Ecole Normale
Supérieure de Lyon

INRIA Sophia Antipolis
Action Mimosa

Etude de l'Expressivité des Ambients Mobiles Purs

Pascal ZIMMER, sous la direction de Davide SANGIORGI

Stage du DEA d'Informatique Fondamentale,
effectué du 7 Février au 7 Juillet 2000
à l'INRIA Sophia Antipolis, dans le groupe Mimosa

Résumé

Nous étudions le *Calcul des Ambients Purs*, qui correspond au *Calcul des Ambients* de Cardelli et Gordon restreint aux seules primitives de mouvement, et nous nous intéressons plus particulièrement à sa puissance expressive. Etant donné qu'il ne possède aucune forme de communication ni de substitution, nous montrons comment ces notions peuvent être simulées par la mobilité et des modifications de la structure hiérarchique des ambients. En guise d'exemple, nous donnons un encodage du π-calcul synchrone dans les ambients purs et nous démontrons un résultat de correspondance opérationnelle. Afin de simplifier la preuve et de faciliter la compréhension intuitive de l'encodage, nous introduisons un langage intermédiaire: le *π-Calcul avec Substitutions et Canaux Explicites*, qui est une extension syntaxique du π-calcul avec une sémantique opérationnelle spécifique.

Table des matières

Présentation du Stage

0.1 L'Action MIMOSA (Migration et Mobilité: Sémantique et Applications)

L'action MIMOSA est commune avec le Centre de Mathématiques Appliquées de l'ENSMP et le Centre de Mathématiques et d'Informatique de l'Université de Provence.

L'objectif général du projet est de concevoir et d'étudier des modèles de la programmation répartie et mobile, d'en tirer des primitives pour cette programmation, et d'élaborer des techniques de raisonnement et de vérification concernant spécifiquement les problèmes liés au code migrant.

On y développe deux approches: l'approche interactive, fondée sur l'idée de processus indépendants et communicants, et l'approche réactive, où les processus réagissent de concert à des événements diffusés. Les travaux visent particulièrement à intégrer les primitives de migration dans la programmation réactive, de façon à pouvoir programmer des "agents" participant à des "assemblées réactives" (jeux en réseau par exemple).

Voici les principaux axes de recherche:

Modèles et langages pour la mobilité, approche interactive Dans cet axe de recherche, l'objectif est d'étudier les primitives pour la mobilité fondées en particulier sur le modèle du π-calcul, et de ses extensions réparties.

Modèles et langages pour la mobilité, approche réactive L'objectif est de poursuivre le développement des outils de la programmation réactive, et d'y intégrer des primitives pour la migration.

Méthodes de raisonnement De telles méthodes pour les modèles de la migration, comme le calcul des Ambients, fondées sur la bisimulation et les logiques temporelles, ainsi que des méthodes de vérification pour des problèmes de sécurité sont développées.

Types Les systèmes de types sont utilisés, dans les modèles de la mobilité, à la fois pour éviter des erreurs, imposer des disciplines de sécurité, spécifier des comportements, mais aussi pour valider des transformations et des équivalences.

0.2 Le Sujet

De nombreuses questions auront été abordées durant ce stage et le sujet retenu pour ce rapport diffère de celui prévu à l'origine, même si la thématique reste identique. Ce sujet concerne l'étude de l'expressivité d'un sous-calcul particulier du calcul des *Ambients Mobiles*. Etant donné que ce sujet est largement développé dans les chapitres suivants, nous ne le détaillerons pas davantage ici.

Par ailleurs, d'autres travaux plus petits ont été réalisés ou commencés durant ce stage. De façon générale, on peut dire que la conséquence de ce stage a plus été l'intégration au groupe Mimosa et la préparation de la continuation en thèse que la réalisation d'un travail précis dans le cadre d'un sujet précis. Les autres thèmes abordés sont les suivants:

- L'étude d'un modèle d'*ambients réactifs* qui intègre dans un même formalisme les notions de programmations distribuée et réactive. Le principe de ce modèle est d'attribuer une horloge à chaque ambient et de découper l'exécution des processus qu'il contient par *instants*. Cette étude a conduit à une première implémentation "jouet" multi-thread des ambients en Java. Dans cette implémentation, chaque ambient est exécuté par un thread et un mécanisme de communication est mis en place pour régir les mouvements sans provoquer d'interférence (ce qui est le principal problème dans toutes les implémentations actuelles). Il s'agissait du sujet originel du stage.
- Parallèlement, une autre implémentation des ambients, distribuée cette fois-ci, fut développée par Andrea Valente de l'Université de Turin. Nous avons ainsi pu comparer nos deux implémentations au fur et à mesure de leur développement.
- Notre implémentation fut également utilisée pour tester et mettre au point l'encodage du π-calcul en ambients mobiles que nous allons développer dans les pages qui suivent.
- Quelques "programmes" de base furent développés et testés par notre implémentation, notamment le calcul des nombres de Fibonacci et le crible d'Eratosthène.
- D'autres questions d'expressivité des ambients furent abordées, notamment la comparaison de leurs différentes versions.

Chapitre 1

Introduction

Le *calcul des ambients* [CG97, CG98] fut conçu pour modéliser dans un même cadre à la fois l'*exécution sur supports mobiles*, comme un portable, et l'*exécution de code mobile*, c'est-à-dire du code se déplaçant entre différents supports comme les applets ou les agents. Il montre également comment les notions de domaines administratifs, de pare-feux, d'autorisations, etc... peuvent être formalisées dans un calcul (pour une étude plus détaillée des problèmes posés par la mobilité et l'exécution sur des réseaux à grande échelle, cf [Car99a, Car99b]). Informellement, un ambient est un lieu délimité à l'intérieur duquel un calcul a lieu. Les ambients peuvent être imbriqués de façon à former une hiérarchie. Chacun d'eux possède un nom (non nécessairement distinct des noms des autres ambients), qui est utilisé pour contrôler l'accès. Un ambient peut être déplacé comme un tout avec toutes les exécutions et les sous-ambients qu'il contient: il peut entrer dans un autre ambient ou en sortir. Il peut aussi être ouvert, de sorte que son contenu devienne visible au niveau courant, et des communications entre deux processus peuvent apparaître à l'intérieur d'un ambient (tout comme en π-calcul).

Le but de ce rapport est d'étudier la puissance expressive du sous-calcul obtenu en supprimant toutes les primitives de communication, le *calcul des ambients purs*. Ce sous-calcul n'a absolument aucune abstraction: il ne possède ni préfixe de lecture ou d'écriture, ni liaison de variable, ni règle de communication, et il ne peut effectuer aucune substitution de variables de façon globale sur un processus. Par conséquent, les seuls "outils" autorisés sont la structure hiérarchique des ambients, leurs mouvements et leurs ouvertures. On peut se demander quelles sont les motivations à l'origine de l'étude des ambients purs. Nous voulions comprendre ce qui rendait le calcul des ambients si expressif et quels constructeurs étaient réellement importants d'un point de vue purement théorique. Une question similaire s'était d'ailleurs déjà posée dans des travaux antérieurs dans le cadre du π-calcul [Pal97]. Après tout, le calcul des ambients purs est au calcul classique des ambients ce que CCS est au π-calcul: le premier n'a pas d'opérateur d'abstraction ni d'instantiation de variables, au contraire du deuxième.

En guise de première étape dans cette direction, nous avons réussi à encoder le π-calcul synchrone fini sans somme [Mil91] dans les ambients purs. Nous donnons un tel encodage à la fin de ce rapport. Le principal problème auquel nous avons dû faire face était la simulation de la substitution: la règle de communication du π-calcul lie une variable x à une valeur m et applique cette substitution au processus de continuation en une seule étape. Avec les ambients purs, nous avons dû adopter un autre mécanisme: toute référence future à x devra être remplacée dynamiquement par une référence à m. Pour réaliser cette tâche, nous créons un ambient x qui jouera le rôle d'un "forwarder". De plus, nous avons introduit des canaux explicites, sous la forme d'un ambient unique pour chaque nom de canal, afin que des primitives de lecture et d'écriture correspondantes puissent s'y rencontrer.

Concernant l'expressivité, il a été montré dans [CG98] que les ambients mobiles sans primitives de communication étaient suffisamment expressifs pour simuler les machines de Turing. Par conséquent, puisque ce calcul est Turing-complet, tout programme en π-calcul peut être simulé par un programme équivalent en ambients purs. Cependant, si les machines de Turing sont un bon modèle pour la programmation séquentielle, elles ne sont pas très bien adaptées à un cadre concurrentiel. Nous voulions un encodage "raisonnable", qui possède au moins la propriété de compositionnalité (i.e. tel que $\langle\!\langle op(P_1,\ldots,P_n)\rangle\!\rangle$ soit une fonction de $\langle\!\langle P_1\rangle\!\rangle,\ldots,\langle\!\langle P_n\rangle\!\rangle$ pour tout opérateur op), ce qui ne serait pas le cas si nous utilisions un encodage via les machines de Turing (CCS est aussi Turing-complet, pourtant le π-calcul est bien plus expressif).

Pour le calcul destination, nous avons préféré les *safe ambients*, qui furent présentés pour la première fois dans [LS00]. Ils diffèrent des ambients mobiles classiques par l'ajout de *coactions*. Dans le calcul des ambients, un mouvement est initié uniquement par l'ambient qui bouge, et l'ambient destination n'a aucun contrôle. Au contraire, en safe ambients, les deux protagonistes doivent être d'accord en utilisant des actions et des coactions correspondantes. Dans nos tentatives, il apparut que les protocoles étaient bien plus simples à implémenter en safe ambients. Par exemple, quand on conçoit un mécanisme de communication basé sur des requêtes auxquelles répondent des serveurs répliqués (tous deux étant des ambients), il est difficile d'empêcher un serveur de répondre deux fois à la même requête. En safe ambients, l'unicité de la réponse est plus facile à obtenir grâce à l'unicité de la coaction dans chaque requête. Obtenir le même effet avec les ambients classiques constitue un défi certain !

Afin de prouver une correspondance opérationnelle entre le π-calcul et notre encodage, nous avons dû concevoir un calcul intermédiaire pour simplifier la preuve, le π-*Calcul avec Substitutions et Canaux Explicites* (π_{esc}-calcul en abrégé). Il s'agit d'une extension du π-calcul, avec de nouvelles primitives pour les variables et les canaux explicites. Ce calcul apparaît comme un effet de bord intéressant et pas seulement un outil technique: il décompose les mécanismes de communication et de substitution du π-calcul en étapes plus simples, des propriétés d'équivalence avec le π-calcul peuvent être prouvées, et il permet une meilleure description intuitive du mécanisme à la base de l'encodage en ambients purs.

Travaux Voisins Quelques encodages du π-calcul en ambients ont déjà été proposés dans la littérature [CG98, LS00], mais jusqu'à présent tous encodaient les communications et substitutions du π-calcul par des communications et substitutions du calcul des ambients, tandis que notre encodage ne peut faire appel à ces mécanismes. En outre, tous encodaient seulement le π-calcul *asynchrone* (π_a) et ne pouvaient facilement être étendus pour encoder sa version synchrone. Enfin, sauf pour l'encodage de Levi et Sangiorgi [LS00], aucun résultat de correspondance opérationnelle ne fut jamais complètement prouvé pour aucun d'entre eux.

Pour certaines restrictions du π-calcul, la substitution peut être simulée d'une autre manière. Le π-*calcul local* ($L\pi$) [MS98] est un π-calcul asynchrone avec une contrainte additionnelle sur le constructeur de lecture $n(x).P$: x ne doit pas apparaître libre dans P dans une instruction de lecture. Dans ce calcul, l'équation suivante est une loi algébrique correcte:

$$P\{^b/_c\} = (\nu c)\ (P \mid c \lhd b)$$

où c ne doit pas apparaître libre dans P dans une instruction de lecture, $b \neq c$ et $c \lhd b \triangleq !c(x).\overline{b}\langle x\rangle$ est un lien qui renvoie tous les messages pour c vers b. Il faut noter que cette loi est fausse dans le π_a-calcul complet, et donc aussi dans le π-calcul.

Dans la même veine, un *equator* fut défini pour la première fois dans [HY95] par:

$$\mathcal{E}(b,c) \triangleq b \lhd c \mid c \lhd b$$

et il fut montré dans [Mer99] que

$$P\{^b/_c\} \cong_{\pi_a} (\nu c)\ (\mathcal{E}(b,c) \mid P)$$

(où $\cong_{\pi_a}$ symbolise la congruence à barbe dans le π_a-calcul). Cependant, cette égalité est fausse dans le π-calcul synchrone complet car l'utilisation de forwarders brise la séquentialité imposée par le préfixage en écriture. De plus, même si ces deux lois montrent une relation entre la substitution et d'autres opérateurs du π-calcul, elles ne sont pas un encodage des substitutions.

Certaines variantes du π-calcul avec des substitutions explicites furent également proposées. Dans le $\pi\xi$-calcul [FMQ96], les processus sont préfixés par un environnement global ξ qui contient les associations de noms créées par les communications passées. La règle principale est:

$$\frac{P \xrightarrow{\omega} P'}{\xi :: P \overset{\delta(\xi,\xi',\omega)}{\longmapsto} \xi' :: P'} \qquad \text{avec} \quad \xi' \in \eta(\xi,\omega)$$

où les fonctions δ et η sont définies en fonction de la sémantique désirée (late, early, open), de telle sorte que l'environnement ξ est étendu par les associations de noms activées par la transition $P \xrightarrow{\omega} P'$. La principale différence de cette approche avec notre π_{esc}-calcul est qu'il n'y a qu'un seul environnement global à l'extérieur du processus, au lieu de plusieurs variables directement incluses dans la syntaxe et tirant avantage de la restriction de noms. En outre, dans le $\pi\xi$-calcul, les substitutions ont lieu à l'extérieur du terme (dans $\delta(\xi,\xi',\omega)$) et ne sont pas incluses dans les réductions.

Une autre variante est le calcul des substitutions explicites $\pi\sigma$ de [Hir99], dans lequel un système de réécriture est utilisé pour appliquer des substitutions de noms à l'intérieur des termes. Etant donné que les processus y sont écrits avec la notation de De Bruijn, ce calcul a l'air très différent du π_{esc}-calcul. De plus, il applique les substitutions dans l'instruction d'écriture toute entière (la règle est $(\overline{a}b)[s] \to \overline{a[s]}b[s]$), de telle sorte que la clôture transitive des substitutions est automatiquement calculée, tandis qu'en π_{esc}-calcul, une chaîne arbitrairement longue de variables peut être créée. En outre, les sémantiques opérationnelles d'aussi bien $\pi\xi$ que $\pi\sigma$ sont définies via un système de transitions étiquetées tandis que notre calcul utilise des règles dans le style de la CHAM (la *Chemical Abstract Machine*), et aucun de ces calculs n'introduit de canaux explicites dans sa syntaxe.

Une dernière remarque est que tous les dialectes et variantes du π-calcul qui furent étudiées jusqu'ici ont toutes un constructeur d'abstraction (en général intégré au préfixe de lecture), et que donc l'exécution met en oeuvre une certaine forme de substitution. Pour nous, le défi consistait précisément dans le fait que nous n'avions pas un tel opérateur.

Plan Dans le Chapitre 2, nous rappelons les concepts nécessaires pour le π-calcul et les safe ambients. Nous introduisons également un type particulier de substitutions. Au Chapitre 3, nous présentons le π_{esc}-calcul de façon exhaustive, ainsi que quelques outils associés. Le Chapitre 4 définit des encodages entre le π-calcul et le π_{esc}-calcul, énonce les principales relations entre eux et donne un aperçu des preuves. La seconde partie de l'encodage, du π_{esc}-calcul dans les ambients purs, est donnée dans le Chapitre 5, avec un résultat de correspondance opérationnelle. Enfin, le Chapitre 6 rassemble les résultats en un théorème principal et donne l'encodage final pour le π-calcul. Les preuves des résultats énoncés dans ce rapport devraient bientôt être disponibles sous la forme d'un rapport technique [Zim00a]. De plus, une version abrégée et en anglais a été acceptée pour EXPRESS'00 (*7th International Workshop on Expressiveness in Concurrency*) et sera publiée prochainement [Zim00b].

Chapitre 2

Rappels et Définitions de Base

2.1 Le π-Calcul

Commençons par rappeler la syntaxe du π-calcul monadique synchrone que nous allons utiliser tout au long de ce rapport.

Nous aurons besoin de distinguer noms de canaux et noms de variables. Dans cette optique, soit $Name$ un ensemble infini dénombrable de noms de canaux (dénotés par les lettres n, m, p, ...), et Var un ensemble infini dénombrable de noms de variables (dénotés par les lettres x, y, ...).

La syntaxe du π-calcul est alors la suivante.

$$
\begin{array}{rcll}
P & ::= & (\nu n)\ P & \text{restriction} \\
 & | & \mathbf{0} & \text{processus nul} \\
 & | & P \mid Q & \text{composition parallèle} \\
 & | & !P & \text{réplication} \\
 & | & \overline{M}\langle M' \rangle.P & \text{écriture} \\
 & | & M(x).P & \text{lecture}
\end{array}
\qquad
\begin{array}{rcll}
M & ::= & n \in Name & \text{nom de canal} \\
 & | & x \in Var & \text{nom de variable}
\end{array}
$$

Dans $(\nu n)\ P$ (resp. $M(x).P$), le nom n (resp. x) est lié dans P. Il est toujours possible de changer ce nom par le mécanisme d'*α-conversion*, et nous considérerons que le processus résultant est égal au processus originel. Si un nom n'est pas lié, il est déclaré *libre*. L'ensemble des noms libres de canaux (resp. des noms libres de variables) de P est noté $fn(P)$ (resp. $fv(P)$).

La sémantique opérationnelle de notre π-calcul est donnée ci-dessous sous la forme d'une relation de réduction par pas élémentaire (*one-step reduction*), notée $\longrightarrow$. La règle principale est (π Red Comm) dans laquelle deux préfixes de lecture et d'écriture sur un même canal n sont consommés, tandis que la variable x est remplacée par la valeur m (la notation $Q\{^m/_x\}$ désigne le processus Q dans lequel on a remplacé chaque occurrence de x par m).

$$
\frac{}{\overline{n}\langle m \rangle.P \mid n(x).Q \;\longrightarrow\; P \mid Q\{^m/_x\}}\ (\pi \text{ Red Comm})
\qquad
\frac{P \;\longrightarrow\; P'}{P \mid Q \;\longrightarrow\; P' \mid Q}\ (\pi \text{ Red Par})
$$

$$
\frac{P \;\longrightarrow\; P'}{(\nu n)\ P \;\longrightarrow\; (\nu n)\ P'}\ (\pi \text{ Red Res})
\qquad
\frac{P \equiv P' \qquad P' \;\longrightarrow\; Q' \qquad Q' \equiv Q}{P \;\longrightarrow\; Q}\ (\pi \text{ Red Struct})
$$

La définition des règles de réduction fait appel à une relation de réécriture $\equiv$, appelée *congruence structurelle*. Sa définition est standard, avec des règles pour commuter des processus en parallèle, pour changer la portée d'un opérateur de restriction, pour "déplier" un processus répliqué, etc...

$$
\begin{array}{ll}
P \equiv P & (\pi \text{ Struct Refl}) \\
P \equiv Q \;\Rightarrow\; Q \equiv P & (\pi \text{ Struct Symm}) \\
P \equiv Q \equiv R \;\Rightarrow\; P \equiv R & (\pi \text{ Struct Trans}) \\
P \equiv Q \;\Rightarrow\; (\nu n)\ P \equiv (\nu n)\ Q & (\pi \text{ Struct Res}) \\
P \equiv Q \;\Rightarrow\; P \mid R \equiv Q \mid R & (\pi \text{ Struct Par}) \\
P \equiv Q \;\Rightarrow\; !P \equiv !Q & (\pi \text{ Struct Repl}) \\
P \equiv Q \;\Rightarrow\; \overline{M}\langle M' \rangle.P \equiv \overline{M}\langle M' \rangle.Q & (\pi \text{ Struct Output}) \\
P \equiv Q \;\Rightarrow\; M(x).P \equiv M(x).Q & (\pi \text{ Struct Input}) \\
P \mid \mathbf{0} \equiv P & (\pi \text{ Struct Par Zero}) \\
P \mid Q \equiv Q \mid P & (\pi \text{ Struct Par Comm}) \\
P \mid (Q \mid R) \equiv (P \mid Q) \mid R & (\pi \text{ Struct Par Assoc}) \\
(\nu n)\ (P \mid Q) \equiv P \mid (\nu n)\ Q \quad \text{si} \;\; n \notin fn(P) & (\pi \text{ Struct Res Par}) \\
(\nu n)\ (\nu m)\ P \equiv (\nu m)\ (\nu n)\ P & (\pi \text{ Struct Res Res}) \\
!P \equiv P \mid !P & (\pi \text{ Struct Repl Par}) \\
!\mathbf{0} \equiv \mathbf{0} & (\pi \text{ Struct Repl Zero})
\end{array}
$$

2.2 Ambients Purs

Nous présentons ici la variante du calcul des Safe Ambients que nous allons utiliser. Elle correspond aux Safe Ambients originaux de [LS00] desquels on a supprimé les primitives de communication. Cette restriction nous permet de simplifier légèrement la syntaxe (la version originale nécessitait un système de types pour rejeter certains termes mal formés). La syntaxe complète est la suivante.

$$
\begin{array}{rcll}
P & ::= & (\nu n)\ P & \text{restriction} \\
 & \mid & \mathbf{0} & \text{processus nul} \\
 & \mid & P \mid Q & \text{composition parallèle} \\
 & \mid & !P & \text{réplication} \\
 & \mid & n[P] & \text{ambient} \\
 & \mid & Cap.P & \text{capacité}
\end{array}
\qquad
\begin{array}{rcll}
Cap & ::= & in\ n & \text{entrée} \\
 & \mid & \overline{in}\ n & \text{co-entrée} \\
 & \mid & out\ n & \text{sortie} \\
 & \mid & \overline{out}\ n & \text{co-sortie} \\
 & \mid & open\ n & \text{ouverture} \\
 & \mid & \overline{open}\ n & \text{co-ouverture}
\end{array}
$$

Les opérateurs de base des calculs concurrentiels sont présents: la restriction de noms, le processus nul, la composition parallèle et la réplication. Ils se comportent comme en π-calcul. Un ambient est noté $n[P]$, où n est le nom de l'ambient et P est le processus exécuté à l'intérieur. Les actions sont appelées des *capacités* et sont notées $Cap.P$. Il existe trois types de capacités: une pour entrer dans un autre ambient ($in\ n$), une pour sortir d'un ambient ($out\ n$) et une pour ouvrir un ambient ($open\ n$). A chacune d'elles correspond une *co-capacité* (respectivement $\overline{in}\ n$, $\overline{out}\ n$ et $\overline{open}\ n$). Pour qu'un mouvement ait lieu, une capacité et une co-capacité correspondantes (c'est-à-dire avec le même nom) doivent être présentes à la bonne place, comme il est spécifié par les règles de réduction suivantes:

$$
\begin{array}{ll}
n[\ in\ m\ .\ P \mid Q] \mid m[\ \overline{in}\ m\ .R \mid S] \;\hookrightarrow\; m[n[P \mid Q] \mid R \mid S] & (\text{SA Red In}) \\
m[n[\ out\ m\ .P \mid Q] \mid\ \overline{out}\ m\ .R \mid S] \;\hookrightarrow\; n[P \mid Q] \mid m[R \mid S] & (\text{SA Red Out}) \\
open\ n\ .P \mid n[\ \overline{open}\ n\ .Q] \;\hookrightarrow\; P \mid Q & (\text{SA Red Open})
\end{array}
$$

Cette sémantique opérationnelle est complétée par quatre autres règles, de telle sorte qu'une réduction puisse avoir lieu sous un opérateur de restriction, dans une composition parallèle, à l'intérieur d'un ambient ou après une réécriture par congruence structurelle (qui est très similaire à la congruence structurelle du π-calcul).

$$\frac{P \hookrightarrow Q}{(\nu n)\ P \hookrightarrow (\nu n)\ Q}\ \text{(SA Red Res)} \qquad \frac{P \hookrightarrow Q}{P \mid R \hookrightarrow Q \mid R}\ \text{(SA Red Par)}$$

$$\frac{P \hookrightarrow Q}{n[P] \hookrightarrow n[Q]}\ \text{(SA Red Amb)} \qquad \frac{P \equiv P' \quad P' \hookrightarrow Q' \quad Q' \equiv Q}{P \hookrightarrow Q}\ \text{(SA Red Struct)}$$

La principale différence avec le calcul classique des ambients est le manque de primitives de communication, à savoir l'écriture asynchrone $\langle M \rangle$ et l'instruction de lecture $(x).P$. De plus, les co-capacités n'existaient pas dans la présentation originelle des ambients [CG98].

2.3 Substitutions

Dans cette Section, nous introduisons une forme particulière de substitutions, qui ont une structure en arbre. Nous garderons la même appellation, mais toute occurrence de "substitution" dans le reste de ce rapport se rapportera implicitement à la définition suivante.

Définition 1 *Une substitution est une application partielle $\sigma : Var \to Var \cup Name$ telle que:*

- $\forall x \in dom(\sigma), x\sigma \in Name \cup dom(\sigma)$ *(i.e. $im(\sigma) \subseteq Name \cup dom(\sigma)$)*
- $\forall x \in dom(\sigma)$, *il existe $k \in \mathbb{N}^*$ tel que $x\sigma^k \in Name$ (i.e. il n'y a pas de cycle)*

Définissons le graphe d'une substitution: l'ensemble de ses sommets est $dom(\sigma) \cup Name$ et ses arêtes sont $(x, x\sigma)$ pour $x \in dom(\sigma)$. Avec cette définition, on peut montrer facilement que le graphe d'une substitution est une forêt (un ensemble d'arbres), dont les racines sont dans $Name$ et tous les autres noeuds sont dans $dom(\sigma) \subseteq Var$. Par conséquent, on peut définir $\sigma^* : dom(\sigma) \to Name$ comme la fermeture transitive de σ, qui associe à chaque variable le nom situé à la racine de l'arbre auquel elle appartient.

Pour $x \notin dom(\sigma)$ et $M \in Name \cup dom(\sigma)$, on définit $\sigma' = \{^M/_x\} \uplus \sigma$ par $x\sigma' = M$ et $y\sigma' = y\sigma$ pour $y \neq x$. Il est facile de vérifier que σ' est encore une substitution correcte.

La substitution vide est notée $\varnothing$, et nous définissons également $fn(\sigma) \triangleq im(\sigma) \cap Name$. De plus, on étend naturellement le domaine des substitutions de telle sorte que nous puissions les appliquer à des processus.

Chapitre 3

Le Calcul Intermédiaire π_{esc}

Dans ce Chapitre, nous introduisons notre π-Calcul avec Substitutions et Canaux Explicites.

3.1 Syntaxe

Syntaxiquement, le π_{esc}-calcul est une extension du π-calcul, avec des constructeurs supplémentaires pour tenir compte des substitutions et des canaux.

Tout d'abord, la construction $(\nu x : M)\ P$ (avec $x \neq M$) représente une nouvelle variable x dont le contenu est M. Le nom x est lié dans P (tout comme n est lié dans $(\nu n)\ P$). Intuitivement, toute occurrence libre du nom x se réfère à cette variable et peut donc être remplacée par M sans affecter le comportement ultérieur du processus P.

La construction $[n : S]$ représente un canal explicite de nom n, dont le contenu est un ensemble S d'*abstractions* et de *concrétions* réalisées sur ce canal. Plus précisément, S n'est pas exactement un ensemble mais une composition parallèle d'abstractions et de concrétions (nous avons préféré cette approche pour conserver une symétrie avec la composition parallèle des processus). S peut être soit ε (le canal vide), une composition parallèle $S \mid S'$, une concrétion $\langle M\rangle.P$ pour une écriture ou une abstraction $(x).P$ pour une lecture (ces opérateurs correspondent respectivement aux processus $\overline{n}\langle M\rangle.P$ et $n(x).P$). Intuitivement, quand un processus réalise une écriture ou une lecture sur n, la requête correspondante est placée à l'intérieur du canal (s'il y en a un).

La syntaxe complète est la suivante:

P	$::=$	$(\nu n)\ P$	restriction	M	$::=$	$n \in Name$	nom de canal
	$\mid$	$\mathbf{0}$	processus nul		$\mid$	$x \in Var$	nom de variable
	$\mid$	$P \mid Q$	composition parallèle				
	$\mid$	$!P$	réplication	S	$::=$	ε	canal vide
	$\mid$	$\overline{M}\langle M'\rangle.P$	écriture		$\mid$	$S \mid S'$	comp. parallèle
	$\mid$	$M(x).P$	lecture		$\mid$	$\langle M\rangle.P$	concrétion
	$\mid$	$[n : S]$	canal explicite		$\mid$	$(x).P$	abstraction
	$\mid$	$(\nu x : M)\ P$	variable explicite avec $x \neq M$				

3.2 Règles de Réduction

Nous donnons maintenant la sémantique opérationnelle de notre calcul. Ses règles de réduction sont de la forme $\sigma : P \longmapsto P'$ pour deux processus P et P', et une substitution σ qui joue le rôle d'un environnement contenant les valeurs des variables libres de P. De plus, nous instaurons une condition annexe restreignant les définitions des règles aux seuls processus P et substitutions σ pour lesquels $fv(P) \subseteq dom(\sigma)$ (afin que nous puissions toujours trouver la valeur de n'importe quelle variable libre apparaissant dans P).

Les deux premières règles nous permettent de remplacer un préfixe de lecture ou d'écriture sur une variable x par le même préfixe sur la valeur M de x. S'il s'avère par la suite que M est une autre variable, il faudra à nouveau appliquer la même règle (ce qui est toujours possible, puisque dans ce cas, $M \in dom(\sigma)$ par définition d'une substitution). On continuera ainsi jusqu'à ce que M soit un nom de canal. Il faut noter aussi que nous n'effectuons aucune substitution sur M' dans la règle (π_{esc} Red Subst Out).

$$\frac{x\sigma = M}{\sigma : \overline{x}\langle M'\rangle.P \;\longmapsto\; \overline{M}\langle M'\rangle.P} \quad (\pi_{esc}\text{ Red Subst Out})$$

$$\frac{x\sigma = M}{\sigma : x(y).P \;\longmapsto\; M(y).P} \quad (\pi_{esc}\text{ Red Subst In})$$

Les deux règles suivantes ont déjà été présentées informellement plus haut: lorsqu'un canal n et un processus préfixé sur n se rencontrent dans une composition parallèle, la requête est placée à l'intérieur du canal (on peut alors omettre le nom n puisque toutes les abstractions et concrétions présentes dans $[n : S]$ se réfèrent nécessairement à n).

$$\frac{}{\sigma : [n : S] \mid \overline{n}\langle M\rangle.P \;\longmapsto\; [n : S \mid \langle M\rangle.P]} \quad (\pi_{esc}\text{ Red Output})$$

$$\frac{}{\sigma : [n : S] \mid n(x).P \;\longmapsto\; [n : S \mid (x).P]} \quad (\pi_{esc}\text{ Red Input})$$

Quand une concrétion $\langle M\rangle.P$ et une abstraction $(x).Q$ sont présentes dans un même canal, une communication peut effectivement avoir lieu. Les deux continuations P et Q sont alors placées à l'extérieur du canal, sauf qu'une nouvelle variable x de contenu M est créée en tête de Q. Ce schéma est formalisé par la règle suivante qui correspond à la règle (π Red Comm) en π-calcul (la condition $x \neq M$ peut toujours être satisfaite par α-conversion sur x).

$$\frac{x \neq M}{\sigma : [n : S \mid \langle M\rangle.P \mid (x).Q] \;\longmapsto\; [n : S] \mid P \mid (\nu x : M)\ Q} \quad (\pi_{esc}\text{ Red Comm})$$

La règle suivante autorise les réductions sous une restriction de variable $(\nu x : M)$. Le seul effet de bord est que la liaison $\{^M/_x\}$ doit être ajoutée à l'environnement σ (la condition $x \notin dom(\sigma)$ peut toujours être satisfaite par α-conversion sur x, et la condition $M \in Name \cup dom(\sigma)$ est automatiquement vérifiée car $fv((\nu x : M)\ P) \subseteq dom(\sigma)$).

$$\frac{x \notin dom(\sigma) \qquad \{^M/_x\} \uplus \sigma : P \;\longmapsto\; P'}{\sigma : (\nu x : M)\ P \;\longmapsto\; (\nu x : M)\ P'} \quad (\pi_{esc}\text{ Red Var})$$

Enfin, les trois dernières règles complètent le calcul: une réduction peut avoir lieu sous un opérateur de restriction d'un nom de canal, dans une composition parallèle ou à la suite d'une réécriture par congruence structurelle.

$$\frac{\sigma : P \;\longmapsto\; P'}{\sigma : (\nu n)\ P \;\longmapsto\; (\nu n)\ P'} \quad (\pi_{esc}\text{ Red Res}) \qquad \frac{\sigma : P \;\longmapsto\; P'}{\sigma : P \mid Q \;\longmapsto\; P' \mid Q} \quad (\pi_{esc}\text{ Red Par})$$

$$\frac{P \equiv P' \qquad \sigma : P' \;\longmapsto\; Q' \qquad Q' \equiv Q}{\sigma : P \;\longmapsto\; Q} \quad (\pi_{esc}\text{ Red Struct})$$

La relation de congruence $\equiv$ est la même qu'en π-calcul avec des règles additionnelles pour les nouvelles instructions et leurs interactions avec les anciennes (en particulier, la portée de $(\nu x : M)$ peut être étendue ou commutée avec (νn) à condition qu'il n'y ait aucune capture de nom indésirable). Voici la liste des règles à rajouter à celle du π-calcul:

$S \equiv S$	(π_{esc} Struct Refl)
$S \equiv S' \;\Rightarrow\; S' \equiv S$	(π_{esc} Struct Symm)
$S \equiv S' \equiv S'' \;\Rightarrow\; S \equiv S''$	(π_{esc} Struct Trans)
$S \equiv S' \;\Rightarrow\; [n : S] \equiv [n : S']$	(π_{esc} Struct Channel)
$P \equiv Q \;\Rightarrow\; (\nu x : M)\ P \equiv (\nu x : M)\ Q$	(π_{esc} Struct Var)
$S' \equiv S'' \;\Rightarrow\; S \mid S' \equiv S \mid S''$	(π_{esc} Struct Abs)
$P \equiv Q \;\Rightarrow\; \langle M \rangle.P \equiv \langle M \rangle.Q$	(π_{esc} Struct Out Abs)
$P \equiv Q \;\Rightarrow\; (x).P \equiv (x).Q$	(π_{esc} Struct In Abs)
$S \mid \varepsilon \equiv S$	(π_{esc} Struct Abs Zero)
$S \mid S' \equiv S' \mid S$	(π_{esc} Struct Abs Comm)
$S \mid (S' \mid S'') \equiv (S \mid S') \mid S''$	(π_{esc} Struct Abs Assoc)
$(\nu x : M)\ (P \mid Q) \equiv P \mid (\nu x : M)\ Q$ si $x \notin fv(P)$	(π_{esc} Struct Var Par)
$(\nu n)\ (\nu x : M)\ P \equiv (\nu x : M)\ (\nu n)\ P$ si $n \neq M$	(π_{esc} Struct Res Var)
$(\nu x : M)\ (\nu y : M')\ P \equiv (\nu y : M')\ (\nu x : M)\ P$ si $x \neq y, x \neq M'$ et $y \neq M$	(π_{esc} Struct Var Var)

3.3 Présentation d'un Canal et Processus Valides

Afin de suivre notre intuition, nous allons devoir restreindre l'ensemble des processus autorisés en π_{esc}-calcul. En fait, nous devons nous assurer que les canaux sont uniques et placés de façon adéquate. Considérons par exemple le processus $\overline{n}\langle m \rangle.[p : S]$. Le canal p serait inatteignable, et donc inutile, tant que l'écriture sur n n'aurait pas été réalisée. Considérons aussi le processus suivant:

$$[n : S] \mid [n : S'] \mid \overline{n}\langle m \rangle.P \mid n(x).Q$$

Puisqu'il y a deux canaux de même nom, les deux processus préfixés pourraient aller dans des canaux différents, conduisant à

$$[n : S \mid \langle m \rangle.P] \mid [n : S' \mid (x).Q]$$

et aucune communication ne pourrait avoir lieu entre P et Q!

Pour ces raisons, nous devons tout d'abord être capable de détecter un canal. Dans ce but, nous définissons un *prédicat de présentation* $P \Downarrow_1 n$, qui signifie intuitivement qu'un canal $[n : S]$ est présent dans P et n'est pas caché par une restriction de nom. La définition formelle de ce prédicat est très facile à écrire: le seul axiome est $[n : S] \Downarrow_1 n$ et toutes les autres règles n'opèrent que des appels récursifs (sauf $(\nu m)\ P \Downarrow_1 n$ qui vérifie que $m \neq n$). De plus, nous noterons $pr(P) \triangleq \{n \in Name / P \Downarrow_1 n\}$ l'ensemble des canaux présentés par un processus P.

De la même façon, on peut facilement définir un autre prédicat $P \Downarrow_2 n$, qui signifie que deux canaux différents de nom n sont présentés par P. Par exemple, on peut dériver $P \mid Q \Downarrow_2 n$ si on a à la fois $P \Downarrow_1 n$ et $Q \Downarrow_1 n$.

Les règles exactes sont (pour $i = 1{,}2$):

$$\frac{P \Downarrow_i n \qquad m \neq n}{(\nu m)\ P \Downarrow_i n}\ (\pi_{esc}\text{ Pres Res}) \qquad\qquad \frac{P \Downarrow_i n}{P \mid Q \Downarrow_i n}\ (\pi_{esc}\text{ Pres ParL})$$

$$\frac{Q \Downarrow_i n}{P \mid Q \Downarrow_i n}\ (\pi_{esc}\ \text{Pres ParR}) \qquad \frac{P \Downarrow_1 n \quad Q \Downarrow_1 n}{P \mid Q \Downarrow_2 n}\ (\pi_{esc}\ \text{Pres Par}_2)$$

$$\frac{P \Downarrow_i n}{!P \Downarrow_i n}\ (\pi_{esc}\ \text{Pres Repl}) \qquad \frac{P \Downarrow_1 n}{!P \Downarrow_2 n}\ (\pi_{esc}\ \text{Pres Repl}_2)$$

$$\frac{P \Downarrow_i n}{\overline{M}\langle M'\rangle.P \Downarrow_i n}\ (\pi_{esc}\ \text{Pres Output}) \qquad \frac{P \Downarrow_i n}{M(x).P \Downarrow_i n}\ (\pi_{esc}\ \text{Pres Input})$$

$$\frac{}{[n : S] \Downarrow_1 n}\ (\pi_{esc}\ \text{Pres Channel}_1) \qquad \frac{S \Downarrow_1 n}{[n : S] \Downarrow_2 n}\ (\pi_{esc}\ \text{Pres Channel}_2)$$

$$\frac{S \Downarrow_i m}{[n : S] \Downarrow_i m}\ (\pi_{esc}\ \text{Pres Channel}) \qquad \frac{P \Downarrow_i n}{(\nu x : M)\ P \Downarrow_i n}\ (\pi_{esc}\ \text{Pres Var})$$

$$\frac{S \Downarrow_1 n}{S \mid S' \Downarrow_1 n}\ (\pi_{esc}\ \text{Pres AbsL}) \qquad \frac{S' \Downarrow_1 n}{S \mid S' \Downarrow_1 n}\ (\pi_{esc}\ \text{Pres AbsR})$$

$$\frac{S \Downarrow_1 n \quad S' \Downarrow_1 n}{S \mid S' \Downarrow_2 n}\ (\pi_{esc}\ \text{Pres Abs}_2) \qquad \frac{P \Downarrow_i n}{\langle M\rangle.P \Downarrow_i n}\ (\pi_{esc}\ \text{Pres Out Abs})$$

$$\frac{P \Downarrow_i n}{(x).P \Downarrow_i n}\ (\pi_{esc}\ \text{Pres In Abs})$$

Lemme 2 $pr(P) \subseteq fn(P)$

Lemme 3 *Si* $P \Downarrow_1 n$ *et* $\vdash P : OK$*, alors* $P \equiv (\nu n_1)\ \ldots(\nu n_k)\ (\nu x_1 : M_1)\ \ldots(\nu x_{k'} : M_{k'})\ ([n : S] \mid P')$ *avec* $n \neq n_i$*.*

Il est maintenant temps d'introduire un petit système de types sur les processus. On définit le prédicat $\vdash P : OK$ par induction sur P en vérifiant que des canaux n'apparaissent pas après un préfixe ou une réplication, et qu'il n'y a au plus qu'un seul canal après une restriction de nom.

$$\frac{\vdash P : OK \quad P \not\Downarrow_2 n}{\vdash (\nu n)\ P : OK}\ (\pi_{esc}\ \text{OK Res})$$

$$\frac{}{\vdash \mathbf{0} : OK}\ (\pi_{esc}\ \text{OK Zero}) \qquad \frac{\vdash P : OK \quad \vdash Q : OK}{\vdash P \mid Q : OK}\ (\pi_{esc}\ \text{OK Par})$$

$$\frac{\vdash P : OK \quad \forall n \in Name\ \ P \not\Downarrow_1 n}{\vdash !P : OK}\ (\pi_{esc}\ \text{OK Repl})$$

$$\frac{\vdash P : OK \quad \forall n \in Name\ \ P \not\Downarrow_1 n}{\vdash \overline{M}\langle M'\rangle.P : OK}\ (\pi_{esc}\ \text{OK Output})$$

$$\frac{\vdash P : OK \quad \forall n \in Name\ \ P \not\Downarrow_1 n}{\vdash M(x).P : OK}\ (\pi_{esc}\ \text{OK Input})$$

$$\frac{\vdash S : OK}{\vdash [n : S] : OK}\ (\pi_{esc}\ \text{OK Channel}) \qquad \frac{\vdash P : OK}{\vdash (\nu x : M)\ P : OK}\ (\pi_{esc}\ \text{OK Var})$$

$$\frac{}{\vdash \varepsilon : OK}\ (\pi_{esc}\ \text{OK Eps}) \qquad \frac{\vdash S : OK \quad \vdash S' : OK}{\vdash S \mid S' : OK}\ (\pi_{esc}\ \text{OK Abs})$$

$$\frac{\vdash P : OK \quad \forall n \in Name\ \ P \not\Downarrow_1 n}{\vdash \langle M\rangle.P : OK}\ (\pi_{esc}\ \text{OK Out Abs})$$

$$\frac{\vdash P : OK \qquad \forall n \in Name \;\; P \not\Downarrow_1 n}{\vdash (x).P : OK} \;(\pi_{esc} \text{ OK In Abs})$$

Finalement, nous dirons qu'un processus P est *valide* et nous noterons $\vdash P : Valid$ si $\vdash P : OK$ et $P \not\Downarrow_2 n$ pour tous les noms $n \in Name$.

$$\frac{\vdash P : OK \qquad \forall n \in Name \;\; P \not\Downarrow_2 n}{\vdash P : Valid} \;(\pi_{esc} \text{ Valid})$$

Le lemme suivant montre que la validité est une propriété préservée par réduction.

Lemme 4 (Subject Reduction) *Si $\sigma : P \longmapsto Q$ et $\vdash P : Valid$, alors $\vdash Q : Valid$.*

3.4 Clôture des Canaux

Maintenant que nous avons éliminé les canaux en trop, nous allons devoir en rajouter quelques-uns ! Considérons le processus $\overline{n}\langle m\rangle.P \mid n(x).Q$. Il ne peut pas se réduire parce qu'aucun canal explicite n n'est présent. Si nous plaçons un canal vide $[n : \varepsilon]$ en parallèle, la communication pourra effectivement avoir lieu. Pour cette raison, nous définirons la *clôture des canaux* comme l'ajout des canaux explicites vides nécessaires. Puisque le même problème peut arriver aussi sous une restriction de nom (par exemple, $(\nu n)\ (\overline{n}\langle m\rangle.P \mid n(x).Q)$ ne peut se réduire), nous devrons aussi tenir compte de ce cas.

Définition 5 *Commençons par nous occuper des restrictions de noms. $cl(P)$ est un homomorphisme pour toutes les instructions, sauf pour:*

$$cl((\nu n)\ P) \triangleq \begin{cases} (\nu n)\ ([n : \varepsilon] \mid cl(P)) & si \quad P \not\Downarrow_1 n \\ (\nu n)\ cl(P) & si \quad P \Downarrow_1 n \end{cases}$$

Puis, la clôture des canaux d'un processus relativement à une substitution σ consiste à ajouter un canal vide pour chaque nom libre dans P ou σ pour lequel P ne présente pas de canal. Formellement,

$$cl_\sigma(P) \triangleq [n_1 : \varepsilon] \mid \ \ldots \ \mid [n_k : \varepsilon] \mid cl(P)$$

où $\{n_1, \ldots, n_k\} = (fn(P) \cup fn(\sigma)) \setminus pr(P)$ (cf. Lemme 2).

Nous dirons que P est clos relativement à σ si $cl_\sigma(P) = P$ (c'est-à-dire si P possède tous les canaux nécessaires pour garantir les communications). Vérifier que cette propriété est préservée par réduction dans le π_{esc}-calcul est une pure routine.

Chapitre 4

Relations entre le π et le π_{esc}-Calculs

4.1 Revenir au π-Calcul

Dans ce Chapitre, nous allons prouver quelques propriétés exprimant l'équivalence entre le π-calcul et le π_{esc}-calcul. Ces preuves reposent principalement sur notre capacité à retraduire un processus du π_{esc}-calcul en un processus du π-calcul. Cette traduction est notée $[\![P]\!]$ (paramétrée par un nom n pour les contenus de canaux) et elle est définie récursivement par les règles suivantes:

$$\begin{array}{ll}
[\![(\nu n)\ P]\!] \triangleq (\nu n)\ [\![P]\!] & [\![[n : S]]\!] \triangleq [\![S]\!]_n \\
[\![\mathbf{0}]\!] \triangleq \mathbf{0} & [\![(\nu x : M)\ P]\!] \triangleq [\![P]\!]\{^M/_x\} \\
[\![P \mid Q]\!] \triangleq [\![P]\!] \mid [\![Q]\!] & [\![\varepsilon]\!]_n \triangleq \mathbf{0} \\
[\![!P]\!] \triangleq ![\![P]\!] & [\![S \mid S']\!]_n \triangleq [\![S]\!]_n \mid [\![S']\!]_n \\
[\![\overline{M}\langle M'\rangle.P]\!] \triangleq \overline{M}\langle M'\rangle.[\![P]\!] & [\![\langle M\rangle.P]\!]_n \triangleq \overline{n}\langle M\rangle.[\![P]\!] \\
[\![M(x).P]\!] \triangleq M(x).[\![P]\!] & [\![(x).P]\!]_n \triangleq n(x).[\![P]\!]
\end{array}$$

En fait, $[\![P]\!]$ est un homomorphisme pour tous les constructeurs, sauf pour les canaux et les restrictions de variables. Dans le premier cas, il suffit juste d'ajouter le nom du canal en préfixe des abstractions et des concrétions. Le deuxième cas s'avère plus intéressant: on effectue la substitution imposée par la restriction de variable, c'est-à-dire $[\![(\nu x : M)\ P]\!]$ devient $[\![P]\!]$ dans lequel on remplace chaque occurrence libre de x par M.

4.2 Résultats

Quand allons-nous affirmer qu'un π-processus et un π_{esc}-processus sont "équivalents" ? Pour suivre notre intuition, le π-processus équivalent à un π_{esc}-processus P qui évolue dans un environnement σ devrait être $[\![P]\!]\sigma^*$. Ici, il faut prendre en compte les liaisons de σ car les variables libres de P issues des communications précédentes doivent être remplacées par leurs valeurs. Nous appliquons la clôture transitive σ^* en une seule étape, de telle sorte que toutes les variables libres soient converties en des noms de canaux (en fait, on peut montrer que $[\![P]\!]\sigma^*$ est égal à $[\![(\nu x_1 : M_1)\ \ldots (\nu x_k : M_k)\ P]\!]$ lorsque $\sigma = \{^{M_k}/_{x_k}\} \uplus \ldots \uplus \{^{M_1}/_{x_1}\}$).

Le lemme technique suivant se prouve assez facilement. Il montre que tout pas de réduction en π_{esc}-calcul correspond à zéro ou un pas de réduction en π-calcul.

Lemme 6 *Si $\sigma : P \longmapsto Q$, alors $[\![P]\!]\sigma^*\ \mathcal{R}\ [\![Q]\!]\sigma^*$ où $\mathcal{R}$ est soit $\equiv$ soit $\longrightarrow$.*

Le lemme inverse est plus complexe. Des hypothèses additionnelles restreignent le résultat uniquement aux processus valides et aux environnements appropriés. Sous ces conditions, ce lemme affirme que tout pas de réduction dans le π-calcul peut être simulé par un ou plusieurs pas dans le π_{esc}-calcul. De plus, cette simulation n'est pas définie directement sur P, mais sur sa clôture des canaux $cl_\sigma(P)$ (par exemple, les π-processus de la Section 3.4 se réduisent en π-calcul, mais seulement leurs clôtures des canaux se réduisent en π_{esc}-calcul).

Lemme 7 *Si $[\![P]\!]\sigma^* \longrightarrow Q$, $\vdash P : Valid$ et $fv(P) \subseteq dom(\sigma)$, alors il existe un processus P' tel que $\sigma : cl_\sigma(P) \longmapsto^+ P'$ et $[\![P']\!]\sigma^* \equiv Q$.*

Ce lemme est beaucoup plus difficile à prouver. Nous essayons d'expliquer pourquoi et de donner un schéma de preuve.

- La clôture des canaux interagit mal avec une preuve par induction. Ceci provient du fait que la clôture des canaux n'est pas définie par induction sur les termes. Par conséquent, pour presque chaque constructeur, il faut un lemme préliminaire qui analyse ce cas particulier et relie la clôture des canaux d'un processus avec les clôtures des canaux de ses sous-composantes. Dans certains cas, il y a plus d'une réponse, suivant le contexte.
- Les canaux vides interagissent difficilement avec la congruence structurelle. Par exemple, si le premier pas de réduction est

$$[\![[n : \varepsilon] \mid P]\!]\sigma^* \;=\; \mathbf{0} \mid [\![P]\!]\sigma^* \;\equiv\; [\![P]\!]\sigma^* \;\longrightarrow\; Q$$

  on ne peut conclure directement par induction car le processus résultant P ne présente plus de canal n (la congruence structurelle l'a en quelque sorte "effacé"), et du coup les clôtures des canaux de $[n : \varepsilon] \mid P$ et P sont différentes. Cet exemple est simple, mais dans le cas général l'effacement de canaux vides peut se produire n'importe où dans un terme. Nous avons donc besoin d'un résultat qui relie la clôture des canaux de P avec P' lorsque $[\![P]\!]\sigma^* \equiv P'$ est le premier pas de réduction.
- Les canaux interagissent mal avec la composition parallèle. Ce problème est celui qui nécessite le plus long développement technique. Supposons que $[\![P \mid P']\!]\sigma^* \longrightarrow Q \mid [\![P']\!]\sigma^*$ a été dérivé de $[\![P]\!]\sigma^* \longrightarrow Q$ par (π Red Par). Supposons aussi que cette réduction implique une communication sur un canal n, et que $P \not\Downarrow_1 n$ et $P' \Downarrow_1 n$ (i.e. le canal explicite n est dans P'). Alors, par induction, nous obtenons une simulation sur $cl_\sigma(P) = [n : \varepsilon] \mid P_1$ puisque $P \not\Downarrow_1 n$. Mais les réductions correspondantes de $cl_\sigma(P \mid P')$ qui impliquent le canal n doivent maintenant utiliser le canal explicite dans P' et non le canal vide $[n : \varepsilon]$ qui a été ajouté dans la clôture des canaux ! Dans le cas général, il faut un résultat qui montre que des réductions impliquant des canaux vides issus de la clôture peuvent être remplacées par des réductions pour lesquelles les communications sont reportées sur des canaux (éventuellement non vides) possédés par un processus en parallèle.

Les deux lemmes précédents sont techniques, mais en pratique et dans le reste de ce rapport, nous nous restreindrons aux processus valides, sans variables libres et clos relativement à $\varnothing$. Dans ce cas, la correspondance opérationnelle devient beaucoup plus simple:

Corollaire 8

- *Si $\varnothing : P \longmapsto Q$, alors $[\![P]\!] \mathcal{R} [\![Q]\!]$.*
- *Si $[\![P]\!] \longrightarrow Q$, si P est clos relativement à $\varnothing$, si $\vdash P : Valid$ et si $fv(P) = \emptyset$, alors il existe un processus P' tel que $\varnothing : P \longmapsto^+ P'$ et $[\![P']\!] \equiv Q$.*

4.3 Equivalence Observationnelle

Afin de compléter nos résultats, nous avons pu prouver une propriété supplémentaire d'*équivalence observationnelle*. Le prédicat d'observabilité $P \downarrow M$ est défini sur les π-processus de façon standard (par exemple, $n(x).P \downarrow n$), et s'étend facilement aux π_{esc}-processus (pour les variables, une substitution doit être effectuée, i.e. $(\nu x : M)\ P \downarrow M$ lorsque $P \downarrow x$).

Pour le π-calcul:

$$\frac{P \downarrow M \qquad n \neq M}{(\nu n)\ P \downarrow M}\ \text{(Obs Res)} \qquad\qquad \frac{P \downarrow M}{P \mid Q \downarrow M}\ \text{(Obs ParL)}$$

$$\frac{Q \downarrow M}{P \mid Q \downarrow M}\ \text{(Obs ParR)} \qquad\qquad \frac{P \downarrow M}{!P \downarrow M}\ \text{(Obs Repl)}$$

$$\frac{}{\overline{\overline{M}}\langle M'\rangle.P \downarrow M}\ \text{(Obs Output)} \qquad\qquad \frac{}{M(x).P \downarrow M}\ \text{(Obs Input)}$$

Pour le π_{esc}-calcul, il faut ajouter:

$$\frac{S \not\equiv \varepsilon}{[n : S] \downarrow n}\ \text{(Obs Channel)}$$

$$\frac{P \downarrow M \qquad x \neq M}{(\nu x : M')\ P \downarrow M}\ \text{(Obs Var}_1\text{)} \qquad\qquad \frac{P \downarrow x}{(\nu x : M)\ P \downarrow M}\ \text{(Obs Var}_2\text{)}$$

On peut alors montrer la relation suivante:

Proposition 9 *Pour tout processus P du π_{esc}-calcul, $P \downarrow M \ \Leftrightarrow\ [\![P]\!] \downarrow M$.*

4.4 Du π-Calcul au π_{esc}-Calcul

Il existe une manière simple de transformer un π-processus en un π_{esc}-processus "correct": il suffit de remplacer chaque construction $(\nu n)\ P$ par $(\nu n)\ ([n : \varepsilon] \mid P)$ et d'ajouter un canal vide pour chaque nom libre dans P. En fait, c'est précisément la définition de la clôture des canaux $cl_\varnothing(P)$ (si on voit le π-processus P comme un π_{esc}-processus). Cette transformation possède les propriétés remarquables suivantes: $cl_\varnothing(P)$ est valide, clos relativement à $\varnothing$ et n'a pas de variable libre si P n'en a pas non plus (ces propriétés nous permettent d'utiliser le Corollaire 8).

4.5 A Propos du Choix du π_{esc}-Calcul

Les canaux explicites et les variables sont de structure similaire, mais nous avons utilisé des syntaxes différentes: deux constructeurs (νn) et $[n : S]$ pour les canaux, et un constructeur unique $(\nu x : M)$ pour les variables. Pourquoi avons-nous retenu cette combinaison? Il est maintenant temps de répondre à cette question.

Nous aurions pu choisir de séparer les variables en une restriction (νx) et une variable explicite $[x : M]$, la règle (π_{esc} Red Subst Out) devenant alors $\sigma : [x : M] \mid \overline{x}\langle M'\rangle.P \ \longmapsto\ [x : M] \mid \overline{M}\langle M'\rangle.P$ (et de même pour (π_{esc} Red Subst In)). Mais, afin de pouvoir évaluer $[\![(\nu x)\ P]\!]$, il aurait fallu un moyen d'atteindre l'objet $[x : M]$ dans P et d'obtenir la valeur M. Ceci nous aurait conduit à un développement technique très long.

A l'inverse, nous aurions pu choisir d'inclure le contenu du canal dans l'opérateur de restriction avec $(\nu n : S)$. Dans ce cas, il peut se produire une interférence entre les restrictions. Par exemple, le processus $(\nu n : \varepsilon)\ (\nu x : n)\ \overline{n}\langle x\rangle.P$ devrait se réduire en plaçant la concrétion $\langle x\rangle.P$ à l'intérieur de n, mais ni $(\nu n : \langle x\rangle.P)\ (\nu x : n)\ \mathbf{0}$ ni $(\nu x : n)\ (\nu n : \langle x\rangle.P)\ \mathbf{0}$ ne serait correct: dans chaque cas, un nom lié devient libre. . . .

Chapitre 5

Encoder le π_{esc}-Calcul en Ambients Purs

5.1 L'Encodage

Le mécanisme principal à la base de l'encodage est une forme de communication basée sur le modèle requête/serveur. En ambients purs, une requête qui cherche à communiquer avec n sera modélisée par un ambient de nom rw contenant le processus $request\ rw\ n$ (dans notre encodage, rw sera seulement $read$ ou $write$). Son premier mouvement sera d'entrer à l'intérieur de n. De façon symétrique, un serveur sera modélisé par un ambient répliqué $enter$, placé dans la destination n et qui essaiera d'entrer dans la requête et d'en prendre le contrôle. Ce mécanisme est similaire à l'encodage des *objective moves* décrit dans [CG98]. Commençons par définir quelques abréviations utiles:

$$
\begin{array}{l}
server\ n\ .P \triangleq\ !\ enter[\ in\ n\ .\ \overline{open}\ enter\ .P] \\
request\ rw\ n\ \triangleq\ in\ n\ .\ \overline{in}\ rw\ .\ open\ enter \\
request\ rw\ x\ \triangleq\ in\ x\ .\ \overline{in}\ rw\ .\ open\ enter\ .\ out\ x \\
fwd\ M\ \triangleq\ server\ write\ .\ request\ write\ M\ \mid\ server\ read\ .\ request\ read\ M \\
n\ be\ m\ .P \triangleq m[\ out\ n\ .\ \overline{in}\ m\ .(\ open\ n\ \mid P)] \mid\ \overline{out}\ n\ .\ in\ m\ .\ \overline{open}\ n \\
allowIO\ n\ \triangleq\ !\ \overline{in}\ n\ \mid\ !\ \overline{out}\ n
\end{array}
$$

En guise d'exemple, voici la forme générale de la réduction entre une requête et un ambient n contenant un serveur:

$$
\begin{array}{rl}
& n[\ server\ rw\ .P \mid\ allowIO\ n\ ] \mid rw[\ request\ rw\ n\ \mid Q] \\
\hookrightarrow^{+} & n[\ server\ rw\ .P \mid\ allowIO\ n\ \mid rw[P \mid Q]]
\end{array}
$$

Une variable x dont la valeur est M sera simplement modélisée par un ambient de nom x contenant deux serveurs qui remplaceront chaque requête sur x par une requête similaire sur M. Par conséquent, une variable agit tout simplement comme un *forwarder*.

$$
\begin{array}{rll}
& x[\ fwd\ M\ \mid\ allowIO\ x\ ] \mid rw[\ request\ rw\ x\ \mid P] & \\
\hookrightarrow^{+} & x[\ fwd\ M\ \mid\ allowIO\ x\ ] \mid rw[\ request\ rw\ M\ \mid P] & \text{pour } rw = read \text{ ou } write
\end{array}
$$

Un canal n est simulé par un ambient de nom n contenant un serveur spécial pour les requêtes $read$ (il n'y a pas de serveur pour les requêtes $write$). Lorsque n contient une requête $read$, elle essaie de trouver une requête $write$ et d'en prendre le contrôle (toujours à l'aide du même mécanisme de requête/serveur). Quand ceci est réalisé, la requête $read$ est remplacée par un ambient x dont le contenu est le forwarder apporté par la requête *write*. Puis, les deux continuations sont activées. De plus, des renommages intermédiaires sont nécessaires pour éviter les interférences.

Nous ne détaillerons pas davantage l'encodage car cela ne serait pas très instructif. Sa définition complète est donnée ci-dessous.

$$
\begin{array}{ll}
\{\!\{(\nu n)\ P\}\!\} \triangleq (\nu n)\ \{\!\{P\}\!\} & \{\!\{\mathbf{0}\}\!\} \triangleq \mathbf{0} \\
\{\!\{P \mid Q\}\!\} \triangleq \{\!\{P\}\!\} \mid \{\!\{Q\}\!\} & \{\!\{!P\}\!\} \triangleq\ !\{\!\{P\}\!\}
\end{array}
$$

$$
\begin{array}{l}
\{\!\{\overline{M}\langle M'\rangle.P\}\!\} \triangleq (\nu p)\quad (\ \mathit{write}\ \ [\ \mathit{request\ write}\ M \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{fwd}\ M' \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \mid p[\ \mathit{out\ read}\ .\ \overline{\mathit{open}}\ p\ .\{\!\{P\}\!\}]] \\
\qquad\qquad\qquad\qquad\quad \mid \mathit{open}\ p\ )
\end{array}
$$

$$
\begin{array}{l}
\{\!\{M(x).P\}\!\} \triangleq (\nu p)\quad (\ \mathit{read}\ \ [\ \mathit{request\ read}\ M \\
\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{open\ write}\ .\ \overline{\mathit{out}}\ \mathit{read}\ .\ (\nu x)\ \ \mathit{read\ be}\ x\ . \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\ \overline{\mathit{out}}\ x\ .\ \mathit{allowIO}\ x \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mid p[\ \mathit{out}\ x\ .\ \overline{\mathit{open}}\ p\ .\{\!\{P\}\!\}])] \\
\qquad\qquad\qquad\qquad \mid \mathit{open}\ p\ )
\end{array}
$$

$$
\begin{array}{l}
\{\!\{[n:S]\}\!\} \triangleq (\nu p_1)\ \ldots(\nu p_k) \qquad\qquad \text{(où } \{p_1,\ldots,p_k\} \text{ sont les noms frais de } S\text{)} \\
\qquad (\ n\ \ [\ \mathit{allowIO}\ n \\
\qquad\qquad \mid \mathit{server\ read}\ .\ (\nu p) \\
\qquad\qquad\qquad (\ \overline{\mathit{out}}\ \mathit{read}\ .\ \mathit{read\ be}\ p\ .\ \overline{\mathit{in}}\ p\ .\ \mathit{out}\ n\ .\ p\ \mathit{be\ read} \\
\qquad\qquad\qquad \mid \mathit{enter}[\ \mathit{out\ read}\ .\ \mathit{in\ write}\ .\ \overline{\mathit{open}}\ \mathit{enter}\ .\ \mathit{in}\ p\ .\ \overline{\mathit{open}}\ \mathit{write}\ ]) \\
\qquad\qquad \mid \{\!\{S\}\!\}_n] \\
\qquad \mid \mathit{open}\ p_1\ \mid \ldots \mid\ \mathit{open}\ p_k\ )
\end{array}
$$

$$
\{\!\{(\nu x : M)\ P\}\!\} \triangleq (\nu x)\quad (x[\ \mathit{fwd}\ M\ \mid\ \mathit{allowIO}\ x\ ] \mid \{\!\{P\}\!\})
$$

$$
\{\!\{\varepsilon\}\!\}_n \triangleq \mathbf{0} \qquad\qquad \{\!\{S \mid S'\}\!\}_n \triangleq \{\!\{S\}\!\}_n \mid \{\!\{S'\}\!\}_n
$$

$$
\begin{array}{l}
\{\!\{\langle M\rangle.P\}\!\}_n \triangleq \mathit{write}\ \ [\ \overline{\mathit{in}}\ \mathit{write}\ .\ \mathit{open\ enter} \\
\qquad\qquad\qquad\qquad\quad \mid \mathit{fwd}\ M \\
\qquad\qquad\qquad\qquad\quad \mid p[\ \mathit{out\ read}\ .\ \overline{\mathit{open}}\ p\ .\{\!\{P\}\!\}]] \qquad \text{(où } p \text{ est frais)}
\end{array}
$$

$$
\begin{array}{l}
\{\!\{(x).P\}\!\}_n \triangleq (\nu q)\quad (\ q\ \ [\ \overline{\mathit{in}}\ q\ .\ \mathit{out}\ n\ .\ q\ \mathit{be\ read} \\
\qquad\qquad\qquad\qquad\qquad \mid \mathit{open\ write}\ .\ \overline{\mathit{out}}\ \mathit{read}\ .\ (\nu x)\ \ \mathit{read\ be}\ x\ . \\
\qquad\qquad\qquad\qquad\qquad\qquad (\ \overline{\mathit{out}}\ x\ .\ \mathit{allowIO}\ x \\
\qquad\qquad\qquad\qquad\qquad\qquad \mid p[\ \mathit{out}\ x\ .\ \overline{\mathit{open}}\ p\ .\{\!\{P\}\!\}])] \qquad \text{(où } p \text{ est frais)} \\
\qquad\qquad\qquad\quad \mid \mathit{enter}[\ \mathit{in\ write}\ .\ \overline{\mathit{open}}\ \mathit{enter}\ .\ \mathit{in}\ q\ .\ \overline{\mathit{open}}\ \mathit{write}\ ])
\end{array}
$$

Afin de tenir compte des substitutions, nous ajoutons la définition suivante:

$$
\begin{array}{lll}
\{\!\{\{{}^{M_1}/_{x_1}\} \uplus \ldots \uplus \{{}^{M_k}/_{x_k}\},P\}\!\} & \triangleq & \quad x_1[\ \mathit{fwd}\ M_1\ \mid\ \mathit{allowIO}\ x_1\ ] \\
 & & \mid \ \ldots \\
 & & \mid \ x_k[\ \mathit{fwd}\ M_k\ \mid\ \mathit{allowIO}\ x_k\ ] \\
 & & \mid \{\!\{P\}\!\}
\end{array}
$$

5.2 Résultats

Avant d'énoncer quelques propriétés, nous devons distinguer deux types de réductions dans les safe ambients. Les *réductions principales*, notées $\overset{pr}{\hookrightarrow}$, correspondent intuitivement aux premières réductions des encodages en ambients purs des règles axiomatiques de réduction du π_{esc}-calcul. Plus précisément, il est possible de les repérer en "marquant" certaines capacités spécifiques dans l'encodage. Il s'agit des capacités $\mathit{in}\ n$ et $\mathit{in}\ x$ dans $\mathit{request\ rw}\ n$ et $\mathit{request\ rw}\ x$, et de la capacité *in write* dans l'ambient *enter* dans $\{\![[n : S]]\!\}$. Toute réduction impliquant l'une de ces capacités marquées sera principale. Toutes les autres seront *auxiliaires* et seront notées $\overset{aux}{\hookrightarrow}$.

Nous pouvons alors montrer que toute réduction en π_{esc}-calcul correspond à une réduction principale suivie de plusieurs réductions auxiliaires après encodage.

Proposition 10 *Si $\sigma : P \longmapsto Q$, alors $\{\![\sigma,P]\!\} \overset{pr}{\hookrightarrow} \overset{aux*}{\hookrightarrow} \{\![\sigma,Q]\!\}$.*

Dans l'autre direction, nous pouvons prouver que si un processus encodé possède une réduction principale, il est possible de l'étendre avec un certain nombre de réductions auxiliaires de telle sorte qu'on simule globalement un seul pas de réduction en π_{esc}-calcul. De plus, cette réduction est en un certains sens unique, modulo la congruence structurelle.

Proposition 11 *Si $\{\![\sigma,P]\!\} \overset{pr}{\hookrightarrow} Q$, alors il existe un processus P' tel que $\sigma : P \longmapsto P'$ et $Q \overset{aux*}{\hookrightarrow} \{\![\sigma,P']\!\}$. De plus, si $\sigma : P \longmapsto P''$ et $Q \overset{aux*}{\hookrightarrow} \{\![\sigma,P'']\!\}$, alors $P' \equiv P''$.*

Il faut expliquer pourquoi nous avons dû distinguer les réductions principales et auxiliaires. Un contre-exemple, écrit dans le style de CCS, est $P \triangleq\ !\, a \mid !\, \overline{a} \mid b.C \mid \overline{b}.D$. On a $P \longrightarrow P$ et $P \longrightarrow P' = !\, a \mid !\, \overline{a} \mid C \mid D$. Si on considère la première réduction, le théorème précédent donnerait $\{\![P]\!\} \hookrightarrow Q$, avec $P \longrightarrow P$ et $Q \hookrightarrow^* \{\![P]\!\}$. Mais nous aurions aussi $P \longrightarrow P'$ et $Q \hookrightarrow^* \{\![P']\!\}$, avec $P \not\equiv P'$. Donc la seconde assertion serait fausse. Ce cas devient impossible avec deux types de réduction: il doit nécessairement y avoir une réduction principale entre Q et $\{\![P']\!\}$.

Cependant, la Proposition 11 n'est pas aussi forte que nous le voudrions: on doit toujours atteindre l'encodage suivant à l'aide de réductions auxiliaires avant la réduction principale suivante. En fait, les réductions auxiliaires ne comptent pas: notre encodage fut conçu de telle sorte qu'un nouveau pas effectif de l'exécution (i.e. une réduction principale) puisse avoir lieu dès que possible (parfois un certain nombre de réductions auxiliaires préalables sont nécessaires pour débloquer la situation). C'est pourquoi nous croyons que la conjecture suivante est vraie. La prouver n'est pas difficile en théorie, mais nous nous heurtons à un très grand nombre de cas à examiner, conduisant à une explosion combinatoire dont seul un outil de démonstration automatique pourrait peut-être venir à bout.

Conjecture 12 *$\overset{aux}{\hookrightarrow}$ est confluent avec $\overset{aux}{\hookrightarrow}$ et $\overset{pr}{\hookrightarrow}$.*

Chapitre 6

L'Encodage Final

6.1 L'Encodage

Il reste à composer les résultats des deux Chapitres précédents. L'encodage d'un π-processus P en ambients purs est défini simplement par:

$$\langle\!\langle P \rangle\!\rangle \triangleq \{\!\{ \varnothing, cl_{\varnothing}(P) \}\!\}$$

A l'aide des définitions du Chapitre 5, on peut donner un encodage direct, et non via le π_{esc}-calcul (ces définitions s'appliquent uniquement aux processus sans noms libres; en cas contraire, il faut ajouter un canal vide pour chaque nom libre):

$$
\begin{array}{l}
\langle\!\langle \mathbf{0} \rangle\!\rangle \triangleq \mathbf{0} \\
\langle\!\langle P \mid Q \rangle\!\rangle \triangleq \langle\!\langle P \rangle\!\rangle \mid \langle\!\langle Q \rangle\!\rangle \\
\langle\!\langle !P \rangle\!\rangle \triangleq !\langle\!\langle P \rangle\!\rangle \\
\langle\!\langle (\nu n)\ P \rangle\!\rangle \triangleq (\nu n) \\
\qquad\qquad (\ n\ \ [\ allowIO\ n \\
\qquad\qquad\qquad \mid server\ read\ .\ (\nu p) \\
\qquad\qquad\qquad\qquad (\ \overline{out}\ read\ .\ read\ be\ p\ .\ \overline{in}\ p\ .\ out\ n\ .\ p\ be\ read \\
\qquad\qquad\qquad\qquad \mid enter[\ out\ read\ .\ in\ write\ .\ \overline{open}\ enter\ .\ in\ p\ .\ \overline{open}\ write\ ])] \\
\qquad\qquad \mid \langle\!\langle P \rangle\!\rangle) \\
\langle\!\langle \overline{M}\langle M' \rangle.P \rangle\!\rangle \triangleq (\nu p) \quad (\ write\ \ [\ request\ write\ M \\
\qquad\qquad\qquad\qquad\qquad\qquad \mid fwd\ M' \\
\qquad\qquad\qquad\qquad\qquad\qquad \mid p[\ out\ read\ .\ \overline{open}\ p\ .\langle\!\langle P \rangle\!\rangle]] \\
\qquad\qquad\qquad\qquad \mid open\ p\ ) \\
\langle\!\langle M(x).P \rangle\!\rangle \triangleq (\nu p) \quad (\ read\ \ [\ request\ read\ M \\
\qquad\qquad\qquad\qquad\qquad\qquad \mid open\ write\ .\ \overline{out}\ read\ .\ (\nu x)\ \ read\ be\ x\ . \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad (\ \overline{out}\ x\ .\ allowIO\ x \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mid p[\ out\ x\ .\ \overline{open}\ p\ .\langle\!\langle P \rangle\!\rangle])] \\
\qquad\qquad\qquad\qquad \mid open\ p\ )
\end{array}
$$

Il ne reste qu'à énoncer quelques propriétés de correspondance opérationnelle. Commençons par définir une relation $\approx$ entre le π-calcul et les ambients purs.

Définition 13 *Soit P un π-processus sans variables libres et R un processus en ambients purs. On dira que P et R sont équivalents (et on notera $P \approx R$) s'il existe un π_{esc}-processus Q tel que Q est valide, clos relativement à $\varnothing$, sans variables libres, $P \equiv [\![Q]\!]$ et $\{\!\{\varnothing, Q\}\!\} \equiv R$.*

On vérifie facilement que $P \approx \langle\!\langle P \rangle\!\rangle$ pour tout π-processus P sans variables libres.

Avec cette définition, nous pouvons énoncer le théorème final de correspondance opérationnelle qui valide notre encodage. Il est obtenu par composition du Corollaire 8 avec les Propositions 10 et 11.

Théorème 14 *Supposons que $P \approx R$.*

- *Si $P \longrightarrow P'$, alors il existe un processus R' tel que $R \hookrightarrow^{+} R'$ et $P' \approx R'$.*
- *Si $R \stackrel{pr}{\hookrightarrow} R'$, alors il existe un processus R'' tel que $R' \stackrel{aux*}{\hookrightarrow} R''$, et soit $P \approx R''$ soit $P \longrightarrow P' \approx R''$.*

Chapitre 7

Valorisation

7.1 Résumé de la Contribution

Nous avons donné un encodage du π-calcul synchrone dans le calcul des ambients sans primitives de communication ni substitutions. Nous avons également prouvé une correspondance opérationnelle pour notre encodage. En chemin, nous avons conçu le π_{esc}-calcul afin de faciliter la preuve. Cependant, ce calcul paraît intéressant en lui-même, en raison des résultats d'équivalence avec le π-calcul.

7.2 Autres Approches Envisagées

La première étape de ce travail fut la conception informelle de l'encodage. Puis vint le temps de la preuve. Comme elle s'annonçait longue et difficile, de nombreuses voies furent envisagées:

L'approche directe Prouver directement la correspondance opérationnelle entre le π-calcul et les ambients purs est vite apparu comme une tâche impossible. Cette difficulté était principalement due à l'impossibilité de revenir d'un terme en ambients purs à un terme équivalent en π-calcul, d'autant que le terme en question peut être fortement modifié par congruence structurelle et réductions auxiliaires, de sorte qu'il diffère beaucoup de l'encodage direct d'un π-processus.

Forme normale C'est pourquoi nous avons songé pendant un certain temps à introduire une notion de forme normale sur les ambients purs, à savoir des termes qui se rapprocherait d'une façon ou d'une autre des termes résultant de π-processus encodés. Malheureusement, cette piste n'a rien donné non plus, car il était difficile de trouver la "bonne" forme normale qui pourrait nous permettre de revenir en arrière.

Un calcul intermédiaire Nous avons alors songé à inventer un calcul intermédiaire afin de couper la preuve en deux parties principales. Il faut en effet bien distinguer deux difficultés complètement disjointes: la simulation de la communication et de la substitution du π-calcul par des mécanismes plus élémentaires, et la traduction de ces mécanismes dans le langage particulier des safe ambients. Le calcul intermédiaire nous a permis de séparer ces deux aspects de la preuve et de les traiter séparément.

La recherche du bon calcul intermédiaire a en revanche demandé de nombreuses tentatives. Nous avons testé différentes combinaisons à base de termes en π-calcul associés à des substitutions, etc... Ces diverses versions n'ont pas été inutiles puisqu'elles nous ont permis de comprendre et d'analyser à chaque fois ce qui n'allait pas, et de corriger notre formalisation dans la bonne direction. Finalement, nous avons abouti à une extension relativement simple et directe du π-calcul (par rapport aux autres calculs envisagés). La forme précise des primitives à ajouter a d'ailleurs déjà été discutée au Chapitre 3. Enfin, nous avons pu mener la preuve jusqu'à son terme, mais elle nous a réservé encore de nombreuses surprises en cours de route, puisqu'elle s'est considérablement allongée et qu'il a fallu introduire de nouveaux outils: le typage $\vdash P : Valid$, la clôture des canaux, des résultats sur les canaux vides et leur extension,...

7.3 Futurs Travaux

Le premier futur travail consistera à faire appel à un outil de démonstration automatique pour prouver la Conjecture 12. En cas de succès, nous pourrions énoncer un théorème final bien plus fort pour notre correspondance opérationnelle (à savoir que seules les réductions principales sont vraiment importantes). De plus, notre encodage fut aussi conçu afin d'éviter toutes interférences avec les autres processus (si on restreint les noms internes utilisés par le mécanisme de requête/serveur). Par conséquent, nous aimerions montrer qu'aucune attaque contre le protocole n'est possible en prouvant que P et $(\nu read)$ $(\nu write)$ $(\nu enter)$ $\langle\!\langle P \rangle\!\rangle$ sont équivalents dans n'importe quel contexte.

On pourrait également songer à étendre l'encodage proposé pour qu'il s'applique au π-calcul *polyadique* (i.e. où les valeurs communiquées peuvent être des tuples de taille quelconque). Le passage d'un calcul monadique à un calcul polyadique ne nous paraît pas difficile à réaliser: il suffit de créer plusieurs ambients-variables en même temps lors d'une communication. La seule difficulté réside dans la vérification que le nombre de valeurs dans le tuple et le nombre de variables sont bien identiques (ceci peut être garanti par un système de types en π-calcul [Mil91]). Ceci risque de compliquer le protocole en ambients, mais ne devrait pas soulever de problème théorique majeur. En effet, un certain nombre d'encodages du π-calcul polyadique dans le π-calcul monadique ont déjà été proposés (généralement, associés à un système de types).

En outre, notre travail soulève quelques questions théoriques. Est-il possible d'encoder le π-calcul avec les ambients mobiles classiques au lieu des safe ambients (nous avons déjà expliqué dans l'introduction pourquoi cela nous semblait difficile) ? Nous serions curieux de voir si un tel encodage est possible... Et encore plus important à nos yeux: est-il possible d'encoder le calcul complet des ambients (safe ou non) avec ses primitives de communication dans le même calcul sans primitives de communication (en fait, c'est cette question qui nous a conduit à faire ce travail) ? La principale différence avec l'encodage du π-calcul est que les variables doivent maintenant être présentes à tous les niveaux de la hiérarchie des ambients et pas seulement au niveau global. Par conséquent, elles devraient se dupliquer et se disperser dynamiquement, même dans les ambients fraîchement créés !

Remerciements J'ai bénéficié de nombreuses discussions avec Davide Sangiorgi pour ce travail. Merci aussi à Ilaria Castellani et à Gérard Boudol, ainsi qu'au reste des équipes Mimosa et Tick.

Bibliographie

[Car99a] L. Cardelli. Abstractions for Mobile Computation. In *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, volume 1603 of *Lecture Notes in Computer Science*, pages 51–94. Springer Verlag, 1999. Un article informel sur la programmation distribuée d'objets mobiles.

[Car99b] L. Cardelli. Wide Area Computation. In *Proceedings of ICALP'99*, volume 1644 of *Lecture Notes in Computer Science*, pages 10–24. Springer Verlag, April 1999. Un article informel sur la programmation sur de grands réseaux.

[CG97] L. Cardelli and A. D. Gordon. A Calculus of mobile Ambients. 1997. Des transparents pour comprendre graphiquement le mécanisme des ambients mobiles.

[CG98] L. Cardelli and A. D. Gordon. Mobile Ambients. In *Proceedings FoSSaCS'98*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer Verlag, 1998. C'est la référence incontournable pour une première introduction complète aux ambients mobiles.

[FMQ96] G. Ferrari, U. Montanari, and P. Quaglia. A π-Calculus with Explicit Substitutions. *Theoretical Computer Science*, 168 (1):53–103, November 1996.

[Hir99] D. Hirschkoff. Handling Substitutions Explicitly in the π-Calculus. In *Proceedings of the Floc Workshop WESTAPP 99*, 1999.

[HY95] K. Honda and N. Yoshida. On Reduction-Based Process Semantics. *Theoretical Computer Science*, 152:437–486, 1995.

[LS00] F. Levi and D. Sangiorgi. Controlling Interference in Ambients. In *Proceedings of POPL'00*. ACM Press, 2000. Le papier fondateur pour les safe ambients.

[Mer99] M. Merro. On Equators in Asynchronous Name-Passing Calculi without Matching. In *Proceedings of EXPRESS'99*, volume 27 of *Electronic Notes in Theoretical Science*. Elsevier, 1999.

[Mil91] R. Milner. The Polyadic π-Calculus: a Tutorial. Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991. La référence de base pour une introduction du π-calcul.

[MS98] M. Merro and D. Sangiorgi. On Asynchrony in Name-Passing Calculi. In *Proceedings of ICALP'98*, volume 1443 of *Lecture Notes in Computer Science*, pages 856–867. Springer Verlag, 1998.

[Pal97] C. Palamidessi. Comparing the Expressive Power of the Synchronous and the Asynchronous π-Calculus. In *Proceedings of POPL'97*, pages 256–265. ACM, 1997.

[Zim00a] P. Zimmer. On the Expressiveness of Pure Mobile Ambients. Technical report, INRIA, 2000. Rapport technique contenant les preuves (bientôt disponible).

[Zim00b] P. Zimmer. On the Expressiveness of Pure Mobile Ambients. In *Proceedings of EXPRESS'00*, 2000. Article à paraître.