_{e^n}) && \text{(A.5)}
\end{aligned}$$

avec $C'(\mathbf{0})$ étant $C(\mathbf{0})$ où le nom frais i^n a été ajouté à I. L'étape (A.1) convertit les noms du contrôleur pour qu'ils correspondent aux noms décorés par l'indice frais n ; l'étape (A.2) déplie le premier lieur `def`, la sous-location h^n, et le deuxième lieur `def` ; l'étape (A.3) déplace la définition `uid` i^n de h^n à h^0 ; l'étape (A.4) est une étape Comm déplaçant le message $amb^m(i^n, a, e^n)$ jusqu'à M^m ; l'équation (A.5) définit le contexte $C''(\,\cdot\,)$. Nous avons $C''(\mathbf{0}) = P_0$.

Par hypothèse d'induction, avec P_0 et Q' en n, nous avons $C''(\llbracket Q' \rrbracket_{e^n}) \twoheadrightarrow_d P'$. Nous concluons une fois de plus en utilisant le lemme A.2.2(12) pour accoler les deux séries d'étapes.

$Q = \overline{\imath}\{Q'\}\text{–}a[Q'']$**, (ou** $Q = \mathtt{o}\{Q'\}\text{–}a[Q'']$**).** Ces deux cas sont très similaires au cas précédent, avec la différence suivante : il n'y a pas de message sur amb^m. Par contre, nous devons considérer une continuation κ fraîche, et nous avons $K^n_a = \{\kappa\}$, $M^n = r^n(e^p, \kappa)$ pour un $p \in N$ (ou $M^n = o^n(\kappa)$ dans le cas d'une souche ouverte). Le nom introduit dans le log est ajouté à F s'il ne fait pas partie de $F_P \cup L_P$ dans le cas d'une souche migrante. Comme dans le cas précédent nous vérifions que le processus obtenu vérifie les conditions d'approximation. C'est le cas avec pour la condition (4) avec le sous-cas (4c) dans le cas d'une souche migrante et le sous-cas (4f) dans le cas d'une souche ouverte. C'est aussi le cas pour la condition (7) la sous-condition (7(a)ii) ou (7(a)iii), sachant qu'aucun autre message sur un amb^r transportant i^n ne peut être présent.

Nous concluons comme dans le cas précédent, en utilisant l'hypothèse d'induction en n, en construisant la traduction étendant P en m.

La deuxième partie du lemme ne peut pas s'appliquer à ce cas, puisqu'il s'agit nécessairement d'un ambient étendu.

□

Corollaire A.2.6 *Pour tout $Q \in \mathcal{E}$, il existe un $Q' \in \mathcal{E}$ et un $P \in \mathcal{T}$ tels que $Q =_\alpha Q'$, $\llbracket Q' \rrbracket^{t\sharp} = P$ et, si $Q \in \mathcal{A}$, $\llbracket Q \rrbracket^t \twoheadrightarrow_d P$.*

Preuve: C'est le cas puisque $\llbracket \mathbf{0} \rrbracket^{t\sharp} = \llbracket \mathbf{0} \rrbracket^t$ est un état d'approximation, en prenant Q' comme étant Q renommé pour que ses noms liés soient deux à deux distincts, distincts des noms libres de Q et distincts des noms de la traduction. Il suffit donc d'appliquer le lemme A.2.5 pour étendre $\llbracket \mathbf{0} \rrbracket^{t\sharp}$ avec la traduction de Q à l'indice 0. Dans le cas où $Q \in \mathcal{A}$, nous avons donc aussi $\llbracket Q' \rrbracket^t \twoheadrightarrow_d P$, donc $\llbracket Q \rrbracket^t =_\alpha \twoheadrightarrow_d P$ par le lemme A.2.3(3), donc $\llbracket Q \rrbracket^t \twoheadrightarrow_d P$ par définition de $\twoheadrightarrow_d$. □

Par la suite, nous ne considérons la relation $\llbracket\ \cdot\ \rrbracket^{t\sharp}$ que pour les couples dont la configuration join est un état d'approximation.

Nous montrons maintenant que l'ensemble $\mathcal{T}$ des états d'approximation est stable par réduction $\twoheadrightarrow$.

Lemme A.2.7 (Invariant d'approximation) *Pour tout $P \in \mathcal{T}$, si $P \mapsto P'$ alors il existe $Q \in \mathcal{T}$ tel que $P' \twoheadrightarrow_d Q$.*

Preuve: Soit $P \in \mathcal{T}$ tel que $P \mapsto P'$.

En nous basant sur la partition des étapes possible pour un état d'approximation, nous procédons par cas sur la réduction $P \mapsto P'$.

Etape Go Nous rappelons qu'une étape Go peut replier la location migrant avant la migration.

Si une telle étape est possible dans un état d'approximation pour une location h^n (avec $n \neq 0$ par condition d'approximation (1)), la condition d'approximation (7) doit nécessairement être le sous-cas (7(b)ii) pour S^n, qui est donc de la forme $\texttt{go}\ h^p; (I^n_{b,e^n,e^p} \mid \kappa() \mid F^n)$ avec $F^n \stackrel{\text{def}}{=} Flush(l^n, in^n, out^n, \kappa)$.

Nous avons donc :

$$
\begin{array}{rcl}
P & = & C \parallel C_{go}^{\alpha^n h^n} \parallel \alpha^p h^p[D_p : P_p] \parallel \alpha^n h^n[D_n : S^n \mid M^n] \\
& \mapsto_{Go} & C \parallel \alpha^p h^p[D_p, h^n[D_n, D_{go} : I^n_{b,e^n,e^p} \mid \kappa() \mid F^n \mid M^n] : P_p] \\
& =_\alpha & C \parallel \alpha^p h^p[D_p, h^n[D_n, D_{go} : I'^n_{b,e^n,e^p} \mid \kappa() \mid F^n \mid M^n] : P_p] \\
& \Rrightarrow & C \parallel C_{go}^{\alpha^p h^p h^n} \parallel \alpha^p h^p[D_p : P_p] \parallel \alpha^p h^p h^n[D_n : I'^n_{b,e^n,e^p} \mid \kappa() \mid F^n \mid M^n] \\
& =_\alpha \Rrightarrow & C \parallel C_{go}^{\alpha^p h^p h^n} \parallel \alpha^p h^p[D_p : P_p] \parallel \\
& & \alpha^p h^p h^n[D_n, \texttt{uid}\ i : s(b, i, e^p, \emptyset) \mid e^p.amb(i, b, e^n) \mid \kappa() \mid F^n \mid M^n] \\
& \mapsto_d & C \parallel C_{go}^{\alpha^p h^p h^n} \parallel \alpha^p h^p[D_p : P_p \mid amb^p(i, b, e^n)] \parallel \\
& & \alpha^p h^p h^n[D_n, \texttt{uid}\ i : s(b, i, e^p, \emptyset) \mid \kappa() \mid F^n \mid M^n] \\
& \mapsto_d & C \parallel C_{go}^{\alpha^p h^p h^n} \parallel \alpha^p h^p[D_p : P_p \mid amb^p(i, b, e^n)] \parallel \\
& & \alpha^p h^p h^n[D_n, \texttt{uid}\ i : s(b, i, e^p, \emptyset) \mid \kappa() \mid G^n \mid M^n] \\
& \mapsto_d^= =_d & Q
\end{array}
$$

avec :
- $C_{go}^{\alpha^n h^n}$ est la configuration des sous-locations de h^n dépliées ;
- D_{go} est la définition correspondant aux mêmes sous-locations repliées ;
- $C_{go}^{\alpha^p h^p h^n}$ est la configuration des sous-locations de h^n de nouveau dépliées ;
- G^n est le résultat d'une étape Flush appliqué à F^n.

L'α-conversion correspond au renommage de la définition de l'uid généré par I afin qu'il soit frais. Le premier dépliage correspond au dépliage de la location h^n et de toutes ses locations. Le deuxième dépliage correspond au dépliage de la définition de l'uid. La première étape $\mapsto_d$ est une étape Comm déplaçant le message sur amb^p. La deuxième étape $\mapsto_d$ est une étape Flush transformant F^n en G^n, la troisième étape $\mapsto_d$ est une étape Comm qui amène le message $\kappa()$ à la location où il est défini si ce n'est pas la location courante (c'est à dire la location h^q telle que $\kappa \in K^q_a$ si $q \neq n$). L'équivalence $=_d$ déplace la définition de l'uid pour la mettre dans la location racine h^0.

Nous remarquons que Q ne possède ni location ni définition repliée en contexte d'évaluation. Nous prouvons maintenant que Q est un état d'approximation. Les différences entre P et Q sont :

1. $\alpha^n_Q = \alpha^p h^p$; $\alpha^m_Q = \alpha^m$ si h^n n'est pas dans α^m ; $\alpha^m_Q = \alpha^p h^p h^n \beta$ si α^m est de la forme $\alpha^n h^n \beta$.
2. $S^n_Q = s^n(b, i, e^p, \emptyset)$; $I_Q = I \uplus \{i\}$;
3. $M^p_Q = M^p \mid amb^p(i, b, e^n)$;
   $M^n_Q = M^n \mid G^n$;
   $M^q_Q = M_Q \mid \kappa()$ (si $\kappa \in K^q_a$ et $q \neq n$).

Nous vérifions maintenant que ces changements n'invalident pas les conditions d'approximation.

1. est satisfaite avec $i^n_Q = i$.

2. est satisfaite pour tout $n \in N$, puisque les messages générés par l'étape Flush sont dans la bonne location, puisque le message sur amb^p est dans h^p, et puisque le message $\kappa()$ est dans q si $\kappa \in K^q_a$.

3. est satisfaite puisque l^n est vide.

4. Nous remarquons tout d'abord que les vieux messages pour i_P^n sont encore vieux pour $i_Q^n = i$ qui est frais. Pour P, le seul sous-cas satisfait était le cas (4d). Nous considérons maintenant chacun des sous-cas pour Q. Le sous-cas (4a) ne peut-être satisfait avec un message qui était déjà présent. Puisque l'étape FLUSH n'a pas réémis l'éventuelle occurrence unique de κ dans le log, les messages de G^n ne contiennent pas κ. Le sous-cas (4b) ne peut-être satisfait (pour les messages qui ne sont pas vieux), parce qu'aucun nouveau message de ce type n'a été émis, et aucun message de ce type ne contenait κ dans P. Les sous-cas (4c,4f,4e) ne peuvent être satisfaits pour la même raison. Le sous-cas (4d) ne peut-être satisfait puisque ce processus a justement disparu et n'a pas été réémis. Le dernier sous-cas (4g) est justement le seul cas qui est satisfait.

Nous considérons maintenant les continuations contenues dans les messages émis par l'étape FLUSH. Ces continuations étaient présente dans le journal l^n, ce qui n'est autorisé que par les sous-cas (4b, 4c, 4d). Le sous-cas (4c) est impossible puisqu'il n'y a pas de message sur r^n dans P. Le sous-cas (4d) ne s'applique pas puisqu'il correspond à la continuation κ qui n'a pas été émise par l'étape FLUSH. Nous considérons donc le sous-cas (4b), donc des messages de la forme $sub_{in}^r(i_P^n, c, \kappa)$ (ou des messages sur sub_{out}^r). Par la condition (7b) pour P, il n'existe aucun message sur un amb^s contenant l'uid i_P^n, donc ces messages étaient vieux avant la réduction, ils le sont donc toujours après. Enfin, nous remarquons que par la condition (3) pour P, les messages émis par l'étape FLUSH transportent bien tous des noms de $F \cup L$. Ces messages satisfont donc uniquement la condition (4a).

5. et ***6.*** ne sont pas modifiées par la réduction.

7. est satisfaite puisque $S^n = s^n(b, i', e^p, \emptyset)$, h^p est la location parente de h^n, et $b \in F \cup L$ par la condition (7(b)ii) pour P. La condition 7(b)ii pour P nous garantit l'absence de messages de la forme $amb^r(i^n, \_, e^n)$, r^n ou o^n dans P. Par contre, un message $amb^p(i, b, e^n)$ est bien présent dans Q, et la condition (7(a)i) est satisfaite en prenant le mot vide pour β.

Etape de continuation, étape de réplication Par définition, il existe un M^n avec $n \in N$ dans lequel le message sur κ qui est consommé apparaît, avec $\kappa \in K_{abc}^n$. Nous commençons par détailler le cas de continuation de migration $\kappa \in K_a^n$.

Nous considérons donc la configuration $C^0 \parallel \alpha^n h^n[D^n, E^n : M^n \mid \kappa()]$, avec la définition $\kappa() \triangleright [\![Q^\kappa]\!]_{e^n}$ présente dans E^n et M'^n étant bien entendu M^n auquel on a enlevé le message $\kappa()$. Par définition de la traduction, nous rappelons que les noms libres de Q sont distincts des noms introduits par la traduction.

Nous avons donc l'étape :

$$C^0 \parallel \alpha^n h^n[D^n, E^n : M'^n \mid \kappa()] \mapsto_\kappa C^0 \parallel \alpha^n h^n[D^n, E^n : M'^n \mid [\![Q^\kappa]\!]_{e^n}]$$

Nous montrons maintenant que le processus $P_0 = C^0 \parallel \alpha^n h^n[D^n, E^n : M'^n]$ est un état d'approximation. C'est effectivement le cas puisque seule la condition (4) a été modifiée, et supprimer le message sur κ ne l'invalide pas. Nous pouvons donc appliquer le lemme A.2.5 pour étendre P_0 en n avec la traduction $[\![Q^\kappa]\!]_{e^n}$, après avoir α-renommé Q^κ en Q'^κ de telle sorte que ses noms liés soient deux à deux distincts et distincts des noms de P_0 (nous utilisons le lemme A.2.3(3)). Il existe donc un état d'approximation P' tel que $C^0 \parallel \alpha^n h^n[D^n, E^n : M'^n \mid [\![Q^\kappa]\!]_{e^n}] =_\alpha \twoheadrightarrow_d P'$.

Nous étudions maintenant le cas où le message est de la forme $\kappa(a)$ avec $\kappa \in K_b^n$, $a \in F \cup L$, et $\kappa(b) \triangleright [\![Q^\kappa]\!]_{e^n}$ dans E^n. Nous procédons exactement comme précédemment, en utilisant le lemme A.2.3(2) :

$$\begin{aligned} C^0 \parallel \alpha^n h^n[D^n, E^n : M'^n \mid \kappa(a)] &\mapsto_\kappa C^0 \parallel \alpha^n h^n[D^n, E^n : M'^n \mid [\![Q^\kappa]\!]_{e^n}\{^a/_b\}] \\ &= C^0 \parallel \alpha^n h^n[D^n, E^n : M'^n \mid [\![Q^\kappa\{^a/_b\}]\!]_{e^n}] \end{aligned}$$

Puisque supprimer le message $\kappa(a)$ préserve le statut d'état d'approximation, nous concluons comme auparavant.

Nous étudions enfin le cas de la réplication : $\kappa \in K_c^n$. Ce cas est identique au cas de la continuation de migration, avec l'exception que le message $\kappa()$ est immédiatement réémis. La configuration obtenue après l'étape $\mapsto_{Repl}$ est donc la configuration initiale avec le processus $[\![Q^\kappa]\!]_{e^n}$ en plus dans M^n. La configuration sans ce nouveau processus est immédiatement un état d'approximation (c'est la configuration initiale), et nous concluons comme auparavant.

Etape de communication de la forme : $C(recv^n(\kappa) \mid send^n(a)) \mapsto_{Recv} C(\kappa(a))$.

Nous avons $a \in F \cup L$ par la condition d'approximation (2) pour P. Si le message sur κ n'est pas dans sa location de définition, il est déplacé par une étape Comm. Les conditions d'approximation sont immédiatement satisfaites pour la configuration résultant, puisque la présence d'un message $recv^n(\kappa)$ dans M^n garantit par la condition (5) pour P que κ n'est présent nul part ailleurs (excepté dans sa définition), donc cette même condition est satisfaite pour la configuration finale avec le message $\kappa(a)$.

Etape 0 de la forme : $C(s(a, i^n, e^m, l^n) \mid in^n(b, \kappa)) \mapsto C(s(a, i^n, e^m, l^n \cup \{\text{In } b\ \kappa\}) \mid sub^m_{in}(i^n, b, \kappa))$ (nous ne détaillons pas le cas similaire pour un message sur out^n).

La plupart des conditions d'approximation ne sont pas modifiées. Nous détaillons celles qui le sont.

La condition d'approximation (2) est satisfaite après une étape Comm amenant le message $sub^m_{in}(i^n, b, \kappa)$ dans M^m. La condition (3) est satisfaite parce que la condition (4a) est satisfaite pour P, donc nous avons $b \in F \cup L$. La condition (4) est satisfaite pour les raisons suivantes. Tous les vieux messages restent vieux. L'unique autre message mentionnant κ (le message $in^n(b, \kappa)$) a disparu (l'unicité de ce message est la conséquence de la condition (4) pour P). Le sous-cas (4b) est satisfait puisque κ n'était pas présent dans l^n dans P, et les autres sous-cas ne peuvent être satisfaits puisqu'ils ne l'étaient pas pour P.

Etape 1 Nous supposons que cette étape utilise une règle de D^n_1 en consommant un message de la forme $amb^n(i^p, b, e^p)$. Dans les deux cas possibles (création d'un message sur r^p ou sur o^p), après transport de ce message dans la location h^p par une étape Comm, les conditions (1,3, 5,6) sont immédiatement satisfaites pour Q. En ce qui concerne la condition (2), nous devons nous assurer que l'indice p n'est pas 0. C'est nécessairement le cas puisque par la condition (7(a)i) qui est satisfaite pour p dans P à cause de la présence du message $amb^n(i^p, b, e^p)$, la location h^p est une sous-location de h^n, donc elle ne peut pas être la location racine.

La condition d'approximation (4) est satisfaite puisque soit seul le sous-cas (4b) est satisfait pour P (le message $sub^n_{in}(i^p, \_, \_)$ ou $sub^n_{out}(i^p, \_, \_)$ ne peut être vieux puisqu'un message $amb^n(i^p, b, e^p)$ est présent, donc nécessairement par (7(a)i), un message $s^p(b, i^p, \_.\_)$ est présent), alors seul le sous-cas (4c) est satisfait pour Q, soit seul le sous-cas (4e) est satisfait pour P, alors seul le sous-cas (4f) est satisfait pour Q.

La condition d'approximation (7a) est toujours satisfaite pour h^n dans Q. En ce qui concerne la location h^p, la présence dans P d'un message $amb^n(i^p, b, e^p)$ implique par la condition (7(a)i) que S^p soit de la forme $s^p(b, i^p, \_, \_)$, et qu'il n'y ait aucun message sur r^p ou o^p dans P. Les trois étapes de D_1 ne consomment pas le message sur s^p, suppriment le message unique de la forme $amb^t(i^p, \_, \_)$ (c'est le message $amb^n(i^p, b, e^p)$) et créent soit un message $r^p(e^r, \kappa)$, soit un message $o^p(\kappa)$. Dans le premier cas, le sous-cas (7(a)ii) est satisfait, dans le second, le sous-cas (7(a)iii) est satisfait.

Etape 2 Nous supposons que la règle utilisée est une règle de D^n_2. Deux cas sont possibles, selon qu'un message r^n ou o^n est consommé. Dans tous les cas, les conditions d'approximation (6, 5, 3) sont immédiatement satisfaites. Nous remarquons aussi que nous avons nécessairement $n \neq 0$ par la condition (2) pour P.

Nous étudions tout d'abord le cas de l'ouverture d'une location. Soit P' la configuration obtenue après la réduction $s^n(a, i^n, e^m, l^n) \mid o^n(\kappa) \triangleright f^n(e^m) \mid \kappa() \mid Flush(l^n, e^m.in, e^m.out, \kappa)$, l'étape Flush et les étapes Comm transportant les messages sur in^m, out^m et éventuellement κ si ce dernier n'est pas dans K^n. La configuration P' satisfait immédiatement les conditions (1,2).

Nous considérons maintenant la condition (4) (ce cas est très similaire à celui de l'étape Go). Par cette condition pour P, une continuations κ' est présente dans l^n seulement si elle est dans un message de la forme $sub^p_{in}(i^n, b, \kappa')$ ou $sub^p_{out}(i^n, b, \kappa)$, ou dans un message sur r^n, ou dans un état transitoire S^n. Puisque la condition (7(a)iii) est satisfaite pour P, les deux derniers cas sont impossibles. De plus, le message unique $s^n(\_, i^n, \_, \_)$ étant consommé, les messages de la forme $sub^p_{in}(i^n, b, \kappa')$ ou $sub^p_{out}(i^n, b, \kappa)$ sont vieux dans P'. Par conséquent, les continuations qui sont présentes dans l^n n'apparaissent dans P' que dans de vieux messages et dans les messages sur in^m et out^m qui ont été générés par l'étape Flush. Puisque ces continuations étaient nécessairement présentes chacune une seule fois dans le log, la condition (4a) est satisfaite.

En ce qui concerne la condition (4) pour κ, elle est satisfaite puisque la seule occurrence de κ (dans le message $o^n(\kappa)$) disparaît.

Nous étudions maintenant la condition (7). En ce qui concerne h^n, S^n est $f^n(e^m)$. La condition (7(b)i) est satisfaite, puisque la condition (7(a)iii) pour P implique qu'aucun message

de la forme $amb^p(i^n, \_, \_)$ ou sur r^n ne soit présent, et le seul message sur o^n a été consommé. De plus, la condition (7a) pour P nous garantit que $\alpha^n = \alpha^m h^m$.

La condition (7) n'est pas modifiée pour les autres locations.

Nous étudions maintenant le cas du repositionnement (un message sur r^n est consommé). Soit P' la configuration obtenue après la réduction $s^n(a, i^n, e^t, l^n) \mid r^n(e^m, \kappa) \triangleright \texttt{go}\ e^m.here; R'$. Cette configuration P' satisfait immédiatement les conditions d'approximation (1,2,3,5,6).

La condition 4 est satisfaite pour κ en passant du sous-cas (4c) pour P au sous-cas (4d) pour P'. Les autres continuations ne sont pas modifiées par cette réduction.

La condition 7 est satisfaite pour n en passant du sous-cas (7(a)ii) au sous-cas (7(b)ii), l'unique message sur r^n et l'unique message sur s^n ayant été consommés par la réduction.

Etape de propagation Nous étudions maintenant le cas de réductions utilisant une règle de la forme $f^n(e^m) \mid C^n \triangleright f^n(e^m) \mid e^m.C$, avec C étant un message pouvant être propagé par les règles de D_F^n. La configuration P' est la configuration obtenue après la réduction et l'étape Comm transportant le message propagé dans h^m. Nous remarquons que dans tous les cas les conditions d'approximation (1,3,6) sont satisfaites pour P'. Les messages sur r^n et o^n n'étant jamais propagés (parce qu'ils ne sont pas propagés par les règles de D_F^n mais aussi parce que la condition (7(b)i) pour h^n dans P garantit qu'il n'y a pas de message sur ces noms), la condition (2) est satisfaite pour P', en particulier pour la location racine h^0.

La condition (7(b)i) pour h^n n'est pas modifiée et est toujours satisfaite pour P'.

Nous étudions maintenant les autres conditions en fonction du message qui est propagé.

Si $C = send(b)$, toutes les conditions sont alors satisfaites.

Si $C = in(b, \kappa)$, $C = out(b, \kappa)$, $C = open(b, \kappa)$ ou $C = recv(\kappa)$, alors un message transportant une continuation est remplacé par un autre message contenant la même continuation. Par conséquent, les conditions (4,5) sont satisfaites. La condition (7) est aussi immédiatement satisfaite pour toutes les locations.

Si $C = sub_{in}(i^p, b, \kappa)$ ou $C = sub_{out}(i^p, b, \kappa)$, alors par la condition (4b) pour P, la continuation κ est aussi présente une fois dans l^p. Cette condition est toujours satisfaite après propagation du message.

Si $C = amb(i^p, b, e^p)$, alors la condition (7(a)i) est satisfaite pour h^p dans P. Nous avons donc $\alpha^n h^n \beta = \alpha^p$ avec toutes les locations de β ouvertes. Après propagation, nous avons un message $amb^m(i^p, b, e^p)$. Par la condition (7(b)i) pour h^n, nous avons $\alpha^n = \alpha^m h^m$. Nous avons donc $\alpha^p = \alpha^m h^m h^n \beta$. Soit $\beta' = h^n \beta$, toutes les locations de β' sont ouvertes, donc la condition (7(a)i) est satisfaite pour h^p dans P'. La condition (7) est immédiatement satisfaite pour les autres locations. □

Le lemme suivant établit le fait que l'état d'approximation atteint après une étape $\mapsto\twoheadrightarrow_d$ ne dépend en fait pas du choix de l'étape $\twoheadrightarrow_d$, à équivalence $\sim_\alpha \equiv_{AC0}$ près.

Lemme A.2.8 *Soient $P \in \mathcal{J}$ et $Q, Q' \in \mathcal{T}$ tels que $P \twoheadrightarrow_d Q$ et $P \twoheadrightarrow_d Q'$. Nous avons $Q \sim_\alpha \equiv_{AC0} Q'$.*

Preuve: Par définition de $\twoheadrightarrow_d$, nous avons $P =_\alpha \Rrightarrow P_1 \mapsto_d^* P_2 \Rightarrow_d Q$ et $P =_\alpha \Rrightarrow P_1' \mapsto_d^* P'2 \Rightarrow_d Q'$. Par conséquent nous avons donc $P \sim_\alpha \Rrightarrow P_1$ et $P \equiv P_1'$. Nous appliquons le lemme A.2.2(4) pour obtenir $P_1' \sim_\alpha \Rrightarrow P_1$. Puisque P_1' est déjà complètement déplié, nous avons en fait $P_1' \sim_\alpha \equiv_{AC0} P_1$. Nous avons : $\sim_\alpha$ est une bisimulation forte pour les étapes $\mapsto$ (lemme A.2.2(1)), la relation $\equiv_{AC0}$ est aussi une bisimulation forte pour les étapes $\mapsto$, et la réduction $\mapsto_d$ est confluente (lemme A.2.2(9)). Puisque aucune étape $\mapsto_d$ n'est possible pour P_2 ni P_2', nous avons donc $P_2 \sim_\alpha \equiv_{AC0} P_2'$ (les relations $\sim_\alpha$ et $\equiv_{AC0}$ commutent). Les relations $=_d$ et $\sim_\alpha$, ainsi que $=_d$ et $\equiv_{AC0}$ commutant, nous avons donc $Q \sim_\alpha \equiv_{AC0} Q'$. □

Par la suite, nous assimilons les états d'approximation qui sont $\sim_\alpha \equiv_{AC0}$ équivalents.

Afin de mettre en relation les processus ambient et processus join, nous introduisons une classe de processus join plus fine que celle des états d'approximation :

Définition A.2.9 (Traduction bien formée) *Nous disons que $P \in \mathcal{J}$ est* bien formé *lorsqu'il existe $(Q, Q') \in \mathcal{A} \times \mathcal{E}$ tels que $Q \rightarrow_{12C}^* Q'$ et $[\![Q']\!]^{t\sharp} \twoheadrightarrow^* P$.*

Par les lemmes A.2.6, A.2.7, A.2.2(2) et A.2.8, tout processus bien formé P est aussi un état d'approximation.

Nous prouvons maintenant que les migrations présentes en contexte d'évaluation dans un processus bien formé peuvent toujours avoir lieu. Ce lemme est très semblable à la condition 3 du lemme A.1.2, et est nécessaire puisque dans un processus du join calcul, une migration est bloquée lorsque sa destination est une sous-location de la location migrant.

Lemme A.2.10 *Soit un processus $P \in \mathcal{T}$ bien formé, avec un ensemble d'indices N. Pour tout $n \in N$ transitoire (satisfaisant la condition d'approximation (7(b)ii)), en notant p l'indice tel que $S^n = \mathtt{go}\ h^p; R'$, nous avons la propriété que la location h^p n'est pas une sous-location de la location h^n.*

Preuve: Soit une configuration $P \in \mathcal{T}$, et N son ensemble d'indices. Nous ordonnons N par la relation d'inclusion entre locations : nous avons $n \leq p$ si et seulement si $h^p \in \alpha^n h^n$. Nous remarquons que la relation $\leq$ est une relation d'ordre partiel sur N. Nous introduisons également une autre relation sur N notée $\lessdot$. Nous avons $n \lessdot p$ si P satisfait la condition (7(a)ii) pour n avec le message $r^n(e^p, \kappa)$, ou si P satisfait la condition (7(b)ii) pour n avec S^n de la forme $\mathtt{go}\ h^p; R'$. De manière intuitive, nous avons $n \lessdot p$ si h^n va migrer en h^p. Nous disons que $P \in \mathcal{T}$ est *sans cycle* si la relation $\lessdot \leq$ ne possède pas de cycle.

Nous commençons par montrer que, pour tous $(R, Q) \in \mathcal{A} \times \mathcal{E}$ et $P \in \mathcal{T}$ tels que $R \to^* Q$ et $[\![Q]\!]^{t\sharp} = P$, P est sans cycle. Puisque par hypothèse nous avons $R \to^* Q$, nous pouvons appliquer le lemme A.1.2, et par conséquent Q satisfait la condition d'imbrication des souches et scions (condition 3). Nous remarquons que nous avons une correspondance univoque entre les locations introduites par la traduction et les ambients du processus traduit. Nous indexons donc les ambients du processus étendu Q avec les mêmes indices que P, et nous remarquons que la définition de la traduction implique que la relation $\leq$ correspond également à la relation d'imbrication d'ambients pour Q. De plus, par définition de la traduction étendue nous avons pour chaque pair $n \lessdot p$ deux contextes d'évaluation ambient $C(\,\cdot\,)$ et $C'_p(\,\cdot\,)$ tels que $Q = C(\,\bar{\imath}_n\{S_n\}\text{–}a_n[\,T_n\,]\,)$ et $Q = C'_p(i_n)$, avec le trou de $C'_p(\,\cdot\,)$ dans l'ambient d'indice p. (Nous ne prenons pas en compte ici les processus de la forme $\mathtt{go}\ h^p; R'$ puisque ceux-ci ne sont ni créés par la traduction, ni créés par la normalisation.) Par conséquent, pour chaque indice q nous avons $n \lessdot \leq q$ s'il existe deux contextes d'évaluation $E(\,\cdot\,)$ et $F(\,\cdot\,)$ tels que $Q = E(\,X_q a_q[\,F(i_n)\,]\,)$. Nous montrons maintenant que $n \lessdot \leq m \lessdot \leq p$ implique $i_n \prec i_m$ pour Q. En utilisant les propriétés ci-dessus deux fois pour les paires n, m et m, p, nous avons :

$$Q = E(\,X_m a_m[\,F(i_n)\,]\,) = C(\,\bar{\imath}_m\{S_m\}\text{–}a_m[\,T_m\,]\,)$$

Par conséquent, puisque cette égalité concerne l'unique ambient portant l'indice m, nous en déduisons $T_m = F_n(i_n)$, donc nous avons bien $i_n \prec i_m$ pour Q. Pour conclure, nous remarquons que de tout cycle de $\lessdot \leq$ nous pouvons extraire un cycle de $\prec$, or ceux-ci sont exclus par la condition (3) du lemme A.1.2, donc $\lessdot \leq$ est sans cycle.

Nous montrons maintenant que pour tout $P \in \mathcal{T}$ bien formé et sans cycle, si $P \twoheadrightarrow P'$, alors P' est sans cycle (et P' est toujours bien formé par définition). Cette preuve est très semblable à la preuve de la condition (3) du lemme A.1.2. Nous remarquons que le renommage initial ne modifie pas la propriété d'être sans cycle, et nous ne prenons pas en compte ce renommage pour simplifier la présentation. Nous écrivons $\lessdot'$ et $\leq'$ pour les relations $\lessdot$ et $\leq$ de P', et examinons les changements dans ces relations dus à la réduction. Cette preuve est par cas selon la réduction ayant lieu, comme la preuve du lemme A.2.7.

Etape Go Par la définition A.2.1, il existe un $n \lessdot p$ tel que h^n migre dans la location h^p. Nous avons donc $\lessdot' = \lessdot \setminus \{n, p\}$ et $\leq' \subseteq (\leq \cup \{n, p\})^*$. Par conséquent, nous avons $\lessdot' \leq' \subseteq \lessdot\ (\leq \cup \lessdot)^*$. Donc si $\lessdot \leq$ est sans cycle, alors $\lessdot\ (\leq \cup \lessdot)^*$ est aussi sans cycle, donc $\lessdot' \leq'$ est sans cycle, donc P' est sans cycle.

Etape de continuation, étape de réplication Ces étapes étendent P comme décrit dans le lemme A.2.5, avec $N' \supseteq N$. Puisque les continuations n'introduisent que des traductions d'ambients classiques, nous avons nécessairement $\lessdot' = \lessdot$ et $\leq' \cap (N \times N') = \leq$ (l'inclusion de nouveaux ambients ne se fait qu'à l'intérieur d'ambients existants). Nous avons donc $\lessdot' \leq' = \lessdot \leq$.

Etape 0, étape de communication Ces étapes ne modifient ni $\lessdot$, ni $\leq$.

Etape de propagation Les messages de la forme $r^n(e^p, \kappa)$ n'étant pas propagés, la relation $\lessdot \leq$ n'est pas modifiée.

Etape In 1 Supposons que la règle utilisée soit une règle de D_1^m. Nous notons $sub_{in}^m(i^n, a, \kappa)$, $amb^m(i^n, \_, e^n)$ et $amb^m(i^p, a, e^p)$ les messages entre autres consommés par cette étape. Le message de repositionnement produit par l'étape est donc $r^n(e^p, \kappa)$. Par conséquent nous avons $\lessdot' = \lessdot \cup \{n, p\}$, et $\leq' = \leq$ (l'arbre des locations n'est pas modifié).

Puisque P satisfait la condition (7(a)i) pour n et p, nous avons $n \leq m$ et $p \leq m$. De plus, comme P ne peut satisfaire les conditions (7(a)ii) ou (7(b)ii), pour P nous avons $p \not\lessdot$. La condition (7(a)i) pour p indique que les locations entre h^m et h^p sont toutes ouvertes (ce sont

les location h^s de β). Puisque ces locations h^s sont ouvertes, elles satisfont la condition (7(b)i), et nous avons donc aussi $s \not\lessdot$.

Supposons que $n \lessdot' p$ apparaisse dans un cycle. Il existe alors un cycle tel que $n \lessdot' p$ n'apparaisse qu'une seule fois, et ce cycle a la forme : $n \lessdot' p \leq q(\lessdot \leq)^* n$. Nous avons nécessairement $q \neq n$, puisque si $p \leq n$, alors nous avons $p \leq n \leq m$, alors n est soit p, soit un s tel que h^s est une location entre h^m et h^p, soit m. Si n est p alors nous avons deux messages de la forme $amb^m(i^p, \_, \_)$ dans P, ce qui est impossible par la condition (7(a)i) pour p. Si $n = m$, alors la condition (7(a)i) pour n n'est plus satisfaite, puisque nous ne pouvons avoir $\alpha^n h^n \beta = \alpha^n$. Si n est un s, alors n est à la fois vivant et ouvert, ce qui est impossible.

Nous avons donc un cycle de la forme : $n \lessdot' p \leq q \lessdot \leq (\lessdot \leq)^* n$, puisque $q \neq n$. Puisque nous avons vu que $p \not\lessdot$ et $s \not\lessdot$ pour tout h^s entre h^p et h^m, l'indice q est nécessairement différent de p et des s. Nous avons donc nécessairement $m \leq q$ (h^q est une location englobant h^p qui n'est ni h^p ni aucune location entre h^p et h^m, donc h^q englobe h^m). Nous construisons donc le cycle (puisque nous avons $n \leq m$) : $n \leq m \leq q \lessdot \leq (\lessdot \leq)^* n$, ce qui est impossible puisque $\lessdot \leq$ n'a pas de cycle. Par conséquent aucun cycle de $\lessdot' \leq'$ ne peut contenir une étape $n \lessdot' p$. Comme c'est la seule étape ajouté à $\lessdot$ et que $\leq$ n'a pas changé, $\lessdot' \leq'$, donc P', est sans cycle.

Etape OUT 1 Supposons que la règle utilisée soit une règle de D_1^m. Nous notons $amb^m(i^n, e^n)$ et $s^m(\_, \_, e^p, \_)$ les messages entre autres consommés par cette étape. Le message de repositionnement généré est le message $r^n(e^p, \_)$. Nous avons donc $\lessdot' = \lessdot \cup \{n, p\}$ et $\leq' = \leq$. Par la condition d'approximation (7a) pour m nous avons $m \leq p$. Par la condition d'approximation (7(a)i) pour n nous avons $n \leq m$. Donc nous avons $n \leq p$. Puisque n est forcément différent de p, nous pouvons associer à tout cycle de P' un cycle de P (en remplaçant l'étape $n \lessdot' p$ par l'étape $n \leq p$), donc P' est sans cycle.

Etape MOVE 2 Un processus $\texttt{go}\ h^p; \_$ est substitué à la place d'un processus $r^n(e^p, \_)$ dans h^n, ce qui ne modifie pas les relations $\lessdot$ ni $\leq$.

Étapes OPEN 1**,** OPEN 2 Ces étapes ne modifient pas les relations. □

A.2.2 Relations de simplification et commutations

Nous définissons maintenant des relations de simplification s'appliquant à des états d'approximation, et exprimons les propriétés de ces relations par des diagrammes de commutation. Une première relation supprime les vieux messages des états d'approximation :

Lemme A.2.11 (Suppression des vieux messages) *Pour tous processus $P, Q \in \mathcal{T}$, nous notons $P \succeq_{stale} Q$ si P est Q avec un vieux message de plus (comme définis par la condition d'approximation (4) de la définition A.2.1). Les diagrammes suivants sont alors satisfaits :*

$$\begin{array}{ccc} P & \xrightarrow{\succeq_{stale}} & Q \\ \downarrow{\scriptstyle Fwd} & (1) & \vdots{\scriptstyle Fwd,=} \\ P' & \overset{\succeq_{stale}}{\cdots\!\succ} & Q' \end{array} \qquad \begin{array}{ccc} P & \xrightarrow{\succeq_{stale}} & Q \\ \downarrow{\scriptstyle z} & (2) & \downarrow{\scriptstyle z} \\ P' & \overset{\succeq_{stale}}{\cdots\!\succ} & Q' \end{array} \qquad \begin{array}{ccc} P & \xrightarrow{\succeq_{stale}} & Q \\ \vdots{\scriptstyle z} & (3) & \downarrow{\scriptstyle z} \\ P' & \overset{\succeq_{stale}}{\cdots\!\succ} & Q' \end{array}$$

où $\twoheadrightarrow_z$ est une réduction de $X \cup B \setminus \{\twoheadrightarrow_{Fwd}\}$ pour le diagramme (2) et où $\twoheadrightarrow_z$ est une réduction de $X \cup B$ pour le diagramme (3).

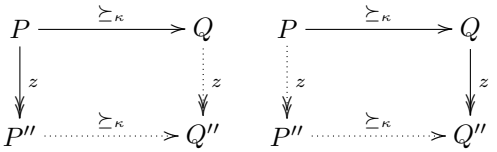
Preuve: Nous remarquons que la relation $\succeq_{stale}$ commute immédiatement avec $\sim_\alpha$. Nous coupons chaque diagramme en deux : le diagramme de commutation avec $\sim_\alpha$, et le reste du diagramme que nous prouvons maintenant, sans renommage initial.

Supposons que M^m contienne un vieux message $sub_{in}^n(i, \_, \_)$ ou $sub_{out}^n(i, \_, \_)$. Ce message peut être consommé soit par une règle de D_1^n, soit par une règle de D_F^n. Cependant, par définition des vieux messages dans P, nous avons $i \neq i^m$ pour tout $m \in N$ qui satisfait la condition (7a). Par conséquent, il ne peut y avoir de message $amb^n(i, \_, \_)$ dans M^n (ce qui satisferait la condition (7(a)i)), donc aucune règle de D_1^n ne peut consommer ce vieux message. Dans le cas où ce message est consommé par une règle de D_F^n, le message est propagé sur un autre nom sub_{in}^m ou sub_{out}^m avec les mêmes arguments, le message reste donc vieux. C'est dans ce cas que Q' est égal à Q si c'est ce message qui est détruit par la relation, dans le premier diagramme. Les autres étapes sont complètement indépendantes en ce qui concerne les vieux messages, puisque les étapes modifiant les uid (continuation ou étape GO) en créent des frais, et les autres étapes ne modifient pas les uid, donc le statut de vieux message n'est jamais modifié. □

Une autre relation de simplification $\succeq_\kappa$ supprime les définitions des noms qui ne sont plus utilisés.

Lemme A.2.12 (Suppression des définitions) *Pour tous $P, Q \in \mathcal{T}$, nous notons $P \succeq_\kappa Q$ si Q est P après avoir enlevé un nom κ de K_{ab} (ou un uid i de I) pour un nom κ (ou i) qui n'est pas présent dans Q.*

Pour toute réduction $\twoheadrightarrow_z$, nous avons les diagrammes :



Preuve: Nous étudions tout d'abord la commutation de la relation $\succeq_\kappa$ avec le renommage $\sim_\alpha$. Si $P \succeq_\kappa Q$ et $P \sim_\alpha P'$ alors $Q \sim_\alpha Q'$ avec le même renommage et $P' \succeq_\kappa Q'$.

Si $P \succeq_\kappa Q$ et $Q \sim_\alpha Q'$, nous considérons le renommage qui est identique à $\sim_\alpha$ excepté qu'il renomme κ (ou i) en un nom frais. Ce renommage est bien injectif et ne capture pas de nom libre, nous avons donc $P \sim_\alpha P'$ et $P' \succeq_\kappa Q'$.

Par hypothèse, le nom κ (ou i) n'apparaît que dans sa définition, il ne peut donc jamais apparaître dans un message que ce soit dans P' ou dans Q'. Supprimer la définition de κ commute donc bien avec toutes les réductions. Dans le cas où le nom κ apparaît en Q'' après la réduction $Q \twoheadrightarrow_z Q''$ dans le diagramme de droite, κ est renommé en un nom frais dans P' par $\sim_\alpha$. □

Les locations sont des boîtes qui ne peuvent pas être dissoutes dans le join calcul, empêchant ainsi une correspondance directe avec les étapes Open 2 du calcul des ambients étendus. Nous utilisons donc une relation de simplification $\succeq_{Fwd}$ qui fusionne une location ouverte avec sa location père. Dans le lemme suivant, la location m est ouverte dans n et toutes les étapes propageant les messages de m dans n ont été effectuées.

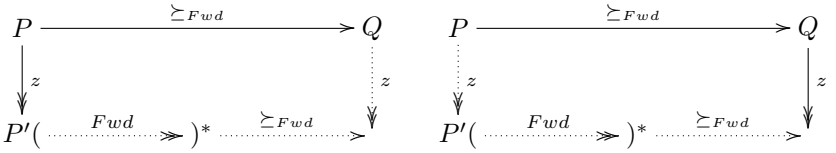
Lemme A.2.13 (Suppression des locations) *Soient $P, Q \in \mathcal{T}$. Nous notons $P \succeq_{Fwd} Q$ s'il existe $m \in N_P^+$, tel que P satisfait les conditions :*

1. *$S_P^m = f^m(e^n)$ (donc $\alpha_P^m = \alpha_P^n h^n$ par la condition d'approximation 7(b)i) ;*
2. *M_P^m est une composition parallèle de messages sur les noms de K_P^m ;*

et tel que Q est P avec les modifications suivantes :

3. *$N_Q = N_P \setminus \{m\}$;*
4. *$K_Q^n = K_P^n \cup K_P^m$ (avec les mêmes processus gardés Q^κ) ;*
5. *$M_Q^n = M_P^n \mid M_P^{'m}$, avec $M_P^{'m}$ étant M_P^m dans lequel les noms de e^m sont remplacés par les noms de e^n ;*
6. *pour tout $p \in N_Q$, la chaîne α_Q^p est α_P^p sans h^m ;*
7. *les noms de e^m sont remplacés par les noms correspondant de e^n.*

Pour toute réduction $\twoheadrightarrow_z \in X \cup B$, nous avons les diagrammes :



De plus, soient $P, Q \in \mathcal{T}$ satisfaisant toutes les conditions précédentes à l'exception de (2). Nous avons $P \twoheadrightarrow^*_{Fwd} \succeq_{Fwd} Q$.

Preuve: Nous commençons par démontrer la seconde partie du lemme. Nous supposons donc que P et Q satisfont toutes les conditions du lemme excepté la condition (2). Puisque P est un état d'approximation, la condition d'approximation (2) pour m nous indique que M_P^m à la forme (à équivalence $\equiv_{AC0}$ près) $M_{P,\kappa}^m \mid M_{P,f}^m$ avec $M_{P,\kappa}^m$ étant une composition parallèle de messages sur des noms de K_P^m et $M_{P,f}^m$ est une composition parallèle de messages de la forme $x^m(\widetilde{y})$ avec $x \in \{amb^n, in^n, out^n, open^n, recv^n, send^n, sub_{in}^n, sub_{out}^n, o^n, r^n\}$. La location h^m étant ouverte, la condition d'approximation (7) nous garantit qu'aucun x^m n'est r^m ou o^m. Par définition, pour chacun des x^m nous avons une règle dans D_F^m de la forme :

$$f^m(e) \mid x^m(\widetilde{v}) \triangleright f^m(e) \mid e.x(\widetilde{v})$$

De plus, nous avons $S_P^m = f^m(e^n)$ par la condition (1) du lemme, donc ces réductions peuvent avoir lieu, et chacun des messages $x^m(\widetilde{v})$ peut être propagé en un message $x^n(\widetilde{v})$ qui est ensuite envoyé en h^n par une étape Comm. La configuration résultant est bien un état d'approximation. Nous appliquons ces deux étapes à chaque message de $M_{P,F}^m$, et nous avons $P \twoheadrightarrow_{Fwd}^* P'$ et $P' \succeq_{Fwd} Q$.

Nous prouvons maintenant les différents diagrammes. Nous remarquons tout d'abord que la relation $\succeq_{Fwd}$ commute avec le renommage $\sim_\alpha$. Nous supposons par conséquent que ce renommage a déjà eu lieu. Nous mettons en correspondance les étapes $P \twoheadrightarrow_z P'$ et $Q \twoheadrightarrow_z Q'$, en substituant n par m dans chaque redex. Pour chaque paire d'étapes en relation, nous utilisons souvent la deuxième partie du lemme pour obtenir la relation $P' \twoheadrightarrow_{Fwd}^* \succeq_{Fwd} Q'$. Nous procédons donc par cas selon la réduction de $X \cup B$ ayant lieu.

Etape Go pour $S^p = \mathsf{go}\ h^q; I^p$. Par la condition (1) du lemme, nous avons nécessairement $p \neq m$. Dans le cas où q est différent de m, le résultat est immédiat et nous avons $P' \succeq_{Fwd} Q'$. Dans le cas où $q = m$, la location h^p migre dans h^m dans P et génère un message sur amb^m, alors que cette location migre dans h^n dans Q et génère un message sur amb^n. Toutes les conditions du lemme excepté la condition (2) étant satisfaites, nous avons par la deuxième propriété du lemme $P' \twoheadrightarrow_{Fwd}^* \succeq_{Fwd} Q'$ (en fait une seule étape $\twoheadrightarrow_{Fwd}$ a lieu, c'est celle qui propage le message sur amb^m).

Etape de continuation, étape de réplication Le résultat est immédiat, excepté lorsque le message consommé est un message sur un nom de K_P^m dans P et sur le même nom de K_Q^n dans Q. Dans ce cas, la condition (4) du lemme garantit que le processus gardé généré est le même, à la seule différence que la traduction est paramétrée par e^m dans le premier cas et par e^n dans le deuxième cas. La configuration obtenue dans les deux cas après normalisation P' étant une extension d'état de P en m et Q' étant une extension d'état de Q et n, nous vérifions que les conditions du lemme sont respectés (excepté la condition (2)). Nous nous basons donc sur la définition A.2.4 pour prouver les conditions du lemme.

La condition (1) n'est pas modifiée. La condition (3) est satisfaite puisque N_Q et N_P sont étendus avec le même ensemble d'indices. La condition (4) est satisfaite puisque K_Q^n et K_P^m sont étendus avec le même ensemble de noms, et les processus ambients gardés associés sont identiques. La condition (5) est satisfaite puisque $M_{Q'}^n = M_Q^n \mid M_{\kappa n} = M_P^n \mid M_P^m \mid M_{\kappa n} = M_P^n \mid M_{P'}^{'m} = M_{P'}^n \mid M_{P'}^{'m}$ puisqu'aucun nouveau message n'est introduit dans M_P^n (par définition de l'extension d'état), puisque les messages de M_P^m ne contiennent pas de noms de e^m, et puisque (notant $M_{\kappa n}$ (respectivement $M_{\kappa m}$) les messages introduits par la traduction paramétré par e^n (respectivement e^m)) $M_{\kappa n}$ est $M_{\kappa m}$ dans lequel les noms de e^m sont remplacés par les noms de e^n. La condition (6) est satisfaite puisque pour toutes les nouvelles locations h^q introduites, nous avons pour $P : \alpha^q = \alpha^n h^n h^m \beta$ et pour $Q : \alpha^q = \alpha^n h^n \beta$. La condition (7) est satisfaite puisque nous remplaçons une traduction paramétrée par e^m par une traduction paramétrée par e^n.

Nous concluons en appliquant la deuxième propriété du lemme.

Etape 0 Si l'étape prend place à l'indice $p \in N_P$ avec S^p de la forme $s^p(\_, \_, e^m, \_)$ dans P, alors S^p a la forme $s^p(\_, \_, e^n, \_)$ dans Q. Après l'étape Comm amenant le message généré dans M^m (respectivement dans M^n), les processus P' et Q' satisfont les conditions du lemme, excepté la condition (2). Nous concluons alors en utilisant la deuxième partie du lemme. Dans les autres cas, le résultat est immédiat puisque cette étape ne peut avoir lieu dans m dans P, par la condition du lemme (1) et la condition d'approximation (7).

Etape de communication, étape 1, étape 2 Par la condition (1) et la condition d'approximation (7) pour m, ces étapes ne peuvent avoir lieu à l'indice m. Nous remarquons qu'aucune des étapes 1 ne peut générer de message sur r^m ou o^m, puisque après normalisation P' ne serait pas un état d'approximation, ce qui contredirait le lemme A.2.7 associé aux lemmes A.2.8 et A.2.2(2). Dans le cas où ces étapes génèrent une continuation sur un nom de $K_{P,ab}^n$, ce message se retrouve respectivement (après une étape Comm) dans M_P^m et M_Q^n. Les conditions (2,5) sont toujours satisfaites. Dans les autres cas, les étapes sont immédiatement en correspondance.

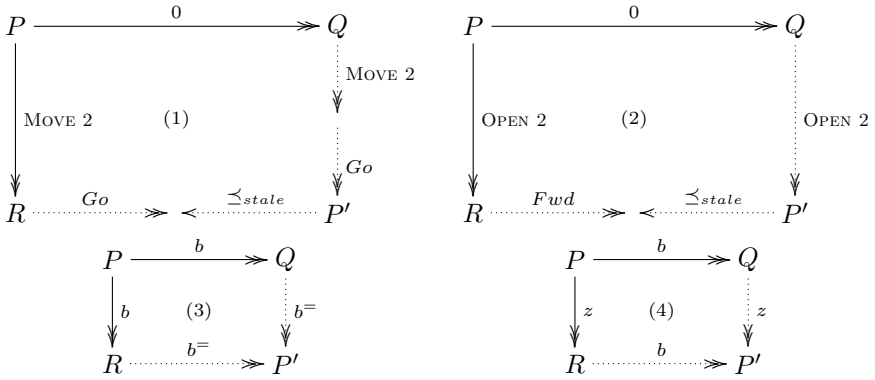
Etape de propagation La condition 2 nous garantit que cette étape ne peut propager un message de M_P^m. Par contre, un message d'une sous-location ouverte de h^m peut être propagé dans M_P^m. Dans ce cas, les conditions du lemme sont satisfaites excepté la condition (2), et nous concluons par la deuxième partie du lemme (qui propage le message en n). Les autres étapes de propagation sont en correspondance directe.

□

Nous étudions maintenant les propriétés de commutation des étapes de bookkeeping $\twoheadrightarrow_b$ avec les étapes $\twoheadrightarrow_z$ pouvant s'appliquer aux états d'approximation bien formés.

Lemme A.2.14 (Bookkeeping) *Pour tout $P \in \mathcal{T}$ bien formé, pour tous $\twoheadrightarrow_b \in B$ et $\twoheadrightarrow_z \in B \cup X$, les diagrammes suivants sont satisfaits avec :*

- *pour $b = 0$ et $z =$* Move 2 *nous avons soit (1), soit (4).*
- *pour $b = 0$ et $z =$* Open 2 *nous avons soit (2), soit (4).*
- *pour $b = z$ nous avons (3).*
- *Pour les autres b et z nous avons (4).*



Preuve: Nous procédons par cas sur l'étape $\twoheadrightarrow_b \in B$ utilisée. Nous nous basons aussi sur la définition A.2.1.

Etape Go Cette étape consomme une instruction `go`, qui commute avec toutes les autres étapes, excepté éventuellement une autre instruction `go`. Cependant, le lemme A.2.10 nous garantit que tout instruction `go` présente dans un état d'approximation peut être exécutée, donc la commutation est aussi possible dans ce cas.

Etape de continuation, étape de réplication Ces étapes consomment un message sur $\kappa \in K$, qui sont uniques par les conditions (4,6) et dont les définitions sont elles aussi uniques, par la condition (1). Ces étapes commutent donc entre elles et avec toutes les autres étapes.

Etape de propagation Une telle étape utilisant une règle de D_F^n impose que P satisfasse la condition (7(b)i) pour n, ce qui exclut tout autre étape de D^n excepté une autre étape de propagation. De plus, les étapes de propagation ne modifient pas le message $f^n(e^m)$, donc elles commutent entre elles.

Etape 0 Soit $n \in N$ tel que cette étape 0 utilise une règle de D_0^n qui consomme un message $in^n(a, \kappa)$ (ou $out^n(a, \kappa)$) et un message $S^n = s(b, i^n, e^m, l^m)$. Cette étape produit un message $amb_{in}^m(i^n, a, \kappa)$ (ou $amb_{out}^m(i^n, a, \kappa)$) et le message $s^n(b, i^n, e^m, l')$ avec l' étant l^n avec une entrée supplémentaire. Nous remarquons tout d'abord que par la condition d'approximation (1) nous ne pouvons avoir $n = 0$, puisque qu'aucun message sur *lock* n'est présent.

Nous remarquons que P satisfait nécessairement la condition (7a) en n, avant et après l'étape. Par conséquent, aucune étape de D_F^n n'est possible à partir de P.

Les étapes 1 commutent immédiatement avec cette étape.

Les étapes 0 commutent aussi entre elles, puisque l'ordre dans le journal n'est pas pris en compte.

Nous étudions maintenant le cas des étapes 2 pouvant ne pas commuter avec cette étape 0, c'est à dire les étapes utilisant une règle de D_2^n. Ces cas correspondent aux diagrammes (1) et (2).

En ce qui concerne le diagramme (1), l'étape Move 2 peut avoir lieu après l'étape 0. Dans les deux cas, l'instruction `go` produite peut être exécutée (comme le garantit le lemme A.2.10), et la normalisation suivant cette étape Go réémet les messages du journal et génère un message sur s^n avec un nouvel uid frais i. Dans le cas où l'étape 0 a lieu avant l'étape Move 2, le journal contient une entrée supplémentaire correspondant au message délégué. Ce message est donc à nouveau généré lorsque le journal est vidé. L'unique différence entre effectuer l'étape 0 avant l'étape Move 2 ou ne pas le faire consiste en un message délégué de la forme $amb_{in}^m(i^n, a, \kappa)$ (ou $amb_{out}^m(i^n, a, \kappa)$), qui est désormais un vieux message puisque l'uid portée par le message sur s^n est fraîche.

Le diagramme (2) décrit le cas d'une étape Open 2. Comme auparavant, cette étape peut avoir lieu avant comme après l'étape 0. Dans les deux cas, la normalisation suivant l'étape

Open 2 vide le journal en utilisant les noms in^m ou out^m au lieu de in^n ou out^n. Les différences entre ces deux séries d'étapes sont donc les suivantes. Si l'étape 0 n'a pas eu lieu, M^n contient un message $in^n(a,\kappa)$ (ou $out^n(a,\kappa)$), alors que dans l'autre cas ce message n'est pas présent et M^m contient le message supplémentaire $in^m(a,\kappa)$ (ou $out^m(a,\kappa)$). Puisque n est ouvert, pour réconcilier cette différence, ce message est propagé en m. L'autre différence est la présence d'un vieux message délégué par l'étape 0 si celle-ci a eu lieu (le message est vieux puisque le message sur s^n contenant l'uid i^n n'existe plus). Cette différence est prise en compte par la relation $\preceq_{stale}$.

□

A.2.3 Correspondance opérationnelle

Nous prouvons maintenant les diagrammes élémentaires qui mettent en correspondance les étapes de réduction du calcul des ambients étendus et les étapes de leur traduction.

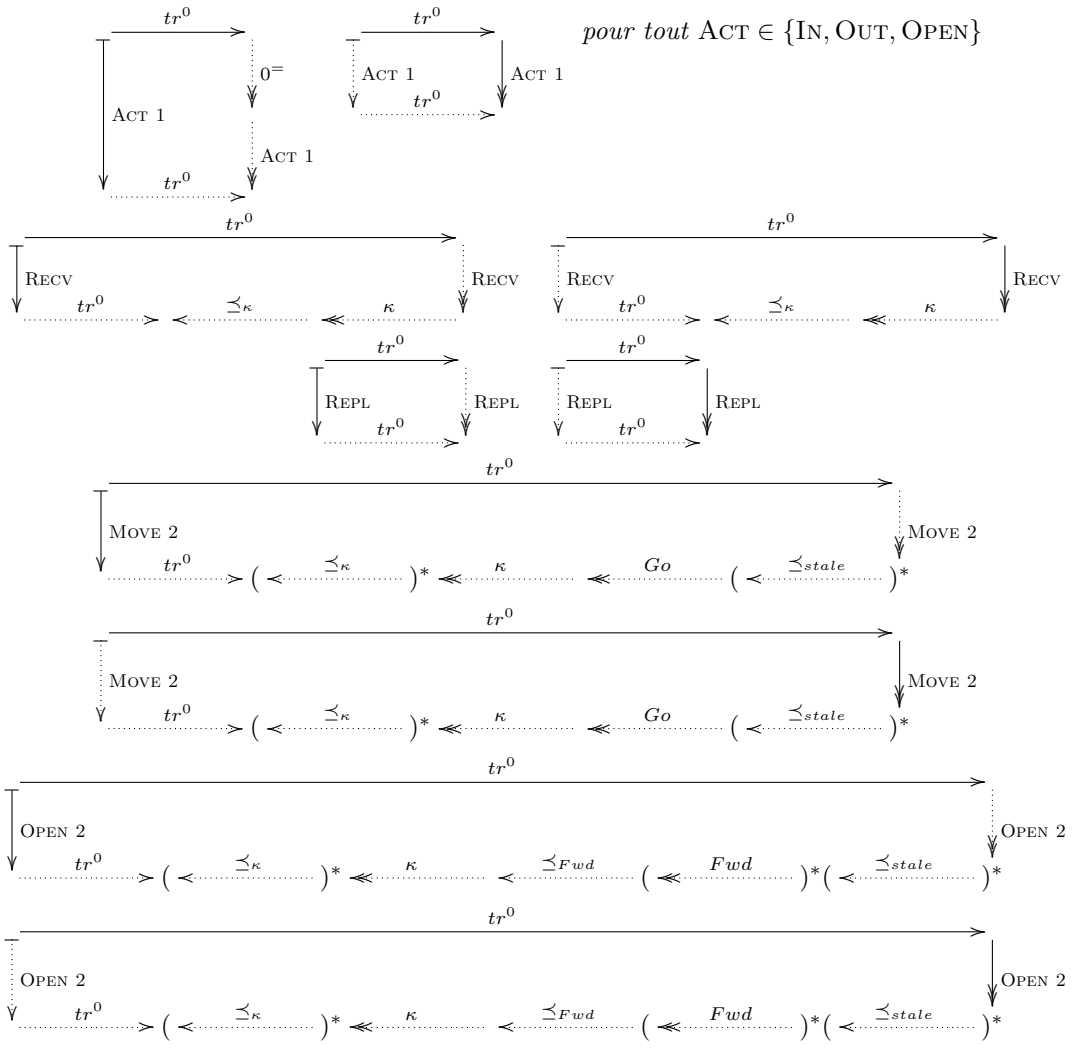
Comme pour les étapes possibles pour un état d'approximation, nous partitionnons les étapes de réduction du calcul des ambients étendus $\rightarrow_{12C}$ en plusieurs familles de réductions :

$$\rightarrow_{\text{In }1}, \rightarrow_{\text{Out }1}, \rightarrow_{\text{Open }1}, \rightarrow_{\text{Move }2}, \rightarrow_{\text{Open }2}, \rightarrow_{\text{Recv}}, \rightarrow_{\text{Repl}}$$

Nous notons $\rightarrow_x$ pour une telle famille de réductions, et notons $\mapsto_x$ pour ces mêmes réductions sans utiliser d'équivalence structurelle avant et après l'étape.

Puisque certaines étapes 1 ne sont pas possibles avant qu'il n'y ait eu une étape de réduction, nous considérons en fait la relation $\rightarrow_{tr^0} \stackrel{\text{def}}{=} [\![\cdot]\!]^{t\sharp} \twoheadrightarrow_0^*$ entre les processus ambients étendus dérivant d'un processus ambient classique et les états d'approximation. Par conséquent, toute configuration à droite de cette relation est un processus bien formé.

Lemme A.2.15 (Correspondance opérationnelle) *Les diagrammes suivants sont satisfaits :*



Preuve: Cette preuve est longue mais conceptuellement simple. Pour chaque diagramme, nous utilisons la correspondance entre la traduction étendue et les états d'approximation, ainsi que l'étape ayant lieu pour en déduire la forme du terme se réduisant. Nous fermons chaque diagramme en appliquant les réductions et relations de simplification nécessaires.

Nous commençons par le cas In 1. Nous avons la réduction ambient suivante, pour un contexte d'évaluation E :

$$E(m[P] \mid n[\mathtt{in}\ m.Q \mid R]) \mapsto_{In1} E(\nu i.\ m[i \mid P] \mid \overline{i}\{Q\}{\multimap}n[R])$$

De la traduction étendue, nous déduisons que trois locations sont impliquées : la location représentant m, celle représentant n et leur location père. Nous avons donc un sous-terme de la traduction de la forme :

$$\begin{array}{ll} & \alpha^p h^p[D^p, E^p : S^p \mid M^p \mid amb^p(i^m, m, e^m) \mid amb^p(i^n, n, e^n)] \\ \parallel & \alpha^p h^p h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^p h^p h^n[D^n, E^n, \kappa() \triangleright [\![Q]\!]_{e^n} : S^n \mid M^n \mid in^n(m, \kappa)] \end{array}$$

Les étapes 0 ne modifient pas la structure de ce sous-terme, à l'exception éventuelle du message sur in^n qui peut être délégué. Par définition de la traduction, nous remarquons que les messages d'état S^p, S^m et S^n sont nécessairement de la forme $s^p(p, i^p, e, l^p)$, $s^m(m, i^m, e^p, l^m)$ et $s^n(n, i^n, e^p, l^n)$.

Supposons que le message sur in^n n'a pas été délégué, nous construisons alors la série de réductions (dans le cas où il a été délégué, il suffit de sauter la première étape) :

$$\begin{array}{ll} & \alpha^p h^p[D^p, E^p : S^p \mid M^p \mid amb^p(i^m, m, e^m) \mid amb^p(i^n, n, e^n)] \\ \parallel & \alpha^p h^p h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^p h^p h^n[D^n, E^n, \kappa() \triangleright [\![Q]\!]_{e^n} : S^n \mid M^n \mid in^n(m, \kappa)] \\ \twoheadrightarrow_0 & \alpha^p h^p[D^p, E^p : S^p \mid M^p \mid amb^p(i^m, m, e^m) \mid amb^p(i^n, n, e^n) \mid sub^p_{in}(i^n, m, \kappa)] \\ \parallel & \alpha^p h^p h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^p h^p h^n[D^n, E^n, \kappa() \triangleright [\![Q]\!]_{e^n} : S'^n \mid M^n] \\ \twoheadrightarrow_1 & \alpha^p h^p[D^p, E^p : S^p \mid M^p \mid amb^p(i^m, m, e^m)] \\ \parallel & \alpha^p h^p h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^p h^p h^n[D^n, E^n, \kappa() \triangleright [\![Q]\!]_{e^n} : S'^n \mid M^n \mid r^n(e^m, \kappa)] \end{array}$$

avec $S'^n = s^n(n, i^n, e^p, l^n \cup \{\text{In } b\ \kappa\})$.

Le diagramme se ferme donc, en appliquant la même série d'étapes 0 qu'initialement, excepté bien entendu la délégation du message sur in^n si elle avait eu lieu. Nous remarquons l'utilité de pouvoir associer une entrée de notre choix au journal d'une souche migrante.

Le diagramme réciproque est établi de la manière suivante. Par définition de la traduction, l'étape de droite est de la forme :

$$\begin{array}{ll} & \alpha^p h^p[D^p, E^p : S^p \mid M^p \mid amb^p(i^m, m, e^m) \mid amb^p(i^n, n, e^n) \mid sub^p_{in}(i^n, m, \kappa)] \\ \parallel & \alpha^p h^p h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^p h^p h^n[D^n, E^n, \kappa() \triangleright [\![Q]\!]_{e^n} : S'^n \mid M^n] \\ \twoheadrightarrow_1 & \alpha^p h^p[D^p, E^p : S^p \mid M^p \mid amb^p(i^m, m, e^m)] \\ \parallel & \alpha^p h^p h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^p h^p h^n[D^n, E^n, \kappa() \triangleright [\![Q]\!]_{e^n} : S'^n \mid M^n \mid r^n(e^m, \kappa)] \end{array}$$

Nous appliquons l'étape In 1 sur le processus ambient, ce qui est possible pour les raisons suivantes. Le message $sub^p_{in}(i^n, m, \kappa)$ ne pouvant être créé par la traduction, il a été nécessairement délégué, et par définition des étapes 0, cela signifie qu'un message $in^n(m, \kappa)$ est présent en n après la traduction, ce qui implique la présence d'une capacité $\mathtt{in}\ m$ dans l'ambient n. La définition de κ implique que la continuation de la capacité est le processus ambient Q. De plus, l'arbre des ambients et des locations étant isomorphes après traduction, l'ambient n est bien voisin de l'ambient m. Nous fermons ensuite le diagramme comme auparavant.

Les cas Out 1 et Open 1 sont quasiment identiques, nous ne les détaillons pas.

Nous détaillons maintenant le cas d'une communication Recv. Soit une étape de la forme : $E((x).Q \mid \langle b \rangle) \mapsto_{Recv} E(Q\{^b/_x\})$. Nous avons après traduction et délégations un sous-terme join de la forme

$$\alpha^n h^n[D^n, E^n, \kappa(x) \triangleright [\![Q]\!]_{e^n} : S^n \mid M^n \mid recv^n(\kappa) \mid send(b)]$$

avec $S^n = s^n(a, i^n, e^p, l^n)$.

Nous avons la série de réductions suivante :

$$\begin{array}{rl} & \alpha^n h^n[D^n, E^n, \kappa(x) \triangleright [\![Q]\!]_{e^n} : S^n \mid M^n \mid recv^n(\kappa) \mid send(b)] \\ \mapsto_{Recv} & \alpha^n h^n[D^n, E^n, \kappa(x) \triangleright [\![Q]\!]_{e^n} : S^n \mid M^n \mid \kappa(b)] \\ \mapsto_{\kappa} & \alpha^n h^n[D^n, E^n, \kappa(x) \triangleright [\![Q]\!]_{e^n} : S^n \mid M^n \mid [\![Q\{^b/_x\}]\!]_{e^n}] \end{array}$$

en appliquant le lemme A.2.3(2) pour faire passer la substitution sous la traduction.

Nous considérons la configuration où l'on a enlevé de h^n la définition de κ ainsi que le processus $[\![Q\{^b/_x\}]\!]_{e^n}$. Cette configuration étant un état d'approximation, nous pouvons lui appliquer le lemme A.2.5 avec le processus ambient $Q\{^b/_x\}$ en n. Nous obtenons donc un état d'approximation P' tel que (en appliquant les même étapes de délégation qu'initialement) :

$$E(Q\{^b/_x\}) \rightarrow_{tr} \twoheadrightarrow_0^* P'$$

et

$$C \parallel \alpha^n h^n[D^n, E^n : S^n \mid M^n \mid [\![Q\{^b/_x\}]\!]_{e^n}] \twoheadrightarrow_d P'$$

Nous pouvons appliquer cette deuxième série de réductions à la configuration :

$$C \parallel \alpha^n h^n[D^n, E^n, \kappa(x) \triangleright [\![Q]\!]_{e^n} : S^n \mid M^n \mid [\![Q\{^b/_x\}]\!]_{e^n}] \twoheadrightarrow_d P''$$

Pour refermer le diagramme, nous remarquons que la condition d'approximation (5) pour κ nous garantit que toutes ses occurrences, excepté sa définition, ont disparu. Nous pouvons donc appliquer la relation de simplification $P' \preceq_\kappa P''$ pour supprimer cette définition.

Le diagramme réciproque se prouve de manière identique, en remarquant que les messages join consommés par l'étape Recv impliquent par définition de la traduction l'existence d'un processus ambient $(x).Q \mid \langle b \rangle$.

Le cas de la réplication est similaire en plus simple, sachant qu'il ne faut pas supprimer la définition de la réplication. Une fois encore, le diagramme réciproque se base sur le fait que la seule possibilité pour avoir un message sur un nom de K_c est d'avoir un processus ambient répliqué.

Nous étudions maintenant le cas d'une étape Move 2. Nous supposons que la location associée à la souche migrante porte l'indice n, et la location contenant le scion correspondant porte l'indice p. L'invariant (2) du lemme A.1.2 nous indique que le processus ambient initial à la forme $P = E(i)(\bar{\imath}\{P_i\}\text{–}n[\, R_i \,])$ pour un contexte d'évaluation étendu $E(\,\cdot\,)(\,\cdot\,)$. Par conséquent le scion i n'est pas dans sa souche, et par définition de la traduction, la traduction de ce terme est donc de la forme :

$$\begin{array}{rl} & C \parallel \alpha^p h^p[D^p, E^p : S^p \mid M^p] \\ \parallel & \alpha^m h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^m h^m h^n[D^n, E^n, \kappa \triangleright [\![P_i]\!]_{e^n} : s^n(n, i^n, e^m, l_0^n) \mid r^n(e^p, \kappa) \mid M^n] \end{array}$$

avec l_n^0 contenant une unique entrée portant la continuation κ. Nous remarquons que n ne peut pas être l'indice 0, c'est pourquoi nous explicitons sa location père, par contre p le peut (la forme du terme pour cette location est alors légèrement différente).

Nous partitionnons les étapes de délégation ayant lieu après la traduction en deux groupes : celles déléguant des messages de h^n, et les autres. Nous remarquons que ce deuxième groupe de délégations peut avoir lieu après traduction du processus ambient $Q = E(\, n[\, P_i \mid R_i \,])(\mathbf{0})$, puisque aucune relation père fils n'est modifiée, excepté celle entre n et m. Nous obtenons donc après délégation de messages de n la configuration :

$$\begin{array}{rl} & C \parallel \alpha^p h^p[D^p, E^p : S^p \mid M^p] \\ \parallel & \alpha^m h^m[D^m, E^m : S^m \mid M^m \mid M'^m] \\ \parallel & \alpha^m h^m h^n[D^n, E^n, \kappa \triangleright [\![P_i]\!]_{e^n} : s^n(n, i^n, e^m, l^n) \mid r^n(e^p, \kappa) \mid M'^n] \end{array}$$

avec M'^m composé de messages de la forme $sub_{in}^m(i^n, b_j, \kappa_j)$ ou $sub_{out}^m(i^n, b_k, \kappa_k)$, le log l^n contenant l'entrée initiale, ainsi que des entrées In b_j κ_j pour chaque j et Out b_k κ_k pour chaque k, le

processus M'^n étant M^n auquel on a retiré les messages $in(b_j, \kappa_j)$ pour chaque j et $out(b_k, \kappa_k)$ pour chaque k.

Ce processus peut effectuer une étape Move 2, et nous obtenons la configuration :

$$\begin{array}{ll} & C \parallel \alpha^p h^p[D^p, E^p : S^p \mid M^p] \\ \parallel & \alpha^m h^m[D^m, E^m : S^m \mid M^m \mid M'^m] \\ \parallel & \alpha^m h^m h^n[D^n, E^n, \kappa \triangleright [\![P_i]\!]_{e^n} : (\texttt{go}\ h^p; (I_{n,e^n,e^p} \mid \kappa() \mid Flush(l^n, in^n, out^n, \kappa))) \mid M'^n] \end{array}$$

Nous remarquons que le seul message de la forme $s^n(\_, i^n, \_, \_)$ a été consommé (ce message est unique par la condition d'approximation (7)). Par conséquent, tous les messages de M'^m sont vieux, et peuvent être supprimés par la relation de simplification $\succeq^*_{stale}$. Nous obtenons la configuration :

$$\begin{array}{ll} & C \parallel \alpha^p h^p[D^p, E^p : S^p \mid M^p] \\ \parallel & \alpha^m h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^m h^m h^n[D^n, E^n, \kappa \triangleright [\![P_i]\!]_{e^n} : s^n(n, i^n, e^m, l^n) \mid r^n(e^p, \kappa) \mid M'^n] \end{array}$$

Les lemmes A.2.10 et A.2.11 nous garantissant que l'étape Go peut s'effectuer, nous obtenons :

$$\begin{array}{ll} & C \parallel \alpha^p h^p[D^p, E^p : S^p \mid M^p] \\ \parallel & \alpha^m h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^p h^p h^n[D^n, E^n, \kappa \triangleright [\![P_i]\!]_{e^n} : \\ & (\texttt{def uid}\ i\ \texttt{in}\ s^n(n, i, e^p, \emptyset) \mid amb^p(i, n, e^n)) \mid \kappa() \mid Flush(l^n, in^n, out^n, \kappa) \mid M'^n] \\ =_\alpha & C \parallel \alpha^p h^p[D^p, E^p : S^p \mid M^p] \\ \parallel & \alpha^m h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^p h^p h^n[D^n, E^n, \kappa \triangleright [\![P_i]\!]_{e^n} : \\ & (\texttt{def uid}\ i'^n\ \texttt{in}\ s^n(n, i'^n, e^p, \emptyset) \mid amb^p(i'^n, n, e^n)) \mid \kappa() \mid Flush(l^n, in^n, out^n, \kappa) \mid M'^n] \\ \twoheadrightarrow & C \parallel \alpha^p h^p[D^p, E^p : S^p \mid M^p] \\ \parallel & \alpha^m h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^p h^p h^n[D^n, E^n, \kappa \triangleright [\![P_i]\!]_{e^n}, \texttt{uid}\ i'^n : \\ & s^n(n, i'^n, e^p, \emptyset) \mid amb^p(i'^n, n, e^n) \mid \kappa() \mid Flush(l^n, in^n, out^n, \kappa) \mid M'^n] \\ \twoheadrightarrow^*_d & C \parallel \alpha^p h^p[D^p, E^p : S^p \mid M^p \mid amb^p(i'^n, n, e^n)] \\ \parallel & \alpha^m h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^p h^p h^n[D^n, E^n, \kappa \triangleright [\![P_i]\!]_{e^n}, \texttt{uid}\ i'^n : s^n(n, i'^n, e^p, \emptyset) \mid \kappa() \mid M^n] \\ \Rightarrow_d & C' \parallel \alpha^p h^p[D^p, E^p : S^p \mid M^p \mid amb^p(i'^n, n, e^n)] \\ \parallel & \alpha^m h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^p h^p h^n[D^n, E^n, \kappa \triangleright [\![P_i]\!]_{e^n} : s^n(n, i'^n, e^p, \emptyset) \mid \kappa() \mid M^n] \end{array}$$

avec C' étant C dans lequel la location h^0 contient la nouvelle définition $\texttt{uid}\ i'^n$, et avec i'^n étant un uid frais. L'étape $\twoheadrightarrow^*_d$ transporte le message sur amb^p en h^p et réémet tous les messages ayant une entrée dans le log ne contenant pas κ. La seule entrée contenant κ étant l'entrée initiale (par la condition d'approximation (4c) sur le processus traduit initialement), tous les messages ayant été délégués sont réémis, redonnant le processus M^n initial.

Nous fermons le diagramme en appliquant la même technique que pour le cas Recv. Nous considérons la configuration dans laquelle le message sur κ, sa définition, ainsi que la définition $\texttt{uid}\ i^n$ de l'ancien uid ont été enlevés :

$$\begin{array}{rcll} C_2 & = & C'' \parallel \alpha^p h^p[D^p, E^p : S^p \mid M^p \mid amb^p(i'^n, n, e^n)] \\ & & \parallel \alpha^m h^m[D^m, E^m : S^m \mid M^m] \\ & & \parallel \alpha^p h^p h^n[D^n, E^n : s^n(n, i'^n, e^p, \emptyset) \mid M^n] \end{array}$$

Nous remarquons que nous avons $E(\, n[\, R_i\,]\,) \rightarrow_{tr} \twoheadrightarrow^*_0 C_2$ en appliquant les mêmes délégations qu'initialement (excepté celles de h^n).

Cette configuration étant un état d'approximation, nous pouvons lui appliquer le lemme A.2.5 avec le processus ambient P_i en n. Nous avons donc $E(\, n[\, P_i \mid R_i\,]\,) \rightarrow_{tr} \twoheadrightarrow^*_0 P'$ et $C'_2 \twoheadrightarrow_d P'$, avec

C_2' étant :

$$\begin{array}{rcl} C_2' & = & C'' \parallel \alpha^p h^p[D^p, E^p : S^p \mid M^p \mid amb^p(i'^n, n, e^n)] \\ & \parallel & \alpha^m h^m[D^m, E^m : S^m \mid M^m] \\ & \parallel & \alpha^p h^p h^n[D^n, E^n : s^n(n, i'^n, e^p, \emptyset) \mid [\![P_i]\!]_{e^n} \mid M^n] \end{array}$$

Nous construisons donc la série de réductions :

$$\begin{array}{rl} & C' \parallel \alpha^p h^p[D^p, E^p : S^p \mid M^p \mid amb^p(i'^n, n, e^n)] \\ \parallel & \alpha^m h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^p h^p h^n[D^n, E^n, \kappa \triangleright [\![P_i]\!]_{e^n} : s^n(n, i'^n, e^p, \emptyset) \mid \kappa() \mid M^n] \\ \mapsto_\kappa & C' \parallel \alpha^p h^p[D^p, E^p : S^p \mid M^p \mid amb^p(i'^n, n, e^n)] \\ \parallel & \alpha^m h^m[D^m, E^m : S^m \mid M^m] \\ \parallel & \alpha^p h^p h^n[D^n, E^n, \kappa \triangleright [\![P_i]\!]_{e^n} : s^n(n, i'^n, e^p, \emptyset) \mid [\![P_i]\!]_{e^n} \mid M^n] \\ \succeq_\kappa \succeq_\kappa \twoheadrightarrow_d & P' \end{array}$$

La première simplification supprime la définition de κ, puisque κ n'est plus présent dans toute la configuration, la seconde supprime la définition `uid` i^n puisque cet uid n'est présent que dans sa définition (les vieux messages le transportant ont été supprimés).

Le diagramme réciproque se prouve de la même façon, en remarquant que la présence d'un message $r^n(e^p, \kappa)$ implique nécessairement que l'ambient correspondant à h^n soit une souche migrante, et que le scion associé soit présent dans l'ambient correspondant à h^p.

Nous étudions enfin le cas d'une étape Open 2. Le processus ambient initial P à la forme $E(\,\mathtt{o}\{P_o\}\text{–}n[\,R_o\,]\,)$. Nous supposons que la location associée à la souche ouverte porte l'indice n. Par la condition d'approximation (1), n ne peut être 0. La configuration obtenue après traduction a donc la forme :

$$C \parallel \alpha^m h^m[D^m, E^m : S^m \mid M^m] \parallel \alpha^m h^m h^n[D^n, E^n, \kappa() \triangleright [\![P_o]\!]_{e^n} : s^n(n, i^n, e^m, \emptyset) \mid o^n(\kappa) \mid M^n]$$

Nous partitionnons C en quatre ensembles de locations : celles qui ne mentionnent pas un nom de e^n (noté C^0), celles qui ne sont pas sous-location de h^n mais mentionnent un nom de e^n (noté C_p, par définition de la traduction, ces locations sont nécessairement de la forme $\alpha^p h^p[D^p, E^p : S^p \mid r^p(e^n, \kappa) \mid M^p]$), les sous-locations immédiates de h^n (noté C_q, de la forme $\alpha^n h^n h^q[D^q, E^q : s^q(q, i^q, e^n, \emptyset) \mid M^q]$, avec éventuellement un message de la forme $r^q(e^n, \kappa)$ dans M^q), et les sous-locations de ces locations (noté C_s, de la forme $\alpha^n h^n \beta h^s[D^s, E^s : S^s \mid M^s]$, avec β non vide et éventuellement un message de la forme $r^s(e^n, \kappa)$). Par définition de la traduction, toutes les occurrences d'un nom de e^n sont ainsi couvertes (nous rappelons que le nom de l'ambient n lui-même n'est pas un nom de e^n).

Comme dans le cas précédent, nous partitionnons les délégations suivant la traduction. Trois cas doivent être considérés : les délégations de messages de n, les délégations dans n de ses sous-locations C_q, et les autres délégations. En ce qui concerne ces dernières, nous ne les détaillons pas puisqu'elle peuvent s'appliquer immédiatement à la traduction du terme ambient final. Après les étapes 0 de n et de ses sous-locations, nous obtenons la configuration :

$$\begin{array}{rl} & C' \parallel \alpha^m h^m[D^m, E^m : S^m \mid M_0^m] \\ \parallel & \alpha^m h^m h^n[D^n, E^n, \kappa() \triangleright [\![P_o]\!]_{e^n} : s^n(n, i^n, e^m, l^n) \mid o^n(\kappa) \mid M'^n \mid M_0^n] \end{array}$$

avec M_0^m étant M^m avec des messages supplémentaires sur sub_{in}^m et sub_{out}^m, le log l^n contenant les entrées correspondant à ces messages, et M'^n étant M^n auquel on a enlevé les messages sur in^n et out^n correspondant au log l^n (donc aux messages délégués). La configuration C' est $C_0 \parallel C_p \parallel C_q' \parallel C_s$, avec les locations de C_q' de la forme $\alpha^n h^n h^q[D^q, E^q : s^q(q, i^q, e^n, l^q) \mid M'^q]$. Le processus M'^q est M^q sans les messages délégués en h^n et enregistrés dans le journal l^q. Le processus M_0^n correspond aux messages sur sub_{in}^n et sub_{out}^n générés par les délégations des sous-locations de n.

Nous effectuons l'étape Open 2, et obtenons :

$$\begin{array}{rl} & C' \parallel \alpha^m h^m[D^m, E^m : S^m \mid M_0^m] \\ \parallel & \alpha^m h^m h^n[D^n, E^n, \kappa() \triangleright [\![P_o]\!]_{e^n} : f^n(e^m) \mid \kappa() \mid Flush(l^n, in^m, out^m, \kappa) \mid M'^n \mid M_0^n] \end{array}$$

Nous effectuons maintenant l'étape FLUSH, suivi du transport des messages sur in^m et out^m générés. Nous remarquons que ces messages M_n^m correspondent aux messages de n qui ont été délégués. Nous obtenons la configuration :

$$\begin{array}{ll} & C' \parallel \alpha^m h^m[D^m, E^m : S^m \mid M_0^m \mid M_n^m] \\ \parallel & \alpha^m h^m h^n[D^n, E^n, \kappa() \triangleright [\![P_o]\!]_{e^n} : f^n(e^m) \mid \kappa() \mid M^{'n} \mid M_0^n] \end{array}$$

Comme dans le cas MOVE 2, les messages de n qui ont été délégués en m sont vieux, et peuvent être éliminés par la simplification $\succeq_{stale}$. Nous obtenons alors la configuration, notée C_1 :

$$\begin{array}{ll} & C' \parallel \alpha^m h^m[D^m, E^m : S^m \mid M^m \mid M_n^m] \\ \parallel & \alpha^m h^m h^n[D^n, E^n, \kappa() \triangleright [\![P_o]\!]_{e^n} : f^n(e^m) \mid \kappa() \mid M^{'n} \mid M_0^n] \end{array}$$

Nous considérons maintenant la configuration C_2 :

$$C'' \parallel \alpha^m h^m[D^m, E^m, E^{'n}, \kappa() \triangleright [\![P_o]\!]_{e^m} : S^m \mid M^m \mid M_n^m \mid M^{''n} \mid M_0^{'n} \mid \kappa()]$$

où $E^{'n} = E^n\{^{e^m}/_{e^n}\}$, $M^{''n} = M^{'n}\{^{e^m}/_{e^n}\}$ et $M_0^{'n} = M_0^n\{^{e^m}/_{e^n}\}$. La configuration C'' est de la forme $C_0 \parallel C'_p \parallel C''_q \parallel C'_s$, avec les locations de C'_p de la forme $\alpha^p h^p[D^p, E^p : S^p \mid r^p(e^m, \kappa) \mid M^p]$, les locations de C''_q de la forme $\alpha^n h^q[D^q, E^q : s^q(q, i^q, e^m, \emptyset) \mid M^{''q}]$ (avec $M^{''q}$ étant $M^{'q}$ excepté pour un éventuel message $r^q(e^n, \kappa)$ devenant $r^q(e^m, \kappa)$), et les locations de C_s de la forme $\alpha^n \beta h^s[D^s, E^s : S^s \mid M^{'s}]$ (avec $M^{'s}$ étant M^s excepté pour un éventuel message $r^s(e^n, \kappa)$ devenant $r^s(e^m, \kappa)$).

La configuration C'' est donc la configuration C' dans laquelle on a supprimé la location h^n des chaînes des sous-locations de h^n, et dans laquelle on a remplacé toute occurrence d'un nom de e^n par le nom correspondant de e^m.

Les configurations C_1 et C_2 remplissent toutes les conditions du lemme A.2.13, excepté la condition (2). Par conséquent nous avons : $C_1 \twoheadrightarrow^*_{Fwd} \succeq_{Fwd} C_2$.

Nous étudions maintenant la forme de la traduction du processus ambient final $E(R_o)$ (sans la continuation P_o). Par définition de la traduction, nous obtenons la configuration C_3 :

$$C_a \parallel \alpha^m h^m[D^m, E^m, E^{'n} : S^m \mid M^m \mid M^{3n}]$$

Nous remarquons que nous avons nécessairement $M^{3n} = M_n^m \mid M^{'n}\{^{e^m}/_{e^n}\} = M_n^m \mid M^{''n}$, puisque $M^{'n}$ est M^n sans les messages délégués puis réémis en M_n^m.

Nous détaillons la configuration C_a. Celle-ci se compose des mêmes locations que C'', puisque l'ambient n a été dissous. Nous avons $C_a = C'_0 \parallel C'_p \parallel C^3_q \parallel C'_s$, avec C'_0 étant C_0 dans lequel on a supprimé dans h^0 la définition `uid` i^n. Les locations de C_0 ne sont pas modifiées parce que les ambients correspondant ne le sont pas. Les locations de C_p étant celles contenant un message de la forme $r^p(e^n, \_)$, elles correspondent nécessairement à des souches dont le scion associé est dans l'ambient n. Après dissolution de cet ambient, le scion est en m et le message dans la traduction devient $r^p(e^m, \_)$; nous obtenons bien C'_p. Le même raisonnement s'applique pour C'_s. La configuration C^3_q est composé de locations de la forme $\alpha^n h^q[D^q, E^q : s^q(q, i^q, e^m, \emptyset) \mid M^{qr}]$ avec M^{qr} étant M^q après modification d'un éventuel message de la forme $r^q(e^n, \_)$ en $r^q(e^m, \_)$ (il s'agit donc de C''_q en ayant annulé les délégations).

Nous reproduisons les délégations des sous-locations de n dans les réductions initiales, et nous obtenons la configuration C_4 :

$$C'_a \parallel \alpha^m h^m[D^m, E^m, E^{'n} : S^m \mid M^m \mid M_n^m \mid M^{''n} \mid M_0^{'n}]$$

avec $M_0^{'n} = M_0^n\{^{e^m}/_{e^n}\}$ et $C'_a = C'_0 \parallel C'_p \parallel C''_q \parallel C'_s$.

Nous appliquons maintenant le lemme A.2.5 avec la configuration C_4 et le processus P_o en m. Nous obtenons un processus P' tel que (en appliquant les mêmes étapes de délégation que pour aller de C_3 à C_4) $E(R_o \mid P_o) \rightarrow_{tr} \twoheadrightarrow^*_0 P'$. Nous avons aussi :

$$C'_a \parallel \alpha^m h^m[D^m, E^m, E^{'n} : S^m \mid M^m \mid M_n^m \mid M^{''n} \mid M_0^{'n} \mid [\![P_o]\!]_{e^m}] \twoheadrightarrow_d P'$$

Nous en déduisons la réduction :

$$C'' \parallel \alpha^m h^m[D^m, E^m, E^{'n}, \kappa() \triangleright [\![P_o]\!]_{e^m} : S^m \mid M^m \mid M_n^m \mid M^{''n} \mid M_0^{'n} \mid [\![P_o]\!]_{e^m}] \twoheadrightarrow_d P''$$

avec $P'' \succeq_\kappa \succeq_\kappa P'$ en enlevant la définition `uid` i^n et la définition de κ qui ne sont plus présentes dans la configuration.

Nous concluons en accolant ces dernières relations avec une réduction $\mapsto_\kappa$:

$$C'' \parallel \alpha^m h^m[D^m, E^m, E'^n, \kappa() \triangleright [\![P_o]\!]_{e^m} : S^m \mid M^m \mid M_n^m \mid M''^n \mid M_0'^n \mid \kappa()] \mapsto_\kappa \twoheadrightarrow_d \succeq_\kappa^* P'$$

Le diagramme réciproque se prouve de la même manière, en remarquant qu'un message $o^n(\kappa)$ ne peut être présent dans la traduction que si l'ambient correspondant à la location h^n est une souche ouverte. □

Nous prouvons maintenant une relation entre l'équivalence structurelle dans le calcul des ambients étendus et la traduction étendue.

Lemme A.2.16 *Soient $Q, Q' \in \mathcal{E}$ et $P \in \mathcal{T}$ tels que $Q =_\alpha Q'$ et $[\![Q']\!]^{t\sharp} = P$. Pour tout $R \in \mathcal{E}$ tel que $Q \equiv R$, il existe un $R' \in \mathcal{E}$ tel que $R =_\alpha R'$ et $[\![R']\!]^{t\sharp} = P$, à équivalence $\equiv_{AC0}$ près.*

Preuve: Nous procédons par cas sur l'étape d'équivalence structurelle ayant lieu, en nous basant sur la dérivation $[\![Q']\!]^{t\sharp} = R$.

En ce qui concerne l'α-conversion, le résultat est immédiat en prenant $R' = Q'$.

En ce qui concerne la commutativité et l'associativité de l'opérateur parallèle, ainsi que la neutralité de $\mathbf{0}$ par rapport à cet opérateur, le résultat est immédiat à équivalence $\equiv_{AC0}$ près.

Nous étudions à présent la modification de portée des restrictions. Nous remarquons que la condition pour effectuer une extrusion ou une intrusion de portée est toujours satisfaite pour Q' puisque ses noms liés sont, par définition de la traduction, deux à deux distincts et distincts des noms libres de Q'.

Nous commençons par le cas de l'extrusion par rapport à la composition parallèle. Nous avons donc une sous-dérivation dans la traduction de la forme (avec n étant un nom d'ambient ou de scion) :

$$\frac{e, \alpha, P \rightarrow_{tr} S_P \qquad \dfrac{e, \alpha, Q \rightarrow_{tr} S_Q}{e, \alpha, Q \rightarrow_{tr} S_Q \oplus (\emptyset; \emptyset; \{n\}; \top; \top; \mathbf{0})}}{e, \alpha, P \mid (\nu n.Q) \rightarrow_{tr} S_P \oplus S_Q \oplus (\emptyset; \emptyset; \{n\}; \top; \top; \mathbf{0})}$$

et une sous-dérivation de la forme :

$$\frac{\dfrac{e, \alpha, P \rightarrow_{tr} S_P \qquad e, \alpha, Q \rightarrow_{tr} S_Q}{e, \alpha, P \mid Q \rightarrow_{tr} S_P \oplus S_Q}}{e, \alpha, \nu n.(P \mid Q) \rightarrow_{tr} S_P \oplus S_Q \oplus (\emptyset; \emptyset; \{n\}; \top; \top; \mathbf{0})}$$

Par définition de l'opérateur $\oplus$, nous passons immédiatement de l'une à l'autre.

Nous étudions maintenant le cas d'un ambient ou d'une souche. Le résultat est aussi immédiat puisque la modification de L dans une traduction est orthogonale à la traduction d'un ambient ou d'une souche. □

Afin d'établir une correspondance entre les observations entre processus ambients étendus et leurs traductions, nous prouvons également des diagrammes pour les barbes. Nous commençons par définir une notion de barbes pour les états d'approximation en utilisant le contexte $T_b(\,\cdot\,) = [p(t) \triangleright t(b) : \mathbf{0}] \parallel (\cdot)$ défini en section 3.4.2.

Définition A.2.17 *Soit la configuration $Q \in \mathcal{J}$ telle que sa location racine h^0 contienne une définition D_t^0 de la forme $s^0(a, e, i^0, \emptyset) \mid amb^0(j, b, e_b) \mid t(b) \triangleright s(a, e, i^0, \emptyset) \mid amb^0(j, b, e_b) \mid yes()$, telle que le nom t ne soit défini dans aucune autre règle ni libre dans Q excepté en D_0^t, telle que le nom yes ne soit pas libre dans Q excepté dans D_0^t, et telle que la location h^0 contienne un unique message $p(t)$ avec p n'étant pas libre dans le reste de Q.*

Nous notons $Q|_p$ la configuration Q sans le message $p(t)$, $Q|_c$ la configuration $Q|_p$ avec un message $t(c)$ en h^0, et $Q|_{yes}$ la configuration $Q|_p$ avec un message sur le nom libre yes.

Nous notons T_b^p le contexte $[p(t) \triangleright t(b) : p(t)] \parallel (\cdot)$ et T_b^b le contexte $[p(t) \triangleright t(b) : t(b)] \parallel (\cdot)$.

Lemme A.2.18 *Soit $Q \in \mathcal{J}$ une configuration satisfaisant les conditions de la définition A.2.17. Nous avons les propriétés suivantes, où à chaque fois la configuration Q' satisfait les conditions de la définition A.2.17.*

Si $T_c(Q) \rightarrow P$, alors soit $P \equiv T_c(Q')$ et $Q \rightarrow Q'$, soit $P \equiv T_c^p(Q|_p)$.

Si $T_c^p(Q|_p) \rightarrow P$, alors soit $P \equiv T_c^p(Q'|_p)$ et $Q \rightarrow Q'$, soit $P \equiv T_c^c(Q|_p)$.

Si $T_c^c(Q|_p) \rightarrow P$, alors soit $P \equiv T_c^c(Q'|_p)$ et $Q \rightarrow Q'$, soit $P \equiv T_c(Q|_c)$.

Si $T_c(Q|_c) \rightarrow P$, alors soit $P \equiv T_c(Q'|_c)$ et $Q \rightarrow Q'$, soit $P \equiv T_c(Q|_{yes})$ et $T_c(Q) \rightarrow^4 T_c(Q|_{yes}) \downarrow_{yes}$.

Preuve: Dans le cas où nous avons $Q \to Q'$, alors Q' satisfait les conditions de la définition A.2.17. En effet, la location racine ne peut pas migrer, le nom t ne peut pas être défini dans une autre règle, le nom *yes* ne peut devenir libre dans Q' puisque la seule règle le générant requiert un message sur t alors que t n'est pas libre, le nom p ne peut être défini puisqu'il est libre, et le message sur p ne peut être ni consommé, ni produit.

Dans tous les cas la location racine du contexte ne migre pas, et la location h^0 ni aucune de ses sous-locations ne peut migrer en cette location racine qui ne possède pas de nom.

Pour la première propriété, soit la réduction se fait sous le contexte $T_c(\,\cdot\,)$, et la propriété est vrai, soit une interaction se produit entre le processus Q et le contexte (aucune réduction n'est possible dans le contexte seul). Dans ce cas, la seule possibilité est le transport par règle Comm du message sur $p(t)$ dans le contexte, et nous obtenons bien $T_c^p(Q|_p)$.

En ce qui concerne la deuxième propriété, soit la réduction a lieu dans $Q|_p$, alors la même réduction peut avoir lieu dans Q et la propriété est satisfaite, soit la réduction consiste en une interaction avec le contexte (ce qui est impossible puisqu'il n'y a plus de message sur p dans $Q|_p$ et qu'il n'y a pas de définition de p dans Q), soit la réduction est une réduction du contexte, ce qui donne la configuration $T_c^c(Q|_p)$.

Pour la troisième propriété, une réduction de $Q|_p$ implique toujours une réduction identique pour Q et la propriété est satisfaite. Il ne peut y avoir de réduction dans le contexte seul. La seule interaction possible entre le contexte et $Q|_p$ consiste en le transport par une étape Comm du message $t(c)$ en h^0. Nous obtenons bien alors la configuration $T_c(Q|_c)$.

En ce qui concerne la dernière propriété, il ne peut y avoir de réduction dans le contexte seul, ni d'interaction entre $Q|_c$ et le contexte. La seule possibilité de réduction est donc dans $Q|_c$. Soit cette réduction n'implique pas le message $t(c)$, alors la même réduction peut avoir lieu dans Q et la propriété est satisfaite. Soit cette réduction implique $t(c)$, et dans ce cas la seule possibilité est une étape Join consommant le message et produisant un message sur *yes*. Nous obtenons alors bien alors la configuration $T_c(Q|_{yes})$. De plus, dans ce cas, nous remarquons que les trois premières étapes peuvent toujours avoir lieu en partant de $T_c(Q)$, donc nous avons $T_c(Q) \to^4 T_c(Q|_{yes})$. De plus, le nom *yes* n'est pas défini dans cette configuration (il n'est libre que dans la définition qui le génère et dans le message $yes()$), par conséquent nous avons bien $T_c(Q|_{yes}) \downarrow_{yes}$. □

Lemme A.2.19 *Soit une configuration $Q \in \mathcal{J}$ satisfaisant les conditions de la définition A.2.17. Nous avons alors $T_c(Q) \Downarrow_{yes}$ si et seulement si il existe un $Q' \in \mathcal{J}$ tel que $Q \to^* Q'$, $T_c(Q) \to T_c(Q')$ et $T_c(Q') \to^4 \downarrow_{yes}$.*

Preuve: Nous commençons par montrer que si nous avons $T_c(Q) \to^* P$ alors P est nécessairement de la forme (à équivalence structurelle près) $T_c(Q')$, $T_c^p(Q'|_p)$, $T_c^c(Q'|_p)$, $T_c(Q'|_c)$, ou $T_c(Q'|_{yes})$, avec $Q \to^* Q'$. Ceci est immédiat par induction sur le nombre de réductions, en appliquant le lemme A.2.18.

De plus, nous remarquons que dans le join calcul, une barbe forte ne peut disparaître. Par conséquent, il existe deux processus R et R' tels que $T_c(Q) \to^* R \to R'$ avec $R \not\downarrow_{yes}$ et $R' \downarrow_{yes}$ (le processus R existe puisque nous avons $T_c(Q) \not\downarrow_{yes}$). Le processus R' devant satisfaire une des formes précédentes, nous avons par conséquent $R' = T_c(Q'|_{yes})$ pour un Q', qui est la seule forme possédant une barbe forte sur *yes*. De plus, comme nous avons $R \to R'$ et $R \not\downarrow_{yes}$, nous avons nécessairement (par le lemme A.2.18 et la contrainte de forme précédente) $R \equiv T_c(Q'|_c)$ avec $Q \to^* Q'$, avec aucun processus intermédiaire ne définissant c (puisque celui-ci est libre dans tous les contextes de la réduction initiale).

Par conséquent, le contexte $T_c(\,\cdot\,)$ étant un contexte d'évaluation, nous avons $T_c(Q) \to^* T_c(Q')$. Nous concluons par le dernier cas du lemme A.2.18 avec la réduction $T_c(Q'|_c) \to T_c(Q'|_{yes})$ impliquant $T_c(Q') \to^4 T_c(Q'|_{yes}) \downarrow_{yes}$.

L'implication réciproque est immédiate par définition des barbes faibles et du contexte d'évaluation $T_c(\,\cdot\,)$. □

Lemme A.2.20 *Pour toute configuration $P \in \mathcal{J}$ satisfaisant les conditions de la définition A.2.17, et pour tout nom c tel que $T_c(P) \to^4 \downarrow_{yes}$, la location h^0 de P contient un message de la forme $amb^0(\_, c, \_)$.*

Preuve: Soit P une configuration, nous avons $T_c(P) \to P_1 \to P_2 \to P_3 \to P_4 \downarrow_{yes}$ par hypothèse. Le seul moyen de créer un message sur *yes* est d'utiliser la règle D_t^0, puisqu'elle contient la seule occurrence de *yes* dans toute la configuration. La règle D_t^0 ne peut être activé que si un message sur s^0 est présent, ainsi que deux messages de la forme $amb^0(\_, c, \_)$ et $t(c)$. Puisque la seule occurrence de t dans P est dans le message $p(t)$, celui-ci doit tout d'abord être consommé par la définition présente dans $T_c(\,\cdot\,)$ afin de produire le message $t(c)$. Trois étapes sont nécessaires : transporter le

message $p(t)$ dans la location de $T_c(\,\cdot\,)$ par une étape Comm, consommer ce message pour produire le message $t(c)$ par une étape Join, et transporter le message $t(c)$ en h^0 par une étape Comm. Toutes ces étapes étant nécessaires, nous avons donc $P_1 \equiv T_c^p(P|_p)$, $P_2 \equiv T_c^c(P|_p)$, $P_3 \equiv T_c(P|_c)$ et $P_4 \equiv T_c(P|_{yes})$. Puisqu'il ne peut y avoir d'autres étapes, nous avons un message $amb^0(\_, c, \_)$ en h^0. □

Lemme A.2.21 (Traduction des barbes) *Pour tout $P \in \mathcal{T}$, nous notons $P \downarrow_{[c]}$ si et seulement si $T_c(P) \rightarrow^4 \downarrow_{yes}$ avec c non défini dans P.*

Nous avons la propriété suivante : $P \downarrow_{[c]}$ si et seulement si M^0 contient un message de la forme $amb^0(\_, c, \_)$ avec $c \in F$.

Preuve: Soit P un état d'approximation. La configuration P satisfaisant la définition A.2.17, nous avons par le lemme A.2.20 un message $amb^0(\_, c, \_)$ en h^0. De plus, nous avons $c \notin L$ (puisque c n'est pas défini dans P), et $c \in F \cup L$ avec $F \cap L = \emptyset$ par les conditions d'approximation (1) et (7a), donc $c \in F$.

L'implication réciproque est prouvé en utilisant les quatre étapes de $T_c(P)$ à $T_c(P|_{yes})$. □

Lemme A.2.22 *Soient $P \in \mathcal{J}$ et $Q \in \mathcal{T}$ tels que $P \equiv \sim_\alpha \twoheadrightarrow_d Q$. Si $T_c(P) \rightarrow^4 \downarrow_{yes}$ avec c non défini dans P, alors nous avons $Q \downarrow_{[c]}$, et si $Q \downarrow_{[c]}$ alors nous avons $T_c(P) \Downarrow_{yes}$ avec c non défini dans P.*

Preuve: Nous commençons par montrer que $P \equiv \sim_\alpha \twoheadrightarrow_d Q$ implique que P satisfait les conditions de la définition A.2.17. C'est immédiatement le cas pour Q. C'est le cas après déplacement de définitions `uid` i et `fresh` a par la relation $=_d$. C'est le cas avant chaque étape Flush et Comm (le message $p(t)$ ne peut être transporté puisqu'il n'est pas défini). C'est le cas après équivalence structurelle, puisque l'α-conversion n'implique que des noms liés, et puisqu'un dépliage de définition ne peut introduire des noms déjà libres dans la configuration. Enfin, c'est le cas après un renommage global $\sim_\alpha$, puisque ce renommage est injectif, ne capture pas de nom, et ne modifie pas la structure des définitions.

Nous prouvons la première implication, en montrant que nous avons $T_c(Q) \rightarrow^4 \downarrow_{yes}$ et que c n'est pas défini dans Q. Par le lemme A.2.20, nous savons que le nom c est libre dans P puisqu'il est présent dans un message de la forme $amb^0(\_, c, \_)$. Par conséquent, la relation $\sim_\alpha$ ne peut renommer une définition de P pour qu'elle définisse c, et les dépliages de `def` ne peuvent pas non plus introduire de telles définitions. Les étapes Flush et Comm n'introduisant pas de nouvelles définitions, et l'équivalence $=_d$ ne déplaçant que des définitions existantes, le nom c n'est donc pas défini dans Q.

Nous notons P_1 et P_2 les configurations intermédiaires telles que $P \equiv \sim_\alpha =_\alpha \twoheadrightarrow P_1 \rightarrow_d^* P_2 \twoheadrightarrow_d Q$. Nous avons par conséquent $T_c(P) \equiv \sim_\alpha \equiv T_c(P_1) \rightarrow_d^* T_c(P_2) =_d T_c(Q)$, puisque les noms libres de $T_c(P)$ sont tous des noms libres de P.

Les relations $\sim_\alpha$ et $\equiv$ étant des bisimulations fortes préservant les barbes, nous avons bien $T_c(P_1) \rightarrow^4 \downarrow_{yes}$. Les étapes Flush et Comm n'impliquant pas les noms p et t commutent immédiatement avec les 4 réductions, et nous avons par conséquent $T_c(P_2) \rightarrow^4 \downarrow_{yes}$. La relation $=_d$ étant elle aussi une bisimulation forte préservant les barbes, nous avons donc $T_c(Q) \rightarrow^4 \downarrow_{yes}$. Donc, par définition de $\downarrow_{[c]}$, nous avons $Q \downarrow_{[c]}$ puisque c n'est pas défini dans Q.

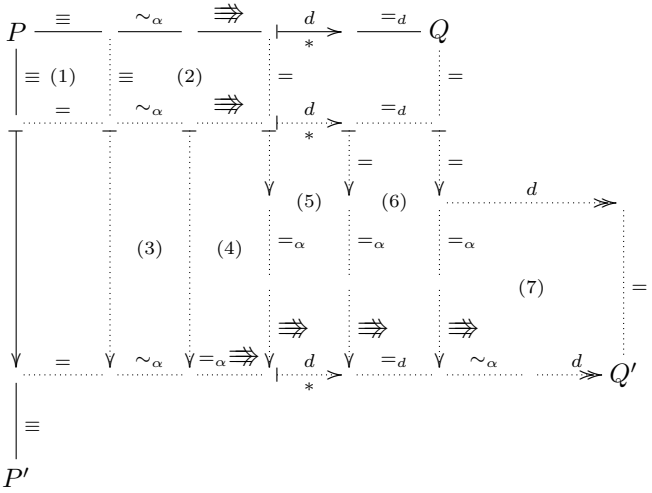
Nous prouvons maintenant que si $Q \downarrow_{[c]}$, alors $T_c(P) \Downarrow_{yes}$ et c n'est pas défini dans P. Comme précédemment, par le lemme A.2.20, nous savons que c est libre dans Q et qu'il n'est pas défini dans Q. Par conséquent, par le même raisonnement que précédemment, c n'est pas défini dans P. Utilisant les mêmes notation P_1 et P_2 que précédemment, nous avons $T_c(P_1) \rightarrow_d^* \rightarrow^4 \downarrow_{yes}$ puisque $=_d$ est une bisimulation forte préservant les barbes. Donc $T_c(P_1) \Downarrow_{yes}$. Les relations $\sim_\alpha$ et $\equiv$ étant des bisimulations fortes préservant les barbes, nous avons donc $T_c(P) \Downarrow_{yes}$. □

Lemme A.2.23 *Soient deux configurations $P \in \mathcal{J}$ et $Q \in \mathcal{T}$ telles que $P \equiv \sim_\alpha \twoheadrightarrow_d Q$.*

Si $P \rightarrow P'$ alors il existe un $Q' \in \mathcal{T}$ tel que $Q \twoheadrightarrow^= Q'$ et $P' \equiv \sim_\alpha \twoheadrightarrow_d Q'$ (nous avons $Q \twoheadrightarrow Q'$ si l'étape $P \rightarrow P'$ n'est pas une étape $\rightarrow_d$).

Si $Q \twoheadrightarrow Q'$ alors il existe un $P' \in \mathcal{J}$ tel que $P \rightarrow^+ P'$ et $P \equiv \sim_\alpha \twoheadrightarrow_d Q'$, en notant $\rightarrow^+$ une série d'étapes composée d'au moins une étape.

Preuve: Nous avons le diagramme :

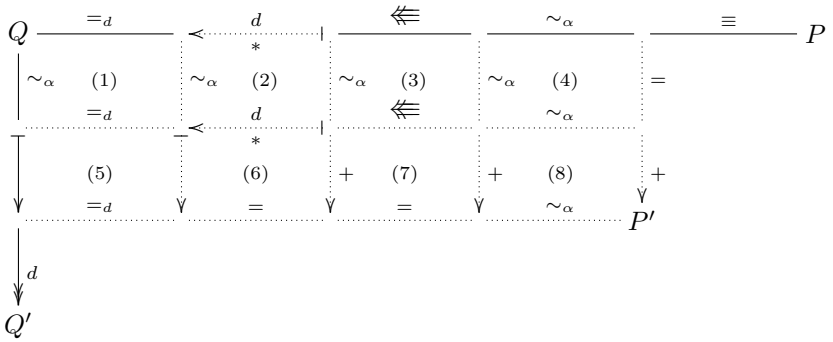


La tuile (1) est satisfaite par définition de l'équivalence structurelle. La tuile (2) correspond au lemme A.2.2(4). La tuile (3) correspond au lemme A.2.2(1). La tuile (4) utilise le lemme A.2.2(6). La tuile (5) correspond au lemme A.2.2(8) appliqué autant de fois qu'il y a de réductions $\mapsto_d$. Dans le cas où la réduction vertical $\mapsto$ est une réduction déterministe, alors cette réduction a aussi lieu dans les réduction $\mapsto_d^*$ horizontales (par la confluence de $\mapsto_d$ (lemme A.2.2(9)) et le fait que aucune étape $\mapsto_d$ n'est possible à partir de Q (lemme A.2.2(10))), et dans ce cas l'étape vertical $\mapsto$ n'a pas lieu à droite de la tuile. Dans ce cas, nous fermons cette tuile en appliquant la commutation de $=_\alpha$ avec $\mapsto_d$ et de $\Rrightarrow$ avec $\mapsto_d$ (lemme A.2.2(7)). La tuile (6) est satisfaite par les trois propriétés du lemme A.2.2(11). La tuile (7) se décompose comme suit. La réduction supérieure horizontale $\twoheadrightarrow_d$ est une conséquence du lemme A.2.7. La fermeture de cette tuile utilise le lemme A.2.2(4), en considérant la réduction vertical comme une équivalence structurelle et en utilisant la définition de $\twoheadrightarrow_d$.

Nous avons donc $P' \equiv\sim_\alpha =_\alpha \Rrightarrow \mapsto_d^* =_d \sim_\alpha \twoheadrightarrow_d Q'$. Comme la relation $\sim_\alpha$ commute immédiatement avec $=_d$ et $\mapsto_d$, et comme $\sim_\alpha$ commute également avec $\Rrightarrow$ (lemme A.2.2(5)), nous avons en fait : $P' \equiv\sim_\alpha \Rrightarrow \mapsto_d^* =_d \twoheadrightarrow_d Q'$ (nous rappelons que la relation $=_\alpha$ est incluse dans $\sim_\alpha$). Par conséquent, les propriétés du lemme A.2.2(12) nous donnent $P' \equiv\sim_\alpha \twoheadrightarrow_d Q'$.

Dans le cas où la réduction verticale initiale est une réduction déterministe, nous ne considérons pas la tuile (7). Soit Q'' tel que $Q =_\alpha Q''$ dans la partie droite de la tuile (6). Aucun dépliage ne peut s'appliquer à Q'' puisque Q est un état d'approximation. Nous avons donc $P' \equiv\sim_\alpha =_\alpha \Rrightarrow \mapsto_d^* =_d =_\alpha Q$, et nous utilisons une fois de plus le lemme A.2.2(5) pour obtenir $P' \equiv\sim_\alpha \Rrightarrow \mapsto_d^* =_d Q$, donc $P' \equiv\sim_\alpha \twoheadrightarrow_d Q$.

Nous prouvons maintenant la deuxième propriété. Nous avons le diagramme :



Les tuiles (1) et (2) sont immédiates par la commutation de $\sim_\alpha$ avec $=_d$ et $\mapsto_d$. La tuile (3) correspond au lemme A.2.2(5). La tuile (4) est immédiate. La tuile (5) correspond au lemme A.2.2(11). La tuile (6) concatène simplement les réductions $\mapsto_d^*$ avec la réduction $\mapsto$, pour former une série de réductions contenant au moins une étape. La tuile (7) est satisfaite par définition des réductions dans le join calcul. La tuile (8) utilise le lemme A.2.2(1). Nous concluons par le lemme A.2.2(12) pour transformer $P' =_d \twoheadrightarrow_d Q'$ en $P' \twoheadrightarrow_d Q'$. □

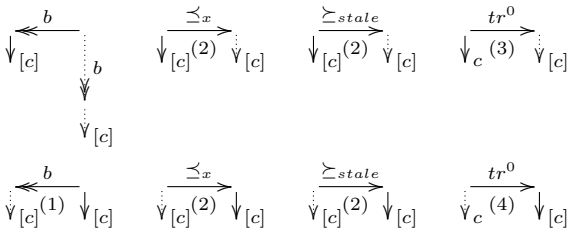
Lemme A.2.24 *Soient $P \in \mathcal{J}$ et $Q \in \mathcal{T}$ tels que $P \equiv\sim_\alpha \twoheadrightarrow_d Q$. Pour tout nom c nous avons*

$$T_c(P) \Downarrow_{yes} \text{ avec } c \notin dn(P) \iff Q \Downarrow_{[c]}$$

Preuve: Ce lemme est une conséquence directe des lemmes A.2.19, A.2.22 et A.2.23. □

Nous étudions maintenant la relation entre les barbes $\downarrow_{[c]}$ et certaines relations horizontales.

Lemme A.2.25 *Pour tout nom c, en notant $\twoheadrightarrow_b$ une réduction bookkeeping entre deux états d'approximation, et en notant $\succeq_x$ une des relations de $\{\succeq_\kappa, \succeq_{stale}, \succeq_{Fwd}\}$, nous avons :*



Preuve: La preuve de ces diagrammes se base sur le lemme A.2.21, puisque les configurations initiales (excepté dans le cas de la traduction où il s'agit d'un ambient étendu) sont des états d'approximation.

Le diagramme (1) est satisfait puisque la seule étape de bookkeeping pouvant supprimer un message sur amb^0 est une étape de propagation, qui ne peut avoir lieu puisque par les conditions d'approximation (1,7a), aucun message sur f^0 ne peut être présent. De plus, le nom c étant libre et non défini dans le processus initial, il ne peut être renommé par $\sim_\alpha$ ni se retrouver défini après dépliage d'un `def`.

Les diagrammes (2) sont satisfait puisque aucune relation de simplification ne supprime de message sur amb^0. Dans le cas de la relation $\preceq_{Fwd}$, cette propriété se base sur le fait que toutes les propagations ont déjà eu lieu.

Le diagramme (3) est satisfait puisque, par définition des barbes fortes dans le calcul des ambients étendus, il existe un ambient à la racine portant le nom c non restreint. Par définition de la traduction étendue, un message de la forme $amb^0(\_, c, \_)$ avec ce même nom c est présent (ce nom ne peut avoir été renommé puisqu'il n'est pas restreint). Les réductions $\twoheadrightarrow^0$ ne supprimant pas ce message, il est toujours présent à l'issue de la relation $\rightarrow_{tr^0}$. De plus, c n'étant pas restreint dans le processus ambient initial, nous avons nécessairement $c \in F$ dans la traduction. Nous concluons par le lemme A.2.21.

Le diagramme (4) est satisfait pour des raisons similaires. Le lemme A.2.21 nous indique la présence d'un message $amb^0(\_, c, \_)$ en M^0 avec $c \in F$. Par définition de la traduction, ce message ne peut être présent que si le processus ambient contient un ambient nommé c à la racine. De plus, comme nous avons $c \in F$, par la condition d'approximation (1), nous avons $c \notin L$ donc c n'est pas restreint.

Le dernier diagramme est immédiat. □

Nous prouvons maintenant une propriété de bisimulation entre des ambients étendus et leur traduction étendue.

Lemme A.2.26 *Soit la relation $\mathcal{H} \subseteq \mathcal{E} \times \mathcal{T}$ définie comme étant*

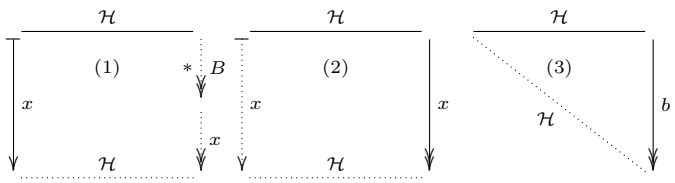
$$\mathcal{H} \stackrel{def}{=} \rightarrow_{tr^0} \left(\twoheadleftarrow_\kappa \cup \twoheadleftarrow_{Fwd} \cup \twoheadleftarrow_0 \cup \twoheadleftarrow_{Go} \cup \preceq_\kappa \cup \preceq_{Fwd} \cup \preceq_{stale} \cup \succeq_{stale} \right)^*$$

Nous avons $[\![\cdot]\!]^{t\sharp} \subseteq \mathcal{H}$.

Preuve: Ceci est immédiat puisque $\rightarrow_{tr^0} = [\![\cdot]\!]^{t\sharp} \twoheadrightarrow_0^*$, nous avons donc $[\![\cdot]\!]^{t\sharp} \subseteq \rightarrow_{tr^0} \twoheadleftarrow_0^* \subseteq \mathcal{H}$. □

Nous prouvons maintenant le cœur de notre bisimulation sous la forme d'une bisimulation forte hybride barbue *up to* bookkeeping. Ce cœur sera étendu de chaque côté pour prendre en compte tous les processus ambients étendus, et non pas seulement ceux dans la relation $\rightarrow_{tr^0}$, et tous les processus join.

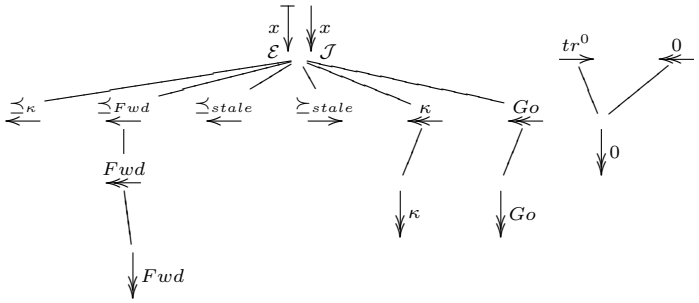
Théorème 8 *La relation $\mathcal{H}$ est telle que $P \; \mathcal{H} \; Q$ implique $P \Downarrow_c$ si et seulement si $Q \Downarrow_{[c]}$, et telle que les diagrammes suivants soit satisfaits pour tout $\twoheadrightarrow_x \in X$ et $\twoheadrightarrow_b \in B$:*



Preuve: Afin de fermer chacun des diagrammes ci-dessus et afin d'établir la correspondance entre les barbes, nous accolons de manière systématique les diagrammes élémentaires des lemmes A.2.11, A.2.12, A.2.13, A.2.14, A.2.15, et A.2.25. La plupart de ces diagrammes introduisant des relations horizontales et verticales supplémentaires, il n'est pas évident que le recouvrement soit fini. Pour s'assurer de l'existence d'un recouvrement fini par ces diagrammes élémentaires, nous utilisons la technique des *diagrammes décroissants* introduite par Van Oostrom pour réaliser des preuves de confluence [Oos94] :

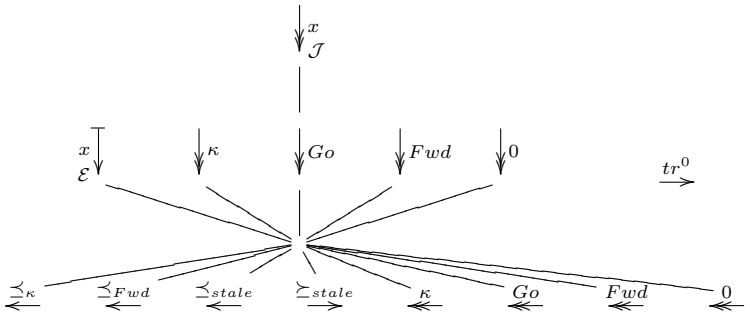
1. Nous recensons toutes les relations "horizontales" et "verticales" utilisées, et choisissons un ordre partiel strict sur ces relation pour chacun des diagrammes du théorème.
2. Nous décrivons un diagramme élémentaire pour chaque paire de relation horizontale et verticale, et nous assurons que ce diagramme *décroît* selon l'ordre choisi (voir [Oos94, définition 3.3]).
3. Nous concluons que pour toute paire de relations obtenue en composant d'une part les relations horizontales et d'autre part les relations verticales, il existe un diagramme composé d'un recouvrement fini des diagrammes élémentaires (voir [Oos94, Lemmes 3.5 et 3.6]).

Pour obtenir le diagramme (1) pour une relation $\mapsto_x$ donnée, nous utilisons l'ordre partiel strict suivant sur les relations, tel qu'il est définit par ce diagramme de Hasse :



Nous conclurons en utilisant l'accolage correspondant à la relation $\mathcal{H}$ en ce qui concerne la relation horizontale, et les seules relations $\mapsto_x$ pour les ambients étendus et $\twoheadrightarrow_x$ pour les états d'approximation en ce qui concerne la relation verticale. La preuve de la propriété sur les barbes (de gauche à droite) se fait en étendant l'ordre précédant avec l'union de $\downarrow_c$ pour les ambients étendus et $\downarrow_{[c]}$ pour les états d'approximation de manière incomparable aux autres relations, et en considérant les diagrammes élémentaires du lemme A.2.25 (nous considérons le prédicat de présence de barbe comme une relation entre le processus et un ensemble singleton). Puisque le lemme A.2.25 montre que $P \downarrow_c$ implique $Q \Downarrow_{[c]}$ si $P \mathcal{H} Q$, nous concluons en utilisant la propriété de bisimulation pour obtenir $P \Downarrow_c$ implique $Q \Downarrow_{[c]}$.

Pour obtenir le diagramme (2) pour toute relation $\twoheadrightarrow_x \in X$ et le diagramme (3), nous utilisons l'ordre partiel :



Comme précédemment, nous étendons cet ordre partiel avec une relation incomparable pour les barbes correspondant à l'union des relations $\downarrow_c$ et $\downarrow_{[c]}$ afin d'obtenir la relation sur les barbes de droite à gauche. Nous nous basons pour ce faire sur le fait que le lemme A.2.25 implique que si $P \mathcal{H} Q$ alors $Q \downarrow_{[c]}$ implique $P \downarrow_c$.

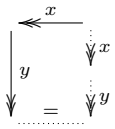
Nous vérifions maintenant que les diagrammes élémentaires nécessaires existent tous et sont bien décroissants pour ces ordres partiels. Le tableau ci-après décrit le produit cartésien des relations verticales et horizontales. Pour chaque paire de relations, nous devons prouver l'existence de deux diagrammes décroissants : le premier lorsque la relation verticale a lieu à gauche de la relation

horizontale en utilisant le premier ordre partiel, le deuxième lorsque la relation verticale a lieu à droite de la relation horizontale, en utilisant le deuxième ordre partiel. Nous indiquons donc pour chaque case le lemme démontrant l'existence du diagramme élémentaire, et notons $\emptyset$ lorsque les relations ne peuvent s'appliquer (par exemple une réduction join pour un processus ambient). Nous vérifions que chaque lemme clôt le diagramme en vérifiant la propriété des diagrammes décroissants.

	$\xrightarrow{tr^0}$	$\xleftarrow{\preceq_\kappa}$	$\xleftarrow{\preceq_{Fwd}}$	$\xleftarrow{\preceq_{stale}}$, $\xrightarrow{\succeq_{stale}}$	$\xleftarrow{B}$
$\downarrow^{B,X}_{\mathcal{J}}$	$\emptyset$ A.2.15 **(1)**	A.2.12	A.2.13	A.2.11	**(2)** A.2.14
$\downarrow^{X}_{\mathcal{E}}$	A.2.15 $\emptyset$	$\emptyset$			
$\downarrow_{a,[a]}$	A.2.25				

Les diagrammes élémentaires de (1) correspondent aux réductions bookkeeping. Par définition de la traduction, seule une étape 0 peut avoir lieu après la relation $\rightarrow_{tr^0}$, et le diagramme élémentaire correspondant est immédiat puisque nous avons $\rightarrow_{tr^0} \twoheadrightarrow_0 = \rightarrow_{tr^0}$.

Les diagrammes élémentaires de (2) sont les diagrammes triviaux, en prenant $\twoheadrightarrow_x \in B$ et $\twoheadrightarrow_y \in B \cup X$:

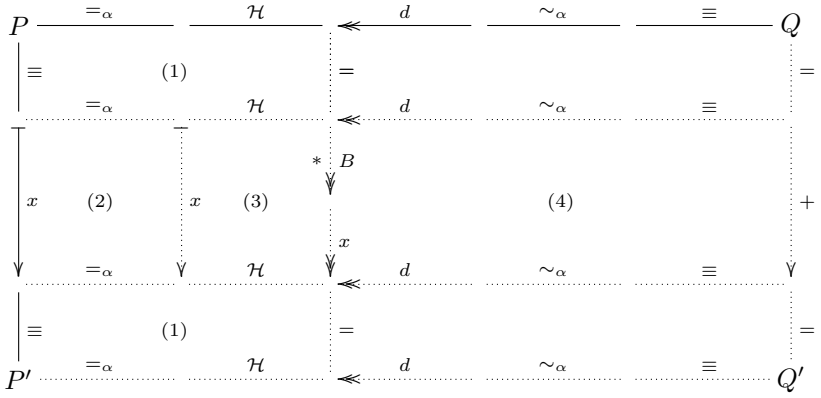


Ces diagrammes sont bien décroissants pour le premier ordre partiel.

□

Preuve du théorème 3: Afin de prouver que $\approx^{aj}$ est une bisimulation faible, nous considérons la relation candidate $\mathcal{R} \stackrel{\text{def}}{=} =_\alpha \mathcal{H} \twoheadleftarrow_d \sim_\alpha \equiv$.

Nous montrons tout d'abord la première propriété de simulation par le diagramme :

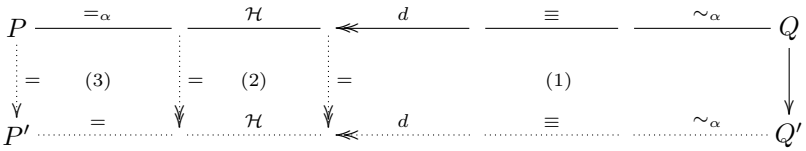


Les diagrammes (1) se prouvent de la manière suivante. Soient $P \in \mathcal{E}$, $P' \in \mathcal{T}$ et $Q \in \mathcal{E}$ tels que $P =_\alpha \mathcal{H}\ P'$ et $P \equiv Q$. Par définition de $\mathcal{H}$, il existe $P_1 \in \mathcal{E}$ et $P_2 \in \mathcal{T}$ tels que :

$$P =_\alpha P_1 \xrightarrow{[\![\cdot]\!]^{t\sharp}} P_2 \twoheadrightarrow_0^* (\twoheadleftarrow_\kappa \cup \twoheadleftarrow_{Fwd} \cup \twoheadleftarrow_0 \cup \twoheadleftarrow_{Go} \cup \preceq_\kappa \cup \preceq_{Fwd} \cup \preceq_{stale} \cup \succeq_{stale})^* P'$$

Nous appliquons le lemme A.2.16, et nous avons par conséquent $Q =_\alpha \xrightarrow{[\![\cdot]\!]^{t\sharp}} P_2$. Le diagramme (2) est une conséquence de la définition des réductions dans le calcul des ambients étendus. En effet, aucune étape ne peut être invalidée après une α-conversion (nous rappelons que les contextes d'évaluation pour les étapes étiquetées Stub et Scion ne possèdent pas de restriction autour de leurs trous). Le diagramme (3) correspond au diagramme (1) du théorème 8. Le diagramme (4) correspond au lemme A.2.23.

Nous montrons maintenant la propriété de simulation réciproque par le diagramme :



Le diagramme (1) correspond au lemme A.2.23. La réduction verticale n'a lieu que si la réduction initiale n'est pas une étape Comm ou Flush. Le diagramme (2) correspond aux diagramme (2) ou (3) du théorème 8. La réduction verticale n'a lieu que si l'étape initiale n'est pas une étape de bookkeeping. Le diagramme (3) est immédiat par définition des réductions dans le calcul des ambients étendus (en reproduisant l'α-conversion, puis l'étape).

Nous montrons maintenant la préservation des barbes faibles : nous avons $P \Downarrow_c$ si et seulement si $c \notin dn(Q)$ et $T_c(Q) \Downarrow_{yes}$. Ces barbes sont préservées par α-conversion dans le calcul étendu et par la relation $\mathcal{H}$ (théorème 8). Elles sont préservés par la relation $\equiv \sim_\alpha \twoheadrightarrow_d$ (avec la condition $c \notin dn(Q)$) par le lemme A.2.24.

Pour conclure, nous montrons que nous avons $[\![\cdot]\!]^t \subseteq \mathcal{R}$. Soient $P \in \mathcal{A}$ et $Q \in \mathcal{J}$ tels que $[\![P]\!]^t = Q$, nous montrons que nous avons $P \ \mathcal{R} \ Q$. Par le lemme A.2.6, il existe un $P' \in \mathcal{E}$ et un $T \in \mathcal{T}$ tels que $P =_\alpha P'$, $[\![P']\!]^{t\sharp} = T$ et $[\![P]\!]^t = Q \twoheadrightarrow_d T$. Nous avons donc $P =_\alpha \xrightarrow{[\![\cdot]\!]^{t\sharp}} T \twoheadleftarrow_d Q = [\![P]\!]^t$. Nous concluons en remarquant que, par le lemme A.2.26, nous avons $[\![\cdot]\!]^{t\sharp} \subseteq \mathcal{H}$. □

Puisqu'une relation de bisimulation faible ne préserve pas nécessairement la divergence, nous devons maintenant prouver que c'est quand même le cas pour notre relation.

Nous remarquons que la preuve précédente nous garantit la conservation de la divergence des ambients étendus vers les processus join. En ce qui concerne l'autre direction, deux types de réductions peuvent se produire qui ne se reflètent pas en une réduction du calcul des ambient : une étape déterministe (Comm ou Flush) ou une étape de bookkeeping. En ce qui concerne les premières, le lemme A.2.2(8) nous garantit qu'il ne peut y avoir une série infinie d'étapes déterministes. Nous prouvons maintenant qu'il n'est pas non plus possible d'avoir une série infinie d'étapes de bookkeeping.

Lemme A.2.27 *Les réductions $\twoheadrightarrow_B$ entre états d'approximation normalisent fortement.*

Preuve: Nous prouvons que tout série de réductions de la forme $P_1 \twoheadrightarrow_B P_2 \twoheadrightarrow_B P_3 \ldots$ est finie.

Par définition de $\twoheadrightarrow$, nous avons $P_i \in \mathcal{T}$ pour tout i. Nous définissons la profondeur d'un message sur un nom k défini en D^n comme étant la longueur de la chaîne β si $\alpha^n h^n = \alpha^p \beta$ où h^p est la location englobante la plus proche qui est vivante (comme défini par la condition d'approximation (7a)). Par exemple, tout message sur un nom de D^n si h^n est vivant a une profondeur de 0. Utilisant la partition B, nous notons g, κ et o pour le nombre d'étapes `go`, d'étapes de continuation et d'étapes 0 pouvant immédiatement avoir lieu. Nous écrivons f_0 pour la somme des profondeurs des messages sur *in* et *out*. Nous notons f_1 pour la somme des profondeurs des messages sur *open*, *amb*, *recv*, *send*, sub_{in} et sub_{out}. Tous les messages pouvant être propagés sont ainsi pris en compte.

Nous considérons les quintuplets (g, κ, f_0, o , f_1) ordonnés par l'ordre lexicographique (qui est bien fondé). Nous associons un quintuplet à chaque P_i et montrons que chaque étape de bookkeeping produit un quintuplet strictement plus petit que le quintuplet précédent. Nous en déduisons que la série d'étapes de bookkeeping est nécessairement finie.

Les étapes de propagation des messages de f_1 font décroître la profondeur de ces messages, puisque la location les définissant est nécessairement ouverte (donc leur profondeur est au moins égale à 1) et qu'ils sont propagés dans la définition englobante. Cette propagation ne rend possible aucune étape de bookkeeping hormis d'autres propagations (à part celles-ci, elle ne rend possible que des étapes 1 ou des étapes Recv). Similairement, la propagation des messages sur *in* et *out* fait strictement décroître f_0, mais cette propagation peut rendre possible de nouvelles étapes 0, faisant croître o. Cependant, le quintuplet final reste strictement plus petit (g et κ ne sont pas modifiés).

Les étapes 0 consomment des messages sur *in* ou *out* qui étaient à profondeur 0 (puisque ces étapes ne peuvent avoir lieu que dans les locations vivantes), donc ne font pas décroître f_0. Par contre, ces étapes font décroître le nombre d'étapes 0 possible, donc o. Les messages générés sur sub_{in} ou sub_{out} peuvent faire croître f_1, mais le quintuplet résultant est toujours strictement plus petit (g et κ ne sont pas modifiés).

Par définition de la traduction, les étapes de continuation ne créent aucun processus de migration `go` en contexte d'évaluation, ni ne rendent possible aucune étape de continuation (nous

rappelons qu'une étape consommant un message $\kappa()$ dans le cas d'un processus répliqué n'est pas une étape de continuation mais une étape de réplication). Nous obtenons donc un quintuplet strictement plus petit.

Enfin, les étapes de migration ne rendent pas possible de nouvelles étapes de migration. □

Lemme A.2.28 *La relation $\mathcal{R}$ préserve la divergence.*

Preuve: Soient $P \in \mathcal{E}$ et $Q \in \mathcal{J}$ tels que $P \mathcal{R} Q$. Nous montrons tout d'abord que si $P \Uparrow$ alors nous avons $Q \Uparrow$. C'est le cas puisque la divergence est préservée comme indiquée dans le premier diagramme de simulation de la preuve du théorème 3 (ceci est également une conséquence directe de la préservation de la divergence à travers l'α-conversion dans les ambients étendus, à travers la relation $\mathcal{H}$ (diagramme (1) du théorème 8), et à travers la relation $\twoheadleftarrow_d \sim_\alpha \equiv$ (lemme A.2.23)).

Nous supposons maintenant que $Q \Uparrow$ et nous montrons que nous avons $P \Uparrow$. Par définition de la relation $\mathcal{R}$, nous avons un Q' tel que $Q \equiv \sim_\alpha \twoheadrightarrow_d Q'$. Nous utilisons le lemme A.2.23 pour construire une série d'étapes infinie $Q' \twoheadrightarrow^*$. Cette série est infinie puisque à une étape de Q correspond une étape de Q' sauf si cette étape est déterministe, et il ne peut y avoir de série infinie d'étapes déterministes (lemme A.2.2(8)). Par le lemme A.2.27, il ne peut y avoir une série infinie d'étapes de bookkeeping $\twoheadrightarrow_B$, par conséquent il existe dans cette série une infinité d'étapes $\twoheadrightarrow_X$. Puisque le diagramme (2) du théorème 8 préserve le nombre d'étapes dans ce cas, nous avons une infinité d'étapes dans le calcul des ambients. Nous concluons en remarquant que l'α-conversion dans les ambients préserve la divergence, et nous obtenons $P \Uparrow$. □

Nous prouvons finalement la correction de notre démarche comme définie en section 3.2.

Preuve du théorème 1: Soit P un processus ambient. En ce qui concerne les barbes faibles et fair-must, nous composons les résultats du théorème 2 ($P \lesssim P$ avec des sémantiques différentes) avec le théorème 3 ($P \approx^{aj} [\![P]\!]^t$).

Nous commençons par un résultat général sur les barbes fair-must. Soit $\lesssim$ une paire de simulations couplées entre deux familles de processus équipés respectivement des réductions $\rightarrow$ et $\rightarrow'$, ainsi que des barbes $\downarrow_b$ et $\downarrow'_b$. Nous notons $\Downarrow'_b$ pour $\rightarrow'\downarrow'_b$. Si $\lesssim$ préserve les barbes faibles, alors $\lesssim$ préserve les barbes fair-must.

Supposons que nous ayons $P \lesssim Q$, $\Box P \Downarrow_b$ et $Q \rightarrow'^* Q'$. Nous avons alors $P \rightarrow^* P' \gtrsim Q'$ puisque $\gtrsim$ est une simulation à droite, et nous avons $P' \rightarrow^* P'' \lesssim Q'$ par couplage. Par définition de $\Box P \Downarrow_b$, nous avons donc $P'' \Downarrow_b$, donc nous avons $Q' \Downarrow'_b$ par préservation des barbes de $\lesssim$, donc $Q \Downarrow'_b$. Nous en déduisons $Q\Box \Downarrow'_b$ pour tout nom b.

Par le théorème 3, nous avons $P \Downarrow_b$ si et seulement si $b \notin dn([\![P]\!]^t)$ et $T_b([\![P]\!]^t) \Downarrow_{yes}$. Comme les noms de $dn([\![P]\!]^t)$ sont uniquement des noms de la traduction (plus précisément les noms de la traduction à la racine) qui sont distincts des noms d'ambients, nous avons donc $P \Downarrow_b$ si et seulement si $T_b([\![P]\!]^t) \Downarrow_{yes}$. Nous montrons maintenant que $\Box P \Downarrow_b$ si et seulement si $\Box T_b([\![P]\!]^t) \Downarrow_{yes}$. Nous supposons que nous avons $\Box P \Downarrow_b$. Nous considérons une série de réductions $T_b([\![P]\!]^t) \rightarrow^* Q$. Nous voulons montrer que nous avons $Q \Downarrow_{yes}$ Nous montrons de manière immédiate par induction sur la longueur de la série de réductions que $b \in fn(Q)$ (dans la définition $p(t) \triangleright t(b)$ qui ne peut disparaître) et que $b \notin dn(Q)$ (il ne peut apparaître après dépliage d'un `def` puisqu'il est libre) en utilisant le lemme A.2.18. Si nous avons $Q \downarrow_{yes}$, nous avons immédiatement $Q \Downarrow_{yes}$. Sinon, nous prouvons par induction sur la longueur de la série de réductions que l'un des cas suivants est vrai (avec à chaque fois Q' satisfaisant les conditions de la définition A.2.17 et $[\![P]\!]^t \rightarrow^* Q'$)

- $Q = T_b(Q')$;
- $Q = T_b^p(Q'|_p)$;
- $Q = T_b^b(Q'|_p)$;
- $Q = T_b(Q'|_b)$.

La preuve est immédiate en appliquant l'hypothèse d'induction puis le lemme A.2.18 pour chaque cas, sachant que nous avons $Q \not\downarrow_{yes}$.

Par le théorème 3, il existe donc un $P' \in \mathcal{E}$ tel que $P \rightarrow^* P'$ et $P' \mathcal{R} Q'$. Par définition des barbes fair-must, nous avons $P' \Downarrow_b$, donc par préservation des barbes faibles, nous avons $T_b(Q') \Downarrow_{yes}$. Nous montrons maintenant que si $T_b(Q') \Downarrow_{yes}$, alors $T_b^p(Q'|_p) \Downarrow_{yes}$, $T_b^b(Q'|_p) \Downarrow_{yes}$ et $T_b(Q'|_b) \Downarrow_{yes}$. Pour se faire, nous montrons en fait que pour tout Q' satisfaisant les conditions de la définition A.2.17, si une des formes $T_b(Q')$, $T_b^p(Q'|_p)$, $T_b^b(Q'|_p)$ ou $T_b(Q'|_b)$ exhibe une barbe faible sur yes, alors toutes les formes exhibent une barbe faible sur yes. Ceci est immédiat par induction sur la longueur de la dérivation et par cas sur la forme du processus initial, en appliquant à chaque fois le lemme A.2.18 (le cas de base est pour une réduction de taille 1 de la forme $T_b(Q'|_b) \rightarrow T_b(Q'|_{yes}) \downarrow_{yes}$ qui est le seul cas de taille 1 exhibant une barbe faible sur yes).

Nous prouvons maintenant l'implication réciproque pour les barbes fair-must. Nous supposons que nous avons $\Box T_b(\llbracket P \rrbracket^t) \Downarrow_{yes}$. Nous considérons une série de réductions $P \to^* P'$. Par le théorème 3, nous avons $\llbracket P \rrbracket^t \to Q'$ avec $P' \mathcal{R} Q'$.

Nous montrons maintenant une propriété générale du join calcul : soient $J_1, J_2 \in \mathcal{J}$ tel que $J_1 \to J_2$, alors $fn(J_1) \setminus dn(J_1) \supseteq fn(J_2) \setminus dn(J_2)$. C'est immédiatement le cas pour les règles Comm et Go, sans équivalence structurelle, et pour les règles d'équivalence structurelle P0-P2, D0-D2 et Tree. En ce qui concerne la règle Join, c'est le cas puisque l'ensemble des noms définis ne change pas, alors que l'ensemble des noms libres diminue (on a $fn(Q\sigma) \subseteq fn(Q) \cup fn(J\sigma)$). En ce qui concerne la règle d'équivalence structurelle Def, c'est le cas le plus compliqué. Lors d'un dépliage, les nouveaux noms définis ne peuvent être libres dans le reste de la configuration par hypothèse, et les noms libres non définis de `def` D' `in` P' restent libres et non définis. Lors d'un repliage, les noms qui ne sont plus définis à cause du repliage ne peuvent être libres dans le reste de la configuration. La règle Def ne modifie donc pas l'ensemble des noms libres non définis.

Nous pouvons étendre cette propriété à une série de réductions (par une induction immédiate sur la longueur de la série). Nous montrons maintenant que pour tout Q_i intermédiaire de la réduction $\llbracket P \rrbracket^t \to^* Q'$, nous avons $b \in fn(Q_i) \setminus dn(Q_i)$, Q_i satisfait les conditions de la définition A.2.17, et $T_b(\llbracket P \rrbracket^t) \to^* T_b(Q_i)$.

En ce qui concerne le processus initial, nous devons seulement prouver que $b \in fn(\llbracket P \rrbracket^t) \setminus dn(\llbracket P \rrbracket^t)$. Par hypothèse, nous avons $T_b(\llbracket P \rrbracket^t) \Downarrow_{yes}$, donc par les lemmes A.2.19 et A.2.20, nous avons $\llbracket P \rrbracket^t \to^* Q_f$, $T_b(\llbracket P \rrbracket^t) \to^* T_b(Q_f) \to^4 \downarrow_{yes}$ avec b libre dans Q_f dans un message $amb^0(\_, b, \_)$. Puisque b est libre et non défini dans $T_b(\llbracket P \rrbracket^t)$ et qu'il reste libre dans à chaque étape intermédiaire (dans le contexte), il n'est donc pas défini dans $T_b(Q_f)$. Par conséquent il est libre et non défini dans Q_f, donc il est libre et non défini dans $\llbracket P \rrbracket^t$.

Nous considérons maintenant une étape d'induction. Nous procédons par cas selon la réduction. Nous considérons tout d'abord l'équivalence structurelle. Ce cas est immédiat puisque celle-ci ne modifie pas l'ensemble des noms libres non définis. Nous considérons maintenant une étape Comm ou Go sans équivalence structurelle. Ce cas est aussi immédiat puisque l'ensemble des nom libres non définis n'est pas modifié, donc nous pouvons appliquer le contexte $T_b(\,\cdot\,)$ sans avoir une capture de b et effectuer la même étape. Nous concluons par le lemme A.2.18. Dans le cas d'une étape Join, l'ensemble des noms libres non définis peut décroître. Supposons que nous ayons une étape $Q_i \mapsto Q'_i$, et montrons que nécessairement nous avons b nom libre non défini de Q'_i (nous concluons alors comme auparavant). Si ce n'est pas le cas, nous avons par récurrence $T_b(\llbracket P \rrbracket^t) \to^* T_b(Q_i)$. Une étape Join ne pouvant pas introduire de définitions, elle ne peut que supprimer des noms libres pour faire décroître l'ensemble des noms libres non définis. Par conséquent, b n'est pas défini dans Q'_i et nous pouvons appliquer le contexte $T_b(\,\cdot\,)$ sans capturer b. Nous avons donc $T_b(\llbracket P \rrbracket^t) \to^* T_b(Q'_i)$. Par définition des barbes fair-must, nous avons $T_b(Q'_i) \Downarrow_{yes}$. Comme pour le cas initial, nous en déduisons que b doit être libre et non défini dans Q'_i, ce qui est impossible puisque nous avons supposé qu'il ne l'était pas. Donc b est libre et non défini dans Q'_i, et nous concluons par le lemme A.2.18.

Par conséquent, nous avons une réduction $T_b(\llbracket P \rrbracket^t) \to^* T_b(Q')$ avec $b \notin dn(Q')$. Par définition des barbes fair-must, nous avons $T_b(Q') \Downarrow_{yes}$, et comme $b \notin dn(Q')$, par le théorème 3, nous avons $P' \Downarrow_b$.

Puisque le théorème 3 utilise une notion de barbe différente du théorème 1, nous montrons maintenant que nous avons $T_b(\llbracket P \rrbracket^t) \Downarrow_{yes}$ si et seulement si $\llbracket P \rrbracket^b \Downarrow_{yes}$. C'est le cas puisque ces deux configurations ont bien b comme nom libre, nous avons $\llbracket P \rrbracket^b \equiv \llbracket P \rrbracket^t|_b$ selon la notation de la définition A.2.17, et puisque les trois étapes $T_b(Q) \to T_b^p(Q|_p) \to T_b^b(Q|_p) \to T_b(Q|_b)$ peuvent toujours être accomplies.

Nous établissons maintenant la préservation de la divergence. Celle-ci est préservée entre $(P, \to)$ et $(P, \to_{12C})$ puisque toutes les étapes correspondent soit exactement à une étape, soit à une paire d'étapes. Elle est également préservée entre $(P, \to_{12C})$ et $(\llbracket P \rrbracket^t, \to)$ par le lemme A.2.28. Enfin, elle est préservée entre $\llbracket P \rrbracket^t$ et $\llbracket P \rrbracket^b$ puisque la seule différence consiste en une étape supplémentaire dans la deuxième configuration consommant le message $t(b)$ pour produire le message $yes()$. □

Annexe B

Preuves du chapitre 4

B.1 Preuves de la sûreté du typage

Dans ce chapitre, nous ne considérons que des dérivations de typage finies.

Définition B.1.1 *Soit n un nom. Nous disons que :*
- *n est un canal* statique *dans Γ si $n : \forall\widetilde{\alpha}\widetilde{\delta}.\langle\tau\rangle \in \Gamma$ pour des $\widetilde{\alpha}$, $\widetilde{\delta}$, et τ ;*
- *n est un canal* redéfinissable *dans Γ si $n : \langle\tau\rangle^+_w \in \Gamma$ pour des τ et w ;*
- *n est une* location *dans Γ si $n : loc(\mathbf{\Delta}) \in \Gamma$ pour un $\mathbf{\Delta}$.*

Nous remarquons qu'un nom ne peut à la fois être un canal statique, une location ou un canal redéfinissable dans un Γ donné, puisque n ne possède qu'une association dans Γ. Nous utilisons cette propriété de nombreuses fois dans les preuves suivantes.

Dans la suite, nous disons souvent que n est statique dans Γ si n est un canal statique dans Γ, et que n est redéfinissable dans Γ si n est un canal redéfinissable dans Γ.

Lemme B.1.2 *Soit Γ un environnement de typage et n un nom. Nous avons :*

1. *si il existe un τ tel que $\Gamma \vdash n : \langle\tau\rangle$ alors n est statique dans Γ ;*
2. *si il existe un τ et un w tel que $\Gamma \vdash n : \langle\tau\rangle^+_w$ alors n est redéfinissable dans Γ ;*
3. *si il existe un $\mathbf{\Delta}$ tel que $\Gamma \vdash n : loc(\mathbf{\Delta})$ alors n est une location dans Γ.*

Preuve: Nous procédons par induction sur la dérivation de typage pour le nom n.

Le cas de base est l'utilisation de la règle Name. Le résultat est immédiat pour les propriétés (2) et (3), puisque leur type est nécessairement monomorphe dans Γ. En ce qui concerne la propriété (1), la liaison polymorphe dans Γ pour n est nécessairement de la forme $n : \forall\widetilde{\alpha}\widetilde{\delta}.\langle\tau'\rangle$ par définition de l'instanciation (en effet, si $\langle\tau'\rangle_{\mathbf{\Delta}'}$ est une instance de $\forall\widetilde{\alpha}\widetilde{\delta}.\langle\tau\rangle_{\mathbf{\Delta}}$, alors $\mathbf{\Delta}$ et $\mathbf{\Delta}'$ ont la même taille). Donc n est statique dans Γ.

Le cas inductif ne peut se produire qu'avec la règle Sub. Nous procédons alors par cas selon la forme du type à droite de la relation de sous-typage, c'est à dire le type de n dans la conclusion.

$\langle\tau\rangle^+_w$ Aucune règle de sous-typage ne pouvant s'appliquer, le type à gauche de la relation de sous-typage est nécessairement $\langle\tau\rangle^+_w$, c'est à dire le même type. Nous concluons par induction que n est redéfinissable dans Γ (propriété (2)).

$\langle\tau\rangle_{\mathbf{\Delta}}$ Ce cas ne peut avoir lieu si $\mathbf{\Delta}$ est non vide, puisque nous ne considérons pas les types de cette forme. Par contre, ce cas peut avoir lieu si le type est de la forme $\langle\tau\rangle_\emptyset$. Dans ce cas, nous prouvons immédiatement par induction que si $\tau'' \leq \langle\tau\rangle_\emptyset$ alors τ'' est de la forme $\langle\tau'\rangle_\emptyset$. En effet, la seule règle de sous-typage pouvant s'appliquer est N-Sub (c'est le cas de base), et les cas inductifs (c'est à dire transitivité et réflexivité) sont immédiats.

Nous concluons par induction que n est statique dans Γ (propriété (1)).

$loc(\mathbf{\Delta})$ Nous montrons comme dans le cas précédent que par induction sur la relation de sous-typage, tous les sous-types de $loc(\mathbf{\Delta})$ sont nécessairement de la forme $loc(\mathbf{\Delta}')$. Nous concluons par induction que n est une location dans Γ (propriété (3)).

□

Nous prouvons quelques propriétés sur les dérivations de typage.

Lemme B.1.3 *Soit* $\mathbf{\Delta};\mathbf{I};\Gamma \vdash \mathcal{D} :: B;\Delta_1;\Theta$ *la conclusion d'une dérivation de typage. Nous avons les propriétés suivantes :*

1. *si* $n \in dsn(\mathcal{D})$ *alors* n *est soit une location soit un canal statique dans* B *et* Γ *;*
2. *si* $n \in ddn(\mathcal{D})$*, alors* n *est redéfinissable dans* Γ *;*
3. $dom(B) = dsn(\mathcal{D})$ *;*
4. $dom(B) \cup ddn(\mathcal{D}) = dn(D)$ *;*
5. $dln(\mathcal{D}) \cap ddn(\mathcal{D}) = \Delta_1$ *;*
6. $\Delta_1 \subseteq dln(\mathcal{D})$.

Preuve: Nous remarquons tout d'abord que la règle de typage Join étend Γ uniquement avec des types bien formés.

La preuve se fait en parallèle pour toutes les propriétés excepté (6) qui est une conséquence immédiate de (5). Nous procédons par induction sur la dérivation de typage, en n'utilisant pas la propriété (6).

Join La propriété (1) est satisfaite puisque n est un $\mathtt{x}_i$ du filtre qui est statique dans B. Le typage de $\mathtt{x}_i$ dans les hypothèses de la règle Join nous indique par le lemme B.1.2(1) que n est statique dans Γ.

La propriété (2) est satisfaite puisque n est un m_j dans le filtre. Le typage de m_j dans les hypothèses de la règle Join nous indique par lemme B.1.2(2) que n est redéfinissable dans Γ.

Les propriétés (3), (4), et (5) sont immédiates par définition des ensembles de noms.

Top Toutes les propriétés sont immédiates.

And La propriété (1) est immédiate par induction, puisque $dsn(\mathcal{D}_1, \mathcal{D}_2) = dsn(\mathcal{D}_1) \cup dsn(\mathcal{D}_2)$.

Les propriétés (2), (3), et (4) sont aussi immédiates par induction.

Pour prouver la propriété (5), nous considérons l'ensemble suivant :

$$\Delta_1 \cup (dln(\mathcal{D}_1) \cap ddn(\mathcal{D}_2))$$

Nous avons :

$$dln(\mathcal{D}_1) \subseteq dn(\mathcal{D}_1)$$

donc :

$$dln(\mathcal{D}_1) \subseteq dln(\mathcal{D}_1) \cap dn(\mathcal{D}_1)$$

Par induction sur les propriétés (3,4), nous avons :

$$dln(\mathcal{D}_1) \subseteq dln(\mathcal{D}_1) \cap (ddn(\mathcal{D}_1) \cup dsn(\mathcal{D}_1))$$

Donc par induction sur les propriétés (1,2), nous avons (puisque $dsn(\mathcal{D}_1) \cap ddn(\mathcal{D}_2) = \emptyset$, les noms de $dsn(\mathcal{D}_1)$ étant statiques dans Γ et ceux de $ddn(\mathcal{D}_2)$ étant redéfinissables dans Γ) :

$$dln(\mathcal{D}_1) \cap ddn(\mathcal{D}_2) \subseteq dln(\mathcal{D}_1) \cap ddn(\mathcal{D}_1) \cap ddn(\mathcal{D}_2) \subseteq dln(\mathcal{D}_1) \cap ddn(\mathcal{D}_1)$$

Par induction sur la propriété (5), nous avons :

$$dln(\mathcal{D}_1) \cap ddn(\mathcal{D}_2) \subseteq \Delta_1$$

donc :

$$\Delta_1 \cup (dln(\mathcal{D}_1) \cap ddn(\mathcal{D}_2)) = \Delta_1$$

Par un argument similaire, nous obtenons également :

$$\Delta_2 \cup (dln(\mathcal{D}_2) \cap ddn(\mathcal{D}_1)) = \Delta_2$$

Nous en déduisons (en utilisant deux fois l'hypothèse d'induction pour la propriété (5)) :

$$\begin{array}{rl} & dln(\mathcal{D}_1, \mathcal{D}_2) \cap ddn(\mathcal{D}_1, \mathcal{D}_2) \\ = & (dln(\mathcal{D}_1) \cup dln(\mathcal{D}_2)) \cap (ddn(\mathcal{D}_1) \cup ddn(\mathcal{D}_2)) \\ = & \Delta_1 \cup \Delta_2 \cup (dln(\mathcal{D}_1) \cap ddn(\mathcal{D}_2)) \cup (dln(\mathcal{D}_2) \cap ddn(\mathcal{D}_1)) \\ = & \Delta_1 \cup \Delta_2 \end{array}$$

Loc La propriété (1) est satisfaite par induction pour tous les noms sauf a, et elle est satisfaite pour a puisque a est une location dans $B + a : loc(\mathbf{\Delta}' \cup \mathbf{I}')$ et dans Γ par le lemme B.1.2(3) appliqué au typage de a dans les hypothèses de la règle de typage.
La propriété (2) est satisfaite puisque $ddn(a[\mathcal{D} : \mathcal{P}]^{\Delta,I}) = ddn(\mathcal{D})$, et par induction.
La propriété (3) est satisfaite puisque $dsn(a[\mathcal{D} : \mathcal{P}]^{\Delta,I}) = \{a\} \cup dsn(\mathcal{D}) = \{a\} \cup dom(B)$.
La propriété (4) est satisfaite puisque nous avons :

$$\begin{aligned} dn(a[\mathcal{D} : \mathcal{P}]^{\Delta,I}) &= \{a\} \cup dn(\mathcal{D}) \\ &= \{a\} \cup dom(B) \cup ddn(\mathcal{D}) \\ &= dom(B + a : loc(\mathbf{\Delta}' \cup \mathbf{I}')) \cup ddn(a[\mathcal{D} : \mathcal{P}]^{\Delta,I}) \end{aligned}$$

La propriété (5) est satisfaite par :

$$dln(a[\mathcal{D} : \mathcal{P}]^{\Delta,I}) = \emptyset$$

□

Preuve du lemme 4.6.3: Nous prouvons ce lemme par induction sur la taille de αa, en utilisant les hypothèses de la règle de typage Configuration présente en conclusion de la dérivation de typage. Cette induction est possible parce que les locations dépliées forment un arbre.

En ce qui concerne le cas de base, c'est à dire la racine b de l'arbre, nous avons $I_b = \emptyset$ par hypothèse. Donc par définition de l'opérateur *Lookup*, nous avons nécessairement $F_b(n) = b \neq \bot$ si et seulement si $n \in dyn(b)$.

Nous étudions maintenant le cas inductif. Soit une location αa. Par définition de *Lookup*, soit n est dans $dyn(a)$, et nous avons $F_{\alpha a} = a$ et la propriété tient. Sinon n est dans I_a et nous avons $F_{\alpha a}(n) = F_\alpha(n)$. Par hypothèse de la règle Configuration, nous avons $I_{\alpha a} \subseteq \Delta_\alpha \cup I_\alpha$, donc par hypothèse d'induction nous avons $F_\alpha(n) = a' \neq \bot$ avec $n \in dyn(a')$, $\alpha = \beta a' \beta'$ et pour tout $c \in \beta'$, $n \notin dln(c)$. Il nous reste à montrer pour conclure que $n \notin dln(a)$. Nous avons déjà $n \notin dyn(a)$. Supposons que la location αa ait la forme : $\alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta_a, I_a, F_{\alpha a}}$. Nous avons donc $dyn(a) = \Delta_a$ et $dln(a) = dln(\mathcal{D})$. Nous considérons maintenant le typage de la définition de cette location par les règles Soup-Loc et Chem-Def. Nous avons :

$$\Delta_a; I_a; \Gamma \vdash \mathcal{D} :: B; \Delta_a; \Theta$$

Nous en déduisons que par le lemme B.1.3(5), nous avons $n \notin ddn(\mathcal{D})$ (sinon nous aurions $n \in \Delta_a$). Comme $dln(\mathcal{D}) \subseteq dn(\mathcal{D}) = dsn(\mathcal{D}) \cup ddn(\mathcal{D})$ par le lemme B.1.3(4,3), si $n \in dln(a)$ et $n \notin \Delta_a$ alors nous avons nécessairement $n \in dsn(\mathcal{D})$, donc par le lemme B.1.3(1) n est soit statique, soit une location dans Γ.

Comme $n \in dyn(a') = \Delta_{a'}$, il existe une location a' nécessairement dépliée, puisque c'est une location englobante de a ($\alpha = \beta a' \beta'$) et que les locations dépliées forment un arbre (hypothèse de la règle de typage Configuration). Cette location est alors de la forme $\beta a'\,[\mathcal{D}_{a'} : \mathcal{P}_{a'}]^{\Delta_{a'}, I_{a'}, F_{\beta a'}}$. Nous considérons le typage de $\mathcal{D}_{a'}$ par les règles Soup-Loc et Chem-Def.

$$\Delta_{a'}; I_{a'}; \Gamma \vdash \mathcal{D}_{a'} :: B'; \Delta_{a'}; \Theta'$$

Donc par le lemme B.1.3(2,5), n est un nom redéfinissable dans Γ. Si n est un nom de $dln(a)$, c'est un nom statique ou de location, ce qui est impossible. Nous avons donc $n \notin dln(a)$. □

Lemme B.1.4 *Soit $\Gamma \vdash \mathcal{S}$ une dérivation de typage. Pour toute location dépliée αa, nous avons $dyn(a) \cup imp(a) \subseteq dom(\Gamma)$.*

Preuve: Nous montrons tout d'abord que pour toute location dépliée αa de $\mathcal{S}$, nous avons $dyn(a) \subseteq dom(\Gamma)$. C'est le cas puisque par hypothèse de la règle Soup-Loc, tout nom de $dyn(a) = \Delta_{\alpha a}$ est de la forme $\mathbf{n}$ et nous avons $\mathbf{n} : \langle \tau \rangle^+_{\mathbf{n}} \in \Gamma$.

Nous montrons maintenant par induction sur la longueur de αa que nous avons $imp(a) \subseteq dom(\Gamma)$. C'est le cas pour la location racine b puisque $I_b = \emptyset$. Pour tout autre location, c'est le cas puisque par hypothèse de la règle configuration nous avons $I_{\alpha a} \subseteq I_\alpha \cup \Delta_\alpha$, et nous concluons par induction et en utilisant la propriété sur Δ_α. □

Lemme B.1.5 *Soit $\Gamma \vdash \mathcal{S}$ une dérivation de typage. Nous avons $fn(\mathcal{S}) \subseteq dom(\Gamma)$. Cette propriété est également vraie pour tous les autres jugements de typage.*

Preuve: Nous procédons par induction sur la dérivation de typage.

La plupart des cas sont immédiats par induction. Nous détaillons les autres cas.

DEF Nous avons :

$$\begin{array}{rcll} fn(\texttt{def } D \texttt{ in } P) & = & (fn(D) \cup fn(P)) \setminus dsn(D) & \text{par définition de } fn \\ fn(D) & \subseteq & dom(\Gamma + A) & \text{par induction} \\ dom(A) = dom(B) & = & dsn(D) & \text{par le lemme B.1.3(3)} \\ dom(A) \cap dom(\Gamma) & = & \emptyset & \text{par hypothèse de la règle DEF} \end{array}$$

Donc nous avons :

$$fn(D) \setminus dsn(D) = fn(D) \setminus dom(A) \subseteq dom(\Gamma + A) \setminus dom(A) = dom(\Gamma)$$

et par le même raisonnement : $fn(P) \setminus dsn(D) \subseteq dom(\Gamma)$.

JOIN Nous avons :

$$fn((\texttt{x}_i\langle\widetilde{u_i}\rangle)^i \mid (m_j\langle\widetilde{y_j}\rangle)^j \triangleright P) = (\{\texttt{x}_i\}^i \cup \{m_j\}^j \cup fn(P)) \setminus (\{\widetilde{u_i}\}^i \cup \{\widetilde{y_j}\}^j)$$

Donc tout nom libre de la règle de réaction est soit un nom défini (qui sont présents dans $dom(\Gamma)$ par induction sur le typage de $\texttt{x}_i$ et m_j dans les hypothèses de la règle), soit un nom libre de P qui n'est pas un nom reçu. Par induction, les noms libres de P sont soient dans le domaine de Γ, soit des noms reçus. Donc les noms libres de P qui ne sont pas des noms reçus sont dans le domaine de Γ.

LOC Ce cas est immédiat par induction, en utilisant l'hypothèse de la règle pour montrer que $a \in dom(\Gamma)$, $\Delta \subseteq dom(\Gamma)$ et $I \subseteq dom(\Gamma)$. □

Nous étudions maintenant des propriétés de renommage.

Définition B.1.6 (Substitution bien formée) *Soit θ une substitution ayant pour domaine un sous-ensemble de l'ensemble des noms n, des variables de types α et des variables de types de noms δ. Nous disons que θ est bien formée si et seulement si $\theta \circ \theta = \theta$ et si c'est une substitution :*

- *des variables de noms dans les noms ;*
- *des noms de canaux dynamiques dans les noms de canaux dynamiques ;*
- *des noms de canaux statiques dans les noms de canaux statiques ;*
- *des noms de locations dans les noms de locations ;*
- *des variables de types de noms dans les variables de types de noms ou les noms de canaux dynamiques ;*
- *des variables de types dans les types.*

Nous étendons la notion de substitution bien formée aux types, aux schémas de types, aux environnements de typage et aux configurations de la manière traditionnelle. Dans la suite, nous définissons le domaine d'une substitution bien formée comme étant l'ensemble des éléments de son domaine pour lesquels elle n'est pas l'identité (c'est à dire $dom(\theta) = \{x, \theta(x) \neq x\}$). Son image est tout simplement l'image de son domaine (c'est à dire $ran(\theta) = \theta(dom(\theta))$).

Dans la suite, nous disons qu'une substitution bien formée θ est *injective pour les noms* si nous avons :

$$\forall x, x' \in dom(\theta), x \neq x' \implies fn(\theta(x)) \cap fn(\theta(x')) = \emptyset$$

Nous définissons la notion similaire de substitution *injective pour les variables* (de types et de types de noms).

Nous remarquons que l'image d'un ensemble de noms dynamiques et de variables de types de noms est un ensemble de noms dynamiques et de variables de types de noms.

Lemme B.1.7 (Substitution des types et sous-typage) *Soit θ une substitution bien formée. Si nous avons $\tau \leq \tau'$, alors nous avons $\theta(\tau) \leq \theta(\tau')$.*

Preuve: La preuve est immédiate par induction sur la dérivation de sous-typage, en nous basant sur le fait que si $\boldsymbol{\Delta} \subseteq \boldsymbol{\Delta}'$, alors nous avons $\theta(\boldsymbol{\Delta}) \subseteq \theta(\boldsymbol{\Delta}')$. □

Lemme B.1.8 (Substitution sur les dérivations de typage) *Soit S une configuration. Pour tout environnement de typage Γ tel que $\Gamma \vdash \mathcal{S}$ possède une dérivation de typage, nous avons les propriétés suivantes :*

1. *Soit Γ' un environnement de typage qui soit Γ excepté pour les variables de types et les variables de types de noms généralisées dans les schémas de types (renommées de manière injective et sans capture). Nous avons alors $\Gamma' \vdash \mathcal{S}$. Cette propriété est également vraie pour tous les jugements de typage excepté pour les règles* Configuration, Soup-Loc *et* Chem-Def. *Dans ce cas, la propriété est vraie si le renommage des variables généralisées se fait vers des variables qui ne sont pas présentes dans la dérivation de typage.*
2. *Soit θ une substitution bien formée, injective pour les noms et les variables, telle que les noms et les variables de $ran(\theta)$ soient toutes distinctes des noms et des variables présentes dans la dérivation. Nous avons alors $\theta(\Gamma) \vdash \theta(\mathcal{S})$. Cette propriété est vraie pour tous les jugements si dès qu'un message résolu $a.n\langle\widetilde{n_i}\rangle$ est présent avec a renommé, alors la location a est également renommée.*

Preuve: Nous remarquons tout d'abord que si le renommage θ est injectif pour les noms et les variables, nous avons les propriétés suivantes (qui sont également vraies pour I et $\mathbf{I}$) :
- $n \in \Delta$ si et seulement si $\theta(n) \in \theta(\Delta)$
- $w \in \mathbf{\Delta}$ si et seulement si $\theta(w) \in \theta(\mathbf{\Delta})$
- $\Delta \cup \Delta' = \theta(\Delta) \cup \theta(\Delta')$
- $\Delta \cap \Delta' = \theta(\Delta) \cap \theta(\Delta')$

Nous procédons par induction sur la dérivation de typage. Les conditions d'induction pour Γ' et θ sont immédiatement préservées lorsque l'on considère l'hypothèse d'une règle. La condition concernant le renommage d'une location si un message résolu est renommé est elle aussi conservée, puisqu'elle est globale.

En ce qui concerne la propriété (2), le typage des définitions se fait en remplaçant Θ par $fv(\theta(\Theta))$.

Name Soit φ la substitution instanciant τ' en $\tau : \tau = \varphi(\tau')$.

Nous commençons par prouver la propriété (1). Nous avons $\sigma = \forall\widetilde{\psi(\alpha)}\widetilde{\psi(\delta)}.\psi(\tau') \in \Gamma'$ si ψ est le renommage des variables généralisées permettant de passer de Γ à Γ'. Puisque ψ est injective et ne capture pas de variable libre de σ, elle possède une fonction réciproque ψ^{-1} avec $\psi^{-1}(\psi(\tau')) = \tau'$. Nous instancions donc σ par $\varphi \circ \psi^{-1}$ et obtenons la conclusion désirée par règle Name.

Nous prouvons maintenant la propriété (2). Nous voulons prouver que $\theta(\Gamma) \vdash \theta(m) : \theta(\tau)$.

Par hypothèse de la règle Name, nous avons $\theta(m) : \forall\widetilde{\alpha}\widetilde{\delta}.\theta'(\tau') \in \theta(\Gamma)$, avec θ' étant θ excepté sur $\widetilde{\alpha}\cup\widetilde{\delta}$ où c'est l'identité. Nous cherchons une instanciation φ' telle que $\varphi'(\theta'(\tau')) = \theta(\varphi(\tau'))$. Nous considérons l'instanciation $\varphi' = \theta \circ \varphi$. Soit β un nom ou une variable. Si nous avons $\beta \in dom(\varphi)(= dom(\varphi') = \widetilde{\alpha} \cup \widetilde{\delta})$, alors $\varphi'(\theta'(\beta)) = \varphi'(\beta) = \theta(\varphi(\beta))$. Si β n'est pas dans le domaine de φ, alors nous ne pouvons avoir $\theta(\beta) \in dom(\varphi') = \widetilde{\alpha} \cup \widetilde{\delta}$ puisque l'image de θ est disjointe des noms et variables de Γ, et puisque le domaine de φ' est présent dans Γ. Par conséquent nous avons $\varphi'(\theta(\beta)) = \theta(\beta) = \theta(\varphi(\beta))$.

Sub La propriété (1) est immédiate par induction.

Nous prouvons la propriété (2). Par hypothèse d'induction, nous avons $\theta(\Gamma) \vdash \theta(m) : \theta(\tau)$. Par le lemme B.1.7, nous avons $\theta(\tau) \leq \theta(\tau')$. Nous concluons par la règle Sub.

Tuple Immédiat par induction.

Msg Immédiat par induction, puisque $\theta(\mathbf{\Delta} \cup \mathbf{I}) = \theta(\mathbf{\Delta}) \cup \theta(\mathbf{I})$.

R-Msg Ce cas est similaire au cas précédent. Nous devons cependant vérifier que $\theta(n) \in dln(\theta(a))$. Puisque que nous avons $n \in dln(a)$ par hypothèse de la règle de typage, il existe une location a dans $\mathcal{S}$ telle que n est un nom défini de a. Cette location étant renommée par hypothèse du lemme, il existe une location $\theta(a)$ telle que $\theta(n)$ est définie dans $\theta(a)$.

Def Nous prouvons tout d'abord la propriété (1). Celle-ci est immédiate par induction puisque $fv(\Gamma') = fv(\Gamma)$ et $dom(\Gamma') = dom(\Gamma)$.

Nous prouvons maintenant la propriété (2). Soit θ' la substitution bien formée θ restreinte à $dom(\theta) \setminus dsn(D)$. Nous avons $\theta(\texttt{def } D \texttt{ in } P) = \texttt{def } \theta'(D) \texttt{ in } \theta'(P)$. La principale difficulté dans la preuve de ce cas est que θ' peut renommer des variables qui sont généralisées. Soit Ξ l'ensemble $fv(B) \setminus (fv(\Gamma) \cup \Theta)$ et θ'' la substitution bien formée θ' restreinte à $dom(\theta') \setminus \Xi$. Par hypothèse d'induction par la propriété 2, nous avons :

$$\begin{aligned} \theta''(\mathbf{\Delta}); \theta''(\mathbf{I}); \theta''(\Gamma + A) &\vdash \theta''(D) :: \theta''(B); \emptyset; fv(\theta''(\Theta)) \\ \theta''(\mathbf{\Delta}); \theta''(\mathbf{I}); \theta''(\Gamma + A) &\vdash \theta''(P) \end{aligned}$$

Puisque θ'' ne diffère de θ' que sur des variables présentes dans B (elles ne sont pas dans Γ par définition de Ξ, donc elles ne peuvent être libres dans $\mathbf{\Delta}$ ni $\mathbf{I}$; elles ne peuvent être libres dans le terme puisque celui-ci ne peut contenir de variables de types ou de types de noms ; et elles ne sont pas libres dans Θ par définition de Ξ), nous avons :

$$\begin{array}{lll} \theta'(\mathbf{\Delta}); \theta'(\mathbf{I}); \theta'(\Gamma + A) & \vdash & \theta'(D) :: \theta''(B); \emptyset; fv(\theta'(\Theta)) \\ \theta'(\mathbf{\Delta}); \theta'(\mathbf{I}); \theta'(\Gamma + A) & \vdash & \theta'(P) \end{array}$$

Soit $A' = Gen(\theta''(B), fv(\theta'(\Gamma)), fv(\theta'(\Theta))))$. Nous montrons que nous avons $A' = \theta'(A)$. Tout d'abord, ces deux environnements ont le même domaine, puisque θ' ne renomme pas les noms de $dsn(D)$ et que $dom(B) = dom(\theta''(B)) = dsn(D)$ par le lemme B.1.3(3). Nous vérifions donc que les schémas de types associés sont identiques. Pour ce faire, nous montrons que les variables généralisées sont les mêmes dans les deux cas :

$$fv(\theta''(B)) \setminus (fv(\theta'(\Gamma)) \cup fv(\theta'(\Theta))) = fv(B) \setminus (fv(\Gamma) \cup \Theta)$$

Soit β une variable du deuxième ensemble (c'est à dire de Ξ). Nous avons nécessairement $\beta \notin dom(\theta'')$. Par conséquent nous avons $\beta \in fv(\theta''(B))$. La variable β étant présente dans la dérivation de typage initiale, elle ne peut donc être présente dans l'image de θ par hypothèse sur θ. Puisque β n'est pas présent dans $fv(\Gamma) \cup \Theta$, nous avons donc $\beta \notin fv(\theta'(\Gamma)) \cup fv(\theta'(\Theta))$. La variable β est donc dans le premier ensemble.

Nous supposons maintenant que β est une variable du première ensemble. Nous avons soit $\beta \notin fv(ran(\theta''))$, soit $\beta \in fv(ran(\theta''))$. Dans le premier cas, nous avons nécessairement $\beta \in fv(B)$ et $\theta''(\beta) = \beta$. Nous montrons que nécessairement $\beta \notin fv(\Gamma) \cup \Theta$. Supposons le contraire, alors nous avons $\beta \notin \Xi$, donc $\theta'(\beta) = \theta''(\beta) = \beta$. Donc, puisque $\beta \in fv(\Gamma) \cup \Theta$, nous avons $\beta \in fv(\theta'(\Gamma)) \cup fv(\theta'(\Theta))$, ce qui est impossible puisque β est dans le premier ensemble. Donc nous avons $\beta \notin fv(\Gamma) \cup \Theta$, donc β est dans le deuxième ensemble. Nous montrons maintenant que le cas $\beta \in fv(ran(\theta''))$ ne peut arriver (c'est à dire qu'aucune variable dans l'image de θ'' ne peut être généralisée). Par hypothèse sur θ, si nous avons $\beta \in fv(ran(\theta'')) \subseteq fv(ran(\theta))$, alors nécessairement $\beta \notin fv(B)$, puisque les noms et variables dans l'image de θ sont distincts de ceux présents dans la dérivation de typage. Puisque nous avons $\beta \in fv(\theta''(B))$, il existe un $\beta' \in dom(\theta'') \cap fv(B)$ tel que $\beta \in fv(\theta''(\beta'))$, avec $\beta' \neq \beta$. Par définition de θ'', nous avons nécessairement $\beta' \notin \Xi$ (sinon $\theta''(\beta') = \beta' = \beta$). Comme β' est une variable de $fv(B)$ qui n'est pas dans Ξ, nous avons donc $\beta' \in fv(\Gamma) \cup \Theta$. Les substitutions θ' et θ'' n'étant différentes que sur Ξ, et β' n'étant pas une variable de Ξ, nous avons également $\beta \in fv(\theta'(\beta'))$. Nous en déduisons $\beta \in fv(\theta'(fv(\Gamma) \cup \Theta)) = fv(\theta'(\Gamma)) \cup fv(\theta'(\Theta))$, ce qui est impossible, puisque β est dans le premier ensemble.

Soit $n : \tau$ une association de l'environnement B. Nous avons $n : \theta''(\tau)$ comme association dans l'environnement $\theta''(B)$, $n : \forall \widetilde{\alpha}\widetilde{\delta}.\tau \in A$, et $n : \forall \widetilde{\alpha'}\widetilde{\delta'}.\theta''(\tau) \in A'$, pour des $\widetilde{\alpha}$, $\widetilde{\delta}$, $\widetilde{\alpha'}$, et $\widetilde{\delta'}$. L'ensemble des variables généralisées de A' et de A étant identiques, nous avons en fait $n : \forall \widetilde{\alpha}\widetilde{\delta}.\theta''(\tau) \in A'$. Pour conclure, nous remarquons que $\theta'(n : \forall \widetilde{\alpha}\widetilde{\delta}.\tau) = n : \forall \widetilde{\alpha}\widetilde{\delta}.\theta''(\tau) \in A'$ (il ne peut y avoir de capture puisque les variables de l'image de θ'' sont toutes différentes de celles présentes dans la dérivation initiale, en particulier des variables généralisées). Nous avons donc $A' = \theta'(A)$.

Nous avons :

$$\begin{array}{lll} \theta'(\mathbf{\Delta}); \theta'(\mathbf{I}); \theta'(\Gamma) + A' & \vdash & \theta'(D) :: \theta''(B); \emptyset; fv(\theta'(\Theta)) \\ \theta'(\mathbf{\Delta}); \theta'(\mathbf{I}); \theta'(\Gamma) + A' & \vdash & \theta'(P) \end{array}$$

Pour conclure, nous remarquons que $dom(A') = dom(A)$, donc $dom(A') \cap dom(\Gamma) = \emptyset$. De plus, θ' étant θ excepté sur les noms de $dom(A) = dsn(D)$ (par le lemme B.1.3(3)) qui sont des noms de canaux statiques (par le lemme B.1.3(1)), nous avons $\theta'(\Gamma) = \theta(\Gamma)$ (en effet, nous avons $dom(\Gamma) \cap dom(A) = \emptyset$, et les noms présents dans les types sont nécessairement des noms dynamiques par définition des types), $\theta'(\mathbf{\Delta}) = \theta(\mathbf{\Delta})$ et $\theta'(\mathbf{I}) = \theta(\mathbf{I})$ (en effet, les noms de $\mathbf{\Delta}$ et $\mathbf{I}$ sont des noms dynamiques ou des variables de types de noms).

Nous concluons par la règle de typage Def.

Par Immédiat par induction.

Go Immédiat par induction.

Nu La propriété (1) est immédiate par induction, puisque $fn(\Gamma) = fn(\Gamma')$.

Nous prouvons maintenant la propriété (2). Soit θ' la substitution bien formée θ restreinte à $dom(\theta) \setminus \{\mathbf{d}\}$. Les noms de la substitution θ' étant frais par rapport à la dérivation initiale, nous avons $\mathbf{d} \notin fn(\theta'(\Gamma))$. Par induction, nous avons donc une dérivation de typage pour $\theta'(\Delta); \theta'(I); \theta'(\Gamma) \vdash \nu\mathbf{d}.\theta'(P) = \theta(\nu\mathbf{d}.P)$. Nous concluons en remarquant que le nom $\mathbf{d}$ n'est pas libre dans Γ, il n'est donc pas libre dans $\mathbf{\Delta}$ ni $\mathbf{I}$, donc θ et θ' coïncident sur Γ, $\mathbf{\Delta}$ et $\mathbf{I}$.

Nil Immédiat.

Join La propriété (1) est immédiate par induction.

Nous prouvons maintenant la propriété (2). Soit θ' la substitution bien formée θ restreinte à $dom(\theta) \setminus (\bigcup_i \widetilde{u_i} \cup \bigcup_j \widetilde{y_j})$. Nous avons :

$$\theta((\mathtt{x_i}\langle\widetilde{u_i}\rangle)^i \mid (m_j\langle\widetilde{y_j}\rangle)^j \triangleright P) = (\theta(\mathtt{x_i})\langle\widetilde{u_i}\rangle)^i \mid (\theta(m_j)\langle\widetilde{y_j}\rangle)^j \triangleright \theta'(P)$$

Par hypothèse d'induction, nous avons :

$$\theta'(\mathbf{\Delta}); \theta'(\mathbf{I}); \theta'(\Gamma) + (\widetilde{u_i} : \theta'(\widetilde{\tau_i}))^i + (\widetilde{y_j} : \theta'(\widetilde{\tau_j}))^j \vdash \theta'(P)$$

La substitution θ' n'est différente de θ que sur les noms reçus, qui ne peuvent être présents dans les types, qui sont différents des noms dynamiques ou des variables de types de noms, et qui ne sont pas présents dans le domaine de Γ par hypothèse de la règle Join. Nous avons donc :

$$\theta(\mathbf{\Delta}); \theta(\mathbf{I}); \theta(\Gamma) + (\widetilde{u_i} : \theta(\widetilde{\tau_i}))^i + (\widetilde{y_j} : \theta(\widetilde{\tau_j}))^j \vdash \theta'(P)$$

Par induction, nous avons également $\theta(\Gamma) \vdash \theta(\mathtt{x_i}) : \langle\theta(\widetilde{\tau_i})\rangle$ et $\theta(\Gamma) \vdash \theta(m_j) : \langle\theta(\widetilde{\tau_j})\rangle^+_{\theta(w_j)}$

Les noms libres dans l'image de θ étant différents des noms présents dans la dérivation initiale, les noms reçus sont bien distincts des noms libres de $\theta(\Gamma)$.

Nous montrons maintenant que la condition sur Θ est satisfaite. Soit β une variable partagée entre $fv(\theta(\tau))$ et $fv(\theta(\tau'))$. Si β n'est pas dans l'image de θ, alors β est à la fois présente dans $fv(\tau)$ et $fv(\tau')$ et nous avons $\theta(\beta) = \beta$. Donc β est présente dans Θ, donc dans $fv(\theta(\Theta))$. Si β est présente dans l'image de θ, il existe alors $\beta_1 \in fv(\tau) \cap dom(\theta)$ et $\beta_2 \in fv(\tau') \cap dom(\theta)$ telles que $\beta \in fv(\theta(\beta_1)) \cap fv(\theta(\beta_2))$. La substitution θ étant injective sur les variables, nous avons nécessairement $\beta_1 = \beta_2$, donc nous avons $\beta_1 \in \Theta$, donc $\beta \in fv(\theta(\Theta))$.

Nous concluons par la règle Join.

Top Immédiat.

And Immédiat par induction.

Loc Immédiat par induction.

Configuration La propriété (1) est immédiate par induction. La propriété (2) est satisfaite, par induction pour le typage de chaque location, et puisque θ est injective sur les noms et les variables.

Soup-Loc Immédiat par induction, puisque les noms dynamiques sont nécessairement renommés en noms dynamiques.

Chem-Def Nous prouvons tout d'abord la propriété (1). Cette propriété n'est pas immédiate puisque la condition $A \subseteq \Gamma$ n'est pas satisfaite avec Γ'. Il est donc nécessaire de modifier A pour que ses noms généralisés reflètent la modification opérée sur Γ. Soit φ la substitution renommant les noms généralisés de Γ pour donner Γ'. Ce renommage est une substitution bien formée, injective pour les noms et les variables et dont les noms et variables de l'image ne sont pas présents dans la dérivation de typage initiale, par hypothèse du lemme. Nous pouvons donc appliquer l'hypothèse d'induction pour la propriété (2) avec la substitution φ à la dérivation donnant le typage de la définition :

$$\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D} :: \varphi(B); \Delta_{\mathcal{D}}; \Theta$$

En effet, le renommage n'implique que des noms généralisés, qui ne peuvent donc être libres ni dans Γ (donc dans $\mathbf{\Delta}$ et $\mathbf{I}$), ni dans Θ. Ces variables n'apparaissent pas dans les termes, donc elles n'apparaissent ni dans $\mathcal{D}$, ni dans $\Delta_{\mathcal{D}}$ (qui est un sous-ensemble de $dln(\mathcal{D})$ par le lemme B.1.3(6))

Soit $A' = Gen(\varphi(B), fv(\Gamma), \Theta)$. Les environnements A et A' ne sont différents que par leurs variables généralisées, renommées par φ (cette substitution ayant pour image des variables fraîches, elle n'empêche aucune généralisation). Comme nous avons $A \subseteq \Gamma$, nous avons alors $A' \subseteq \Gamma'$. Nous utilisons l'hypothèse d'induction pour la propriété (1) pour obtenir :

$$\mathbf{\Delta}; \mathbf{I}; \Gamma' \vdash \mathcal{D} :: \varphi(B); \Delta_{\mathcal{D}}; \Theta$$

Nous remarquons que le renommage de Γ en Γ' n'est plus vers des variables fraîches, puisque elles sont présentes dans $\varphi(B)$. Ceci n'est cependant pas nécessaire pour appliquer l'hypothèse d'induction aux simples définitions. Nous concluons par la règle Chem-Def pour obtenir $\mathbf{\Delta}; \mathbf{I}; \Gamma' \vdash \mathcal{D} : \Delta_{\mathcal{D}}$.

Nous prouvons maintenant la propriété (2). Soit Ξ l'ensemble des variables généralisées de A. Ces variables étant généralisées, elles ne sont présentes ni $fv(\Gamma)$ (donc $\mathbf{\Delta}$ et $\mathbf{I}$), ni dans Θ. De plus, comme ce sont des variables de types elles ne sont pas présentes dans $\mathcal{D}$. Puisque nous avons $\Delta_{\mathcal{D}} \subseteq dln(\mathcal{D}) \subseteq fn(\mathcal{D})$ (par le lemme B.1.3(6)), elles ne sont pas non plus présentes dans $\Delta_{\mathcal{D}}$. Soit θ' la substitution bien formée θ restreinte à $dom(\theta) \setminus \Xi$. Par induction nous avons :

$$\theta'(\mathbf{\Delta}); \theta'(\mathbf{I}); \theta'(\Gamma) \vdash \theta'(\mathcal{D}) :: \theta'(B); \theta'(\Delta_{\mathcal{D}}); fv(\theta'(\Theta))$$

La substitution θ' étant θ excepté pour des noms libres de B, nous avons :

$$\theta(\mathbf{\Delta}); \theta(\mathbf{I}); \theta(\Gamma) \vdash \theta(\mathcal{D}) :: \theta'(B); \theta(\Delta_{\mathcal{D}}); fv(\theta(\Theta))$$

Nous reproduisons l'argument du cas Def pour en déduire :

$$A' = Gen(\theta'(B), fv(\theta(\Gamma)), fv(\theta(\Theta))) = \theta'(A) = \theta(A)$$

Donc nous avons bien $A' \subseteq \theta(\Gamma)$. Nous concluons par la règle Chem-Def.

□

Dans la suite, nous utilisons le lemme B.1.8(1) pour identifier les dérivations de typage ayant des environnements de typage identiques au renommage de variables généralisées en des variables fraîches près.

Lemme B.1.9 *Soient deux dérivations de typage* $\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash P_1$ *et* $\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash P_2$. *Soit* $C(\cdot)$ *un contexte tel que la première dérivation puisse être étendue en une dérivation* $\Gamma' \vdash C(P_1)$, *alors la deuxième dérivation peut également être étendue pour donner* $\Gamma' \vdash C(P_2)$.

Cette propriété est également vraie pour les définitions, et le terme $C(P_1)$ *peut être une définition, un processus, une soupe ou une configuration.*

Preuve: Nous procédons par induction sur le contexte, en ne considérant que les termes syntaxiquement corrects.

$C = (\cdot)$ Immédiat.

$C = n\langle\widetilde{m}\rangle; C'$ La dernière règle de typage utilisée est nécessairement la règle Msg. Nous concluons par induction.

$C = C' \mid P$ La dernière règle de typage utilisée pour typer $C(P_1)$ est nécessairement la règle Par. Nous concluons par induction. Le cas symétrique ($C = P \mid C'$) est identique.

$C = \texttt{def}\ C'\ \texttt{in}\ P$ La dernière règle de typage utilisée pour typer $C(P_1)$ est nécessairement la règle Def. Par hypothèse de cette règle, nous avons une dérivation :

$$\mathbf{\Delta}'; \mathbf{I}'; \Gamma' + A \vdash C'(P_1) :: B; \emptyset; \Theta$$

Par induction, nous avons donc :

$$\mathbf{\Delta}'; \mathbf{I}'; \Gamma' + A \vdash C'(P_2) :: B; \emptyset; \Theta$$

Nous concluons par la règle Def.

$C = \texttt{def}\ \mathcal{D}\ \texttt{in}\ C'$ Comme dans le cas précédent, la dernière règle de typage utilisée est nécessairement la règle Def. Le résultat est immédiat par induction.

$C = \nu\mathbf{d}.C'$ Immédiat par induction par la règle de typage Nu.

$C = J \triangleright C'$ Si la dernière règle de typage utilisée est Join, le résultat est immédiat par induction puisque le seul changement dans les hypothèses de la règle est dans le processus gardé.

Si la dernière règle de typage utilisée est Chem-Def, le typage de la définition utilise alors la règle Join. Nous concluons par induction pour le typage du processus gardé, et par application de la règle Join et de la règle Chem-Def (la généralisation est la même).

$C = C', \mathcal{D}$ Ce résultat est immédiat par induction si la dernière règle utilisée est la règle And. Dans le cas où la dernière règle est Chem-Def, alors le typage de la définition se termine par la règle And et nous concluons par induction, application de la règle And et de la règle Chem-Def.

La règle symétrique est identique.

$C = a\,[\mathcal{D} : C']^{\Delta,I}$, $C = a\,[C' : \mathcal{P}]^{\Delta,I}$ Comme dans les deux cas précédent, le résultat est immédiat par induction si la dernière règle utilisée est la règle Loc. Dans la cas où la dernière règle de typage est la règle Chem-Def, la règle précédente pour typer la définition est nécessairement la règle Loc, et nous concluons par induction, application de la règle Loc puis de la règle Chem-Def.

$C = \mathcal{S} \parallel C' \parallel \mathcal{S}'$ Par définition des contextes, $C'(P_1)$ et $C'(P_2)$ sont des locations dépliées qui ne peuvent différer que par leur définition ou leur processus. La dernière règle de typage appliquée est la règle Configuration, et le résultat est immédiat par induction.

$C = \alpha a\,[\mathcal{D} : C']^{\Delta,I,F}$, $C = \alpha a\,[C' : \mathcal{P}]^{\Delta,I,F}$ La dernière règle appliquée est nécessairement Soup-Loc. Le résultat est immédiat par induction.

□

Lemme B.1.10 *Soient $\mathcal{S}$ et $\mathcal{S}'$ deux configurations telles que $\mathcal{S} =_\alpha \mathcal{S}'$. Si nous avons $\Gamma \vdash \mathcal{S}$, alors nous avons $\theta(\Gamma) \vdash \mathcal{S}'$ pour une substitution bien formée θ.*

Preuve: Soit θ_α la substitution correspondant à l'α-conversion ayant lieu. Nous remarquons que dans chaque cas θ_α est une substitution bien formée injective pour les noms (elle ne renomme pas les variables de types). Nous commençons par renommer les noms qui sont à la fois présents dans la dérivation et dans $ran(\theta_\alpha)$. Par définition de l'α-conversion, ces noms ne sont pas présents dans $\mathcal{S}$. Soit θ une substitution bien formée injective pour les noms et variables qui renomme ces noms en des noms frais (différents de tous les noms présents dans la dérivation de typage et dans l'image de θ_α).

Par le lemme B.1.8(2), nous avons $\theta(\Gamma) \vdash \mathcal{S}$ (puisque $\theta(\mathcal{S}) = \mathcal{S}$). Dans le reste de ce lemme, nous ne considérons que cet environnement renommé.

Nous procédons par cas selon les noms qui sont renommés. Cela peut être les noms reçus d'une règle de réaction, les noms définis statiques d'une définition locale ou le nom restreint d'une restriction.

Dans le cas de noms reçus d'une règle de réaction, nous avons :

$$\mathcal{S} = C(J \triangleright Q) =_\alpha C(\theta_\alpha(J) \triangleright \theta_\alpha(Q)) = \mathcal{S}'$$

avec $dom(\theta_\alpha) \subseteq rn(J)$.

Nous considérons la règle de typage Join utilisée pour typer $J \triangleright Q$. Elle est de la forme :

$$\frac{\begin{array}{c}\mathbf{\Delta};\mathbf{I};\Gamma_J + (\widetilde{u_i} : \widetilde{\tau_i})^i + (\widetilde{y_j} : \widetilde{\tau_j})^j \vdash P \\ \Gamma_J \vdash \mathtt{x}_i : \langle \widetilde{\tau_i} \rangle \qquad \Gamma_J \vdash m_j : \langle \widetilde{\tau_j} \rangle^+_{w_j} \qquad \left(\bigcup_i \widetilde{u_i} \cup \bigcup_j \widetilde{y_j}\right) \cap dom(\Gamma_J) = \emptyset \\ \forall (n:\tau), (n':\tau') \in \left(\{\mathtt{x}_i : \langle \widetilde{\tau_i} \rangle\} \cup \{m_j : \langle \widetilde{\tau_j} \rangle^+_{w_j}\}\right).n \neq n' \implies fv(\tau) \cap fv(\tau') \subseteq \Theta\end{array}}{\mathbf{\Delta};\mathbf{I};\Gamma_J \vdash (\mathtt{x}_i\langle \widetilde{u_i} \rangle)^i \mid (m_j\langle \widetilde{y_j} \rangle)^j \triangleright P :: (\mathtt{x}_i : \langle \widetilde{\tau_i} \rangle)^i ; \bigcup_j \{m_j\} ; \Theta}\ [\textsc{Join}]$$

Le processus gardé par la règle de réaction ne pouvant contenir aucun message résolu, nous utilisons le lemme B.1.8(2) pour obtenir la dérivation :

$$\mathbf{\Delta};\mathbf{I};\Gamma_J + (\widetilde{\theta_\alpha(u_i)} : \widetilde{\tau_i})^i + (\widetilde{\theta_\alpha(y_j)} : \widetilde{\tau_j})^j \vdash \theta_\alpha(P)$$

(les noms reçus ne sont pas présents dans le domaine de Γ_J par hypothèse de la règle, et ils ne peuvent être présents ni dans les types, ni dans $\mathbf{\Delta}$, ni dans $\mathbf{I}$; ils sont également distincts des m_j par le lemme B.1.5 et le typage des m_j).

Toutes les autres hypothèses de la règle Join étant toujours satisfaites, et le renommage se faisant vers des noms frais (donc $(\bigcup_i \widetilde{\theta_\alpha(u_i)} \cup \bigcup_j \widetilde{\theta_\alpha(y_j)}) \cap dom(\Gamma) = \emptyset$), nous pouvons conclure par la règle Join et le lemme B.1.9.

Nous considérons maintenant l'α-renommage de noms définis statiques :

$$\mathcal{S} = C(\texttt{def } D \texttt{ in } Q) =_\alpha C(\texttt{def } \theta_\alpha(D) \texttt{ in } \theta_\alpha(Q)) = \mathcal{S}'$$

avec $dom(\theta_\alpha) \subseteq dsn(D)$ et θ_α est injective pour les noms vers des noms qui ne sont pas présents dans toute la dérivation de typage (en utilisant bien entendu l'environnement $\theta(\Gamma)$).

Nous considérons le jugement de typage Def typant le processus $\texttt{def } D \texttt{ in } Q$. Il est de la forme :

$$\frac{A = Gen(B, fv(\Gamma_D), \Theta) \qquad \begin{array}{c}\mathbf{\Delta};\mathbf{I};\Gamma_D + A \vdash D :: B;\emptyset;\Theta \\ \mathbf{\Delta};\mathbf{I};\Gamma_D + A \vdash Q\end{array} \qquad dom(A) \cap dom(\Gamma_D) = \emptyset}{\mathbf{\Delta};\mathbf{I};\Gamma_D \vdash \texttt{def } D \texttt{ in } Q} \text{ [Def]}$$

Les définitions D et processus P ne contenant aucune occurrence d'un message résolu, nous pouvons appliquer le lemme B.1.8(2) pour obtenir :

$$\begin{array}{rcl}\mathbf{\Delta};\mathbf{I};\Gamma_D + \theta_\alpha(A) & \vdash & \theta_\alpha(D) :: \theta_\alpha(B);\emptyset;\Theta \\ \mathbf{\Delta};\mathbf{I};\Gamma_D + \theta_\alpha(A) & \vdash & \theta_\alpha(Q)\end{array}$$

(les noms définis statiques ne peuvent être présents dans le domaine de Γ_D par hypothèse de la règle, ils ne peuvent être présents ni dans les types, ni dans $\mathbf{\Delta}$, ni dans $\mathbf{I}$, ni dans Θ puisque ce sont des noms de canaux statiques).

Le renommage n'impliquant que des noms statiques, il ne modifie pas les types des environnements B et A, et nous avons $\theta_\alpha(A) = Gen(\theta_\alpha(B), fv(\Gamma_D), \Theta)$. De plus, le renommage étant vers des noms frais, nous avons $dom(\theta_\alpha(A)) \cap dom(\Gamma_D) = \emptyset$. Nous concluons par la règle Def et le lemme B.1.9.

Nous considérons maintenant l'α-renommage des noms restreints :

$$\mathcal{S} = C(\nu\mathbf{d}.P) =_\alpha C(\nu\theta_\alpha(\mathbf{d}).\theta_\alpha(P)) = \mathcal{S}'$$

avec $dom(\theta_\alpha) \subseteq \{\mathbf{d}\}$, et θ_α est injective pour les noms vers des noms qui ne sont pas présents dans toute la dérivation de typage.

Le jugement de typage Nu typant le processus $\nu\mathbf{d}.P$ est de la forme :

$$\frac{\mathbf{d} \notin fn(\Gamma_d) \qquad \mathbf{\Delta};\mathbf{I};\Gamma_d + \mathbf{d} : \langle\tau\rangle^+_{\mathbf{d}} \vdash P}{\mathbf{\Delta};\mathbf{I};\Gamma_d \vdash \nu\mathbf{d}.P} \text{ [Nu]}$$

Comme dans les cas précédents, nous appliquons le lemme B.1.8(2) pour obtenir :

$$\mathbf{\Delta};\mathbf{I};\Gamma_d + \theta_\alpha(\mathbf{d}) : \langle\theta_\alpha(\tau)\rangle^+_{\theta_\alpha(\mathbf{d})} \vdash \theta_\alpha(P)$$

Puisque nous avons $\theta_\alpha(\mathbf{d}) \notin fn(\Gamma_d)$, nous concluons par la règle Nu et le lemme B.1.9. □

Lemme B.1.11 (Variable inutile) *Soit n un nom, Γ un environnement de typage tel que $n \notin dom(\Gamma)$, $\mathcal{S}$ une configuration et σ un schéma de type tels que l'ensemble $fn(n : \sigma) \setminus fn(\Gamma)$ est un ensemble de noms qui ne sont pas liés dans la configuration. Nous avons :*

$$\Gamma \vdash \mathcal{S} \quad \Longrightarrow \quad \Gamma + n : \sigma \vdash \mathcal{S}$$

La même propriété est satisfaite pour tous les jugements.

Preuve: Nous procédons par induction sur la dérivation de typage. L'hypothèse sur les noms liés pour appliquer l'induction pour les jugements de la forme $\mathbf{\Delta};\mathbf{I};\Gamma \vdash \ldots$ est que les noms de $fn(n : \sigma) \setminus fn(\Gamma)$ n'apparaissent pas liés dans le terme. Cette hypothèse est immédiatement satisfaite pour un sous-terme si elle est satisfaite pour le terme et si l'ensemble $fn(\Gamma)$ croît.

Name Immédiat.

Sub, Tuple, Msg, R-Msg Immédiat par induction.

Def Par hypothèse, nous avons une dérivation de typage de la forme $\mathbf{\Delta};\mathbf{I};\Gamma + A \vdash \mathcal{D} :: B;\emptyset;\Theta$ avec $A = Gen(B, fv(\Gamma), \Theta)$. Les variables généralisées pouvant entrer en conflit avec les variables du schéma σ qui ne sont pas libres dans Γ, nous renommons ces variables en des variables fraîches en utilisant le lemme B.1.8(2) pour obtenir le jugement $\mathbf{\Delta};\mathbf{I};\Gamma + A \vdash D :: B';\emptyset;\Theta$ (les variables renommées ne sont pas libres dans Γ par hypothèse, elles ne sont donc pas libres dans $\mathbf{\Delta}$ ni dans $\mathbf{I}$; elles ne sont pas libres dans Θ puisqu'elles sont généralisées ; elles ne sont

pas libres dans A pour la même raison ; elles ne sont pas présentes dans D puisque ce sont des variables de types ou de types de noms). Nous pouvons effectivement appliquer ce lemme puisqu'aucun nom de location n'est renommé. Nous obtenons : $\mathbf{\Delta}; \mathbf{I}; \Gamma + A' \vdash D :: B'; \emptyset; \Theta$ avec $A' = \mathit{Gen}(B', \mathit{fv}(\Gamma); \Theta)$, qui ne diffère de A uniquement pour les variables généralisées.

Le nom n n'étant pas dans $\mathit{dom}(A') = \mathit{dom}(A)$ (c'est un nom lié dans `def` D `in` P), nous avons par induction : $\mathbf{\Delta}; \mathbf{I}; \Gamma + n : \sigma + A' \vdash D :: B'; \emptyset; \Theta$, $\mathbf{\Delta}; \mathbf{I}; \Gamma + n : \sigma + A' \vdash P$ (en utilisant le lemme B.1.8(1) et l'hypothèse d'induction), $A' = \mathit{Gen}(B', \mathit{fv}(\Gamma + n : \sigma), \Theta)$ (puisque toute variable de type ou de type de nom de $n : \sigma$ qui n'est pas dans $\mathit{fv}(\Gamma)$ n'est pas une variable de B' qui est généralisée), et $\mathit{dom}(A') \cap \mathit{dom}(\Gamma + n : \sigma) = \emptyset$ (puisque le domaine de A' ne contient que des noms liés de `def` D `in` P).

Nous concluons par la règle Def.

Par, Go Immédiat par induction.

Nu Immédiat par induction, puisque le nom $\mathbf{d}$ ne peut être présent dans $n : \sigma$ puisqu'il est lié et qu'il n'est pas dans $\mathit{fv}(\Gamma)$.

Nil Immédiat.

Join Le nom n ne peut être un des u_i ou v_j puisque ceux-ci sont liés dans la règle de réaction et ne sont pas présents dans Γ (ils ne sont pas dans $\mathit{dom}(\Gamma)$, et ils ne peuvent être présents dans un type). La condition sur Θ n'est pas modifiée par l'ajout de l'association $n : \sigma$. Nous concluons donc par induction et par la règle Join.

Top Immédiat.

And, Loc Immédiat par induction.

Configuration, Soup-Loc Immédiat par induction.

Chem-Def Nous procédons comme dans le cas Def, en renommant Γ en Γ' de telle sorte que les variables généralisées de Γ' soient distinctes des variables libres de σ (nous rappelons que nous identifions les environnements de typage dont les variables généralisées ont été renommées injectivement en des variables fraîches).

□

Lemme B.1.12 *Soit Γ un environnement de typage. Si nous avons $\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash D :: B; \Delta_1; \Theta$ et $A = \mathit{Gen}(B, \mathit{fv}(\Gamma), \Theta) \subseteq \Gamma$, alors il existe un ensemble Θ' tel que $\Theta' \subseteq \mathit{fv}(\Gamma)$, $\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash D :: B; \Delta_1; \Theta'$ et $A = \mathit{Gen}(B, \mathit{fv}(\Gamma), \Theta')$.*

Preuve: Nous considérons $\Theta' = \Theta \cap \mathit{fv}(\Gamma)$. Nous prouvons par induction sur la dérivation de typage de la définition que $\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash D :: B; \Delta_1; \Theta'$, puis nous prouverons que la généralisation obtenue est la même. Ceci est immédiat pour tous les cas excepté la règle Join. Dans le cas Join, nous considérons une variable β qui est partagée par deux associations, donc présente dans Θ. Si l'une de ces associations est l'association d'un canal redéfinissable, alors par définition du sous-typage et comme nous ne considérons que des types bien formés (donc les types des canaux redéfinissables sont monomorphes), la même association est présente dans Γ. Par conséquent β est libre dans Γ, et est donc présent dans Θ', et la condition pour la règle Join est satisfaite. Sinon les deux associations sont présentes dans B et comme β est dans Θ, β est également libre dans A. Puisque nous avons $A \subseteq \Gamma$, β est libre dans Γ, par conséquent β est dans Θ' et la condition pour la règle Join est satisfaite.

Nous montrons maintenant que la généralisation est la même, en prouvant que les mêmes variables sont généralisées, c'est à dire que $\mathit{fv}(B) \setminus (\mathit{fv}(\Gamma) \cup \Theta) = \mathit{fv}(B) \setminus (\mathit{fv}(\Gamma) \cup \Theta')$. Comme nous avons $\Theta' \subseteq \Theta$, nous avons immédiatement $\mathit{fv}(B) \setminus (\mathit{fv}(\Gamma) \cup \Theta) \subseteq \mathit{fv}(B) \setminus (\mathit{fv}(\Gamma) \cup \Theta')$. Nous prouvons l'inclusion réciproque. Soit $\beta \in \mathit{fv}(B)$ tel que β n'est libre ni dans Γ, ni dans Θ'. Si β est libre dans Θ, alors il est n'est pas généralisé, donc il est libre dans A, donc dans Γ, ce qui est impossible. Par conséquent β n'est pas libre dans Θ. □

Lemme B.1.13 (Association de définitions) *Si nous avons $\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D}_1 : \Delta_1$ et $\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D}_2 : \Delta_2$ avec $\mathit{dsn}(\mathcal{D}_1) \cap \mathit{dsn}(\mathcal{D}_2) = \emptyset$, alors nous avons $\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D}_1, \mathcal{D}_2 : \Delta_1 \cup \Delta_2$.*

Preuve: Par hypothèse, nous avons deux dérivations de typage de la forme :

$$\frac{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D}_1 :: B_1; \Delta_1; \Theta_1 \qquad A_1 = \mathit{Gen}(B_1, \mathit{fv}(\Gamma), \Theta_1) \qquad A_1 \subseteq \Gamma}{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D}_1 : \Delta_1}$$

$$\frac{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D}_2 :: B_2; \Delta_2; \Theta_2 \qquad A_2 = \mathit{Gen}(B_2, \mathit{fv}(\Gamma), \Theta_2) \qquad A_2 \subseteq \Gamma}{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D}_2 : \Delta_2}$$

Nous voulons montrer que :

$$\frac{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D}_1, \mathcal{D}_2 :: B_1 \oplus B_2; \Delta_1 \cup \Delta_2; \Theta \qquad A = \mathit{Gen}(B_1 \oplus B_2, \mathit{fv}(\Gamma), \Theta) \qquad A \subseteq \Gamma}{\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D}_1, \mathcal{D}_2 : \Delta_1 \cup \Delta_2}$$

Nous commençons par montrer que $B_1 \oplus B_2$ existe. C'est le cas, puisque par le lemme B.1.3(3), nous avons :

$$\mathit{dom}(B_1) \cap \mathit{dom}(B_2) = \mathit{dsn}(\mathcal{D}_1) \cap \mathit{dsn}(\mathcal{D}_2) = \emptyset$$

Nous devons maintenant trouver un Θ permettant de typer les deux définitions.

Pour ce faire, nous utilisons le lemme B.1.12 deux fois, ce qui nous donne Θ'_1 et Θ'_2 ainsi que les dérivations de typage :

$$\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D}_1 :: B_1; \Delta_1; \Theta'_1$$
$$\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D}_2 :: B_2; \Delta_2; \Theta'_2$$

Le lemme B.1.12 nous indique également que $A_1 = \mathit{Gen}(B_1, \mathit{fv}(\Gamma), \Theta'_1) \subseteq \Gamma$, avec $\Theta'_1 \subseteq \mathit{fv}(\Gamma)$ (avec la même propriété pour A_2).

Une induction immédiate sur le typage des définitions nous donne :

$$\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D}_1 :: B_1; \Delta_1; \Theta'_1 \cup \Theta'_2$$
$$\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash \mathcal{D}_2 :: B_2; \Delta_2; \Theta'_1 \cup \Theta'_2$$

Pour conclure, puisque $A = A_1 \cup A_2$, nous montrons que $A_1 = \mathit{Gen}(B_1, \mathit{fv}(\Gamma), \Theta'_1 \cup \Theta'_2)$ et que $A_2 = \mathit{Gen}(B_2, \mathit{fv}(\Gamma), \Theta'_1 \cup \Theta'_2)$. C'est le cas puisque $\Theta'_1 \subseteq \mathit{fv}(\Gamma)$, $\Theta'_2 \subseteq \mathit{fv}(\Gamma)$, et puisque nous avons $A_1 = \mathit{Gen}(B_1, \mathit{fv}(\Gamma), \Theta'_1)$ et $A_2 = \mathit{Gen}(B_2, \mathit{fv}(\Gamma), \Theta'_2)$. □

Nous montrons maintenant que la typabilité est stable par équivalence structurelle.

Preuve du lemme 4.6.2: Nous montrons ce lemme pour chaque étape élémentaire définie dans la figure 4.4. L'extension à la fermeture réflexive, transitive et symétrique est immédiate.

Nous procédons par cas selon l´équivalence $\mathcal{S} \equiv \mathcal{S}'$.

Str-α Par le lemme B.1.10, il existe une substitution bien formée θ telle que $\theta(\Gamma) \vdash \mathcal{S}'$. L'environnement de typage Γ étant bien formé, l'environnement $\theta(\Gamma)$ est aussi bien formé.

Tree Nous avons la transformation suivante (sans mentionner le reste de la configuration qui ne participe pas à l'étape d'équivalence structurelle) :

$$\frac{\forall \beta \in \mathit{loc}(S), a \notin \beta \qquad G = \mathit{Lookup}(F, I, \Delta, a)}{\alpha b \left[a \left[\mathcal{D} : \mathcal{P} \right]^{\Delta, I}, \mathcal{D}_0 : \mathcal{P}_0 \right]^{\Delta', I', F} \equiv \alpha b \left[\mathcal{D}_0 : \mathcal{P}_0 \right]^{\Delta', I', F} \parallel \alpha b a \left[\mathcal{D} : \mathcal{P} \right]^{\Delta, I, G}} \text{ [Tree]}$$

Nous remarquons que pour tout nom $\mathbf{n}$ tel que $\mathbf{n} : \langle \tau \rangle^+_w \in \Gamma$, nous avons nécessairement $w = \mathbf{n}$ puisque Γ est bien formé. Dans la suite, nous simplifions par conséquent les hypothèses de la règle de typage Loc.

Nous détaillons maintenant les dérivations de typage pour les locations impliquées dans l'étape d'équivalence structurelle, en ne mentionnant pas certaines hypothèses de typage qui ne sont pas modifiées par cette étape :

$$\{\alpha\} \text{ forme un arbre de racine c} \tag{B.1}$$
$$\{\alpha\} + \alpha ab \text{ forme un arbre de racine c} \tag{B.2}$$
$$\Delta'; I'; \Gamma \vdash \mathcal{D}_0 :: B_0; \Delta'; \Theta \tag{B.3}$$
$$A_b = \mathit{Gen}(B_0 \oplus B + a : \mathit{loc}(\Delta \cup I), \mathit{fv}(\Gamma), \Theta) \tag{B.4}$$
$$A_b \subseteq \Gamma \tag{B.5}$$
$$A_0 = \mathit{Gen}(B_0, \mathit{fv}(\Gamma), \Theta) \tag{B.6}$$
$$A_0 \subseteq \Gamma \tag{B.7}$$
$$A_a = \mathit{Gen}(B, \mathit{fv}(\Gamma); \Theta_a) \tag{B.8}$$
$$A_a \subseteq \Gamma \tag{B.9}$$

La configuration repliée est typée de la manière suivante :

$$
\dfrac{
  (B.1) \qquad
  \dfrac{
    \dfrac{
      (B.4, B.5) \qquad
      \dfrac{
        (B.3) \qquad
        \dfrac{
          \Delta; I; \Gamma \vdash \mathcal{D} :: B; \Delta; \Theta \qquad
          \Delta; I; \Gamma \vdash \mathcal{P} \qquad \Gamma \vdash a : loc(\Delta \cup I) \qquad I \subseteq (\Delta' \cup I')
        }{
          \Delta'; I'; \Gamma \vdash a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I} :: B + a : loc(\Delta \cup I); \emptyset; \Theta
        }\,[\textsc{Loc}]
        \;\vdots
      }{
        \Delta'; I'; \Gamma \vdash a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I}, \mathcal{D}_0 :: B_0 \oplus B + a : loc(\Delta \cup I); \Delta'; \Theta
      }\,[\textsc{And}]
      \;\vdots
    }{
      \Delta'; I'; \Gamma \vdash a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I}, \mathcal{D}_0 : \Delta'
    }\,[\textsc{Chem-Def}]
  }{
    \Gamma \vdash \alpha b \left[a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I}, \mathcal{D}_0 : \mathcal{P}_0 \right]^{\Delta',I',F}
  }\,[\textsc{Soup-Loc}]
}{
  \Gamma \vdash \alpha b \left[a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I}, \mathcal{D}_0 : \mathcal{P}_0 \right]^{\Delta',I',F}
}\,[\textsc{Configuration}]
$$

La configuration dépliée est typée de la manière suivante :

$$
\dfrac{
  (B.2) \qquad
  \dfrac{
    \dfrac{
      (B.3, B.6, B.7)
    }{
      \Delta'; I'; \Gamma \vdash \mathcal{D}_0 : \Delta'
    }\,[\textsc{Chem-Def}]
  }{
    \Gamma \vdash \alpha b\,[\mathcal{D}_0 : \mathcal{P}_0]^{\Delta',I',F}
  }\,[\textsc{Soup-Loc}]
  \qquad
  \dfrac{
    \dfrac{
      (B.8, B.9) \qquad \Delta; I; \Gamma \vdash \mathcal{D} :: B; \Delta; \Theta_a
    }{
      \Delta; I; \Gamma \vdash \mathcal{D} : \Delta
    }\,[\textsc{Chem-Def}]
    \qquad a : loc(\Delta \cup I) \in \Gamma \qquad \Delta; I; \Gamma \vdash \mathcal{P}
  }{
    \Gamma \vdash \alpha ba\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,G}
  }\,[\textsc{Soup-Loc}]
  \;\vdots
  \qquad I \subseteq I' \cup \Delta'
}{
  \Gamma \vdash \alpha b\,[\mathcal{D}_0 : \mathcal{P}_0]^{\Delta',I',F} \;\parallel\; \alpha ba\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,G}
}\,[\textsc{Configuration}]
$$

Nous détaillons maintenant comment passer d'une dérivation de typage à l'autre.

Les hypothèses (B.1) et (B.2) sont équivalentes puisque la location a est gelée.

Nous montrons que dans la première dérivation nous avons nécessairement $a : loc(\Delta \cup I) \in \Gamma$. Par hypothèse de la règle Loc pour a, toutes les variables de $\Delta \cup I$ sont libres dans Γ. Nous déduisons donc des hypothèses (B.4) et (B.5) que la liaison $a : loc(\Delta \cup I)$ est présente dans Γ et n'est pas généralisée.

Nous remarquons également que les noms définis statiques de $\mathcal{D}_0$ et $\mathcal{D}$ étant définis dans deux locations différentes, ils sont forcément tous distincts et différents de a. Nous avons donc $A_b = A_0 \uplus \{a : loc(\Delta \cup I)\} \uplus Gen(B, fv(\Gamma), \Theta)$.

Pour passer de la dérivation de la configuration repliée à la configuration dépliée, nous procédons comme suit.

La condition non indiquée pour la règle Soup-Loc de la location a qui requiert tout nom de Δ d'être un nom dynamique, et qui requiert la présence de ce nom $\mathbf{n}$ dans Γ avec une association de la forme $\mathbf{n} : \langle\tau\rangle^{+}_{\mathbf{n}}$, est satisfaite par l'hypothèse de la règle Loc (qui requiert tout nom de Δ d'être présent dans Γ avec un type redéfinissable) et parce que Γ est bien formé (donc toute association de nom redéfinissable est de la forme $\mathbf{n} : \langle\tau\rangle^{+}_{\mathbf{n}}$).

En ce qui concerne le typage de la définition de b, l'hypothèse (B.7) se déduit immédiatement de la propriété $A_b = A_0 \uplus \{a : loc(\Delta \cup I)\} \uplus Gen(B, fv(\Gamma), \Theta)$ et de l'hypothèse (B.4). En ce qui concerne le typage de la définition de a, nous prenons tout simplement $\Theta_a = \Theta$ et déduisons (B.9) de la propriété sur A_b et de (B.4).

Nous décrivons maintenant comment passer de la configuration dépliée à la configuration repliée.

Nous montrons que nous avons une dérivation de typage pour :

$$\Delta'; I'; \Gamma \vdash a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I} :: B + a : loc(\Delta \cup I); \emptyset; \Theta_a$$

Nous utilisons pour ce faire la règle Loc. Par le lemme B.1.4, nous avons $\Delta' \cup I' \subseteq dom(\Gamma)$. Nous utilisons le fait que Γ est bien formé pour en déduire que pour tout $n \in \Delta \cup I$, nous avons $n = \mathbf{n}$ et $\mathbf{n} : \langle\tau\rangle^{+}_{\mathbf{n}} \in \Gamma$. Nous utilisons la dérivation de typage $\Delta; I; \Gamma \vdash \mathcal{D} :: B; \Delta; \Theta_a$, la dérivation de typage non mentionnée $\Delta; I; \Gamma \vdash \mathcal{P}$, l'hypothèse de la règle Configuration $I \subseteq \Delta' \cup I'$ et le fait que $a : loc(\Delta \cup I) \in \Gamma$ pour conclure par la règle Loc.

Nous montrons maintenant que $Gen(B + a : loc(\Delta \cup I), fv(\Gamma), \Theta_a) \subseteq \Gamma$. Comme nous l'avons vu auparavant, nous avons $Gen(B + a : loc(\Delta \cup I), fv(\Gamma), \Theta_a) = Gen(B, fv(\Gamma), \Theta_a) \uplus \{a :$

$loc(\Delta \cup I)\}$. Nous concluons par (B.9) et le fait que $a : loc(\Delta \cup I) \in \Gamma$. Nous pouvons donc appliquer la règle de typage CHEM-DEF, et nous obtenons :

$$\Delta'; I'; \Gamma \vdash a\,[\mathcal{D} : \mathcal{P}]^{\Delta, I} : \emptyset$$

Nous avons également par hypothèse :

$$\Delta'; I'; \Gamma \vdash \mathcal{D}_0 : \Delta'$$

Nous montrons maintenant que nous avons $dsn(\mathcal{D}_0) \cap dsn(a\,[\mathcal{D} : \mathcal{P}]^{\Delta, I}) = \emptyset$. C'est le cas puisque ces noms définis statiques sont définis dans des locations différentes. Nous pouvons donc appliquer le lemme B.1.13 pour obtenir :

$$\Delta'; I'; \Gamma \vdash a\,[\mathcal{D} : \mathcal{P}]^{\Delta, I}, \mathcal{D}_0 : \Delta'$$

Nous pouvons alors conclure par les règles SOUP-LOC et CONFIGURATION.

□

Lemme B.1.14 *Soient* $\mathbf{\Delta}$ *et* $\mathbf{I}$ *deux ensembles de noms dynamiques ou de variables de types de noms,* Γ *un environnement de typage, et pour* $i \in [1..n]$, u_i *des variables,* m_i *des noms,* τ_i *et* τ'_i *des types. Soit* θ *une substitution bien formée telle que :*

1. $dom(\theta) \subseteq \bigcup_i (\{u_i\} \cup fv(\tau_i))$ *;*
2. $\theta(u_i) = m_i$ *pour tout* $i \in [1..n]$ *;*
3. $dom(\theta) \cap dom(\Gamma) = \emptyset$ *;*
4. $fn(\theta(\mathbf{\Delta} \cup \mathbf{I})) \subseteq fn(\theta(\Gamma))$ *;*
5. $fnv(\theta(\mathbf{\Delta} \cup \mathbf{I})) \subseteq fnv(\theta(\Gamma))$.

Si nous avons $\mathbf{\Delta}; \mathbf{I}; \Gamma + (u_i : \tau_i)^{i \in [1..n]} \vdash P$ *et* $(\theta(\Gamma) \vdash_{\text{NAME}} v_i : \tau'_i)^{i \in [1..n]}$, *avec* $\tau'_i \leq \theta(\tau_i)$ *et* $fn(ran(\theta)) \cap bn(P) = \emptyset$ *; alors nous avons* $\theta(\mathbf{\Delta}); \theta(\mathbf{I}); \theta(\Gamma) \vdash \theta(P)$.

Preuve: Nous remarquons tout d'abord que ce lemme ne s'applique qu'à des processus P ne contenant par conséquent aucun message résolu.

Dans cette preuve nous notons $\widetilde{u_i} : \widetilde{\tau_i}$ pour $(u_i : \tau_i)^{i \in [1..n]}$.

Nous procédons par induction sur la dérivation de typage $\mathbf{\Delta}; \mathbf{I}; \Gamma + \widetilde{u_i} : \widetilde{\tau_i} \vdash P$. La dérivation de typage obtenue par application du lemme dans le cas des noms est de la forme $\theta(\Gamma) \vdash \theta(n) : \theta(\tau)$. Dans le cas des définitions, elle est de la forme $\theta(\mathbf{\Delta}); \theta(\mathbf{I}); \theta(\Gamma) \vdash \theta(D) :: \theta(B); \theta(\Delta_1); fv(\theta(\Theta))$. Nous ajoutons une hypothèse supplémentaire dans le cas des définitions pour pouvoir appliquer le lemme : pour toute variable de type ou de type de nom β de B, nous avons $\beta \notin \Theta \implies \beta \notin (dom(\theta) \cup fv(ran(\theta)))$.

Afin d'utiliser l'hypothèse d'induction, nous devons dans certains cas étendre l'environnement $\theta(\Gamma)$ pour les jugements $(\theta(\Gamma) \vdash_{\text{NAME}} v_i : \tau'_i)^{i \in [1..n]}$. Nous utilisons pour ce faire le lemme B.1.11, en nous assurant que l'association ajoutée n'entre pas en conflit avec les noms déjà associés dans $\theta(\Gamma)$.

Nous remarquons que l'hypothèse sur les noms liés de P est toujours vraie pour un sous-processus si elle est vraie pour le processus.

Lors de l'utilisation de l'hypothèse d'induction, nous ne vérifions les hypothèses (4,5) que dans le cas où les ensembles $\mathbf{\Delta}$ ou $\mathbf{I}$ sont modifiés, ou si Γ possède moins d'associations.

Enfin, nous remarquons que ce lemme est différent du lemme B.1.8(2) puisque nous n'exigeons pas que θ soit injective.

NAME Nous avons :

$$\frac{m : \forall \widetilde{\alpha}\widetilde{\delta}.\tau' \in \Gamma + \widetilde{u_i} : \widetilde{\tau_i} \qquad \tau = Inst(\forall \widetilde{\alpha}\widetilde{\delta}.\tau')}{\Gamma + \widetilde{u_i} : \widetilde{\tau_i} \vdash m : \tau}\ [\text{NAME}]$$

Si le nom m est un des u_i, alors nous avons $\tau = \tau' = \tau_i$ et par hypothèse $\theta(\Gamma) \vdash m_i : \tau'_i$ avec $\tau'_i \leq \theta(\tau_i)$. Nous utilisons alors la règle de typage SUB pour obtenir $\theta(\Gamma) \vdash m_i : \theta(\tau_i)$. Nous concluons en remarquant que $m_i = \theta(u_i)$.

Sinon, le nom m n'est pas un u_i. Nous devons alors montrer que nous avons $\theta(\Gamma) \vdash \theta(m) : \theta(\tau)$. Comme m n'est pas un u_i, nous avons nécessairement $m : (\sigma = \forall \widetilde{\alpha}\widetilde{\delta}.\tau') \in \Gamma$ par hypothèse de la règle de typage NAME. Dans ce cas, nous avons $m \in dom(\Gamma)$, donc par

l'hypothèse 3 nous avons $m \notin dom(\theta)$. Nous supposons que les variables généralisées $\widetilde{\alpha}$ et $\widetilde{\delta}$ ont été renommées pour être distinctes des variables du domaine et de l'image de θ. Nous avons donc $m : \forall\widetilde{\alpha}\widetilde{\delta}.\theta(\tau') \in \theta(\Gamma)$.

Soit φ l'instanciation de τ' en τ. Comme dans le cas NAME du lemme B.1.8(2), nous montrons que nous avons $\theta(\varphi(\theta(\tau'))) = \theta(\varphi(\tau')) = \theta(\tau)$. Soit β une variable. Si cette variable est dans le domaine de φ (c'est un des $\widetilde{\alpha}$ ou un des $\widetilde{\delta}$), alors $\theta(\varphi(\theta(\beta))) = \theta(\varphi(\beta))$ puisque les noms généralisés ne sont pas dans le domaine de θ. Sinon, β n'est pas dans le domaine de φ. Comme l'image de θ est distincte du domaine de φ, nous avons $\theta(\varphi(\theta(\beta))) = \theta(\theta(\beta)) = \theta(\beta) = \theta(\varphi(\beta))$ (nous rappelons que les substitutions bien formées sont telles que $\theta \circ \theta = \theta$).

Par conséquent, $\theta \circ \varphi$ instancie $\theta(\tau')$ en $\theta(\tau)$. Nous concluons par la règle NAME.

SUB Nous avons :

$$\frac{\Gamma + \widetilde{u_i} : \widetilde{\tau_i} \vdash m : \tau \qquad \tau \leq \tau'}{\Gamma + \widetilde{u_i} : \widetilde{\tau_i} \vdash m : \tau'} \text{ [SUB]}$$

Par hypothèse d'induction sur la dérivation $\Gamma + \widetilde{u_i} : \widetilde{\tau_i} \vdash m : \tau$, nous obtenons : $\theta(\Gamma) \vdash \theta(m) : \theta(\tau)$. Par le lemme B.1.7, nous avons également $\theta(\tau) \leq \theta(\tau')$. Nous concluons par la règle de typage SUB.

TUPLE Immédiat par induction.

MSG Nous avons la dérivation de typage suivante :

$$\frac{\Gamma + \widetilde{u_i} : \widetilde{\tau_i} \vdash n : \langle\widetilde{\tau}\rangle_{\mathbf{\Delta}\cup\mathbf{I}} \qquad \Gamma + \widetilde{u_i} : \widetilde{\tau_i} \vdash \widetilde{m} : \widetilde{\tau} \qquad \mathbf{\Delta};\mathbf{I};\Gamma + \widetilde{u_i} : \widetilde{\tau_i} \vdash P}{\mathbf{\Delta};\mathbf{I};\Gamma + \widetilde{u_i} : \widetilde{\tau_i} \vdash n\langle\widetilde{m}\rangle;P} \text{ [MSG]}$$

Par hypothèse d'induction, nous avons donc :

$$\begin{array}{rcl} \theta(\Gamma) & \vdash & \theta(n) : \langle\theta(\widetilde{\tau})\rangle_{\theta(\mathbf{\Delta})\cup\theta(\mathbf{I})} \\ \theta(\Gamma) & \vdash & \theta(\widetilde{m}) : \theta(\widetilde{\tau}) \\ \theta(\mathbf{\Delta});\theta(\mathbf{I});\theta(\Gamma) & \vdash & \theta(P) \end{array}$$

Nous concluons par la règle MSG.

R-MSG Ce cas ne peut avoir lieu puisqu'il ne concerne que des messages résolus. Nous remarquons que la seule difficulté de ce cas est de montrer que nous avons $\theta(n) \in dln(a)$, ou $\theta(n) \in dln(\theta(a))$, ce qui exigerait que le type des locations indique l'ensemble des noms qui y sont définis localement.

DEF Nous avons la dérivation de typage :

$$\frac{\begin{array}{c}\mathbf{\Delta};\mathbf{I};\Gamma + \widetilde{u_i} : \widetilde{\tau_i} + A \vdash D :: B;\emptyset;\Theta \qquad A = Gen(B, fv(\Gamma + \widetilde{u_i} : \widetilde{\tau_i}), \Theta) \\ \mathbf{\Delta};\mathbf{I};\Gamma + \widetilde{u_i} : \widetilde{\tau_i} + A \vdash P \qquad dom(A) \cap dom(\Gamma + \widetilde{u_i} : \widetilde{\tau_i}) = \emptyset\end{array}}{\mathbf{\Delta};\mathbf{I};\Gamma + \widetilde{u_i} : \widetilde{\tau_i} \vdash \texttt{def } D \texttt{ in } P} \text{ [DEF]}$$

Par hypothèse de typage de la règle DEF, nous savons que aucun des u_i n'est dans le domaine de A (qui est aussi $dsn(D)$ par le lemme B.1.3(3)). Par conséquent nous avons $dom(\theta) \cap dom(A) = \emptyset$. De plus, par l'hypothèse sur les noms liés et θ, nous avons également $fn(ran(\theta)) \cap dsn(D) = \emptyset$, donc aucune capture de nom ne peut avoir lieu. Par conséquent, nous avons $\theta(\texttt{def } D \texttt{ in } P) = \texttt{def } \theta(D) \texttt{ in } \theta(P)$.

Nous voulons montrer : $\theta(\mathbf{\Delta});\theta(\mathbf{I});\theta(\Gamma) \vdash \texttt{def } \theta(D) \texttt{ in } \theta(P)$.

Pour utiliser l'hypothèse d'induction sur la dérivation de typage de la définition, nous devons tout d'abord montrer que nous avons la propriété : $\beta \in fv(B) \wedge \beta \notin \Theta \implies \beta \notin (dom(\theta) \cup fv(ran(\theta)))$. Nous commençons par étendre Θ de telle sorte que les variables de B qui ne sont pas dans Θ sont nécessairement généralisées. Soit $\Theta' = \Theta \cup fv(\Gamma + \widetilde{u_i} : \widetilde{\tau_i})$. Par définition de l'opérateur de généralisation, nous avons : $Gen(B, fv(\Gamma + \widetilde{u_i} : \widetilde{\tau_i}), \Theta') = Gen(B, fv(\Gamma + \widetilde{u_i} : \widetilde{\tau_i}), \Theta)$.

Nous prouvons de manière immédiate par induction sur la dérivation de typage d'une définition que si $\Theta \subseteq \Theta'$ et si $\mathbf{\Delta};\mathbf{I};\Gamma \vdash D :: B;\Delta_1;\Theta$ alors nous avons $\mathbf{\Delta};\mathbf{I};\Gamma \vdash D :: B;\Delta_1;\Theta'$. Le seul cas non trivial est le cas JOIN, c'est à dire le cas de base, où il faut montrer que $fv(\tau) \cap fv(\tau') \subseteq \Theta \implies fv(\tau) \cap fv(\tau') \subseteq \Theta'$, ce qui est immédiat.

Nous avons donc une dérivation de typage :

$$\mathbf{\Delta};\mathbf{I};\Gamma + A + \widetilde{u_i} : \widetilde{\tau_i} \vdash D :: B;\emptyset;\Theta'$$

Notons $\widetilde{\beta}$ l'ensemble des variables de $fv(B)$ qui ne sont pas dans Θ', c'est à dire qui sont généralisées, mais qui sont libres dans $ran(\theta)$: $\widetilde{\beta} = (fv(B) \setminus \Theta') \cap fv(ran(\theta))$. Puisque nous avons $fv(\widetilde{\tau_i}) \subseteq \Theta'$ par définition de Θ', nous avons $\widetilde{\beta} \cap fv(\widetilde{\tau_i}) = \emptyset$ par définition de $\widetilde{\beta}$. Comme toutes les variables de types ou de types de noms du domaine de θ sont présentes dans $fv(\widetilde{\tau_i})$ par l'hypothèse (1), nous avons $\widetilde{\beta} \cap dom(\theta) = \emptyset$. De plus, les variables $\widetilde{\beta}$ étant généralisées dans A, elle ne sont pas libres dans $\Gamma + A + \widetilde{u_i} : \widetilde{\tau_i}$. Nous renommons donc ces variables $\widetilde{\beta}$ en des variables $\widetilde{\beta'}$ qui ne sont pas présentes dans la dérivation de typage de D ni dans $fv(ran(\theta)) \cup dom(\theta)$ en utilisant le lemme B.1.8(2) (nous rappelons qu'aucun message résolu n'est présent dans D). Nous obtenons alors la dérivation :

$$\mathbf{\Delta}; \mathbf{I}; \Gamma + A + \widetilde{u_i} : \widetilde{\tau_i} \vdash D :: B'; \emptyset; \Theta'$$

Les variables $\widetilde{\beta'}$ étant fraîches, l'environnement $B' \stackrel{\text{def}}{=} B\{^{\widetilde{\beta'}}/_{\widetilde{\beta}}\}$ se généralise en A, au renommage des variables généralisées près. De plus, aucune variable libre de B' qui n'est pas libre dans Θ' n'est présente dans $dom(\theta) \cup fv(ran(\theta))$.

Nous pouvons par conséquent utiliser l'hypothèse d'induction sur la dérivation de typage de la définition, et nous obtenons :

$$\theta(\mathbf{\Delta}); \theta(\mathbf{I}); \theta(\Gamma) + \theta(A) \vdash \theta(D) :: \theta(B'); \emptyset; fv(\theta(\Theta'))$$

Nous obtenons également :

$$\theta(\mathbf{\Delta}); \theta(\mathbf{I}); \theta(\Gamma) + \theta(A) \vdash \theta(P)$$

Soit $A' = Gen(\theta(B'), \theta(fv(\Gamma)), fv(\theta(\Theta')))$, nous montrons maintenant que $A' = \theta(A)$.

Nous commençons par montrer qu'ils ont le même domaine. C'est le cas puisque $dom(A) \cap dom(\theta) = \emptyset$.

Nous montrons que les mêmes variables sont généralisées, c'est à dire que :

$$fv(\theta(B')) \setminus (fv(\theta(\Gamma)) \cup fv(\theta(\Theta'))) = fv(B') \setminus (fv(\Gamma + \widetilde{u_i} : \widetilde{\tau_i}) \cup \Theta')$$

Par définition de Θ', nous avons $fv(\Gamma + \widetilde{u_i} : \widetilde{\tau_i}) \subseteq \Theta'$, et $fv(\theta(\Gamma)) \subseteq fv(\theta(\Theta'))$. Nous devons donc montrer que :

$$fv(\theta(B')) \setminus fv(\theta(\Theta')) = fv(B') \setminus \Theta'$$

Soit β une variable du deuxième ensemble. Grâce au renommage de B en B', nous avons $\beta \notin dom(\theta)$, donc $\beta \in fv(\theta(B'))$ (puisque $\beta \in fv(B')$), ainsi que $\beta \notin fv(ran(\theta))$, donc $\beta \notin fv(\theta(\Theta'))$ (puisque $\beta \notin \Theta'$). La variable β est donc dans le premier ensemble.

Nous considérons maintenant une variable β présente dans le premier ensemble. Nous avons alors soit $\beta \notin fv(ran(\theta))$, soit $\beta \in fv(ran(\theta))$. Dans le premier cas, comme nous avons $\beta \in fv(\theta(B'))$ alors nécessairement $\beta \in fv(B')$ et $\theta(\beta) = \beta$. Si nous avons $\beta \in \Theta'$, alors β est libre dans $\theta(\Theta')$, ce qui est impossible puisque β est dans le premier ensemble. Par conséquent, nous avons $\beta \notin \Theta'$, et β est dans le deuxième ensemble.

Nous supposons maintenant qu'il existe un β dans le premier ensemble tel que $\beta \in fv(ran(\theta))$. Il existe donc un $\beta' \in dom(\theta) \cap fv(B')$ tel que $\beta \in fv(\theta(\beta'))$ et $\beta' \neq \beta$. La variable β' est donc dans le domaine de θ, c'est donc une variable libre dans au moins un des τ_i par l'hypothèse (1), elle est donc présente dans Θ' par définition de ce dernier. Nous avons donc $\beta \in fv(\theta(\Theta'))$, ce qui est impossible puisque β est dans le premier ensemble. Nous en déduisons donc qu'il n'existe aucun β dans le premier ensemble tel que $\beta \in fv(ran(\theta))$.

Ainsi, nous avons $n : \forall \widetilde{\alpha}\widetilde{\delta}.\tau \in A$ si et seulement si $n : \forall \widetilde{\alpha}\widetilde{\delta}.\theta(\tau) \in A'$ puisque les variables généralisées sont les même et que le domaine et l'image de θ sont disjoints de ces variables généralisées. Nous avons donc $A' = \theta(A)$, donc :

$$\theta(\mathbf{\Delta}); \theta(\mathbf{I}); \theta(\Gamma) + A' \vdash \theta(D) :: \theta(B'); \emptyset; fv(\theta(\Theta'))$$

et

$$\theta(\mathbf{\Delta}); \theta(\mathbf{I}); \theta(\Gamma) + A' \vdash \theta(P)$$

De plus, comme l'ensemble $dom(A) = dom(\theta(A)) = dsn(D)$ est disjoint des noms libres de l'image de θ, nous avons $dom(A') \cap dom(\theta(\Gamma)) = \emptyset$.

Nous concluons par la règle Def.

Par, Go Immédiat par induction.

Nu Nous avons :

$$\frac{\mathbf{d} \notin fn(\Gamma + \widetilde{u_i} : \widetilde{\tau_i}) \qquad \mathbf{\Delta}; \mathbf{I}; \Gamma + \widetilde{u_i} : \widetilde{\tau_i} + \mathbf{d} : \langle \tau \rangle^+_{\mathbf{d}} \vdash P}{\mathbf{\Delta}; \mathbf{I}; \Gamma + (u_i : \tau_i)^{i \in [1..n]} \vdash \nu \mathbf{d}.P} \text{ [Nu]}$$

Le nom $\mathbf{d}$ étant un nom de canal dynamique, il ne peut être dans le domaine de θ par l'hypothèse (1). L'hypothèse (3) est donc satisfaite, et nous pouvons appliquer l'hypothèse d'induction pour obtenir :

$$\theta(\mathbf{\Delta}); \theta(\mathbf{I}); \theta(\Gamma) + \mathbf{d} : \langle \theta(\tau) \rangle^+_{\mathbf{d}} \vdash \theta(P)$$

Le nom $\mathbf{d}$ étant lié dans le processus final, il ne peut donc être libre dans l'image de θ. Comme il n'est pas non plus libre dans Γ, nous avons donc $\mathbf{d} \notin fn(\theta(\Gamma))$.

Nous concluons par la règle Nu.

Nil Immédiat.

Join Nous avons :

$$\text{[Join]} \frac{\begin{array}{c} \mathbf{\Delta}; \mathbf{I}; \Gamma + \widetilde{u_i} : \widetilde{\tau_i} + (\widetilde{v_i} : \widetilde{\tau_i^x})^i + (\widetilde{y_j} : \widetilde{\tau_j^n})^j \vdash P \qquad \Gamma + \widetilde{u_i} : \widetilde{\tau_i} \vdash \mathtt{x}_i : \langle \widetilde{\tau_i^x} \rangle \\ \Gamma + \widetilde{u_i} : \widetilde{\tau_i} \vdash n_j : \langle \widetilde{\tau_j^n} \rangle^+_{w_j} \qquad \left(\bigcup_i \widetilde{v_i} \cup \bigcup_j \widetilde{y_j} \right) \cap dom(\Gamma + \widetilde{u_i} : \widetilde{\tau_i}) = \emptyset \\ \forall (n : \tau), (n' : \tau') \in (\{\mathtt{x}_i : \langle \widetilde{\tau_i^x} \rangle\} \cup \{n_j : \langle \widetilde{\tau_j^n} \rangle^+_{w_j}\}) \,.\, n \neq n' \implies fv(\tau) \cap fv(\tau') \subseteq \Theta \end{array}}{\mathbf{\Delta}; \mathbf{I}; \Gamma + \widetilde{u_i} : \widetilde{\tau_i} \vdash (\mathtt{x}_i \langle \widetilde{v_i} \rangle)^i \mid (n_j \langle \widetilde{y_j} \rangle)^j \triangleright P :: (\mathtt{x}_i : \langle \widetilde{\tau_i^x} \rangle)^i ; \bigcup_j \{n_j\} ; \Theta}$$

Les noms $\widetilde{v_i}$ et $\widetilde{y_j}$ étant liés dans la règle de réaction, ils ne sont donc pas libres dans l'image de θ. Il ne peut donc y avoir de capture de nom. De plus, les noms variables du domaine de θ sont inclus dans les $\widetilde{u_i}$ (nous rappelons que les noms variables sont des noms, à ne pas confondre avec les variables de types ou les variables de types de noms), donc par hypothèse de la règle de typage Join le domaine de θ est disjoint des noms $\widetilde{v_i}$ et $\widetilde{y_j}$. Par conséquent, l'hypothèse (3) est satisfaite pour θ et l'environnement $\Gamma + (\widetilde{v_i} : \widetilde{\tau_i^x})^i + (\widetilde{y_j} : \widetilde{\tau_j^n})^j$. Nous avons également :

$$\left(\bigcup_i \widetilde{v_i} \cup \bigcup_j \widetilde{y_j} \right) \cap dom(\theta(\Gamma)) = \emptyset$$

Nous remarquons que les noms $\mathtt{x}_i$ étant des noms de canaux statiques, ils ne sont pas dans le domaine de θ. Par hypothèse d'induction, nous avons donc :

$$\begin{array}{rcl} \theta(\mathbf{\Delta}); \theta(\mathbf{I}); \theta(\Gamma) + (\widetilde{v_i} : \widetilde{\theta(\tau_i^x)})^i + (\widetilde{y_j} : \widetilde{\theta(\tau_j^n)})^j & \vdash & \theta(P) \\ \theta(\Gamma) \vdash \mathtt{x}_i : \langle \widetilde{\theta(\tau_i^x)} \rangle & & \\ \theta(\Gamma) \vdash \theta(n_j) : \langle \widetilde{\theta(\tau_j^n)} \rangle^+_{\theta(w_j)} & & \end{array}$$

Nous montrons maintenant que $(\theta(n) : \theta(\tau)), (\theta(n') : \theta(\tau')) \in (\{\mathtt{x}_i : \langle \widetilde{\theta(\tau_i^x)} \rangle\} \cup \{\theta(n_j) : \langle \widetilde{\theta(\tau_j^n)} \rangle^+_{\theta(w_j)}\})$ avec $\theta(n) \neq \theta(n')$ implique nécessairement $fv(\theta(\tau)) \cap fv(\theta(\tau')) \subseteq fv(\theta(\Theta))$. Soit β une telle variable partagée. Soient β_1 et β_2 des variables libres de τ et τ' respectivement tels que $\beta \in fv(\theta(\beta_1))$ et $\beta \in fv(\theta(\beta_2))$ respectivement. Ces variables existent puisque β est présent à la fois dans $fv(\theta(\tau))$ et $fv(\theta(\tau'))$, et chacune peut être égale à β (nous n'exigeons pas qu'elles soient dans le domaine de θ).

Si $\beta_1 = \beta_2 = \beta$, nous avons alors nécessairement $\beta \in \Theta$ puisque β est une variable partagée de τ et τ', (et $\theta(n) \neq \theta(n')$ implique nécessairement $n \neq n'$), et nous avons $\beta \in fv(\theta(\beta))$, donc nous avons $\beta \in fv(\theta(\Theta))$. Sinon, une des variables β_1 ou β_2 est différente de β. Supposons, sans perte de généralité, qu'il s'agisse de β_1. Nous avons donc $\theta(\beta_1) \neq \beta_1$ (puisque sinon nous aurions $\beta_1 = \beta$), donc $\beta_1 \in dom(\theta)$. Comme nous avons $\beta_1 \in fv(B)$ et $\beta_1 \in dom(\theta)$, nous avons nécessairement $\beta_1 \in \Theta$ (sinon, par l'hypothèse supplémentaire sur les définitions, nous aurions $\beta \notin \Theta$ qui impliquerait $\beta \notin (dom(\theta) \cup fv(ran(\theta)))$, donc en particulier $\beta \notin dom(\theta)$, ce qui est impossible). Puisque nous avons $\beta_1 \in \Theta$ et $\beta \in fv(\theta(\beta_1))$, nous avons donc $\beta \in fv(\theta(\Theta))$.

Nous pouvons par conséquent appliquer la règle de typage JOIN pour obtenir le résultat souhaité :

$$\theta(\mathbf{\Delta});\theta(\mathbf{I});\theta(\Gamma)\vdash(\mathtt{x}_i\langle\widetilde{v_i}\rangle)^i\mid(\theta(n_j)\langle\widetilde{y_j}\rangle)^j\triangleright\theta(P)::(\mathtt{x}_i:\langle\theta(\widetilde{\tau_i^x})\rangle)^i;\bigcup_j\{\theta(n_j)\};fv(\theta(\Theta))$$

TOP Immédiat.

AND Le résultat est immédiat par induction, puisque la condition supplémentaire sur les définitions implique moins de variables de types pour chaque sous-dérivation.

LOC Nous avons :

$$\frac{\begin{array}{c}\forall n_i\in\Delta\ .\ n_i:\langle\tau_i^n\rangle^+_{w_i}\in\Gamma+\widetilde{u_i}:\widetilde{\tau_i}\\ \forall n_j\in I\ .\ n_j:\langle\tau_j^n\rangle^+_{w_j}\in\Gamma+\widetilde{u_i}:\widetilde{\tau_i}\qquad\mathbf{\Delta}'=\bigcup_i w_i\qquad\mathbf{I}'=\bigcup_j w_j\\ \mathbf{\Delta}';\mathbf{I}';\Gamma+\widetilde{u_i}:\widetilde{\tau_i}\vdash D::B;\Delta;\Theta\qquad\mathbf{\Delta}';\mathbf{I}';\Gamma+\widetilde{u_i}:\widetilde{\tau_i}\vdash P\qquad\mathbf{I}'\subseteq(\mathbf{\Delta}''\cup\mathbf{I}'')\end{array}}{\mathbf{\Delta}'';\mathbf{I}'';\Gamma+\widetilde{u_i}:\widetilde{\tau_i}\vdash a\,[D:P]^{\Delta,I}::B+a:loc(\mathbf{\Delta}'\cup\mathbf{I}');\emptyset;\Theta}\ [\text{LOC}]$$

Afin d'appliquer l'hypothèse d'induction, nous vérifions les hypothèses (4,5) pour $\mathbf{\Delta}$ et $\mathbf{I}$. Nous détaillons le cas de $\mathbf{\Delta}$, celui pour $\mathbf{I}$ est identique.

Soit $n:\langle\tau\rangle^+_w\in\Gamma+(u_i:\tau_i)^{i\in[1..n]}$. Si $n:\langle\tau\rangle^+_w\in\Gamma$, alors n est dans le domaine de Γ. Ainsi, par l'hypothèse (3), nous avons $\theta(n)=n$, et $n:\langle\theta(\tau)\rangle^+_{\theta(w)}\in\theta(\Gamma)$. Sinon le nom n est un des $\widetilde{u_i}$, et par hypothèse nous avons $\theta(\Gamma)\vdash_{\text{NAME}}\theta(n):\tau_i'$ avec $\tau_i'\leq\theta(\tau_i)=\langle\theta(\tau)\rangle^+_{\theta(w)}$. Par définition du sous-typage, le seul sous-type d'un type de canal redéfinissable est lui-même. Nous avons donc $\tau_i'=\theta(\tau_i)$. De plus, le type des canaux redéfinissables étant nécessairement monomorphe, $\theta(n)$ n'a pas de type polymorphe dans Γ et nous avons $\theta(n):\langle\theta(\tau)\rangle^+_{\theta(w)}\in\theta(\Gamma)$.

Soit $n''\in\theta(\Delta)$. Il existe alors un $n\in\Delta$ tel que $n''=\theta(n)$. Par hypothèse de la règle de typage LOC, nous avons donc nécessairement $n:\langle\tau\rangle^+_w\in\Gamma+\widetilde{u_i}:\widetilde{\tau_i}$. Par la propriété prouvée ci-dessus, nous avons donc $n'':\langle\theta(\tau)\rangle^+_{\theta(w)}\in\theta(\Gamma)$. De plus, pour tout $n'\in\Delta$ tel que $\theta(n')=n''$, nous avons $n':\langle\tau'\rangle^+_{w'}\in\Gamma$, par hypothèse de la règle LOC, donc $n'':\langle\theta(\tau')\rangle^+_{\theta(w')}\in\theta(\Gamma)$. Chaque nom n´étant associé qu'une seule fois dans Γ, cette propriété est préservée pour $\theta(\Gamma)$ par l'hypothèse (3), et nous avons nécessairement $\theta(\tau')=\theta(\tau)$ et $\theta(w')=\theta(w)$.

Par conséquent nous avons :

$$\begin{array}{rcl}\forall n_i'\in\theta(\Delta) & . & n_i':\langle\theta(\tau_i^n)\rangle^+_{\theta(w_i)}\in\theta(\Gamma)\ \text{ avec } n_i'=\theta(n_i)\\ \forall n_j'\in\theta(I) & . & n_j':\langle\theta(\tau_j^n)\rangle^+_{\theta(w_j)}\in\theta(\Gamma)\ \text{ avec } n_j'=\theta(n_j)\\ \theta(\mathbf{\Delta}') & = & \bigcup_i\theta(w_i)\\ \theta(\mathbf{I}') & = & \bigcup_j\theta(w_j)\end{array}$$

Ceci montre également que $\theta(\mathbf{\Delta}')$ et $\theta(\mathbf{I}')$ satisfont les hypothèses (4, 5).

L'hypothèse supplémentaire pour les définitions est immédiatement satisfaite pour le typage de D puisque nous avons $fv(B)\subseteq fv(B+a:loc(\mathbf{\Delta}'\cup\mathbf{I}'))$.

Nous utilisons donc l'hypothèse d'induction pour obtenir :

$$\begin{array}{rcl}\theta(\mathbf{\Delta}');\theta(\mathbf{I}');\theta(\Gamma) & \vdash & \theta(D)::\theta(B);\theta(\Delta);fv(\theta(\Theta))\\ \theta(\mathbf{\Delta}');\theta(\mathbf{I}');\theta(\Gamma) & \vdash & \theta(P)\end{array}$$

Nous avons immédiatement $\theta(\mathbf{I}')\subseteq\theta(\mathbf{\Delta}'')\cup\theta(\mathbf{I}'')$.

Nous pouvons donc appliquer la règle de typage LOC, pour obtenir le résultat désiré :

$$\theta(\mathbf{\Delta}'');\theta(\mathbf{I}'');\theta(\Gamma)\vdash a\,[\theta(D):\theta(P)]^{\theta(\Delta),\theta(I)}::\theta(B)+a:loc(\theta(\mathbf{\Delta}')\cup\theta(\mathbf{I}'));\emptyset;fv(\theta(\Theta))$$

(a étant un nom de location, nous avons $\theta(a)=a$).

□

Nous prouvons maintenant la stabilité de la typabilité par réduction.

Preuve du théorème 4: Nous procédons par cas selon la règle de réduction utilisée, sans équivalence structurelle avant ou après la réduction.

DEF Nous étudions la réduction suivante :

$$\alpha a\,[\mathcal{D} : (\texttt{def}\ D\ \texttt{in}\ P) \mid \mathcal{P}]^{\Delta,I,F} \ \parallel\ \mathcal{S} \longrightarrow \alpha a\,[\mathcal{D}, D : \mathcal{P} \mid P]^{\Delta,I,F} \ \parallel\ \mathcal{S}$$

Par hypothèse, les noms de $dsn(D)$ ne sont pas libres dans la configuration, puisqu'ils sont liés dans `def` D `in` P, et ils ne sont liés nul part ailleurs dans la configuration.

La configuration obtenue est bien formée, puisque les noms qui deviennent libres (les noms de $dsn(D)$) ne sont pas liés ailleurs.

Dans les dérivations de typage décrites ci-après, nous abrégeons les hypothèses suivantes :

$$\begin{aligned}
& & \Delta; I; \Gamma \vdash \mathcal{D} :: B; \Delta; \Theta & \quad (B.1)\\
& & A = Gen(B, fv(\Gamma), \Theta) & \quad (B.2)\\
& & A \subseteq \Gamma & \quad (B.3)\\
& & a : loc(\Delta \cup I) \in \Gamma & \quad (B.4)\\
n \in \Delta \quad & \Longrightarrow \quad & n = \mathbf{n} \wedge \mathbf{n} : \langle\tau\rangle^{+}_{\mathbf{n}} \in \Gamma & \quad (B.5)\\
& & \Delta; I; \Gamma \vdash \mathcal{P} & \quad (B.6)
\end{aligned}$$

Nous avons comme dérivation de typage initiale :

$$\dfrac{\dfrac{\dfrac{\dfrac{\Delta; I; \Gamma + A_D \vdash D :: B_D; \emptyset; \Theta_D \qquad A_D = Gen(B_D, fv(\Gamma), \Theta_D)}{\Delta; I; \Gamma + A_D \vdash P \qquad dom(A_D) \cap dom(\Gamma) = \emptyset}\,[\textsc{Def}]}{\Delta; I; \Gamma \vdash \texttt{def}\ D\ \texttt{in}\ P}}{\vdots}}{\dfrac{(B.6)}{\Delta; I; \Gamma \vdash (\texttt{def}\ D\ \texttt{in}\ P) \mid \mathcal{P}}\,[\textsc{Par}]}}{\vdots}$$

$$\dfrac{\dfrac{(B.1) \qquad (B.2) \qquad (B.3)}{\Delta; I; \Gamma \vdash \mathcal{D} : \Delta}\,[\textsc{Chem-Def}] \qquad (B.4, B.5)}{\Gamma \vdash \alpha a\,[\mathcal{D} : (\texttt{def}\ D\ \texttt{in}\ P) \mid \mathcal{P}]^{\Delta,I,F}}\,[\textsc{Soup-Loc}]$$

Nous commençons par vérifier que l'environnement A_D est bien formé. Pour ce faire, nous prouvons par induction sur le typage des définitions la propriété suivante : si $\mathbf{\Delta}; \mathbf{I}; \Gamma \vdash D :: B; \Delta_1; \Theta$ est la conclusion d'une dérivation de typage, alors pour toute association de la forme $a : loc(\mathbf{\Delta'} \cup \mathbf{I'}) \in B$, pour tout $w \in \mathbf{\Delta'} \cup \mathbf{I'}$ il existe une association $m : \langle\tau\rangle^{+}_{w} \in \Gamma$.

JOIN La propriété est immédiate pour ce cas de base puisque B ne contient aucune association de la forme $a : loc(\mathbf{\Delta'} \cup \mathbf{I'})$

TOP Immédiat.

AND La propriété est immédiate par induction.

LOC La propriété est satisfaite pour B par induction, et pour la location typée a par hypothèse de la règle de typage.

Nous utilisons maintenant cette propriété sur le typage $\Delta; I; \Gamma + A_D \vdash D :: B_D; \emptyset; \Theta_D$. Le lemme B.1.3(3) nous indique que $dom(A_D) = dom(B_D) = dsn(D)$, nous déduisons donc du lemme B.1.3(1) que toutes les associations de A_D impliquent soit des canaux statiques, soit des locations. De plus, pour toute association de la forme $b : loc(\mathbf{\Delta'} \cup \mathbf{I'})$ dans B_D, nous utilisons le fait que Γ est bien formée et la propriété précédente pour en déduire que $\mathbf{\Delta'}$ et $\mathbf{I}$ ne sont composés que de noms de canaux dynamiques, et donc ne comportent aucune variable de type de nom. Par conséquent, les locations de A_D sont associées à des types monomorphes ne contenant pas de variable de type de nom, donc A_D est bien formé.

Afin d'utiliser le lemme B.1.11 pour le reste de la configuration, nous devons tout d'abord étudier les noms libres de A_D qui ne sont pas libres dans Γ. Soit Λ_D l'ensemble des noms libres de A_D qui ne sont pas libres dans Γ mais liés dans la configuration $\alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,F} \ \parallel\ \mathcal{S}$. Nous remarquons que nous avons nécessairement $dom(A_D) \cap \Lambda_D = \emptyset$ puisque les noms de $dom(A_D)$ ne peuvent être liés dans cette configuration par hypothèse de la règle de réduction. Nous avons donc $dom(\Gamma + A_D) \cap \Lambda_D = \emptyset$. De plus, les noms de Λ_D n'étant pas libres dans Γ par définition, nous en déduisons par le lemme B.1.5 qu'ils ne peuvent être libres dans `def` D `in` P. Comme $dom(A_D) \cap \Lambda_D = \emptyset$ et $dsn(D) = dom(A_D)$, nous avons $(fn(D) \cup fn(P)) \cap \Lambda_D = \emptyset$. Nous en déduisons que les noms de Λ_D ne peuvent être que des noms présents dans les types de A_D, ce sont donc des noms de canaux dynamiques.

Nous renommons ces noms de Λ_D en des noms qui ne sont pas présents dans tout la dérivation de typage par une substitution bien formée θ, injective sur les noms, en utilisant le lemme B.1.8(2) sur les dérivations de typage de D et de P. Nous remarquons que nous pouvons utiliser ce lemme puisque aucun message résolu n'est présent dans ces termes. Nous obtenons les dérivations de typage :

$$\begin{array}{lcl}\Delta; I; \Gamma + \theta(A_D) & \vdash & D :: \theta(B_D); \emptyset; \Theta_D \\ \Delta; I; \Gamma + \theta(A_D) & \vdash & P\end{array}$$

(en effet, le renommage n'implique que des noms qui ne sont pas libres dans Γ, donc ils ne sont libres ni dans Δ, ni dans I, et Θ ne contient aucun nom).

Nous remarquons que aucun des noms de Λ_D n'est généralisé dans A_D puisque seules les variables de types et variables de types de noms peuvent être généralisées. Nous avons donc $\theta(A_D) = \mathit{Gen}(\theta(B_D), \mathit{fv}(\Gamma), \Theta_D)$.

Nous montrons maintenant que nous pouvons dériver le jugement $\Delta; I; \Gamma + \theta(A_D) \vdash D : \emptyset$. Ceci est immédiat par application de la règle Chem-Def, les dérivations de D et P précédemment montrées, et le fait que $\theta(A_D) \subseteq \Gamma + \theta(A_D)$.

Nous extrayons de la dérivation de typage initiale une dérivation de typage ayant la conclusion suivante :

$$\Gamma \vdash \alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,F} \quad \| \ \mathcal{S}$$

Les noms libres de $\theta(A_D)$ qui ne sont pas libres dans Γ ne sont pas liés dans cette dernière configuration. De plus, nous avons $\mathit{dom}(\theta(A_D)) \cap \mathit{dom}(\Gamma) = \emptyset$. Nous pouvons appliquer le lemme B.1.11 pour chaque association de A_D (nous remarquons que ce faisant l'ensemble $\mathit{fn}(\Gamma)$ augmente, ce qui ne remet pas en cause l'utilisation du lemme). Nous obtenons alors la dérivation :

$$\Gamma + \theta(A_D) \vdash \alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,F} \quad \| \ \mathcal{S}$$

Nous extrayons de cette dérivations les deux sous-dérivations (obtenues après application de la règle Configuration et de la règle Soup-Loc) :

$$\begin{array}{c}\Delta; I; \Gamma + \theta(A_D) \vdash \mathcal{D} : \Delta \\ \Delta; I; \Gamma + \theta(A_D) \vdash \mathcal{P}\end{array}$$

Puisque nous avons $\mathit{dsn}(\mathcal{D}) = \mathit{dom}(A) \subseteq \mathit{dom}(\Gamma)$ et $\mathit{dsn}(D) \cap \mathit{dom}(\Gamma) = \emptyset$, nous avons nécessairement $\mathit{dsn}(\mathcal{D}) \cap \mathit{dsn}(D) = \emptyset$. Nous utilisons le lemme B.1.13 pour obtenir :

$$\Delta; I; \Gamma + \theta(A_D) \vdash \mathcal{D}, D : \Delta$$

Nous pouvons alors construire la dérivation :

$$\dfrac{\dfrac{\Delta; I; \Gamma + \theta(A_D) \vdash \mathcal{P} \qquad \Delta; I; \Gamma + \theta(A_D) \vdash P}{\Delta; I; \Gamma + \theta(A_D) \vdash \mathcal{P} \mid P}\ [\textsc{Par}] \qquad \vdots \qquad (B.4', B.5')}{\Gamma + \theta(A_D) \vdash \alpha a\,[\mathcal{D}, D : \mathcal{P}, P]^{\Delta,I,F}}\ [\textsc{Soup-Loc}]$$

avec (B.4',B.5') étant (B.4,B.5) après avoir remplacé Γ par $\Gamma + \theta(A_D)$.

Nous concluons par la règle Configuration, avec $\Gamma + \theta(A_D)$ bien formé.

Nu Comme dans le cas Def, nous savons par hypothèse que le nom $\mathbf{n}$ n'est pas un nom lié dans la configuration, excepté par la restriction étant dissoute. Ce nom étant lié avant l'étape de réduction, il ne peut donc être libre dans la configuration.

La dérivation de typage de la configuration initiale a une conclusion de la forme :

$$\Gamma \vdash \alpha a\,[\mathcal{D} : \nu\mathbf{n}.P \mid \mathcal{P}]^{\Delta,I,F} \quad \| \ \mathcal{S}$$

avec Γ bien formé.

Nous avons donc une sous-dérivation de la forme :

$$\Delta; I; \Gamma + \mathbf{n} : \langle\tau\rangle^{+}_{\mathbf{n}} \vdash P$$

Comme pour le cas Def, nous considérons l'ensemble Λ des noms qui sont libres dans $\mathbf{n} : \langle\tau\rangle^{+}_{\mathbf{n}}$, qui ne sont pas libres dans Γ et qui sont liés dans la configuration $\alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,F} \quad \| \ \mathcal{S}$. Par

hypothèse de la règle de réduction, $\mathbf{n}$ n'est pas dans Λ. Ainsi, et par le lemme B.1.5, aucun nom libre de P n'est présent dans Λ. Nous renommons les noms de Λ vers des noms qui ne sont pas présents dans toute la dérivation de typage par une substitution bien formée injective pour les noms, en utilisant le lemme B.1.8(2), puisque aucun message résolu n'est présent dans P. Comme dans le cas Def, aucun nom de Λ n'est libre dans Γ par définition, donc aucun nom de Λ n'est présent dans $\Delta \cup I$. La dérivation de typage précédente devient donc :

$$\Delta; I; \Gamma + \mathbf{n} : \langle \tau' \rangle^+_{\mathbf{n}} \vdash P$$

en notant τ' pour τ après la substitution.

Comme pour le cas Def, nous extrayons de la dérivation de typage initiale une dérivation de typage ayant pour conclusion :

$$\Gamma \vdash \alpha a\,[\mathcal{D} : \mathcal{P}]^{\Delta,I,F} \;\parallel\; \mathcal{S}$$

Nous utilisons maintenant le lemme B.1.11 avec cette dérivation et l'association $\mathbf{n} : \langle \tau' \rangle^+_{\mathbf{n}}$ pour obtenir (après utilisation d'une règle de typage Par pour typer le processus de la location a) :

$$\Gamma + \mathbf{n} : \langle \tau' \rangle^+_{\mathbf{n}} \vdash \alpha a\,[\mathcal{D} : P \mid \mathcal{P}]^{\Delta,I,F} \;\parallel\; \mathcal{S}$$

L'environnement de typage final est bien formé.

Join Nous avons l'étape de réduction suivante :

$$\alpha a\,[\mathcal{D}, J \triangleright P : a.J\sigma_{rn} \mid \mathcal{P}]^{\Delta,I,F} \;\parallel\; \mathcal{S} \;\longrightarrow\; \alpha a\,[\mathcal{D}, J \triangleright P : P\sigma_{rn} \mid \mathcal{P}]^{\Delta,I,F} \;\parallel\; \mathcal{S}$$

Dans les dérivations de typages ci-après, nous abrégeons les hypothèses suivantes :

$$\begin{aligned}
& \Delta; I; \Gamma \vdash \mathcal{D} :: B_a; \Delta_a; \Theta && \text{(B.7)}\\
& B_a \oplus B_J = B && \text{(B.8)}\\
& \Delta_a \cup \Delta_J = \Delta && \text{(B.9)}\\
& \Delta; I; \Gamma + (\widetilde{u_i} : \widetilde{\tau_i})^i + (\widetilde{y_j} : \widetilde{\tau_j})^j \vdash P && \text{(B.10)}\\
& \Gamma \vdash \mathtt{x}_i : \langle \widetilde{\tau_i} \rangle && \text{(B.11)}\\
& \Gamma \vdash \mathbf{m}_j : \langle \widetilde{\tau_j} \rangle^+_{\mathbf{m}_j} && \text{(B.12)}\\
& (\bigcup_i \widetilde{u_i} \cup \bigcup_j \widetilde{y_j}) \cap dom(\Gamma) = \emptyset && \text{(B.13)}\\
& \forall (n : \tau), (n' : \tau') \in (\{\mathtt{x}_i : \langle \widetilde{\tau_i} \rangle\} \cup \{\mathbf{m}_j : \langle \widetilde{\tau_j} \rangle^+_{\mathbf{m}_j}\})\\
& \qquad n \neq n' \implies fv(\tau) \cap fv(\tau') \subseteq \Theta && \text{(B.14)}\\
& A = Gen(B, fv(\Gamma), \Theta) && \text{(B.15)}\\
& A \subseteq \Gamma && \text{(B.16)}\\
n \in \Delta \implies\; & n = \mathbf{n} \;\wedge\; \mathbf{n} : \langle \tau \rangle^+_{\mathbf{n}} \in \Gamma && \text{(B.17)}\\
& a : loc(\Delta \cup I) \in \Gamma && \text{(B.18)}\\
& \Delta; I; \Gamma \vdash \mathcal{P} && \text{(B.19)}
\end{aligned}$$

Nous avons la dérivation de typage suivante pour la location αa initiale :

$$\dfrac{\dfrac{\dfrac{(B.10, B.11, B.12, B.13, B.14)}{\Delta; I; \Gamma \vdash J \triangleright P :: B_J; \Delta_J; \Theta} \qquad (B.7, B.8, B.9)}{\Delta; I; \Gamma \vdash \mathcal{D}, J \triangleright Q :: B; \Delta; \Theta}\ [\text{And}]\text{/}[\text{Join}] \qquad \dfrac{(*) \qquad (B.19)}{\Delta; I; \Gamma \vdash a.J\sigma_{rn} \mid \mathcal{P}}\ [\text{Par}]}{\dfrac{\dfrac{\vdots \qquad (B.15, B.16)}{\Delta; I; \Gamma \vdash \mathcal{D}, J \triangleright Q : \Delta}\ [\text{Chem-Def}] \qquad (B.17, B.18) \qquad \vdots}{\Gamma \vdash \alpha a\,[\mathcal{D}, J \triangleright P : a.J\sigma_{rn} \mid \mathcal{P}]^{\Delta,I,F}}\ [\text{Soup-Loc}]}$$

Nous avons $B_J = (\mathtt{x}_i : \langle \widetilde{\tau_i} \rangle)^i$ et $\Delta_J = \bigcup_j \{\mathbf{m}_j\}$.

La dérivation de typage pour le processus étant réduit est de la forme :

$$\dfrac{\dfrac{\Gamma \vdash \mathtt{x}_i : \langle \widetilde{\gamma_i} \rangle_{\Delta \cup I} \qquad \Gamma \vdash \sigma_{rn}(\widetilde{u_i}) : \widetilde{\gamma_i}}{(\Delta; I; \Gamma \vdash a.\mathtt{x}_i \langle \sigma_{rn}(\widetilde{u_i}) \rangle)^i} \qquad \dfrac{\Gamma \vdash \mathbf{m}_j : \langle \widetilde{\gamma_j} \rangle_{\Delta \cup I} \qquad \Gamma \vdash \sigma_{rn}(\widetilde{y_j}) : \widetilde{\gamma_j}}{(\Delta; I; \Gamma \vdash a.\mathbf{m}_j \langle \sigma_{rn}(\widetilde{y_j}) \rangle)^j}\ [\text{Par}]}{(*)\ \ \Delta; I; \Gamma \vdash a.J\sigma_{rn}}\ [\text{R-Msg}]\text{/}[\text{R-Msg}]$$

Dans cette dérivation, nous ne mentionnons pas certaines hypothèses des règles de typage R-Msg qui ne seront pas utiles dans cette preuve.
Nous attirons l'attention sur l'hypothèse (B.12) : dans cette hypothèse, le nom du canal et le nom associé dans son type sont les mêmes. Ceci est une conséquence directe du fait que Γ est bien formé.
Pour obtenir la dérivation souhaitée, il suffit de remplacer la dérivation $(*)$ pour $a.J\sigma_{rn}$ en une dérivation pour $P\sigma_{rn}$. Pour ce faire, nous utiliserons l'hypothèse (B.10) ainsi que le lemme B.1.14.
Dans la dérivation $(*)$, nous étudions tout d'abord la dérivation pour les noms définis du filtre J. Ces dérivations comporte un nombre fini de règles Sub avec à leur racine une règle Name de la forme :

$$\frac{\mathtt{x}_i : \forall\widetilde{\alpha}_i\widetilde{\delta}_i.\langle\widetilde{\tau}_i\rangle \in \Gamma \qquad \langle\theta_i(\widetilde{\tau}_i)\rangle = Inst(\forall\widetilde{\alpha}_i\widetilde{\delta}_i.\langle\widetilde{\tau}_i\rangle)}{\Gamma \vdash \mathtt{x}_i : \langle\theta_i(\widetilde{\tau}_i)\rangle}\ [\textsc{Name}] \qquad \frac{\mathbf{m}_j : \langle\widetilde{\tau}_j\rangle^+_{\mathbf{m}_j} \in \Gamma}{\Gamma \vdash \mathbf{m}_j : \langle\widetilde{\tau}_j\rangle^+_{\mathbf{m}_j}}\ [\textsc{Name}]$$

Les types des $\mathtt{x}_i$ sont déduits du fait que ce sont des noms définis statiques de $J \triangleright P$ et que nous avons $A \subseteq \Gamma$. De plus, nous remarquons que les variables généralisées de ces schémas de types ne peuvent être dans $fv(\Gamma)$ par définition de la généralisation. Les types pour les $\mathbf{m}_j$ sont déduits des hypothèses (B.12), et du fait que les types redéfinissables n'ont que eux-mêmes comme sous-type et ne peuvent être polymorphes dans Γ.
Une des hypothèses du lemme B.1.14 est que les noms libres dans l'image de la substitution ne sont pas liés dans le processus. Une autre hypothèse est que la substitution est bien formée, donc entre autres idempotente. Pour satisfaire ces hypothèses, il est nécessaire de modifier les instanciations θ_i. Soit Λ l'ensemble :

$$\Lambda = \left(\bigcup_i fn(ran(\theta_i)) \cap bn(P)\right) \bigcup \left(\bigcup_i fv(ran(\theta_i)) \cap \bigcup_i dom(\theta_i)\right)$$

Cet ensemble comprend les noms libres des images des θ_i qui sont liés dans P, ainsi que les variables de types et de types de noms qui sont libres dans l'image d'un θ_i et présentes dans le domaine d'un θ_i, pas nécessairement le même.
Nous remarquons que nous avons $\Lambda \cap (fn(\Gamma) \cup fv(\Gamma)) = \emptyset$. Nous vérifions ceci pour les noms en montrant que tout nom lié de P n'est pas libre dans Γ. Pour ce faire, nous procédons par induction sur la dérivation de typage. Les seuls cas intéressants étant les règles introduisant des noms liés (Def, Nu et Join) qui vérifient toutes que le nom lié n'est pas libre dans Γ (dans le cas des règles Def et Join, les noms liés ne sont pas des noms dynamiques, ils ne peuvent être libres dans Γ que s'ils sont dans son domaine). Les autres cas sont immédiats par induction, puisque Γ n'est pas modifié dans les hypothèses des autres règles. En ce qui concerne les variables, nous remarquons que le domaine de chaque θ_i est inclus dans l'ensemble des variables généralisées pour le type du canal considéré, donc ces variables ne peuvent être libres dans Γ. Par conséquent, l'ensemble Λ est aussi disjoint des ensembles Δ et I. De plus, aucun nom de Λ n'étant libre dans Γ, nous avons par le lemme B.1.5 la propriété $\Lambda \cap fn(a.J\sigma_{rn}) = \emptyset$.
Soit θ_Λ une substitution bien formée de Λ vers des noms et variables de types (ou de types de noms) qui ne sont pas présentes dans toute la dérivation de typage (donc en particulier ni dans le domaine ni dans l'image de chaque θ_i). Nous remarquons que $\theta_\Lambda \circ \theta_\Lambda = \theta_\Lambda$, et que θ_Λ est injective pour les noms et pour les variables (elle substitue un nom frais pour un nom et une variable fraîche pour une variable). De plus, aucun nom de location n'est renommé par θ_Λ puisque Λ ne contient aucun nom de location (les seuls noms qui sont présents dans l'image des θ_i sont des noms pouvant apparaître dans des types, c'est à dire des noms dynamiques). Nous pouvons donc utiliser le lemme B.1.8(2) sur chaque dérivation de typage pour les messages $a.\mathtt{x}_i\langle\sigma_{rn}(\widetilde{u}_i)\rangle$ et obtenir :

$$[\text{R-Msg}]\ \frac{\Gamma \vdash \mathtt{x}_i : \langle\widetilde{\gamma'_i}\rangle_{\Delta\cup I} \qquad \Gamma \vdash \sigma_{rn}(\widetilde{u}_i) : \widetilde{\gamma'_i}}{(\Delta; I; \Gamma \vdash a.\mathtt{x}_i\langle\sigma_{rn}(\widetilde{u}_i)\rangle)^i}$$

avec $\gamma'_i = \theta_\Lambda(\gamma_i)$.
Comme auparavant, les dérivations de typage pour les noms statiques définis par le filtre J (ce sont les seuls impliqués dans le renommage) comportent un nombre fini d'utilisations de la règle Sub précédées à la racine d'une règle Name de la forme :

$$\frac{\mathtt{x}_i : \forall\widetilde{\alpha}_i\widetilde{\delta}_i.\langle\widetilde{\tau}_i\rangle \in \Gamma \qquad \langle\theta'_i(\widetilde{\tau}_i)\rangle = Inst(\forall\widetilde{\alpha}_i\widetilde{\delta}_i.\langle\widetilde{\tau}_i\rangle)}{\Gamma \vdash \mathtt{x}_i : \langle\theta'_i(\widetilde{\tau}_i)\rangle}\ [\textsc{Name}]$$

Nous déduisons de la définition de la règle de typage SUB ainsi que de la transitivité de la relation de sous-typage les relations $\gamma'_i \leq \theta'_i(\tau_i)$, $\gamma_j \leq \tau_j$, et $\{\mathbf{m}_j\} \subseteq \Delta \cup I$.

Nous étudions maintenant les dérivations de typage pour les noms reçus. Elles sont également composées d'une succession de règles SUB avec une règle NAME à la racine de la dérivation. Par conséquent, à chaque nom reçu est associé un jugement de typage de la forme :

$$\Gamma \vdash_{\text{NAME}} \sigma_{rn}(u_i) : \lambda_i$$

avec$\lambda_i \leq \gamma'_i$ (ou $\lambda_j \leq \gamma_j$) par définition de la règle SUB et la transitivité de la relation de sous-typage.

Les hypothèses (B.14,B.15) et la définition de la généralisation nous indiquent que les θ'_i ont des domaines deux à deux disjoints. En effet, si une variable de type (ou de type de nom) est dans le domaine de deux θ'_i différents, elle est nécessairement partagée par deux types $\widetilde{\tau_i}$, donc elle n'est pas généralisée, donc elle ne peut être dans le domaine d'un θ'_i. De plus, le domaine des θ'_i ne comportant que des variables de types ou de types de noms, ils ne peuvent contenir de variables de noms, donc ils sont aussi disjoints du domaine de σ_{rn}. Nous notons θ pour la substitution étant l'union de tous les θ'_i et de σ_{rn}.

Nous prouvons tout d'abord que $\theta \circ \theta = \theta$, qui est une conséquence de $\mathit{fn}(\mathit{dom}(\theta)) \cap \mathit{fn}(\mathit{ran}(\theta)) = \mathit{fv}(\mathit{dom}(\theta)) \cap \mathit{fv}(\mathit{ran}(\theta)) = \emptyset$. En ce qui concerne la première intersection d'ensembles, nous avons $\mathit{fn}(\mathit{dom}(\theta)) = \mathit{dom}(\sigma_{rn})$. Nous remarquons tout d'abord que les seuls noms pouvant être présents dans l'image des θ'_i sont des noms pouvant être présents dans des types, c'est à dire des noms dynamiques. Ils sont donc nécessairement distincts des variables de noms du domaine de σ_{rn}. En ce qui concerne l'image de σ_{rn}, nous remarquons que nous avons pour chacun des $\widetilde{u_i}$ et des $\widetilde{y_j}$ un jugement de la forme $\Gamma \vdash \sigma_{rn}(\widetilde{u_i}) : \widetilde{\gamma'_i}$. Par conséquent nous avons nécessairement $\mathit{ran}(\sigma_{rn}) \subseteq \mathit{dom}(\Gamma)$. Comme l'hypothèse B.13 nous indique que $\mathit{dom}(\sigma_{rn}) \cap \mathit{dom}(\Gamma) = \emptyset$, nous avons $\mathit{dom}(\sigma_{rn}) \cap \mathit{ran}(\sigma_{rn}) = \emptyset$. Donc nous avons bien $\mathit{fn}(\mathit{dom}(\theta)) \cap \mathit{fn}(\mathit{ran}(\theta)) = \emptyset$.

Nous montrons maintenant que nous avons $\mathit{fv}(\mathit{dom}(\theta)) \cap \mathit{fv}(\mathit{ran}(\theta)) = \emptyset$. Puisque aucune variable de type ou de type de nom n'est présente dans $\mathit{dom}(\sigma_{rn})$, nous avons $\mathit{fv}(\mathit{dom}(\theta)) = \bigcup_i \mathit{dom}(\theta'_i) = \bigcup_i \mathit{dom}(\theta_i)$. De plus, aucune variable de type ou de type de nom n'est non plus présente dans $\mathit{ran}(\sigma_{rn})$, donc $\mathit{fv}(\mathit{ran}(\theta)) = \bigcup_i \mathit{fv}(\mathit{ran}(\theta'_i))$. Les variables de l'image des θ'_i étant toutes différentes des variables du domaine des θ'_i (par définition du renommage θ_Λ), nous avons $\mathit{fv}(\mathit{dom}(\theta)) \cap \mathit{fv}(\mathit{ran}(\theta)) = \emptyset$.

La substitution θ est une substitution des variables de noms vers les noms (par définition de σ_{rn}), des variables de types vers les types et des variables de types de noms vers des variables de types de noms ou des noms dynamiques (par définition des θ'_i). C'est donc une substitution bien formée.

Nous vérifions maintenant les hypothèses requises pour utiliser le lemme B.1.14 avec les dérivations $\Delta; I; \Gamma + (\widetilde{u_i} : \widetilde{\tau_i})^i + (\widetilde{y_j} : \widetilde{\tau_j})^j \vdash P$, $\Gamma \vdash_{\text{NAME}} \sigma_{rn}(u_i) : \lambda_i$, et $\Gamma \vdash_{\text{NAME}} \sigma_{rn}(y_j) : \lambda_j$.

L'hypothèse (1) est immédiatement satisfaite, puisque les variables renommées par θ sont les variable généralisées des τ_i. L'hypothèse (2) est immédiate. L'hypothèse (3) du lemme B.1.14 est satisfaite pour les noms reçus à cause de l'hypothèse (B.13) de la dérivation de typage, et est immédiatement satisfaite pour les variables puisque celles-ci ne peuvent être dans le domaine de Γ.

Nous remarquons maintenant que les variables de noms présents dans Γ ne peuvent l'être que dans son domaine, donc nous avons $\mathit{dom}(\sigma_{rn}) \cap \mathit{fn}(\Gamma) = \emptyset$. De plus, le domaine des θ'_i ne comporte que des variables généralisées, qui ne peuvent donc être libres dans Γ par définition de la généralisation, donc nous avons également $\mathit{dom}(\theta'_i) \cap \mathit{fv}(\Gamma) = \emptyset$ pour chaque i. Nous en déduisons que $\theta(\Gamma) = \Gamma$. Comme nous avons $\mathit{fn}(\Delta \cup I) \subseteq \mathit{fn}(\Gamma)$ et $\mathit{fv}(\Delta \cup I) \subseteq \mathit{fv}(\Gamma)$, nous avons donc $\theta(\Delta) = \Delta$ et $\theta(I) = I$. Les hypothèses (4,5) sont donc satisfaites.

Les variables des τ_j étant libres dans Γ (puisque nous avons $\mathbf{m}_j : \langle \widetilde{\tau_j} \rangle^+_{\mathbf{m}_j} \in \Gamma$), nous avons donc $\theta(\tau_j) = \tau_j$.

Par transitivité de la relation de sous-typage, nous avons $\lambda_i \leq \gamma'_i \leq \theta(\tau_i)$ donc $\lambda_i \leq \theta(\tau_i)$, ainsi que $\lambda_j \leq \gamma_j \leq \tau_j = \theta(\tau_j)$ donc $\lambda_j \leq \theta(\tau_j)$.

Nous vérifions maintenant que les noms libres de l'image de θ sont disjoints des noms liés de P. C'est le cas pour les noms dans l'image de σ_{rn}, puisqu'ils sont dans le domaine de $\Gamma + (\widetilde{u_i} : \widetilde{\tau_i})^i + (\widetilde{y_j} : \widetilde{\tau_j})^j$ qui est l'environnement utilisé pour typer P, ils ne peuvent donc être liés dans P. C'est également le cas pour les noms dans l'image des θ'_i, puisque le renommage θ_Λ s'en est assuré.

Nous pouvons donc utiliser le lemme B.1.14 pour obtenir la dérivation de typage :

$$\Delta; I; \Gamma \vdash \theta(P) = P\sigma_{rn}$$

Nous concluons en remplaçant la dérivation $(*)$ par la dérivation précédente dans la règle PAR.

L'environnement de typage Γ n'ayant pas été modifié, il est toujours bien formé.

NL-STAT L'étape de réduction suivante a lieu :

$$\alpha a\,[\mathcal{D} : \mathcal{P} \mid \mathtt{n}\langle\widetilde{v}\rangle; P]^{\Delta,I,F} \longrightarrow \alpha a\,[\mathcal{D} : \mathcal{P} \mid b.\mathtt{n}\langle\widetilde{v}\rangle \mid P]^{\Delta,I,F}$$

avec $\mathtt{n} \in \mathit{dln}(b)$.

Le typage du canal $\mathtt{n}$, de ses arguments $\widetilde{v}$ et du processus P sont identiques avant et après l'étape de réduction. Le typage du reste de la configuration n'est pas non plus modifié.

Il nous reste donc simplement à vérifier que nous avons $\mathtt{n} \in \mathit{dln}(b)$ (ce qui est immédiat par hypothèse de la règle de réduction) et $\Gamma \vdash b : \mathit{loc}(\emptyset)$. Puisque nous avons $\mathtt{n} \in \mathit{dln}(b)$, il existe une location active b, qui est typée soit par une règle de typage SOUP-LOC si elle est dépliée, soit par une règle de typage LOC si elle est repliée. Nous avons donc soit $b : \mathit{loc}(\Delta_b \cup I_b) \in \Gamma$, soit $\Gamma \vdash b : \mathit{loc}(\Delta_b \cup I_b)$ pour des ensembles Δ_b et I_b. Nous concluons donc dans le premier cas en utilisant la règle de typage NAME suivie de la règle SUB (en utilisant la règle de sous-typage L-SUB), et dans le deuxième cas avec simplement la règle SUB et le même sous-typage, pour en déduire $b : \mathit{loc}(\emptyset)$.

NL-DYN La réduction suivante a lieu :

$$\alpha a\,[\mathcal{D} : \mathcal{P} \mid \mathbf{n}\langle\widetilde{v}\rangle; P]^{\Delta,I,F} \longrightarrow \alpha a\,[\mathcal{D} : \mathcal{P} \mid b.\mathbf{n}\langle\widetilde{v}\rangle \mid P]^{\Delta,I,F}$$

avec $F(\mathbf{n}) = b \neq \bot$.

Le typage du canal $\mathbf{n}$, de ses arguments $\widetilde{v}$ et du processus P est le même avant et après l'étape de réduction. Le typage du reste de la configuration n'est pas non plus modifié.

Le lemme 4.6.3 nous indique que $\mathbf{n} \in \mathit{dyn}(b)$. Nous montrons maintenant que ceci implique nécessairement que $\mathbf{n} \in \mathit{dln}(b)$. Le lemme 4.6.3 précise que la location b est présente dans αa. Par l'hypothèse de structure d'arbre de la règle de typage CONFIGURATION, cette location b est dépliée. Elle est donc typée par une règle SOUP-LOC : $\Gamma \vdash \beta b\,[\mathcal{D}_b : \mathcal{P}_b]^{\Delta_b,I_b,F_b}$. Nous avons donc une dérivation de la forme $\Delta_b; I_b; \Gamma \vdash \mathcal{D}_b : \Delta_b$, avec $\Delta_b = \mathit{dyn}(b)$. Donc par la règle CHEM-DEF, nous avons une dérivation de typage $\Delta_b; I_b; \Gamma \vdash \mathcal{D}_b :: B; \Delta_b; \Theta$. Le lemme B.1.3(6) nous indique alors que $\Delta_b = \mathit{dyn}(b) \subseteq \mathit{dln}(\mathcal{D})$.

Nous dérivons le jugement $\Gamma \vdash b : \mathit{loc}(\emptyset)$ exactement comme dans le cas NL-STAT.

COMM La réduction suivante a lieu :

$$\begin{array}{rl} & \alpha a\,[\mathcal{D}_a : \mathcal{P}_a \mid b.n\langle\widetilde{v}\rangle]^{\Delta_a,I_a,F_a} \parallel \beta b\,[\mathcal{D}_b : \mathcal{P}_b]^{\Delta_b,I_b,F_b} \\ \longrightarrow & \alpha a\,[\mathcal{D}_a : \mathcal{P}_a]^{\Delta_a,I_a,F_a} \parallel \beta b\,[\mathcal{D}_b : \mathcal{P}_b \mid b.n\langle\widetilde{v}\rangle]^{\Delta_b,I_b,F_b} \end{array}$$

Le typage de tous les sous-termes est identique avant et après l'étape de réduction, excepté pour le nom n. Avant l'étape de réduction, nous avons une dérivation de typage pour n de la forme : $\Gamma \vdash n : \langle\widetilde{\tau_i}\rangle_{\Delta_a \cup I_a}$ (c'est une hypothèse de la règle R-MSG). Cette dérivation est nécessairement composée d'un nombre fini d'utilisations de la règle SUB, avec à la racine une règle NAME. Il existe donc un schéma de type σ tel que $n : \sigma \in \Gamma$. De plus, par définition de la règle SUB et transitivité de la relation de sous-typage, nous avons $\mathit{Inst}(\sigma) \leq \langle\widetilde{\tau_i}\rangle_{\Delta_a \cup I_a}$. Par définition du sous-typage, et l'environnement Γ étant bien formé, la type $\mathit{Inst}(\sigma)$ a nécessairement soit la forme $\langle\widetilde{\tau'_i}\rangle$ (si $n = \mathtt{n}$) ou la forme $\langle\widetilde{\tau'_i}\rangle^{+}_{\mathbf{n}}$ (si $n = \mathbf{n}$) avec $\widetilde{\tau_i} \leq \widetilde{\tau'_i}$ et $\mathbf{n} \in \Delta_a \cup I_a$ dans le deuxième cas.

Nous considérons maintenant le typage par la règle CHEM-DEF de la définition de la location b. Ce typage a pour hypothèse :

$$\Delta_b; I_b; \Gamma \vdash \mathcal{D}_b :: B_b; \Delta_b; \Theta$$

Le lemme B.1.3(4) nous indique que $\mathit{dn}(\mathcal{D}_b) = \mathit{dom}(B_b) \cup \mathit{ddn}(\mathcal{D}_b)$. Si n est un canal redéfinissable (cas $n = \mathbf{n}$), alors nous avons $n \in \mathit{ddn}(\mathcal{D}_b)$. Comme n est dans $\mathit{dln}(\mathcal{D}_b) \cap \mathit{ddn}(\mathcal{D}_b)$, nous avons par le lemme B.1.3(5) la propriété $n \in \Delta_b$. Par conséquent, que n soit un canal statique ou un canal dynamique, nous avons une dérivation $\Gamma \vdash n : \langle\widetilde{\tau_i}\rangle_{\Delta_b \cup I_b}$ par les règles NAME et SUB. En effet, nous avons $\mathit{Inst}(\sigma) \leq \langle\widetilde{\tau_i}\rangle_{\Delta_b \cup I_b}$ par la règle de sous-typage N-SUB puisque, en prenant la même instanciation, $\widetilde{\tau_i} \leq \widetilde{\tau'_i}$ et soit $\emptyset \subseteq \Delta_b \cup I_b$ si n est statique, soit $\{\mathbf{n}\} \subseteq \Delta_b \cup I_b$ si n est dynamique (dans ce deuxième cas, nous utilisons également la règle de sous-typage PLUS avant la règle N-SUB).

Go Nous ne nous intéressons qu'aux deux définitions qui sont impliquées par l'étape de réduction, le typage du reste de la configuration n'étant pas modifié.

Nous abrégeons dans la suite les hypothèses suivantes :

$$
\begin{aligned}
n \in \Delta_d \implies\ & n = \mathbf{n} \wedge \mathbf{n} : \langle \tau \rangle^+_{\mathbf{n}} \in \Gamma && \text{(B.20)}\\
& d : loc(\Delta_d \cup I_d) \in \Gamma && \text{(B.21)}\\
& \Delta_d; I_d; \Gamma \vdash \mathcal{P}_d && \text{(B.22)}\\
& \Delta_d; I_d; \Gamma \vdash \mathcal{D}_d :: B_d; \Delta_d; \Theta_d && \text{(B.23)}\\
& A_d = Gen(B_d, fv(\Gamma), \Theta_d) && \text{(B.24)}\\
& A_a = Gen(B_a + a : loc(\Delta_a \cup I_a), fv(\Gamma), \Theta_d) && \text{(B.25)}\\
& A_d \subseteq \Gamma && \text{(B.26)}\\
& \Delta_a; I_a; \Gamma \vdash \mathcal{D}_a :: B_a; \Delta_a; \Theta_d && \text{(B.27)}\\
& \Delta_a; I_a; \Gamma \vdash \mathcal{P}_a && \text{(B.28)}\\
& \Delta_a; I_a; \Gamma \vdash Q && \text{(B.29)}\\
n \in \Delta_b \implies\ & n = \mathbf{n} \wedge \mathbf{n} : \langle \tau \rangle^+_{\mathbf{n}} \in \Gamma && \text{(B.30)}\\
& b : loc(\Delta_b \cup I_b) \in \Gamma && \text{(B.31)}\\
& \Delta_b; I_b; \Gamma \vdash \mathcal{P}_b && \text{(B.32)}
\end{aligned}
$$

Avant l'étape Go, nous avons les dérivations de typage pour chaque location :

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash b : loc(I_a) \quad (B.29)}{\Delta_a; I_a; \Gamma \vdash \texttt{go}\ b; Q}\,[\textsc{Go}] \quad (B.28)}{\Delta_a; I_a; \Gamma \vdash \mathcal{P}_a \mid \texttt{go}\ b; Q}\,[\textsc{Par}] \quad (B.27) \quad I_a \subseteq (\Delta_d \cup I_d)}{\Delta_d; I_d; \Gamma \vdash a\,[\mathcal{D}_a : \mathcal{P}_a \mid \texttt{go}\ b; Q]^{\Delta_a, I_a} :: B_a + a : loc(\Delta_a \cup I_a); \emptyset; \Theta_d}\,[\textsc{Loc}] \quad (B.23)}{\Delta_d; I_d; \Gamma \vdash a\,[\mathcal{D}_a : \mathcal{P}_a \mid \texttt{go}\ b; Q]^{\Delta_a, I_a}, \mathcal{D}_d :: B_a + a : loc(\Delta_a \cup I_a) \oplus B_d; \Delta_d; \Theta_d}\,[\textsc{And}]}{\vdots}}{\dfrac{(B.20, B.21, B.22) \quad \dfrac{(B.24, B.25) \quad A_a + A_d \subseteq \Gamma}{\Delta_d; I_d; \Gamma \vdash a\,[\mathcal{D}_a : \mathcal{P}_a \mid \texttt{go}\ b; Q]^{\Delta_a, I_a}, \mathcal{D}_d : \Delta_d}\,[\textsc{Chem-Def}]}{\Gamma \vdash \alpha d \left[a\,[\mathcal{D}_a : \mathcal{P}_a \mid \texttt{go}\ b; Q]^{\Delta_a, I_a}, \mathcal{D}_d : \mathcal{P}_d\right]^{\Delta_d, I_d, F_d}}\,[\textsc{Soup-Loc}]}
$$

et :

$$
\dfrac{(B.30, B.31, B.32) \quad \Delta_b; I_b; \Gamma \vdash \mathcal{D}_b : \Delta_b}{\Gamma \vdash \beta b\,[\mathcal{D}_b : \mathcal{P}_b]^{\Delta_b, I_b, F_b}}\,[\textsc{Soup-Loc}]
$$

Nous remarquons que par hypothèse Γ est bien formé, donc nous avons nécessairement $\mathbf{\Delta}'_a = \Delta_a$ et $\mathbf{I}'_a = I_a$. Les hypothèses de la règle de typage Loc pour a on été simplifiées en conséquence.

De plus, les noms définis statiques de $B_a + a : loc(\Delta_a \cup I_a)$ sont définis dans une location différente que ceux de B_d, nous avons par conséquent $dom(B_a + a : loc(\Delta_a \cup I_a)) \cap dom(B_d) = \emptyset$. Donc nous avons $Gen(B_a + a : loc(\Delta_a \cup I_a) \oplus B_d, fv(\Gamma), \Theta_d) = A_a + A_d$.

Nous remarquons également que puisque $b : loc(\Delta_b \cup I_b) \in \Gamma$ et $\Gamma \vdash b : loc(I_a)$, nous avons nécessairement par définition du sous-typage sur les types des locations $I_a \subseteq \Delta_b \cup I_b$.

Comme dans le cas Def, la seule difficulté est de prendre en compte les Θ différents dans le typage des définitions.

Nous construisons tout d'abord la dérivation de typage :

$$
\dfrac{\dfrac{\dfrac{\dfrac{(B.28, B.29)}{\Delta_a; I_a; \Gamma \vdash \mathcal{P}_a \mid Q}\,[\textsc{Par}] \quad (B.27) \quad I_a \subseteq \Delta_b \cup I_b}{\Delta_b; I_b; \Gamma \vdash a[\mathcal{D}_a : \mathcal{P}_a \mid Q]^{\Delta_a, I_a} :: B_a + a : loc(\Delta_a \cup I_a); \emptyset; \Theta_d}\,[\textsc{Loc}]}{\vdots}}{\Delta_b; I_b; \Gamma \vdash a\,[\mathcal{D}_a : \mathcal{P}_a \mid Q]^{\Delta_a, I_a} : \emptyset}\,[\textsc{Chem-Def}]
$$

Comme nous avons $dsn(a\,[\mathcal{D}_a : \mathcal{P}_a \mid Q]^{\Delta_a, I_a}) \cap dsn(\mathcal{D}_b) = \emptyset$ (ces noms définis statiques sont définis dans des locations différentes), nous utilisons le lemme B.1.13 pour obtenir :

$$\Delta_b; I_b; \Gamma \vdash a\,[\mathcal{D}_a : \mathcal{P}_a \mid Q]^{\Delta_a, I_a}, \mathcal{D}_b : \Delta_b$$

Nous construisons maintenant les dérivations de typage pour les locations après l'étape Go :

$$\dfrac{(B.20, B.21, B.22) \qquad \dfrac{(B.23, B.24, B.26)}{\Delta_d; I_d; \Gamma \vdash \mathcal{D}_d : \Delta_d}\,[\textsc{Chem-Def}]}{\Gamma \vdash \alpha d\,[\mathcal{D}_d : \mathcal{P}_d]^{\Delta_d, I_d, F_d}}\,[\textsc{Soup-Loc}]$$

et

$$\dfrac{(B.30, B.31, B.32) \qquad \Delta_b; I_b; \Gamma \vdash a\,[\mathcal{D}_a : \mathcal{P}_a \mid Q]^{\Delta_a, I_a}, \mathcal{D}_b : \Delta_b}{\Gamma \vdash \beta b \left[a\,[\mathcal{D}_a : \mathcal{P}_a \mid Q]^{\Delta_a, I_a}, \mathcal{D}_b : \mathcal{P}_b\right]^{\Delta_b, I_b, F_b}}\,[\textsc{Soup-Loc}]$$

□

Preuve du théorème 5: Nous commençons par prouver une propriété de *préservation de l'environnement*. Soit $\Gamma \vdash \mathcal{S}$ la conclusion d'une dérivation de typage. Pour tout processus $\mathcal{P}$ ou définition $\mathcal{D}$ sous-terme de $\mathcal{S}$ qui n'est pas sous une garde (définition locale, restriction, règle de réaction), la sous-dérivation de typage de ce processus est de la forme $\Delta; I; \Gamma \vdash \mathcal{P}$ et la sous-dérivation de cette définition est de la forme $\Delta; I; \Gamma \vdash \mathcal{D} :: B; \Delta_1; \Theta$, pour des Δ, I, B, Δ_1, et Θ (en d'autres termes, l'environnement Γ est le même que celui utilisé pour typer la configuration). Cette propriété est prouvée par induction sur la dérivation de typage, et tous les cas sont immédiats puisque les seules règles de typage modifiant l'environnement Γ dans leurs hypothèses sont celles utilisées pour typer des termes gardés (Def pour les définitions locales, Nu pour les restrictions, et Join pour les règles de réactions).

Nous procédons par cas selon le blocage, en supposant avoir une dérivation de typage pour $\Gamma \vdash \mathcal{S}$ avec Γ bien formé.

Pour le cas (1), le message sur n étant en contexte d'évaluation, il est typé par une règle Msg ou R-Msg avec le même environnement Γ. Par hypothèse de ces règles de typage, n est dans le domaine de Γ avec un type de canal. Puisque Γ est bien formé, n ne peut être qu'un canal statique ou un canal redéfinissable (il ne peut pas être un nom de variable ou une location).

En ce qui concerne le cas (2), nous commençons par montrer par induction sur la structure de $\mathcal{S}$ que l'ensemble $dn(\mathcal{S})$ est égale à l'union des ensembles $dn(\mathcal{D})$ pour toute définition $\mathcal{D}$ de $\mathcal{S}$ qui n'est pas présente sous une garde. Chaque cas est immédiat en utilisant la définition de dn. Par préservation de l'environnement, le typage de chacune de ces définitions a la forme : $\Delta_{\mathcal{D}}; I_{\mathcal{D}}; \Gamma \vdash \mathcal{D} :: B_{\mathcal{D}}; \Delta_{\mathcal{D}}; \Theta_{\mathcal{D}}$. Par le lemme B.1.3(3,4), nous avons $dn(\mathcal{D}) = dsn(\mathcal{D}) \cup ddn(\mathcal{D})$. Si un nom n de $dn(\mathcal{D})$ est dans $dsn(\mathcal{D})$, alors par le lemme B.1.3(1), ce nom est soit statique, soit une location dans Γ. Puisque Γ est bien formé, le nom est alors soit un nom de canal statique, soit un nom de location. Si le nom n est dans $ddn(\mathcal{D})$, alors par le lemme B.1.3(2), n est redéfinissable dans Γ, donc, par bonne formation de Γ, n est un nom de canal dynamique. Par conséquent tout nom de $dn(\mathcal{S})$ est soit un nom de canal, soit un nom de location.

Pour le cas (3), nous utilisons également la préservation de l'environnement pour le processus $\mathsf{go}\ n; P$. Ce processus est nécessairement typé en utilisant la règle de typage Go, donc n est une location dans Γ. L'environnement Γ étant bien formé, n est donc un nom de location.

Nous considérons maintenant le cas (4). Nous remarquons tout d'abord par une induction immédiate sur la relation de sous-typage que cette relation ne modifie pas l'arité du type d'un canal. Nous remarquons également que l'arité du type d'un canal est la même que l'arité du canal lors du typage de ses définitions (règle de typage Join), et que ces arités sont aussi égales lors du typage d'un message (règles Msg et R-Msg). Nous utilisons la préservation de l'environnement pour conclure. Soient un message non gardé sur n avec i arguments, et une définition non gardée de n attendant j arguments. Le message est typé dans l'environnement Γ et cet environnement contient une association pour n dont le type est un canal d'arité i. La définition est typée dans l'environnement Γ et cet environnement contient une association pour n dont le type est un canal d'arité j. Comme il ne peut y avoir qu'une seule association pour un nom donné dans un environnement donné, nous avons donc $i = j$.

Nous montrons maintenant que le cas de blocage (5) ne peut se produire. Nous considérons un message $\mathbf{n}\langle\widetilde{m}\rangle$ non gardé, présent dans une location active αa (si cette location n'est pas dépliée, nous utilisons l'équivalence structurelle pour la déplier). Par préservation de l'environnement, le nom $\mathbf{n}$ est typé dans l'environnement $\Gamma : \Gamma \vdash \mathbf{n} : \langle\widetilde{\tau}\rangle_{\Delta_a \cup I_a}$. Puisque Γ est bien formé, nous avons $\mathbf{n} : \langle\tau\rangle^+_{\mathbf{n}} \in \Gamma$. Par définition du sous-typage, nous avons donc nécessairement $\mathbf{n} \in \Delta_a \cup I_a$. Par le lemme 4.6.3, nous avons $F(\alpha a) = b \neq \bot$ puisque $\Delta_a = \mathit{dyn}(a)$ et $I_a = \mathit{imp}(a)$. Nous pouvons donc utiliser la règle de réduction NL-Dyn.

Le dernier cas de blocage (6) ne peut se produire puisque le typage d'un message résolu (règle R-Msg) a pour hypothèse que le canal sur lequel le message est envoyé est bien défini dans la location ciblée. □

Annexe C

Preuves du chapitre 5

C.1 Unicité des noms des domaines actifs

Nous prouvons dans cette section que tout domaine actif d'une configuration bien typée du M-calcul possède un nom unique, et que la typabilité d'une configuration est stable par réduction.

C.1.1 Lemmes préliminaires

Proposition C.1.1 (Propriétés de la relation de sous-typage) *1. La relation $\leq$ est un ordre partiel sur les types.*

2. *Soit θ une substitution des variables de types vers les types, des variables de multi-ensembles vers les multi-ensembles, et des variables de types de noms vers les variables de types de noms ou les noms de domaines. Pour tout type τ_1 et τ_2, si $\tau_1 \leq \tau_2$ alors $\tau_1\theta \leq \tau_2\theta$.*
3. *Soient τ_1', τ_2', τ_1 et τ_2 tels que $\tau_1' \leq \tau_1$, $\tau_2' \leq \tau_2$. Si $\tau_1 \sqcup \tau_2$ est défini, alors $\tau_1' \sqcup \tau_2'$ est défini et nous avons $\tau_1' \sqcup \tau_2' \leq \tau_1 \sqcup \tau_2$. Si $\tau_1' \sqcap \tau_2'$ est défini alors $\tau_1 \sqcap \tau_2$ est défini et nous avons $\tau_1' \sqcap \tau_2' \leq \tau_1 \sqcap \tau_2$.*
4. *Soient τ_1 et τ_2. Si $\tau_1 \sqcup \tau_2$ est défini, alors nous avons $\tau_1 \leq \tau_1 \sqcup \tau_2$. Si $\tau_1 \sqcap \tau_2$ est défini, alors nous avons $\tau_1 \sqcap \tau_2 \leq \tau_1$.*
5. *Soient τ_1 et τ_2 deux types. Si $\tau_1 \sqcup \tau_2$ est défini, alors $fn(\tau_1 \sqcup \tau_2) \subseteq fn(\tau_1) \cup fn(\tau_2)$. Si $\tau_1 \sqcap \tau_2$ est défini, alors $fn(\tau_1 \sqcap \tau_2) \subseteq fn(\tau_1) \cup fn(\tau_2)$.*

Preuve: Nous montrons la propriété (1), c'est à dire nous montrons que la relation $\leq$ est réflexive, antisymétrique et transitive. Nous commençons par la réflexivité, par induction sur la structure du type. Le cas des multi-ensembles, des variables de types, de `unit` et de $\texttt{dom}(w)$ est immédiat. Le cas des nuplet, des types fonctionnels et de ressources est également immédiat, en utilisant l'hypothèse d'induction.

Nous prouvons maintenant l'antisymétrie de $\leq$. Soient τ_1 et τ_2 tels que $\tau_1 \leq \tau_2$ et $\tau_2 \leq \tau_1$. Nous procédons par induction sur la somme des tailles des deux types. Les cas $\texttt{dom}(w)$, `unit` et le cas des variables de types sont immédiats. Nous considérons le cas des multi-ensembles. Soient Δ_1 et Δ_2 tels que $\Delta_1 \leq \Delta_2$ et $\Delta_2 \leq \Delta_1$. Nous montrons que pour tout nom ou variable de multi-ensemble, le nombre d'occurrences dans Δ_1 et Δ_2 est identique. Nous ne considérons que le cas des noms. Soit a un nom. Par définition de $\leq$, nous avons $\Delta_1 \subseteq \Delta_2$ et $\Delta_2 \subseteq \Delta_1$, donc le nombre d'occurrences de a dans Δ_1 est à la fois inférieur ou égal et supérieur ou égal à celui de Δ_2, il est donc identique. Les deux ensembles sont par conséquent les mêmes. Le cas nuplet est immédiat par induction, puisque les deux types sont nécessairement des nuplets de même taille. Dans le cas où τ_1 est un type fonctionnel, puisque $\tau_1 \leq \tau_2$, le type τ_2 est nécessairement fonctionnel et nous concluons par induction. Dans le cas où τ_1 est un type de ressource, puisque $\tau_2 \leq \tau_1$, le type τ_2 est nécessairement un type de ressource, et nous concluons par induction.

Nous montrons que $\leq$ est transitive : soient τ_1, τ_2 et τ_3 tels que $\tau_1 \leq \tau_2$ et $\tau_2 \leq \tau_3$, alors nous montrons que $\tau_1 \leq \tau_3$. Nous procédons par induction sur la somme des tailles des trois types. Le résultat est immédiat pour les types $\texttt{dom}(w)$, `unit` et les variables de types puisque par définition de $\leq$, $\tau_1 = \tau_2 = \tau_3$. La relation est aussi transitive pour les multi-ensembles. En effet, par définition de $\leq$, nous avons nécessairement $\tau_1 = \Delta_1$, $\tau_2 = \Delta_2$ et $\tau_3 = \Delta_3$ (les trois types sont des types de multi-ensembles), et $\Delta_1 \subseteq \Delta_2 \subseteq \Delta_3$. Nous voulons montrer que $\Delta_1 \subseteq \Delta_3$. Soit un nom a. Par hypothèse, le nombre d'occurrences de a dans Δ_1 est inférieur ou égal au nombre d'occurrences de a dans Δ_2 qui est inférieur ou égal au nombre d'occurrences de a dans Δ_3. Nous avons donc bien

$\Delta_1 \subseteq \Delta_3$. Enfin, nous prouvons la transitivité pour les nuplets et les fonctions. Dans le cas des nuplets, les trois types sont nécessairement des types de nuplets de même taille, et nous concluons par induction. Le cas des types fonctionnels et de ressources est légèrement plus complexe. Si τ_1 est un type fonctionnel, alors nécessairement τ_2 et τ_3 le sont aussi, et nous concluons par induction. Si τ_3 est un type de ressource, alors nécessairement τ_1 et τ_2 le sont aussi et nous concluons par induction. Si nous avons :

$$\langle\sigma_1\rangle_{\Delta_1} \leq \sigma_2 \to \Delta_2 \leq \sigma_3 \to \Delta_3$$

ou

$$\langle\sigma_1\rangle_{\Delta_1} \leq \langle\sigma_2\rangle_{\Delta_2} \leq \sigma_3 \to \Delta_3$$

alors par définition de $\leq$, nous avons $\sigma_3 \leq \sigma_2 \leq \sigma_1$ et $\Delta_1 \leq \Delta_2 \leq \Delta_3$, et nous concluons par induction et en utilisant la règle de sous-typage entre types de ressources et types fonctionnels.

Nous montrons maintenant la propriété (2) par induction sur la taille des types. Cette propriété est immédiate pour $\texttt{dom}(w)$ et $\texttt{unit}$. Elle est vraie pour les variables de types puisque $\leq$ est réflexive. Elle est vraie pour les multi-ensembles puisque si $\Delta_1 \subseteq \Delta_2$ alors $\Delta_2 = \Delta_1, \Delta_1'$, et nous avons $\Delta_1\theta \subseteq \Delta_1\theta, \Delta_1'\theta = \Delta_2\theta$. Elle est vraie pour les nuplets, les fonctions et les ressources en utilisant l'hypothèse d'induction.

Nous prouvons la propriété (3) par induction sur la somme des tailles de τ_1 et τ_2. Si τ_1 est $\texttt{unit}$, $\texttt{dom}(w)$ ou une variable de type, alors nécessairement nous avons $\tau_1 = \tau_2 = \tau_1' = \tau_2'$ et la propriété est immédiate pour $\sqcup$ et $\sqcap$. Si τ_1 est un multi-ensemble, alors τ_2, τ_1' et τ_2' sont nécessairement des multi-ensembles. Par conséquent $\tau_1' \sqcup \tau_2'$ est bien défini et $\tau_1 \sqcap \tau_2$ également. Nous considérons maintenant un nom a. Par définition de $\sqcup$, le nombre d'occurrences de a dans $\tau_1' \sqcup \tau_2'$ est le maximum du nombre d'occurrences de a dans τ_1' et τ_2'. Supposons que ce nombre soit le nombre d'occurrences de a dans τ_1', alors il est inférieur au nombre d'occurrences de a dans τ_1 par définition de $\subseteq$. Par conséquent, il est inférieur au maximum du nombre d'occurrences de a dans τ_1 et τ_2. Nous avons donc bien $\tau_1' \sqcup \tau_2' \leq \tau_1 \sqcup \tau_2$. Nous considérons maintenant le cas de $\sqcap$. Soit un nom a. Le nombre d'occurrences de a dans $\tau_1' \sqcap \tau_1$ est le minimum du nombre d'occurrences de a dans τ_1' et τ_2'. Par définition de $\subseteq$, le nombre d'occurrences de a dans τ_1' (respectivement τ_2') est inférieur au nombre d'occurrences de a dans τ_1 (respectivement τ_2). Donc le nombre d'occurrences de a dans $\tau_1' \sqcap \tau_2'$ est nécessairement inférieur au nombre d'occurrences de a dans $\tau_1 \sqcap \tau_2$. Nous avons donc bien $\tau_1' \sqcap \tau_2' \leq \tau_1 \sqcap \tau_2$.

Le cas nuplet est immédiat par induction, puisque tous les types sont nécessairement des nuplets de même taille. Nous considérons maintenant le cas des types fonctionnels et des types de ressources.

Nous étudions le cas $\sigma_1' \to \tau_1' \leq \sigma_1 \to \tau_1$ et $\sigma_2' \to \tau_2' \leq \sigma_2 \to \tau_2$. Si $\sigma_1 \to \tau_1 \sqcup \sigma_2 \to \tau_2$ est défini, alors $\sigma_1 \sqcap \sigma_2$ est défini et $\tau_1 \sqcup \tau_2$ est défini. Puisque nous avons $\sigma_1 \leq \sigma_1'$, $\sigma_2 \leq \sigma_2'$, $\tau_1' \leq \tau_1$ et $\tau_2' \leq \tau_2$, nous en déduisons par induction que $\sigma_1' \sqcap \sigma_2'$ est défini avec $\sigma_1 \sqcap \sigma_2 \leq \sigma_1' \sqcap \sigma_2'$, et que $\tau_1' \sqcup \tau_2'$ est défini avec $\tau_1' \sqcup \tau_2' \leq \tau_1 \sqcup \tau_2$. Par conséquent $\sigma_1' \to \tau_1' \sqcup \sigma_2' \to \tau_2'$ est défini et $\sigma_1' \to \tau_1' \sqcup \sigma_2' \to \tau_2' = (\sigma_1' \sqcap \sigma_2') \to (\tau_1' \sqcup \tau_2') \leq (\sigma_1 \sqcap \sigma_2) \to (\tau_1 \sqcup \tau_2) = \sigma_1 \to \tau_1 \sqcup \sigma_2 \to \tau_2$. Le cas où $\sigma_1' \to \tau_1' \sqcap \sigma_2' \to \tau_2'$ est défini est prouvé de manière identique.

Le cas où les quatre types sont des types de ressources est prouvé de manière identique au cas fonctionnel.

Nous détaillons maintenant le cas $\langle\sigma_1'\rangle_{\Delta_1'} \leq \sigma_1 \to \Delta_1$ et $\langle\sigma_2'\rangle_{\Delta_2'} \leq \langle\sigma_2\rangle_{\Delta_2}$. Si $\sigma_1' \to \Delta_1' \sqcap \langle\sigma_2'\rangle_{\Delta_2'}$ est défini, alors $\sigma_1' \sqcup \sigma_2'$ est défini ainsi que $\Delta_1' \sqcap \Delta_2'$. Par définition de $\leq$ nous avons $\sigma_1 \leq \sigma_1'$, $\sigma_2 \leq \sigma_2'$, $\Delta_1' \leq \Delta_1$ et $\Delta_2' \leq \Delta_2$. Par induction, nous avons $\sigma_1 \sqcup \sigma_2$ est défini et $\sigma_1 \sqcup \sigma_2 \leq \sigma_1' \sqcup \sigma_2'$, et $\Delta_1 \sqcap \Delta_2$ est défini et $\Delta_1' \sqcap \Delta_2' \leq \Delta_1 \sqcap \Delta_2$. Nous avons donc $\sigma_1 \to \Delta_1 \sqcap \langle\sigma_2\rangle_{\Delta_2}$ qui est défini et

$$\sigma_1' \to \Delta_1' \sqcap \langle\sigma_2'\rangle_{\Delta_2'} = \langle\sigma_1' \sqcup \sigma_2'\rangle_{\Delta_1' \sqcap \Delta_2'} \leq \langle\sigma_1 \sqcup \sigma_2\rangle_{\Delta_1 \sqcap \Delta_2} = \sigma_1 \to \Delta_1 \sqcap \langle\sigma_2\rangle_{\Delta_2}$$

Nous ne détaillons pas les autres cas qui sont similaires.

Nous prouvons la propriété (4). Nous procédons une fois de plus sur la somme des tailles de τ_1 et τ_2. La propriété est immédiate par définition pour les multi-ensembles pour les cas $\sqcup$ et $\sqcap$. Elle l'est aussi pour $\texttt{unit}$, $\texttt{dom}(w)$ ou une variable de type puisque dans ce cas là nous avons $\tau_1 = \tau_2$. Elle est enfin immédiate par induction pour les cas nuplets. Nous détaillons le cas des types fonctionnels et des types de ressources.

Si les deux types sont des types fonctionnels ou des types de ressources, la propriété est immédiate par induction. Nous considérons le cas $\langle\sigma_1\rangle_{\Delta_1}$ et $\sigma_2 \to \Delta_2$. Si $\langle\sigma_1\rangle_{\Delta_1} \sqcup \sigma_2 \to \Delta_2$ est défini, alors $\sigma_1 \sqcap \sigma_2$ est défini et $\Delta_1 \sqcup \Delta_2$ est défini. Par induction, nous avons donc $\sigma_1 \sqcap \sigma_2 \leq \sigma_1$ et $\Delta_1 \leq \Delta_1 \sqcup \Delta_2$. Nous avons donc $\langle\sigma_1\rangle_{\Delta_1} \leq (\sigma_1 \sqcap \sigma_2) \to (\Delta_1 \sqcup \Delta_2) = \langle\sigma_1\rangle_{\Delta_1} \sqcup \sigma_2 \to \Delta_2$. Le cas symétrique est identique. Si $\langle\sigma_1\rangle_{\Delta_1} \sqcap \sigma_2 \to \Delta_2$ est défini, alors $\sigma_1 \sqcup \sigma_2$ est défini et $\Delta_1 \sqcap \Delta_2$ est défini. Par induction, nous avons $\sigma_1 \leq \sigma_1 \sqcup \sigma_2$ et $\Delta_1 \sqcap \Delta_2 \leq \Delta_1$. Nous avons donc $\langle\sigma_1\rangle_{\Delta_1} \sqcap \sigma_2 \to \Delta_2 = \langle\sigma_1 \sqcup \sigma_2\rangle_{\Delta_1 \sqcap \Delta_2} \leq \langle\sigma_1\rangle_{\Delta_1}$. Le cas symétrique est identique.

Nous prouvons enfin la propriété (5). Nous procédons par induction sur la somme de la taille des deux types. Le résultat est immédiat pour `unit`, $\texttt{dom}(w)$ et les variables de types, puisque $\tau_1 = \tau_2 = \tau_1 \sqcap \tau_2 = \tau_1 \sqcup \tau_2$. Dans le cas des multi-ensembles, τ_1 et τ_2 sont nécessairement des multi-ensembles, et nous avons $fn(\Delta_1 \sqcup \Delta_2) = fn(\Delta_1) \cup fn(\Delta_2)$, et $fn(\Delta_1 \sqcap \Delta_2) \subseteq fn(\Delta_1) \cup fn(\Delta_2)$. Le cas des nuplets, des types fonctionnels et des types de ressources sont immédiats par induction. □

Lemme C.1.2 *Si $\Gamma \vdash \mathcal{S} : \Delta$, alors $fsv(\mathcal{S}) = ftv(\mathcal{S}) = fwv(\mathcal{S}) = \emptyset$. La même propriété est satisfaite par tous les jugements, excepté les contextes.*

Preuve: Cette propriété est immédiate par induction sur la dérivation de typage, puisque les seules constructions contenant des variables de types sont les restrictions de ressources, et leur typage vérifie que aucune variable de type n'est libre (règles Nu.Res ou Top.Nu.Res). □

Lemme C.1.3 *Si $\Gamma \vdash \mathcal{S} : \Delta$ alors $set(\Delta)$.*

Preuve: Nous procédons par induction sur la dérivation de typage de la configuration.

Top Immédiat par hypothèse de la règle de typage.

Top.Nu.Res Immédiat par induction.

Top.Nu.Dom La propriété est satisfaite puisque par hypothèse d'induction nous avons $set(\Delta)$, donc nous avons immédiatement $set(\Delta - a)$.

Top.Hole Immédiat par hypothèse de la règle de typage.

□

Lemme C.1.4 *Soit $\Gamma \vdash P : \tau$ la conclusion d'un jugement de typage. Nous avons $fn(\tau) \subseteq fn(\Gamma)$. Cette propriété est également satisfaite pour les configurations.*

Preuve: Nous procédons par induction sur la dérivation de typage.

Nil, Void Immédiat.

Name Les noms libres pouvant apparaître dans $\sigma\theta$ sont soit présents dans l'image de θ, et sont alors libres dans Γ (puisque dans son domaine) par hypothèse de la règle Name, soit présents dans σ et ils sont libres dans Γ (les noms ne sont pas généralisés).

Dest.Boss, Dest.Cont Immédiat puisque $fn(\texttt{dest}) = \emptyset$.

Addr Par induction nous avons $fn(\langle\sigma\rangle_\Delta) \subseteq fn(\Gamma)$. Nous avons donc immédiatement $fn(\sigma \to \Delta) = fn(\langle\sigma\rangle_\Delta) \subseteq fn(\Gamma)$.

Fun Par induction nous avons $fn(\tau) \subseteq fn(\Gamma + x : \sigma)$. Comme une variable x ne peut apparaître dans les types, et comme nous avons par hypothèse de la règle Fun $fn(\sigma) \subseteq dom(\Gamma)$, nous en déduisons $fn(\sigma \to \tau) = fn(\sigma) \cup fn(\tau) \subseteq fn(\Gamma)$.

Dom, Par, Tuple Immédiat par induction.

Nu.Res, Top.Nu.Res Par induction, nous avons $fn(\Delta_1) \subseteq fn(\Gamma + r : \forall\widetilde{\beta}.\langle\sigma\rangle_\Delta)$. Comme r ne peut apparaître dans les types, et comme $fn(\forall\widetilde{\beta}.\langle\sigma\rangle_\Delta) \subseteq dom(\Gamma)$, nous avons $fn(\Delta_1) \subseteq fn(\Gamma)$.

Nu.Dom, Top.Nu.Dom Nous prouvons tout d'abord que $a \notin \Delta - a$. C'est immédiat par hypothèse dans le cas de la règle Nu.Dom. Dans le cas Top.Nu.Dom, nous utilisons le lemme C.1.3 sur la dérivation $\Gamma + a : \texttt{dom}(a) \vdash \mathcal{S} : \Delta$ pour en déduire $set(\Delta)$, donc a est présent au plus une fois dans Δ, donc $a \notin \Delta - a$.

Nous avons par hypothèse d'induction $fn(\Delta) \subseteq fn(\Gamma) \cup \{a\}$. Puisque nous avons $a \notin \Delta - a$, nous avons donc $fn(\Delta - a) \subseteq fn(\Gamma)$.

Pass Immédiat par induction puisque $fn(\Delta) \cup fn(\Delta_1) \cup fn(\Delta_2) \subseteq fn(\Gamma)$.

App Immédiat par induction puisque $fn(\tau) \subseteq fn(\sigma \to \tau) \subseteq fn(\Gamma)$, ou $fn(\Delta) \subseteq fn(\langle\sigma\rangle_\Delta) \subseteq fn(\Gamma)$.

Test Nous avons $fn(\tau_1 \sqcup \tau_2) \subseteq fn(\tau_1) \cup fn(\tau_2)$ par la proposition C.1.1(5). Nous concluons par induction.

Def Immédiat.

Top Immédiat par induction.

□

Lemme C.1.5 *Soit $\Gamma \vdash P : \tau$ la conclusion d'une dérivation de typage, et n un nom ou une variable. Soit n' un nom n'apparaissant pas dans la dérivation de typage, tel que n' est un nom de domaine si n est un nom de domaine, n' est un nom de ressource si n est un nom de ressource, ou n' est une variable si n est une variable. Nous avons $\Gamma\{^{n'}/_{n}\} \vdash P\{^{n'}/_{n}\} : \tau\{^{n'}/_{n}\}$. La même propriété est satisfaite pour le typage des configurations et des définitions.*

Preuve: Nous procédons par induction sur la dérivation de typage.

Nil, Void Immédiat.

Name Les noms ni les variables ne pouvant être généralisées, nous avons immédiatement $u\{^{n'}/_{n}\} : \forall\widetilde{\alpha}\widetilde{\delta}\widetilde{\rho}.\sigma\{^{n'}/_{n}\} \in \Gamma\{^{n'}/_{n}\}$. Nous considérons l'instanciation $\theta\{^{n'}/_{n}\}$ et nous montrons que nous avons $\sigma\{^{n'}/_{n}\}\theta\{^{n'}/_{n}\} = \sigma\theta\{^{n'}/_{n}\}$. C'est immédiatement le cas puisque le domaine de θ ne comprend que des variables de types, et puisque n' est un nom frais donc $\{^{n'}/_{n}\}\{^{n'}/_{n}\} = \{^{n'}/_{n}\}$. Il nous reste à montrer que $\mathit{fn}(\mathit{ran}(\theta\{^{n'}/_{n}\})) \subseteq \mathit{dom}(\Gamma\{^{n'}/_{n}\})$. C'est le cas parce que si n n'est pas libre dans $\mathit{ran}(\theta)$, nous avons alors $\mathit{fn}(\mathit{ran}(\theta\{^{n'}/_{n}\})) = \mathit{fn}(\mathit{ran}(\theta)) \subseteq \mathit{dom}(\Gamma)$ et nous concluons en remarquant que comme n n'est pas libre dans $\mathit{fn}(\mathit{ran}(\theta))$ nous avons également $\mathit{fn}(\mathit{ran}(\theta)) \subseteq \mathit{dom}(\Gamma\{^{n'}/_{n}\})$. Si n est dans $\mathit{fn}(\mathit{ran}(\theta))$ alors le résultat est immédiat, puisqu'il est alors dans $\mathit{dom}(\Gamma)$.

Nous concluons par la règle Name.

Proc.Hole Immédiat.

Dest.Boss, Dest.Cont Immédiat par induction, puisque nous remplaçons un nom de domaine par un nom de domaine ou une variable par une variable.

Addr Immédiat par induction, puisque la substitution conserve le statut du nom (de domaine, de ressource ou de variable).

Fun Si n est x, alors la substitution ne change rien, puisque les variables n'apparaissent pas dans les types et puisque x n'est pas dans le domaine de Γ. Sinon le résultat est immédiat par induction puisque n' n'apparaît pas dans la dérivation initiale, donc la condition $x \notin \mathit{dom}(\Gamma\{^{n'}/_{n}\})$ est satisfaite. La condition sur les noms libres de σ est elle aussi préservée par le renommage.

Dom Immédiat par induction puisque le statut de $\mathbf{a}$ est préservé par la substitution.

Par, Tuple Immédiat par induction.

Nu.Res Si n est r, alors la substitution ne fait rien puisque les noms de ressources n'apparaissent pas dans les types, et puisque $r \notin \mathit{dom}(\Gamma)$. Sinon le résultat est immédiat par induction, puisque n' étant frais nous avons bien $r \notin \mathit{dom}(\Gamma\{^{n'}/_{n}\})$. De plus, comme nous avons $\mathit{fn}(\forall\widetilde{\beta}.\langle\sigma\rangle_{\Delta}) \subseteq \mathit{dom}(\Gamma)$, nous avons $\mathit{fn}(\forall\widetilde{\beta}.\langle\sigma\{^{n'}/_{n}\}\rangle_{\Delta\{^{n'}/_{n}\}}) \subseteq \mathit{dom}(\Gamma\{^{n'}/_{n}\})$ puisque ni n ni n' ne peuvent être généralisés.

Nu.Dom Si n est a, alors la substitution ne fait rien puisque a n'apparaît libre ni dans Γ, ni dans le type $\Delta - a$ par hypothèse de la règle de typage. Sinon, le résultat est immédiat par induction puisque n' est frais et ne peut donc être a.

Pass Nous remarquons que ni n' ni n ne peuvent être ρ_1 ou ρ_2 qui sont des variables de multi-ensembles. Nous montrons que nous avons $(\Delta - w, \rho_1, \rho_2)\{^{n'}/_{n}\} = \Delta\{^{n'}/_{n}\} - (w, \rho_1, \rho_2)\{^{n'}/_{n}\}$. C'est effectivement le cas puisque n' est frais. La propriété est donc vraie par induction.

App Immédiat par induction, puisque la structure des types et le sous-typage sont préservés par renommage des noms de domaines.

Test Immédiat par induction puisque les opérateurs $\sqcup$ et $\sqcap$ commutent avec le renommage d'un nom de domaine vers un nom de domaine frais.

Def,And Immédiat par induction.

Def.Hole,Def.$\top$ Immédiat.

Join Si n est un des x_i, la substitution ne fait rien (puisque les noms de variables sont différents des noms de ressources et n'apparaissent pas dans les types, et puisque x_i n'est pas dans le domaine de Γ). Sinon le résultat est immédiat par induction puisque le renommage est vers des noms frais et n'implique pas de variable de type.

Top Immédiat par induction, puisque renommer un nom de domaine en un nom frais préserve le prédicat *set*.

Top.Nu.Res Identique au cas Nu.Res.

Top.Nu.Dom Identique au cas Nu.Dom, en utilisant le lemme C.1.3 pour montrer que $a \notin \Delta - a$, puisque $\Gamma + a : \mathtt{dom}(a) \vdash \mathcal{S} : \Delta$ implique $\mathit{set}(\Delta)$, donc a est présent au plus une fois dans Δ.

Top.Hole Immédiat puisque n' est frais.

□

Dans le lemme suivant, nous notons $\nu n.\ldots$ pour $\nu r : s.\ldots$ ou $\nu a.\ldots$. Nous disons qu'un nom a le même statut qu'un autre si ce sont tous les deux des noms de domaines, de ressource ou de variable.

Lemme C.1.6 (α-conversion) *Si $\Gamma \vdash \nu n.\mathcal{S} : \Delta$ (respectivement $\Gamma \vdash \nu n.P : \tau$), pour tout nom frais n' de même statut que n nous avons $\Gamma \vdash \nu n'.(\mathcal{S}\{^{n'}/_{n}\}) : \Delta$ (respectivement $\Gamma \vdash \nu n'.(P\{^{n'}/_{n}\}) : \tau$).*

Si $\Gamma \vdash \lambda x.P : \tau$, pour tout variable fraîche y nous avons $\Gamma \vdash \lambda y.P\{^{y}/_{x}\} : \tau$.

Si $\Gamma \vdash r_1\widetilde{x_1} \mid \ldots \mid r_n\widetilde{x_n} \triangleright P$, pour tout nuplet de variables fraîches $(\widetilde{y_i})^{i\in[1..n]}$ (de même taille que $\widetilde{x_i}$), nous avons $\Gamma \vdash r_1\widetilde{y_1} \mid \ldots \mid r_n\widetilde{y_n} \triangleright P\{^{\widetilde{y_i}}/_{\widetilde{x_i}}\}$.

Preuve: Dans chacun des cas, la dernière règle de typage utilisée est nécessairement Top.Nu.Res pour un nom de ressource et une configuration, Nu.Res pour un nom de ressource et un processus, Top.Nu.Dom pour un nom de domaine et une configuration, Nu.Dom pour un nom de domaine et un processus, Fun pour une λ-abstraction et Join pour une règle de réaction. Dans chaque cas nous appliquons au typage du processus gardé ou lié le lemme C.1.5 (plusieurs fois dans le cas d'une règle de réaction). Dans le cas d'une λ-abstraction ou d'une ressource, le type obtenu est le même puisque ni les noms de ressources ni les variables n'apparaissent dans les types, et l'environnement Γ est inchangé. Nous en dérivons la conclusion attendue par la règle Fun, Top.Nu.Res ou Nu.Res, selon le cas, puisque le nom est frais (dans les cas Nu.Res et Top.Nu.Res, l'α-conversion ne modifie pas la condition sur les noms libres du type de la ressource). Pour le cas de la règle de réaction, le type obtenu ainsi que l'environnement Γ sont eux aussi inchangés. Les noms substitués étant frais, il ne sont pas présents dans le domaine de Γ et les autres conditions sur les types sont inchangées. Nous concluons par la règle Join. En ce qui concerne le cas d'un nom de domaine et d'un processus, nous savons par hypothèse de la règle Nu.Dom que a n'est pas libre dans Γ, donc celui-ci est inchangé par la substitution. De plus, a est présent au plus une fois dans Δ, donc $\Delta\{^{a'}/_{a}\} - a' = \Delta - a$ et le type final est identique. Nous concluons par la règle Nu.Dom puisque a' est frais (donc ne peut être libre dans Γ). Enfin, le cas Top.Nu.Dom est identique à la différence près que nous utilisons le lemme C.1.3 sur la dérivation $\Gamma + a : \mathtt{dom}(a) \vdash \mathcal{S} : \Delta$ pour montrer que a est présent au plus une fois dans Δ, donc n'est pas présent dans $\Delta - a$. □

Proposition C.1.7 (Sous-type et passivation) *Soit σ un type de la forme $(\mathtt{unit} \to \Delta_1) \to (\mathtt{unit} \to \Delta_2) \to \Delta$ ou $(\mathtt{unit} \to \Delta_1) \to \langle \mathtt{unit} \to \Delta_2\rangle_\Delta$. Si nous avons $\sigma' \leq \sigma$, alors σ' est de la forme $(\mathtt{unit} \to \Delta'_1) \to (\mathtt{unit} \to \Delta'_2) \to \Delta'$ ou $(\mathtt{unit} \to \Delta'_1) \to \langle \mathtt{unit} \to \Delta'_2\rangle_{\Delta'}$ avec $\Delta' \subseteq \Delta$, $\Delta_1 \subseteq \Delta'_1$ et $\Delta_2 \subseteq \Delta'_2$.*

Preuve: Nous supposons que σ a la forme $(\mathtt{unit} \to \Delta_1) \to (\mathtt{unit} \to \Delta_2) \to \Delta$ (l'autre cas est plus simple, nous ne le détaillons pas). Puisque $\sigma' \leq \sigma$, nous avons nécessairement $\sigma' = \sigma_1 \to \tau_2$ avec $\mathtt{unit} \to \Delta_1 \leq \sigma_1$ et $\tau_2 \leq (\mathtt{unit} \to \Delta_2) \to \Delta$ (on ne peut appliquer le cas d'une ressource étant sous-type d'une fonction, puisque le résultat de la première abstraction n'est pas un multi-ensemble). Nous avons donc nécessairement $\sigma_1 = \sigma'_1 \to \tau'_1$ avec $\sigma'_1 \leq \mathtt{unit}$ (donc $\sigma'_1 = \mathtt{unit}$) et $\Delta_1 \leq \tau'_1$ (donc $\tau'_1 = \Delta'_1$ avec $\Delta_1 \subseteq \Delta'_1$). Le type σ' est donc de la forme $(\mathtt{unit} \to \Delta'_1) \to \tau_2$. Comme nous avons $\tau_2 \leq (\mathtt{unit} \to \Delta_2) \to \Delta$, nous avons τ_2 qui est soit de la forme $\sigma_2 \to \Delta'$, soit de la forme $\langle\sigma_2\rangle_{\Delta'}$, avec dans les deux cas $\Delta' \leq \Delta$ (donc $\Delta' \subseteq \Delta$) et $\mathtt{unit} \to \Delta_2 \subseteq \sigma_2$. Par le même raisonnement que pour σ_1, nous en déduisons que $\sigma_2 = \mathtt{unit} \to \Delta'_2$ avec $\Delta_2 \subseteq \Delta'_2$. □

Lemme C.1.8 (Typage et contextes) *Soit $C(\cdot : \Delta)$ un contexte pour configurations (respectivement $C(\cdot : \tau)$ un contexte pour processus et $C(\cdot)$ un contexte pour définitions) et $\mathcal{S}$ une configuration (respectivement P un processus et D une définition) tels que $\Gamma \vdash C(\cdot : \Delta) : \Delta_1$ (respectivement $\Gamma \vdash C(\cdot : \tau) : \Delta_1$ et $\Gamma \vdash C(\cdot) : \Delta_1$) et $\Gamma' \vdash \mathcal{S} : \Delta'$ (respectivement $\Gamma' \vdash P : \tau'$ et $\Gamma' \vdash D$) avec $\Delta' \leq \Delta$ (respectivement $\tau' \leq \tau$) où Γ' est l'environnement de typage utilisé dans la règle typant le trou.*

Nous avons alors $\Gamma \vdash C(\mathcal{S}) : \Delta'_1$ (respectivement $\Gamma \vdash C(P) : \Delta'_1$ et $\Gamma \vdash C(D) : \Delta'_1$) avec $\Delta'_1 \subseteq \Delta_1$.

La même propriété est satisfaite si le terme obtenu en bouchant le trou est un processus ou une définition. De plus, nous avons la propriété que le type final est le même (et non plus un sous-type de Δ_1) si le type du terme bouchant le trou est le même que le type du trou.

Preuve: Nous procédons par induction sur la taille du contexte, en utilisant le fait que le typage du contexte est dirigé par la syntaxe. La propriété d'égalité des types résultats si le type du trou est le même que le type du terme le bouchant est immédiate pour chaque cas, et n'est pas détaillée.

$\nu a.C$ Immédiat par induction en utilisant la règle Top.Nu.Dom si C est une configuration, ou la règle Nu.Dom si C est un processus. Dans ce deuxième cas, si $\Delta' \subseteq \Delta$ et $a \notin \Delta - a$, nous avons nécessairement $a \notin \Delta' - a$ et $\Delta' - a \subseteq \Delta - a$.

$\nu r : s.C$ Immédiat par induction en utilisant la règle de typage Top.Nu.Res si C est une configuration, et Nu.Res si C est un processus.

$\epsilon[C]$ Immédiat par induction en utilisant la règle Top. Le prédicat *set* étant préservé si $\Delta' \subseteq \Delta$.

$(\cdot : \Delta)$ Comme nous typons le trou, nous avons $\Gamma' = \Gamma$. Par hypothèse, nous avons $\Gamma' \vdash \mathcal{S} : \Delta' \leq \Delta$, qui est le résultat escompté.

$(\cdot : \tau)$ Comme dans le cas précédent, nous avons $\Gamma' = \Gamma$ et par hypothèse $\Gamma' \vdash P : \tau' \leq \tau$.

$\lambda x.C$ Immédiat par induction en utilisant la règle de typage Fun, puisque $\tau' \leq \tau$ implique $\sigma \to \tau' \leq \sigma \to \tau$ par définition. La condition sur les noms libres de σ est inchangée.

$\mathbf{a}(C)[Q]$ **et** $\mathbf{a}(P)[C]$ Immédiat par induction, en utilisant la règle de typage Dom.

$C \mid Q$ **et** $P \mid C$ Immédiat par induction, en utilisant la règle de typage Par.

$P_1, \ldots, P_q$ Immédiat par induction en utilisant la règle de typage Tuple.

$\mathtt{pass}_{\mathbf{a}}\ C$ La règle de typage s'appliquant est Pass. Par induction nous avons donc un typage (en prenant le cas du processus) $\Gamma \vdash C(P) : \sigma'_V$ avec $\sigma'_V \leq \sigma_V$. Nous appliquons la proposition C.1.7 pour en déduire que σ'_V a la forme escomptée, que ρ_1 est bien présent dans Δ'_1 (puisque $\rho_1 \in \Delta_1 \subseteq \Delta'_1$), que ρ_2 est bien présent dans Δ'_2, et que ρ_1 et ρ_2 ne sont pas dans $\Delta' - \rho_1, \rho_2$ (puisque $\Delta' \subseteq \Delta$). Les autres conditions sont donc immédiatement satisfaites, et nous avons $\Delta' - (w, \rho_1, \rho_2) \subseteq \Delta - (w, \rho_1, \rho_2)$.

CQ **et** PC Dans le premier cas, par induction nous avons $\Gamma \vdash C(P) : \sigma'_P$ avec $\sigma'_P \leq \sigma_P$. Par définition de la relation de sous-typage, nous avons nécessairement $\sigma'_P = \langle \sigma'' \rangle_{\Delta'}$ ou $\sigma'_P = \sigma'' \to \tau'$ avec (en prenant $\Delta' = \tau'$) $\sigma \leq \sigma''$ et $\tau' \leq \tau$. Par transitivité de la relation de sous-typage (proposition C.1.1(1)), nous avons bien $\sigma' \leq \sigma''$ et nous concluons par la règle App pour obtenir un type $\tau' \leq \tau$.

Le deuxième cas est immédiat par induction, utilisant une fois encore la transitivité de la relation de sous-typage.

$[pat = C]P, Q$**,** $[pat = V]C, Q$ **et** $[pat = V]P, C$ La seule règle de typage s'appliquant est Test. Le résultat est immédiat par induction puisque le sous-typage est stable par $\sqcup$ (proposition C.1.1(3) qui indique que $\tau'_1 \sqcup \tau'_2$ existe et est un sous-type de $\tau_1 \sqcup \tau_2$).

$\langle C \rangle$ La seule règle de typage s'appliquant étant Def, le résultat est immédiat par induction.

C, D' **et** D, C Immédiat par induction (règle de typage And).

$(\cdot)$ Immédiat avec $\Gamma' = \Gamma$.

$J \triangleright C$ La seule règle de typage s'appliquant étant Join, le résultat est immédiat par induction en utilisant la transitivité du sous-typage (proposition C.1.1(1)).

□

Dans la suite de cette section, nous travaillons à α-renommage près des noms ou variables liées vers des variables fraîches, et justifions ce choix par les lemmes C.1.6 et C.1.8.

Lemme C.1.9 (Instanciation des variables de types) *Soit $\Gamma \vdash P : \tau$ la conclusion d'une dérivation de typage, et θ une substitution des variables de types vers les types, des variables de types de noms vers les variables de types de noms ou les noms de domaines, et des variables de multi-ensembles vers les multi-ensembles, telle que $\theta\theta = \theta$ et $fn(ran(\theta)) \subseteq dom(\Gamma\theta) = dom(\Gamma)$. Nous avons alors $\Gamma\theta \vdash P : \tau' \leq \tau\theta$. La même propriété est vraie pour les configurations et les définitions. De plus, si θ renomme injectivement les variables de types de noms vers des variables de types de noms et les variables de multi-ensembles vers des variables de multi-ensembles qui ne sont pas présentes dans la dérivation de typage, nous avons $\tau' = \tau\theta$*

Preuve: Nous remarquons tout d'abord qu'aucune variable de type ne peut être libre dans P, par le lemme C.1.2. Nous procédons par induction sur la dérivation de typage, quelle que soit la substitution. Le cas d'égalité entre τ et τ' est immédiat par induction pour tous les cas, excepté pour les règles de typage Pass et Test.

Nil,Void Immédiat.

Name Nous supposons que les variables généralisées de $s = \forall\widetilde{\alpha}\widetilde{\delta}\widetilde{\rho}.\sigma$ sont toutes distinctes du domaine et de l'image de θ. Nous notons φ l'instanciation de s en $\sigma\varphi$. Nous avons $u : \forall\widetilde{\alpha}\widetilde{\delta}\widetilde{\rho}.\sigma\theta \in \Gamma\theta$. De plus, la substitution $\varphi\theta$ est une instanciation de s en $\sigma\varphi\theta$. Nous montrons que nous avons $\sigma\theta\varphi\theta = \sigma\varphi\theta$. Nous considérons tout d'abord une variable de type généralisée de s, par exemple α. Par hypothèse, α n'est pas dans le domaine de θ donc $\alpha\theta = \alpha$. Sinon, soit α une variable de type qui n'est pas généralisée (donc $\alpha\varphi = \alpha$). L'image de θ étant distincte des variables généralisées, nous avons $\alpha\theta\varphi = \alpha\theta$, donc $\alpha\theta\varphi\theta = \alpha\theta\theta = \alpha\theta$, donc $\alpha\theta\varphi\theta = \alpha\varphi\theta$.

Nous montrons enfin que $fn(ran(\varphi\theta)) \subseteq dom(\Gamma\theta)$. C'est immédiatement le cas puisque $fn(ran(\varphi\theta)) \subseteq fn(ran(\varphi)) \cup fn(ran(\theta)) \subseteq dom(\Gamma) \cup dom(\Gamma\theta) = dom(\Gamma\theta)$.

Dest.Boss, Dest.Cont Immédiat par induction et par définition de $\leq$, puisque `dest` est le seul sous-type de `dest` et $\texttt{dom}(w)$ est le seul sous-type de $\texttt{dom}(w)$.

Addr Immédiat par induction puisque tout sous-type d'un type de ressource est nécessairement un type de ressource, et que le seul sous-type de `dest` est `dest`.

Fun Immédiat par induction, les variables n'étant pas présentes dans les types. Nous concluons en remarquant que si $\tau' \leq \tau\theta$ alors $\sigma\theta \to \tau' \leq \sigma\theta \to \tau\theta$, et que un nom de $fn(\sigma\theta)$ est soit libre dans l'image de θ et alors présent dans le domaine de $\Gamma\theta$ par hypothèse du lemme, soit présent dans $fn(\sigma)$ donc dans le domaine de Γ par hypothèse de la règle de typage (donc présent dans le domaine de $\Gamma\theta$, θ ne renommant pas les noms de domaines).

Dom, Par, Tuple Immédiat par induction.

Nu.Res, Top.Nu.Res Immédiat par induction, le type associé à r n'étant pas modifié (il ne contient aucune variable de type libre), les noms de ressources n'apparaissant pas dans les types, et $dom(\Gamma\theta) = dom(\Gamma)$.

Nu.Dom, Top.Nu.Dom Immédiat par induction, après α-conversion de a s'il est présent dans l'image de θ.

Pass Si les variables ρ_1 ou ρ_2 sont présentes dans le domaine ou l'image de θ, nous les renommons en utilisant l'hypothèse d'induction en des noms frais distincts ρ'_1 et ρ'_2 qui ne sont présents ni dans la dérivation de typage initiale, ni dans le domaine et l'image de θ. Nous avons $fn(ran(\{{}^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\})) = \emptyset$. Les variables ρ_1 et ρ_2 n'étant pas libres dans Γ par hypothèse de la règle de typage, nous avons donc (nous sommes dans le cas d'égalité du type résultat) :

$$\Gamma \vdash V : \sigma_V\{{}^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\}$$

Comme ρ_1 et ρ_2 sont présents au plus une fois dans Δ et comme ρ'_1 et ρ'_2 sont frais, ces derniers sont présents au plus une fois dans $\Delta\{{}^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\}$, et $\Delta - (w, \rho_1, \rho_2) = \Delta\{{}^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\} - (w, \rho'_1, \rho'_2)$.

Nous utilisons maintenant l'hypothèse d'induction avec θ et nous obtenons :

$$\Gamma\theta \vdash V : \sigma'_V$$

avec $\sigma'_V \leq \sigma_V\{{}^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\}\theta$. Par la proposition C.1.7, nous avons (si $\sigma'_V = (\texttt{unit} \to \Delta'_1) \to (\texttt{unit} \to \Delta'_2) \to \Delta'$ ou $\sigma'_v = (\texttt{unit} \to \Delta'_1) \to \langle \texttt{unit} \to \Delta'_2 \rangle_{\Delta'}$) la propriété que σ'_V possède la forme escomptée, et les inclusions suivantes : $\Delta_1\{{}^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\}\theta \subseteq \Delta'_1$, $\Delta_2\{{}^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\}\theta \subseteq \Delta'_2$, $\Delta' \subseteq \Delta\{{}^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\}\theta$.

Comme nous avons $\rho_1 \in \Delta_1$, nous avons $\rho'_1 \in \Delta_1\{{}^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\}$. Comme ρ'_1 n'est ni dans le domaine, ni dans l'image de θ, nous avons $\rho'_1 \in \Delta_1\{{}^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\}\theta$. Donc nous avons $\rho'_1 \in \Delta'_1$. Par un raisonnement analogue, nous avons $\rho'_2 \in \Delta'_2$.

Nous avons également $\rho'_1, \rho'_2 \notin fsv(\Gamma\theta) \cup (\Delta' - \rho'_1, \rho'_2)$.

Nous avons aussi par induction :

$$\Gamma\theta \vdash \mathbf{a} : \texttt{dom}(w\theta)$$

puisque le seul sous-type de $\texttt{dom}(w\theta)$ est lui-même.

Nous pouvons appliquer la règle Pass. Nous avons l'inclusion suivante :

$$\begin{aligned} & \Delta' - (w\theta, \rho'_1, \rho'_2) \subseteq \Delta\{{}^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\}\theta - (w\theta, \rho'_1, \rho'_2) \\ \subseteq\ & (\Delta\{{}^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\} - (w, \rho'_1, \rho'_2))\theta = (\Delta - (w, \rho_1, \rho_2))\theta \end{aligned}$$

Dans le cas où la substitution des variables de types de noms se fait vers des variables de types de noms n'étant pas présentes dans la dérivation, nous avons $\sigma'_V = \sigma_V\{^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\}\theta$ et :

$$\begin{aligned}&\Delta' - (w\theta, \rho'_1, \rho'_2) = \Delta\{^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\}\theta - (w\theta, \rho'_1, \rho'_2)\\ =\ & (\Delta\{^{\rho'_1,\rho'_2}/_{\rho_1,\rho_2}\} - (w, \rho'_1, \rho'_2))\theta = (\Delta - (w, \rho_1, \rho_2))\theta\end{aligned}$$

App Par induction nous avons $\Gamma\theta \vdash P : \sigma'_P$ avec $\sigma'_P \leq \sigma_P\theta$. Par définition de la relation de sous-typage, nous avons nécessairement $\sigma'_P = \langle\sigma''\rangle_{\Delta'}$ ou $\sigma'_P = \sigma'' \to \tau'$ avec (en prenant $\Delta' = \tau'$) $\sigma\theta \leq \sigma''$ et $\tau' \leq \tau\theta$.

Nous avons également par induction $\Gamma\theta \vdash Q : \sigma'_Q \leq \sigma'\theta$ avec $\sigma'\theta \leq \sigma\theta$ par la propriété C.1.1(2). Nous avons donc par transitivité du sous-typage $\sigma'_Q \leq \sigma''$, et nous obtenons par application de la règle App :

$$\Gamma\theta \vdash PQ : \tau' \leq \tau\theta$$

Test Nous montrons tout d'abord que si $\tau_1 \sqcup \tau_2$ (respectivement $\tau_1 \sqcap \tau_2$) est défini, alors $\tau_1\theta \sqcup \tau_2\theta$ (respectivement $\tau_1\theta \sqcap \tau_2\theta$) est défini et $\tau_1\theta \sqcup \tau_2\theta \leq (\tau_1 \sqcup \tau_2)\theta$ (respectivement $(\tau_1 \sqcap \tau_2)\theta \leq \tau_1\theta \sqcap \tau_2\theta$). Ceci est immédiat par induction sur la somme de la taille des types, et se base sur le fait que $(\Delta, \Delta')\theta = \Delta\theta, \Delta'\theta$, que $(\Delta \cap \Delta')\theta \subseteq \Delta\theta \cap \Delta'\theta$ (nous avons l'égalité dans le cas du renommage injectif vers des variables fraîches), et que $\tau \sqcup \tau = \tau \sqcap \tau = \tau$.

Nous pouvons donc conclure par induction, en utilisant la propriété C.1.1(3) et la transitivité du sous-typage.

Def,And Immédiat par induction.

Def.$\top$ Immédiat.

Join Immédiat par induction après avoir renommé les variables généralisées de telle sorte qu'elles soient distinctes du domaine et de l'image de θ. Les conditions de généralisation sont inchangées, ainsi que la condition sur les paramètres formels. Nous avons ainsi $(r_i : s_i = \forall\widetilde{\alpha_i}\widetilde{\delta_i}\widetilde{\rho_i}.\widetilde{\sigma_i}\theta \to \Delta_i\theta \in \Gamma\theta)^{i\in[1..n]}$, $\Gamma\theta + \widetilde{x_1} : \widetilde{\sigma_1}\theta + \ldots + \widetilde{x_n} : \widetilde{\sigma_n}\theta \vdash P : \Delta'' \subseteq \Delta'\theta$ et $\Delta'' \subseteq \Delta'\theta \subseteq \Delta_1\theta, \ldots, \Delta_n\theta$. Nous concluons par la règle Join.

Top Immédiat, puisque $set(\Delta)$ implique que $\Delta\theta = \Delta$ (Δ ne contient aucune variable de types de noms ni aucune variable de multi-ensembles).

□

Lemme C.1.10 (Extension de l'environnement de typage) *Soit $\Gamma \vdash P : \tau$ la conclusion d'une dérivation de typage, et u un nom tel que $u \notin dom(\Gamma)$, nous avons alors $\Gamma + u : \sigma \vdash P : \tau$.*

Preuve: Nous procédons par induction sur la dérivation de typage. La plupart des cas sont immédiats, nous les omettons.

Fun Si x est u, nous l'α-renommons préalablement en une variable fraîche (x ne peut être présent dans σ). La condition sur les noms libres de σ est immédiatement satisfaite.

Nu.Res Si r est u, nous l'α-renommons préalablement en un nom de ressource frais (r ne peut être présent dans σ). La condition sur les noms libres du type de la ressource est immédiatement satisfaite.

Nu.Dom Si a est u ou si a est libre dans σ, nous l'α-renommons préalablement en un nom de domaine frais.

Pass Si ρ_1 ou ρ_2 est libre dans σ, nous les renommons en des noms de variables frais par le lemme C.1.9. Comme dans le cas Pass de ce lemme, la conclusion de la règle est identique après renommage. Nous utilisons ensuite l'hypothèse d'induction pour conclure.

Join Si u est un des $\widetilde{x_i}$, nous α-renommons préalablement les paramètres formels du filtre. De plus, si une des variables généralisées est libre dans σ, nous la renommons également. Nous concluons par induction sur le typage du processus gardé.

□

Lemme C.1.11 (Renforcement de l'environnement de typage) *Soit $\Gamma + a : \mathtt{dom}(a) \vdash P : \tau$ la conclusion d'une dérivation de typage, a un nom tel que $a \notin fn(P) \cup dom(\Gamma)$. Nous avons alors $\Gamma\{^{\emptyset}/_a\} \vdash P : \tau\{^{\emptyset}/_a\}$.*

Soit $\Gamma + r : s \vdash P : \tau$ la conclusion d'un jugement de typage et r un nom de ressource tel que $r \notin fn(P)$. Nous avons alors $\Gamma \vdash P : \tau$.

Les mêmes propriétés sont vraies pour les configurations et les définitions.

Preuve: Nous remarquons tout d'abord que $\Gamma\{^{\emptyset}/_{a}\}$ est un environnement de typage puisque $a \notin dom(\Gamma)$. Nous avons donc $dom(\Gamma) = dom(\Gamma\{^{\emptyset}/_{a}\})$.

Nous procédons par induction sur la dérivation de typage. Nous prouvons les deux propriétés en parallèle.

NIL,VOID Immédiat.

NAME Dans le cas d'un nom de ressource, le nom u typé est nécessairement différent de r par hypothèse du lemme. Nous remarquons que les seuls noms libres présents dans $ran(\theta)$ sont des noms de domaines, la condition $fn(ran(\theta)) \subseteq dom(\Gamma)$ est donc immédiatement satisfaite.

Dans le cas d'un nom de domaine, nous avons $u : \forall\widetilde{\alpha}\widetilde{\delta}\widetilde{\rho}.\sigma \in \Gamma$ (a étant différent de u par hypothèse du lemme). Nous avons donc $u : \forall\widetilde{\alpha}\widetilde{\delta}\widetilde{\rho}.\sigma\{^{\emptyset}/_{a}\} \in \Gamma\{^{\emptyset}/_{a}\}$. Si la substitution réalisant l'instanciation est initialement θ, nous considérons la substitution $\theta\{^{\emptyset}/_{a}\}$. Cette substitution est bien une instanciation, et nous avons $\sigma\{^{\emptyset}/_{a}\}\theta\{^{\emptyset}/_{a}\} = \sigma\theta\{^{\emptyset}/_{a}\}$ ainsi que $fn(ran(\theta\{^{\emptyset}/_{a}\})) \subseteq dom(\Gamma\{^{\emptyset}/_{a}\}) = dom(\Gamma)$ puisque $fn(ran(\theta)) \subseteq dom(\Gamma) \cup \{a\}$.

Nous concluons par la règle NAME que $\Gamma\{^{\emptyset}/_{a}\} \vdash u : \sigma\theta\{^{\emptyset}/_{a}\}$.

DEST.BOSS, DEST.CONT Immédiat par induction.

ADDR Immédiat par induction.

FUN Nous pouvons appliquer l'hypothèse d'induction dans le cas d'un nom de ressource puisque r ne peut pas être une variable, nous avons donc $r \in fn(P) \iff r \in fn(\lambda x.P)$. Comme nous supprimons un nom de ressource du domaine de Γ, cela n'invalide pas la condition sur les noms libres de σ.

Nous pouvons appliquer l'induction en ce qui concerne un nom de domaine a puisque a ne peut pas être une variable, donc nous avons bien $a \notin dom(\Gamma + x : \sigma)$ et $a \in fn(P) \iff a \in fn(\lambda x.P)$. Nous avons donc :

$$\Gamma\{^{\emptyset}/_{a}\} + x : \sigma\{^{\emptyset}/_{a}\} \vdash P : \tau\{^{\emptyset}/_{a}\}$$

Nous vérifions que $fn(\sigma\{^{\emptyset}/_{a}\}) \subseteq dom(\Gamma\{^{\emptyset}/_{a}\}) = dom(\Gamma)$. C'est le cas puisque nous avons $fn(\sigma) \subseteq dom(\Gamma) \cup \{a\}$. Nous pouvons conclure par la règle FUN.

DOM, PAR, TUPLE Immédiat par induction

NU.RES, TOP.NU.RES Nous remarquons tout d'abord que le nom restreint ne peut pas être r dans le cas d'un nom de ressource, puisque celui-ci est présent dans le domaine de l'environnement de typage. Donc nous avons $r \notin fn(P)$. Le résultat est alors immédiat par induction.

En ce qui concerne le cas d'un nom de domaine, nous remarquons que puisque nous avons $a \notin fn(\nu r : \forall\beta.\langle\sigma\rangle_{\Delta}.P)$, nous avons nécessairement $a \notin fn(P)$ (a n'est pas un nom de ressource), et $a \notin fn(\forall\beta.\langle\sigma\rangle_{\Delta})$. Par hypothèse d'induction, nous avons donc le typage :

$$\Gamma\{^{\emptyset}/_{a}\} + r : \forall\beta.\langle\sigma\rangle_{\Delta} \vdash P : \Delta_1\{^{\emptyset}/_{a}\}$$

et nous concluons par la règle NU.RES ou TOP.NU.RES.

NU.DOM, TOP.NU.DOM Le cas d'un nom de ressource est immédiat par induction.

En ce qui concerne le cas d'un nom de domaine, nous appelons b le nom de domaine restreint. Nous avons nécessairement $b \neq a$ puisque a est présent dans l'environnement de typage initial, donc nous avons bien $a \notin fn(P)$ et $a \notin dom(\Gamma + b : \mathtt{dom}(b))$. Nous pouvons appliquer l'hypothèse d'induction pour obtenir :

$$\Gamma\{^{\emptyset}/_{a}\} + b : \mathtt{dom}(b) \vdash P : \Delta\{^{\emptyset}/_{a}\}$$

Le nom b étant différent de a, nous avons $(\Delta - b)\{^{\emptyset}/_{a}\} = \Delta\{^{\emptyset}/_{a}\} - b$. Les autres conditions des règles NU.DOM et TOP.NU.DOM sont satisfaites et nous pouvons conclure.

PASS Immédiat par induction dans le cas d'un nom de ressource.

Dans le cas d'un nom de domaine, nous avons par induction :

$$\Gamma\{^{\emptyset}/_{a}\} \vdash V : \sigma_V\{^{\emptyset}/_{a}\}$$

avec $\sigma_V\{^{\emptyset}/_{a}\} = (\mathtt{unit} \to \Delta_1\{^{\emptyset}/_{a}\}) \to (\mathtt{unit} \to \Delta_2\{^{\emptyset}/_{a}\}) \to \Delta\{^{\emptyset}/_{a}\}$ ou $\sigma_v = (\mathtt{unit} \to \Delta_1\{^{\emptyset}/_{a}\}) \to \langle \mathtt{unit} \to \Delta_2\{^{\emptyset}/_{a}\}\rangle_{\Delta\{^{\emptyset}/_{a}\}}$ et

$$\Gamma\{^{\emptyset}/_{a}\} \vdash \mathbf{b} : \mathtt{dom}(w\{^{\emptyset}/_{a}\})$$

(nous ne pouvons avoir $\mathbf{b} = a$ puisque a n'est pas libre dans le terme). Le nom a et $\emptyset$ n'étant pas des variables de multi-ensembles, les autres conditions de la règle PASS sont satisfaites et nous obtenons :

$$\Gamma\{^{\emptyset}/_{a}\} \vdash \mathtt{pass}_{\mathbf{b}}\ V : \Delta\{^{\emptyset}/_{a}\} - (w\{^{\emptyset}/_{a}\}, \rho_1, \rho_2)$$

Nous concluons en remarquant que $(\Delta - (w, \rho_1, \rho_2))\{^{\emptyset}/_{a}\} = \Delta\{^{\emptyset}/_{a}\} - (w\{^{\emptyset}/_{a}\}, \rho_1, \rho_2)$.

APP Immédiat par induction dans le cas d'un nom de ressource.

Dans le cas d'un nom de domaine, nous avons par induction les dérivations :

$$\Gamma\{^{\emptyset}/_{a}\} \vdash P : \sigma\{^{\emptyset}/_{a}\} \to \tau\{^{\emptyset}/_{a}\}$$

et :

$$\Gamma\{^{\emptyset}/_{a}\} \vdash Q : \sigma'\{^{\emptyset}/_{a}\}$$

Comme nous avons $\sigma' \leq \sigma$, nous en déduisons $\sigma'\{^{\emptyset}/_{a}\} \leq \sigma\{^{\emptyset}/_{a}\}$, et nous concluons par la règle APP.

TEST Immédiat par induction pour le cas d'un nom de ressource. Dans le cas d'un nom de domaine, c'est également immédiat par induction en utilisant le fait que $\tau_1\{^{\emptyset}/_{a}\} \sqcup \tau_2\{^{\emptyset}/_{a}\} = (\tau_1 \sqcup \tau_2)\{^{\emptyset}/_{a}\}$.

DEF, AND, DEF.HOLE, DEF.$\top$ Immédiat par induction.

JOIN Immédiat par induction dans le cas d'un nom de ressource puisque celui-ci ne peut être un des noms du filtre par hypothèse du lemme.

En ce qui concerne les noms de domaines, nous avons par induction (les variables $\widetilde{x_i}$ sont toutes différentes de a) :

$$\Gamma\{^{\emptyset}/_{a}\} + \widetilde{x_1} : \widetilde{\sigma_1}\{^{\emptyset}/_{a}\} + \ldots + \widetilde{x_n} : \widetilde{\sigma_n}\{^{\emptyset}/_{a}\} \vdash P : \Delta\{^{\emptyset}/_{a}\}$$

Les noms de domaines n'étant pas généralisés, nous avons :

$$(r_i : s_i = \forall \widetilde{\beta_i}.\langle \widetilde{\sigma_i}\{^{\emptyset}/_{a}\}\rangle_{\Delta_i\{^{\emptyset}/_{a}\}} \in \Gamma\{^{\emptyset}/_{a}\})^{i \in [1..n]}$$

La condition $\Delta'\{^{\emptyset}/_{a}\} \subseteq \Delta_1\{^{\emptyset}/_{a}\}, \ldots, \Delta_n\{^{\emptyset}/_{a}\}$ est également satisfaite. Les conditions de généralisation sont immédiatement satisfaites, puisque les variables de types des types des ressources du filtre et de l'environnement de typage ne sont pas modifiées. Nous pouvons donc conclure par la règle JOIN.

TOP Immédiat par induction, puisque le prédicat *set* est satisfait par la substitution de a par $\emptyset$.

□

C.1.2 Stabilité et sûreté du typage

Lemme C.1.12 (Bonne formation et contextes d'évaluations) *Soit $\Gamma \vdash \mathbf{E}\{\cdot\} : \tau$ une dérivation de typage d'un contexte d'évaluation avec Γ bien formé. Si le trou de $\mathbf{E}\{\cdot\}$ est typé dans un environnement Γ', alors Γ' est bien formé.*

Preuve: Nous procédons par induction sur la taille du contexte, et par cas sur la forme du contexte. Nous déduisons de cette forme les règles de typage pouvant s'appliquer. Nous remarquons que cette induction procède dans le sens inverse d'une induction sur la dérivation de typage : nous supposons que l'environnement utilisé dans la conclusion de la règle de typage est bien formé, montrons que l'environnement utilisé dans l'hypothèse de la règle de typage du contexte plus petit est bien formé, et en déduisons par induction que le trou du contexte est typé dans un environnement bien formé.

En ce qui concerne l'hypothèse sur les noms libres de Γ, nous ne montrons que $\mathit{fn}(\Gamma) \subseteq \mathit{dom}(\Gamma)$ puisque nous avons immédiatement $\mathit{dom}(\Gamma) \subseteq \mathit{fn}(\Gamma)$.

$\{\cdot\}$ Immédiat, puisque $\Gamma = \Gamma'$.

$\mathbf{E}V$, $P\mathbf{E}$ Immédiat par induction sur le typage de $\mathbf{E}$, puisque la règle de typage utilisée (APP) ne modifie pas Γ.

$\nu n.\mathbf{E}$ Dans le cas d'une restriction d'un nom de domaine, le typage de $\mathbf{E}$ (par les règles NU.DOM ou TOP.NU.DOM) a lieu dans l'environnement $\Gamma + a : \mathtt{dom}(a)$. Cet environnement est bien formé puisque $\mathit{fn}(\Gamma + a : \mathtt{dom}(a)) = \mathit{fn}(\Gamma) \cup \{a\} \subseteq \mathit{dom}(\Gamma) \cup \{a\} = \mathit{dom}(\Gamma + a : \mathtt{dom}(a))$. Dans le cas d'un nom de ressource, le typage de $\mathbf{E}$ (par les règle NU.RES ou TOP.NU.RES) a lieu dans l'environnement $\Gamma + r : \forall \widetilde{\beta}.\langle\sigma\rangle_{\Delta}$. Cet environnement est bien formé puisque par hypothèse de la règle de typage nous avons $\mathit{fn}(\forall \widetilde{\beta}.\langle\sigma\rangle_{\Delta}) \subseteq \mathit{dom}(\Gamma)$.

$\mathbf{E} \mid P$, $P \mid \mathbf{E}$, $a(P)[\mathbf{E}]$, $a(\mathbf{E})[P]$, $P_1, \ldots, \mathbf{E}, \ldots, P_q$, $\epsilon[\mathbf{E}]$ Immédiat par induction.

□

Preuve du lemme 5.3.3: Nous procédons par induction sur la dérivation de $\mathcal{S} \equiv \mathcal{S}'$. La réflexivité et la transitivité sont immédiates. La symétrie est prouvée dans chaque cas en montrant les deux sens possibles.

STRUCT.NU.PAR Nous considérons l'équivalence structurelle : $(\nu n.P) \mid Q \equiv \nu n.(P \mid Q)$

Quatre cas différents sont possibles : n peut être un nom de ressource ou un nom de domaine, et la portée peut être étendue ou réduite. Dans tous les cas, nous avons $n \notin fn(Q)$.

Nous considérons tout d'abord l'extrusion de portée.

Si n est un nom de ressource, nous avons la dérivation de typage initiale :

$$\dfrac{\dfrac{\Gamma + r : s \vdash P : \Delta_1 \qquad r \notin dom(\Gamma)}{\Gamma \vdash \nu r : s.P : \Delta_1}\,[\text{NU.RES}] \qquad \Gamma \vdash Q : \Delta_2}{\Gamma \vdash (\nu r : s.P) \mid Q : \Delta_1, \Delta_2}\,[\text{PAR}]$$

avec des conditions supplémentaires sur s que nous ne détaillons pas. Comme nous avons $r \notin dom(\Gamma)$, nous pouvons appliquer le lemme C.1.10 à la dérivation pour Q et obtenir :

$$\Gamma + r : s \vdash Q : \Delta_2$$

Nous concluons par les règles PAR et NU.RES.

Dans le cas où n est un nom de domaine a, nous avons la dérivation suivante :

$$[\text{NU.DOM}]\ \dfrac{\dfrac{\Gamma + a : \mathtt{dom}(a) \vdash P : \Delta_1 \qquad a \notin fn(\Gamma) \qquad a \notin (\Delta_1 - a)}{\Gamma \vdash \nu a.P : \Delta_1 - a} \qquad \Gamma \vdash Q : \Delta_2}{\Gamma \vdash (\nu a.P) \mid Q : (\Delta_1 - a), \Delta_2}\,[\text{PAR}]$$

Comme a n'est pas libre dans Γ, nous avons par le lemme C.1.4 $a \notin \Delta_2$.

Comme $a \notin dom(\Gamma)$, nous utilisons le lemme C.1.10 avec la dérivation de Q et obtenons :

$$\Gamma + a : \mathtt{dom}(a) \vdash Q : \Delta_2$$

Puisque $a \notin \Delta_2$ et $a \notin \Delta_1 - a$, nous avons $a \notin (\Delta_1, \Delta_2) - a$. Nous pouvons donc conclure par les règles PAR et NU.DOM.

Nous considérons maintenant la réduction de portée. Dans le cas d'un nom de ressource, nous avons initialement le typage suivant :

$$\dfrac{\dfrac{\Gamma + r : s \vdash P : \Delta_1 \qquad \Gamma + r : s \vdash Q : \Delta_2}{\Gamma + r : s \vdash P \mid Q : \Delta_1, \Delta_2}\,[\text{PAR}] \qquad r \notin dom(\Gamma)}{\Gamma \vdash \nu r : s.P \mid Q : \Delta_1, \Delta_2}\,[\text{NU.RES}]$$

Comme dans le cas précédent, nous omettons les conditions supplémentaires sur s qui ne sont pas modifiées.

Comme nous avons $r \notin fn(Q)$, nous pouvons appliquer le lemme C.1.11 à la dérivation de typage pour Q pour obtenir :

$$\Gamma \vdash Q : \Delta_2$$

Nous concluons par les règles de typage NU.RES pour P et PAR.

Nous considérons maintenant le cas d'un nom de domaine a. Nous avons la dérivation de typage initiale :

$$[\text{PAR}]\ \dfrac{\dfrac{\Gamma + a : \mathtt{dom}(a) \vdash P : \Delta_1 \qquad \Gamma + a : \mathtt{dom}(a) \vdash Q : \Delta_2}{\Gamma + a : \mathtt{dom}(a) \vdash P \mid Q : \Delta_1, \Delta_2}}{\vdots \qquad a \notin fn(\Gamma) \qquad a \notin ((\Delta_1, \Delta_2) - a)}\,[\text{NU.DOM}]$$
$$\Gamma \vdash \nu a.P \mid Q : (\Delta_1, \Delta_2) - a$$

Comme nous avons $a \notin fn(Q)$, et $a \notin fn(\Gamma)$ (donc $a \notin dom(\Gamma)$), nous pouvons appliquer le lemme C.1.11 à la dérivation pour Q et obtenir :

$$\Gamma \vdash Q : \Delta_2\{^{\emptyset}/_{a}\}$$

puisque $\Gamma\{^{\emptyset}/_{a}\} = \Gamma$.

Comme nous avons par hypothèse $a \notin (\Delta_1, \Delta_2) - a$, a est présent au plus une fois dans Δ_1, Δ_2. Si a n'est pas présent dans Δ_2, nous avons alors $(\Delta_1, \Delta_2) - a = (\Delta_1 - a), \Delta_2 = (\Delta_1 - a), \Delta_2\{^{\emptyset}/_{a}\}$ et $a \notin \Delta_1 - a$. Sinon a est présent au plus une fois dans Δ_2 et n'est pas dans Δ_1, et nous avons $(\Delta_1, \Delta_2) - a = (\Delta_1 - a), (\Delta_2 - a) = (\Delta_1 - a), \Delta_2\{^{\emptyset}/_{a}\}$ (puisque cette substitution ne supprime que l'unique occurrence de a), avec toujours $a \notin \Delta_1 - a$.

Dans tous les cas, nous pouvons appliquer la règle Nu.Dom à P et conclure avec la règle Par pour obtenir le même type.

Struct.Nu.Top Nous considérons l'étape d'équivalence structurelle suivante : $\epsilon[\nu n.P] \equiv \nu n.\epsilon[P]$.

Une fois de plus, nous distinguons quatre cas : extension ou restriction de portée, nom de ressource ou nom de domaine. Nous ne présentons que le cas d'extrusion de portée, le cas de la diminution étant identique. Dans le cas d'un nom de ressource, nous avons le typage initial suivant :

$$\dfrac{\dfrac{\Gamma + r : s \vdash P : \Delta \qquad r \notin dom(\Gamma)}{\Gamma \vdash \nu r : s.P : \Delta}\,[\text{Nu.Res}] \qquad set(\Delta)}{\Gamma \vdash \epsilon[\nu r : s.P] : \Delta}\,[\text{Top}]$$

sans détailler les conditions supplémentaires sur s. Puisque nous avons $set(\Delta)$, nous pouvons directement appliquer la règle Top au processus P et obtenir le jugement $\Gamma + r : s \vdash \epsilon[P] : \Delta$. Les conditions pour appliquer la règle Top.Nu.Res sont satisfaites (ce sont les mêmes que celles de la règle Nu.Res), et nous pouvons conclure par le jugement $\Gamma \vdash \nu r : s.\epsilon[P] : \Delta$.

Dans le cas d'un nom de domaine, nous avons la dérivation de typage suivante :

$$\dfrac{\dfrac{\Gamma + a : \mathtt{dom}(a) \vdash P : \Delta \qquad a \notin (\Delta - a) \qquad a \notin fn(\Gamma)}{\Gamma \vdash \nu a.P : \Delta - a}\,[\text{Nu.Dom}] \qquad set(\Delta - a)}{\Gamma \vdash \epsilon[\nu a.P] : \Delta - a}\,[\text{Top}]$$

Comme nous avons $set(\Delta - a)$ et comme $a \notin \Delta - a$, nous avons donc $set(\Delta)$, et nous pouvons directement appliquer la règle Top avec P pour obtenir la dérivation : $\Gamma + a : \mathtt{dom}(a) \vdash \epsilon[P] : \Delta$. Nous avons toujours $a \notin fn(\Gamma)$ et nous pouvons conclure avec la règle Top.Nu.Dom et obtenir $\Gamma \vdash \nu a.(\epsilon[P]) : \Delta - a$.

Dans le cas de la diminution de portée, nous remarquons que $set(\Delta)$ implique nécessairement $a \notin \Delta - a$.

Struct.Nu.Boss Nous considérons l'étape d'équivalence structurelle $a(\nu n.P)[Q] \equiv \nu n.a(P)[Q]$, avec $n \neq a$ et $n \notin fn(Q)$.

Une fois de plus, nous distinguons les quatre cas.

Dans le cas d'extrusion de portée et d'un nom de ressource, nous avons la dérivation de typage initiale :

$$\dfrac{\dfrac{\Gamma + r : s \vdash P : \Delta_1 \qquad r \notin dom(\Gamma)}{\Gamma \vdash \nu r : s.P : \Delta_1}\,[\text{Nu.Res}] \qquad \Gamma \vdash a : \mathtt{dom}(a) \qquad \Gamma \vdash Q : \Delta_2}{\Gamma \vdash a(\nu r : s.P)[Q] : a, \Delta_1, \Delta_2}\,[\text{Dom}]$$

Le type de a, c'est à dire $dom(a)$, est déduit de la bonne formation de Γ.

Comme nous avons $r \notin dom(\Gamma)$, nous utilisons le lemme C.1.10 avec le typage de Q et de a et obtenons les dérivations :

$$\Gamma + r : s \vdash Q : \Delta_2$$

et :

$$\Gamma + r : s \vdash a : \mathtt{dom}(a)$$

Nous pouvons maintenant appliquer la règle Dom pour obtenir la dérivation

$$\Gamma + r : s \vdash a(P)[Q] : a, \Delta_1, \Delta_2$$

et nous concluons par la règle Nu.Res.

Nous ne détaillons pas le cas de la diminution de portée qui est identique, excepté dans l'utilisation du lemme C.1.11 pour le typage de Q et de a.

Nous considérons maintenant le cas d'un nom de domaine. Nous avons la dérivation de typage initiale suivante :

$$[\textsc{Nu.Dom}]\ \frac{\dfrac{\Gamma + b : \mathtt{dom}(b) \vdash P : \Delta_1 \qquad b \notin \mathit{fn}(\Gamma) \qquad b \notin \Delta_1 - b}{\Gamma \vdash \nu b.P : \Delta_1 - b}}{\vdots \qquad\qquad \dfrac{\Gamma \vdash a : \mathtt{dom}(a) \qquad \Gamma \vdash Q : \Delta_2}{\Gamma \vdash a(\nu b.P)[Q] : a, (\Delta_1 - b), \Delta_2}\ [\textsc{Dom}]}$$

Nous utilisons le lemme C.1.4 et le fait que $b \notin \mathit{fn}(\Gamma)$ avec la dérivation de typage $\Gamma \vdash Q : \Delta_2$ pour en déduire $b \notin \Delta_2$. Comme nous avons $b \notin \mathit{dom}(\Gamma)$, nous utilisons le lemme C.1.10 pour obtenir les dérivations :

$$\Gamma + b : \mathtt{dom}(b) \vdash a : \mathtt{dom}(a)$$

et :

$$\Gamma + b : \mathtt{dom}(b) \vdash Q : \Delta_2$$

Nous appliquons ensuite la règle Dom pour obtenir :

$$\Gamma + b : \mathtt{dom}(b) \vdash a(P)[Q] : a, \Delta_1, \Delta_2$$

Comme nous avons $b \neq a$ et $b \notin \Delta_2$, nous en déduisons $(a, \Delta_1, \Delta_2) - b = a, (\Delta_1 - b), \Delta_2$. Nous avons également $b \notin (a, \Delta_1, \Delta_2) - b$ et nous concluons par la règle Nu.Dom.

Nous considérons le cas de la diminution de portée. Comme $b \notin \mathit{fn}(Q)$ et $b \notin \mathit{dom}(\Gamma)$, nous déduisons de la dérivation $\Gamma + b : \mathtt{dom}(b) \vdash Q : \Delta_2$ la dérivation :

$$\Gamma \vdash Q : \Delta_2\{^{\emptyset}/_b\}$$

Nous obtenons également la dérivation :

$$\Gamma \vdash a : \mathtt{dom}(a)$$

(puisque nous avons $b \notin \mathit{fn}(\Gamma)$ et $b \neq a$).

Pour conclure nous devons montrer que $b \notin \Delta_1 - b$ et que $a, (\Delta_1 - b), \Delta_2\{^{\emptyset}/_b\} = (a, \Delta_1, \Delta_2) - b$, sachant que nous avons $b \notin (a, \Delta_1, \Delta_2) - b$.

Si b n'est pas dans Δ_2, et puisque $b \neq a$, nous avons nécessairement $b \notin \Delta_1 - b$ et $(a, \Delta_1, \Delta_2) - b = a, (\Delta_1 - b), \Delta_2 = a, (\Delta_1 - b), \Delta_2\{^{\emptyset}/_b\}$.

Si b est dans Δ_2, il l'est au plus une fois et ne peut être dans Δ_1. Nous avons donc $\Delta_2 - b = \Delta_2\{^{\emptyset}/_b\}$, et $\Delta_1 - b = \Delta_1$ donc $b \notin \Delta_1 - b$. Nous avons donc $(a, \Delta_1, \Delta_2) - b = a, \Delta_1, (\Delta_2 - b) = a, (\Delta_1 - b), \Delta_2\{^{\emptyset}/_b\}$.

Dans tous les cas nous concluons par la règle Dom.

Struct.Nu.Cont Ce cas est identique au cas précédent, en inversant les rôles de P et Q.

Struct.α Ce cas est immédiat puisque, par les lemmes C.1.6 et C.1.8 le typage est réalisé à α-équivalence près.

Struct.Context Ce cas est immédiat par induction, en utilisant le lemme C.1.12 qui nous indique que l'environnement de typage utilisé pour typer le processus bouchant le trou est bien formé, et en utilisant le lemme C.1.8 sans inclusion du type résultat pour reconstituer la dérivation de typage finale.

□

Lemme C.1.13 (Lemme de substitution) *Soient $\Gamma + x : \sigma \vdash P : \tau$ et $\Gamma \vdash V : \sigma'$ deux dérivations de typage avec $\sigma' \leq \sigma$ et Γ tel que tout nom de ressource dans $\mathit{dom}(\Gamma)$ possède un type de la forme $\forall\widetilde{\beta}.\langle\sigma_r\rangle_{\Delta_r}$ et tout nom de domaine un type de la forme $\mathtt{dom}(w)$. Nous avons alors une dérivation $\Gamma \vdash P\{^V/_x\} : \tau'$ avec $\tau' \leq \tau$.*

Preuve: Nous procédons par induction sur la dérivation pour P en étendant la propriété à tous les jugements de typage, excepté pour les configurations.

Nous faisons la remarque préliminaire que si σ' est un type de la forme $\mathtt{dom}(w)$, alors nécessairement V est un nom de domaine ou une variable, et si σ' est un type de la forme $\langle\sigma_r\rangle_{\Delta_r}$, alors V est un nom de ressource ou une variable. Nous prouvons ceci par cas sur V en utilisant à chaque fois les seules règles de typage s'appliquant : si c'est () alors son type ne peut-être que $\mathtt{unit}$ (Void), si c'est

un d alors son type ne peut être que `dest` (DEST.BOSS, DEST.CONT), si c'est une λ-abstraction alors son type ne peut-être que de la forme $\sigma \to \tau$ (FUN), si c'est un nuplet alors sont type est un nuplet (TUPLE), si c'est un nom de ressource adressé, alors son type est fonctionnel (ADDR). Il ne reste que le cas des variables, d'un nom de ressource ou d'un nom de domaine. Si c'est un nom de ressource, alors la seule règle de typage pouvant s'appliquer est NAME, et le type obtenu ne peut-être qu'un type de la forme $\langle \sigma_r \rangle_{\Delta_r}$ par hypothèse du lemme. Par le même raisonnement, si c'est un nom de domaine, le seul type possible est de la forme $\mathtt{dom}(w)$.

NIL, VOID Immédiat.

NAME Si x est u, le résultat est immédiatement l'hypothèse $\Gamma \vdash V : \sigma'$ puisque $\sigma' \leq \sigma$. Sinon le résultat est immédiat puisque enlever l'association $x : \sigma$ ne supprime pas l'association pour u, et nous avons toujours $\mathit{fn}(\mathit{ran}(\theta)) \subseteq \mathit{dom}(\Gamma)$, puisque les noms présents dans l'image de θ ne peuvent être des variables.

DEST.BOSS, DEST.CONT Immédiat par induction, puisque le seul sous-type de $\mathtt{dom}(w)$ est $\mathtt{dom}(w)$ et que si x est $\mathbf{a}$ alors V est un nom de domaine ou une variable par la remarque préliminaire, et le terme final est bien formé.

ADDR Immédiat par induction, puisqu'un sous-type de $\langle \sigma \rangle_\Delta$ est nécessairement de la forme $\langle \sigma' \rangle_{\Delta'}$ avec $\sigma \leq \sigma'$ et $\Delta' \leq \Delta$. Le type du résultat $\sigma' \to \Delta'$ est donc bien un sous-type de $\sigma \to \Delta$. Si le nom $\mathbf{r}$ est x, alors par la remarque préliminaire V est nécessairement un nom de ressource ou une variable, et le terme final est bien formé.

FUN Par hypothèse de la règle FUN, nous savons que la variable substituée est différente de la variable liée par l'abstraction y. Pour appliquer l'hypothèse d'induction, nous étendons le typage $\Gamma \vdash V : \sigma'$ avec l'association pour y par le lemme C.1.10. Nous remarquons également que l'environnement étendu ne possède pas de liaison supplémentaire pour des noms de ressources ou de domaines. Les seuls noms libres d'un type étant des noms de domaines, donc différents de x, la condition sur le type associé à y sont toujours satisfaites, et nous concluons par la règle FUN en remarquant que si $\tau' \subseteq \tau$ alors nous avons bien $\sigma_y \to \tau \subseteq \sigma_y \to \tau'$.

DOM Le résultat est immédiat par induction, en remarquant le seul sous-type de $\mathtt{dom}(w)$ est $\mathtt{dom}(w)$, et que par la remarque préliminaire, si $\mathbf{a}$ est x, alors V est un nom de domaine ou une variable et le terme final est bien formé.

PAR, TUPLE Immédiat par induction.

NU.RES Immédiat par induction puisque r ne peut être x, et puisque l'environnement Γ est étendu avec une association pour un nom de ressource ayant la forme désirée. Il est également nécessaire pour appliquer l'hypothèse d'induction d'utiliser le lemme C.1.10 pour étendre Γ dans le typage de $\Gamma \vdash V : \sigma'$. Nous remarquons enfin que nous supprimons une variable de nom de $\mathit{dom}(\Gamma)$, et que les variables de noms ne peuvent apparaître dans les types, donc la condition sur le type de la ressource est toujours satisfaite et nous pouvons conclure par la règle NU.RES.

NU.DOM Ce cas est identique au cas précédent, l'environnement Γ étant étendu avec une liaison ayant la forme correcte. La condition pour Δ est satisfaite avec $\Delta' \leq \Delta$.

PASS Nous avons par induction $\Gamma \vdash V'\{^V/_x\} : \sigma'_V$ avec $\sigma'_V \leq \sigma_V$. Nous utilisons la proposition C.1.7 pour en déduire que σ'_V a la forme correcte, et que les conditions sur ρ_1 et ρ_2 sont satisfaites. Si le nom $\mathbf{a}$ est x, alors par la remarque préliminaire V est soit un nom de domaine, soit une variable, et le terme final est bien formé. Nous concluons par la règle PASS.

APP Par induction nous avons $\Gamma \vdash P\{^V/_x\} : \sigma'_P$ avec $\sigma'_P \leq \sigma_P$. Par définition de $\leq$, nous avons $\sigma'_P = \langle \sigma''_P \rangle_{\Delta'}$ avec $\Delta' = \tau'$ ou $\sigma'_P = \sigma''_P \to \tau'$, avec dans les deux cas $\sigma^0_P \leq \sigma''_P$ et $\tau' \leq \tau^0_P$. Nous avons également par induction $\Gamma \vdash Q\{^V/_x\} : \sigma'_Q$ avec $\sigma'_Q \leq \sigma_Q$. Comme nous avons $\sigma_Q \leq \sigma^0_P$ par hypothèse de la règle APP initial, nous avons par transitivité de $\leq$ la propriété $\sigma'_Q \leq \sigma''_P$. Nous concluons avec la règle APP pour en déduire $\Gamma \vdash (PQ)\{^V/_x\} : \tau' \leq \tau^0_P$.

TEST Immédiat par induction, en utilisant la propriété C.1.1(3), et en rappelant que les substitutions n'ont pas lieu dans les filtres.

DEF Immédiat par induction.

AND Immédiat par induction.

DEF.$\top$ Immédiat.

JOIN Immédiat par induction, puisque x ne peut-être un des r_i, et puisque x est différent de tous les $\widetilde{x_i}$ par hypothèse de la règle de typage initiale. Nous utilisons une fois encore le lemme C.1.10 pour étendre la dérivation de typage $\Gamma \vdash V : \sigma'$ avec les associations pour les $\widetilde{x_i}$, en remarquant que cette extension n'ajoute aucune liaison pour un nom de ressource ou un nom de domaine.

□

Preuve du théorème 6: Nous procédons par induction sur la réduction ayant lieu.

RED.BETA Nous considérons la réduction : $(\lambda x.P)V \to P\{^V/_x\}$

La dérivation de typage pour le terme initial est :

$$\dfrac{\dfrac{\Gamma + x : \sigma \vdash P : \tau \qquad x \notin dom(\Gamma) \qquad fn(\sigma) \subseteq dom(\Gamma)}{\Gamma \vdash \lambda x.P : \sigma \to \tau}\ [\textsc{Fun}] \qquad \Gamma \vdash V : \sigma' \qquad \sigma' \leq \sigma}{\Gamma \vdash (\lambda x.P)V : \tau}\ [\textsc{App}]$$

L'environnement Γ étant bien formé, nous pouvons utiliser le lemme de substitution C.1.13 avec le typage de P pour obtenir $\Gamma \vdash P\{^V/_x\} : \tau'$ avec $\tau' \leq \tau$.

RED.IF.THEN Nous considérons la réduction : $([pat = V]P, Q) \to P$ avec $match(pat, V)$

La dérivation de typage pour le terme initial est :

$$\dfrac{\Gamma \vdash V : \tau \qquad \Gamma \vdash P : \tau_1 \qquad \Gamma \vdash Q : \tau_2}{\Gamma \vdash ([pat = V]P, Q) : \tau_1 \sqcup \tau_2}\ [\textsc{Test}]$$

Nous avons donc une dérivation de typage $\Gamma \vdash P : \tau_1$, avec $\tau_1 \leq \tau_1 \sqcup \tau_2$ par la proposition C.1.1(4).

RED.IF.ELSE Nous considérons la réduction : $([pat = V]P, Q) \to Q$ avec $\neg match(pat, V)$.

La dérivation de typage pour le terme initial est :

$$\dfrac{\Gamma \vdash V : \tau \qquad \Gamma \vdash P : \tau_1 \qquad \Gamma \vdash Q : \tau_2}{\Gamma \vdash ([pat = V]P, Q) : \tau_1 \sqcup \tau_2}\ [\textsc{Test}]$$

Nous avons donc une dérivation de typage $\Gamma \vdash Q : \tau_2$ avec $\tau_2 \leq \tau_1 \sqcup \tau_2$ par C.1.1(4).

RED.PASSIV Nous considérons la réduction :

$$a(\mathtt{pass}_a\ V \mid P)[Q] \to V(\lambda.P)(\lambda.Q)$$

La dérivation de typage pour le terme initial est :

$$[\textsc{Par}]\ \dfrac{\dfrac{\Gamma \vdash V : \sigma_V \qquad \Gamma \vdash a : \mathtt{dom}(a)}{\Gamma \vdash \mathtt{pass}_a\ V : \Delta_3}\ [\textsc{Pass}] \qquad \Gamma \vdash P : \Delta_P}{\Gamma \vdash (\mathtt{pass}_a\ V \mid P) : \Delta_3, \Delta_P} \qquad \vdots \qquad \dfrac{\vdots \qquad \Gamma \vdash a : \mathtt{dom}(a) \qquad \Gamma \vdash Q : \Delta_Q}{\Gamma \vdash a(\mathtt{pass}_a\ V \mid P)[Q] : a, \Delta_3, \Delta_P, \Delta_Q}\ [\textsc{Dom}]$$

avec $\sigma_V = (\mathtt{unit} \to \Delta_1) \to (\mathtt{unit} \to \Delta_2) \to \Delta$ ou $\sigma_v = (\mathtt{unit} \to \Delta_1) \to \langle \mathtt{unit} \to \Delta_2 \rangle_\Delta$, $\rho_1 \in \Delta_1$, $\rho_2 \in \Delta_2$, $\rho_1 \neq \rho_2$ et $\rho_1, \rho_2 \notin fsv(\Gamma) \cup (\Delta - \rho_1, \rho_2)$. Nous notons $\Delta_3 = \Delta - a, \rho_1, \rho_2$. Le type de a dérive du fait que Γ est bien formé.

Nous supposons que ρ_1 et ρ_2 ne sont pas présents dans Δ_P ni Δ_Q, quitte à les renommer dans la dérivation $\Gamma \vdash \mathtt{pass}_a\ V : \Delta_3$ (la conclusion de la dérivation est la même) par le lemme C.1.9.

Par conséquent, nous avons $\{^{\Delta_P;\Delta_Q}/_{\rho_1;\rho_2}\}\{^{\Delta_P;\Delta_Q}/_{\rho_1;\rho_2}\} = \{^{\Delta_P;\Delta_Q}/_{\rho_1;\rho_2}\}$.

De plus, par le lemme C.1.4 et la bonne formation de Γ, nous avons $fn(\Delta_P) \subseteq fn(\Gamma) = dom(\Gamma)$ et $fn(\Delta_Q) \subseteq fn(\Gamma) = dom(\Gamma)$.

Nous pouvons donc utiliser le lemme C.1.9 avec la substitution $\theta = \{^{\Delta_P;\Delta_Q}/_{\rho_1;\rho_2}\}$ (qui satisfait la propriété que son image est incluse dans le domaine de Γ) pour obtenir le typage :

$$\Gamma \vdash V : \sigma'_V$$

puisque ρ_1 et ρ_2 ne sont pas libres dans Γ, avec $\sigma'_V \leq \sigma_V\theta$.

Puisque nous avons $\rho_1 \in \Delta_1$ et $\rho_2 \in \Delta_2$, nous notons $\Delta_1 = \rho_1, \Delta'_1$ et $\Delta_2 = \rho_2, \Delta'_2$. Nous pouvons détailler la forme de $\sigma_V\theta$ comme étant soit

$$(\mathtt{unit} \to \Delta_P, (\Delta'_1\theta)) \to (\mathtt{unit} \to \Delta_Q, (\Delta'_2\theta)) \to \Delta\theta$$

soit

$$(\mathtt{unit} \to \Delta_P, (\Delta'_1\theta)) \to \langle \mathtt{unit} \to \Delta_Q, (\Delta'_2\theta)\rangle_{\Delta\theta}$$

Par la proposition C.1.7, le type σ'_V est de la forme $(\mathtt{unit} \to \Delta''_1) \to (\mathtt{unit} \to \Delta''_2) \to \Delta''$ ou $(\mathtt{unit} \to \Delta''_1) \to \langle \mathtt{unit} \to \Delta''_2\rangle_{\Delta''}$, avec $\Delta_P, (\Delta'_1\theta) \subseteq \Delta''_1$ et $\Delta_Q, (\Delta'_2\theta) \subseteq \Delta''_2$ et $\Delta'' \subseteq \Delta\theta$.
Nous avons donc $\mathtt{unit} \to \Delta_P$ qui est un sous-type de $\mathtt{unit} \to \Delta''_1$ et $\mathtt{unit} \to \Delta_Q$ qui est un sous-type de $\mathtt{unit} \to \Delta''_2$. Par deux application de la règle Fun, nous typons les valeurs $\lambda.P$ et $\lambda.Q$ (les conditions de la règle Fun sont immédiatement satisfaites) qui ont respectivement les types $\mathtt{unit} \to \Delta_P$ et $\mathtt{unit} \to \Delta_Q$. Nous utilisons deux fois la règle App pour obtenir la dérivation de typage :

$$\Gamma \vdash V(\lambda.P)(\lambda.Q) : \Delta''$$

Pour conclure nous remarquons que $\Delta_3 = \Delta - a, \rho_1, \rho_2$, donc nous avons $\Delta \subseteq \Delta_3, a, \rho_1, \rho_2$. Par conséquent nous avons $\Delta\theta \subseteq \Delta_3, a, \Delta_P, \Delta_Q$ puisque ρ_1 et ρ_2 ne sont pas présents dans Δ_3. Nous avons donc un type final Δ'' qui est bien un sous-type du type initial $\Delta_3, a, \Delta_P, \Delta_Q$.

Red.Res Nous avons la réduction suivante :

$$\langle D\rangle \mid r_1\widetilde{V_1} \mid \ldots \mid r_n\widetilde{V_n} \to \langle D\rangle \mid P\{\widetilde{V_i}/\widetilde{x_i}\}$$

avec $\langle D\rangle = \langle D_0; r_1\widetilde{x_1} \mid \ldots \mid r_n\widetilde{x_n} \triangleright P\rangle$.
Le typage du terme initial est de la forme ;

$$\dfrac{[\text{And}]\ \dfrac{(*)}{\Gamma \vdash D}\quad [\text{Def}]\ \dfrac{}{\Gamma \vdash \langle D\rangle : \emptyset} \qquad \Gamma \vdash r_1\widetilde{V_1} \mid \ldots \mid r_n\widetilde{V_n} : \Delta}{\Gamma \vdash \langle D\rangle \mid r_1\widetilde{V_1} \mid \ldots \mid r_n\widetilde{V_n} : \Delta}\ [\text{Par}]$$

Pour conclure, il nous suffit de remplacer la dérivation $\Gamma \vdash r_1\widetilde{V_1} \mid \ldots \mid r_n\widetilde{V_n} : \Delta$ par une dérivation $\Gamma \vdash P\{\widetilde{V_i}/\widetilde{x_i}\} : \Delta_0$ avec $\Delta_0 \subseteq \Delta$.
Pour ce faire, nous détaillons la dérivation du filtre :

$$\dfrac{\begin{array}{c}(r_i : s_i = \forall\widetilde{\beta_i}.\langle\widetilde{\sigma_i}\rangle_{\Delta_i} \in \Gamma)^{i\in[1..n]} \\ \Delta' \leq \Delta_1, \ldots, \Delta_n \qquad (\widetilde{x_i})^i \cap dom(\Gamma) = \emptyset \qquad \Gamma + \widetilde{x_1} : \widetilde{\sigma_1} + \ldots + \widetilde{x_n} : \widetilde{\sigma_n} \vdash P : \Delta' \\ \forall i \in [1..n].\widetilde{\beta_i} \cap fv(\Gamma) = \emptyset \qquad \forall i, j \in [1..n]^2.i \neq j \implies \widetilde{\beta_i} \cap \widetilde{\beta_j} = \emptyset\end{array}}{(*)\ \ \Gamma \vdash r_1\widetilde{x_1} \mid \ldots \mid r_n\widetilde{x_n} \triangleright P}\ [\text{Join}]$$

ainsi que la dérivation de typage pour chaque message :

$$[\text{Name}]\ \dfrac{r_i : \forall\widetilde{\beta_i}.\langle\widetilde{\sigma_i}\rangle_{\Delta_i} \in \Gamma \qquad \langle\widetilde{\sigma_i}\theta_i\rangle_{\Delta_i\theta_i} = Inst(\forall\widetilde{\beta_i}.\langle\widetilde{\sigma_i}\rangle_{\Delta_i})}{\Gamma \vdash r_i : \langle\widetilde{\sigma_i}\theta_i\rangle_{\Delta_i\theta_i}} \qquad \Gamma \vdash \widetilde{V_i} : \widetilde{\sigma'_i}\ [\text{App}]$$
$$\Gamma \vdash r_i\widetilde{V_i} : \Delta_i\theta_i$$

avec $fn(ran(\theta_i)) \subseteq dom(\Gamma)$ et $\widetilde{\sigma'_i} \leq \widetilde{\sigma_i}\theta_i$.
Nous avons $\Delta = \Delta_1\theta_1, \ldots, \Delta_n\theta_n$, en utilisant la règle Par plusieurs fois avec les messages.
Nous supposons dans la suite que les variables généralisées sont disjointes de l'image des θ_i (quitte à les renommer en des variables fraîches dans leurs associations dans Γ et dans le typage du processus gardé par le lemme C.1.9, en utilisant le cas où le type final est identique au type initial).
La deuxième condition de généralisation de la règle Join nous garantit que aucune variable généralisée n'est partagée par deux types, donc les domaines des θ_i sont disjoints. De plus, l'image des θ_i est disjointe de leurs domaines, puisque les variables généralisées ont été renommées pour que cela soit le cas. Nous appelons θ la substitution $\theta_1 \ldots \theta_n$ et nous avons $\theta\theta = \theta$.
De plus, nous avons par la condition de la règle Name $fn(ran(\theta)) \subseteq dom(\Gamma)$. Nous pouvons donc utiliser le lemme C.1.9 pour obtenir la dérivation :

$$\Gamma + \widetilde{x_1} : \widetilde{\sigma_1}\theta_1 + \ldots + \widetilde{x_n} : \widetilde{\sigma_n}\theta_n \vdash P : \Delta''$$

avec $\Delta'' \subseteq \Delta'\theta$. Nous avons

$$\Delta'' \subseteq \Delta'\theta \subseteq \Delta_1\theta, \ldots, \Delta_n\theta = \Delta$$

Nous utilisons plusieurs fois le lemme de substitution C.1.13 (ce qui est à chaque fois possible puisque Γ, qui est bien formé, est étendu avec des associations pour des variables) et obtenons :

$$\Gamma \vdash P\{\widetilde{V_i}/\widetilde{x_i}\} : \Delta_0$$

avec $\Delta_0 \subseteq \Delta''$. Nous remarquons que l'ordre des substitutions n'est pas important puisque les $\widetilde{x_i}$ ne sont pas présents dans Γ, ils ne peuvent donc être présents dans les $\widetilde{V_i}$.

Nous concluons par la règle PAR.

RED.CONTEXT Nous considérons la réduction : $\mathbf{E}\{P\} \to \mathbf{E}\{Q\}$ avec $P \to Q$.

Puisque nous avons une dérivation de typage de $\Gamma \vdash \mathbf{E}\{P\} : \Delta$, nous découpons cette dérivation en $\Gamma' \vdash P : \tau$ et $\Gamma \vdash \mathbf{E}\{\cdot : \tau\} : \Delta$, avec Γ' étant l'environnement utilisé pour typer le trou. Par le lemme C.1.12, Γ' est bien formé. Par induction, nous avons une dérivation de typage $\Gamma' \vdash Q : \tau'$ avec $\tau' \leq \tau$. Nous utilisons le lemme C.1.8 pour obtenir la dérivation $\Gamma \vdash \mathbf{E}\{Q\} : \Delta'$ avec $\Delta' \subseteq \Delta$.

RED.TOP.EQUIV Nous considérons la réduction : $\mathcal{S}_1 \to \mathcal{S}_2$ avec $\mathcal{S}_1 \equiv \mathcal{S}_1'$, $\mathcal{S}_1' \to \mathcal{S}_2'$, et $\mathcal{S}_2' \equiv \mathcal{S}_2$.

Nous avons la dérivation de typage initiale : $\Gamma \vdash \mathcal{S}_1 : \Delta_1$. Par le lemme 5.3.3, nous avons la dérivation de typage : $\Gamma \vdash \mathcal{S}_1' : \Delta_1$. Par induction, nous avons la dérivation de typage : $\Gamma \vdash \mathcal{S}_2' : \Delta_2$. avec $\Delta_2 \subseteq \Delta_1$. Enfin, par le lemme 5.3.3, nous avons la dérivation de typage : $\Gamma \vdash \mathcal{S}_2 : \Delta_2$.

RED.PROC.EQUIV Ce cas est identique au cas RED.TOP.EQUIV ci-dessus.

ROUTING Toutes les règles de routage satisfont immédiatement la propriété de stabilité du typage si nous vérifions les cas suivant : $\Gamma \vdash d.r\widetilde{V} : \Delta$ implique $\Gamma \vdash r\widetilde{V} : \Delta$, $\Gamma \vdash d.r\widetilde{V} : \Delta$ implique $\Gamma \vdash \mathbf{i}(d, r, \widetilde{V}) : \Delta$ et $\Gamma \vdash \mathbf{o}(d, r, \widetilde{V}) : \Delta$.

Dans le premier cas, nous avons la dérivation de typage :

$$\dfrac{\dfrac{\Gamma \vdash d : \mathtt{dest} \qquad \Gamma \vdash r : \langle\sigma\rangle_\Delta}{\Gamma \vdash d.r : \sigma \to \Delta}\,[\textsc{Addr}] \qquad \Gamma \vdash \widetilde{V} : \sigma' \qquad \sigma' \leq \sigma}{\Gamma \vdash d.r\widetilde{V} : \Delta}\,[\textsc{App}]$$

Nous construisons alors la dérivation :

$$\dfrac{\Gamma \vdash r : \langle\sigma\rangle_\Delta \qquad \Gamma \vdash \widetilde{V} : \sigma' \qquad \sigma' \leq \sigma}{\Gamma \vdash r\widetilde{V} : \Delta}\,[\textsc{App}]$$

En ce qui concerne les canaux spéciaux, nous ne détaillons que le cas **i**, le cas **o** étant identique. La dérivation initiale est la même que celle décrite ci-dessus. Nous construisons la dérivation :

$$\dfrac{\dfrac{\mathbf{i} : \forall\alpha\rho.\langle \mathtt{dest}, \langle\alpha\rangle_\rho, \alpha\rangle_\rho \qquad \mathit{fn}(\sigma) \cup \mathit{fn}(\Delta) \subseteq \mathit{dom}(\Gamma)}{\Gamma \vdash \mathbf{i} : \langle \mathtt{dest}, \langle\sigma\rangle_\Delta, \sigma\rangle_\Delta}\,[\textsc{Name}] \qquad \dfrac{\Gamma \vdash d : \mathtt{dest} \qquad \Gamma \vdash r : \langle\sigma\rangle_\Delta \qquad \Gamma \vdash \widetilde{V} : \sigma'}{\Gamma \vdash (d, r, \widetilde{V}) : (\mathtt{dest}, \langle\sigma\rangle_\Delta, \sigma')}\,[\textsc{Tuple}] \quad \vdots}{\Gamma \vdash \mathbf{i}(d, r, \widetilde{V}) : \Delta}\,[\textsc{App}]$$

Nous remarquons que la condition $\mathit{fn}(\sigma) \cup \mathit{fn}(\Delta) \subseteq \mathit{dom}(\Gamma)$ est satisfaite pour la raison suivante : nous appliquons le lemme C.1.4 à la dérivation $\Gamma \vdash d.r : \sigma \to \Delta$, qui nous indique que $\mathit{fn}(\sigma) \cup \mathit{fn}(\Delta) \subseteq \mathit{fn}(\Gamma)$, puis nous utilisons le fait que Γ est bien formé pour en déduire $\mathit{fn}(\Gamma) \subseteq \mathit{dom}(\Gamma)$.

□

Preuve du théorème 7: Nous procédons en trois temps. Nous montrons tout d'abord que pour un processus P en contexte d'évaluation, le multi-ensemble $\mathit{doms}(P)$ est le multi-ensemble des domaines libres et actifs de P. Puis nous montrons que si nous avons une dérivation de typage $\Gamma \vdash P : \Delta$ alors $\mathit{doms}(P) \subseteq \Delta$. Enfin nous montrons que si le type d'un processus en contexte d'évaluation est Δ alors nous avons $\mathit{set}(\Delta)$.

Afin de montrer le premier point, nous devons justifier que $\mathit{doms}(PQ) = \emptyset$ et $\mathit{doms}(P_1, \ldots, P_q) = \emptyset$, alors que ces deux constructions font parties des contextes d'évaluation. Pour ce faire, nous

montrons que si nous avons une dérivation de typage $\Gamma \vdash P : \tau$ avec τ n'étant pas un multi-ensemble, le processus P ne contient pas de processus ayant un multi-ensemble pour type en contexte d'évaluation, donc entre autres aucun domaine en contexte d'évaluation. Nous procédons par induction sur la dérivation de typage, et par cas sur la dernière règle de typage appliquée, en ne considérant que les règles de typage dont la conclusion fournit un type qui n'est pas un multi-ensemble. Les cas NAME, VOID, DEST.BOSS, DEST.CONT et ADDR sont immédiat par la forme du processus ou du nom typé. Le cas FUN est immédiat parce que le processus final ne contient rien en contexte d'évaluation (on n'évalue pas sous une λ-abstraction). Les cas TUPLE et APP sont immédiats par induction (que l'on peut appliquer puisque les processus dans l'hypothèse des règles ont des types qui ne sont pas des multi-ensembles). Les autres cas ne sont pas des contextes d'évaluation (TEST et définitions). Par conséquent, les constructions PQ et $P_1, \ldots, P_q$ ne peuvent contenir de domaine en contexte d'évaluation si elles sont bien typées. Les autres constructions de la fonction $\mathit{doms}(P)$ correspondent immédiatement à la définition du multi-ensemble de domaines libres et actifs (la notion *libre* est traduite par $\mathit{doms}(\nu a.P) = \mathit{doms}(P) \setminus \{a\}$, c'est à dire le multi-ensemble $\mathit{doms}(P)$ auquel on a enlevé toutes les occurrences de a).

Nous montrons maintenant que si nous avons une dérivation $\Gamma \vdash P : \Delta$ avec Γ bien formé, alors $\mathit{doms}(P) \subseteq \Delta$. Nous procédons par induction sur la dérivation de typage (la propriété est également vraie pour les configurations). Nous ne considérons bien entendu que les cas où le résultat peut être un multi-ensemble.

NIL Immédiat.

DOM Comme Γ est bien formé, nous avons $w = a = \mathbf{a}$. Le résultat est immédiat par induction.

PAR Immédiat par induction.

NU.RES, TOP.NU.RES Immédiat par induction, en remarquant que nous pouvons appliquer l'induction puisque l'environnement étendu est bien formé, et que $\mathit{doms}(P) \setminus \{r\} = \mathit{doms}(P)$.

NU.DOM, TOP.NU.DOM Immédiat par induction, en remarquant une fois encore que l'environnement étendu est bien formé, et que $\mathit{doms}(P) \subseteq \Delta$ implique $\mathit{doms}(P) \setminus \{a\} \subseteq \Delta - a$.

APP Immédiat (sans utiliser l'induction), puisque $\mathit{doms}(PQ) = \emptyset$.

PASS Immédiat puisque $\mathit{doms}(\mathtt{pass}_a\ V) = \emptyset$.

TEST Immédiat puisque $\mathit{doms}([pat = V]P, Q) = \emptyset$.

DEF Immédiat puisque $\mathit{doms}(\langle D \rangle) = \emptyset$.

TOP Immédiat par induction.

Nous montrons maintenant que si $\Gamma \vdash \mathbf{E}\{\cdot : \Delta'\} : \Delta$ avec $\mathit{set}(\Delta)$ alors nous avons $\mathit{set}(\Delta')$. Nous procédons, comme dans le lemme C.1.12 par induction sur la taille du contexte, en déduisant de la forme du contexte la règle de typage à appliquer. Nous rappelons que cette induction se fait à l'envers par rapport aux inductions sur les dérivations de typage : nous devons montrer que le type de l'hypothèse de la règle de typage est bien un ensemble pour appliquer l'hypothèse d'induction.

$\{\cdot\}$ Immédiat puisque $\Delta' = \Delta$.

$\mathbf{E}V$, $P\mathbf{E}$, $P_1, \ldots, \mathbf{E}, \ldots, P_q$ Ces cas ne peuvent s'appliquer puisque, comme nous l'avons montré ci-dessus, les processus n'ayant pas de type multi-ensemble ne peuvent contenir de processus ayant un type multi-ensemble en contexte d'évaluation.

$\mathbf{E} \mid P$, $P \mid \mathbf{E}$, $a(P)[\mathbf{E}]$, $a(\mathbf{E})[P]$, $\nu r : s.\mathbf{E}$, $\epsilon[\mathbf{E}]$ Le type du contexte dans l'hypothèse des règles de typage est nécessairement inclus dans le type final, c'est donc bien un ensemble et nous pouvons appliquer l'hypothèse d'induction et conclure.

$\nu a.\mathbf{E}$ Dans le cas où $\mathbf{E}$ est une configuration, le résultat est immédiat par le lemme C.1.3 qui nous permet d'appliquer l'hypothèse d'induction.

Sinon, nous avons par hypothèse $\mathit{set}(\Delta - a)$. Puisque a est présent au plus une fois dans Δ (la règle de typage utilisée est nécessairement NU.DOM, et nous avons $a \notin \Delta - a$), nous avons bien $\mathit{set}(\Delta)$.

Nous assemblons ces trois résultats pour conclure. Soit $\Gamma \vdash \mathcal{S} : \Delta$ une dérivation de typage avec Γ bien formé. Soit $\mathbf{E}$ un contexte d'évaluation et P un processus tel que $\mathcal{S} = \mathbf{E}\{P\}$. Par la première propriété, nous savons que le multi-ensemble des domaines libres et actifs de P est $\mathit{doms}(P)$. Nous savons également que P ne contient de domaines actifs que si la dérivation de typage pour P est de la forme $\Gamma' \vdash P : \Delta'$. Par le lemme C.1.12, nous savons que Γ' est bien formé, donc par le deuxième résultat nous avons $\mathit{doms}(P) \subseteq \Delta'$. Comme nous avons par le lemme C.1.3 $\mathit{set}(\Delta)$, par le troisième résultat nous avons $\mathit{set}(\Delta')$, donc par conséquent $\mathit{set}(\mathit{doms}(P))$: aucun domaine libre et actif de P ne partage son nom avec un autre domaine libre et actif de P. □

Bibliographie

[ABL99] Roberto M. Amadio, Gérard Boudol, and Cédric Lhoussaine. The receptive distributed pi-calculus (extended abstract). In *FST-TCS'99*, volume 1738 of *Lecture Notes in Computer Science*, pages 302–315. Springer-Verlag, 1999.

[Ama97] R. Amadio. An asynchronous model of locality, failure, and process mobility. Technical Report 3109, INRIA Sophia-Antipolis, 1997.

[BB92] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96 :217–248, 1992.

[BC01] M. Bugliesi and G Castagna. Secure safe ambients. In *Proceedings of POPL '01*, pages 222–235. ACM Press, 2001.

[BCC01] M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *4th International Symposium on Theoretical Aspects of Computer Software (TACS)*, Lecture Notes in Computer Science, 2001.

[Bou92] Gérard Boudol. Asynchrony and the π-calculus (note). Technical Report 1702, INRIA Sophia-Antipolis, May 1992.

[Bou98] G. Boudol. The π-calculus in direct style. *Higher-Order and Symbolic Computation, vol.11*, 11 :177–208, 1998.

[BSS01] G. Boudol, A. Schmitt, and J.B. Stefani. Marvel programming model v1. Deliverable D2.1, Marvel RNRT Project, INRIA, February 2001.

[Car97a] Luca Cardelli. Ambit, 1997. Available from `http://www.luca.demon.co.uk/Ambit/Ambit.html`.

[Car97b] Luca Cardelli. Mobile ambient synchronization. Technical note 1997-013, Digital Systems Research Center, July 1997.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *Foundations of Software Science and Computational Structures*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer-Verlag, 1998.

[CG99] L. Cardelli and A. D. Gordon. Types for mobile ambients. In *POPL'99*, pages 79–92. ACM, January 1999.

[CGG99] L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility types for mobile ambients. In *ICALP'99*, volume 1644 of *Lecture Notes in Computer Science*, pages 230–239. Springer-Verlag, 1999.

[CGG00a] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Ambient groups and mobility types. In *Proceedings of the International IFIP Conference TCS 2000 (Sendai, Japan)*, volume 1872 of *LNCS*, pages 333–347. IFIP, Springer, August 2000.

[CGG00b] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Secrecy and group creation. In Catuscia Palamidessi, editor, *CONCUR 2000 : Concurrency Theory (11th International Conference, University Park, PA, USA)*, volume 1877 of *LNCS*, pages 365–379. Springer, August 2000.

[CL99] Sylvain Conchon and Fabrice Le Fessant. Jocaml : Mobile agents for objective-caml. In *ASA/MA'99*, pages 22–29. IEEE Computer Society, October 1999.

[CP01] Sylvain Conchon and François Pottier. JOIN(X) : Constraint-Based Type Inference for the Join-Calculus. In David Sands, editor, *Proceedings of the 10th European Symposium on Programming (ESOP'01)*, volume 2028 of *Lecture Notes in Computer Science*, pages 221–236. Springer Verlag, April 2001.

[CS01] Tom Chothia and Ian Stark. A distributed calculus with local areas of communication. In *HLCL '00 : Proceedings of the 4th International Workshop on High-Level Concurrent Languages*, number 41.2 in Electronic Notes in Theoretical Computer Science. Elsevier, 2001.

[Dug99] D. Duggan. Dynamic typing for distributed programming in polymorphic languages. *ACM Transactions on Programming Languages and Systems, Vol. 21, No. 1*, 1999.

[FG96] Cédric Fournet and Georges Gonthier. The Reflexive CHAM and the Join-Calculus. In *Conference Record of POPL'96 : The* 23rd *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 372–385. ACM Press, 1996.

[FGL+96] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, pages 406–421, Pisa, Italy, August 26-29 1996. Springer-Verlag. LNCS 1119.

[FMLR00] Cédric Fournet, Luc Maranget, Cosimo Laneve, and Didier Rémy. Inheritance in the join calculus. In *Foundations of Software Technology and Theoretical Computer Science*, volume 1974 of *Lecture Notes in Computer Science*. Springer, December 2000.

[Fou98] Cédric Fournet. *The Join-Calculus : a Calculus for Distributed Mobile Programming*. PhD thesis, Ecole Polytechnique, Palaiseau, November 1998. INRIA, TU-0556.

[FS99] Cédric Fournet and Alan Schmitt. An implementation of Ambients in JoCAML. Software available from `http://join.inria.fr/ambients.html`, 1999.

[GC99] A. D. Gordon and L. Cardelli. Equational properties of mobile ambients. In *FoSSaCS'99*, volume 1578 of *Lecture Notes in Computer Science*, pages 212–226. Springer-Verlag, 1999.

[Gro99] Object Management Group. Corba components, 1999. OMG document orbos/99-02-01.

[HRes] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. *Information and Computation*, To appear. (47 pages). Preliminary version available as Sussex CSTR 1998 :02. Available from `http://www.cogs.susx.ac.uk/`.

[KSS00] Naoki Kobayashi, Shin Saito, and Eijiro Sumii. An implicitly-typed deadlock-free process calculus. In Catuscia Palamidessi, editor, *11th International Conference on Concurrency Theory (CONCUR 2000)*, volume 1877 of *Lecture Notes in Computer Science*, pages 489–503. Springer Verlag, August 2000. `http://www.yl.is.s.u-tokyo.ac.jp/members/koba/papers/deadlock-inference-concur.ps.gz`.

[Le 98] Fabrice Le Fessant. The JoCaml system prototype. Software and documentation available from `http://pauillac.inria.fr/jocaml`, 1998.

[Lev97] J.J. Levy. Some results on the join-calculus. In *3rd International Symposium on Theoretical spects of Computer Software (TACS), Lecture Notes in Computer Science no 1281*. Springer-Verlag, 1997.

[LS00] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proceedings 27th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2000)*, pages 352–364. ACM, January 2000.

[LSML00] Jeffrey Lewis, Mark Shields, Erik Meijer, and John Launchbury. Implicit parameters : Dynamic scoping with static types. In *Proceedings of the 27th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Boston, Massachusetts*, pages 108–118, Jan 2000.

[MH02] M. Merro and M. Hennessy. Bisimulation congruences in safe ambients. In *29th ACM Symposium on Principles of Programming Languages (POPL), Portland, Oregon, 16-18 January*, 2002.

[Mic98] Sun Microsystems. Enterprise java beans, March 1998. Specification v1.0.

[Mil80] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer-Verlag, 1980.

[Mil99] R. Milner. *Communicating and mobile systems : the π-calculus*. Cambridge University Press, 1999.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100(1) :1–77, September 1992.

[NFPV00] Rocco De Nicola, Gian Luigi Ferrari, Rosario Pugliese, and Betti Venneri. Types for access control. *Theoretical Computer Science*, 240(1) :215–254, 2000.

[NP96] Uwe Nestmann and Benjamin C. Pierce. Decoding choice encodings. In *CONCUR'96*, volume 1119 of *Lecture Notes in Computer Science*, pages 179–194. Springer-Verlag, August 1996. Revised full version as report ERCIM-10/97-R051, 1997.

[Oos94] Vincent van Oostrom. Confluence by decreasing diagrams. *Theoretical Computer Science*, 126 :259–280, 1994.

[PS92] Joachim Parrow and Peter Sjödin. Multiway synchronization verified with coupled simulation. In *CONCUR'92*, volume 630 of *Lecture Notes in Computer Science*, pages 518–533. Springer-Verlag, 1992.

[PS94] Joachim Parrow and Peter Sjödin. The complete axiomatization of cs-congruence. In *STACS'94*, volume 775 of *Lecture Notes in Computer Science*, pages 557–568. Springer-Verlag, 1994.

[SGN00] J.B. Stefani, F. Germain, and E. Najm. Elements of an object-based model for distributed and mobile computation. In *4th International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS 2000)*, 2000.

[SV99] Peter Sewell and Jan Vitek. Secure composition of insecure components. In *Proceedings of CSFW 99 : The 12th IEEE Computer Security Foundations Workshop (Mordano, Italy)*, pages 136–150. IEEE Computer Society, June 1999.

[SV01] D. Sangiorgi and A. Valente. A distributed abstract machine for Safe Ambients. In *Proc. ICALP'01*, volume 2076 of *Lecture Notes in Computer Science*. Springer-Verlag, 2001.

[US01] Asis Unyapoth and Peter Sewell. Nomadic Pict : Correct communication infrastructure for mobile computation. In *Proceedings of POPL 2001 : The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (London)*, pages 116–127, January 2001.

[VC98] J. Vitek and G. Castagna. Towards a calculus of secure mobile computations. In *Proceedings Workshop on Internet Programming Languages, Chicago, Illinois, USA*, 1998.

[YH99] N. Yoshida and M. Hennessy. Subtyping and locality in distributed higher-order processes. In *Proceedings CONCUR 99, Lecture Notes in Computer Science no 1664*. Springer, 1999.

[YH00] Nobuko Yoshida and Matthew Hennessy. Assigning types to processes (extended abstract). In *Fifteenth Annual IEEE Symposium on Logic in Computer Science*, pages 334–348. IEEE Computer Society, 2000.