Un format pour que la bisimulation soit une congruence dans les langages de processus avec mobilité

Axelle Ziegler

14 septembre 2004

Equipe

Ce travail a été effectué sous la co-direction de Catuscia Palamidessi et Dale Miller, dans les équipes Comete et Parsifal de l'INRIA-Futurs dans les locaux du LIX. Parsifal (Preuves Automatiques et Raisonnements sur des Spécifications Logiques) et Comete (COncurrence Mobilité et Transaction) sont deux nouveaux projets présentés à l'INRIA respectivement par Catuscia Palamidessi et Dale Miller. Le projet Parsifal s'intéresse à l'utilisation de spécifications logiques pour la sémantique opérationnelle de différents calculs, dont le π-calcul. Un des intérêts de l'utilisation de telles spécifications est bien sûr de pouvoir espérer mécaniser les déductions dans ces systèmes, en ayant un formalisme mieux adapté à l'encodage dans des outils formels. Le projet Comete, quant à lui s'intéresse aux fondations théoriques des langages concurrents et distribués, en particulier la théorie du π-calcul asynchrone probabiliste et l'utilisation de langages fonctionels de haut niveau pour le développement d'applications distribuées. Mon travail a consisté à essayer d'utiliser des résultats sur la formalisation du π-calcul dans le cadre formel développé dans le cadre du projet Parsifal, en particulier par Dale Miller et Alwen Tiu (voir [1], [2]) pour prouver des résultats généraux sur la bisimulation des langages concurrents avec mobilité. Ce travail s'inscrit dans le prolongement du travail effectué dans [1], et vise à appliquer celui-ci à un problème plus spécifique.

Chapitre 1

Cadre logique

1.1 Introduction

On utilise dans ce travail une extension du calcul des séquents avec un opérateur ∇, ainsi que des définitions. L'introduction de comportements infinis exige la présence d'un principe de co-induction, pour pouvoir par exemple utiliser le "bang" du π-calcul. Enfin, la présentation utilise la "syntaxe abstraite d'ordre supérieur". On donnera dans cette section quelques résultats sur ces sujets, en particulier tirées de [2] pour le cadre logique, et [3] pour l'utilisation de la syntaxe abstraite d'ordre supérieur.

1.1.1 Syntaxe abstraite d'ordre supérieure

Dans ce travail, on encodera les algèbres de processus en utilisant des λ-termes pour encoder la liaison des noms. L'avantage de cette syntaxe est de permettre une présentation unifiée de l'abstraction et de la liaison, d'utiliser directement les notions d'α-équivalence et de substitution sans capture de variables de la théorie du λ-calcul. Il faut garder à l'esprit que dans ce cas, la liaison par un λ ne représente pas nécessairement un terme dont le modèle est une fonction. Dans le cadre de la mobilité, les abstractions ne porteront que sur des variables représentant des noms, mais on peut parfaitement envisager d'utiliser une syntaxe du même type pour représenter des langages avec un véritable ordre supérieur. Pour plus de détails sur l'encodage du π-calcul dans cette syntaxe, on peut se référer à [3]. Notons que cette présentation est très proche de celle employée par Robin Milner dans [4] pour le π-calcul avec abstractions et concrétions. Nous reviendrons plus tard sur cette similitude. Un des intérêts de cette syntaxe permet une représentation unifiée des liaisons dans les langages avec mobilité, faisant le lien, par exemple entre la communication et l'extrusion de portée.

1.2 la logique $FO\lambda^{\Delta\nabla}$

Pour raisonner sur cette présentation syntaxique, nous utiliserons la logique $FO\lambda^{\Delta\nabla}$ (voir [2]). On présente cette logique sous forme d'une extension du calcul des séquents LJ de Gentzen. La simple utilisation du calcul des séquents pour encoder le π-calcul en utilisant des variables pour encoder les noms pose un problème : si l'on a une preuve du séquent $\vdash Pxy$, où x et y sont deux variables différentes, alors on peut également prouver $\vdash Pzz$, attendu que les variables libres sont considérées quantifiées universellement. Si l'on veut essayer de traduire la bisimulation, cette dérivation pose problème. L'intérêt de l'utilisation de la logique $FO\lambda^{\Delta\nabla}$ réside dans l'extension du calcul des séquents traditionnels en ajoutant une notion de portée locale. Les séquents sont alors de la forme

$$\Sigma\,;\, \sigma_1 \triangleright B_1, \ldots, \sigma_n \triangleright B_n \vdash \sigma_0 \triangleright B_0$$

où Σ est la signature *globale* qui contient les variables dont la portée est le séquent entier, et σ_i sont des signatures *locales* c'est-à-dire qui contiennent des variables dont la portée est limitée à B_i. Les signatures locales fournissent une forme de constantes locales, c'est-à-dire qu'elles ne peuvent pas être instantiées au cours de la preuve. Toutes ces signatures sont considérées comme des liaisons, et on peut donc α-renommer les variables à l'intérieur de leur portée.

On donne l'intégralité des règles d'inférence pour $FO\lambda^{\Delta\nabla}$ dans Figure 1.1. Ces règles sont les mêmes que pour la présentation habituelle de LJ, avec deux différences majeures : on introduit un nouveau quantificateur ∇, qui introduit des variables dans les signatures locales pendant la recherche de preuve, et les règles d'inférence pour $\forall$ et $\exists$ doivent garantir que la portée des variables de la signature locale est respectée. Enfin, on a ici une logique de *premier odre*, dans le sens où on n'autorise pas la quantification sur les propositions (même si l'on peut bien sûr quantifier sur les λ-termes).

Dans la logique que nous utiliserons, nous ajouterons à ce noyau propositionnel des définitions et des principes d'induction et de co-induction, afin de pouvoir encoder complètement nos calculs à l'intérieur de la logique.

Définition 1. *Une* définition *s'écrit* $\forall \bar{x}[p\,\bar{t} \stackrel{\triangle}{=} B]$*, où* p *est une constante, toutes les variables libres de* B *apparaissent dans au moins un terme de* $\bar{t}$ *et toutes les variables libres de* $\bar{t}$ *apparaissent dans* $\bar{x}$*. Le prédicat* p *n'a que des occurences positives dans* B*, c'est à dire qu'il n'apparait pas à gauche d'une implication.*

Intuitivement, le sens de ces définitions est que l'on peut prouver $p(\bar{t})$ si l'on peut prouver B. Pour les détails de l'utilisation de ces définitions,

$$\frac{}{\Sigma\,;\,\sigma \triangleright B, \Gamma \vdash \sigma \triangleright B}\ \mathit{init} \qquad \frac{\Sigma\,;\,\Delta \vdash \mathcal{B} \qquad \Sigma\,;\,\mathcal{B}, \Gamma \vdash \mathcal{C}}{\Sigma\,;\,\Delta, \Gamma \vdash \mathcal{C}}\ \mathit{cut}$$

$$\frac{\Sigma\,;\,\sigma \triangleright B, \sigma \triangleright C, \Gamma \vdash \mathcal{D}}{\Sigma\,;\,\sigma \triangleright B \wedge C, \Gamma \vdash \mathcal{D}}\ \wedge\mathcal{L} \qquad \frac{\Sigma\,;\,\Gamma \vdash \sigma \triangleright B \qquad \Sigma\,;\,\Gamma \vdash \sigma \triangleright C}{\Sigma\,;\,\Gamma \vdash \sigma \triangleright B \wedge C}\ \wedge\mathcal{R}$$

$$\frac{\Sigma\,;\,\sigma \triangleright B, \Gamma \vdash \mathcal{D} \qquad \Sigma\,;\,\sigma \triangleright C, \Gamma \vdash \mathcal{D}}{\Sigma\,;\,\sigma \triangleright B \vee C, \Gamma \vdash \mathcal{D}}\ \vee\mathcal{L}$$

$$\frac{\Sigma\,;\,\Gamma \vdash \sigma \triangleright B}{\Sigma\,;\,\Gamma \vdash \sigma \triangleright B \vee C}\ \vee\mathcal{R} \qquad \frac{\Sigma\,;\,\Gamma \vdash \sigma \triangleright C}{\Sigma\,;\,\Gamma \vdash \sigma \triangleright B \vee C}\ \vee\mathcal{R}$$

$$\frac{\Sigma\,;\,\Gamma \vdash \sigma \triangleright B \qquad \Sigma\,;\,\sigma \triangleright C, \Gamma \vdash \mathcal{D}}{\Sigma\,;\,\sigma \triangleright B \supset C, \Gamma \vdash \mathcal{D}}\ \supset\mathcal{L} \qquad \frac{\Sigma, h\,;\,\Gamma \vdash \sigma \triangleright B[(h\ \sigma)/x]}{\Sigma\,;\,\Gamma \vdash \sigma \triangleright \forall x.B}\ \forall\mathcal{R}$$

$$\frac{\Sigma, \sigma \vdash t : \gamma \qquad \Sigma\,;\,\sigma \triangleright B[t/x], \Gamma \vdash \mathcal{C}}{\Sigma\,;\,\sigma \triangleright \forall_\gamma x.B, \Gamma \vdash \mathcal{C}}\ \forall\mathcal{L} \qquad \frac{\Sigma, \sigma \vdash t : \gamma \qquad \Sigma\,;\,\Gamma \vdash \sigma \triangleright B[t/x]}{\Sigma\,;\,\Gamma \vdash \sigma \triangleright \exists_\gamma x.B}\ \exists\mathcal{R}$$

$$\frac{\Sigma, h\,;\,\sigma \triangleright B[(h\ \sigma)/x], \Gamma \vdash \mathcal{C}}{\Sigma\,;\,\sigma \triangleright \exists x.B, \Gamma \vdash \mathcal{C}}\ \exists\mathcal{L}$$

$$\frac{\Sigma\,;\,(\sigma, y) \triangleright B[y/x], \Gamma \vdash \mathcal{C}}{\Sigma\,;\,\sigma \triangleright \nabla x\ B, \Gamma \vdash \mathcal{C}}\ \nabla\mathcal{L} \qquad \frac{\Sigma\,;\,\Gamma \vdash (\sigma, y) \triangleright B[y/x]}{\Sigma\,;\,\Gamma \vdash \sigma \triangleright \nabla x\ B}\ \nabla\mathcal{R}$$

$$\frac{}{\Sigma\,;\,\sigma \triangleright \bot, \Gamma \vdash \mathcal{B}}\ \bot\mathcal{L} \qquad \frac{}{\Sigma\,;\,\Gamma \vdash \sigma \triangleright \top}\ \top\mathcal{R}$$

$$\frac{\Sigma\,;\,\mathcal{B}, \mathcal{B}, \Gamma \vdash \mathcal{C}}{\Sigma\,;\,\mathcal{B}, \Gamma \vdash \mathcal{C}}\ c\mathcal{L} \qquad \frac{\Sigma\,;\,\Gamma \vdash \mathcal{C}}{\Sigma\,;\,\mathcal{B}, \Gamma \vdash \mathcal{C}}\ w\mathcal{L}$$

FIG. 1.1 – Les règles d'inférence dans $FO\lambda^{\Delta\nabla}$.

et pour les résultats sur les propriétés de la logique, nous renvoyons le lecteur à [2]. Signalons que si les définitions respectent certaines contraintes de stratification, ce qui sera toujours le cas dans la suite, on conserve la propriété d'élimination des coupures. Pour raisonner à l'aide de ces définitions, on introduit deux règles supplémentaires dans notre logique.

$$\frac{\{\Sigma\theta\,;\,\mathcal{B}\theta, \Gamma\theta \vdash \mathcal{C}\theta \mid \theta \in Unif(\mathcal{A},\mathcal{H}) \text{ il y a une définition telle que } \mathcal{H} \stackrel{\triangle}{=} \mathcal{B}\}}{\Sigma\,;\,\mathcal{A}, \Gamma \vdash \mathcal{C}}\ def\mathcal{L}$$

Dans cette règle, $Unif(\mathcal{A},\mathcal{H})$ désigne l'ensemble complet des unificateurs de $\mathcal{A}$ et $\mathcal{H}$, c'est à dire un ensemble de substitutions tel que pour toute substitution θ telle que $\mathcal{A}\theta = \mathcal{H}\theta$, il existe $\sigma \in Unif(\mathcal{A},\mathcal{H})$ et une substitution σ' tel que $\theta = \sigma \circ \sigma'$. Cette règle est en fait une analyse de cas sur les différentes preuves possible de $\mathcal{A}$. Selon la forme des définitions de $\mathcal{A}$, cela peut même représenter une forme de *pattern matching*.

$$\frac{\Sigma\,;\,\Gamma \vdash \mathcal{B}\theta}{\Sigma\,;\,\Gamma \vdash \mathcal{A}}\ def R, \quad \text{où } \mathcal{H} \stackrel{\triangle}{=} \mathcal{B} \text{ est une définition, et} \mathcal{H}\theta = \mathcal{A}$$

Cette règle représente elle la recherche d'une preuve possible de $\mathcal{A}$, à la manière du *backchaining* en programmation logique.

Si l'on définit des prédicats de façon récursive, ces définitions encodent une forme de point fixe : supposons donnée une définition de la forme $p\ x \stackrel{\triangle}{=} B\ p\ x$, on définit $\mathcal{S}$ comme étant l'ensemble des termes tels que $p\ x$ est prouvable, et une fonction F des ensembles de termes dans les ensembles de termes, définie par

$$F(\mathcal{T}) = \{t \in \mathcal{T} | B\ p\ t\ est\ prouvable\}$$

Il est clair que $\mathcal{S}$ est un point fixe de cette fonction. On a cependant aucune garantie que ça ne soit le plus petit, ou le plus grand, ce qui est indispensable dans certains cas.

Les restrictions que l'on a posé sur les occurences de p à l'intérieur du corps de la définition garantissent que la fonction F que nous avons définie sera monotone, et on peut donc utliser des pré- et post- points fixes pour calculer le plus petit et le plus grand. Si on a une définition de la forme $p\ \bar{x} \stackrel{\triangle}{=} B\ p\ \bar{x}$, et S un prédicat de même type que p, les régles d'induction et de co-induction pour ce prédicat seront :

$$\frac{\bar{x}\,;\,B\ S\ \bar{x} \vdash S\ \bar{x} \quad \Sigma\,;\,\Gamma, \bar{y} \triangleright S\ \bar{t} \vdash \mathcal{C}}{\Sigma\,;\,\Gamma, \bar{y} \triangleright S\ \bar{t} \vdash \mathcal{C}}\ \mu\mathcal{L} \quad \frac{\Sigma\,;\,\Gamma \vdash \bar{y} \triangleright B\ p\ \bar{t}}{\Sigma\,;\,\Gamma \vdash \bar{y} \triangleright p\ \bar{t}}\ \mu\mathcal{R}$$

Ici, S représente un invariant de l'induction, c'est-à-dire un pré-point-fixe de la définition B. On a des règles duales pour le principe de co-induction.

$$\frac{\Sigma\,;\,\bar{y} \triangleright B\ p\ \bar{t} \vdash \mathcal{C}}{\Sigma\,;\,\bar{y} \triangleright p\ \bar{t} \vdash \mathcal{C}}\ \nu\mathcal{L} \qquad \frac{\Sigma\,;\,\Gamma \vdash \bar{y} \triangleright S\ \bar{t}\ \bar{x}\,;\ S\ \bar{x} \vdash (B\ S\ \bar{x})}{\Sigma\,;\,\Gamma \vdash \bar{y} \triangleright p\ \bar{t}}\ \nu\mathcal{R}$$

ici, S représente un post-point-fixe de notre définition. Pour ne pas rendre la logique inconsistante, on doit annoter respectivement par $\overset{\mu}{=}$ et $\overset{\nu}{=}$ les définitions inductives ou co-inductives, et une définition ne peut alors être utilisée qu'avec les règles correspondantes. Pour plus de détails sur la théorie de ces définitions, on peut se référer à [5].

1.3 Encodage du π-calcul

Pour motiver notre travail, il est important de rappeler que le cadre logique que nous venons de présenter permet d'encoder complètement le π-calcul et à la fois la bisimulation forte et la bisimulation ouverte dans la sémantique "late" de celui-ci (voir [1] pour le π-calcul fini et [5] pour la version avec "bang"). Tout l'intérêt de ce cadre est de pouvoir faire cet encodage en supprimant toute forme de conditions à côté des règles : tout est encodé à l'intérieur de la logique elle-même, ce qui rend les raisonnements généraux plus faciles.

Chapitre 2

Présentation du travail

2.1 Buts

Le but de ce travail est de formaliser dans le cadre présenté précedemment des restrictions sur les règles utilisées pour définir notre système de transition afin que l'"open bisimulation" soit une congruence. On commencera par définir précisément ce qu'on entend par système de transition, par bisimulation, par congruence dans ce cadre, avant de présenter le résultat obtenu, puis quelques applications. De nombreux résultats similaires existent pour le premier ordre sans mobilité (citons en particulier [6], [7]).

On considère un langage de processus avec mobilité avec deux types de base : les noms de canaux, de type n et les processus de types p. On considérera par ailleurs des λ-abstractions de processus, de type $n \to p$. Celles-ci permettent d'exprimer la liaison des noms, et d'utiliser la théorie d'α-équivalence et de substitutions sans capture de variables disponible dans le λ-calcul simplement typé. On a par ailleurs un ensemble C de constructeurs, de type $t_1 \to \cdots \to t_i \to p$, avec $i \in \mathbb{N}$, and $t_i \in \{n, p, n \to p)$.

Les éléments que nous considérerons seront toujours typés dans un certain contexte Γ, qui sera une liste de paires $(x : t)$, où x est une variable et $t \in \{n, p, n \to p$ un type, tel que si $\Gamma \blacktriangleright P : t$, alors $fv(P) \subseteq \Gamma$, et comme nous ne considérerons que des processus bien typés, ceux-ci viendront toujours avec un contexte de typage. On dénotera les variables désignant des noms par $x, y, \ldots$, celles désignant des abstractions de processus par $M, N, \ldots$ et celles désignant des processus par $P, Q, \ldots$. Si le contexte de typage est omis, on considérera qu'il contient exactement les variables du terme, avec les conventions de notation que l'on vient d'énoncer.

2.2 Définitions

2.2.1 Cadre formel

Avant toute chose, il est nécessaire de rappeler un certain nombre de points. On encode ici exclusivement la mobilité, on n'a donc de λ-abstractions que sur les noms, et pas sur les processus. Par ailleurs, on peut trouver des noms quantifiés à tout endroit dans la spécification de nos transitions (i.e. avec n'importe quelle portée). Ceci n'affecte en rien les résultats, puisque la seule relation que l'on considère sur les noms (et donc sur les actions) est l'égalité, à α-équivalence près dans le cas des actions.

On va définir ici les notions de relation et de congruence dans le cadre utilisé, et discuter de la pertinence de ces notions. On introduira une distinction entre deux types d'abstraction, selon le contexte dans lequel celle-ci sont utilisées, que l'on notera $n \xrightarrow{\nabla} p$ et $n \xrightarrow{\forall} p$. Cette distinction n'intervient que pour affiner notre notion de congruence, et n'a aucune influence sur la sémantique de notre calcul. Elle joue cependant un rôle central, en distinguant les occurences d'abstraction dénotant de véritables fonctions des noms vers les processus des occurences où l'abstraction ne représente que la liaison d'une variable de nom qui n'est pas destinée à être instantiée. On considérera que chacune des occurences d'une abstraction dans un constructeur du langage est ainsi annotée. Nous attirons l'attention du lecteur sur le fait que cette annotation peut intervenir après la spécification du système de transition : elle est spécifique au système dans son ensemble et pas à l'algèbre des termes.

On considère qu'une relation est dans ce cadre définie par quatre ensembles : deux ensembles R_n et R_p qui représentent respectivement des couples de noms et de processus bien typés, et deux ensembles $R_n \xrightarrow{\nabla} p$ et $R_n \xrightarrow{\forall} p$ représentant deux relations sur les abstractions de processus. Dans toute la suite, R_n sera toujours l'égalité structurelle des noms. Considérer des relations plus complexes sur les noms pourrait constituer une extension intéressante de ce travail, par exemple pour encoder directement la fusion des noms dans certains calculs.

Définition 2. *Soit R un prédicat binaire sur les processus (i.e. un terme de type $p \to p \to o$ dans la logique), la relation associée à R sera définie de la façon suivante :*

R_n *: égalité*

R_p *:* $\{\Gamma \blacktriangleright (P : p, Q : p) / \vdash \forall \Gamma R\ P\ Q\}$ *(où* $\forall\Gamma$ *signifie "* $\forall x \in \Gamma$*")*

$R_{n \xrightarrow{\nabla} p}$ *:* $\{\Gamma \blacktriangleright (M : n \xrightarrow{\nabla} p, N : n \xrightarrow{\nabla} p) / \vdash \nabla y \forall \Gamma R M y N y\}$

$$R_{n \xrightarrow{\forall} p} : \{\Gamma \blacktriangleright (M : n \xrightarrow{\forall} p, N : n \xrightarrow{\forall} p) / \vdash \forall y \forall \Gamma R M y N y\}$$

La différence entre $n \xrightarrow{\nabla} p$ et $n \xrightarrow{\forall} p$ apparait clairement ici. Cette différence est extrèmement importante, car c'est elle qui permet de donner une expressivité suffisante pour encoder le fait qu'une relation soit une congruence. En effet, si on quantifie universellement toutes les variables, on "cache" le problème de l'identification de noms différents à l'origine, et on peut par exemple prouver que la bisimulation forte est une congruence. Il importe également de remarquer que si $\vdash \forall\ x\ P \supset \nabla x\ P$ n'est pas un théorème de la logique, on peut toujours construire une preuve de $\vdash \nabla x\ P$ à partir d'une preuve de $\vdash \forall x\ P$. Par conséquent, on obtient facilement que $R_{n \xrightarrow{\forall} p} \subseteq R_{n \xrightarrow{\nabla} p}$, c'est-à-dire qu'une de ces deux relations est un raffinement de l'autre.

On propose la définition suivante pour la notion de congruence dans ce cadre.

Définition 3. *On dit qu'une relation associée à un prédicat R est une congruence si elle satisfait les propriétés suivantes :*

(Equ) : $R_n, R_p, R_{n \xrightarrow{\nabla} p}, R_{n \xrightarrow{\forall} p}$ sont des relations d'équivalences.

(λ) : Si $\Gamma, x : n \blacktriangleright P : p, Q : p \in R_p$ alors $\Gamma \blacktriangleright (\lambda x.P : n \xrightarrow{\forall} p, \lambda x.Q : n \xrightarrow{\forall} p) \in R_{n \xrightarrow{\forall} p}$ et $(\lambda x.P, \lambda x.Q) \in R_{n \xrightarrow{\nabla} p}$.

(Cons) : Si c est un constructeur de type $t_1 \to t_2 \to \cdots \to t_n \to p$, et si (P_1, Q_1), $\ldots$, (P_n, Q_n) sont respectivement dans $R_{t_1}, \ldots, R_{t_n}$, $c(P_1, \ldots, P_n)$ est dans R_p.

Notons que dans le cas qui nous préoccupe on peut réduire la condition λ à

$$Si\ \Gamma, x : n \blacktriangleright P : p, Q : p \in R_p\ alors\ \Gamma \blacktriangleright (\lambda x.P : n \xrightarrow{\forall} p, \lambda x.Q : n \xrightarrow{\forall} p) \in R_{n \xrightarrow{\forall} p}$$

Cette définition mérite sans doute qu'on s'y attarde quelque peu. En effet contrairement à par exemple la présentation du calcul avec abstraction et concrétion, dans [4], on définit ici une notion de congruence qui "recolle" les différents niveaux de la présentation (abstraction et processus), en traitant l'abstraction comme les autres constructeurs. Cette définition de la congruence a donc une certaine généralité, et on peut l'éspérer l'utiliser avec la syntax abstraite d'ordre supérieur dans d'autres cas que celui qui nous préoccupe. En particulier, le lien entre cette définition et la théorie des relations logiques (voir [8] par exemple), pourrait permettre d'énoncer une définition de la congruence suffisamment générale pour s'appliquer à des calculs d'ordre supérieur.

$$
\begin{aligned}
\mathit{obisim}\ P\ Q \overset{\nu}{=}\ & \forall A \forall P'\ [(P \xrightarrow{A} P') \supset \exists Q'.(Q \xrightarrow{A} Q') \wedge \mathit{obisim}\ P'\ Q'] \wedge \\
& \forall A \forall Q'\ [(Q \xrightarrow{A} Q') \supset \exists P'.(P \xrightarrow{A} P') \wedge \mathit{obisim}\ Q'\ P'] \wedge \\
& \forall X \forall P'\ [(P \overset{\downarrow X}{\rightharpoonup} P') \supset \exists Q'.(Q \overset{\downarrow X}{\rightharpoonup} Q') \wedge \forall w.\mathit{obisim}\ (P'w)\ (Q'w)] \wedge \\
& \forall X \forall Q'\ [(Q \overset{\downarrow X}{\rightharpoonup} Q') \supset \exists P'.(P \overset{\downarrow X}{\rightharpoonup} P') \wedge \forall w.\mathit{obisim}\ (Q'w)\ (P'w)] \wedge \\
& \forall X \forall P'\ [(P \overset{\uparrow X}{\rightharpoonup} P') \supset \exists Q'.(Q \overset{\uparrow X}{\rightharpoonup} Q') \wedge \nabla w.\mathit{obisim}\ (P'w)\ (Q'w)] \wedge \\
& \forall X \forall Q'\ [(Q \overset{\uparrow X}{\rightharpoonup} Q') \supset \exists P'.(P \overset{\uparrow X}{\rightharpoonup} P') \wedge \nabla w.\mathit{obisim}\ (Q'w)\ (P'w)]
\end{aligned}
$$

FIG. 2.1 – Spécification de la bisimulation

2.2.2 L'encodage du système de transition

Enfin, notre langage contient un type a, dénotant les "actions", i.e les étiquettes sur les transitions. Ce type dispose de 3 constructeurs : $\uparrow$: $n \to n \to a$, $\downarrow .n \to n \to a$ et $\tau : a$, dénotant naturellement les actions d'entrée, de sortie, et les transitions silencieuses. On a également deux constructeurs pour coder les prédicats de transitions, $\to$ de type $p \to p \to a \to o$ et $\rightharpoonup$ type $p \to (n \to p) \to (n \to a) \to o$, qui dénotent respectivement les transitions "libres" et celles dans lesquelles le nom transmis est lié. On note que dans le cas d'une transition liée, on n'étiquette les transitions que par le nom du canal, le nom transmis étant lié par une abstraction. Dans ce cadre, on définit un prédicat *obisim* (Figure 2.1), qui correspond à l'encodage de la "bisimulation ouverte" du π-calcul. On notera la différence de traitement entre les entrées et les sorties liées, qui relève de la même logique que notre distinction entre $n \xrightarrow{\nabla} p$ et $n \xrightarrow{\forall} p$. Cette définition de la bisimulation n'est donc adaptée que pour des langages où la sortie et l'entrée sont assymétriques et où l'instanciation et la substitution sont complètement décalées du coté du "receveur", à la manière du π-calcul. Cependant, changer la quantification à l'intérieur de cette définition demanderait juste de modifier les contraintes sur les annotations de type des abstractions, donc on peut très facilement généraliser notre résultat à une bisimulation "symétrique".

Le but de notre travail est d'essayer de déterminer des contraintes sur le langage et sur le système de transition pour que la relation induite par la définition 2 pour ce prédicat soit une congruence.

2.3 Le format des règles

Dans la spécification des règles du système de transition, on utilise des règles de la forme de la syntaxe SOS ([9]) :
$P \overset{A}{\rightsquigarrow} Q \overset{\triangle}{=} \bigwedge_{i\in I} P_i \overset{A_i}{\rightsquigarrow} Q_i$, avec $I \subseteq \mathbb{N}$, $\rightsquigarrow \in \{\rightarrow, \multimap\}$, P, P_i de type p, Q, Q_i de type p ou $n \rightarrow p$ et A, A_i de type a ou $n \rightarrow a$ selon le type de $\rightsquigarrow$ pour chaque règle. Par ailleurs, on autorise la présence de quantifications sur les noms ou les actions (on peut assez facilement voir qu'une quantification sur les actions peut assez facilement se décomposer en quantification sur les noms) à l'intérieur de notre défintion. Par contre, toutes les variables de processus qui apparaissent dans ces définitions sont considérées universellement quantifiées à l'extérieur de la définition.

On va définir ici les différentes contraintes que l'on impose à notre système.

Définition 4. *Des règles seront dites en format tyft/tyxt si elles sont dans une des deux formes suivantes :*

- $f(x_1, \ldots, x_n) \overset{A}{\rightsquigarrow} Q \overset{\triangle}{=} \bigwedge_{i\in I} P_i \overset{A_i}{\rightsquigarrow} y_i$ *où les* x_j *et les* y_i *sont des variables toutes différentes*
- $x \overset{A}{\rightsquigarrow} Q \overset{\triangle}{=} \bigwedge_{i\in I} P_i \overset{A_i}{\rightsquigarrow} y_i$ *où* x *et les* y_i *sont des variables toutes différentes.*

Notons que la présence de la deuxième possibilité n'ajoute pas de pouvoir expressif, puisqu'elle est intégralement codable dans la première, en rajoutant une règle pour chaque constructeur. Toutefois, elle permet une présentation syntaxiquement plus agréable.

Définition 5. *On définit le graphe de dépendance d'une règle comme le graphe orienté dont les sommets sont les variables libres apparaissant dans le prémisse de la règle et dont les arètes sont les flèches reliant les variables apparaissant à droite d'un prédicat de transition et les variables apparaissant à gauche du même prédicat. Une règle sera dite* sans dépendance circulaire *si ce graphe est acyclique.*

Ces restrictions sont exactement identiques à celles formalisées dans [6] pour le premier ordre sans mobilité, i.e. des calculs du type de CCS. Nous avons besoin d'une restriction supplémentaire, afin de s'assurer que les deux types de quantification pour les noms n'entrent pas en conflit :

Un constructeur dont le type contient $n \overset{\nabla}{\longrightarrow} p$ ne devra pas apparaitre à gauche d'une transition de type $P \overset{\downarrow x}{\multimap} M$.

En fait cette restriction peut être relachée en : si un terme est utilisé à gauche d'une telle transition avec le type $n \overset{\nabla}{\longrightarrow} p$, ni lui ni aucun de ses

$$
\begin{aligned}
BpPQ = \;& \forall A \forall P' \; [(P \xrightarrow{A} P') \supset \exists Q'.(Q \xrightarrow{A} Q') \wedge pP'Q'] \wedge \\
& \forall A \forall Q' \; [(Q \xrightarrow{A} Q') \supset \exists P'.(P \xrightarrow{A} P') \wedge pQ'P'] \wedge \\
& \forall X \forall P' \; [(P \xrightarrow{\downarrow X} P') \supset \exists Q'.(Q \xrightarrow{\downarrow X} Q') \wedge \forall w.p(P'w)(Q'w)] \wedge \\
& \forall X \forall Q' \; [(Q \xrightarrow{\downarrow X} Q') \supset \exists P'.(P \xrightarrow{\downarrow X} P') \wedge \forall w.p(Q'w)(P'w)] \wedge \\
& \forall X \forall P' \; [(P \xrightarrow{\uparrow X} P') \supset \exists Q'.(Q \xrightarrow{\uparrow X} Q') \wedge \nabla w.p(P'w)(Q'w)] \wedge \\
& \forall X \forall Q' \; [(Q \xrightarrow{\uparrow X} Q') \supset \exists P'.(P \xrightarrow{\uparrow X} P') \wedge \nabla w.p(Q'w)(P'w)]
\end{aligned}
$$

sous-termes ne doivent apparaitre à droite de la flèche, et ce tant dans les prémisses que dans la conclusion de la règle.

Enfin, on peut énoncer le résultat suivant :

Theorème 6. *Si les règles contenues dans la spécification de notre système de transition sont toutes en format tyft/tyxt, ne contiennent pas de dépendances circulaires et respectene la restriction sur la quantification, la relation engendrée par le prédicat obisim pour ce système est une congruence au sens de la définition 3*

La base de la preuve est une co-induction : on construit un sur-ensemble de la bisimulation qui est une congruence, et on montre qu'il constitue un post-point fixe de la définition de la bisimulation. On définit tout d'abord un prédicat *congr* par induction :

- $congr\ P\ Q \stackrel{\triangle}{=} obisim\ P\ Q$
- $congr\ f(P_1, \ldots, P_n)\ f(Q_1, \ldots, Q_n) \stackrel{\mu}{=} \bigwedge congr\ P_i\ Q_i$(si P_i, Q_i sont de type p)$\bigwedge \forall y congr\ P_i y\ Q_i y$ (si P_i, Q_i sont utilisés avec un type $n \xrightarrow{\forall} p$) $\bigwedge \nabla y congr\ P_i y\ Q_i y$ (si P_i, Q_i sont utlisés avec un type $n \xrightarrow{\nabla} p$)

Nous allons maintenant prouver une série de lemmes nous conduisant au résultat recherché :

Lemme 7. *Dans le système dans lequel nous nous sommes placés, on peut prouver*

$$P : p, Q : p \,;\, congr\ P\ Q \vdash B\ congr\ P\ Q$$

où B est défini figure 2.3, et est tel que $obisim \stackrel{\nu}{=} BobisimPQ$.

On ne donnera ici que le principe de la preuve, les détails ne présentant que peu d'intérèt. Le lecteur pardonnera l'imprécision de certaines parties de la preuve, afin que celle-ci reste lisible. Pour des raisons de clarté de la

présentation, on supposera que dans la suite Cong désigne le prédicat suivant. Cong n'est en aucun cas une relation, au sens des définitions précédentes. C'est simplement une commodité de notations pour désigner les différents prédicats dans lesquels *congr* apparait.

$$\text{Si } (P : p,\ Q : p),\ Cong\ P\ Q \text{ représente } congr\ P\ Q$$
$$\text{Si } (M : n \xrightarrow{\forall} p,\ N : n \xrightarrow{\forall} p),\ Cong\ M\ N \text{ représente } \forall y,\ congr\ My\ Ny$$
$$\text{Si } (M : n \xrightarrow{\nabla} p,\ N : n \xrightarrow{\nabla} p),\ Cong\ M\ N \text{ représente } \nabla y,\ congr\ My\ Ny$$

On commence par appliquer $(\forall\mathcal{R})$ précédé de $(defR)$, avec deux définitions possibles. On a donc un des deux séquents suivants à prouver (Dans le deuxième cas, on unifie P avec $f(P_1, \ldots, P_n, \bar{x})$ et Q avec $f(Q_1, \ldots, Q_n, \bar{x})$, où $\bar{x}$ sont tous les noms parmi les arguements de f. Bien entendu, cette permutation des arguments de f est uniquement utilisée pour rendre la preuve plus lisible.

$$P : p,\ Q : p,\ \Gamma\ ;\ obisim\ P\ Q \vdash B\ congr\ P\ Q$$
$$P_1,\ Q_1 : t_1,\ \ldots,\ P_n,\ Q_n : t_n, \hat{x}, \Gamma\ ;\ \bigwedge Cong\ P_i\ Q_i \vdash B\ congr\ P\ Q$$

Dans le premier cas, une application de $defR$ donne

$$P : p,\ Q : p,\ \Gamma\ ;\ B\ obisim\ P\ Q \vdash B\ congr\ P\ Q$$

Il est évident que ce séquent est prouvable.

Dans le deuxième cas, les choses sont un peu plus compliquées. En appliquant $\forall\mathcal{R}$, puis $\wedge\mathcal{R}$ on obtient six séquents à prouver, plus précisément deux groupes de 3 séquents exactement symétriques, de la forme (on notera $\bar{P} = (P_1, \ldots, P_n)$, $\bar{Q} = (Q_1, \ldots, Q_n)$ et $\Sigma = \Gamma,\ \bar{P},\ \bar{Q},\ \bar{x}$).

$$\Sigma\ ;\ \bigwedge Cong\ P_i\ Q_i \vdash$$
$$\forall A \forall P'(f(\bar{P}, \bar{x}) \xrightarrow{A} P') \supset \exists Q'.(f(\bar{Q}, \bar{x}) \xrightarrow{A} Q') \wedge congr\ P'\ Q'$$
$$--$$
$$\Sigma\ ;\ \bigwedge Cong\ P_i\ Q_i \vdash$$
$$\forall X \forall P'(f(\bar{P}, \bar{x}) \xrightarrow{\downarrow X} P') \supset \exists Q'.(f(\bar{Q}, \bar{x}) \xrightarrow{\downarrow X} Q') \wedge \forall w.congr\ (P'w)\ (Q'w)$$
$$--$$
$$\Sigma\ ;\ \bigwedge Cong\ P_i\ Q_i \vdash$$
$$\forall X \forall P'(f(\bar{P}, \bar{x}) \xrightarrow{\uparrow X} P') \supset \exists Q'.(f(\bar{Q}, \hat{x}) \xrightarrow{\uparrow X} Q') \wedge \nabla w.congr\ (P'w)\ (Q'w)$$

Considérons le deuxième séquent, par exemple, la preuve étant essentiellement la même dans tous les cas. On applique alors $\forall\mathcal{R}$, puis $\supset\mathcal{R}$

$$\Sigma, A : a, P' : n \to p\,;\, \bigwedge Cong\ P_i\ Q_i, f(\bar{P}, \bar{x}) \stackrel{\downarrow X}{\rightharpoonup} P' \vdash$$
$$\exists Q'. f(\bar{Q}, \bar{x}) \stackrel{A}{\rightharpoonup} Q' \wedge \forall congr\ P'\ Q'$$

On applique à présent *def*$\mathcal{L}$. C'est ici que commence la partie délicate de la preuve. Si on a aucune définition avec laquelle unifier notre transition, la preuve est finie. Sinon, on a un nouveau un séquent pour chaque règle pouvant s'appliquer. On n'a besoin de s'intéresser qu'à un seul de ces séquents, puisqu'on a les mêmes restrictions sur toutes les règles de notre format. On applique donc une règle de cette forme, avec éventuellement des noms quantifiés.

$$f(t_1, \ldots, t_n, \bar{x}) \stackrel{B}{\rightsquigarrow} T \stackrel{\triangle}{=} \bigwedge_{i \in I} T_i \stackrel{B_i}{\rightsquigarrow} y_i$$

L'application de *def*$\mathcal{L}$ nous donne donc une substitution θ telle que $\theta(t_i) = P_i$, et à prouver un séquent de la forme :

$$\Sigma, \theta(B) : a, \theta(T) : n \to p\,;\, \bigwedge Cong\ P_i\ Q_i,\ \bigwedge_{i \in I} \theta(T_i) \stackrel{\theta(B_i)}{\rightsquigarrow} \theta(y_i) \vdash$$
$$\exists Q'. f(\bar{Q}, \bar{x}) \stackrel{\theta(B)}{\rightharpoonup} Q' \wedge \forall congr\ (\theta(T)w)(\ Q'w)$$

Nous allons maintenant chercher à construire une substitution θ' qui nous permette d'instancier Q' avec $\theta'(T)$. Notons Z l'ensemble de toutes les variables libres de notre règle. Pour chaque variable z, on note $rg(z)$ la longueur du plus grand arc du graphe de dépendance partant de cette variable. On définit ainsi une partition de Z en $Z_i = \{z / rg(z) = i\}$, puisqu'on a pas de dépendances circulaires. On peut remarquer immédiatement que vu le format des règles, seul les y_i seront de rang strictement positif.

On va tout d'abord définir θ' pour l'ensemble des variables de rang nul.

- Pour les t_i, on pose $\theta'(t_i) = Q_i$
- Pour les variables de nom et les variables libres de la règle, on définit $\theta'(z) = \theta(z)$

Pour le reste des variables, on va définir θ par induction sur le rang i de la variable, en préservant l'invariant suivant : pour toutes les variables de rang inférieur à i, $(\theta(z), \theta'(z)$ doivent être dans l'ensemble de Congr du type approprié. Si z est de type p, on n'a pas d'ambiguïté. Si z est de type $n \to p$, les restrictions énoncées s'appliquent : on se contentera de $Congr_{nabarrow}$ sauf si z apparait en conclusion d'une entrée liée.

Il est clair que θ' vérifie cette propriété au rang 0. La seule ambigüité est dans le cas des t_i. Cependant, si ces variables apparaissent en conclusion d'une entrée liée, alors f a été annotée avec le type $n \stackrel{\forall}{\to} p$, et on aura bien la restriction nécessaire.

Avant de construire θ', on a besoin du lemme suivant.

Lemme 8. *Soit T un terme de type p. Si $\forall x \in fv(T)$, $Cong\theta(x)\ \theta'(x)$ est prouvable, alors $(\theta(T), \theta'(T)) \in Congr_t$.*

Démonstration. Par induction structurelle sur T : si T est une variable, le résultat est évident. Soit T un terme de type p, $T = f(T_1, \ldots, T_n)$. En utilisant l'hypothèse d'induction, on a des preuves de $(\theta(T_i), \theta'(T_i)) \in Congr_{t_i}$. *via* une application de $defR$, ces preuves suffisent à construire une preuve de $congr(\theta(T), \theta'(T))$ ce qui est exactement $(\theta(T), \theta'(T)) \in Congr_p$. □

Soit $z \in Z_{i+1}$ z est nécessairement un y_i et il y a donc une transition de la forme $P_i \overset{A_i}{\rightsquigarrow} z$, où $\forall z' \in fv(P_i),\ rg(z') \leq i$. D'après le lemme 8 et l'hypothèse d'induction, $congr\ \theta(T_i)\ \theta'(T_i)$ est prouvable, et on a donc une preuve de $\theta T_i \overset{A}{\rightarrow} \theta(z) iimp \exists T'.\theta'(T_i) \overset{A}{\rightarrow} T' \wedge CongTT'$. On peut alors appliquer $\supset\mathcal{L}$ et poser $\theta'(z) = T'$, et obtenir notre résultat. On peut ainsi définir intégralement θ'.

Il est évident que θ' vérifie les propriétés suivantes.

1. $\theta'(f(t_1, \ldots, t_n, \bar{x}) = f(\bar{Q}, \bar{x})$
2. $\theta'(B) = \theta(B)$
3. Pour toute variable libre z de type t figurant dans notre règle, $(\theta(z), \theta'(z)) \in Congr_t$

Si l'on revient au séquent que l'on voulait prouver :

$$\Sigma, \theta(B) : a, \theta(T) : n \rightarrow p\,;\, \bigwedge Cong\ P_i\ Q_i,\ \bigwedge_{i\in I} \theta(T_i) \overset{\theta(B_i)}{\rightsquigarrow} \theta(y_i) \vdash \exists Q'.f(\bar{Q}, \bar{x}) \overset{\theta(B)}{\rightharpoonup} Q' \wedge \forall w congr\ (\theta(T)w)(\ Q'w)$$

On va maintenant appliquer $\exists\mathcal{R}$, en instanciant Q' par $\theta'(T)$, puis $\wedge\mathcal{R}$, ce qui nous laisse à prouver :

$$\Sigma, \theta(B) : a, \theta(T) : n \rightarrow p, \theta'(T) : n \rightarrow p\,;\, \bigwedge Cong\ P_i\ Q_i,\ \bigwedge_{i\in I} \theta(T_i) \overset{\theta(B_i)}{\rightsquigarrow} \theta(y_i) \vdash \theta'(f(\bar{t}, \bar{x})) \overset{\theta(B)}{\rightharpoonup} \theta'(T)$$

$$\Sigma, \theta(B) : a, \theta(T) : p, \theta'(T) : p\,;\, \bigwedge Cong\ P_i\ Q_i,\ \bigwedge_{i\in I} \theta(T_i) \overset{\theta(B_i)}{\rightsquigarrow} \theta(y_i) \vdash \forall y congr\ (\theta(T)\ y)(\theta'(T)\ y)$$

Pour prouver le premier séquent, on peut maintenant appliquer $defR$ avec la même règle que précedemment. Avec des applications successives de $\wedge\mathcal{R}$, puis de $\wedge\mathcal{L}$ pour chaque séquent produit, on a à prouver :

$$\Sigma, \theta(B) : a, \theta(T) : n \to p, \theta'(T) : n \to p \,;\, \bigwedge Cong\ P_i\ Q_i,\ \theta(T_i) \overset{\theta(B_i)}{\rightsquigarrow} \theta(y_i) \vdash \theta'(T_i) \overset{\theta'(B_i)}{\rightsquigarrow} \theta'(y_i)$$

Et ce séquent est prouvable par construction de θ'. Quant au deuxième séquent, on peut appliquer le lemme 8 à $(\theta(T)y)$ et $(\theta'(T)y$, après avoir appliqué $\forall\mathcal{L}$.

On a donné ici une preuve du lemme 7.

Soient $(P, Q) \in Congr_p$. Grâce à une application de $\nu\mathcal{L}$, on peut obtenir :

$$\frac{\Sigma \,;\, \Gamma \vdash \bar{y} \triangleright congr\ P\ Q \quad P : p, Q : p \,;\, congr\ P\ Q \vdash B\ congr\ P\ Q}{\Sigma \,;\, \Gamma \vdash \bar{y} \triangleright obisim\ P\ Q}\ \nu\mathcal{L}$$

Le lemme 7 nous assure de l'existence d'une preuve de $P : p, Q : p \,;\, congr\ P\ Q \vdash B\ congr\ P\ Q$, alors que $(P,\ Q) \in Congr_p$, nous fournit une preuve de $\Sigma \,;\, \Gamma \vdash \bar{y} \triangleright congr\ P\ Q$. On a donc une preuve de $\Sigma \,;\, \Gamma \vdash \bar{y} \triangleright obisim\ P\ Q$ et $(P, Q) \in Obisim_p$. On a donc $Congr_p \subseteq Obisim_p$. Comme il est par ailleurs clair que $Obisim_p \subseteq Congr_p$, on a $Obisim_p = Congr_p$. On peut tenir un raisonnement similaire pour les autres composantes de $Obisim$. On peut enfin prouver le résultat suivant, qui constitue le théorème principal :

Theorème 9. *La relation Obisim ainsi définie est bien une congruence au sens de la définition 3*

Démonstration. On doit prouver les trois conditions :

(Equ) : $Obisim_n$ est bien sûr une relation d'équivalence. Au vu de la définition de *obisim*, la réfléxivité et la symétrie sont immédiates dans les autres cas. Soient M, N, O tels que $\forall y\ obisim\ My\ Ny$ et $\forall y\ obisim\ Ny\ Oy$ soient prouvables. En décomposant les deux conjonctions, on obtiendra des preuves de séquents du type

$$y : n, A : a, P' : p, Q' : p \,;\, My \xrightarrow{A} P' \vdash Ny \xrightarrow{A} Q'$$
$$y : n, A : a, Q' : p, T' : p \,;\, Ny \xrightarrow{A} Q' \vdash Oy \xrightarrow{A} T'$$

on peut donc obtenir une preuve de

$$y : n, A : a, P' : p, T' : p \,;\, My \xrightarrow{A} P' \vdash Oy \xrightarrow{A} T'$$

, et sur ce modèle, une preuve de la transitivité de nos relations.

λ Soient P et Q tels que $\Gamma \blacktriangleright (P : p, Q : p) \in Obisim_p$. Choisissons un x dans Γ.

On cherche à prouver que $(\Gamma - x) \blacktriangleright (\lambda x.P, \lambda x.Q) \in Obisim_{n \xrightarrow{\forall} p}$, or $\forall \Gamma obisim\ P\ Q \supset \forall y \forall \Gamma - xobisim\ \lambda x.P\ \lambda x.Q$ est évidemment vrai, on a donc une preuve de $\forall y \forall \Gamma - xobisim\ \lambda x.P\ \lambda x.Q$, ce qui est le résultat recherché.

($Cons$) $Congr$ respecte cette propriété. Or on a montré $Obisim = Congr$. □

2.4 Conclusion, application et travail futur

Applications et Résultats

La première application est de vérifier qu'on peut bien retrouver la propriété que la bisimulation ouverte est une congruence pour le π-calcul. On donnera ici les deux sémantiques :

Si on reformule les règles de la figure 2.2 pour encoder le système de transition "early" du π-calcul ([10]), c'est-à-dire rajouter la règle

$$(in - e): \quad in\ X\ M \overset{\downarrow}{\rightarrow}{}^{X}{}^{u}\ M(u) \overset{\triangle}{=} \top$$

et changer les deux règles (com) qui deviennent :

$$P \mid Q \xrightarrow{\tau} P' \mid Q' \overset{\triangle}{=} \exists X\ Y.P \overset{\uparrow XY}{\rightarrow} P' \wedge Q \overset{\downarrow XY}{\rightarrow} Q' \quad (com - e)$$
$$P \mid Q \xrightarrow{\tau} P' \mid Q' \overset{\triangle}{=} \exists X\ Y.P \overset{\downarrow XY}{\rightarrow} P' \wedge Q \overset{\uparrow XY}{\rightarrow} Q' \quad (com - e)$$

ainsi que la règle ($!com$) qui devient

$$!P \xrightarrow{\tau} (P' \mid Q') \overset{\triangle}{=} P \overset{\uparrow xy}{\rightarrow} P' \wedge P \overset{\downarrow xy}{\rightarrow} Q' \quad (!com - e)$$

la règle $open$ devient

$$\nu y.Py \overset{X}{\multimap} Q \overset{\triangle}{=} \nabla y(Py \overset{Xy}{\rightarrow} Qy). \quad (open)$$

Notons qu'on conserve la règle (in) originale, ainsi que les différentes instances de ($close$), pour coder l'extrusion de portée (?). On donne les types suivants à nos constructeurs :$in : n \rightarrow n \xrightarrow{\forall} p \rightarrow p$ et $\nu : n \rightarrow n \xrightarrow{\nabla} p \rightarrow p$. Toutes les règles sont en format tyft/tyxt sans dépendance circulaire. Il est immédiat que la contrainte sur la quantification des types est respectées dans le cas de la sémantique "late". Il n'est pas très difficile de prouver que c'est également le cas pour la sémantique "early" : en effet, la règle ($open$) ne peut produire une entrée liée que si le processus restreint contient un préfixe in,

$\tau\ P \xrightarrow{\tau} P \triangleq \top.\ (\tau)$
$in\ X\ M \xrightarrow{\downarrow X} M \triangleq \top.\ (in)$
$out\ x\ y\ P \xrightarrow{\uparrow xy} P' \triangleq \top.\ (out)$
$match\ x\ x\ P \xrightarrow{A} Q \triangleq P \xrightarrow{A} Q.\ (match)$
$match\ x\ x\ P \xrightarrow{A} Q \triangleq P \xrightarrow{A} Q.\ (match)$
$P + Q \xrightarrow{A} R \triangleq P \xrightarrow{A} R.\ (plus)$
$P + Q \xrightarrow{A} R \triangleq Q \xrightarrow{A} R.\ (plus)$
$P + Q \xrightarrow{A} R \triangleq P \xrightarrow{A} R.\ (plus)$
$P + Q \xrightarrow{A} R \triangleq Q \xrightarrow{A} R.\ (plus)$
$P \mid Q \xrightarrow{A} P' \mid Q \triangleq P \xrightarrow{A} P'.\ (par)$
$P \mid Q \xrightarrow{A} P \mid Q' \triangleq Q \xrightarrow{A} Q'\ (par)$
$P \mid Q \xrightarrow{A} \lambda n(M\,n \mid Q) \triangleq P \xrightarrow{A} M.\ (par)$
$P \mid Q \xrightarrow{A} \lambda n(P \mid N\,n) \triangleq Q \xrightarrow{A} N.\ (par)$
$\nu n.Pn \xrightarrow{A} \nu n.Qn \triangleq \nabla n(Pn \xrightarrow{A} Qn).\ (res)$
$\nu n.Pn \xrightarrow{A} \lambda m \nu n.P'nm \triangleq \nabla n(Pn \xrightarrow{A} P'n).\ (res)$
$\nu y.Py \xrightarrow{\uparrow X} Q \triangleq \nabla y(Py \xrightarrow{\uparrow Xy} Qy).\ (open)$
$P \mid Q \xrightarrow{\tau} \nu y.My \mid Ny \triangleq \exists X.P \xrightarrow{\downarrow X} M \wedge Q \xrightarrow{\uparrow X} T\ (close)$
$P \mid Q \xrightarrow{\tau} \nu y.My \mid Ny \triangleq \exists X.P \xrightarrow{\uparrow X} M \wedge Q \xrightarrow{\downarrow X} T.\ (close)$
$P \mid Q \xrightarrow{\tau} MY \mid Q' \triangleq \exists X.P \xrightarrow{\downarrow X} M \wedge Q \xrightarrow{\uparrow XY} Q'\ (com)$
$P \mid Q \xrightarrow{\tau} P' \mid NY \triangleq \exists X.P \xrightarrow{\uparrow XY} P' \wedge Q \xrightarrow{\downarrow X} N\ (com)$
$!P \xrightarrow{A} P' \mid !P \triangleq P \xrightarrow{A} P'\ (!act)$
$!P \xrightarrow{X} \lambda x P' x \mid !P \triangleq P \xrightarrow{X} P'\ (!act)$
$!P \xrightarrow{\tau} (P' \mid My) \triangleq P \xrightarrow{\uparrow xy} P' \wedge P \xrightarrow{\downarrow x} M\ (!com)$
$!P \xrightarrow{\tau} \nu y.(Ny \mid My) \triangleq P \xrightarrow{\uparrow x} M \wedge P \xrightarrow{\downarrow x} N\ (!close)$

FIG. 2.2 – Définition des prédicats de transition pour le π-calcul "late"

et donc l'abstraction que l'on transmet sera bien de type $n \xrightarrow{\forall} p$. On peut retrouver le résultat selon lequel la bisimulation ouverte est une congruence pour le π-calcul, dans les deux sémantiques. En fait, un peu de réflexion suffit à s'apercevoir que la spécification de la bisimulation ouverte supprime toute différence entre les deux sémantiques, et que c'est donc la même relation.

Dans le même esprit, on peut très facilement coder différentes extensions du π-calcul en s'assurant qu'elles ne "cassent" pas la congruence : il est trivial d'ajouter un opérateur $; : p \to p \to p$ pour encoder la séquentialité avec les règles :

$$0; P \xrightarrow{A} P' \triangleq P \xrightarrow{A} P' \quad (seq) \quad 0; P \xrightarrow{X} M \triangleq P \xrightarrow{X} M$$

Enfin, on peut encoder le π-calcul asynchrone (voir par exemple [11]) dans ce format, sans difficulté particulière : notre algèbre de terme change légèrement : *out* n'est plus de type $n \to n \to p \to p$, mais de type $n \to n \to p$. On considérera une version très simple du calcul, afin de rester dans le cadre de ([11]) où les autres opérateurs sont *in* , ν., $|$, et !, ainsi que 0. On garde les mêmes règles que précedemment, dans la sémantique early, en supprimant celles qui sont devenues inutiles. La bisimulation ouverte est ici aussi une congruence, et coïncide par ailleurs pour ce sous-cacul avec la bisimulation forte, et la "ground bisimulation" (voir [12]).

L'encodage du π-calcul polyadique est assez immédiat en utilisant la présentation avec abstraction et concrétion de Milner ([13]). Nous nous interromprons ici dans ce catalogue de variantes du π-calcul. On peut simplement ajouter qu'il est extrèmement simple dans notre présentation d'ajouter des types sur les noms, comme dans [13], sans changer le reste de la sémantique.

Si les principales applications *telles quelles* de ce format sont effectivement autour du π-calcul et de ses dérivés, on peut imaginer d'autres applications, en adaptant éventuellement quelque peu le cadre. En particulier, il peut être très intéressant de modéliser des calculs comme le calcul avec fusion de Joachim Parrow et Björn Victor ([14]). Comme mentionné plus haut, il faudra alors probablement changer la spécification de la bisimulation pour la rendre symétrique, et donc changer légèrement la restriction sur les types. En pratique, le type $n \xrightarrow{\nabla} p$ disparaitra totalement, ce qui est logique puisque tout nom peut-être fusionné dans le calcul, mais aucune difficulté majeure ne se présente. En revanche, il n'est pas clair de savoir quelle bisimulation notre prédicat représente dans ce cadre.

Travail futur

L'extension principale que l'on pourrait donner à ce travail serait bien sûr de l'étendre à un véritable ordre supérieur, du type de CHOCS, ou du π-calcul

d'ordre supérieur. C'était le but originel de ce travail, mais aucun résultat n'a malheureusement été obtenu pour l'instant, les preuves étant beaucoup plus difficiles dans ce cas. Peut-être qu'une approche plus sémantique du problème, par exemple en généralisant le travail ([15]) sur les sémantiques bialgébriques, éviterait certains de ces problèmes. Notons qu'il est probable que des restrictions beaucoup plus strictes soient nécessaires dans ce cas, et que dans beaucoup de cas intéressants en théorie du calcul distribué, la mobilité suffit (voir par exemple [11] sur l'expressivité relative du π-calcul asynchrone avec ou sans ordre supérieur). Par ailleurs, il pourrait être intéressant, quitte à perdre de la généralité dans la spécification du système de transition lui-même, d'obtenir également une certaine modularité pour la définition de la bisimulation, afin de pouvoir utiliser plusieurs types de bisimulation.

Bibliographie

[1] Dale Miller and Alwen Tiu. A proof search specification for the pi-calculus. In *Proceedings of FGUC 2004*, 2004.

[2] Dale Miller and Alwen Tiu. A proof theory for generic judgments : An extended abstract. In *Proceedings of LICS 2003*, pages 118–127, June 2003.

[3] Dale Miller and Catuscia Palamidessi. Foundational aspects of syntax. *ACM Computing Surveys Symposium on Theoretical Computer Science : A Perspective*, 1999.

[4] Robin Milner. *Communicating and Mobile Systems : The π-calculus*. Cambridge University Press, 1999.

[5] Alwen Tiu. *A Logical Framework for Reasonning about Logical Specifications*. PhD thesis, Pennsylvania State University, may 2004.

[6] Jan Friso Groote and Frits Vaandrager. Structured operational semantics and bisimulation as a congruence. *Information and Computation*, pages 202–260, 1992.

[7] S. Istrail B. Bloom and A.R. Meyer.

[8] John C. Mitchell. *Type Systems for Programming Languages*, volume B, chapter 8, pages 367–458. MIT Press, 1990.

[9] G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, University of Aarhus, 1981.

[10] Joachim Parrow. An introduction to the pi-calculus.

[11] Davide Sangiorgi. Asynchronous process calculi : the first-order and higher-order paradigms (tutorial). *Theoretical Computer Science*, 253(2) :311–350, 2001.

[12] Davide Sangiorgi. Lazy functions and mobile processes. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction : Essays in Honour of Robin Milner*. MIT Press, 2000.

[13] R. Milner. The polyadic pi-calculus : a tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, pages 203–246. Springer-Verlag, 1993.

[14] Joachim Parrow and Bjorn Victor. The fusion calculus : Expressiveness and symmetry in mobile processes. In *Logic in Computer Science*, pages 176–185, 1998.

[15] Daniele Turi and Gordon D. Plotkin. Towards a mathematical operational semantics. In *Proceedings 12th Ann. IEEE Symp. on Logic in Computer Science, LICS'97, Warsaw, Poland, 29 June – 2 July 1997*, pages 280–291. IEEE Computer Society Press, Los Alamitos, CA, 1997.