Logique, Réalisabilité et Concurrence

Emmanuel Beffara

Université Paris 7 – Denis-Diderot
UFR d'Informatique

Thèse
pour l'obtention du diplôme de
Docteur de l'Université Paris 7, spécialité informatique

Logique, Réalisabilité et Concurrence

présentée et soutenue publiquement par

Emmanuel Beffara

le 6 décembre 2005

devant le jury composé de

M. Vincent	Danos	directeur de thèse
M. Thomas	Ehrhard	
M. Marcelo	Fiore	rapporteur
M. Kohei	Honda	
M. Martin	Hyland	rapporteur
M. Jean-Louis	Krivine	

Merci

Merci à Vincent Danos qui, tout en me laissant toujours une grande liberté, a beaucoup influencé mon travail et ma vision de la science.

Merci à Marcelo Fiore et à Martin Hyland d'avoir accepté d'être les rapporteurs de cette thèse.

Merci à Thomas Ehrhard, Marcelo Fiore, Kohei Honda, Martin Hyland et Jean-Louis Krivine d'avoir accepté de faire partie du jury. Je suis extrèmement honoré de la présence de chacun d'entre eux.

Merci à mon coauteur François Maurel, dont l'influence sur cette thèse ne se limite pas à sa participation au chapitre 8.

Merci à tous les membres de l'équipe PPS, et notamment à Juliusz Chroboczek, Jean-Louis Krivine, Olivier Laurent, Paul-André Melliès et Alexandre Miquel qui, au cours de nombreuses discussions sur la science et le reste, ont beaucoup contribué à mon édification.

Merci à ceux d'ailleurs qui m'ont incité à leur raconter l'avancement de mes travaux, parmi lesquels Laurent Regnier et Daniel Hirschkoff, et merci à Jade Alglave dont le stage a contribué à me mettre les idées en place.

Merci aux joyeux thésards de PPS et du Liafa qui, par leurs discussions pas toujours scientifiques, sont largement responsables de la bonne ambiance qui règne à Chevaleret.

Merci à mes amis et parents pour leur soutien et leur encouragement, et mes excuses pour n'avoir pas toujours su leur expliquer mes travaux.

Et merci à ceux qui se sentiraient vexés de ne pas avoir été remerciés ici.

Table des matières

Notations

Réalisabilité classique

notation	*ensemble*	*entité*	*définition*
x, y, z		λ-variables	2.1 (p. 14)
t, u, v	Λ	λ-termes	〃
ρ, ρ_i		constantes de pile	〃
π, π_i	Π	piles	〃
$t * \pi$	$\Lambda \times \Pi$	exécutables	〃
$\perp\!\!\!\perp$		observation, orthogonalité	2.4 (p. 16)
$[A]$	$\mathcal{P}(\Pi)$	valuation	2.6 (p. 17)
$\llbracket A \rrbracket$	$\mathcal{P}(\Lambda)$	interprétation, synonyme de $[A]^{\perp\!\!\!\perp}$	〃
$t \Vdash A$		t réalise A, synonyme de $t \in \llbracket A \rrbracket$	〃

Ensembles et variables

variable	*ensemble*	*entité*	*définition*
ε	$\mathbf{P}$	polarité	4.1 (p. 52)
u, v, x	$\mathbf{N}$	nom (de canal)	4.2 (p. 52)
α, β		action élémentaire	〃
S, T	Σ	branchement / processus séquentiel	〃
P, Q	Π	processus / terme du $\vec{\pi}$-calcul	〃
e		étiquette de transition	4.10 (p. 56)
ℓ, m	$\mathbf{L}$	localité	8.1 (p. 139)
π, ρ	Υ	garde	〃
s, t		type de canal	4.19 (p. 62)
$\mathcal{I}, \mathcal{J}, \mathcal{K}$		interface	〃
$\mathcal{A}, \mathcal{B}$	$\mathbf{C}_{\mathcal{I}}$	comportement	5.9 (p. 76)
p, q		processus anonyme	4.29 (p. 66)
I, J, K		sorte / interface anonyme	〃
A, B	$\mathbf{C}_I$	type / comportement anonyme	5.9 (p. 76)
Γ, Δ		séquent / type de processus	5.1 (p. 72)

Syntaxe des processus

construction	*signification*	*définition*
$\alpha.P$	préfixe	4.2 (p. 52)
$!S$	réplication	〃
$S + T$	choix externe	〃
$P \mid Q$	composition parallèle	〃
$(\boldsymbol{\nu} x)P$	restriction d'un nom	〃
$(\boldsymbol{\nu}\mathcal{I})P$	restriction d'une interface	4.23 (p. 64)
$u^{\varepsilon}\langle\vec{x}\rangle.P$	action non liante	4.28 (p. 66)
$u^{\varepsilon}(\vec{x})P$	action liante asynchrone	〃
$(\vec{x})P$	abstraction	4.29 (p. 66)
$p[\vec{x}]$	localisation	〃
$\ell : \alpha$	définition de localité	8.1 (p. 139)
$\langle\pi\rangle P$	garde	〃
$(\boldsymbol{\nu}\ell)P$	restriction d'une localité	〃
$\ell : P$	définition généralisée de localité	8.34 (p. 157)

Logique linéaire, types de processus

notation	*logique / comportement*	*définition*
$A^{\perp}$	négation / orthogonal	5.7 (p. 75)
$\forall X.F(X)$	quantification universelle / borne inférieure	5.13 (p. 77)
$\exists X.F(X)$	quantification existentielle / borne supérieure	〃
$\mu X.F(X)$	plus petit point fixe	5.52 (p. 93)
$\nu X.F(X)$	plus grand point fixe	〃
$A \otimes B$	tenseur / produit séparant	5.21 (p. 80)
$A \parr B$	"par" / disjonction spatiale	〃
$A \multimap B$	implication linéaire / spatiale	〃
$[\varepsilon]A$	modalité	5.27 (p. 82)
εA	modalité linéaire	5.46 (p. 91)
$!A$	"bien sûr" / réplication gardée	5.54 (p. 95)
$?A$	"pourquoi pas"	〃

Relations

notation	*signification*	*définition*
$P = Q$	égalité syntaxique des termes	
$P \equiv Q$	congruence structurelle	4.3 (p. 53)
$P \cong Q$	bisimilarité (forte)	4.12 (p. 58)
$P \simeq Q$	équivalence observationnelle	5.10 (p. 76)
$P \vdash \Gamma$	P a le type Γ (syntaxiquement)	5.3 (p. 73)
$P \Vdash \Gamma$	P réalise le type Γ (est élément de Γ)	5.32 (p. 83)

Chapitre 1

Introduction

1.1 Logique et calcul

Logique intuitionniste et constructivisme La réflexion sur la signification calculatoire des démonstrations mathématiques remonte essentiellement aux travaux de Brouwer et Heyting sur les fondements des mathématiques dites *constructives* et la logique intuitionniste. En substance, l'approche constructiviste n'accepte que les démonstrations qui permettent de calculer les objets dont elles prouvent l'existence. La principale différence par rapport à l'approche mathématique classique, et donc avec la *logique classique*, se situe dans le refus du tiers-exclu, car c'est ce principe, et la technique de démonstration par contradiction qui en découle, qui produit des preuves non-constructives.

Dans sa forme minimale, le calcul propositionnel intuitionniste construit des formules à partir de propositions atomiques avec l'implication $\rightarrow$ comme unique connecteur. Les règles de construction des preuves sont les suivantes :

$$\begin{array}{ccc} \text{axiome} & \text{introduction} & \text{élimination} \\ \dfrac{}{\Gamma, A \Rightarrow A} & \dfrac{\Gamma, A \Rightarrow B}{\Gamma \Rightarrow A \rightarrow B} & \dfrac{\Gamma \Rightarrow A \rightarrow B \qquad \Gamma \Rightarrow A}{\Gamma \Rightarrow B} \end{array}$$

Ce système de construction des démonstrations, appelé *déduction naturelle*, se caractérise donc par ses règles d'introduction et d'élimination des connecteurs, et il peut s'étendre aux autres connecteurs usuels comme la conjonction $\wedge$ et la disjonction $\vee$. Comme le nom des règles le suggère, il y a une interaction entre ces deux règles de déduction élémentaires. En effet, considérons le cas d'une démonstration qui se termine par une introduction suivie d'une élimination :

$$\dfrac{\dfrac{\begin{array}{c}\vdots\\ \Gamma, A \Rightarrow B\end{array}}{\Gamma \Rightarrow A \rightarrow B}\ \text{intro} \qquad \begin{array}{c}\vdots\\ \Gamma \Rightarrow A\end{array}}{\Gamma \Rightarrow B}\ \text{élim}$$

Dans ce cas il est possible de transformer cette dérivation en une démonstration directe de $\Gamma \Rightarrow B$, en remplaçant chaque utilisation d'un axiome $A \Rightarrow A$ par la démonstration de $\Gamma \Rightarrow A$ dans l'arbre de preuve de conclusion $\Gamma, A \Rightarrow B$.

La situation ci-dessus, où l'on déduit $\Gamma \Rightarrow B$ à partir d'une démonstration de $\Gamma, A \Rightarrow B$ qui utilise une hypothèse A supplémentaire et d'une démonstration de $\Gamma \Rightarrow A$ qui valide cette hypothèse, est appelée *coupure*, car elle coupe une démonstration en deux parties complémentaires. Dans un système déductif légèrement différent, appelé *calcul des séquents*, on peut la présenter comme une règle à part entière :

$$\frac{\Gamma, A \Rightarrow B \qquad \Delta \Rightarrow A}{\Gamma, \Delta \Rightarrow B}$$

de sorte que les seules autres règles sont l'axiome et l'introduction des connecteurs à gauche et à droite du symbole $\Rightarrow$. La situation précédente, où l'on élimine un schéma d'introduction/élimination d'un connecteur, devient maintenant une procédure systématique d'*élimination des coupures*, grâce à laquelle on démontre que la règle de coupure est redondante en calcul des séquents.

L'élimination des coupures est un résultat fondamental de la théorie de la démonstration. Une conséquence est la propriété dite de la sous-formule, selon laquelle un séquent $\Gamma \Rightarrow \Delta$ est démontrable si et seulement s'il en existe une démonstration où n'interviennent que des sous-formules de Γ et Δ. Dans le cas de la logique intuitionniste, elle a aussi pour conséquence la *propriété de la disjonction* qui montre que, de toute démonstration d'une formule $A \vee B$, il est possible d'extraire soit une démonstration de A soit une démonstration de B. Cette propriété est la clé de la constructivité, et elle n'est pas satisfaite par la logique classique. En effet, à partir d'une démonstration du tiers-exclu $A \vee \neg A$, on ne peut en général extraire ni une démonstration de A ni une démonstration de $\neg A$.

Correspondance de Curry-Howard Si la logique intuitionniste parle de calcul et de construction d'objets, elle se pose au départ principalement comme une prise de position philosophique, et elle ne fait pas directement référence à un modèle de calcul pour formaliser ses principes. Le lien précis avec la calculabilité est établi au moyen du λ-calcul, introduit par Church comme un formalisation générique du calcul fonctionnel. Sous sa forme la plus simple, la syntaxe de ce calcul s'écrit

$$t, u ::= x \mid t(u) \mid \lambda x.t$$

pour représenter respectivement une variable x, l'application d'une fonction t à un argument u, et la construction d'une fonction $x \mapsto t$ par abstraction d'une variable x dans une expression t. On en fait un modèle de calcul en définissant la règle d'évaluation fondamentale suivante, dite de β-réduction :

$$(\lambda x.t)(u) \longrightarrow t[u/x]$$

Cette formulation du calcul, si elle se fonde sur la notion de fonction et fournit une méthode *syntaxique* de calcul, ne fait pas référence au domaine de définition des fonctions, et permet en effet d'écrire des termes en apparence contradictoires, comme par exemple $\delta = \lambda x.x(x)$, qui est un fonction qui applique son argument à lui-même. Il arrive même que le processus de calcul par substitution ne termine pas, comme dans le terme $\delta\delta$, qui se réduit indéfiniment en lui-même. C'est pour éviter ce genre de cas et retrouver une notion de fonction plus intuitive qu'apparaît le typage, qui décrit l'espace fonctionnel dans lequel se trouve un

terme. Si l'on note $A \to B$ l'ensemble des fonctions d'un ensemble A vers un ensemble B, on obtient les règles de *typage* suivantes :

$$\frac{}{\Gamma, x \in A \Rightarrow x \in A} \qquad \frac{\Gamma, x \in A \Rightarrow t \in B}{\Gamma \Rightarrow \lambda x.t \in A \to B} \qquad \frac{\Gamma \Rightarrow t \in A \to B \qquad \Gamma \Rightarrow u \in A}{\Gamma \Rightarrow t(u) \in B}$$

On remarque immédiatement que ces règles, si l'on oublie les termes, sont celles de la logique intuitionniste minimale : c'est la correspondance de Curry-Howard. La coupure consiste alors à construire une fonction f et l'appliquer à une valeur v, et l'élimination de la coupure correspond à l'évaluation directe de la valeur $f(v)$. La coïncidence entre la β-réduction et la formulation logique de l'élimination des coupures traduit alors le fait fondamental que le type d'un terme est conservé lors de l'évaluation, et que cette évaluation termine, ce qui constitue une preuve de la cohérence du calcul.

Sémantiques dénotationnelles Une part importante de l'étude des formalismes mathématiques, qu'il s'agisse de modèles de calcul, de systèmes logiques, ou même de théories algébriques plus traditionnelles, est ce que l'on range sous le nom de sémantique. Ce vocable hérité de la linguistique désigne toute l'approche consistant à distinguer les symboles écrits (la syntaxe) et leur signification (la sémantique), c'est-à-dire les objets mathématiques qu'ils représentent. Cette vision est fondamentale, en particulier par son aspect épistémologique, car elle souligne la nécessité de prise de recul face au côté artificiel des notations et la distance toujours présente entre les phénomènes évoqués et leurs modélisations formelles.

Si la recherche d'une syntaxe appropriée à une sémantique préexistante est la raison d'être de la modélisation mathématique, à l'inverse formuler proprement une sémantique pour un formalisme donné est la meilleure façon de comprendre ce formalisme, et cette remarque est cruciale dans le cas de la logique. Si la logique classique a l'ambition de parler de *vérités*, comme en témoigne sa sémantique la plus naturelle qui est celle des algèbres de Boole, en revanche la logique intuitionniste parle de *causalité*, notion absente en classique, et même de *fonctions*. Ainsi $A \to B$ en logique classique signifie « si A est vrai alors B est vrai, » alors qu'en logique intuitionniste il signifie « de toute démonstration de A on peut déduire une démonstration de B. »

L'histoire de l'étude sémantique de la logique intuitionniste découle de cette idée que l'objet sous-jacent est la fonction et non la valeur de vérité. Ainsi, parmi les modèles dénotationnels usuels de la logique intuitionniste, on trouve les domaines de Scott, qui sont un modèle ensembliste simple et un outil standard dans la modélisation du λ-calcul simplement typé et des langages de programmation fonctionnels. Car une fois établi l'isomorphisme entre un modèle de calcul et un système logique, il est naturel de donner une sémantique commune aux deux formalismes, dans le but de comprendre l'un sous l'éclairage de l'autre.

C'est dans le contexte de l'informatique théorique, dont le but est la modélisation et la compréhension des modèles de calcul et de programmation, que la sémantique s'est le plus développée. En particulier, on y distingue plusieurs formes de sémantiques, comme plusieurs façons d'exprimer la signification des objets. D'une part, la sémantique *opérationnelle* est celle qui formalise la dynamique des systèmes, en décrivant quelles sont leurs étapes de calcul, comme le fait la β-réduction dans le λ-calcul. À l'inverse, la sémantique *dénotationnelle*

formalise la nature des objets, au-delà de leurs règles de calcul, en associant à une représentation syntaxique l'objet mathématique qu'elle dénote, par exemple des fonctions dans le cas des termes du λ-calcul, ou des stratégies d'interaction dans le cas de modèles de calcul plus concurrents.

Logique linéaire De l'étude sémantique de formes plus riches de la logique intuitionniste, comme le système F, et dans la suite des idées des domaines de Scott est né le formalisme voisin des espaces de cohérence, dont l'étude a conduit au pas décisif que constitue l'introduction par Girard de la logique linéaire [35]. Cette logique apparaît comme une décomposition de la logique intuitionniste dans laquelle les règles structurelles de contraction et d'affaiblissement deviennent explicitement et finement contrôlées, ce qui fait apparaître une notion importante de linéarité. Ce changement de point de vue est analogue aux rapports entre l'analyse fonctionnelle et l'algèbre linéaire, qui concerne des notions de plus bas niveau.

Avec la logique linéaire s'est développée une intuition nouvelle sur la sémantique de la logique intuitionniste. Le principe est de considérer une démonstration non plus comme une fonction d'un domaine vers un autre, mais comme un moyen de réfuter des contre-preuves potentielles. C'est une vision plus interactive et calculatoire que purement fonctionnelle, qui a mené à la définition des sémantiques de jeux [3, 46], où le rôle de fonction est joué par des *stratégies* qui se définissent comme des ensemble de dialogues possibles entre un *joueur* et un *adversaire*. Dans ce cadre, le calcul ne se conçoit plus comme l'évaluation explicite de la valeur d'un objet, mais comme la description de sa façon de se comporter face à d'autres objets.

Un effet intéressant de ce changement de point de vue est qu'il rapproche les visions antagonistes de sémantique dénotationnelle et opérationnelle, et par suite la sémantique de la syntaxe, alors que toute la démarche de la sémantique a été précisément de séparer ces deux notions afin de comprendre leurs rapports. C'est dans cet esprit que Girard développe la ludique [39] comme un fondement purement interactif de la logique et qui ne se veut ni syntaxe ni sémantique.

Logique classique L'étude calculatoire de la logique et la correspondance de Curry-Howard se sont développées sous le postulat que la constructivité était propre à la logique intuitionniste et que la logique classique n'avait pas de sens calculatoire. Cette idée vient notamment du fait que l'élimination des coupures en logique classique, notamment dans le système LK de Gentzen, n'est pas déterministe. En particulier, toutes les démonstrations d'une même formule sont convertibles par introduction et élimination de coupures. Il semble donc que les démonstrations classiques n'aient pas d'autre contenu sémantique que la véracité de leur conclusion.

C'est du monde de la programmation qu'est venue la réfutation de cette croyance : en 1990, Griffin [41] découvre que des opérateurs de contrôle connus en programmation fonctionnelle peuvent être correctement typés au moyen de formules de logique classique fausses en logique intuitionniste. À partir de cette remarque, plusieurs systèmes logiques classiques constructifs sont développés, dont des extensions avec contrôle du λ-calcul comme le $\lambda\mu$-calcul introduit par Parigot [67]. Le principe de ces différents systèmes est de retrouver le déterminisme en modifiant les règles logiques et l'élimination des coupures.

L'étude de ces systèmes constructifs met au jour des connexions profondes entre la logique classique et les mécanismes de manipulation du flot de contrôle connus en programmation. Ainsi l'étude des traductions de la logique classique dans la logique linéaire, menée par Danos, Joinet et Schellinx [20, 21], donne un statut logique aux stratégies de réduction en appel par nom et par valeur, et elles montrent des liens avec les techniques de programmation par passage de continuation [70]. La dualité entre programme et environnement se retrouve dans le rôle important joué en logique par les polarités, phénomène illustré en logique classique par le système LC de Girard [36] et étudié en profondeur par Laurent dans la variante polarisée de la logique linéaire [58]. C'est dans ce contexte que Krivine développe la réalisabilité classique [50, 51] pour traiter le problème dit de la spécification, c'est-à-dire explorer le sens calculatoire des démonstrations classiques, et en particulier celui des démonstrations en mathématiques usuelles.

1.2 Programmation concurrente

L'idée de représenter l'interaction entre deux entités par l'ensemble de leurs dialogues possibles n'est pas nouvelle, elle est même l'un des fondements de la théorie de la concurrence. Par le mot *concurrence*, on désigne généralement l'ensemble des phénomènes issus de l'interaction entre plusieurs agents au sein d'un système donné. Le besoin de formuler cette interaction apparaît naturellement dans l'étude des systèmes informatiques et des langages de programmation, d'une part quand on considère un programme comme la réunion de plusieurs composants chargés de tâches différentes, et d'autre part dès qu'il s'agit de modéliser des systèmes composés de plusieurs ordinateurs qui communiquent. Il est habituel, pour justifier l'étude de ces systèmes, de prendre l'exemple des réseaux d'ordinateurs qui foisonnent aujourd'hui et sur lesquels évoluent divers services et fonctionnalités susceptibles d'interagir et d'interférer, de façon plus ou moins contrôlable.

Modèles de la concurrence Les mêmes raisons qui poussent à définir la sémantique formelle des langages de programmation séquentiels ou fonctionnels s'appliquent au cas concurrent : le besoin de spécifications précises, de techniques de modélisation et d'analyse par des outils mathématiques est plus que jamais nécessaire pour garantir le bon fonctionnement des systèmes et l'absence d'erreurs.

Les modèles opérationnels trouvent leur origine dans la théorie des automates, qui avait également fourni des modèles pour la calculabilité et la complexité. Si les réseaux de Petri généralisent les automates finis en autorisant plusieurs états simultanés et synchronisés, une approche différente est employée dans le cas des arbres de synchronisation, qui décrivent un processus par les suites de messages qu'il est susceptible d'échanger avec l'extérieur, sans mention explicite d'un état interne et sans prendre en compte la distribution du processus entre plusieurs agents.

La recherche de modèles dénotationnels pour la concurrence est une tâche plus ardue. Une solution largement acceptée est celle des structures d'événements [75], qui donnent des structures mathématiques proprement axiomatisées dans le style de la théorie des domaines pour parler de cohérence et d'exclusion mutuelle entre événements. Plus récemment sont apparus des modèles em-

ployant les techniques des sémantiques de jeux, issus d'une réflexion sur la notion de causalité.

Calculs de processus Les arbres de synchronisation, évoqués plus haut, peuvent se voir comme des ensembles (employons même le mot langage) de traces d'interactions, c'est-à-dire de suites d'opérations de synchronisation élémentaires. La tradition syntaxique des langages réguliers permet de décrire ces objets, et a mené effectivement au pas important que fut CCS [61], première forme de calcul à réduction pour la description de processus. On y décrit un processus par une algèbre de termes assez simple :

$$P, Q ::= \mathbf{0} \mid a.P \mid (P \mid Q) \mid P + Q \mid \mathrm{rec}X.P \mid X$$

pour désigner respectivement l'arrêt, l'échange d'un message (suivi d'un nouvel état), la mise en parallèle de deux agents, le choix entre deux agents, et une forme de récursion qui permet la description de comportements infinis. La règle de calcul est ici la synchronisation de deux agents par l'échange d'un message a et de son opposé $\bar{a}$:

$$a.P \mid \bar{a}.Q \longrightarrow P \mid Q$$

L'interprétation de cette synchronisation comme une communication sur un canal a mène immédiatement à l'extension du langage dans le but de communiquer des valeurs, et CCS s'étend sans douleur à ce genre de communication, en considérant $a.P$ comme une réception et $\bar{a}.Q$ comme une émission. La règle de calcul devient alors quelque chose comme

$$a(x).P \mid \bar{a}\langle v\rangle.Q \longrightarrow P[v/x] \mid Q$$

où la valeur v est envoyée sur le canal a par un agent et attribuée à la variable x par l'autre agent. Ainsi la substitution redevient un mécanisme fondamental du calcul.

Le véritable saut conceptuel se fait lorsqu'un considère a dans la formule ci-dessus comme un véritable canal de communication, reconnu en tant qu'objet de première classe du langage. On peut alors communiquer des noms de canaux de communication, ce qui permet à un processus d'établir de nouvelles connexions au cours de son exécution. On parle alors de *mobilité*, et c'est le fondement du π-calcul [74], qui fait actuellement figure de référence pour la modélisation de systèmes concurrents.

Langages de spécification Les formalismes concurrents évoqués ici sont surtout considérés pour leurs sémantiques opérationnelles, plus naturelles que les formes dénotationnelles mais aussi plus difficiles à employer dans la spécification et la démonstration de validité des programmes, qui sont pourtant les motivations informatiques de la sémantique. Dans le but de gagner en abstraction et en généralité, différentes logiques de spécification ont été développées afin de décrire les systèmes concurrents.

Une forme importante est la logique de Hennessy-Milner [42], dont la caractéristique principale est la présence de modalités temporelles qui permettant d'exprimer des jugements comme « la propriété P sera toujours vérifiée » ou « la condition P sera vraie en un temps fini ». Cette logique a une expressivité suffisante pour capturer tous les comportements observables des processus de

CCS, néanmoins elle n'est pas totalement satisfaisante pour étudier les modèles de la mobilité comme le π-calcul. C'est pourquoi d'autres formalismes ont été développés, comme ceux des logiques spatiales (logique de la séparation, etc.), qui ont été récemment appliqués au cas de calculs de processus [16] en raison de leur capacité à décrire la structure géométrique des objets.

Les formalismes de spécification pour la concurrence, bien qu'ils soient souvent appelés « logiques », ont rarement les aspects caractéristiques des logiques formelles que sont les systèmes d'inférence et l'élimination des coupures. Établir des liens dans la tradition de Curry-Howard entre des modèles du calcul concurrent et des logiques susceptibles de parler de spécification paraît pourtant envisageable et porteur de sens. Ainsi par exemple la *logic of bunched implication* de O'Hearn et Pym [65] est un système logique au sens le plus strict issu de l'étude de logiques de la séparation. Des liens entre de telles logiques et la logique linéaire ont notamment été explorés par Mads Dam [18] dans une variante de CCS.

D'autre part, plusieurs interprétations calculatoires de la logique linéaire en termes concurrents ont été formulées. Les réseaux d'interaction de Lafont [53] sont un modèle de calcul concurrent issu de l'étude de la dynamique d'élimination des coupures dans les réseaux de preuves et plus particulièrement dans le fragment multiplicatif. Des liens avec les calculs de processus ont été étudiés par Abramsky, qui propose un modèle de calcul pour représenter la réduction des réseaux de preuve [1] dans un style similaire à la machine abstraite chimique de Berry et Boudol [12], et qui développe une interprétation de la logique linéaire par réalisabilité dans une variante de CCS.

Du fonctionnel au concurrent L'idée qu'il existe de liens significatifs entre les logiques de spécification des processus concurrents et les systèmes logiques traditionnels est renforcée par le fait que, d'un point de vue calculatoire, des liens entre le calcul fonctionnel et les modèles de la concurrence sont très développés. En particulier, dès l'introduction du π-calcul, Milner montre qu'il est possible d'y coder le λ-calcul [62] et que le comportement des processus obtenus correspond à celui de machines abstraites pour l'évaluation des λ-termes.

Parce qu'il fournit un langage commun où peuvent se coder différents modèles de calcul, le π-calcul s'avère un bon moyen de comparer les caractéristiques de divers calculs fonctionnels. C'est dans cet esprit que Honda, Yoshida et Berger ont développé un codage du $\lambda\mu$-calcul [44] pour étudier la dynamique du contrôle en λ-calcul. De même, Bellin et Scott [10] emploient un codage des réseaux de preuve pour étudier le flot d'information dans les démonstrations en logique linéaire. L'utilisation du π-calcul comme formalisme unificateur entre divers langages est comparable l'utilisation des sémantiques de jeux qui permettent de réunir dans un même cadre formel différents traits de programmation comme le contrôle, l'état ou le non-déterminisme. Des connexions entre les deux ont d'ailleurs été explorées, ainsi Hyland et Ong [45] utilisent le π-calcul comme une syntaxe pour représenter des stratégies et ainsi dériver des traductions de PCF en π-calcul.

1.3 Plan de la thèse

Cette thèse a pour but, en résumé, de mettre en place une notion de réalisabilité classique à la Krivine dans un langage concurrent. En d'autres termes, il s'agit de formuler une notion de constructivité classique qui ne repose pas sur l'idée de fonction mais sur l'idée processus interactif. Cette nouvelle réalisabilité contient son analogue classique en tant que fragment, et elle a plusieurs applications. L'une est d'adapter la technique de spécification aux processus et ainsi créer des combinateurs concurrents à partir des types. Une autre est de reconstruire rationnellement, sur des fondements logiques, les diverses traductions des divers λ-calculs dans les divers π-calculs.

Plan Dans une première partie, on s'attache à l'étude du contenu calculatoire des démonstrations classiques. Dans le chapitre 2, on rappelle le formalisme de la réalisabilité classique dû à Krivine, en on illustre la technique en définissant une extension de cette théorie à un langage de programmation simple avec données et opérateurs de contrôle. Dans le chapitre 3, on applique ce formalisme à l'étude de structures de contrôle typées, en dérivant une description précise du contenu calculatoire des tautologies classiques sous forme normale disjonctive. Les comportements obtenus s'interprètent avantageusement en termes de synchronisation entre processus séquentiels, ce qui motive une étude réellement concurrente des mécanismes sous-jacents.

Dans une seconde partie, on développe donc une technique de réalisabilité analogue pour le calcul concurrent. Dans le chapitre 4, on définit pour cela une variante symétrique du π-calcul, le π-calcul polarisé, et on étudie les propriétés élémentaires de ce calcul et son système de typage. Dans le chapitre 5, on définit à partir de ce calcul une notion de réalisabilité concurrente, inspirée de la réalisabilité classique de Krivine et de la ludique de Girard, d'où l'on dérive une logique des comportements de processus.

La technique développée pour l'étude des processus concurrents est ensuite appliquée à l'étude calculatoire de systèmes logiques existants. Dans le chapitre 6, on revient à la logique classique au moyen de ses plongements dans la logique linéaire dus à Danos, Joinet et Schellinx, d'où l'on dérive des traductions typées de variantes du λ-calcul dans le π-calcul polarisé. On en déduit une reconstruction de la réalisabilité classique sur le calcul concurrent, ce qui permet de donner corps à l'intuition concurrente que suggérait l'étude directe de la logique classique. Le chapitre 7 suggère quelques applications de la technique de réalisabilité concurrente ; d'une part on y esquisse des techniques de spécification pour les processus concurrents typés, d'autre part on l'utilise pour développer des combinateurs concurrents dans un λ-calcul séquentiel.

Dans le chapitre 8, qui est relativement indépendant, on mène une étude de l'ordonnancement dans les calculs concurrents au moyen de techniques inspirées de la logique. On définit un système de réseaux pour représenter les processus, grâce auquel on caractérise différentes variantes de calculs concurrents par des arguments géométriques. On montre ensuite comment le préfixage explicite peut être transformé en causalité dans les communications, par une technique qui rappelle le passage de continuations connu en λ-calcul.

Références Une part importante du contenu de cette thèse est aussi disponible en anglais sous forme de publications ou de notes. La section 2.1 est une présentation d'un cadre développé par Krivine [52], la section 2.2 est un développement de la courte note [5] *Realizability with constants*. Le chapitre 3 est une version étendue de l'article [7] *Disjunctive normal forms and local exceptions* (avec Vincent Danos). L'article [6] *A concurrent model for linear logic* contient une présentation courte du calcul du chapitre 4, une version préliminaire de la théorie exposée dans le chapitre 5, et les traductions exposées en section 6.2.1. Le chapitre 8 est essentiellement une traduction de l'article [8] *Concurrent nets : a study of prefixing in process calculi* (avec François Maurel). Ces références sont toutes accessibles en ligne à l'adresse `http://www.pps.jussieu.fr/~beffara/`.

Chapitre 2

Réalisabilité classique

L'isomorphisme de Curry-Howard a d'abord été développé et bien compris entre la logique intuitionniste et le λ-calcul typé. La logique classique peut être vue comme la logique intuitionniste munie de l'axiome du tiers exclu, ou de toute autre formule intuitionnistiquement équivalente. À ce titre, pour étendre la correspondance au raisonnement classique, il est naturel de se fonder sur le λ-calcul en lui ajoutant des constantes munies de règles de réduction particulières pour représenter ces axiomes supplémentaires.

Le moyen d'étendre le λ-calcul à la logique classique a été initialement trouvé par Griffin [41] qui remarqua qu'il est possible d'attribuer à la primitive de contrôle `call/cc` (pour *call with current continuation*) du langage Scheme le type $((A \to B) \to A) \to A$, c'est-à-dire la loi de Peirce. Cet axiome est équivalent au tiers exclu, c'est-à-dire que la logique intuitionniste augmentée de cet axiome a la même expressivité que la logique classique. L'effet calculatoire de `call/cc` est assez riche car il permet de réifier les continuations, c'est-à-dire qu'il permet au programme de manipuler explicitement son flot de contrôle. Tandis qu'en calcul fonctionnel pur chaque terme a une valeur de retour unique, grâce au contrôle un programme peut disposer de plusieurs sorties, chacune éventuellement utilisée plusieurs fois. Cet enrichissement s'exprime en logique par le fait que les séquents intuitionnistes ont toujours une formule unique à droite du symbole $\vdash$, alors qu'en logique classique plusieurs formules peuvent s'y trouver et se voir appliquer les règles de contraction et d'affaiblissement.

À partir de cette observation fondamentale sur le sens calculatoire du raisonnement classique, plusieurs systèmes logiques classiques constructifs ont été développés. Le $\lambda\mu$-calcul de Parigot [67] (et sa variante à appel par valeur formulée par Ong et Stewart [66]) est une formulation calculatoire de la déduction naturelle classique et de la déduction libre qui rend explicite la manipulation des multiples sorties d'un terme au moyen des termes nommés et μ-abstractions. Le système LC de Girard [36] introduit une notion de formule active au moyen de polarités pour retrouver une élimination des coupures déterministe. L'approche la plus économique, et que nous utiliserons dans ce chapitre, est celle de Krivine [50, 51], qui utilise un λ-calcul muni d'une stratégie de réduction totalement déterministe, auquel s'ajoute une constante cc pour représenter `call/cc`. Il s'agit en fait d'une variante simplifiée du calcul $\lambda\mathcal{C}$ de Felleisen [28].

2.1 Curry-Howard en logique classique

Dans cette section, on rappelle les principes de la réalisabilité classique [52] développée par Krivine pour résoudre le problème dit de la *spécification*, dont l'enjeu est d'extraire le contenu calculatoire des énoncés mathématiques, c'est-à-dire le comportement commun à tous les termes qui représentent des démonstrations d'une formule donnée. Le formalisme a été appliqué à l'arithmétique du second ordre puis à la théorie des ensembles [50, 51] ; l'application à l'arithmétique est étudiée plus en détail dans la section 2.2.

2.1.1 Un λ-calcul avec contrôle

Le modèle de calcul considéré ici est un λ-calcul avec une stratégie de réduction totalement déterministe. Cette réduction est définie explicitement comme l'interaction entre un terme (le programme) et une continuation (son environnement). La stratégie de réduction employée est la réduction de tête faible, qui est une forme d'appel par nom dans laquelle les seuls contextes d'évaluation sont des applications, c'est-à-dire qu'ils ont une forme de pile de termes.

Définition 2.1 (λc-termes)**.** On suppose donnés un ensemble dénombrable de variables (notées $x, y \ldots$) et un ensemble dénombrable de constantes de pile (notées ρ). Les termes, piles et exécutables du λc-calcul sont définis par la grammaire suivante :

termes	$t, u := x \mid t\,u \mid \lambda x.t \mid \mathrm{cc} \mid k_\pi$
piles	$\pi := \rho \mid t \cdot \pi$
exécutables	$e := t * \pi$

On note Λ l'ensemble de termes et Π l'ensemble des piles.

Un exécutable $t * \pi$ est donc la donnée d'un terme t et d'une pile d'arguments π, et l'évaluation est définie uniquement sur de tels couples. Les termes de la forme k_π sont des continuations réifiées, c'est-à-dire qu'ils représentent des piles sauvegardées et prêtes à être restaurées.

Définition 2.2 (réduction)**.** La réduction des exécutables est la relation $\to$ sur $\Lambda \times \Pi$ engendrée par les règles suivantes :

push :	$t\,u * \pi \to t * u \cdot \pi$	save :	$\mathrm{cc} * t \cdot \pi \to t * k_\pi \cdot \pi$
pop :	$\lambda x.t * u \cdot \pi \to t[u/x] * \pi$	restore :	$k_\pi * t \cdot \pi' \to t * \pi$

On note $\to^*$ la clôture réflexive et transitive de $\to$.

Les règles de typage, détaillées table 2.1, sont les règles usuelles du système F (on utilise ici une quantification à la Curry), avec un axiome pour typer la constante cc, qui représente `call/cc`. Notons qu'on ne donne pas de type aux piles ni aux continuations ; il serait possible de les typer, moyennant un typage des constantes de piles, mais ce n'est pas nécessaire pour le propos qui suit.

Le système de types employé est donc la logique propositionnelle minimale, avec les variables propositionnelles comme formules atomiques et avec l'implication $\to$ et la quantification universelle du second ordre pour seuls connecteurs.

Axiomes :

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{}{\Gamma \vdash \mathrm{cc} : ((A \to B) \to A) \to A}$$

Abstraction et application :

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash t\, u : B}$$

Quantification :

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t : \forall X\, A}\ X \notin \mathrm{fv}(\Gamma) \qquad \frac{\Gamma \vdash t : \forall X\, A}{\Gamma \vdash t : A[B/X]}$$

TAB. 2.1 – Règles de typage du λc-calcul.

Plus loin, on enrichira le langage des types avec des symboles de prédicats et une quantification du premier ordre. Ce langage de formules est néanmoins suffisant pour exprimer toute la logique propositionnelle classique, au moyen des notations standard suivantes :

Notation 2.3. Pour tous types A, B, X, on pose

curryfication	$A, B \to X := A \to B \to X$
conjonction	$A \wedge B := \forall X\, (A, B \to X) \to X$
disjonction	$A \vee B := \forall X (A \to X), (B \to X) \to X$
absurde	$\bot := \forall X\, X$
négation	$\neg A := A \to \bot$

Pour la plupart des systèmes de types, une propriété fondamentale est la *subject reduction*, ou conservation du type par réduction. Son intérêt est de permettre de garantir des propriétés sur les termes typés en n'étudiant que les formes normales habitant les types. Ici cette notion n'a pas de sens, car la relation de réduction ne s'applique pas aux termes typés, mais la technique développée dans la suite permet de raisonner par des arguments plus sémantiques.

2.1.2 Réalisabilité classique

La réalisabilité standard à la Kleene construit des modèles de la logique intuitionniste au moyen d'algèbres combinatoires. Étant donné un ensemble M muni d'une opération d'application, chaque formule A est interprétée comme une partie $[\![A]\!]$ de M, comprise comme l'ensemble des témoins de la validité de A. L'application sur M sert alors à définir l'interprétation de l'implication, en posant $[\![A \to B]\!] = \{\, x \in M \mid \forall y \in [\![A]\!], xy \in [\![B]\!] \,\}$. Dans une telle interprétation, la quantification est interprétée naturellement par une intersection : $[\![\forall X\, A]\!] = \bigcap_{X \subseteq M} [\![A]\!]$. L'absurde étant défini comme $\bot = \forall X\, X$, son interprétation est alors l'ensemble vide, puisqu'aucun objet ne peut témoigner de sa véracité. La limite de cette définition, qui fait qu'elle ne s'applique qu'à la logique intuitionniste, réside dans l'interprétation qu'elle induit pour la négation : en prenant la définition $\neg A = A \to \bot$, on obtient $[\![\neg A]\!] = \{\, x \mid \forall y \in [\![A]\!], xy \in \emptyset \,\}$,

donc la formule $\neg A$ est interprétée par l'ensemble vide si A est réalisable, et par M tout entier si A n'est pas réalisable. En d'autres termes, la négation vide les formules de toute signification calculatoire.

L'apport crucial de la logique linéaire, et de sa sémantique de phase [37] pour ce qui nous concerne ici, est le changement d'interprétation pour la négation. Le rôle de formule non réalisable est joué par la constante additive $\mathbf{0}$, alors que la constante multiplicative $\bot$, distincte de $\mathbf{0}$, sert à définir la négation linéaire. La réalisabilité classique développée par Krivine définit de même la négation au moyen d'un objet de référence, avec une différence notable : la négation considérée est une opération hétérogène, les formules et leurs négations sont maintenant des objets d'espèces différentes.

L'algèbre combinatoire considérée est donc l'ensemble Λ des λ-termes, muni de l'application de $\Lambda \times \Lambda$ dans Λ, et les formules sont toujours interprétées comme des parties de Λ. À l'inverse, les négations de formules sont interprétées par des ensembles de piles, c'est-à-dire des parties de Π. Cet ensemble est muni d'une opération duale de l'application, notée $(\cdot) : \Lambda \times \Pi \to \Pi$, qui correspond à ajouter un terme sur une pile. L'objet dualisant, que l'on appellera *observation* dans la suite, est une partie de $\Lambda \times \Pi$.

Définition 2.4 (orthogonalité)**.** On appelle observation toute partie $\perp\!\!\!\perp$ de $\Lambda \times \Pi$ close par anti-réduction (c'est-à-dire que $t * \pi \to^* t' * \pi'$ et $t' * \pi' \in \perp\!\!\!\perp$ entraînent $t * \pi \in \perp\!\!\!\perp$). On note $t \perp\!\!\!\perp \pi$ pour $t * \pi \in \perp\!\!\!\perp$, et pour $A \subseteq \Lambda$ et $B \subseteq \Pi$ on note $A \perp\!\!\!\perp B$ si $t \perp\!\!\!\perp \pi$ pour tous $t \in A$ et $\pi \in B$. L'orthogonal d'un ensemble $A \subseteq \Lambda$ est défini comme $A^{\perp\!\!\!\perp} := \{ \pi \mid \forall t \in A, t * \pi \in \perp\!\!\!\perp \}$, l'orthogonal d'une partie de Π est définie de façon analogue.

Remarquons que le bi-orthogonal est une opération de clôture sur les parties de Λ et de Π, et que les ensembles $A^{\perp\!\!\!\perp}$ sont toujours clos. En d'autres termes, l'orthogonalité vérifie les propriétés suivantes, pour toutes parties X et Y de Λ ou Π :

$$X \subseteq Y \Rightarrow Y^{\perp\!\!\!\perp} \subseteq X^{\perp\!\!\!\perp} \qquad\qquad X \subseteq X^{\perp\!\!\!\perp\perp\!\!\!\perp} \tag{2.1}$$

Elles impliquent en particulier $X^{\perp\!\!\!\perp\perp\!\!\!\perp\perp\!\!\!\perp} = X^{\perp\!\!\!\perp}$, donc l'ensemble des parties closes de Λ (respectivement de Π) est un treillis complet pour l'inclusion, avec l'intersection comme borne inférieure et la clôture de l'union comme borne supérieure, de façon analogue aux sous-espaces d'un espace vectoriel. Le plus grand élément du treillis en est Λ (resp. Π) et le plus petit élément est $\emptyset^{\perp\!\!\!\perp\perp\!\!\!\perp} = \Pi^{\perp\!\!\!\perp}$ (resp. $\Lambda^{\perp\!\!\!\perp}$). On comprend le bi-orthogonal comme une complétion par équivalence observationnelle, puisque $X^{\perp\!\!\!\perp\perp\!\!\!\perp}$ est l'ensemble des termes qui mènent à $\perp\!\!\!\perp$ face aux mêmes continuations que X, donc qui passent avec succès les mêmes tests que X. Les parties closes de Λ et Π seront donc appelées *comportements* de termes et de piles respectivement.

Remarque 2.5. L'orthogonalité engendre une équivalence dite observationnelle sur les termes : $t \simeq u$ si $\{t\}^{\perp\!\!\!\perp} = \{u\}^{\perp\!\!\!\perp}$. Cette équivalence est donc paramétrée par le choix de $\perp\!\!\!\perp$, et elle est en général sans rapport direct avec les équivalences usuelles sur le λ-calcul. On peut aussi considérer l'équivalence $t \cong u$ si $t \simeq u$ pour tout $\perp\!\!\!\perp$. Dans ce cas on constate que $t \cong u$ si et seulement si, pour toute pile π, on a $t * \pi \to^* u * \pi$ et $u * \pi \to^* t * \pi$: si c'est le cas, la condition de clôture par anti-réduction garantit que les orthogonaux de t et u sont égaux ; si ce n'est pas le cas, il existe une pile π telle que $t * \pi$ ne se réduit pas en $u * \pi$ (ou

l'inverse), donc en prenant pour $\perp\!\!\!\perp$ l'ensemble des exécutables qui se réduisent en $u * \pi$ on a $u \perp\!\!\!\perp \pi$ mais pas $t \perp\!\!\!\perp \pi$.

Les treillis de comportements de termes et de piles sont donc en bijection par la négation $(\cdot)^{\perp\!\!\!\perp}$, et ils sont isomorphes à inversion près de l'ordre. D'après les propriétés du bi-orthogonal, être orthogonal à un ensemble X est équivalent à être orthogonal à $X^{\perp\!\!\!\perp\perp\!\!\!\perp}$, donc on s'autorisera à considérer toute partie de Π comme un comportement de piles, en identifiant un ensemble de piles et le comportement qu'il engendre. On peut ainsi poser les définitions suivantes :

Définition 2.6 (valuation)**.** Soit $\perp\!\!\!\perp$ une observation. Une valuation est la donnée, pour chaque variable propositionnelle X, d'un ensemble $[X] \subseteq \Pi$. Étant donnée une telle valuation, l'ensemble de piles $[A]$ associé à une formule A est défini inductivement par

$$[A \to B] := [A]^{\perp\!\!\!\perp} \cdot [B] = \left\{ t \cdot \pi \;\middle|\; t \in [A]^{\perp\!\!\!\perp}, \pi \in [B] \right\} \tag{2.2}$$

$$[\forall X\, A(X)] := \bigcup_{[X]\subseteq\Pi} [A(X)] \tag{2.3}$$

On pose $[\![A]\!] = [A]^{\perp\!\!\!\perp}$, et on dit qu'un terme t réalise un type A si $t \in [\![A]\!]$, ce que l'on note également $t \Vdash A$.

Dans la suite, on garde toujours implicite la dépendance en $\perp\!\!\!\perp$ pour simplifier les notations, car on travaille en général à observation fixée.

Cette définition donne une interprétation sémantique des types, qui correspond à l'intuition de ce qu'est le comportement d'un terme de chaque type. La définition des comportements est fondée sur un raisonnement par dualité, qui est explicite dans l'interprétation de l'implication, et la valeur de certains types en donne une bonne illustration. Pour le type $\bot$, qui représente logiquement l'absurde, on a $[\bot] = [\forall X\, X] = \bigcup_{[X]\subseteq\Pi}[X] = \Pi$, donc $[\![\bot]\!]$ est le plus petit comportement de termes. C'est l'ensemble des termes t tels que pour toute pile π on ait $t * \pi \in \perp\!\!\!\perp$, autrement dit l'ensemble des t qui mènent à $\perp\!\!\!\perp$ quel que soit le contexte où il sont évalués. Pour chaque $u * \pi \in \perp\!\!\!\perp$ et pour toute pile π', on a $k_\pi u * \pi' \to k_\pi * u \cdot \pi' \to u * \pi$ donc $k_\pi u * \pi' \in \perp\!\!\!\perp$ puisque $\perp\!\!\!\perp$ est clos par anti-réduction, ce qui implique que $k_\pi u$ est élément de $[\![\bot]\!]$. Cet exemple montre que la similarité entre $\perp\!\!\!\perp$ et $\bot$ n'est pas qu'une affaire de notation.

En poursuivant ce raisonnement, on peut décrire le comportement $[\![\neg X]\!]$ étant donné $[X]$: on a par définition $[\neg X] = [X \to \bot] = [\![X]\!] \cdot \Pi$, donc les piles de $[\neg X]$ sont toutes les piles qui ont en tête un élément de $[\![X]\!]$. Pour toute pile $\pi \in [X]$ et tout terme $t \in [\![X]\!]$ on a $t * \pi \in \perp\!\!\!\perp$ par définition, donc par anti-réduction $k_\pi * t \cdot \pi' \in \perp\!\!\!\perp$ pour toute pile $\pi' \in \Pi$, en d'autres termes on a $k_\pi \perp\!\!\!\perp [X \to \bot]$. L'interprétation de la négation correspond donc bien à l'orthogonalité : $\pi \in [X]$ implique $k_\pi \in [\![\neg X]\!]$.

Le point crucial de cette interprétation sémantique des types est son adéquation avec la définition syntaxique du typage. L'adéquation se formule de la façon suivante :

Théorème 2.7 (adéquation)**.** *Si le jugement $x_1 : A_1, \ldots, x_n : A_n \vdash t : B$ est dérivable, alors pour tout $\perp\!\!\!\perp$, toute valuation des variables propositionnelles et tous termes $t_1 \in [\![A_1]\!]$ ... $t_n \in [\![A_n]\!]$ on a $t[t_1/x_1, \ldots, t_n/x_n] \in [\![B]\!]$.*

Démonstration. La démonstration se fait par induction sur la dérivation de typage. On note Γ l'environnement de typage $x_1 : A_1, \ldots, x_n : A_n$ et on suppose

choisis une valuation des variables propositionnelles et des termes $t_i \in [\![A_i]\!]$. On note $\bar{t}$ pour $t[t_1/x_1, \ldots, t_n/x_n]$.

Pour la règle d'axiome, le jugement est de la forme $x : A \vdash x : A$, donc si $t \in [\![A]\!]$, par définition $t * \pi$ est élément $\perp\!\!\!\perp$ pour toute pile π de $[A]$.

Pour l'application, soient t et u tels que $\Gamma \vdash t : A \to B$ et $\Gamma \vdash u : A$ soient dérivables. Par hypothèse d'induction, on a donc $\bar{t} \in [\![A \to B]\!]$ et $\bar{u} \in [\![A]\!]$. Soit $\pi \in [B]$, par définition $[A \to B] = [\![A]\!] \cdot [B]$ donc $\bar{u} \cdot \pi$ est élément de $[A \to B]$, et par conséquent $\bar{t} * \bar{u} \cdot \pi$ est élément de $\perp\!\!\!\perp$, et comme $\perp\!\!\!\perp$ est clos par anti-réduction on a bien $\overline{t\,u} * \pi \in \perp\!\!\!\perp$.

Pour l'abstraction, supposons $\Gamma, x : A \vdash t : B$ dérivable. Alors, par hypothèse d'induction, pour tout terme $u \in [\![A]\!]$ et toute pile $\pi \in [B]$ on a $\bar{t}[u/x] * \pi \in \perp\!\!\!\perp$, d'où par anti-réduction $\overline{\lambda x.t} * u \cdot \pi \in \perp\!\!\!\perp$. Comme l'ensemble $\{\, u \cdot \pi \mid u \in [\![A]\!], \pi \in [B] \,\}$ est, par définition, égal à $[A \to B]$, on a bien le résultat voulu.

Pour la constante cc, considérons une pile $t \cdot \pi \in [(A \to B) \to A) \to A]$, donc avec $t \in [\![(A \to B) \to A]\!]$ et $\pi \in [A]$. On a la réduction $\mathrm{cc} * t \cdot \pi \to t * k_\pi \cdot \pi$, donc il suffit de démontrer $k_\pi \in [\![A \to B]\!]$. Soit donc $u \cdot \pi'$ une pile de $[A \to B]$, c'est-à-dire telle que $u \in [\![A]\!]$ et $\pi' \in [B]$. L'exécutable $k_\pi * u \cdot \pi'$ se réduit en $u * \pi$, or $\pi \in [A]$ par hypothèse donc $u * \pi$ est élément de $\perp\!\!\!\perp$, et par anti-réduction on a bien $k_\pi * u \cdot \pi' \in \perp\!\!\!\perp$, d'où $k_\pi \in [\![A \to B]\!]$, ce qui permet de conclure.

Pour la quantification, supposons $\Gamma \vdash t : A(X)$ dérivable, où X n'est pas une variable libre dans Γ. Par hypothèse d'induction, on prouve donc que pour toute valuation de X on a $\bar{t} \in [\![A(X)]\!]$, puisque les substitutions $[t_i/x_i]$ sont indépendantes de X, donc on a $\bar{t} * \pi \in \perp\!\!\!\perp$, pour toute valeur de X et toute pile $\pi \in [A(X)]$, c'est-à-dire pour toute pile de $[\forall X\, A(X)]$.

Pour le cas de l'élimination du quantificateur, par hypothèse d'induction on montre que $\bar{t} \in [\![\forall X\, A(X)]\!]$, donc que pour toute valeur de X et toute pile $\pi \in [A(X)]$ on a $t * \pi \in \perp\!\!\!\perp$. En attribuant à X la valeur $[B]$, on vérifie que $[A(X)] = [A(B)]$, ce qui permet de déduire $\pi \in [A(B)]$ et donc $\bar{t} \in [\![A(B)]\!]$. □

En corollaire, on a donc le cas particulier suivant, pour les formules closes :

Corollaire 2.8. *Si $\vdash t : A$ est dérivable, alors on a $t \Vdash A$ pour tout $\perp\!\!\!\perp$.*

Le choix du paramètre $\perp\!\!\!\perp$ engendre donc une famille de treillis de comportements qui sont autant de modèles de la logique classique, et le théorème d'adéquation permet de construire des objets réalisant les formules démontrables indépendamment du paramètre.

Remarque 2.9. L'énoncé du théorème d'adéquation ne suppose rien d'autre que la clôture de $\perp\!\!\!\perp$ par anti-réduction, en particulier il reste valide si $\perp\!\!\!\perp$ est vide. Dans ce cas il y a exactement deux comportements : l'ensemble vide et l'ensemble de tous les termes. On retombe donc sur un modèle classique à deux valeurs, et l'interprétation d'une formule est alors sa valeur de vérité dans le calcul des prédicats.

Selon la terminologie de Krivine [52], on appelle *quasi-preuves* les λ-termes clos dans lesquels n'apparaît aucun k_π. L'observation $\perp\!\!\!\perp$ est dite cohérente si aucune quasi-preuve ne réalise $\bot$. Dans ce cas, en munissant le treillis des comportements d'un préordre plus large que celui de l'inclusion, il est possible de retrouver une structure d'algèbre de Boole pour les connecteurs $\wedge$ et $\vee$ définis plus haut sur les types.

Théorème 2.10 (Krivine)**.** *Soit $\leqslant$ le préordre sur les comportements défini comme $U \leqslant V$ s'il existe une quasi-preuve t telle que $t \Vdash U \to V$. Pour tout $\perp\!\!\!\perp$ tel qu'aucune quasi-preuve ne réalise $\bot$, l'ensemble des comportements ordonnés par $\leqslant$ est une algèbre de Boole.*

Démonstration. Il est clair que $\leqslant$ est un préordre puisque $\lambda x.x$ réalise $U \to U$ pour tout U et que $t \Vdash U \to V$ et $u \Vdash V \to W$ entraînent $\lambda x.u(tx) \Vdash U \to W$. On montre ensuite que $\wedge$ et $\vee$, définis par la notation 2.3, sont les bornes inférieure et supérieure pour ce préordre, et les axiomes voulus sont réalisés par les opérations standard qui représentent les couples et les sommes disjointes en λ-calcul. Les détails de la démonstration se trouvent dans [52]. □

L'intérêt d'avoir $\perp\!\!\!\perp$ comme paramètre dans la construction d'un modèle est que la signification de l'orthogonalité peut être modulée pour démontrer diverses propriétés des termes typés, comme on l'illustrera abondamment dans la suite. On utilise cette propriété en particulier pour obtenir des résultats de spécification, c'est-à-dire pour décrire le comportement commun de tous les termes réalisant un type donné. La spécification la plus simple est celle de l'identité :

Proposition 2.11. *Soit t un terme de type $\forall X\, X \to X$. Pour tout terme u et toute pile π on a $t * u \cdot \pi \to^* u * \pi$.*

Démonstration. Définissons $\perp\!\!\!\perp$ comme la clôture par anti-réduction de l'exécutable $u * \pi$. D'après le théorème d'adéquation, $\vdash t : \forall X\, X \to X$ implique $t \Vdash \forall X\, X \to X$, donc pour toute valeur de $[X]$ on a $t \perp\!\!\!\perp [\![X]\!] \cdot [X]$. En posant $[X] = \{\pi\}$, $u * \pi \in \perp\!\!\!\perp$ implique $u \in [\![X]\!]$ donc $u \cdot \pi \in [\![X]\!] \cdot [X]$ et $t * u \cdot \pi \in \perp\!\!\!\perp$. Par définition de $\perp\!\!\!\perp$ on a donc $t * u \cdot \pi \to^* u * \pi$. □

La même technique s'applique par exemple à spécification des booléens à la Church :

Proposition 2.12. *Soit le type $B_b = \forall X_0 X_1 (X_0, X_1 \to X_b)$ pour $b \in \{0, 1\}$. Pour tout terme t de type B_b, pour tous termes u_0 et u_1 et toute pile π on a $t * u_0 \cdot u_1 \cdot \pi \to^* u_b * \pi$.*

Démonstration. Soit $b \in \{0, 1\}$ et soit $\perp\!\!\!\perp$ la clôture par anti-réduction de $\{u_b * \pi\}$. Par le théorème d'adéquation on a $t \Vdash \forall X_0 X_1 (X_0, X_1 \to X_b)$ donc pour toute valeur de $[X_0]$ et $[X_1]$ on a $t \perp\!\!\!\perp [X_0, X_1 \to X_b]$. En posant $[X_b] = \{\pi\}$ on a $u_b \in [\![X_b]\!]$, et en posant $[X_{\neg b}] = \emptyset$ on a trivialement $u_{\neg b} \in [\![X_{\neg b}]\!]$. Ainsi on a $u_0 \in [\![X_0]\!]$ et $u_1 \in [\![X_1]\!]$, d'où $u_0 \cdot u_1 \cdot \pi \in [X_0, X_1 \to X_b]$ et donc $t * u_0 \cdot u_1 \cdot \pi$ est élément de $\perp\!\!\!\perp$, donc il se réduit en $u_b * \pi$. □

En particulier, cette spécification des booléens signifie qu'aucun terme ne peut réaliser à la fois B_0 et B_1 pour tout $\perp\!\!\!\perp$, ce qui constitue une preuve de la cohérence de l'interprétation des types. On réemploiera ce genre d'argument quand le calcul sera enrichi de nouveaux combinateurs typés, afin de montrer que la cohérence est préservée.

2.1.3 Quantification du premier ordre

Dans l'énoncé de la spécification des booléens, on définit un type B_b en fonction du booléen b. Le rôle de b ici est purement syntaxique, mais il suggère la possibilité d'étendre le typage en lui ajoutant de la quantification du premier

ordre. On pourra ainsi exprimer dans les types tout énoncé de la logique classique du second ordre.

Supposons donnée une signature $\mathcal{L}$, c'est-à-dire un ensemble de symboles de fonctions munis d'une arité. Par exemple, pour représenter l'arithmétique, on se donne les symboles $0, s, +, \times$ d'arités respectives 0, 1, 2 et 2. On suppose donné également un ensemble de variables du premier ordre, ou variables d'individus (notées $x, y, z \ldots$). Les variables propositionnelles ont maintenant une arité, et le langage des formules est engendré par la grammaire

$$A, B := X^n e_1 \ldots e_n \mid \forall x\, A \mid \forall X\, A \mid A \to B \tag{2.4}$$

où X^n représente une variable de prédicat d'arité n et où les e_i sont des expressions bien formées sur la signature $\mathcal{L}$. Les règles de typage du λc-calcul restent essentiellement inchangées : l'introduction du quantificateur du premier ordre est analogue à celle du second ordre, et l'élimination des quantificateurs (ou schéma de compréhension) s'écrit maintenant

$$\frac{\Gamma \vdash t : \forall X^n\, A}{\Gamma \vdash t : A[F/X]} \qquad \frac{\Gamma \vdash t : \forall x\, A}{\Gamma \vdash t : A[e/x]} \tag{2.5}$$

où F est une formule paramétrée par n individus. La substitution $A[F/X]$ est définie inductivement en remplaçant dans A chaque formule atomique $Xe_1 \ldots e_n$ par la formule $F(e_1, \ldots, e_n)$. Par exemple, avec $F(x, y) = P(sx, x + y)$ on a $(X(0 + su, sv) \to A(u))[F/X] = P(s(0 + su), 0 + su + sv) \to A(u)$

Pour étendre la réalisabilité, on suppose maintenant donné un modèle de $\mathcal{L}$, c'est-à-dire un ensemble M d'individus muni d'une interprétation de chaque symbole $f \in \mathcal{L}$ d'arité k par une fonction $[\![f]\!] : M^k \to M$. On déduit la valeur $[\![e]\!]$ de chaque expression e sur $\mathcal{L}$, en fonction de la valeur des variables. Une valuation est alors la donnée d'une valeur $[\![x]\!] \in M$ pour chaque variable x du premier ordre, et pour chaque variable propositionnelle X d'arité k, d'une fonction $[X] : M^k \to \mathcal{P}(\Pi)$, de sorte que l'interprétation d'une formule atomique $Xe_1 \ldots e_n$ est $[X]([\![e_1]\!], \ldots, [\![e_n]\!])$. Tout le reste des définitions est inchangé, et le théorème d'adéquation 2.7 s'étend immédiatement.

Les types, interprétés comme des formules logiques, permettent maintenant de décrire des propriétés de la structure M qui interprète la signature $\mathcal{L}$. Dans le cas où $\mathcal{L}$ est le langage usuel de l'arithmétique (c'est le cas qu'on utilisera dans la suite, mais la technique s'adapte à n'importe quelle théorie), M contient donc en particulier des entiers, c'est-à-dire des $s^n 0$, mais aussi éventuellement d'autres objets. On définit donc un prédicat $\mathrm{Int}(x)$ pour identifier les entiers, à partir de la propriété caractéristique suivante : dire que les entiers sont les $s^n 0$ revient à dire qu'ils sont les objets concernés par le schéma de récurrence induit par 0 et s. Il s'agit de la définition de Dedekind, qui se formalise comme suit :

$$\mathrm{Int}(x) := \forall X((\forall y\, Xy \to Xsy) \to X0 \to Xx) \tag{2.6}$$

En d'autres termes, « x est un entier s'il vérifie toute propriété X stable par passage au successeur et vérifiée par 0. » Sans surprise, le type $\mathrm{Int}(n)$ est donc le type de l'entier de Church n. Cette définition entraîne immédiatement $\vdash \lambda fx.x : \mathrm{Int}(0)$ et d'autre part on a $\vdash \lambda nfx.f(nfx) : \forall x.\, \mathrm{Int}(x) \to \mathrm{Int}(sx)$, au moyen de

la dérivation suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{n : \mathrm{Int}(x) \vdash n : (\forall y\, Xy \to Xsy) \to X0 \to Xx}}{n : \mathrm{Int}(x), f : \forall y\, Xy \to Xsy, x : X0 \vdash nfx : Xx}}{n : \mathrm{Int}(x), f : \forall y\, Xy \to Xsy, x : X0 \vdash f(nfx) : Xsx}}{n : \mathrm{Int}(x) \vdash \lambda fx.f(nfx) : (\forall y\, Xy \to Xsy) \to X0 \to Xsx}}{n : \mathrm{Int}(x) \vdash \lambda fx.f(nfx) : \mathrm{Int}(sx)}}{\vdash \lambda nfx.f(nfx) : \forall x\ \mathrm{Int}(x) \to \mathrm{Int}(sx)}$$

On montre de même que les opérations habituelles sur les entiers de Church ont bien le type voulu :

$$\lambda mn.\lambda fx.mf(nfx) : \forall xy\ \mathrm{Int}(x), \mathrm{Int}(y) \to \mathrm{Int}(x + y) \tag{2.7}$$
$$\lambda mn.\lambda f.m(nf) : \forall xy\ \mathrm{Int}(x), \mathrm{Int}(y) \to \mathrm{Int}(x \times y) \tag{2.8}$$

et de même pour toute expression définissable par ces opérations.

Pour exprimer des énoncés d'arithmétique, il reste à définir le prédicat d'égalité. On utilise pour cela une formulation à la Leibniz, à savoir

$$(a = b) := \forall X\, Xa \to Xb \tag{2.9}$$

Cette formulation donne une relation d'équivalence, au sens où les implications suivantes sont démontrables :

$$\lambda x.x : \forall x\, (x = x) \tag{2.10}$$
$$\lambda f.f(\lambda x.x) : \forall xy\, (x = y) \to (y = x) \tag{2.11}$$
$$\lambda fgx.g(fx) : \forall xyz\, (x = y), (y = z) \to (x = z) \tag{2.12}$$

Grâce à la quantification du second ordre, on déduit que $a = b$ est équivalent à l'identité si $a = b$ dans le modèle et qu'il ne peut pas être réalisé par un terme clos si $a \neq b$ dans le modèle. Dans l'énoncé suivant, $\top$ désigne la valeur de vérité maximale, c'est-à-dire $[\![\top]\!] = \emptyset$ et $\llbracket\top\rrbracket = \Lambda$.

Proposition 2.13. *Supposons donnés un modèle M et une observation $\perp\!\!\!\perp$. Pour tous $a, b \in M$, on a $[a = b] = [\forall X\, X \to X]$ si $a = b$ et $[a = b] = [\top \to \perp]$ si $a \neq b$.*

Démonstration. L'ensemble $[a = b]$ s'écrit par définition $\bigcup_{[X] \in M \to \mathcal{P}(\Pi)} [X](a)^{\perp\!\!\!\perp} \cdot [X](b)$. Si a et b sont le même élément, cette union s'écrit $\bigcup_{Y \subseteq \Pi} Y^{\perp\!\!\!\perp} \cdot Y$ qui est bien la définition de $[\forall Y\, Y \to Y]$. Au contraire, si a et b sont distincts, $[X](a)$ et $[X](b)$ sont indépendants, donc l'union s'écrit $\bigcup_{Y \subseteq \Pi, Z \subseteq \Pi} Y^{\perp\!\!\!\perp} \cdot Z$ qui est égal à $\Lambda \cdot \Pi = [\top \to \perp]$. □

Comme conséquence immédiate de ce résultat, toute formule équationnelle $\forall x_1 \dots x_n\, t(x_1 \dots x_n) = u(x_1 \dots x_n)$ vraie dans le modèle est réalisée par l'identité. De plus, pour tout $\perp\!\!\!\perp$ cohérent (au sens du théorème 2.10), aucune quasi-preuve ne peut réaliser une telle formule si elle est fausse.

2.2 Types de données

Pour représenter des données en λ-calcul, il y a essentiellement deux approches. L'une consiste à utiliser un codage imprédicatif, à la manière des entiers de Church, comme on vient de le voir en étudiant la quantification du premier ordre, ce qui a l'avantage de se faire au sein du calcul existant, sans affecter la théorie. L'autre approche consiste à utiliser des données explicites dans les termes du calcul et à étendre le typage en conséquence. Cette seconde méthode est celle employée pour modéliser les langages de programmation par des formalismes comme PCF.

Afin d'illustrer la flexibilité de la méthode de réalisabilité classique, on s'intéresse maintenant à une extension du λc-calcul avec types de données. On traite le cas où un seul type de donnée est ajouté, la théorie pouvant se généraliser sans difficulté à une situation avec plusieurs types.

2.2.1 Définition d'un type

On considère à nouveau une signature $\mathcal{L}$. On note $|s|$ l'arité d'un symbole $s \in \mathcal{L}$. On suppose donné un ensemble de variables d'individus $x, y \ldots$ et on note $\mathcal{T}(\mathcal{L})$, ou simplement $\mathcal{T}$ s'il n'y a pas d'ambiguïté, l'ensemble des termes avec variables construits sur ce langage, et $\mathcal{T}_0(\mathcal{L})$ le langage des termes sans variables. Le type de données, d'un point de vue syntaxique, est en fait ce langage $\mathcal{T}_0$.

Définition 2.14 (λc-termes avec données)**.** Pour une signature $\mathcal{L}$, l'ensemble $\Lambda(\mathcal{L})$ des termes du λc-calcul enrichi par le type de données $\mathcal{L}$ et l'ensemble $\Pi(\mathcal{L})$ des contextes enrichis sont définis par la grammaire

$$\begin{aligned} t, u &::= x \mid t\,u \mid \lambda x.t \mid \mathrm{cc} \mid k_{\mathcal{C}} \mid s\,t_1 \ldots t_{|s|} \mid \langle e \rangle \mid \mathrm{ind}(t, \vec{u}) \\ \mathcal{C} &::= [\,] \mid \mathcal{C}\,t \mid s\,e_1 \ldots e_{i-1}\,\mathcal{C}\,t_{i+1} \ldots t_{|s|} \mid \mathrm{ind}(\mathcal{C}, \vec{u}) \end{aligned}$$

où s est un élément de $\mathcal{L}$ avec $|s| \geqslant 1$, i est un entier entre 1 et $|s|$, et e est un élément de $\mathcal{T}_0$. L'arité de ind est 1 plus le nombre d'éléments de $\mathcal{L}$.

On a donc plongé le langage $\mathcal{L}$ dans les λc-termes, pour représenter des calculs sur des objets de $\mathcal{T}_0$. À chaque symbole de $\mathcal{L}$ correspond une construction de terme de même arité, et $\langle e \rangle$ représente un élément de $\mathcal{T}_0$ en tant que valeur. La construction ind représente le schéma d'induction sur les valeurs, comme nous allons l'expliquer plus loin.

Il faut maintenant étendre la relation de réduction sur les exécutables, c'est-à-dire les éléments de $\Lambda(\mathcal{L}) \times \Pi(\mathcal{L})$. On notera $\mathcal{C}\{t\}$ l'exécutable $(t, \mathcal{C})$, pour souligner le fait qu'il s'agit simplement d'un terme avec un sous-terme marqué. Comme le suggère la forme des contextes d'évaluation, on garde la réduction de tête faible, mais en évaluant les données de façon stricte. Un contexte $s\,e_1 \ldots e_{i-1}[\,]\,t_{i+1} \ldots t_n$ représente la situation où l'on évalue le i-ème argument de la fonction s, après avoir calculé les valeurs des premiers arguments. Le seul opérateur dont l'évaluation n'est pas évidente est le schéma d'induction.

L'ensemble $\mathcal{T}_0$ est l'algèbre libre engendrée par les opérateurs de $\mathcal{L}$, c'est-à-dire que ses éléments se décomposent en un symbole racine s, élément de $\mathcal{L}$, et $|s|$ sous-termes directs, éléments de $\mathcal{T}_0$. Prouver une propriété sur $\mathcal{T}_0$ se fait donc naturellement par induction, avec un cas d'induction par symbole de $\mathcal{L}$. Traduit en schéma de preuve, ce principe se formule de la façon suivante :

Définition 2.15 (induction)**.** Soit $\mathcal{L}$ une signature, soit e un terme de $\mathcal{T}_0(\mathcal{L})$ et soit $\vec{u}$ une famille de termes de $\Lambda(\mathcal{L})$ indexée par $\mathcal{L}$. L'induction sur $\mathcal{T}_0$ par les termes de $\vec{u}$ est définie par

$$[\![s\, e_1 \dots e_{|s|}]\!]\vec{u} := u_s([\![e_1]\!]\vec{u}) \cdots ([\![e_{|s|}]\!]\vec{u}) \tag{2.13}$$

Considérons l'exemple des entiers, c'est-à-dire l'algèbre engendrée par la signature $\mathcal{I} = \{0, s\}$ avec 0 d'arité 0 et s d'arité 1. L'ensemble des termes sur $\mathcal{I}$ est alors l'ensemble des $s^n 0$ pour $n \in \mathbb{N}$. Le schéma d'induction dans ce cas prend deux paramètres, et sa définition se réduit en $[\![s^n 0]\!]u_0 u_s = u_s^n u_0$, c'est-à-dire que $[\![s^n 0]\!]$ est intuitivement l'entier de Church n.

Définition 2.16 (réduction)**.** La relation de réduction $\to$ est la clôture réflexive transitive du système de règles suivant :

$$\begin{array}{ll}
\text{push :} & \mathcal{C}\{t\, u\} \to \mathcal{C}[\{t\}\, u] \\
\text{pop :} & \mathcal{C}[\{\lambda x.t\}\, u] \to \mathcal{C}\{t[u/x]\} \\
\text{save :} & \mathcal{C}[\{\mathrm{cc}\}\, t] \to \mathcal{C}[\{t\}\, k_{\mathcal{C}}] \\
\text{restore :} & \mathcal{C}'[\{k_{\mathcal{C}}\}\, t] \to \mathcal{C}\{t\}
\end{array}$$

qui sont les règles du λc-calcul en appel par nom,

$$\begin{array}{ll}
\text{first :} & \mathcal{C}\{s\, t_1 \dots t_{|s|}\} \to \mathcal{C}[s\, \{t_1\}\, t_2 \dots t_{|s|}] \\
\text{next :} & \mathcal{C}[s\, e_1 \dots e_{i-1}\, \{\langle e_i \rangle\}\, t_{i+1} \dots t_{|s|}] \to \mathcal{C}[s\, e_1 \dots e_{i-1} e_i\, \{t_{i+1}\}\, t_{i+2} \dots t_{|s|}] \\
\text{cons :} & \mathcal{C}[s\, e_1 \dots e_{|s|-1}\, \{\langle e_{|s|} \rangle\}] \to \mathcal{C}\{\langle s\, e_1 \dots e_{|s|} \rangle\}
\end{array}$$

pour chaque symbole s d'arité $|s| \geqslant 1$, et

$$\begin{array}{ll}
\text{val :} & \mathcal{C}\{\mathrm{ind}(t, \vec{u})\} \to \mathcal{C}[\mathrm{ind}(\{t\}, \vec{u})] \\
\text{fold :} & \mathcal{C}[\mathrm{ind}(\{\langle e \rangle\}, \vec{u})] \to \mathcal{C}\{[\![e]\!]\vec{u}\}
\end{array}$$

pour le schéma d'induction.

À partir de cette notion de réduction, les définitions de la réalisabilité classiques se redéfinissent à l'identique, pour toute la logique du second ordre. Les individus intervenant dans les types sont les termes de $\mathcal{T}(\mathcal{L})$, c'est-à-dire les termes avec variables. Rappelons que la quantification du premier ordre s'énonce

$$\frac{\Gamma \vdash t : A(x)}{\Gamma \vdash t : \forall x\, A(x)}\, x \notin \mathrm{fv}(\Gamma) \qquad \frac{\Gamma \vdash t : \forall x\, A(x)}{\Gamma \vdash t : A(e)}\, e \in \mathcal{T}(\mathcal{L}) \tag{2.14}$$

c'est-à-dire qu'on ne quantifie que sur les individus (et pas sur les fonctions, par exemple).

Pour typer les données, on introduit un prédicat unaire D qui identifie quels individus sont considérés comme des données. Les règles de typage pour l'introduction de ce prédicat s'écrivent

$$\frac{}{\vdash \langle e \rangle : D(e)}\, e \in \mathcal{T}_0 \qquad \frac{\Gamma \vdash t_1 : D(e_1) \qquad \cdots \qquad \Gamma \vdash t_{|s|} : D(e_{|s|})}{\Gamma \vdash s\, t_1 \dots t_{|s|} : D(s\, e_1 \dots e_{|s|})}\, s \in \mathcal{L} \tag{2.15}$$

Le typage du constructeur d'induction lui donne tout son sens :

$$\frac{\Gamma \vdash t : D(e) \qquad \Gamma \vdash u_s : \forall x_1 \dots x_{|s|}\, A(x_1), \dots, A(x_{|s|}) \to A(s\, x_1 \dots x_{|s|})}{\Gamma \vdash \mathrm{ind}(t, \vec{u}) : A(e)} \tag{2.16}$$

où il y a une prémisse par symbole s de $\mathcal{L}$. Cette règle se lit de la façon suivante : soit A un prédicat, si pour chaque $s \in \mathcal{L}$ le terme u_s est une preuve de $A(x_1), \dots, A(x_{|s|}) \to A(s\, x_1 \dots x_{|s|})$ et si t est une preuve que e est une donnée, alors $\mathrm{ind}(t, \vec{u})$ est une preuve de $A(e)$.

2.2.2 Réalisabilité

Il s'agit maintenant d'étendre la définition des modèles de réalisabilité définis dans la section 2.1.2. Rappelons que l'observation $\perp\!\!\!\perp$ une partie de $\Lambda(\mathcal{L}) \times \Pi(\mathcal{L})$, supposée close par anti-réduction, d'où l'on déduit la notion d'orthogonalité entre parties de $\Lambda(\mathcal{L})$ et parties de $\Pi(\mathcal{L})$. Pour le domaine de variation des variables du premier ordre, on va toujours considérer un ensemble M contenant $\mathcal{T}_0$, bien que par souci de généralité on puisse vouloir considérer n'importe quel ensemble muni d'une interprétation de $\mathcal{L}$.

À chaque type A on associe une partie $[A]$ de $\Pi(\mathcal{L})$, d'où l'on déduit la valeur de A par $[\![A]\!] = [A]^{\perp\!\!\!\perp}$. Cet ensemble $[A]$ se définit par induction sur la formule A, dans un environnement qui définit l'interprétation de chaque variable. L'interprétation des quantificateurs se fait par extensionnalité ; dans la stratégie de réduction de tête faible telle qu'utilisée ici, la valuation de l'implication est définie comme l'ajout d'un argument dans le contexte :

$$[\forall x\, A(x)] := \bigcup_{x \in M} [A(x)] \tag{2.17}$$

$$[\forall X^k\, A(X)] := \bigcup_{[X] \in M^k \to \mathcal{P}(\Pi)} [A(X)] \tag{2.18}$$

$$[A \to B] := \{\ \mathcal{C}[[\,]\, u] \mid u \in [\![A]\!], \mathcal{C} \in [B]\ \} \tag{2.19}$$

Reste à définir l'interprétation du prédicat D. Il doit s'agir d'une fonction $\mathcal{D}$ de M dans les parties de Π, en adéquation avec les règles de typage.

Définition 2.17 (valuation)**.** Étant donnés une signature $\mathcal{L}$, un modèle M et une observation $\perp\!\!\!\perp$, une valuation du type de données $\mathcal{L}$ est une fonction $\mathcal{D}$ de M dans $\mathcal{P}(\Pi(\mathcal{L}))$ telle que

1. pour tout $e \in \mathcal{T}_0$, $\mathcal{D}(e) \subseteq \{\langle e \rangle\}^{\perp\!\!\!\perp}$;
2. pour tout $e \in \mathcal{T}_0$, toute famille de termes $\vec{u}$ indexée sur $\mathcal{L}$ et tout contexte $\mathcal{C}$ tels que $\mathcal{C}\{[\![e]\!]\vec{u}\} \in \perp\!\!\!\perp$, $\mathcal{C}[\mathrm{ind}([\,], \vec{u})] \in \mathcal{D}(e)$;
3. pour tout symbole $s \in \mathcal{L}$, pour tout $\vec{e} \in \mathcal{T}_0^{|s|}$, tout $i \leqslant |s|$, toute famille $(u_j)_{i<j\leqslant|s|}$ de termes avec $u_j \in \mathcal{D}(e_j)^{\perp\!\!\!\perp}$, tout contexte $\mathcal{C} \in \mathcal{D}(s\, e_1 \dots e_{|s|})$, $\mathcal{C}[s\, e_1 \dots e_{i-1}\, [\,]\, u_{i+1} \dots u_{|s|}] \in \mathcal{D}(e_i)$.

La dernière condition est assez lourde à formuler, mais elle signifie simplement que le typage doit assurer que les symboles de $\mathcal{L}$ construisent des valeurs. Pour l'illustrer, remarquons que dans le cas des entiers, elle signifie que pour tout entier n on doit avoir $\{\, \mathcal{C}[s[\,]] \mid \mathcal{C} \in \mathcal{D}(sn)\,\} \subseteq \mathcal{D}(n)$, c'est-à-dire que si un contexte $\mathcal{C}$ accepte l'entier $n+1$ alors le contexte $\mathcal{C}[s[\,]]$ accepte l'entier n. Les conditions imposées par la définition 2.17 ne sont pas triviales, il n'est même pas clair a priori qu'elles sont cohérentes.

Proposition 2.18. *Pour toute fonction $\mathcal{X}$ de M dans $\Pi(\mathcal{L})$ telle que pour tout $e \in \mathcal{T}_0$, $\mathcal{X}(e) \subseteq \{\langle e\rangle\}^{\perp\!\!\!\perp}$, il existe une valuation du type de données $\mathcal{L}$ qui contient $\mathcal{X}$. La plus petite valuation contenant $\mathcal{X}$ sera notée $\mathcal{D}^*(\mathcal{X})$.*

Démonstration. Les deux premières conditions bornent la valeur de $\mathcal{D}(e)$ par $\mathcal{D}_0(e)$ et $\mathcal{D}_\vee(e)$, avec pour tout $e \in \mathcal{T}_0$:

$$\mathcal{D}_0(e) := \big\{\mathcal{C}[\mathrm{ind}([\,], \vec{u})] \;\big|\; \mathcal{C}\{[\![e]\!]\vec{u}\} \in \perp\!\!\!\perp\big\} \qquad \text{et} \qquad \mathcal{D}_\vee(e) := \{\langle e\rangle\}^{\perp\!\!\!\perp}$$

Elles expriment le typage des constantes et de l'induction. Leur cohérence vient de la règle de réduction fold : pour tout contexte $\mathcal{C}$ et toute famille de termes $\vec{u}$ tels que $\mathcal{C}\{[\![e]\!]\vec{u}\} \in \perp\!\!\!\perp$, par anti-réduction on a $\mathcal{C}[\mathrm{ind}(\{\langle e\rangle\}, \vec{u})] \in \perp\!\!\!\perp$, c'est-à-dire que $\mathcal{D}_0 \subseteq \mathcal{D}_\vee$. La troisième condition peut s'exprimer comme un point fixe en introduisant l'opérateur de complétion $\mathcal{F}$ suivant :

$$\mathcal{F}(\mathcal{D})(e) := \mathcal{D}(e) \cup \big\{\mathcal{C}[s\, e_1 \dots e_{i-1}\, [\,]\, u_{i+1} \dots u_{|s|}] \;\big|\; 1 \leqslant i \leqslant |s|, \forall j\, u_j \in \mathcal{D}(e_j)^{\perp\!\!\!\perp}\big\}$$

pour $e \in \mathcal{T}_0$ et $\mathcal{F}(\mathcal{D})(e) = \mathcal{D}(e)$ pour $e \notin \mathcal{T}_0$. La condition est alors $\mathcal{F}(\mathcal{D}) \subseteq \mathcal{D}$. Il est clair que $\mathcal{F}$ est croissant, de plus si $\mathcal{D} \subseteq \mathcal{D}_\vee$ on a aussi $\mathcal{F}(\mathcal{D}) \subseteq \mathcal{D}_\vee$ grâce aux règles next et cons. En conséquence, le plus petit point fixe de $\mathcal{F}$ contenant $\mathcal{D}_0$ et $\mathcal{X}$, qui s'écrit $\mathcal{D}^*(\mathcal{X}) := \mathcal{F}^\omega(\mathcal{D}_0 \cup \mathcal{X})$, est une valuation du type de données $\mathcal{L}$. □

Comme de coutume, pour tout terme t et tout type A, si $t \in [\![A]\!]$ on dit que t réalise A et écrit $t \Vdash A$. La définition d'une valuation est fait de sorte qu'on puisse étendre le lemme d'adéquation en complétant la preuve pour le λc-calcul sans données :

Proposition 2.19 (adéquation)**.** *Pour toute signature $\mathcal{L}$, toute observation $\perp\!\!\!\perp$ et toute valuation $\mathcal{D}$ du type D, si $x_1 : A_1, \dots, x_n : A_n \vdash t : B$ est prouvable et si pour chaque i, $t_i \Vdash A_i$, alors $t[t_1/x_1, \dots, t_n/x_n] \Vdash B$.*

À partir de là, on peut prouver un résultat de spécification, trivial à première vue, mais qui montre l'avantage de cette définition des données par rapport à un codage à la Church :

Théorème 2.20. *Pour toute donnée e et tout terme t tel que $\vdash t : D(e)$ est prouvable, pour tout contexte $\mathcal{C}$ on a $\mathcal{C}\{t\} \to \mathcal{C}\{\langle e\rangle\}$.*

Démonstration. On définit $\perp\!\!\!\perp$ comme la clôture par anti-réduction de $\mathcal{C}\{\langle e\rangle\}$, et on choisit une valuation $\mathcal{D}$ pour laquelle $\mathcal{C} \in \mathcal{D}(e)$. Une telle valuation existe car $\mathcal{C} \in \{\langle e\rangle\}^{\perp\!\!\!\perp}$, d'après la proposition 2.18. Comme $\vdash t : D(e)$ est prouvable, par adéquation on a $t \Vdash D(e)$, c'est-à-dire $t \in \mathcal{D}(e)^{\perp\!\!\!\perp}$, d'où on déduit $\mathcal{C}\{t\} \in \perp\!\!\!\perp$. □

C'est-à-dire que les règles de typage garantissent qu'un programme de type $D(n)$ termine nécessairement en renvoyant la valeur n, dans tout contexte, et quels que soient les opérateurs de contrôle employés dans l'évaluation. Ceci est un avantage par rapport aux représentations en λ-calcul pur : dans le cas des entiers de Church, on peut par exemple écrire un entier non-standard $n = \lambda fx.\mathrm{cc}\lambda k.f(f(f(k(fx))))$ qui représente l'entier 1 car il renvoie fx sur la continuation k mais commence par itérer trois fois f, ce qui permet de le distinguer de $\lambda fx.fx$. Ce genre de comportement est un problème bien connu de la représentation des données dans les calculs pour la logique classique ; il peut être résolu en λ-calcul pur au moyen d'opérateurs de mise en mémoire [49], et la démarche présentée ici est une façon différente mais comparable de traiter la question.

2.2.3 Opérateurs

Dans le formalisme utilisé jusqu'ici, on considère que les données sont exactement les termes du langage engendré par les symboles d'un ensemble $\mathcal{L}$. Cette formulation ne permet pas d'exprimer des opérations sur les données. En prenant l'exemple des entiers, on peut exprimer l'addition de la façon suivante :

$$\dfrac{n : D(n) \vdash n : D(n) \qquad \dfrac{\dfrac{}{x : D(s^m x) \vdash sx : D(s^m sx)}}{\vdash \lambda x.sx : \forall x\, D(s^m x) \to D(s^m sx)} \qquad m : D(s^m 0) \vdash m : D(s^m 0)}{\dfrac{m : D(s^m 0), n : D(n) \vdash \mathrm{ind}(n, m, \lambda x.sx) : D(s^m n)}{\vdash \lambda mn.\mathrm{ind}(n, m, \lambda x.sx) : \forall n\, D(s^m 0) \to D(n) \to D(s^m n)}}$$

mais il n'est pas possible de quantifier sur m dans le type obtenu car ce n'est pas une variable mais seulement une notation.

Pour ajouter des opérateurs visibles dans le système de types, on enrichit la signature $\mathcal{L}$ en une signature $\mathcal{L}^+$, avec des symboles supplémentaires pour les fonctions à ajouter. On se donne ensuite un système d'équations $\mathcal{E}$ sur $\mathcal{T}(\mathcal{L}^+)$, qui représente les propriétés de ces fonctions. On note $\mathcal{T}/\mathcal{E}$ le langage $\mathcal{T}$ quotienté par les équations de $\mathcal{E}$.

Définition 2.21 (système d'opérateurs)**.** Un système d'opérateurs sur une signature $\mathcal{L}$ est la donnée d'une signature disjointe $\mathcal{O}$ et d'un système d'équations $\mathcal{E}$ sur $\mathcal{T}(\mathcal{L} \cup \mathcal{O})$. Un tel système est bien défini si l'application qui à $e \in \mathcal{T}_0(\mathcal{L})$ associe la classe d'équivalence de e dans $\mathcal{T}_0(\mathcal{L} \cup \mathcal{O})/\mathcal{E}$ est une injection.

Les modèles de réalisabilité ne sont plus construits sur $\mathcal{T}_0(\mathcal{L})$ mais sur les termes enrichis, c'est-à-dire que le modèle M sous-jacent est maintenant $\mathcal{T}_0(\mathcal{L} \cup \mathcal{O})/\mathcal{E}$. Si $(\mathcal{O}, \mathcal{E})$ est bien défini, ce modèle peut être considéré comme une extension de $\mathcal{T}_0(\mathcal{L})$, donc on peut conserver les mêmes définitions. Les équations sont de la forme $t = u$ avec t et u éléments de $\mathcal{T}(\mathcal{L}^+)$. En interprétant ces équations par l'égalité à la Leibniz, on en fait des types en posant

$$(t = u) := \forall X\, Xt \to Xu \tag{2.20}$$

On en déduit la valeur des types $a = b$ quand a et b sont des valeurs sans variables, comme énoncé dans la proposition 2.13 :

$$[a = b] = \bigcup_{[X] \in M \to \mathcal{P}(\Pi)} \big[[X](a) \to [X](b)\big] = \begin{cases} [\forall X\, X \to X] = [\![\lambda x.x]\!] & \text{si } a = b \\ [\top \to \bot] & \text{si } a \neq b \end{cases}$$

C'est-à-dire que, par construction, l'identité réalise les axiomes de $\mathcal{E}$, donc on pourrait poser des axiomes de typage $\vdash \lambda x.x : a = b$ pour chaque égalité $a = b$ qui est conséquence de $\mathcal{E}$. Appliquer l'identité à une preuve permettrait alors de substituer une valeur dans le type du résultat :

$$\dfrac{\dfrac{\dfrac{\mathcal{E} \vdash a = b}{\vdash \lambda x.x : \forall X\, Xa \to Xb}}{\vdash \lambda x.x : A(a) \to A(b)} \qquad \Gamma \vdash t : A(a)}{\Gamma \vdash (\lambda x.x)t : A(b)}$$

mais par le même argument que dans l'expression de la valeur de $a = b$, on remarque que si $a = b$ et $t \Vdash A(a)$, on a toujours $t \Vdash A(b)$. On ajoute donc au typage la règle suivante :

$$\frac{\Gamma \vdash t : A(a) \qquad \mathcal{E} \vdash a = b}{\Gamma \vdash t : A(b)} \tag{2.21}$$

Par construction, cette règle conserve l'adéquation.

Reprenons l'exemple des entiers. On ajoute par exemple le symbole + binaire avec deux équations : $x+0 = x$ et $x+sy = s(x+y)$. On peut maintenant typer l'addition correctement en utilisant ces axiomes :

$$\dfrac{n : D(n) \vdash n : D(n) \qquad \dfrac{\dfrac{\dfrac{}{x : D(m+x) \vdash sx : D(s(m+x))}}{x : D(m+x) \vdash sx : D(m+sx)}}{\vdash \lambda x.sx : \forall x\, D(m+x) \to D(m+sx)} \qquad \dfrac{m : D(m) \vdash m : D(m)}{m : D(m) \vdash m : D(m+0)}}{\dfrac{m : D(s^m 0), n : D(n) \vdash \mathrm{ind}(n, m, \lambda x.sx) : D(m+n)}{\vdash \lambda mn.\mathrm{ind}(n, m, \lambda x.sx) : \forall mn\, D(m) \to D(n) \to D(m+n)}}$$

Le système obtenu permet donc de typer les termes en simplifiant l'expression des types selon un système d'équations donné, sans pour autant modifier la structure des termes. Ce mécanisme, qui est essentiellement celui de la déduction modulo [26], permet de distinguer les calculs sur les termes du premier ordre, qui sont la partie purement calculatoire de la démonstration de correction d'un programme, et la réduction des exécutables, qui est le calcul effectivement mené par le programme.

Dans ce chapitre, on a donc défini un modèle de calcul avec contrôle et établi une correspondance à la Curry-Howard entre ce calcul et une présentation de la logique classique. La correspondance s'établit au moyen d'une technique très flexible de réalisabilité qui permet de dériver des preuves de spécification pour les termes typés, par des arguments plus intuitifs que ceux qu'emploieraient des méthodes plus syntaxiques. On a montré que ce mécanisme peut s'appliquer utilement à un modèle de calcul enrichi de types de données primitifs, ce qui permet de modéliser des langages de programmations avec contrôle. C'est le passage de la logique intuitionniste à la logique classique qui permet de bien typer les mécanismes de contrôle, et le sens calculatoire du contrôle mis en jeu est assez complexe, c'est pourquoi le chapitre suivant est consacré à une étude approfondie du contenu calculatoire des types classiques.

Chapitre 3

Structures de contrôle typées

L'extension de la correspondance de Curry-Howard à la logique classique s'est faite en introduisant dans le calcul des opérateurs de contrôle. Le λc-calcul, présenté dans le chapitre précédent, emploie une instruction `call/cc` pour sauvegarder des continuations et ainsi permettre au programme de les réutiliser dans la suite de son exécution. En plus de ses variables, un terme contient alors des constantes de pile qui correspondent à plusieurs possibles du programme, et les opérateurs de contrôle permettent de changer de sortie active en cours de calcul. On cherche maintenant

On peut utilement envisager les multiples sorties d'un programme comme des fils d'exécutions distincts (en anglais des *threads*), manipulés par les instructions de contrôle qui influencent l'ordonnancement de l'exécution. Il est alors naturel de se demander s'il est possible d'écrire des mécanismes de communication entre ces threads. De plus, comme notre motivation est l'interprétation calculatoire de la logique classique, il importe que de tels mécanismes soient bien typés, c'est-à-dire qu'ils réalisent des formules de logique classique.

Plus précisément, c'est le sens calculatoire de la disjonction classique qu'il s'agit d'explorer : c'est la disjonction qui connecte les multiples conclusions d'un séquent en logique classique, et elle correspond à la corrélation qui existe entre les multiples sorties d'un programme avec contrôle. De telles formules vont spécifier des structures de contrôle, faisant apparaître des mécanismes voisins de ceux que l'on rencontre dans les langages de programmation [22]. La démarche est inhabituelle : plutôt que de deviner un typage pour une construction de programmation existante, on inverse la proposition en partant d'un type qui semble intéressant et en dérivant, par la réalisabilité, une nouvelle construction, qui de facto est bien typée.

Le contenu de ce chapitre est essentiellement le même que l'article [7], qui est une version en anglais de la section 3.2.

3.1 Approches de la disjonction

Dans cette première section, on cherche à se faire une idée des mécanismes qui entrent en jeu dans les spécifications issues de la logique classique. On étudiera ensuite le cas général des formes normales disjonctives, en utilisant les intuitions issues de l'étude de formules plus simples.

3.1.1 Loi de Peirce

Le cas initial est celui de la loi de Peirce. Le passage du typage intuitionniste au typage classique s'est fait en introduisant dans ce type la constante cc, et on montre en fait que le comportement de cet opérateur est en fait le seul possible dans notre calcul :

Proposition 3.1 (spécification de la loi de Peirce)**.** *Soit c un terme réalisant $P = \forall X\,((X \to \bot) \to X) \to X$ pour tout $\perp\!\!\!\perp$. Pour toute pile π et tous termes f et a tels que pour tout α on ait $f * \alpha \cdot \pi \to \alpha * a \cdot \pi'$ pour une certaine pile π', l'exécutable $c * f \cdot \pi$ se réduit en $a * \pi$.*

Démonstration. Posons pour $\perp\!\!\!\perp$ la clôture par anti-réduction de $\{a * \pi\}$ et pour $[X]$ le singleton $\{\pi\}$. Par définition, on a alors $a \in [\![X]\!]$. Pour tout élément α de $[\![X \to \bot]\!]$, par hypothèse $f * \alpha \cdot \pi$ se réduit en un $\alpha * a \cdot \pi'$, or $[X \to \bot] = [\![X]\!] \cdot \Pi$ donc $a \cdot \pi' \in [X \to \bot]$. Par conséquent $\alpha * a.\pi'$ est dans $\perp\!\!\!\perp$ et $f * \alpha \cdot \pi$ également, d'où $f \in [\![(X \to \bot) \to X]\!]$. Comme $\pi \in [X]$ par définition, la pile $f \cdot \pi$ est élément de $[((X \to \bot) \to X) \to X]$, et donc de $[P]$. Comme c est élément de $[\![P]\!]$ par hypothèse, $c * f \cdot \pi$ est alors élément de $\perp\!\!\!\perp$, ce qui signifie qu'il se réduit en $a * \pi$. □

Ceci signifie en fait que si la fonction f utilise son argument, donc le met en tête (nécessairement sur une pile non vide, d'après le typage), alors $c * f \cdot \pi$ se comporte comme $f * k_\pi \cdot \pi$. Si f ne met pas son argument en tête, le résultat est trivial, donc cette proposition a bien le sens que l'on veut lui donner : cc applique son argument à un objet qui sauvegarde le contexte initial.

Il est bien évident que le fait de tomber sur cette spécification dépend fortement du modèle de calcul considéré, et il est légitime de se demander si l'on ne tourne pas en rond en extrayant une spécification de la loi de Peirce après avoir introduit cc dans ce type explicitement. Néanmoins, le fait d'avoir défini cc comme on l'a fait n'a effectivement aucune influence sur la spécification, puisque la réalisabilité et l'interprétation des types sont définis uniquement à partir de l'application et de la stratégie de réduction.

Le combinateur cc est l'élément de base sur lequel on construit dans la suite des démonstrations de logique classique. Il est en général appliqué sous la forme ccλx, c'est-à-dire suivi d'une abstraction (et c'est d'ailleurs aussi le cas pour `call/cc` dans les langages où cette primitive de contrôle existe). Afin d'alléger les notations, on reformulera souvent cette construction comme un nouveau lieur dual de λ, qui substitue des continuations là où λ substitue des termes :

Notation 3.2. On note $\kappa x.t$ l'expression cc$(\lambda x.t)$. La règle de typage de ce lieur est la reformulation de la loi de Peirce comme règle d'inférence :

$$\frac{\Gamma, x : A \to B \vdash t : A}{\Gamma \vdash \kappa x.t : A}$$

Cette règle de typage est simplement celle de cc$\lambda x.t$, donc son adéquation avec l'interprétation des types est immédiate.

3.1.2 Le tiers-exclu

Le tiers-exclu, sous forme disjonctive, s'écrit $\forall A(\neg A \vee A)$. Sa reformulation dans notre système de types, selon la notation 2.3 page 15, s'écrit donc

$$T = \forall A \forall B\ (\neg A \to B) \to (A \to B) \to B$$

Un terme de type $\neg A$ est un terme qui, appliqué à un terme de type A, donne un objet de type $\bot$, donc de n'importe quel type. Il est facile de voir que, puisque le système de typage utilisé est logiquement cohérent, aucun terme pur ne peut avoir cette propriété dès que le type A est habité. Par conséquent, un terme de type $\neg A \to B$ ne peut être appliqué qu'à un objet contenant une continuation, c'est-à-dire un objet produit en utilisant cc.

Le type T est une proposition démontrable en logique classique, il existe donc des termes qui la réalisent. Considérons par exemple :

$$M_T := \lambda f.\lambda h.\kappa x.f(\lambda y.x(hy))$$

Ce terme a T pour type, selon la preuve suivante où Γ est le contexte de typage $x : B \to C, h : A \to B, f : (A \to C) \to B$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash x : B \to C \qquad \Gamma, y : A \vdash hy : B}{\Gamma, y : A \vdash x(hy) : C}}{\Gamma \vdash \lambda y.x(hy) : A \to C} \qquad \Gamma \vdash f : (A \to C) \to B}{\Gamma \vdash f(\lambda y.x(hy)) : B}}{h : A \to B, f : (A \to C) \to B \vdash \kappa x.f(\lambda y.x(hy)) : B}}{f : (A \to C) \to B \vdash \lambda h.\kappa x.f(\lambda y.x(hy)) : (A \to B) \to B}}{\vdash \lambda f.\lambda h.\kappa x.f(\lambda y.x(hy)) : ((A \to C) \to B) \to (A \to B) \to B}$$

Étudions donc le comportement de ce terme M_T. Pour tous termes f et h et toute pile π, on a

$$M_T * f \cdot h \cdot \pi \to f * \alpha_{h,\pi} \cdot \pi$$

en posant $\alpha_{h,\pi} = \lambda y.k_\pi(hy)$. Dans le cas où la réduction met ensuite $\alpha_{h,\pi}$ en tête, le typage garantit que la pile sur laquelle il se trouve n'est pas vide, et la règle qui s'applique est

$$\alpha_{h,\pi} * a \cdot \pi' \to h * a \cdot \pi$$

En considérant ce comportement avec un œil de programmeur, on peut reconnaître une sorte de structure d'exception. On commence par évaluer f (le corps) avec en argument $\alpha_{h,\pi}$ (l'exception). Si cette évaluation met $\alpha_{h,\pi}$ en tête (lève l'exception) sur une pile avec un terme a au sommet (la donnée passée à l'exception), on évalue h (le gestionnaire d'exception) avec a en argument sur la pile π (dans le contexte de départ). La différence avec les exceptions telles qu'on les trouve par exemple dans Caml [59] est qu'ici le mécanisme est correctement typé (en effet, l'opérateur **raise**, qui a le type absurde exn $\to \alpha$, est un artifice qui sert à permettre le typage des exceptions), assurant que toute exception est rattrapée car il s'agit d'un mécanisme local. Ceci dit, la comparaison avec Caml est à considérer avec prudence, en particulier parce que ce langage fonctionne en appel par valeur, alors que notre calcul est à appel par nom.

On peut montrer que ce que spécifie le type T est en fait le comportement de M_T ou un comportement symétrique :

Proposition 3.3 (spécification du tiers-exclu)**.** *Soit c un terme tel que $\vdash c : T$ soit dérivable. Pour tous termes f, h et a et pour toutes piles π, π_f et π_h pour lesquels*

- *pour tout α, $f * \alpha \cdot \pi \to \alpha * a \cdot \pi_f$,*
- *pour tout α, $h * \alpha \cdot \pi \to \alpha * \pi_h$,*

$c * f \cdot h \cdot \pi$ se réduit en $a * \pi_h$.

Démonstration. Supposons c, f, h, a, π, π_f et π_h définis selon l'énoncé. Définissons $\perp\!\!\!\perp$ comme la clôture de $\{a * \pi_h\}$ par anti-réduction, et posons $[A] = \{\pi_h\}$ et $[B] = \{\pi\}$. Pour tout α de $[\![A]\!]$, $\alpha * \pi_h \in \perp\!\!\!\perp$ implique $h * \alpha \cdot \pi \in \perp\!\!\!\perp$, donc h est élément de $[\![A \to B]\!]$. De même, pour tout α de $[\![\neg A]\!]$ on a $\alpha * a \cdot \pi_g \in \perp\!\!\!\perp$ car $a \in [\![A]\!]$, d'où $f * \alpha \cdot \pi \in \perp\!\!\!\perp$, ce qui prouve $f \in [\![\neg A \to B]\!]$. Par conséquent, comme c est élément de $[\![T]\!]$ par adéquation, $c * f \cdot h \cdot \pi$ est élément de $\perp\!\!\!\perp$, donc il se réduit en $a * \pi_h$. □

Ceci dit, les termes de type T peuvent se comporter différemment de M_T. On peut en effet donner une définition symétrique :

$$N_T := \lambda f.\lambda h.\kappa x.h(\kappa y.x(fy))$$

Le comportement de N_T est alors symétrique dans la mesure où il évalue h avant f, selon les règles

$$N_T * f \cdot h \cdot \pi \to h * \beta_{f,\pi} \cdot \pi$$
$$\beta_{f,\pi} * \pi' \to f * k_{\pi'} \cdot \pi$$

C'est-à-dire qu'au lieu de passer une donnée de f à h, on passe une continuation de h à f. On peut tenter d'expliquer intuitivement ce comportement en disant que N_T évalue h avec en argument $\beta_{f,\pi}$ (sorte de co-exception). Si h met $\beta_{f,\pi}$ en tête (demande une valeur), f est évaluée dans le contexte initial avec en argument la pile où se trouvait $\beta_{f,\pi}$ (le point où l'évaluation de h s'est interrompue). f peut alors soit ignorer cette continuation, soit l'utiliser pour envoyer une donnée à h. On retrouve d'ailleurs ici le passage de donnée de f à h.

Le calcul est manifestement différent, or la proposition 3.3 signifie que les exécutables $M_T * f \cdot h \cdot \pi$ et $N_T * f \cdot h \cdot \pi$ doivent aboutir au même résultat. Ceci signifie simplement que si f et g mettent tous deux leur argument (quel qu'il soit) en tête, alors le résultat sera le même, ce dont on peut se convaincre aisément. Moyennant cette condition, l'ordre dans lequel f et h sont appelés n'a pas d'importance.

Reprenons les réductions de M_T. On avait donc

$$M_T * f \cdot h \cdot \pi \to f * \alpha_{h,\pi} \cdot \pi$$
$$\alpha_{h,\pi} * a \cdot \pi' \to h * a \cdot \pi$$

Comme le terme a est issu de la réduction de f sur une pile contenant $\alpha_{h,\pi}$, il peut lui aussi contenir $\alpha_{h,\pi}$. La réduction de $h * a \cdot \pi$ peut donc le mettre en tête, et dans ce cas on a les réductions suivantes :

$$h * a \cdot \pi \to a * \pi_1 \to \alpha_{h,\pi} * a' \cdot \pi_2$$

La règle sur $\alpha_{h,\pi}$ s'applique alors à nouveau et on a

$$\alpha_{h,\pi} * a' \cdot \pi_2 \to h * a' \cdot \pi \to a' * \pi_1[a'/a]$$

où la deuxième réduction est justifiée par le fait que la réduction de $h * a \cdot \pi$ est indépendante de a jusqu'à ce que celui-ci arrive en tête, dans l'exécutable $a * \pi_1$.

On peut donc prédire toute la réduction qui mène de $h * a' \cdot \pi$ à $a' * \pi_1[a/a']$. On pourrait l'éviter en réduisant directement $\alpha_{h,\pi} * a' \cdot \pi_2$ en $a' * \pi_1[a'/a]$, ce qui serait la cas si $\alpha_{h,\pi}$ avait été remplacé par $\lambda x.(k_{\pi_1[x/a]}x)$ dans a, mais ce terme contient une substitution à l'intérieur de la continuation k_{π_1}, ce que n'autorise pas le calcul tel que nous l'avons défini.

Introduisons alors artificiellement dans l'algèbre des termes une constante C_T ainsi qu'une famille de termes $\alpha_{t,\pi}$, avec des règles de réduction qui rendent compte de ce processus. Ici, on va en fait ignorer la substitution qui a lieu dans la continuation k_{π_1}, en introduisant les règles suivantes :

$$\begin{array}{ll} \text{try :} & C_T * f \cdot h \cdot \pi \to f * \alpha_{h,\pi} \cdot \pi \\ \text{catch :} & \alpha_{h,\pi} * a \cdot \pi' \to h * \kappa x.a[x/\alpha_{h,\pi}] \cdot \pi \end{array}$$

où la variable x est naturellement supposée libre dans a. Ceci revient donc à remplacer $\alpha_{h,\pi}$ par k_{π_1}, avec les notations précédentes. On n'effectue pas de substitution dans ce k_{π_1} parce qu'une telle opération pose des problèmes de définition ; par exemple, on ne doit pas substituer x à tous les sous-termes égaux à a, mais uniquement ceux qui sont issus de la réduction de $f * \alpha_{h,\pi} \cdot \pi$.

Dans ce nouveau système de réductions, le combinateur synthétique C_T ne se comporte pas de la même façon que M_T. Cependant, il réalise bien T, si l'on suppose quelques conditions supplémentaires sur $\perp\!\!\!\perp$:

Définition 3.4 (séquentialité)**.** L'ensemble d'observables $\perp\!\!\!\perp$ est dit *séquentiel* s'il vérifie les conditions suivantes :

1. $\perp\!\!\!\perp$ est clos par réduction,
2. si $e[t/x] \in \perp\!\!\!\perp$ et si la réduction de e ne place jamais x en tête, alors $e[u/x] \in \perp\!\!\!\perp$ pour tout u,
3. $\perp\!\!\!\perp$ ne contient aucun exécutable de la forme $\lambda x.t * \rho$.

Il reste encore à supposer que toute réduction dans $\perp\!\!\!\perp$ termine. Ce n'est pas fondamentalement nécessaire, on pourrait énoncer une condition moins restrictive.

Proposition 3.5. *Pour tout $\perp\!\!\!\perp$ séquentiel, le combinateur C_T réalise T.*

Démonstration. Considérons un $\perp\!\!\!\perp$ vérifiant les conditions de séquentialité et une pile $f \cdot h \cdot \pi$ de l'ensemble $[T]$. Il existe donc des ensembles de piles $\mathcal{A}$ et $\mathcal{B}$ tels que l'on ait $f \in [\![(\mathcal{A} \to \bot) \to \mathcal{B}]\!]$, $h \in [\![\mathcal{A} \to \mathcal{B}]\!]$ et $\pi \in \mathcal{B}$. Pour prouver que $C_T * f \cdot h \cdot \pi$ est élément de $\perp\!\!\!\perp$, il suffit alors de prouver que $f * \alpha_{h,\pi} \cdot \pi$ est dans $\perp\!\!\!\perp$, ce qui est assuré si $\alpha_{h,\pi}$ est élément de $[\![\mathcal{A} \to \bot]\!]$. Considérons donc une pile $a \cdot \pi'$ élément de $[\mathcal{A} \to \bot] = [\![\mathcal{A}]\!] \cdot \Pi$. L'exécutable $\alpha_{h,\pi} * a \cdot \pi'$ se réduit en $h * \kappa x.a[x/\alpha_{h,\pi}] \cdot \pi$, or par hypothèse on a $h \in [\![\mathcal{A} \to \mathcal{B}]\!]$ et $\pi \in \mathcal{B}$, donc pour conclure il suffit de prouver que $\kappa x.a[x/\alpha_{h,\pi}] \in [\![\mathcal{A}]\!]$, sachant que $a \in [\![\mathcal{A}]\!]$ par hypothèse.

Soit π_1 une pile de $\mathcal{A}$. La réduction de $\kappa x.a[x/\alpha_{h,\pi}] * \pi_1$ donne $a[k_{\pi_1}/\alpha_{h,\pi}] * \pi_1$, puis deux cas se présentent :

- soit la réduction de $a*\pi_1$ ne place pas $\alpha_{h,\pi}$ en tête, auquel cas la deuxième condition sur $\perp\!\!\!\perp$ prouve que $a[k_{\pi_1}/\alpha_{h,\pi}]*\pi_1$ est élément de $\perp\!\!\!\perp$;
- soit $a*\pi_1$ se réduit en $\alpha_{h,\pi}*b\cdot\pi_2$ avec $b\in[\![\mathcal{A}]\!]$, alors $a[k_{\pi_1}/\alpha_{h,\pi}]*\pi_1$ se réduit en $k_{\pi_1}*b[k_{\pi_1}/\alpha_{h,\pi}]\cdot\pi_2$ puis en $b[k_{\pi_1}/\alpha_{h,\pi}]*\pi_1$.

Dans le deuxième cas, on se retrouve à vouloir prouver sur b ce que l'on voulait prouver sur a. Si l'on suppose que la réduction de $a[k_{\pi_1}/\alpha_{h,\pi}]*\pi_1$ termine, on a une induction bien fondée, ce qui permet de conclure. □

Moyennant un certain nombre de restrictions sur le choix des $\perp\!\!\!\perp$, on a donc prouvé que le combinateur C_T, qui n'est pas exprimable par un λc-terme, était cohérent avec le reste du calcul et du typage. Il faut cependant se demander ce qui a été perdu en restreignant la forme de $\perp\!\!\!\perp$ de cette façon.

Dans les preuves évoquées ici, on n'a jamais besoin d'utiliser de $\perp\!\!\!\perp$ qui ne vérifie explicitement pas ces conditions, donc les résultats restent vrais dans le calcul enrichi, si on les interprète correctement. Par exemple, dans la preuve de la proposition 2.11 page 19 on définissait $\perp\!\!\!\perp$ comme la clôture par anti-réduction du singleton $\{a*\pi\}$, il faut donc maintenant considérer à la place le plus petit $\perp\!\!\!\perp$ séquentiel contenant ce singleton. Le résultat que l'on démontre est alors le suivant :

Proposition 3.6 (2.11 revisité)**.** *Si $\vdash t:\forall X(X\to X)$ est dérivable, alors pour tout terme a et toute pile π tels que $a*\pi$ ne soit pas de la forme $\lambda x.t*\rho$, il existe un exécutable en lequel se réduisent $t*a\cdot\pi$ et $a*\pi$.*

La formulation est plus lourde, mais elle signifie essentiellement que $t*a\cdot\pi$ et $a*\pi$ ont (intuitivement) la même valeur dans le λc-calcul enrichi du combinateur C_T.

3.1.3 Le tiers-exclu symétrique

À titre d'exemple, étudions maintenant une variante du tiers-exclu : la formule de Gödel, qui s'écrit $\forall A\forall B\,(A\to B)\vee(B\to A)$ sous forme disjonctive. Sa formulation dans les types est donc

$$G=\forall A\,\forall B\,\forall C\;((A\to B)\to C)\to((B\to A)\to C)\to C$$

Cette forme est plus symétrique, et le comportement le sera aussi.

Le type G spécifie un comportement un peu plus complexe que T :

Proposition 3.7 (spécification du tiers-exclu symétrique)**.** *Soit c un terme tel que $\vdash c:G$ soit dérivable. Pour tous termes f, g, a et b, et pour toutes piles π, π_f et π_g pour lesquels on a*

- *pour tout α, $f*\alpha\cdot\pi\to\alpha*a\cdot\pi_f$,*
- *pour tout β, $g*\beta\cdot\pi\to\beta*b\cdot\pi_g$,*

$c*f\cdot g\cdot\pi$ se réduit soit en $b*\pi_f$ soit en $a*\pi_g$.

Démonstration. Supposons définis c, f, g, a, b, ainsi que π, π_f et π_g selon l'énoncé. On choisit pour $\perp\!\!\!\perp$ la clôture par anti-réduction de $\{b*\pi_f, a*\pi_g\}$. On pose ensuite $\mathcal{A}=\{\pi_g\}$, $\mathcal{B}=\{\pi_f\}$, et $\mathcal{C}=\{\pi\}$. On a donc $a\in[\![\mathcal{A}]\!]$ et $b\in|\mathcal{B}|$. Par conséquent, pour tout $\alpha\in[\![\mathcal{A}\to\mathcal{B}]\!]$, on a $\alpha*a\cdot\pi_f\in\perp\!\!\!\perp$, d'où $f*\alpha\cdot\pi\in\perp\!\!\!\perp$, donc $f\in[\![(\mathcal{A}\to\mathcal{B})\to\mathcal{C}]\!]$. De même, pour tout $\beta\in[\![\mathcal{B}\to\mathcal{A}]\!]$, on a $\beta*b\cdot\pi_g\in\perp\!\!\!\perp$, et par conséquent $g*\beta\cdot\pi\in\perp\!\!\!\perp$, d'où $g\in[\![(\mathcal{B}\to\mathcal{A})\to\mathcal{C}]\!]$. L'adéquation montre

que c est élément de $[\![((\mathcal{A} \to \mathcal{B}) \to \mathcal{C}) \to ((\mathcal{B} \to \mathcal{A}) \to \mathcal{C}) \to \mathcal{C}]\!]$, par conséquent l'exécutable $c * f \cdot g \cdot \pi$ est élément de $\perp\!\!\!\perp$, ce qui signifie qu'il se réduit en $b * \pi_f$ ou en $a * \pi_g$. □

Ici encore, on peut construire un terme qui a G pour type. Par exemple :

$$M_G := \lambda f.\lambda g.\kappa x.f(\lambda a.x(g(\lambda b.a)))$$

Bien entendu, ce terme ne traite pas f et g de façon symétrique. On pourrait définir un terme ayant le comportement inverse, c'est-à-dire qui applique g à un terme contenant la continuation courante.

Ce terme M_G réalise en fait plus que G, puisqu'il réalise aussi la formule $\forall A \forall B \forall C\,(A \to B) \vee (C \to A)$. Dans un contexte ou plusieurs exécutables évolueraient en parallèle, on peut imaginer de réaliser G de façon plus stricte (et plus intuitive) en implémentant un mécanisme de synchronisation entre deux processus. On pourrait en fait réaliser G ici par un mécanisme qui simulerait l'exécution parallèle de deux processus.

Selon le même principe que pour la construction de C_T à partir du tiers exclu, on peut aussi ajouter au langage un combinateur C_G, avec des règles de réduction associées, pour obtenir une réalisation de G. On introduit en plus de C_G une famille de termes $\beta_{t,\pi}$, et on définit les règles suivantes :

$$\begin{aligned} C_G * f \cdot g \cdot \pi &\to f * \beta_{g,\pi} \cdot \pi \\ \beta_{g,\pi} * a \cdot \pi &\to g * \lambda x.\kappa y.a[y/\beta_{g,\pi}] \cdot \pi \end{aligned}$$

Cette extension du calcul est construite à partir du terme M_G de la même façon que l'extension avec le combinateur C_T avait été construite à partir du terme M_T, en exploitant le fait que certaines réductions sont prévisibles, et le résultat est effectivement similaire.

3.1.4 Tautologies disjonctives

On a donc mis en évidence sur trois exemples de tautologies classiques d'une part une spécification de leur comportement, d'autre part une façon de les réaliser à l'aide de combinateurs synthétiques. On peut maintenant tenter de généraliser ces résultats. Ces trois exemples sont en fait des tautologies *disjonctives* en ce sens qu'elles sont de la forme générale suivante :

Définition 3.8 (tautologie disjonctive)**.** Une formule purement disjonctive est une formule de la forme

$$A = \bigvee_{i=1}^{n} A_i \qquad \text{avec} \qquad A_i = B_{i,1}, \ldots, B_{i,n_i} \to C_i$$

où les $B_{i,j}$ et les C_i sont des variables propositionnelles. On appelle *ensemble de vérité* d'une telle formule A l'ensemble

$$tr(A) := \{\, (i,j,k) \mid B_{i,j} = C_k \,\}$$

Lorsque $tr(A)$ n'est pas vide, A est une tautologie, et on parle alors de tautologie disjonctive.

On parle de formule *disjonctive* parce que de telles formules, en appliquant la transformation de $A \to B$ en $\neg A \vee B$, se mettent sous la forme $\bigvee_{i=1}^{n}\big(\bigvee_{j=1}^{n_i} \neg B_{i,j}\big) \vee C_i$, qui est une disjonction de littéraux. Dans le cadre des types du λc-calcul, on transformera plutôt ces formule en implications de la forme

$$\big((B_{1,1},\dots,B_{1,n_1} \to C_1) \to D\big),\dots,\big((B_{n,1},\dots,B_{n,n_n} \to C_n) \to D\big) \to D$$

Il apparaît alors que les deux exemples précédents se mettent sous cette forme (ce n'est pas le cas de la loi de Peirce). En effet :

– pour le tiers exclu, $A_1 = A \to B$ et $A_2 = A$ conduisent bien à

$$\big((A \to B) \to C\big) \to \big(A \to C\big) \to C\,;$$

– pour le tiers exclu symétrique, $A_1 = A \to B$ et $A_2 = B \to A$ conduisent à

$$\big((A \to B) \to C\big) \to \big((B \to A) \to C\big) \to C\,.$$

Ce que spécifient ces formules dans le λc-calcul est connu [22], et s'énonce de la façon suivante :

Théorème 3.9 (spécification des tautologies disjonctives)**.** *Soit A une formule purement disjonctive et soit c un terme tel que $\vdash c : A$ soit dérivable. Soient des termes f_i et $b_{i,j}$ et des piles π et π_i tels que pour terme a et pour tout i on ait $f_i * a \cdot \pi \to a * b_{i,1}\cdots b_{i,n_i} \cdot \pi_i$ (en utilisant les notations de la définition 3.8), alors il existe un (i,j,k) dans $tr(A)$ tel que $c * f_1 \cdots f_n \cdot \pi$ se réduise en $b_{i,j} * \pi_k$.*

Démonstration. Supposons que A est vraie, et donc que $tr(A)$ n'est pas vide. Soient alors des familles de termes f_i et $b_{i,j}$, une pile π et une famille de piles π_i vérifiant les conditions de la proposition. Définissons $\perp\!\!\!\perp$ comme la clôture de $\{\, b_{i,j} * \pi_k \mid (i,j,k) \in tr(A) \,\}$ par anti-réduction, et définissons les valeurs de vérité des variables par $[D] = \{\pi\}$ et $[C_i] = \{\, \pi_k \mid C_k = C_i \,\}$. Ceci ne peut ne pas définir la valeur de chaque variable intervenant dans A ; si une variable X n'apparaît pas parmi les C_i, on pose $[X] = \emptyset$ et donc $[\![X]\!] = \Lambda$.

Avec ces définitions, chaque $b_{i,j}$ appartient au $[\![B_{i,j}]\!]$ associé, soit parce que $B_{i,j}$ est trivial, soit parce qu'il existe k tel que $(i,j,k) \in tr(A)$ et donc $[B_{i,j}] = [C_k]$ et $b_{i,j} * \pi_k \in \perp\!\!\!\perp$. Par conséquent, pour tout i, pour tout $\alpha \in |A_i|$ et pour toute pile $\pi_i \in [C_i]$, l'exécutable $\alpha \cdot b_{i,1}\cdots b_{i,n_i} \cdot \pi$ est élément de $\perp\!\!\!\perp$ puisque $[A_i] = [\![B_{i,1}]\!]\cdots[\![B_{i,n_i}]\!] \cdot [C_i]$, et donc $f_i * \alpha \cdot \pi$ est aussi dans $\perp\!\!\!\perp$ par anti-réduction, d'où $f_i \in [\![A_i \to D]\!]$. Comme c réalise le type A par hypothèse, on a donc $c * f_1 \cdots f_n \cdot \pi \in \perp\!\!\!\perp$, ce qui signifie que cet exécutable se réduit en l'un des $b_{i,j} * \pi_k$ avec $(i,j,k) \in tr(A)$. □

Notons que ce résultat ne suppose pas que A est une tautologie. En effet, si la formule A est fausse, par adéquation elle n'est tout simplement pas réalisable, ce qui rend le théorème trivialement vrai. La formulation de cette spécification un peu lourde, mais on voit facilement que les propositions 3.3 et 3.7 en sont des cas particuliers.

Selon la même méthode que précédemment, on peut aussi définir des combinateurs artificiels pour réaliser ces tautologies. On peut même à nouveau introduire une liaison sur un terme pour optimiser le calcul.

Considérons donc une tautologie disjonctive A, avec les notation de la définition 3.8. Considérons un terme t qui réalise A, en utilisant un triplet particulier $(i,j,k) \in tr(A)$. Ce qu'exprime la proposition 3.9 est que si f_i et f_k utilisent leur argument selon les règles

$$f_i * a \cdot \pi \to a * b_{i,1} \cdots b_{i,n_i} \cdot \pi_i$$
$$f_k * a \cdot \pi \to a * b_{k,1} \cdots b_{k,n_k} \cdot \pi_k$$

quel que soit a, alors $t * f_1 \cdots f_n \cdot \pi$ se réduit en $b_{i,j} * \pi_k$. Le terme t se comporte donc comme le combinateur (non optimisé) suivant :

$$C_A^{i,j,k} * f_1 \cdots f_n \cdot \pi \to f_i * \alpha \cdot \pi$$
$$\alpha * b_1 \cdots b_{n_i} \cdot \pi' \to f_k * \lambda c_1 \ldots \lambda c_{n_k}.b_j \cdot \pi$$

Pour construire un combinateur optimisé, étudions alors la chaîne de réductions suivante :

$$\begin{aligned}
C_A^{i,j,k} * f_1 \cdots f_n \cdot \pi &\to f_i * \alpha \cdot \pi && \text{par définition}\\
&\to \alpha * b_1 \cdots b_{n_i} \cdot \pi' && \text{par hypothèse}\\
&\to f_k * \lambda c_1 \ldots \lambda c_{n_k}.b_j \cdot \pi && \text{par définition}\\
&\lambda c_1 \ldots \lambda c_{n_k}.b_j * c_1 \cdots c_{n_k} \cdot \pi_k && \text{par hypothèse}\\
&\to b_j * \pi_k &&\\
\text{supposons}\quad &\to \alpha * d_1 \cdots d_{n_i} \cdot \pi'' &&\\
\text{alors}\quad &\to f_k * d_j \cdot \pi && \text{par définition}\\
\text{puis}\quad &\to d_j * \big(c_1 \cdots c_{n_k} \cdot \pi_k\big)[d_j/b_j] &&
\end{aligned}$$

où la dernière ligne est justifiée par le fait que la réduction est indépendante de b_j. Donc si à la sixième étape on remplaçait α par un opérateur correspondant intuitivement à

$$\lambda\vec{d}.\big((k_{\pi_k} b_j)[d_j/b_j]\big) \qquad \text{avec } \vec{d} \text{ d'arité } n_i,$$

la réduction serait la même mais en effectuant les deux dernières chaînes de réductions en une seule étape. À nouveau, la substitution $[d_j/b_j]$ pose un problème de définition, mais si l'on remplace la règle de réduction de α par une version optimisée en ignorant cette substitution, on obtient les règles

$$C_A^{i,j,k} * f_1 \cdots f_n \cdot \pi \to f_1 * \alpha \cdot \pi$$
$$\alpha * b_1 \cdots b_{n_i} \cdot \pi' \to f_k * \lambda\vec{c}.\kappa x.\big(b_j[\lambda\vec{d}.(x\, d_j)/\alpha]\big) \cdot \pi$$

avec $\vec{c}$ d'arité n_k et $\vec{d}$ d'arité n_i. Ce combinateur, dans le nouveau système de réductions, est *valide* sous les mêmes conditions que dans le cas particulier du tiers-exclu évoqué plus haut :

Théorème 3.10 (validité des combinateurs synthétiques)**.** *Soit A une tautologie disjonctive, soit (i,j,k) un élément de l'ensemble de vérité $tr(A)$. Pour tout $\perp\!\!\!\perp$ séquentiel, le combinateur $C_A^{i,j,k}$ réalise A.*

Démonstration. Considérons un $\perp\!\!\!\perp$ vérifiant les conditions de séquentialité et une pile $f_1 \cdots f_n \cdot \pi$ de l'ensemble $A^{\perp\!\!\!\perp}$. Il existe donc une valuation des variables

de A telle que l'on ait $f_p \in [\![A_p \to D]\!]$ pour chaque p et $\pi \in [D]$. Pour prouver que $C_A^{i,j,k} * \vec{f} \cdot \pi$ est élément de $\perp\!\!\!\perp$, il suffit alors de prouver que $f_i * \alpha \cdot \pi$ est dans $\perp\!\!\!\perp$, ce qui est assuré si α est élément de $[\![A_i]\!]$, c'est-à-dire si

$$\alpha \in [\![B_{i,1}, \cdots, B_{i,n_i} \to C_i]\!].$$

Considérons donc une pile $b_1 \cdots b_{n_i} \cdot \pi'$ élément de $[A_i] = [\![B_{i,1}]\!] \cdots [\![B_{i,n_i}]\!] \cdot [C_i]$. On a donc la réduction

$$\alpha * b_1 \cdots b_{n_i} \cdot \pi' \to f_k * \lambda\vec{c}.\kappa x.(b_j[\lambda\vec{d}(xd_j)/\alpha]) \cdot \pi$$

or par hypothèse on a $f_k \in [\![A_k \to D]\!]$ et $\pi \in [D]$, donc pour conclure il suffit de prouver que $\lambda\vec{c}.\kappa x.(b_j[\lambda\vec{d}(xd_j)/\alpha])$ est élément de $[\![A_k]\!]$, donc que

$$\lambda\vec{c}.\kappa x.(b_j[\lambda\vec{d}(xd_j)/\alpha]) \in [\![B_{k,1}, \cdots, B_{k,n_k} \to C_k]\!],$$

sachant que $b_j \in [\![B_{i,j}]\!]$, donc $b_{i,j} \in [\![C_k]\!]$, par hypothèse. Soit $c_1 \cdots c_{n_k} \cdot \pi_1$ une pile de $[A_k]$. La réduction de $\lambda\vec{c}.\kappa x.b_j[\lambda\vec{d}(xd_j)/\alpha]$ sur cette pile donne $b_j[\lambda\vec{d}(k_{\pi_1}d_j)/\alpha] * \pi_1$, puis deux cas se présentent :

- soit la réduction de $b_j * \pi_1$ ne place pas α en tête, auquel cas la deuxième condition sur $\perp\!\!\!\perp$ prouve que $b_j[\lambda\vec{d}(k_{\pi_1}d_j)/\alpha] * \pi_1$ est élément de $\perp\!\!\!\perp$;
- soit $b_j * \pi_1$ se réduit en un $\alpha * d_1 \cdots d_{n_i} \cdot \pi_2$ avec $d_p \in [\![B_{i,p}]\!]$ pour chaque p, alors $b_j[\lambda\vec{d}(k_{\pi_1}d_j)/\alpha] * \pi_1$ se réduit en $k_{\pi_1} * d_j[\lambda\vec{d}(k_{\pi_1}d_j)/\alpha] \cdot \pi_2$ puis en $d_j[\lambda\vec{d}(k_{\pi_1}d_j)/\alpha] * \pi_1$.

Dans le deuxième cas, on veut donc prouver pour d_j ce que l'on voulait prouver sur b_j. On a alors une induction bien fondée si l'on suppose que la réduction de $b_j[\lambda\vec{d}(k_{\pi_1}d_j)/\alpha] * \pi_1$ termine, ce qui permet de conclure. □

3.2 Formes normales disjonctives

On a donc identifié le comportement des tautologies purement disjonctives comme une sorte de schéma de synchronisation qui plusieurs processus dans un ordre indéterminé et en fait interagir deux qui correspondent à des types complémentaires. On va maintenant généraliser cette étude à la classe des formes normales disjonctives (ou DNF pour *disjunctive normal form*), c'est-à-dire au conjonctions de disjonctions de littéraux.

Définition 3.11. Étant donné un ensemble V de variables propositionnelles, l'ensemble des littéraux est $L = V \cup \neg V$, l'ensemble des variables et des négations de variables. Deux littéraux X et $\neg X$ sont dits opposés. Un clause est une conjonction de littéraux, ou encore une partie finie de L, et une DNF est une disjonction de clauses, donc un ensemble fini de parties finies de L.

Dans la suite, on suppose donc implicitement que chaque littéral apparaît au plus une fois par clause et que chaque clause, à permutation près, apparaît au plus une fois par formule. Cette restriction naturelle a pour effet de simplifier les notations sans affecter l'expressivité des résultats. Selon le cas, les formules seront notées de façon ensembliste ou avec les connecteurs $\vee$ et $\wedge$.

Pour étudier le contenu calculatoire des DNF dans notre modèle de calcul, il est nécessaire de considérer les formules comme des types du système F,

construits sur les connecteurs $\forall$ et $\to$, comme expliqué dans les notations 2.3 page 15. Ainsi $\neg X$ représente $X \to \forall Z\, Z$, pour une clause $\{L_1, \ldots, L_k\}$ et un type T on note

$$\{L_1, \ldots, L_k\} \to T := L_1 \to \cdots \to L_k \to T$$

où l'ordre des littéraux est soit clair à partir du contexte, soit indifférent. De même, pour une formule $\Gamma = \{c_1, \ldots, c_n\}$ et une variable fraîche Z, on interprète Γ comme un type en posant

$$\Gamma := \forall Z\ (c_1 \to Z) \to \cdots \to (c_n \to Z) \to Z$$

pour un ordre approprié des clauses. Un terme de type Γ doit donc prendre en arguments n fonctions de types $c_i \to Z$. Comme le garantissent les notations définies en 2.3, l'interprétation de Γ comme type est isomorphe à Γ en tant que DNF (de façon intuitionniste), donc la conversion est neutre y compris d'un point de vue calculatoire.

3.2.1 Spécifications

L'étude des tautologies disjonctives a mis en évidence le fait que la structure de l'ensemble des comportements admissibles pour une tautologie est lié à une caractérisation combinatoire de la validité de ces tautologies. En effet, l'ensemble de triplets $tr(A)$ qualifié d'ensemble de vérité dans la définition 3.8 contient un élément par façon de prouver la validité de A. Pour traiter les formes normales disjonctives, on emploie donc la même technique, avec une caractérisation plus générale des tautologies, fondée sur la notion suivante de section :

Définition 3.12 (section)**.** Soit $\Gamma = c_1 \vee \cdots \vee c_n$ une forme normale disjonctive. L'ensemble des sections de Γ est le produit $c_1 \times \cdots \times c_n$.

En d'autres termes, une section σ est le choix d'un littéral dans chaque clause de Γ. Pour une clause $c \in \Gamma$ on note $\sigma(c)$ ce littéral et on note $L \in \sigma$ si L est un littéral choisi par σ dans l'une quelconque des clauses. Une section peut être interprétée comme un contre-exemple potentiel à la formule Γ, et en effet une telle formule est une tautologie exactement quand elle n'admet aucun contre-exemple de ce type.

Proposition 3.13. *Une forme normale disjonctive* Γ *est une tautologie si et seulement si chaque section de* Γ *contient deux littéraux opposés.*

Démonstration. Supposons qu'il existe une section σ de Γ qui ne contienne pas de littéraux opposés. On peut donc définir la valuation v des variables propositionnelles telle que $v(X)$ est vraie si $\neg X \in \sigma$, fausse si $X \in \sigma$, et quelconque sinon. Par construction, pour chaque clause $c \in \Gamma$, v rend faux un littéral de c donc $v(c)$ est faux et donc $v(\Gamma)$ est faux, ce qui prouve que Γ n'est pas une tautologie.

Réciproquement, si Γ n'est pas une tautologie, il existe une valuation v des variables propositionnelles telle que $v(c)$ soit faux pour chaque $c \in \Gamma$. La valuation d'une clause est fausse si et seulement si la clause contient un littéral de valuation fausse, donc on peut déduire de v une section σ telle que $v(\sigma(c))$ soit faux pour chaque $c \in \Gamma$. Comme tous les $\sigma(c)$ on la même valuation par v, il ne peut y avoir de littéraux opposés dans σ. □

Les sections sont donc reliées à la prouvabilité des tautologies, c'est pourquoi elle jouent un rôle important dans les spécifications. Le sens intuitif du théorème 3.14 est le suivant : si Γ est une tautologie, un terme c qui réalise Γ attend une famille d'arguments $(f_c)_{c\in\Gamma}$ indexée par les clauses de Γ. Le terme c exécute alors chaque f_c indépendamment en lui passant en argument une famille d'exceptions $(\alpha_L)_{L\in c}$ indexée par les littéraux de c. Si chaque f_c utilise l'un de ses arguments (en le plaçant en position de tête), on se retrouve avec une famille d'exécutables $\alpha_L * \pi_c$ avec $L \in c$, ce qui définit une section de Γ. Le combinateur c doit alors choisir deux littéraux opposés dans cette section, donc un terme de type X et une continuation de type $\neg X$, et les faire communiquer. La proposition 3.13 garantit précisément qu'il y a toujours deux littéraux opposés, donc deux processus à faire interagir, c'est-à-dire que la validité de la formule Γ garantit l'absence de blocage.

Notons que dans l'énoncé du théorème il n'est pas fait mention d'exceptions comme dans l'explication informelle qui précède. Ce qu'il faut supposer est en fait que chaque f_c place l'un de ses arguments en tête, quelle que soit sa valeur. L'intuition des exceptions n'est qu'une façon de formuler les choses (qui ne sera formalisée que plus loin), en insistant sur le fait que les arguments passés aux f_c ont pour rôle d'interrompre l'évaluation normale des exécutables pour rendre la main au combinateur chargé d'établir une synchronisation. Dans l'énoncé, $\vec{f}$ représente une famille (f_c) indexée par les clauses de Γ et $\vec{u}$ représente une famille (u_L) indexée par les littéraux de L. Dans les deux cas, la famille est bien sûr ordonnée comme les formules de Γ dans le type utilisé.

Théorème 3.14. *Soit Γ une tautologie en forme normale disjonctive, soient t un terme de type Γ, $\vec{f}$ une famille de termes indexée par Γ et π une pile. Supposons qu'il existe une section σ de Γ et des familles de termes et de piles (v_c) et (π_c) telles que, pour chaque clause c,*

$$\begin{array}{lll} \textit{si } \sigma(c) \in V \textit{ alors} & \forall \vec{u}\ \exists \pi' & f_c * \vec{u} \cdot \pi \to u_{\sigma(c)} * v_c \cdot \pi' \\ \textit{si } \sigma(c) \in \neg V \textit{ alors} & \forall \vec{u} & f_c * \vec{u} \cdot \pi \to u_{\sigma(c)} * \pi_c \end{array}$$

où $\vec{u}$ désigne une famille de termes indexée par les littéraux de c. Alors il existe une paire de clauses $c, c' \in \Gamma$ telle que $\sigma(c)$ et $\sigma(c')$ sont opposés et

$$t * \vec{f} \cdot \pi \to v_c * \pi_{c'}$$

Démonstration. L'ensemble de clauses Γ se décompose en fonction de la section σ en $\Gamma^+ = \{\, c \mid \sigma(c) \in V \,\}$ et $\Gamma^- = \{\, c \mid \sigma(c) \in \neg V \,\}$. Définissons alors $\perp\!\!\!\perp$ comme la clôture par anti-réduction de l'ensemble

$$\left\{\, v_c * \pi_{c'} \;\middle|\; c \in \Gamma^-, c' \in \Gamma^+, \sigma(c) = \neg\sigma(c') \,\right\}$$

et définissons l'interprétation des variables propositionnelles par

$$[Z] = \{\pi\} \qquad\qquad [X] = \left\{\, \pi_{c'} \;\middle|\; c' \in \Gamma^+, \sigma(c') = X \,\right\}$$

où Z est la variable libre utilisée dans la traduction de Γ en un type et où X énumère les autres variables de Γ.

Soit c une clause de Γ^-. Pour toute famille de termes (u_L) indexée par c telle que $u_L \Vdash \neg L$ pour chaque L, par hypothèse $f_c * \vec{u} \cdot \pi$ se réduit en $u_{\sigma(c)} * v_c \cdot \pi'$

pour un certain π'. Par hypothèse sur c, le littéral $\sigma(c)$ est un $\neg X$, et le terme v_c réalise X, puisque pour chaque $\pi'_c \in [X]$ on a $\sigma(c) = \neg\sigma(c')$ et donc $v_c * \pi_{c'} \in \bot\!\!\!\bot$, donc $u_{\sigma(c)} * v_c \cdot \pi'$ est élément de $\bot\!\!\!\bot$, c'est donc aussi le cas de $f_c * \vec{u} \cdot \pi$ par anti-réduction, donc $f_c \Vdash c \to Z$. De la même façon, on démontre que pour toute clause $c' \in \Gamma^+$ on a $f_{c'} \Vdash c' \to Z$.

Par hypothèse, $\vdash t : \Gamma$ est dérivable, donc par le lemme d'adéquation on a $t \Vdash \Gamma$. Pour chaque clause $c \in \Gamma$ on a $f_c \Vdash c \to Z$, et par définition on a $\pi \in [Z]$, donc $\vec{f} \cdot \pi \in [\Gamma]$ et par conséquent $t * \vec{f} \cdot \pi$ est élément de $\bot\!\!\!\bot$, c'est-à-dire qu'il se réduit en un $v_c * \pi_{c'}$ avec $\sigma(c) = \neg\sigma(c')$. □

Malgré son énoncé un peu technique, cette spécification est en fait plutôt superficielle. En effet, les hypothèses du théorème imposent que les termes v_c et les piles π_c soient indépendantes des arguments $\vec{u}$ passés aux fonctions f_c, en d'autres termes on impose que les f_c utilisent exactement un de leurs arguments et de façon linéaire. Il est possible de raffiner la spécification en affaiblissant cette contrainte, donc en supposant qu'il existe une famille de termes $v_c[]$ et de piles $\pi_c[]$ paramétrés par les arguments $\vec{u}$, de sorte que pour chaque clause c on ait

$$\begin{array}{lll} \text{si } \sigma(c) \in V \text{ alors} & \forall \vec{u}\ \exists \pi' & f_c * \vec{u} \cdot \pi \to u_{\sigma(c)} * v_c[\vec{u}] \cdot \pi' \\ \text{si } \sigma(c) \in \neg V \text{ alors} & \forall \vec{u} & f_c * \vec{u} \cdot \pi \to u_{\sigma(c)} * \pi_c[\vec{u}] \end{array}$$

On démontre alors sans plus de difficulté qu'au dessus qu'il existe une paire de clauses c, c' avec $\sigma(c) = \neg\sigma(c')$ et des familles de termes $\vec{a}$ et $\vec{b}$ pour lesquelles l'exécutable $t * \vec{f} \cdot \pi$ se réduit en $v_c[\vec{a}] * \pi_{c'}[\vec{b}]$. Démontrer quelque chose sur ces familles de termes $\vec{a}$ et $\vec{b}$ est une affaire plus délicate, en particulier à cause du grand nombre de paramètres à considérer.

Pour illustrer la technique à appliquer, étudions de ce point de vue le cas simple du tiers exclu, donc $\Gamma = X \vee \neg X$. Sous forme de type, il s'écrit donc $\forall X \forall Y\, (X \to Y), (\neg X \to Y) \to Y$, comme expliqué plus haut. On peut alors énoncer le « développement limité » suivant :

Proposition 3.15. *Soit c un terme de type $\neg X \vee X$. Soit π une pile et soient f et g deux termes. Supposons qu'il existe une pile π_g et une famille de contextes $(v_i[])_{0 \leqslant i \leqslant n}$ tels que*

$$\begin{array}{ll} \forall \alpha\ \exists \pi' & f * \alpha \cdot \pi \to \alpha * v_0[\alpha] \cdot \pi' \\ \forall \alpha & g * \alpha \cdot \pi \to \alpha * \pi_g \\ \forall i < n\ \forall \alpha\ \exists \pi' & v_i[\alpha] * \pi_g \to \alpha * v_{i+1}[\alpha] \cdot \pi' \end{array}$$

Alors il existe un terme t tel que $c * f \cdot g \cdot \pi \to v_n[t] * \pi_g$.

Démonstration. On définit $\bot\!\!\!\bot$ comme la clôture de $\{\, v_n[t] * \pi_g \mid t \in \Lambda \,\}$ par anti-réduction et on instancie les variables propositionnelles par $[A] = \{\pi_g\}$ et $[B] = \{\pi\}$. Il en découle que pour tout terme t, le terme $v_n[t]$ réalise A. Soit alors un entier $i < n$. Supposons que pour tout α réalisant $\neg A$ le terme $v_{i+1}[\alpha]$ réalise $\neg A$. Pour tout tel α et toute pile π', l'exécutable $\alpha * v_{i+1}[\alpha] \cdot \pi'$ est élément de $\bot\!\!\!\bot$, donc par anti-réduction $v_i[\alpha] * \pi_g$ l'est également, ce qui prouve que $\alpha \Vdash \neg A$ implique $v_i[\alpha] \Vdash A$. Par récurrence, le résultat est donc prouvé pour tout le contexte $v_0[]$. Par suite, dès que α réalise $\neg A$, pour toute pile π' on a $\alpha * v_0[\alpha] \cdot \pi' \in \bot\!\!\!\bot$ et donc $f * \alpha \cdot \pi \in \bot\!\!\!\bot$, donc $f \Vdash \neg A \to B$. On prouve trivialement que g réalise le type $A \to B$, donc l'exécutable $c * f \cdot g \cdot \pi$ est élément de $\bot\!\!\!\bot$ et le résultat est démontré. □

Cette spécification s'applique donc au cas où les valeurs appliquées aux exceptions contiennent elles-mêmes des exceptions (sans pour autant lever la contrainte sur le contexte d'évaluation). Elle signifie que la réduction de $c{*}f\cdot g\cdot\pi$ mène à un exécutable équivalent à $v_0[k_{\pi_g}] * \pi_g$, c'est-à-dire que les termes que c place dans les contextes $v_i[]$ se comportent comme la continuation k_{π_g}, ce qui peut s'interpréter en disant que l'exception X peut être levée plusieurs fois mais qu'elle sera toujours rattrapée de la même façon par g.

3.2.2 Synthèse de combinateurs

Toutes les tautologies (sous forme normale disjonctive) sont bien sûr démontrables par le système de type du λc-calcul puisqu'il est équivalent à la logique propositionnelle classique, donc chaque tautologie est habitée par des termes typés. Cependant, en comparaison de la spécification du théorème 3.14, ces termes ont un comportement plutôt maladroit, et il peut donc être intéressant d'ajouter au calcul des combinateurs remplissant le même rôle de façon plus propre.

Une logique ad-hoc Afin de définir systématiquement de tels combinateurs, on développe ici une logique spécialisée, avec son calcul de séquents, pour démontrer les tautologies sons forme normale disjonctive. On utilisera ensuite une version spécialisée de la correspondance de Curry-Howard pour en extraire la structure calculatoire voulue.

Définition 3.16. La logique *dnf* a pour formules les formes normales disjonctives, selon les notations du paragraphe 3.11. Les séquents $\vdash_{dnf} \Gamma$ sont dérivés par les règles d'échange et de contraction et le schéma de règle n-aire suivant :

$$\frac{\vdash_{dnf} \Gamma_1, \Delta \qquad \cdots \qquad \vdash_{dnf} \Gamma_n, \Delta}{\vdash_{dnf} (\neg X_1 \wedge \cdots \wedge \neg X_n), (X_1 \wedge \Gamma_1), \ldots, (X_n \wedge \Gamma_n), \Delta}$$

où $X \wedge \Gamma$ représente la distribution de X sur toutes les clauses de Γ :

$$X \wedge (c_1, \ldots, c_n) := (X \wedge c_1), \ldots, (X \wedge c_n)$$

Dans le cas $n = 0$, on obtient la règle de la constante $\top$ (considérée comme la conjonction triviale), qui pose $\vdash_{dnf} \top, \Delta$ comme unique axiome. Par construction, ce système logique ne dérive que des formes normales disjonctives. La validité du système est facile à vérifier :

Proposition 3.17. *Si* $\vdash_{dnf} \Gamma$ *est dérivable, alors* Γ *est une tautologie.*

Démonstration. On utilise la caractérisation de la proposition 3.13, par induction sur la démonstration de $\vdash_{dnf} \Gamma$. Le cas $n = 0$ est trivial puisque la présence de la clause vide $\top$ dans Γ implique que l'ensemble des sections de Γ est vide. Sinon, soit σ une section de Γ. D'après la règle de dérivation des séquents, Γ contient une clause $\neg X_1 \wedge \cdots \wedge \neg X_n$, donc σ contient un littéral $\neg X_i$. Si la section σ ne contient pas le littéral X_i, elle doit choisir un autre littéral dans chaque clause de $X_i \wedge \Gamma_i$, ce qui définit une section de Γ_i, et donc σ contient une section de Γ_i, Δ. Par hypothèse d'induction, Γ_i, Δ est une tautologie, donc cette section contient deux littéraux opposés, et c'est donc le cas de σ. □

Pour démontrer la complétude de la logique *dnf*, on procède en partant d'une forme normale disjonctive supposée être une tautologie et on reconstruit une dérivation de cette formule, au moyen des deux lemmes suivants :

Lemme 3.18. *Toute tautologie sous forme normale disjonctive contient une clause composée uniquement de littéraux négatifs.*

Démonstration. Si toute clause contenait un littéral positif, il serait possible de de définir une section ne contenant pas de littéraux opposés, ce qui contredirait la proposition 3.13. □

Lemme 3.19. *Soient* Γ *une forme normale disjonctive et* X *une variable propositionnelle. Soit* $\Gamma/X = \{ c \setminus \{X\} \mid c \in \Gamma, \neg X \notin c \}$ *le quotient de* Γ *par* X*. Si* Γ *est une tautologie, alors* Γ/X *aussi.*

Démonstration. Soit v une valuation des variables de Γ/X. Par construction la variable X n'apparaît pas dans la formule Γ/X, donc on peut étendre v en une valuation v' telle que $v'(X) = \top$. Comme Γ est une tautologie, $v'(\Gamma)$ est vrai donc il existe une clause $c \in \Gamma$ telle que $v'(c)$ soit vrai. Comme $v'(X) = \top$, la clause c ne peut pas contenir $\neg X$, donc $c \setminus \{X\}$ apparaît dans Γ/X, or $v(c \setminus \{X\}) = \top$ donc la valuation v valide Γ/X. □

Comme il apparaît dans la démonstration précédente, Γ/X représente la formule Γ simplifiée sous l'hypothèse que la variable X est vraie, c'est pourquoi toute les clauses qui contiennent $\neg X$ sont supprimées et X est effacé des autres clauses. Par exemple, on a $\{\{X, Y\}, \{\neg X, Z\}, \{\neg Z\}\}/X = \{\{Y\}, \{\neg Z\}\}$. C'est cette opération qui permet de reconstruire les branches d'un arbre de preuve à partir de sa conclusion, ce qui permet de prouver la complétude du système *dnf*.

Proposition 3.20. *Pour toute tautologie* Γ *sous forme normale disjonctive,* $\vdash_{dnf} \Gamma$ *est dérivable.*

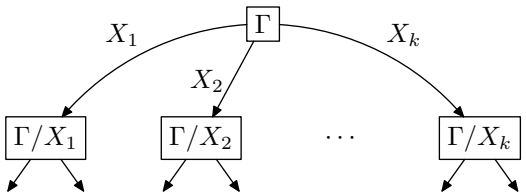
Démonstration. On démontre en fait, par récurrence sur le nombre de variables de Γ, que pour toute formule Δ le jugement $\vdash_{dnf} \Gamma, \Delta$ est dérivable. Comme Γ est une tautologie, le lemme 3.18 garantit l'existence d'une clause purement négative, donc on a $\Gamma = (\neg X_1 \wedge \cdots \wedge \neg X_n), \Gamma'$. Si $n = 0$ alors Γ contient $\top$ donc $\vdash_{dnf} \Gamma, \Delta$ est une instance de la règle de déduction d'arité nulle. Sinon, pour chaque i, on pose $\Gamma_i = \{ c \mid c \wedge X_i \in \Gamma, \neg X_i \notin c \}$ et $\Delta_i = \{ c \mid c \in \Gamma, \neg X_i \notin c \}$. Le quotient Γ/X_i est donc l'union de Γ_i et Δ_i, et par le lemme 3.19 on sait donc que c'est une tautologie. De plus cette formule contient une variable de moins que Γ puisque X_i y a été supprimée, donc $\vdash_{dnf} \Gamma_i, \Delta_i, \Delta'$ est dérivable pour tout Δ', par hypothèse d'induction. On a donc l'inférence suivante :

$$\frac{\vdash_{dnf} \Gamma_1, \Delta_1, \Delta' \qquad \cdots \qquad \vdash_{dnf} \Gamma_n, \Delta_n, \Delta'}{\vdash_{dnf} (\neg X_1 \wedge \cdots \wedge \neg X_n), \mathrm{X}_1 \wedge \Gamma_1, \ldots, X_n \wedge \Gamma_n, \Delta_1, \ldots, \Delta_n, \Delta'}$$

Tous les $X_i \wedge \Gamma_i$ et les Δ_i sont inclus dans Γ, Δ, donc en prenant pour Δ' l'ensemble des clauses de Γ, Δ manquantes, on déduit bien la formule voulue. □

Cette démonstration de complétude met en évidence la structure des démonstrations considérées : une dérivation de $\vdash_{dnf} \Gamma$ se termine par une règle qui introduit une clause « active » purement négative qui permet d'énumérer les branches de la démonstration, indexées par des littéraux positifs.

Interprétation interactive La structure des démonstrations dans la logique *dnf* peut s'interpréter comme une stratégie d'interaction entre une preuve d'une tautologie et une famille de contre-exemples potentiels, en quelque sorte à la manière d'une *no-counterexample interpretation* à la Kreisel [48]. Dans une démonstration d'une formule Γ qui se termine par l'introduction d'une clause $\neg X_1 \wedge \cdots \wedge \neg X_n$, on commence par jouer cette clause comme témoin de la validité de Γ. L'adversaire peut répondre par une réfutation de l'un des littéraux X_i, dans ce cas X_i est supposé vrai et l'interaction se poursuit par la branche qui démontre Γ/X_i. On représente ce mécanisme par le diagramme suivant :



où l'étiquette X_i sur une arête représente le passage d'une preuve de X_i, c'est-à-dire la réfutation du littéral $\neg X_i$ dans la clause active de Γ. La règle d'arité zéro est donc interprétée par un diagramme à un seul nœud étiqueté $\top$.

La stratégie commence donc par la racine du diagramme, en jouant la clause active de la formule indiquée, puis en empruntant les arêtes étiquetées par les littéraux que joue l'adversaire. Le fait que la formule initiale Γ soit une tautologie garantit que la construction est valide et que le processus parvient toujours à un nœud contenant $\top$ (où aucune réfutation n'est possible), c'est-à-dire que la validité de Γ garantit l'existence d'une stratégie gagnante. Comme c'est uniquement la clause active qui est utilisée dans la stratégie d'interaction, il suffit de mentionner celle-ci dans les nœuds du diagramme pour représenter toute la stratégie.

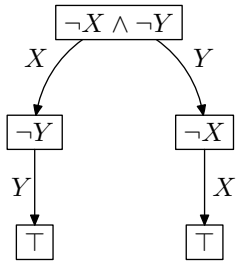
Exemple 3.21. Considérons l'exemple d'une forme de tiers-exclu à deux variables X et Y, c'est-à-dire la formule

$$\Gamma = \{\neg X \wedge \neg Y, X \wedge \neg Y, \neg X \wedge Y, X \wedge Y\}$$

La démonstration de cette formule (unique à permutations près) dans notre système s'écrit

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\top}\,\emptyset}{\neg Y, Y}\,\{Y\} \qquad \dfrac{\dfrac{}{\top}\,\emptyset}{\neg X, X}\,\{X\}}{\neg X \wedge \neg Y, X \wedge \neg Y, X \wedge Y, Y \wedge \neg X, Y \wedge X}\,\{X, Y\}}{\neg X \wedge \neg Y, X \wedge \neg Y, \neg X \wedge Y, X \wedge Y}\,\text{contraction}$$

Sous forme de diagramme, on écrit donc



La stratégie d'interaction décrite par ce diagramme peut s'implémenter en λc-calcul à l'aide d'un combinateur C_Γ défini par les règles suivantes :

$$\begin{aligned}
C_\Gamma * \vec{f} \cdot \pi &\to f_{\neg X \wedge \neg Y} * \alpha_X \cdot \alpha_Y \cdot \pi \\
\alpha_X * a \cdot \pi' &\to f_{X \wedge \neg Y} * a \cdot \alpha_{XY} \cdot \pi \\
\alpha_Y * b \cdot \pi' &\to f_{\neg X \wedge Y} * b \cdot \alpha_{YX} \cdot \pi \\
\alpha_{XY} * c \cdot \pi' &\to f_{X \wedge Y} * a \cdot c \cdot \pi \\
\alpha_{YX} * d \cdot \pi' &\to f_{X \wedge Y} * d \cdot b \cdot \pi
\end{aligned}$$

Dans ces notations, la famille de termes (f_c) représente l'opposant, elle correspond aux nœuds du diagramme, et la famille (α_L) correspond aux arêtes, chaque α_L correspond à la réfutation par l'opposant d'un littéral $\neg L$. La formulation ci-dessus n'est pas parfaitement précise car il y manque de l'information, en effet les α_L sont reliés à une instance de C_Γ, et ils connaissent les arguments initiaux $\vec{f}$ et π, les autres α_L et les valeurs qui leur ont été appliquées en cours d'interaction. Par exemple, le terme α_{XY} devrait en fait s'écrire $\alpha_{XY,\Gamma,\vec{f},\pi,a}$. Pour gagner en lisibilité, on n'écrira pas ce terme avec tout le détail.

Un autre point, plus subtil, est que dans la définition précédente il se peut que les termes a et b contiennent des occurrences de α_X et α_Y respectivement, et donc que ces exceptions peuvent êtres levées plus tard, alors que α_{XY} ou α_{YX} aura déjà été levée. Bien que ce comportement soit correct du point de vue du typage, il est possible de faire mieux en évitant ce comportement non-linéaire, en replaçant des lieurs κ lors du rattrapage des exceptions :

$$\begin{aligned}
\alpha_X * a \cdot \pi' &\to f_{X \wedge \neg Y} * (\kappa \alpha_X.a) \cdot \alpha_{XY} \cdot \pi \\
\alpha_Y * b \cdot \pi' &\to f_{\neg X \wedge Y} * (\kappa \alpha_Y.b) \cdot \alpha_{YX} \cdot \pi \\
\alpha_{XY} * c \cdot \pi' &\to f_{X \wedge Y} * (\kappa \alpha_X.a) \cdot (\kappa \alpha_{XY}.c) \cdot \pi \\
\alpha_{YX} * d \cdot \pi' &\to f_{X \wedge Y} * (\kappa \alpha_{YX}.d) \cdot (\kappa \alpha_Y.b) \cdot \pi
\end{aligned}$$

Le comportement obtenu est plus proche ainsi du traitement des exceptions qui existe dans les langages de programmation fonctionnels, où une exception donnée n'est rattrapée qu'une seule fois.

L'utilisation de cette re-liaison n'est pas une manipulation purement syntaxique, car elle correspond au sens de la spécification observée dans la proposition 3.15 : une exception $\alpha_X a$ est toujours rattrapée de la même façon, en appelant la fonction $f_{X \wedge \neg Y}$ avec a comme argument et sur la pile π. Si l'évaluation de cette fonction mène à $a * \pi_X$, alors appliquer α_X à une autre valeur a' dans le même contexte mènera à $a' * \pi_X$, donc remplacer α_X par k_{π_X} donnera le même comportement. On emploiera cet argument dans la démonstration du théorème 3.24 page 47.

On peut maintenant définir les combinateurs associés aux diagrammes de contrôle de façon générique. Formellement, un diagramme $\mathcal{D}$ de type Γ est exactement un arbre de preuve dans la logique *dnf* dont la conclusion est $\vdash_{dnf} \Gamma$. On appelle *clause active* de $\mathcal{D}$ la clause totalement négative introduite par la dernière règle de l'arbre, et on note $\mathcal{D}/X$ le sous-arbre de $\mathcal{D}$ correspondant au littéral $\neg X$ dans cette clause active.

Définition 3.22. Soit $\mathcal{D}$ une diagramme de contrôle de type Γ et de clause active $c = \neg X_1 \wedge \cdots \wedge \neg X_n$. L'implémentation de $\mathcal{D}$ est définie par induction à

partir de celle des $\mathcal{D}/X_i$ par la constante $C_\mathcal{D}$ et les constantes α_{X_i} munies des règle de réduction suivantes :

$$C_\mathcal{D} * \vec{f} \cdot \pi \to f_c * \alpha_{X_1} \cdots \alpha_{X_n} \cdot \pi$$
$$\alpha_{X_i} * v \cdot \pi' \to C_{\mathcal{D}/X_i} * \vec{f'} \cdot \pi$$

où $\vec{f}$ est indexé par les clauses de Γ, et où $\vec{f'}$ est indexé par les clauses du type de $\mathcal{D}/X_i$ et défini par

$$\begin{aligned} f'_c &= f_c && \text{si } c \in \Gamma \\ f'_c &= f_{X_i \wedge c}(\kappa x.v[x/\alpha_{X_i}]) && \text{si } X_i \wedge c \in \Gamma \end{aligned}$$

pour une variable fraîche x, en supposant (sans perte de généralité) que l'argument de type X_i de $f_{X_i \wedge c}$ est le premier.

Pour être parfaitement précise, cette définition devrait écrire $\alpha_{\mathcal{D},X_i,\vec{f},\pi}$ au lieu de α_{X_i}, car la réduction de ces constantes dépend du contexte où $C_\mathcal{D}$ est appliqué initialement. De plus, il est nécessaire de pouvoir identifier dans $\kappa x.v[x/\alpha_{X_i}]$ quelles sont les constantes α_{X_i} à remplacer par x, car seules celles produites par $C_\mathcal{D}$ sont concernées par cette substitution. Ces informations supplémentaires seront gardées implicites partout où elles seront claires à partir du contexte, les notations étant déjà suffisamment lourdes.

Comme expliqué dans l'exemple 3.21 page précédente, le terme $\kappa x.v[x/\alpha_{X_i}]$ a pour effet de substituer pour l'exception α_{X_i} le premier contexte où le terme v sera employé, fixant la façon de rattraper l'exception α_{X_i} dans la suite du calcul. Syntaxiquement, cette construction garantit que chaque exception est levée au plus une fois.

Le cas de base dans cette définition par induction est celui d'une DNF qui contient la clause $\top$ en position active. Dans ce cas la règle de réduction du combinateur $C_\mathcal{D}$ s'écrit simplement

$$C_\mathcal{D} * \vec{f} \cdot \pi \to f_\top * \pi$$

Il s'agit donc d'une projection, construction définissable en λ-calcul pur. En effet, les DNF contenant la clause triviale $\top$ sont exactement celles qui sont démontrables en logique intuitionniste.

3.2.3 Correction

On a donc défini formellement une famille de combinateurs à partir de la structure des formes normales disjonctives, il reste à démontrer qu'ils réalisent bien le type voulu. La re-liaison dynamique, nécessaire pour implémenter le comportement attendu, nécessite une restriction sur la forme des observations afin d'obtenir la réalisation voulue.

Définition 3.23. Une observation $\perp\!\!\!\perp$ est dite *régulière* si elle est close par réduction et vérifie la propriété de substitution : pour tout contexte e et tout terme t tel que $e[t] \in \perp\!\!\!\perp$, si la réduction de $e[x]$ ne place pas x en tête alors $e[u] \in \perp\!\!\!\perp$ pour tout terme u.

Ces contraintes apparaissent naturellement dans la démonstration de correction des combinateurs synthétiques, mais elles peuvent se justifier indépendamment. La clôture par réduction rend compte du fait que les propriétés que l'on considère sont liées à l'évaluation, c'est-à-dire qu'on n'observe que l'état final du calcul, ce qui est nécessaire parce que les combinateurs synthétiques prennent des libertés avec l'évaluation des termes. La propriété de substitution signifie simplement que si un sous-terme de l'exécutable n'est jamais placé en tête, il ne doit pas intervenir dans l'observation, puisque le principe de l'orthogonalité est fondamentalement d'observer l'interaction entre un terme et son contexte (qui est purement dictée par la forme du terme en position de tête), en ne tenant pas compte des détails syntaxiques de la définition des termes.

Théorème 3.24. *Soit* Γ *une tautologie en forme normale disjonctive, et soit* $\mathcal{D}$ *un diagramme de contrôle de type* Γ*. Pour tout* $\perp\!\!\!\perp$ *régulier, on a* $C_{\mathcal{D}} \Vdash \Gamma$*.*

Démonstration. On procède par induction sur la structure du diagramme, pour un $\perp\!\!\!\perp$ fixé non vide (si $\perp\!\!\!\perp$ est vide, le résultat est trivial puisque Γ est prouvable et que dans ce cas $[\![\Gamma]\!] = \Lambda$). Si $\mathcal{D}$ est une feuille, on a $\Gamma = \top, c_1, \dots, c_n$, qui correspond au type $\forall Z\, Z, (c_1 \to Z), \dots, (c_n \to Z) \to Z$. Selon la définition 3.22, on a donc $C_{\mathcal{D}} * \vec{f} \cdot \pi \to f_\top * \pi$, donc pour tout $\perp\!\!\!\perp$ et toute valeur de $[Z]$, si $f_\top \Vdash Z$ alors $C_{\mathcal{D}} * \vec{f} \cdot \pi$ est élément de $\perp\!\!\!\perp$, ce qui prouve que $C_{\mathcal{D}}$ réalise Γ, indépendamment des autres variables qui y apparaissent.

Supposons donnée une valuation $[X] \subseteq \Pi$ pour chaque variable propositionnelle, et soit Z le type de retour (introduit dans la traduction de Γ en un type). Pour chaque clause $c' \in \Gamma$, sont $f_{c'}$ un terme réalisant $c' \to Z$. Notons $c = \neg X_1 \wedge \dots \wedge \neg X_n$ la clause active de $\mathcal{D}$, par définition $C_{\mathcal{D}} * \vec{f} \cdot \pi$ se réduit en $f_c * \alpha_{X_1} \cdots \alpha_{X_n} \cdot \pi$, il suffit donc de montrer que le réduit est élément de $\perp\!\!\!\perp$.

Si la réduction de cet exécutable ne place aucun des α_{X_i} en tête, alors $f_c * \vec{u} \cdot \pi$ ignorera les termes $\vec{u}$ quels qu'ils soient. En prenant $u_i \Vdash X_i$ pour chaque i (ce qui est possible car les X_i ne sont pas vides, puisque $\perp\!\!\!\perp$ ne l'est pas), on a donc $f * \vec{u} \cdot \pi \in \perp\!\!\!\perp$, d'où on déduit $f * \vec{\alpha} \cdot \pi \in \perp\!\!\!\perp$ par la propriété de substitution, d'où $C_{\mathcal{D}} * \vec{f} \cdot \pi \in \perp\!\!\!\perp$.

Si la réduction de $f_c * \alpha_{X_1} \cdots \alpha_{X_n} \cdot \pi$ place en tête un certain α_X, alors il existe un contexte de terme $v[]$ et une pile π' tels qu'on ait la réduction

$$f_c * \vec{\alpha} \cdot \pi \to \alpha_X * v[\alpha_X] \cdot \pi' \to C_{\mathcal{D}/X} * \vec{f'} \cdot \pi$$

avec les notations de la définition 3.22. Par hypothèse d'induction, on sait que $C_{\mathcal{D}/X}$ réalise le type Δ de $\mathcal{D}/X$, donc il suffit de démontrer que f'_d réalise $d \to Z$ pour chaque clause $d \in \Delta$. Pour une telle clause d, si $d \in \Gamma$ alors $f'_d = f_d$ et $f_d \Vdash d \to Z$ est vrai par hypothèse. Sinon $X \wedge d$ est une clause de Γ et on a $f'_d = f_{X \wedge d}(\kappa x.v[x])$. Par hypothèse $f_{X \wedge d} \Vdash X \to d \to Z$ donc il suffit de montrer $\kappa x.v[x] \Vdash X$.

Notons $c = \neg X \wedge c'$ la clause active de $\mathcal{D}$, en supposant (sans perte de généralité) que $\neg X$ est le type de premier argument de f_c, et étudions la réduction qui produit le contexte $v[]$. Soit π_X une pile de $[X]$, alors clairement $k_{\pi_X} \Vdash \neg X$, donc $f_c k_{\pi_X}$ réalise $c' \to Z$, donc on a les deux réductions convergentes suivante, où $\vec{\alpha}$ est une famille quelconque indexée par c' telle que $\alpha_L \Vdash L$ pour chaque $L \in c'$:

$$\begin{aligned} f_c * k_{\pi_X} \cdot \vec{\alpha} \cdot \pi \to k_{\pi_X} * v[k_{\pi_X}] \cdot \pi' &\to v[k_{\pi_X}] * \pi_X \\ \kappa x.v[x] * \pi_X &\to v[k_{\pi_X}] * \pi_X \end{aligned}$$

Notons que, puisque α_X est frais lorsqu'il est appliqué à f_c, les occurrences de α_X dans $\alpha_X\, v[\alpha_X]$ sont exactement les résidus de cet α_X, ce qui justifie la première réduction. On a $f_c \Vdash X \to c' \to Z$, $k_{\pi_X} \Vdash \neg X$, $\pi \in [Z]$ et $\alpha_L \Vdash L$ pour chaque $L \in c'$ donc le premier exécutable est élément de $\bot\!\!\!\bot$. La clôture par réduction implique donc $v[k_{\pi_X}] * \pi_X \in \bot\!\!\!\bot$ et par anti-réduction $\kappa x.v[x]$ est orthogonal à π_X, donc il réalise X, ce qui permet de conclure. □

Par conséquent, on peut étendre le λc-calcul avec ces combinateurs $C_{\mathcal{D}}$ et attribuer le type Γ à $C_{\mathcal{D}}$ si $\mathcal{D}$ est de type Γ, en conservant la cohérence du système de types et de la sémantique. Une façon de démontrer que la cohérence du système est préservée est d'exprimer une spécification qui montre clairement que rien ne s'est effondré, par exemple en montrant que les types des booléens spécifient toujours deux comportements disjoints, comme c'était le cas dans la proposition 2.12 page 19. Rappelons que pour $b \in \{0,1\}$ on pose $B_b = \forall X_0 X_1 (X_0, X_1 \to X_b)$.

Proposition 3.25. *Pour tout $b \in \{0,1\}$, tout terme t de type B_b, tous termes u_0 et u_1 et toute pile π on a $t * u_0 \cdot u_1 \cdot \pi \to u_b * \pi$.*

Démonstration. Soient x_0 et x_1 deux variables fraîches et ρ une constante de pile. Soit $\bot\!\!\!\bot$ la clôture par anti-réduction de $\{x_b * \rho\}$, qui est clairement une observation régulière. Par hypothèse $\vdash t : B_b$ est dérivable, donc $t \Vdash B_b$, et pour toute valeur de $[X_0]$ et $[X_1]$ on a $t \bot\!\!\!\bot [X_0, X_1 \to X_b]$. Posons $[X_b] = \{\rho\}$ et $[X_{\neg b}] = \emptyset$, alors $x_0 \Vdash X_0$ et $x_1 \Vdash X_1$, donc $x_0 \cdot x_1 \cdot \rho \in [X_0, X_1 \to X_b]$ et $t * x_0 \cdot x_1 \cdot \rho$ est élément de $\bot\!\!\!\bot$, donc il se réduit en $x_b * \rho$. Même en présence des nouveaux combinateurs, la réduction est préservée par substitution de variables et de constantes de pile, donc en substituant chaque u_i pour x_i et π pour ρ on déduit que $t * u_0 \cdot u_1 \cdot \pi$ se réduit en $u_b * \pi$. □

Les autres théorèmes de spécification restent valides de la même façon, et la technique s'étend sans modification lorsque le calcul est enrichi de types de données à la façon de la section 2.2. Ainsi on obtient une méthode systématique pour spécifier des structures de contrôle local bien typées dans un langage du style PCF.

Un peu de syntaxe La programmation combinatoire est élégante et modulaire d'un point de vue sémantique, mais délicate à comprendre faute de syntaxe expressive. Dans le but de clarifier le fonctionnement des combinateurs $C_{\mathcal{D}}$ définis plus haut, on peut donner une syntaxe qui serait aux combinateurs ce que le lieur $\kappa x.t$ est à la constante cc, dans le style d'un **try** ... **with** à la ML. On écrira

$$\begin{array}{llll} \textbf{sync} & \textbf{catch}\ C_1^1 x_1, \ldots, C_{p_1}^1 x_{p_1} & \textbf{throw}\ T_1^1, \ldots, T_{n_1}^1 & \textbf{do}\ t_1 \\ & & \vdots & \\ & \textbf{catch}\ C_1^k x_1, \ldots, C_{p_k}^k x_{p_k} & \textbf{throw}\ T_1^k, \ldots, T_{n_k}^k & \textbf{do}\ t_k \end{array}$$

pour représenter l'application d'un $C_{\mathcal{D}}$ aux termes $t_1, \ldots, t_k$ précédés des abstractions appropriées. Les T_j^i et C_j^i sont des noms d'un ensemble d'identificateurs qui correspond aux variables de la formule Γ :

– T_j^i est à voir comme un exception, c'est-à-dire un objet d'un type négatif $\neg X$ qui peut être appliqué à un terme v pour lancer une exception, en

écrivant $(T\,v)$ ou **raise** $(T\,v)$ pour souligner le rapport avec le mécanisme de contrôle ;
– $C^i_j x_j$ se voit comme une co-exception, c'est-à-dire un objet qui reçoit une valeur d'un type positif X émis par le déclenchement d'une exception du même nom C^i_j (et de type $\neg X$) dans un autre partie de la structure, la valeur étant substituée pour la variable x_j dans le terme t_i.

Le type du combinateur considéré est donc la formule

$$\Gamma = \bigvee_{i=1}^{k} \Big(\bigwedge_{j=1}^{n_k} \neg T^i_j \wedge \bigwedge_{j=1}^{p_k} C^i_j \Big)$$

Le diagramme $\mathcal{D}$ est construit en partant de la première clause totalement négative mentionnée dans la structure, avec comme ramifications les structures obtenus par quotient par chaque variable de cette clause active. Le fait qu'une structure traitée de cette façon donne une preuve valide de Γ fait partie de la vérification du typage, et en cas de succès du typage il est garanti par construction qu'aucune exception ne peut s'échapper. Dans le cadre d'un langage de programmation, il est nécessaire d'employer des identificateurs pour les variables propositionnelles, faute de quoi l'inférence de type serait insuffisante pour retrouver la structure précise du diagramme, en particulier parce qu'il est possible que des exceptions distinctes servent à passer des valeurs du même type.

Illustrons cette suggestion de syntaxe en reprenant les exemples mentionnés précédemment. Dans le cas du tiers-exclu à deux variables de l'exemple 3.21, la syntaxe

sync		**throw** A, B	**do** t_1
	catch Ax	**throw** B	**do** t_2
	catch By	**throw** A	**do** t_3
	catch Ax, By		**do** t_4

représente l'application suivante du combinateur :

$$C_{\mathcal{D}}(\lambda A.\lambda B.t_1)(\lambda x.\lambda B.t_2)(\lambda y.\lambda A.t_3)(\lambda x.\lambda y.t_4)$$

La sémantique de cette structure consiste à évaluer t_1 et renvoyer sa valeur, sauf si l'une des deux exceptions est levée. Si A est levée, c'est-à-dire si l'évaluation de t_1 mène à **raise** $(A\,a)$, elle est rattrapée et la valeur a est substituée à x dans t_2 et t_4, alors que les clauses t_1 et t_3 (susceptibles de lever A) sont supprimées. L'évaluation continue donc sur

sync	**throw** B	**do** $t_2[a/x]$
	catch By	**do** $t_4[a/x]$

Notons que dans ce cas particulier, dans la stratégie qui consiste à commencer par les clauses purement négatives, il n'y a qu'un seul combinateur possible, et l'ordre des clauses est donc purement cosmétique.

On remarque que la forme obtenue dans le cas du tiers-exclu simple s'écrit

sync	**throw** *Exn*	**do** *body*
	catch *Exn x*	**do** *handler*

qui rappelle la forme usuelle d'un

try *body* **with** *Exn x* $\rightarrow$ *handler*

à la différence importante près que l'exception *Exn* est ici locale, et que la structure est bien typée.

En conclusion, on a donc décrit le comportement des formes normales disjonctives comme des schémas d'ordonnancement entre plusieurs branches d'exécution possibles qui correspondent aux différentes clauses d'une formule. Il y a malgré tout un point qui n'est pas satisfaisant : une tautologie donnée est réalisable par de nombreux comportements différents, qui apparaissent comme différentes façons de séquentialiser l'exécution parallèle des mêmes processus. Si l'on considère que les différentes clauses d'une structure d'exceptions s'exécutent en parallèle, les étapes de rattrapage d'exception s'interprètent plutôt comme la synchronisation de deux processus avec passage de données. Il semblerait donc naturel que les comportements associés aux formes normales disjonctives, et plus généralement à la disjonction classique, se décrivent avantageusement au moyen d'un calcul concurrent. On va donc maintenant s'attacher à définir une réalisabilité, similaire à celle employée ici, mais pour une variante du π-calcul. Comme le π-calcul permet de décomposer le λ-calcul et ses variantes, on utilisera une logique qui permet de décomposer la logique intuitionniste et classique : la logique linéaire.

Chapitre 4

Le π-calcul polarisé

Dans ce chapitre, on définit une forme de calcul concurrent à passage de noms dans la lignée du π-calcul polyadique. Ce calcul servira de langage de référence pour l'étude logique qui suit, comme le λc-calcul dans les chapitres précédents. Les modifications par rapport aux calculs existants sont destinées à gagner en symétrie et en régularité, afin d'obtenir un bon formalisme pour une étude logique. Ces modifications, qui sont discutées plus précisément dans la section 4.4, ont pour la plupart été déjà étudiées indépendamment :

1. Toutes les actions d'émission et réception sont des lieurs, ce qui rend les communications symétriques et régulières. Cette restriction appliquée au π-calcul, et qualifiée de *mobilité interne*, a notamment été étudiée par Sangiorgi [74] sous le nom de Private π ou πI.
2. Pour retrouver l'expressivité du passage de noms, on introduit la notion de routeur qui sert à introduire explicitement des connexions entre canaux. Cette notion combine celle des fusions explicites de Gardner et Wischik [34] avec l'idée de connexion orientée, étudiée par les mêmes auteurs avec Laneve [33] à des fins de programmation. Ces intuitions concrètes sur le routage prennent un sens également très clair dans le cadre logique, en permettant un contrôle fin de l'unification des noms.
3. Par symétrie, on n'impose pas aux actions de réception d'être bloquantes. Cette extension, initialement apparue dans le Fusion Calculus de Parrow et Victor [69] et le χ-calcul de Fu [32], puis exploitée plus avant par Laneve et Victor dans le calcul des solos [57], est une conséquence naturelle des deux aspects précédents.

Enfin, on imposera par typage une contrainte supplémentaire selon laquelle toutes les communications se passent sur des canaux privés, ce qui donnera une distinction claire entre les noms privés, où peuvent se produire des interactions, et les noms publics, où un processus donné ne peut faire que des émissions ou que des réceptions. Il ne s'agit que d'une forme de prototypage rudimentaire pour structurer les processus, le véritable typage est celui qu'on définira dans le chapitre 5 et qui servira à décrire le comportement des processus.

Ces modifications par rapport aux autres calculs ne restreignent pas l'expressivité du langage, mais elles ont pour but de lui donner une sémantique opérationnelle adaptée à notre étude. On verra en particulier qu'il est possible de coder le π-calcul ordinaire dans notre formalisme.

4.1 Termes et réductions

On commence par définir le langage de termes du π-calcul polarisé, ou $\vec{\pi}$-calcul, qui sera utilisé dans la suite comme langage de référence pour décrire des systèmes concurrents et d'interaction, lorsqu'on étudiera des variations sur ce langage et des reformulations plus abstraites des processus. Au cœur de la notion d'interaction telle qu'on la considère ici se trouve l'idée de polarité :

Définition 4.1. L'ensemble des polarités est $\mathbf{P} = \{\downarrow, \uparrow\}$. Pour $\varepsilon \in \mathbf{P}$, on note $\bar{\varepsilon}$ ou $\neg\varepsilon$ l'autre polarité.

Les polarités servent à identifier l'éternelle dualité entre les deux parties d'une interaction que sont le *joueur* et l'*adversaire*, le programme et l'environnement, Alice et Bob, ∃loïse et ∀belard. Par convention on dit que ↓ est positive et correspond aux capacités de réception, et que ↑ est négative et correspond aux capacités d'émission. Ce choix est assez arbitraire, mais il est motivé par l'analogie entre l'exponentielle $!A$ de la logique linéaire, qui est un changement de polarité positif, et le mécanisme usuel de réplication gardée par une réception $!u(x).P$.

L'involution canonique, notée indifféremment $\neg x$ ou $\bar{x}$, s'étendra à tous les objets munis d'une polarité. Il est parfois utile de considérer une inversion de polarité optionnelle, ainsi si X est un objet sur lequel peut s'appliquer une inversion de polarité, en l'absence d'autre définition on notera X^ε l'objet X si $\varepsilon = \uparrow$ et $\neg X$ si $\varepsilon = \downarrow$. Cette définition s'applique notamment à $\mathbf{P}$ lui-même, ce qui revient à le munir d'une structure de groupe avec ↑ pour neutre et ↓ involutif.

Définition 4.2. On suppose donné un ensemble dénombrable $\mathbf{N}$ de noms de canaux, désignés par les minuscules latines. Les actions (minuscules grecques) et les processus (majuscules latines) sont engendrés par la grammaire suivante :

actions :	$\alpha :=$	$u^\varepsilon(x_1 \dots x_n)$	avec $\varepsilon \in \mathbf{P}$
branchements :	$S, T :=$	1	arrêt
		$\alpha.P$	action bloquante
		$S + T$	choix externe
		$!S$	réplication
termes :	$P, Q :=$	S	processus séquentiel
		$u{\to}v$	routeur
		$P \mid Q$	composition
		$(\boldsymbol{\nu} x)P$	restriction

Dans une action $u^\varepsilon(x_1 \dots x_n)$, les noms x_i sont supposés deux à deux distincts. Une telle action est un lieur pour les noms en x_i, $(\boldsymbol{\nu} x)$ est lieur pour x, et les termes sont considérés à renommage près des variables liées. On note Σ l'ensemble des branchements et Π l'ensemble des processus (ou $\vec{\pi}$-termes).

Pour alléger l'écriture, une suite de noms $x_1 \dots x_n$ sera notée comme un vecteur $\vec{x}$, et on emploiera la notation $(\boldsymbol{\nu}\vec{x})P = (\boldsymbol{\nu} x_1) \cdots (\boldsymbol{\nu} x_n)P$. Les actions de réception seront en général notées $u(\vec{x})$, et les actions d'émission seront donc

notées $\bar{u}(\vec{x})$, selon les conventions énoncées plus haut et en accord avec la tradition du π-calcul. La notation $\sum_{i\in I} S_i$ représente la somme des branchements S_i pour chaque $i \in I$, quels que soient l'ordre et le parenthésage. De même, on notera $\prod_{i\in I} P_i$ toute composition parallèle des processus P_i.

L'involution $\neg$ s'étend aux processus en inversant la polarité de toutes les actions et en posant $\neg(u{\rightarrow}v) = v{\rightarrow}u$. Cette notation sera utile en particulier sous la forme $u{\rightarrow}^{\varepsilon}v$ qui désigne une connexion de u vers v si $\varepsilon = \uparrow$ ou de v vers u si $\varepsilon = \downarrow$. Cette convention est justifiée par l'équivalence, définie formellement plus loin (voir définition 4.12 et proposition 4.25) entre $(\boldsymbol{\nu}u)(u^{\varepsilon}(x).P \mid u{\rightarrow}^{\varepsilon}v)$ et $v^{\varepsilon}(x).P$: un routeur de polarité ε fait suivre des actions de polarité ε.

4.1.1 Sémantique par réduction

La définition la plus naturelle de la sémantique opérationnelle du $\vec{\pi}$-calcul se fait au moyen d'une relation de réduction définie modulo congruence structurelle. Dans la section suivante on donne une formulation équivalente sous forme de système de transition étiqueté, plus adaptée au raisonnement compositionnel.

Définition 4.3. La congruence structurelle $\equiv$ est la relation d'équivalence sur Σ et sur Π engendrée par les règles de la table 4.1.

La congruence structurelle est la relation qui définit la géométrie des termes, car la syntaxe pure donne une description beaucoup plus rigide que ce que veut l'intuition. Les axiomes de monoïde commutatif pour la somme et la composition et les règles de portée des noms sont parfaitement standard, la règle de la réplication l'est un peu moins. Elle permet de développer une réplication tout en conservant un branchement séquentiel, à l'inverse de la méthode classique qui consiste à poser $!P \equiv !P \mid P$, et qui s'étend moins bien en présence de sommes. Il s'agit en fait d'une forme spécialisée d'opérateur de récursion qui consiste à poser $!\alpha.P := \operatorname{rec} X\ \alpha.(P \mid X)$; en effet, dans le cas le plus simple, la règle s'écrit $!\alpha.P \equiv \alpha.(P \mid !\alpha.P)$. Dans la suite on se contentera néanmoins d'une réplication gardée, car elle est plus pratique à manipuler que la réplication (en particulier pour le typage), tout en ayant la même expressivité.

L'autre partie un peu plus inhabituelle de la congruence structurelle concerne les routeurs, les deux règles expriment en fait la réflexivité et la transitivité de la relation de connexion entre canaux de communication. Moyennant ces règles, la sémantique opérationnelle du calcul est définie par une unique règle de réduction :

Définition 4.4. La relation de réduction $\rightarrow$ sur Π est la plus petite relation close par $\equiv$, qui commute avec la composition et la restriction et qui contient

$$P' + \bar{u}(\vec{x}).P \mid u{\rightarrow}v \mid v(\vec{x}).Q + Q' \quad \rightarrow \quad (\boldsymbol{\nu}\vec{x})(P \mid Q) \mid u{\rightarrow}v$$

Deux actions élémentaires peuvent donc communiquer si elles sont connectées par des routeurs, orientés de l'émetteur vers le récepteur. Notons que le routeur reste présent dans le terme après réduction, c'est-à-dire en termes concrets qu'on travaille sous l'hypothèse que la connexion est permanente, et fiable dans la mesure où une émission $\bar{u}(x).P$ ne peut être consommée que s'il y a effectivement une réception correspondante.

Lois de monoïde commutatif pour la somme :

$$S + T \equiv T + S \qquad (S + T) + U \equiv S + (T + U) \qquad S + 1 \equiv S$$

Développement des réplications :

$$!\textstyle\sum_{i\in I} \alpha_i.P_i \equiv \sum_{j\in I} \alpha_j.\big(P_j \mid !\sum_{i\in I} \alpha_i.P_i\big)$$

Lois de monoïde commutatif pour la composition :

$$P \mid Q \equiv Q \mid P \qquad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad P \mid 1 \equiv P$$

Règles de portée, où $x \notin \mathrm{fv}(P)$:

$$(\boldsymbol{\nu} x)P \equiv P \qquad (\boldsymbol{\nu} y)(\boldsymbol{\nu} z)P \equiv (\boldsymbol{\nu} z)(\boldsymbol{\nu} y)P \qquad (\boldsymbol{\nu} x)(P \mid Q) \equiv P \mid (\boldsymbol{\nu} x)Q$$

Réflexivité et transitivité des connexions :

$$x{\to}x \equiv 1 \qquad\qquad x{\to}y \mid y{\to}z \equiv x{\to}y \mid y{\to}z \mid x{\to}z$$

Passage au contexte :

$$\frac{P \equiv P'}{\alpha.P \equiv \alpha.P'} \qquad \frac{S \equiv S'}{!S \equiv !S'} \qquad \frac{S \equiv S' \qquad T \equiv T'}{S + T \equiv S' + T'}$$

$$\frac{P \equiv P'}{P \mid Q \equiv P' \mid Q} \qquad \frac{P \equiv P'}{(\boldsymbol{\nu} x)P \equiv (\boldsymbol{\nu} x)P'}$$

Règle de réduction :

$$P' + \bar{u}(\vec{x}).P \mid u{\to}v \mid v(\vec{x}).Q + Q' \quad \to \quad (\boldsymbol{\nu}\vec{x})(P \mid Q) \mid u{\to}v$$

TAB. 4.1 – Congruence structurelle et réduction du $\vec{\pi}$-calcul.

Exemple 4.5. Un routeur $u{\to}v$ peut se voir comme une sorte de substitution de nom orientée $[\bar{v}/\bar{u}, u/v]$. L'intuition est qu'une émission $\bar{u}(x).P$ connectée à un routeur $u{\to}v$ doit se comporter comme l'émission $\bar{v}(x).P$. Par symétrie, une réception $v(y).Q$ connectée à un routeur $u{\to}v$ se comporte comme comme la même réception déplacée sur le canal u.

Exemple 4.6. Des connexions peuvent être établies entre des noms privés et des noms publics. Ainsi, le terme $(\boldsymbol{\nu}x)(\bar{x} \mid x{\to}y)$ peut lancer l'action $\bar{y}$ sur le nom public y, même si l'action d'émission $\bar{x}$ qu'il contient est située sur un canal privé. La règle d'extension de portée s'applique lors de l'établissement d'une connexion, de sorte que $P := (\boldsymbol{\nu}x)(\bar{x}().1 \mid x{\to}y) \mid (\boldsymbol{\nu}z)(y{\to}z \mid z().1)$ a une réduction qui s'écrit

$$\begin{aligned}
P &= (\boldsymbol{\nu}x)(\bar{x}().1 \mid x{\to}y) \mid (\boldsymbol{\nu}z)(y{\to}z \mid z().1) \\
&\equiv (\boldsymbol{\nu}xz)(\bar{x}().1 \mid x{\to}y \mid y{\to}z \mid z().1) && \text{extension de portée} \\
&\equiv (\boldsymbol{\nu}xz)(\bar{x}().1 \mid x{\to}y \mid y{\to}z \mid x{\to}z \mid z().1) && \text{transitivité} \\
&\equiv (\boldsymbol{\nu}xz)(\bar{x}().1 \mid x{\to}z \mid z().1 \mid x{\to}y \mid y{\to}z) && \text{associativité, commutativité} \\
&\to (\boldsymbol{\nu}xz)(1 \mid 1 \mid x{\to}z \mid x{\to}y \mid y{\to}z) && \text{réduction}
\end{aligned}$$

C'est cette capacité des routeurs qui permet de retrouver le comportement d'un calcul avec communication de noms publics. Ainsi on peut écrire la réduction suivante :

$$\begin{aligned}
Q &= u(x).\bar{x}(y).R \mid \bar{u}(z).z{\to}v \mid v(y).S \\
&\equiv u(x).\bar{x}(y).R \mid \bar{u}(x).x{\to}v \mid v(y).S && \text{renommage du nom lié } z \\
&\to (\boldsymbol{\nu}x)(\bar{x}(y).R \mid x{\to}v) \mid v(y).S \\
&\equiv (\boldsymbol{\nu}x)(\bar{x}(y).R \mid x{\to}v \mid v(y).S) && \text{extension de portée} \\
&\to (\boldsymbol{\nu}x)((\boldsymbol{\nu}y)(R \mid S) \mid x{\to}v)
\end{aligned}$$

Ce terme Q se comporte donc comme un processus qui, en π-calcul standard, utiliserait une émission de nom libre : $u(x).(\boldsymbol{\nu}y)(\bar{x}\langle y\rangle.R) \mid \bar{u}\langle v\rangle \mid v(y).S)$.

Exemple 4.7. Un routeur n'a d'effet que s'il est en position active, c'est-à-dire s'il n'est pas derrière un préfixe. Considérons par exemple dans le terme $P := u(x).a{\to}b \mid \bar{a}() \mid b() \mid \bar{u}(y)$. La seule réduction possible se fait par communication sur u, en écrivant $P \equiv \bar{u}(y) \mid u{\to}u \mid u(x).a{\to}b \mid \bar{a}() \mid b()$ par la règle de réflexivité, donc P se réduit en $a{\to}b \mid \bar{a}() \mid b()$, et ce n'est qu'après cette réduction que $\bar{a}()$ et $b()$ peuvent interagir.

Exemple 4.8. La présence d'un routeur $u{\to}v$ dans un processus n'implique pas que toute émission $\bar{u}(x)$ doive être renommée en $\bar{v}(x)$. Considérons par exemple le processus

$$P := \bar{u}(x) \mid u{\to}v \mid u{\to}w \mid v(x).Q \mid w(x).R$$

L'émetteur $\bar{u}(x)$ peut ici se synchroniser soit avec $v(x).Q$, soit avec $w(x).R$. En quelque sorte, les routeurs $u{\to}v$ et $u{\to}w$ interfèrent sur le nom u, mais cette interférence ne doit pas mener à un blocage.

En cela les routeurs du $\vec{\pi}$-calcul se différencient donc de processus « répéteurs » que l'on pourrait exprimer en π-calcul standard par $!u(\vec{x}).\bar{v}\langle\vec{x}\rangle$. En effet, un tel agent peut se synchroniser avec un émetteur $\bar{u}(\vec{y}).P$, en libérant la continuation P, même s'il n'y a pas de récepteur sur le canal v. En revanche, il est

connu que de tels agents agissent effectivement comme des substitutions dans le fragment asynchrone du π-calcul. On se reportera à la section 4.4.1 pour une discussion sur le sujet.

4.1.2 Système de transition étiqueté

La sémantique opérationnelle par réduction exposée plus haut donne une formulation naturelle de la dynamique du $\vec{\pi}$-calcul. Il y a cependant quelques inconvénients à utiliser une définition à congruence structurelle près. D'une part, le raisonnement par induction sur la structure des termes est délicat, car la sémantique n'est pas compositionnelle : le seul fait de connaître les réductions possibles de deux termes P et Q ne permet pas de déduire l'ensemble des réductions de $P \mid Q$ car les deux processus, en plus de leurs réductions internes, peuvent aussi interagir.

D'autre part, la définition naturelle de bisimulation associée à la réduction ne donne pas une congruence. En effet, l'environnement dans lequel on place un processus peut contenir des routeurs qui font apparaître des réductions. Ainsi le terme $\bar{x}() \mid y()$ est irréductible, alors que sa composition avec un routeur $x{\to}y$ permet une réduction.

Pour résoudre ces questions, on définit maintenant la sémantique opérationnelle du calcul comme un système de transition étiqueté. La définition se décompose en deux éléments : d'une part, à chaque terme on associe une relation de *connexion*, relation réflexive et transitive mais pas forcément symétrique sur l'ensemble des noms, engendrée par les routeurs $u{\to}v$ actifs dans un terme. Cette relation intervient ensuite dans les règles de dérivation pour rendre compte de l'effet des routeurs.

Définition 4.9. Pour tout terme P, la relation de connexion $\mathcal{F}(P)$ est définie inductivement par les règles suivantes, où $(\cdot)^*$ désigne la clôture réflexive et transitive et où la restriction de relation, pour tout $X \subseteq \mathbf{N}$, est définie par $R \setminus X = (R \cap (\mathbf{N} \setminus X)^2) \cup \mathrm{id}_X$:

$$\mathcal{F}(u{\to}v) = \mathrm{id}_{\mathbf{N}} \cup \{(u,v)\} \qquad \mathcal{F}(P \mid Q) = (\mathcal{F}(P) \cup \mathcal{F}(Q))^*$$
$$\mathcal{F}((\boldsymbol{\nu} x)P) = \mathcal{F}(P) \setminus \{x\} \qquad \mathcal{F}(\alpha.P) = \mathcal{F}(!\alpha.P) = \mathcal{F}(1) = \mathrm{id}_{\mathbf{N}}$$

Définition 4.10. Le système de transition étiqueté du $\vec{\pi}$-calcul est défini par les règles de la table 4.2. Les trois formes de transitions sont les suivantes :

silencieuse :	τ ,	
pondérée :	$[u{\to}v]$,	
visible :	$u^{\varepsilon}(x_1 \dots x_n)$	avec $x_1, \dots, x_n$ frais et tous distincts.

On note $\mathrm{n}(e)$ l'ensemble des noms qui apparaissent dans une étiquette e, ainsi $\mathrm{n}(\tau) = \emptyset$, $\mathrm{n}(u^{\varepsilon}(x_1 \dots x_n)) = \{u, x_1, \dots, x_n\}$ et $\mathrm{n}([u \to v]) = \{u, v\}$. Dans une transition $P \xrightarrow{u^{\varepsilon}(x_1 \dots x_n)} P'$, les noms x_i sont toujours tous distincts et ils n'apparaissent pas parmi les noms libres de P.

Une transition visible $u^{\varepsilon}(\vec{x})$ correspond une action de polarité ε sur le canal u, avec comme donnée la suite de canaux $\vec{x}$. Une transition pondérée $[u{\to}v]$ se comprend comme une synchronisation entre un $u^{\uparrow}(\vec{x})$ et un $v^{\downarrow}(\vec{y})$ à l'intérieur

Actions :

$$\frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \frac{S \xrightarrow{\alpha} P}{!S \xrightarrow{\alpha} P \mid !S} \qquad \frac{S \xrightarrow{\alpha} P}{S + T \xrightarrow{\alpha} P} \qquad \frac{S \xrightarrow{\alpha} P}{T + S \xrightarrow{\alpha} P}$$

Synchronisation :

$$\frac{P \xrightarrow{\bar{u}^{\varepsilon}(\vec{x})} P' \qquad Q \xrightarrow{v^{\varepsilon}(\vec{x})} Q'}{P \mid Q \xrightarrow{[u \to^{\varepsilon} v]} (\boldsymbol{\nu}\vec{x})(P' \mid Q')}$$

Connexions :

$$\frac{P \xrightarrow{[u \to v]} P'}{P \xrightarrow{[u' \to v']} P'} (u, u'), (v', v) \in \mathcal{F}(P) \qquad \frac{P \xrightarrow{[u \to v]} P'}{P \xrightarrow{\tau} P'} (u, v) \in \mathcal{F}(P)$$

$$\frac{P \xrightarrow{\bar{u}(\vec{x})} P'}{P \xrightarrow{\bar{v}(\vec{x})} P'} (u, v) \in \mathcal{F}(P) \qquad \frac{P \xrightarrow{v(\vec{x})} P'}{P \xrightarrow{u(\vec{x})} P'} (u, v) \in \mathcal{F}(P)$$

Règles contextuelles (où e est une étiquette quelconque) :

$$\frac{P \xrightarrow{e} P'}{P \mid Q \xrightarrow{e} P' \mid Q} \qquad \frac{P \xrightarrow{e} P'}{Q \mid P \xrightarrow{e} Q \mid P'} \qquad \frac{P \xrightarrow{e} P'}{(\boldsymbol{\nu}x)P \xrightarrow{e} (\boldsymbol{\nu}x)P'} x \notin \mathrm{n}(e)$$

TAB. 4.2 – Système de transition étiqueté du $\vec{\pi}$-calcul.

du processus, c'est-à-dire comme une τ-transition conditionnée par la présence dans l'environnement d'une suite de routeurs du canal u vers le canal v. Les règles de renommage des transitions dans la table 4.2 formalisent cette idée.

Remarque 4.11. Le système de transition est défini strictement par induction sur la forme des termes, sans quotienter l'ensemble des termes pas autre chose que le renommage des noms liés. Pour cette raison, et contrairement à ce que l'on pourrait penser, chacune des quatre règles de renommage des transitions est nécessaire. Si le système était défini à congruence structurelle près, seule la règle d'introduction de τ-transitions serait nécessaire. Considérons par exemple

$$P := (\boldsymbol{\nu}v)\big((\boldsymbol{\nu}u)(\bar{u}(x) \mid u{\to}u') \mid v(x) \mid v'{\to}v\big)$$

Alors P a une unique transition $[u'{\to}v']$ et que l'on dérive de la façon exposée en figure 4.1 page suivante. On constate aisément qu'il est nécessaire de renommer les actions avant d'appliquer la règle de traversée de $(\boldsymbol{\nu}u)$ et $(\boldsymbol{\nu}v)$ si l'on raisonne strictement par induction structurelle sur les termes.

De même, il est nécessaire de définir $\mathcal{F}(P)$ comme une relation réflexive et transitive sur les noms. Renoncer à la réflexivité reviendrait à renoncer à la règle $1 \equiv x{\to}x$, ce qui donnerait un système différent où un routeur doit toujours être explicitement présent pour autoriser des connexions, à moins de multiplier le nombre de règles de renommage. La transitivité est nécessaire pour interpréter correctement l'opérateur de restriction : si on n'avait $(u, v) \in \mathcal{F}(P)$ que lorsque P contient $u{\to}v$ syntaxiquement, il serait délicat de représenter le fait que $(\boldsymbol{\nu}y)(x{\to}y \mid y{\to}z)$ permet effectivement une connexion de x vers z.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\bar{u}(x) \xrightarrow{\bar{u}(x)} 1}}{\bar{u}(x) \mid u{\to}u' \xrightarrow{\bar{u}(x)} 1 \mid u{\to}u'}}{\bar{u}(x) \mid u{\to}u' \xrightarrow{\bar{u}'(x)} 1 \mid u{\to}u'}}{(\boldsymbol{\nu}u)(\bar{u}(x) \mid u{\to}u') \xrightarrow{\bar{u}'(x)} (\boldsymbol{\nu}u)(1 \mid u{\to}u')} \quad \dfrac{}{v(x) \xrightarrow{v(x)} 1}}{(\boldsymbol{\nu}u)(\bar{u}(x) \mid u{\to}u') \mid v(x) \xrightarrow{[u'\to v]} (\boldsymbol{\nu}u)(1 \mid u{\to}u') \mid 1}}{(\boldsymbol{\nu}u)(\bar{u}(x) \mid u{\to}u') \mid v(x) \mid v'{\to}v \xrightarrow{[u'\to v]} (\boldsymbol{\nu}u)(1 \mid u{\to}u') \mid 1 \mid v'{\to}v}}{(\boldsymbol{\nu}u)(\bar{u}(x) \mid u{\to}u') \mid v(x) \mid v'{\to}v \xrightarrow{[u'\to v']} (\boldsymbol{\nu}u)(1 \mid u{\to}u') \mid 1 \mid v'{\to}v}}{(\boldsymbol{\nu}v)((\boldsymbol{\nu}u)(\bar{u}(x) \mid u{\to}u') \mid v(x) \mid v'{\to}v) \xrightarrow{[u'\to v']} (\boldsymbol{\nu}v)((\boldsymbol{\nu}u)(1 \mid u{\to}u') \mid 1 \mid v'{\to}v)}$$

FIG. 4.1 – Exemple de dérivation d'une transition.

4.2 Bisimulation

La sémantique opérationnelle présentée sous forme de système de transition étiqueté permet d'obtenir les bonnes définitions de simulation et de bisimulation. Ces notions sont standard pour les systèmes de transition étiquetés, il faut toutefois prendre en compte la relation de connexion, qui est la partie visible de l'état d'un processus.

Définition 4.12. Une simulation est une relation binaire $\mathcal{S}$ sur Π telle que $P \mathcal{S} Q$ implique $\mathcal{F}(P) \subseteq \mathcal{F}(Q)$ et pour toute transition $P \xrightarrow{e} P'$ il existe une transition $Q \xrightarrow{e} Q'$ avec $P' \mathcal{S} Q'$. Une bisimulation est une simulation dont l'inverse est une simulation. Deux processus P et Q sont bisimilaires s'ils sont en relation par une bisimulation, on note alors $P \cong Q$.

Cette notion de simulation est l'extension naturelle de la simulation traditionnelle au cas présent, en tenant compte de l'état de connexion. Notons que la bisimilarité de deux processus implique l'égalité de cet état. La bisimilarité (forte) est la plus stricte des équivalences considérées comme sémantiques, puisqu'elle contient toute la dynamique des processus mais aucune propriété vraiment syntaxique.

On peut maintenant démontrer l'équivalence entre les deux sémantiques opérationnelles, c'est-à-dire le fait que les τ-transitions dérivées par les règles de la table 4.2 sont exactement les réductions issues de la définition 4.4.

Lemme 4.13. *La congruence structurelle est une bisimulation.*

Démonstration. Soient deux processus équivalents $P \equiv Q$, supposons l'existence d'une transition étiquetée $P \xrightarrow{e} P'$. On raisonne par induction sur la dérivation de cette équivalence, selon la nature de la dernière règle :

- si la dernière règle est une règle de commutativité ou d'associativité pour la somme on vérifie immédiatement que les transitions sont les mêmes et que les réduits sont égaux ;

– dans le cas de la réplication, on a $P = !\sum_i \alpha_i.R_i$ et $Q = \sum_i \alpha_i.(R_i \mid P)$ (ou l'inverse), donc P et Q ont les mêmes transitions, de plus après une transition α_i on a les deux dérivations suivantes :

$$\dfrac{\dfrac{\dfrac{\alpha_i.R_i \xrightarrow{\alpha_i} R_i}{\sum_i \alpha_i.R_i \xrightarrow{\alpha_i} R_i}}{!\sum_i \alpha_i.R_i \xrightarrow{\alpha_i} R_i \mid !\sum_i \alpha_i.R_i}}{P \xrightarrow{\alpha_i} R_i \mid P} \qquad \dfrac{\dfrac{\alpha_i.(R_i \mid P) \xrightarrow{\alpha_i} R_i \mid P}{\sum_i \alpha_i.(R_i \mid P) \xrightarrow{\alpha_i} R_i \mid P}}{Q \xrightarrow{\alpha_i} R_i \mid P}$$

donc les réduits sont égaux par chaque transition ;
– si la dernière règle est un axiome de monoïde commutatif pour la composition, la dérivation de transition se termine par des règles de passage au contexte, on vérifie sans peine la commutation de ces règles ;
– si la dernière règle est une règle de portée, on vérifie sans problème qu'elle s'applique aussi aux dérivations de transitions ;
– le cas des axiomes de réflexivité et transitivité des connexions découle du fait que les routeurs n'ont pas de transitions et que la relation de connexion est conservée ;
– les règles de passage au contexte sont immédiates ;
– si $P \equiv Q$ est issu par transitivité de $P \equiv R$ et $R \equiv Q$, par hypothèse d'induction on déduit l'existence d'une transition $R \xrightarrow{e} R'$ avec $P' \equiv R'$, puis l'existence d'une transition $Q \xrightarrow{e} Q'$ avec $R' \equiv Q'$, d'où $P' \equiv Q'$.

La relation $\equiv$ est donc une simulation. De plus elle est symétrique par construction, c'est donc une bisimulation. □

Les règles de commutation et d'association de la congruence structurelle permettent de normaliser les termes en séparant les actions et les routeurs :

Lemme 4.14. *Pour tout terme $P \in \Pi$, il existe une composition d'actions P', une composition de routeurs F et une suite de noms $\vec{x}$ tels que $P \equiv (\boldsymbol{\nu}\vec{x})(F \mid P')$.*

Démonstration. On raisonne par induction sur la structure du terme P. Le cas des branchements et des routeurs est trivial. Le cas des restrictions est immédiat. Dans le cas de la composition parallèle, on a par induction $P \equiv (\boldsymbol{\nu}\vec{x})(F_P \mid P')$ et $Q \equiv (\boldsymbol{\nu}\vec{y})(F_Q \mid Q')$ d'où on déduit

$$\begin{aligned} P \mid Q &\equiv (\boldsymbol{\nu}\vec{x})(F_P \mid P') \mid (\boldsymbol{\nu}\vec{y})(F_Q \mid Q') && \text{par congruence,} \\ &\equiv (\boldsymbol{\nu}\vec{x}\vec{y})((F_P \mid P') \mid (F_Q \mid Q')) && \text{par ouverture de portées,} \\ &\equiv (\boldsymbol{\nu}\vec{x}\vec{y})((F_P \mid F_Q) \mid (P' \mid Q')) && \text{par associativité et commutativité.} \end{aligned}$$

Le terme obtenu vérifie la propriété voulue, ce qui conclut la démonstration. □

Une conséquence de cette normalisation de la forme des termes est la caractérisation de la relation de connexion au moyen de la congruence structurelle :

Lemme 4.15. *Pour tout terme P et toute paire de noms (u, v), on a $(u, v) \in \mathcal{F}(P)$ si et seulement si $P \equiv P \mid u{\rightarrow}v$.*

Démonstration. On montre d'abord le résultat sur les compositions de routeurs. Par induction sur le nombre de noms apparaissant dans un tel processus F, on montre l'équivalence $F \equiv \prod_{(u,v)\in\mathcal{F}(F)} u{\rightarrow}v$. Par suite, pour chaque couple

(u, v) on montre facilement $u{\rightarrow}v \equiv u{\rightarrow}v \mid u{\rightarrow}v$, d'où $F \equiv F \mid u{\rightarrow}v$ pour chaque $(u, v) \in \mathcal{F}(F)$. Réciproquement, si $F \equiv F \mid u{\rightarrow}v$, les deux processus on la même connexion, donc $(u, v) \in \mathcal{F}(F)$. Le résultat s'étend à tous les processus au moyen du lemme 4.14 en remarquant que (u, v) est élément de $\mathcal{F}((\boldsymbol{\nu}\vec{x})(F \mid P'))$ si et seulement si soit $u = v$, soit $(u, v) \in \mathcal{F}(F)$ et u et v sont deux noms n'apparaissant pas dans $\vec{x}$. □

Dans les termes de la forme obtenue au lemme 4.14, comme P' est une composition parallèle de branchements séquentiels, on a $\mathcal{F}(P') = \mathrm{id}$, à l'inverse F n'a aucune transition mais fournit toute la connectivité du processus. Par conséquent, toute transition émise par P est émise par P' puis renommée par F.

Proposition 4.16. *Pour tous processus P et Q, il existe une réduction $P \to Q$ si et seulement si il existe une transition $P \xrightarrow{\tau} Q' \equiv Q$.*

Démonstration. Pour le sens direct, il suffit de remarquer que la réduction est une τ-transition dérivable dans le système des transition étiqueté, si le membre gauche s'écrit $(\bar{u}(\vec{x}).P + P' \mid v(\vec{x}).Q + Q') \mid u{\rightarrow}v$, et que la τ-transition et la réduction passent au contexte de la même façon. En conséquence toute réduction est dérivable puisque la congruence structurelle est une bisimulation.

Pour la réciproque, grâce au lemme 4.14 on peut écrire $P \equiv (\boldsymbol{\nu}\vec{y})(F \mid P')$, et comme la congruence structurelle est une bisimulation on a $(\boldsymbol{\nu}\vec{y})(F \mid P') \xrightarrow{\tau}{\equiv} Q$. Comme la transition ne peut provenir que de P' dont la connexion est triviale, en réordonnant les composants de P' on peut écrire $P' \equiv S_1 \mid S_2 \mid P''$ de sorte que la transition soit la synchronisation d'une émission de S_1 et d'une réception de S_2. En appliquant les règles de la somme et le développement des réplications, on peut de plus écrire $S_1 \equiv \bar{u}(\vec{x}).P_1 + T_1$ et $S_2 \equiv v(\vec{x}).P_2 + T_2$. La transition s'écrit alors

$$\dfrac{\dfrac{\dfrac{\bar{u}(\vec{x}).P_1 \xrightarrow{\bar{u}(\vec{x})} P_1}{\bar{u}(\vec{x}).P_1 + T_1 \xrightarrow{\bar{u}(\vec{x})} P_1} \quad \dfrac{v(\vec{x}).P_2 \xrightarrow{v(\vec{x})} P_2}{v(\vec{x}).P_2 + T_2 \xrightarrow{v(\vec{x})} P_2}}{\bar{u}(\vec{x}).P_1 + T_1 \mid v(\vec{x}).P_2 + T_2 \xrightarrow{[u\rightarrow v]} (\boldsymbol{\nu}\vec{x})(P_1 \mid P_2)}}{F \mid \bar{u}(\vec{x}).P_1 + T_1 \mid v(\vec{x}).P_2 + T_2 \mid P'' \xrightarrow{[u\rightarrow v]} F \mid (\boldsymbol{\nu}\vec{x})(P_1 \mid P_2) \mid P''}$$

$$\dfrac{F \mid \bar{u}(\vec{x}).P_1 + T_1 \mid v(\vec{x}).P_2 + T_2 \mid P'' \xrightarrow{\tau} F \mid (\boldsymbol{\nu}\vec{x})(P_1 \mid P_2) \mid P''}{(\boldsymbol{\nu}\vec{y})(F \mid \bar{u}(\vec{x}).P_1 + T_1 \mid v(\vec{x}).P_2 + T_2 \mid P'') \xrightarrow{\tau} (\boldsymbol{\nu}\vec{y})(F \mid (\boldsymbol{\nu}\vec{x})(P_1 \mid P_2) \mid P'')}$$

Le passage de $[u{\rightarrow}v]$ à τ se déduit de $(u, v) \in \mathcal{F}(F)$, qui implique l'équivalence $F \equiv F \mid u{\rightarrow}v$ par les règles de réflexivité et transitivité, donc à associativité et commutativité près cette transition s'écrit

$$\begin{aligned} P \equiv (\boldsymbol{\nu}\vec{y})\big(\bar{u}(\vec{x}).P_1 + T_1 \mid v(\vec{x}).P_2 + T_2 \mid u{\rightarrow}v \mid F \mid P''\big) \\ \xrightarrow{\tau} \quad (\boldsymbol{\nu}\vec{y})\big((\boldsymbol{\nu}\vec{x})(P_1 \mid P_2) \mid u{\rightarrow}v \mid F \mid P''\big) \equiv Q \end{aligned}$$

et cette transition est une réduction au sens de la définition 4.4. □

Cet énoncé montre que les deux sémantiques sont équivalentes, puisque la réduction d'un terme modulo congruence structurelle correspond à la τ-transition, qui est son analogue étiquetée. On peut en déduire une caractérisation des transitions visibles, au moyen de contextes particuliers :

Proposition 4.17. *Soit P un processus, il existe une transition $P \xrightarrow{u^\varepsilon(\vec{x})} \equiv P'$ si et seulement si pour tout processus Q on a $P \mid \bar{u}^\varepsilon(\vec{x}).Q \to (\boldsymbol{\nu}\vec{x})(P' \mid Q)$.*

Démonstration. Le sens direct est évident. Pour la réciproque, soit v un nom qui n'apparaît pas dans P, et soit le test $T = \bar{u}^\varepsilon(\vec{x}).v$. On a alors $P|T \to (\boldsymbol{\nu}\vec{x})(P'|v)$, donc par la proposition 4.16 on sait qu'il existe une transition $P \mid T \xrightarrow{\tau} R \equiv (\boldsymbol{\nu}\vec{x})(P' \mid v)$. R a donc une transition étiquetée v, qui ne peut provenir que de $\bar{u}^\varepsilon(\vec{x}).v$ puisque ce terme est le seul à contenir v (et donc que v n'est connecté à aucun nom). Par conséquent cette τ-transition est la composition de $T \xrightarrow{\bar{u}^\varepsilon(\vec{x})} v$ avec un $P \xrightarrow{u^\varepsilon(\vec{x})} \equiv P'$. □

En revanche, il ne semble pas exister de caractérisation similaire pour les transitions pondérées. En effet, de $P \xrightarrow{[u\to v]} P'$ on déduit évidemment $P|u{\to}v \to P'$, mais à partir de cette réduction on ne peut pas déduire une transition précise de P. Tout ce qu'il est possible de déduire est que P a une transition vers un processus équivalent à P' étiquetée soit τ soit $[u'{\to}v']$ avec $(u', v') \in \mathcal{F}(P|u{\to}v)$.

Proposition 4.18. *La bisimilarité est une congruence.*

Démonstration. On procède en montrant que la bisimulation est préservée par chacune des constructions du langage, ce qui permet de conclure par transitivité de la bisimilarité.

- Pour la composition, soit $\mathcal{S} = \{\, (P \mid R, Q \mid R) \mid P \cong Q \,\}$. Soit $(P|R, Q|R)$ un élément de $\mathcal{S}$, on a évidemment $\mathcal{F}(P \mid R) = \mathcal{F}(Q \mid R)$, d'autre part toute transition $P \mid R \xrightarrow{e} S$ est issue de P ou R, dans le premier cas on conclut en simulant la transition dans Q, dans l'autre cas on applique la même transition à R dans $Q \mid R$. De même toute transition de $Q \mid R$ est simulée dans $\mathcal{S}$ par $P \mid R$, donc $\mathcal{S}$ est une bisimulation.
- Pour la restriction, on pose $\mathcal{S} = \{\, ((\boldsymbol{\nu}x)P, (\boldsymbol{\nu}x)Q) \mid P \cong Q \,\}$ et on déduit sans peine que $\mathcal{S}$ est une bisimulation.
- Pour les actions bloquantes, il est évident que $P \cong Q$ entraîne $\alpha.P \cong \alpha.Q$ puisque les deux processus ont une connexion vide et une unique transition qui mène à P et Q.
- Pour le choix externe, on remarque que $S \cong T$, avec S et T séquentiels, entraîne que toute transition de $S + U$ issue de S est simulée par une transition de T dans $T + U$ menant dans chaque cas à des réduits de S et T qui sont donc bisimilaires, et toute transition issue de U mène à des processus identiques.
- Pour la réplication, on pose $\mathcal{S} = \{\, (!S \mid P, !T \mid Q) \mid S \cong T, P \cong Q \,\}$. Soit $(!S \mid P, !T \mid Q) \in \mathcal{S}$, il est clair que les deux processus ont même connexion. Toute transition $!S \mid P \xrightarrow{e} R$ est issue de P ou de $!S$; le premier cas est immédiat, dans le deuxième cas on a une dérivation

$$\dfrac{\dfrac{S \xrightarrow{e} S'}{!S \xrightarrow{e} !S \mid S'}}{!S \mid P \xrightarrow{e} !S \mid S' \mid P}$$

Par hypothèse T simule S donc il existe un T' tel que $S' \cong T'$ et

$$\frac{\dfrac{T \xrightarrow{e} T'}{!T \xrightarrow{e} !T \mid T'}}{!T \mid Q \xrightarrow{e} !T \mid T' \mid Q}$$

or $S' \cong T'$ et $P \cong Q$ impliquent $S' \mid P \cong T' \mid Q$ d'après le premier cas de cette congruence, donc on a bien $(!S \mid S' \mid P)\ \mathcal{S}\ (!T \mid T' \mid Q)$. La simulation réciproque est identique, donc $\mathcal{S}$ est une bisimulation. Enfin, si $S \cong T$ on a $!S \cong (!S \mid 1)\ \mathcal{S}\ (!T \mid 1) \cong !T$ donc $!S \cong !T$ par transitivité.

Toutes les constructions préservent donc la bisimilarité. □

4.3 Typage des noms

Comme le calcul est polyadique, il y a une restriction implicite sur la synchronisation selon laquelle deux actions $u(\vec{x})$ et $\bar{v}(\vec{y})$ ne peuvent se synchroniser que si les vecteurs $\vec{x}$ et $\vec{y}$ sont de même arité. Il parait donc naturel de vouloir imposer un typage sur les noms pour éviter qu'il n'apparaisse des actions d'arités différentes sur un même canal. D'autre part, dans l'étude logique du calcul qu'on mènera plus loin, il sera important que chaque nom libre d'un processus ait une polarité fixée. On combine donc ces deux propriétés dans un système typage des noms.

4.3.1 Types et interfaces

Définition 4.19. On se donne une famille dénombrable de variables de type de canaux. L'algèbre des types de canaux (notés $s, t, \ldots$) et celle des interfaces (notées $\mathcal{I}, \mathcal{J}, \mathcal{K}, \ldots$) sont définies comme suit, où ε désigne une polarité et α une variable de type :

types de canaux :	$s := \varepsilon(s_1, \ldots, s_n) \mid \alpha \mid \bar{\alpha}$	
interfaces :	$\mathcal{I} := x_1 : s_1, \ldots, x_n : s_n$	avec les x_i distincts

On appelle *sorte* une suite de types de canaux. L'involution canonique sur les polarités s'étend aux types de canaux et aux interfaces en inversant toutes les polarités des types récursivement.

Le support d'une interface est l'ensemble des noms de canaux qui y apparaissent. Deux interfaces sont disjointes si leurs supports sont disjoints. Si $\mathcal{I}$ et $\mathcal{J}$ sont deux interfaces disjointes, on note $\mathcal{I} \to \mathcal{J} = \neg\mathcal{I} \cup \mathcal{J}$.

Le langage des types de canaux donné ici est minimal mais il suffit pour le propos ; on peut imaginer sans problème de l'étendre par des quantificateurs, points fixes et autres constructions, notamment pour pouvoir parler de polymorphisme.

Définition 4.20. Un processus P respecte l'interface $\mathcal{I}$ si le jugement $P :: \mathcal{I}$ est dérivable par les règles de la table 4.3. On tolère également la règle $x{\to}x :: \mathcal{I}$ si x apparaît dans $\mathcal{I}$.

$$\frac{}{u{\to}v :: \mathcal{I}, u : {\downarrow}(s_1, \ldots, s_n), v : {\uparrow}(\bar{s}_1, \ldots, \bar{s}_n)}$$

$$\frac{P :: \mathcal{I}, u : \varepsilon(s_1 \ldots s_n), x_1 : s_1, \ldots, x_n : s_n}{u^{\varepsilon}(\vec{x}).P :: \mathcal{I}, u : \varepsilon(s_1 \ldots s_n)}$$

$$\frac{}{1 :: \mathcal{I}} \qquad \frac{S :: \mathcal{I}}{!S :: \mathcal{I}} \qquad \frac{S :: \mathcal{I} \qquad T :: \mathcal{I}}{S + T :: \mathcal{I}} \qquad \frac{P :: \mathcal{I} \qquad Q :: \mathcal{I}}{P \mid Q :: \mathcal{I}}$$

$$\frac{P :: \mathcal{I}, x_1 : s, x_2 : \bar{s}}{(\boldsymbol{\nu} x)P[x/x_1, x/x_2] :: \mathcal{I}}$$

TAB. 4.3 – Règles de typage des noms.

La notation utilisée pour les jugements de bonne formation est inhabituelle : on note $P :: \mathcal{I}$ là où d'autres systèmes exprimeraient la bonne formation d'un processus en écrivant $\mathcal{I} \vdash P$. On évite ici l'utilisation du symbole de séquent pour éviter la confusion avec le système de typage des comportements que l'on construit dans le chapitre 5.

Les règles de typage pour les noms sont assez restrictives. Toutes sont naturelles, sauf peut-être celle de la restriction : comme le système ne doit pas permettre à un processus typé d'avoir des actions de polarités opposées sur un nom public, seule cette règle doit permettre l'apparition de cette situation. Ainsi, pour typer $(\boldsymbol{\nu} x)P$, on utilise les règles pour typer P dans lequel toutes les occurrences de x sont remplacées par x_1 ou x_2 selon leur polarité, et c'est la règle de typage qui remplace x_1 et x_2 par x en imposant que leurs types soient bien opposés.

Dans une interface donnée, chaque nom a une polarité fixée, donc toute interface $\mathcal{I}$ peut s'écrire de façon unique sous la forme $\mathcal{I}_1 \to \mathcal{I}_2$ de sorte que chaque nom dans $\mathcal{I}_1$ et $\mathcal{I}_2$ soit de polarité négative. Une conséquence notable est que pour un processus bien interfacé $P :: \mathcal{I}$ la relation $\mathcal{F}(P)$ ne contient que des couples (u, v) avec $u = v$ ou $u \in \mathcal{I}_1$ et $v \in \mathcal{I}_2$. En particulier la règle de transitivité du routage $u{\to}v \mid v{\to}w \equiv u{\to}v \mid v{\to}w \mid u{\to}w$ ne peut donc s'appliquer que sous un $(\boldsymbol{\nu} v)$ si les trois noms sont distincts.

Proposition 4.21. *Pour tous termes P et Q tels que $P \equiv Q$, $P :: \mathcal{I}$ est dérivable si et seulement si $Q :: \mathcal{I}$ est dérivable.*

Démonstration. Les règles de monoïde commutatif pour la composition sont compatibles avec le typage des noms puisqu'elles s'appliquent à des termes de même interface. Les règles de composition des connexions ne s'appliquent pas, selon la remarque précédente (sauf pour la réflexivité, qui est acceptée par la définition 4.20). Les règles de portées s'adaptent sans problème, de même que celles de passage au contexte. □

La signification du système des interfaces s'exprime par la propriété suivante, qui décrit la forme des transitions possibles d'un processus en fonction de son interface et garantit que la bonne formation est préservée par toute transition étiquetée :

Proposition 4.22. *Soit P un processus d'interface $\mathcal{I}$ et soit $P \xrightarrow{e} Q$ une transition de P, alors*
- *si $e = \tau$ alors Q est d'interface $\mathcal{I}$;*
- *si $e = [u{\to}v]$ alors Q est d'interface $\mathcal{I}$ et il existe une sorte J telle que $\mathcal{I}$ contienne $u : \uparrow J$ et $v : \downarrow \bar{J}$;*
- *si $e = u^{\varepsilon}(x_1 \dots x_n)$ alors $\mathcal{I}$ contient un $u : \varepsilon(s_1, \dots, s_n)$ et Q est d'interface $\mathcal{I}, x_1 : s_1, \dots, x_n : s_n$.*

Démonstration. Par induction sur la dérivation des transitions. □

Ainsi, pour un processus P bien interfacé, il ne peut peut pas exister de transition étiquetée $[u{\to}v]$ pour laquelle $(u, v) \in \mathcal{F}(P)$, donc toute τ-transition est issue d'une communication sur des canaux privés. En conséquence, pour une paire de processus bien interfacés $P :: \mathcal{I}$ et $Q :: \mathcal{I}$ la composition parallèle $P \mid Q$ n'a aucune réduction qui soit issue de l'interaction entre P et Q. Pour établir des interactions entre P et Q, il est donc nécessaire d'unifier deux noms de polarités opposées en les rendant privés, par la règle d'introduction du $(\boldsymbol{\nu} x)$.

La notation $\mathcal{I} \to \mathcal{J}$, introduite dans la définition 4.19, est un artifice de notation qui permet de diviser arbitrairement l'interface d'un processus typé en une entrée et une sortie, sans lien avec la polarité associée au noms. L'intérêt de cette technique, employée en particulier par Abramsky dans ses catégories d'interaction [2], est qu'elle permet de définir une composition de processus avec interaction possible, associative, et qui respecte la contrainte de polarisation :

Définition 4.23. Soient $\mathcal{I}, \mathcal{J}, \mathcal{K}$ trois interfaces disjointes, et soient $P :: \mathcal{I} \to \mathcal{J}$ et $Q :: \mathcal{J} \to \mathcal{K}$ deux processus à canaux typés. La composition typée de P et Q sur $\mathcal{J}$, notée $P \cdot_{\mathcal{J}} Q$ (ou $P \cdot Q$ s'il n'y a pas d'ambiguïté), est le processus d'interface $\mathcal{I} \to \mathcal{K}$ défini comme $P \cdot_{\mathcal{J}} Q = (\boldsymbol{\nu} \vec{x})(P \mid Q)$, où $\vec{x}$ est le support de $\mathcal{J}$.

Dans la définition de la composition par une interface $\mathcal{J}$, on introduit une restriction sur l'ensemble des noms de l'interface $\mathcal{J}$. Il sera souvent pratique de parler du support d'une interface dans ce genre de contexte, et puisqu'il n'y a pas d'ambiguïté on confondra souvent une interface et son support. Ainsi $x \in \mathcal{J}$ signifie que x apparaît dans le support de $\mathcal{J}$, et la composition de P et Q par $\mathcal{J}$ se notera aussi $(\boldsymbol{\nu}\mathcal{J})(P \mid Q)$.

Proposition 4.24. *Soient $\mathcal{I}, \mathcal{J}, \mathcal{K}, \mathcal{L}$ quatre interfaces deux à deux disjointes, et soient $P :: \mathcal{I} \to \mathcal{J}$, $Q :: \mathcal{J} \to \mathcal{K}$ et $R :: \mathcal{K} \to \mathcal{L}$ trois processus à canaux typés. Alors $(P \cdot_{\mathcal{J}} Q) \cdot_{\mathcal{K}} R \equiv P \cdot_{\mathcal{J}} (Q \cdot_{\mathcal{K}} R)$.*

Démonstration. Les compositions $\cdot_{\mathcal{J}}$ et $\cdot_{\mathcal{K}}$ introduisent des restrictions sur des familles de noms disjointes, et par hypothèse les noms de $\mathcal{J}$ n'apparaissent pas dans R ni ceux de $\mathcal{K}$ dans P, donc on a

$$\begin{aligned}(P \cdot_{\mathcal{J}} Q) \cdot_{\mathcal{K}} R &= (\boldsymbol{\nu}\mathcal{K})((\boldsymbol{\nu}\mathcal{J})(P \mid Q) \mid R) \equiv (\boldsymbol{\nu}\mathcal{J}\mathcal{K})((P \mid Q) \mid R) \\ &\equiv (\boldsymbol{\nu}\mathcal{J}\mathcal{K})(P \mid (Q \mid R)) \equiv (\boldsymbol{\nu}\mathcal{J})(P \mid (\boldsymbol{\nu}\mathcal{K})(Q \mid R)) = P \cdot_{\mathcal{J}} (Q \cdot_{\mathcal{K}} R)\end{aligned}$$

selon la définition de la congruence structurelle. □

4.3.2 Connexion et substitution

La connexion entre canaux induite par les routeurs peut être interprétée comme une forme polarisée de substitution explicite sur les noms. En effet, la

présence d'un routeur $u{\to}v$ entraîne que toute émission sur u peut être réécrite en une émission sur v. Dans le cas où toutes les autres occurrences de u correspondent à une émission, ce routeur agit effectivement comme une substitution.

Proposition 4.25. *Soit P un processus et u un nom dont toutes les occurrences dans P sont de polarité ε. Pour tout nom v on a $(\boldsymbol{\nu} u)(P \mid u{\to}^{\varepsilon} v) \cong P[v/u]$.*

Démonstration. On détaille la démonstration dans le cas $\varepsilon = \uparrow$, l'autre cas étant analogue. Soit $\mathcal{S}$ la relation constituée de l'ensemble des couples $((\boldsymbol{\nu} u)(P \mid u{\to}v), P[v/u])$ où toutes les occurrences de u dans P sont négatives. Cette propriété est évidemment préservée par toute transition.

Soit un tel P, alors $\mathcal{F}(P)$ ne contient pas de (u, x) avec $u \neq x$, donc $\mathcal{F}(P \mid u{\to}v)$ est égal à $\mathcal{F}(P) \cup \{(u, v)\} \cup \{\,(x, v) \mid (x, u) \in \mathcal{F}(P)\,\}$, et donc $\mathcal{F}((\boldsymbol{\nu} u)(P \mid u{\to}v)) = \{\,(x, y) \mid (x, y) \in \mathcal{F}(P), x \neq u, y \neq u\,\} \cup \{\,(x, v) \mid (x, u) \in \mathcal{F}(P)\,\}$, c'est-à-dire $\mathcal{F}((\boldsymbol{\nu} u)(P \mid u{\to}v)) = \mathcal{F}(P[v/u])$.

Toute transition $(\boldsymbol{\nu} u)(P \mid u{\to}v) \xrightarrow{a} (\boldsymbol{\nu} u)(Q \mid u{\to}v)$ est issue d'une transition $P \xrightarrow{a'} Q$. Par induction sur le système de transition, on montre qu'il existe une transition $P[v/u] \xrightarrow{a'[v/u]} Q[v/u]$. Si u apparaît dans a', alors on a par hypothèse $a' = [w{\to}u]$ ou $a' = \bar{u}(\vec{x})$, et donc nécessairement $a = a'[v/u]$; si u n'apparaît pas dans a' on a évidemment $a = a'[v/u]$. On a donc $P[v/u] \xrightarrow{a} Q[v/u]$.

Réciproquement, soit une transition $P[v/u] \xrightarrow{a} Q$. Par induction sur la dérivation de cette transition, on montre sans problème qu'il existe une transition $P \xrightarrow{a'} Q'$ avec $a = a'[v/u]$ et $Q = Q'[v/u]$ en utilisant le fait que toute transition sur u dans P est une émission et qu'aucun routage ne peut s'y appliquer. Si a' contient une occurrence de u, celle-ci est forcément négative, donc on a une transition $(\boldsymbol{\nu} u)(P \mid u{\to}v) \xrightarrow{a'[v/u]} (\boldsymbol{\nu} u)(Q' \mid u{\to}v)$. □

Soulignons que l'énoncé de cette proposition impose, dans $(\boldsymbol{\nu} u)(P \mid u{\to}^{\varepsilon} v)$, que le nom u n'apparaisse dans P qu'avec la polarité ε. Si cette condition n'est pas remplie la substitution est invalide, par exemple pour $P = \bar{u} \mid u$ le terme $(\boldsymbol{\nu} u)(P \mid u{\to}v)$ a une τ-transition, mais le terme substitué est $\bar{v} \mid u$ qui n'a pas de τ-transition.

Définition 4.26. Soit $I = \varepsilon_1 I_1 \dots \varepsilon_n I_n$ une sorte, soient $\vec{u}$ et $\vec{v}$ deux suites de noms tous distincts de longueur n. Le routeur de sorte I de $\vec{u}$ à $\vec{v}$ est le processus $\vec{u}{\to}^{I}\vec{v} = u_1{\to}^{\varepsilon_1} v_1 \mid \dots \mid u_n{\to}^{\varepsilon_n} v_n$.

Cette définition est une extension de la notation déjà introduite pour les routeurs élémentaires, en effet on a $u{\to}^{\varepsilon} v = u{\to}^{\varepsilon(I)} v$ pour toute sorte I. De plus, il est évident que pour toute sorte I et toutes suites de noms $\vec{u}$ et $\vec{v}$ on a $\vec{u}{\to}^{I}\vec{v} :: \vec{u} : \neg I, \vec{v} : I$. La propriété de substitution s'étend à ces routeurs de façon naturelle :

Proposition 4.27. *Soient $\mathcal{I}$ une interface, J une sorte, et $\vec{u}$ et $\vec{v}$ deux suites de noms tous distincts de longueurs appropriées. Pour tout processus $P :: \mathcal{I} \to \vec{u} : J$ on a $(\boldsymbol{\nu}\vec{u})(P \mid \vec{u}{\to}^{J}\vec{v}) \cong P[v_1/u_1, \dots, v_n/u_n]$.*

Démonstration. Par récurrence immédiate sur la longueur de J, à partir de la proposition 4.25. □

Notation 4.28. On appelle actions non liantes et actions asynchrones les deux schémas d'actions suivants :

$$\begin{array}{ll} \text{action non liante :} & u^\varepsilon\langle\vec{x} : I\rangle.P = u^\varepsilon(\vec{y}).(\vec{y}\rightarrow^I\vec{x} \mid P) \\ \text{action asynchrone :} & u^\varepsilon(\vec{x})P = (\boldsymbol{\nu}\vec{x})(u^\varepsilon\langle\vec{x}\rangle \mid P) \end{array}$$

On s'autorisera à garder l'interface implicite dans une action $u\langle\vec{x}\rangle$ si cette interface est claire en fonction du contexte. On a évidemment les règles d'interfaçage suivantes :

$$\frac{P :: u : \varepsilon I, \vec{x} : I, \mathcal{J}}{u^\varepsilon\langle\vec{x} : I\rangle.P :: u : \varepsilon I, \vec{x} : I, \mathcal{J}} \qquad \frac{P :: u : \varepsilon I, \vec{x} : I, \mathcal{J}}{u^\varepsilon(\vec{x})P :: u : \varepsilon I, \mathcal{J}}$$

4.3.3 Localisation et abstraction

Les définitions précédentes font référence à des interfaces où des noms de canaux apparaissent explicitement. Cependant, l'intuition est que ces noms ne sont pas importants en tant que tels mais qu'ils servent uniquement à la composition. Les énoncés qui supposent des interfaces disjointes renforcent d'ailleurs cette impression que les noms ajoutent trop d'information.

Pour contourner ce problème, on définit maintenant une version anonyme des interfaces et processus, essentiellement en les considérant à renommage injectif près. Pour l'exposé, on suppose donnée une fois pour toutes une énumération $(n_k)_{k\in\mathbb{N}}$ des éléments de l'ensemble $\mathbf{N}$ des noms.

Définition 4.29. Une interface anonyme, ou sorte, est une suite finie de types de canaux. Si $I = s_1 \dots s_k$ est une sorte de longueur k et $\vec{x}$ une suite de k noms distincts, on définit l'interface $\vec{x} : I$ comme $x_1 : s_1, \dots, x_k : x_k$. Un processus anonyme d'arité k est un processus dont les noms libres sont parmi $n_1, \dots, n_k$. Un processus anonyme de sorte I est un processus d'interface $n_1 : s_1, \dots, n_k : s_k$.

Pour un processus anonyme p d'arité k et une suite de noms $\vec{x}$ de longueur k, notons $p[\vec{x}]$ le processus $p[x_1/n_1, \dots, x_k/n_k]$. Réciproquement, pour un processus P et une suite $\vec{x}$ de noms distincts de longueur suffisante, notons $(\vec{x})P$ l'unique processus anonyme p tel que $p[\vec{x}] = P$. Ces deux opérations sont appelées respectivement *localisation* et *abstraction*.

Définition 4.30. Pour deux sortes I et J, on note I, J la concaténation des deux sortes et on note $I \to J$ l'interface $(\neg I), J$. Soient trois sortes I, J, K et soient deux processus anonymes $p : I \to J$ et $q : J \to K$, la composition de p et q est le processus anonyme

$$q \circ p = p; q = (\vec{x}\vec{z})\,(\boldsymbol{\nu}\vec{y})(p[\vec{x}\vec{y}] \mid q[\vec{y}\vec{z}])$$

où $\vec{x}$, $\vec{y}$, $\vec{z}$ sont trois vecteurs de noms tous distincts de mêmes arités que I, J, K. Ce processus anonyme est de sorte $I \to K$.

Pour toute sorte I, on appelle identité sur I le processus

$$\mathrm{id}_I = (\vec{x}\vec{y})\,(\vec{x}\rightarrow^I\vec{y})$$

La proposition 4.24 montre alors que cette composition est associative modulo congruence structurelle. La proposition 4.27 montre que l'identité id_I, à

bisimilarité près, est neutre pour cette composition. On a donc une catégorie de processus dont les objets sont les sortes et dont l'ensemble des morphismes de I vers J est l'ensemble des processus anonymes de sorte $I \to J$ quotienté par bisimilarité. Comme la division d'une sorte sous la forme $I \to J$ est arbitraire, il est possible de transformer un entrée en sortie et réciproquement par simple renommage, ce qui donne une structure de catégorie compacte close.

4.4 Comparaison avec d'autres calculs

Le calcul défini dans ce chapitre présente naturellement des similitudes avec d'autres variantes du π-calcul, notamment les calculs de fusion. Dans cette section, volontairement informelle, on s'intéresse à différents calculs connus qui peuvent être considérés comme des fragments du π-calcul polarisé, afin d'illustrer les différences d'expressivité.

4.4.1 Substitution, fusion, connexion

Dans le π-calcul polarisé, sous sa forme non typée, deux calculs importants peuvent être vus comme des fragments bien définis : le π-calcul à mobilité interne et celui avec fusions explicites.

Mobilité interne Le calcul πI de Sangiorgi est simplement le fragment sans routeurs du π-calcul polarisé, et sa relation de transition étiquetée est la même sans les transitions pondérées. Ce calcul est connu pour avoir une bonne expressivité, tout en se restreignant à de la mobilité *interne*, différence notable avec le π-calcul et ses variantes qui peuvent échanger des noms publics.

Il est possible de coder le π-calcul asynchrone dans πI, au moyen d'un codage proposé par Boreale [13] qui simule la communication de noms publics au moyen de processus liens. Ces processus, introduits par Sangiorgi [73] pour coder le λ-calcul dans πI, fonctionnent en reproduisant sur un canal de sortie les actions reçues sur un canal d'entrée :

$$a{\to}b := !a(x_1 \dots x_n).\bar{b}(y_1 \dots y_n).\big(y_1{\to}x_1 \mid \dots \mid y_n{\to}x_n\big)$$

L'orientation des $x_i{\to}y_i$ dépend bien sûr de l'interface à laquelle est appliqué le lien. Cette façon de représenter une connexion au moyen d'un processus répéteur est tout à fait similaire aux stratégies *copycat* des sémantiques de jeux [46]. Dans le cadre asynchrone, ces liens ont le comportement des routeurs du π-calcul polarisé, à une forme de bisimulation faible près.

Ce qui fait que les processus liens ont un effet de renommage est qu'en calcul asynchrone, par définition, le seules émissions sont de la forme $\bar{a}\langle x\rangle$ sans continuation. Ainsi le fait que $a{\to}b = !a(x).\bar{b}(y).y{\to}x$ consomme cette émission n'est pas observable par le processus émetteur. Pour cette raison, il est impossible en général de coder des routeurs de façon similaire dans un calcul synchrone.

Indépendamment de cette propriété d'observabilité, l'emploi de processus répéteurs pour représenter le partage de noms pose aussi des problèmes d'interférence. Considérons le processus suivant :

$$P := \bar{u}\langle 1\rangle \mid \bar{u}\langle 2\rangle \mid \bar{a}\langle u\rangle \mid \bar{a}\langle u\rangle \mid !a(x).x(y).Q$$

C'est-à-dire que P contient un serveur qui, pour chaque requête $\bar{a}\langle x\rangle$, attend une donnée sur x puis lance le programme Q avec cette donnée. Un codage à base de répéteurs s'écrit donc

$$P' := \bar{u}\langle 1\rangle \mid \bar{u}\langle 2\rangle \mid \bar{a}(v).u{\rightarrow}v \mid \bar{a}(w).u{\rightarrow}w \mid\, !a(x).x(y).Q$$

Après deux communications sur a, en effaçant le serveur $!a(x).x(y).Q$ qui devient inactif, on a donc la situation suivante :

$$P' \rightarrow P'' = (\boldsymbol{\nu} vw)\big(\bar{u}\langle 1\rangle \mid \bar{u}\langle 2\rangle \mid u{\rightarrow}v \mid u{\rightarrow}w \mid v(y).Q \mid w(y).Q\big)$$

Si les répéteurs $u{\rightarrow}v$ et $u{\rightarrow}w$ sont des processus actifs, qui consomment effectivement les messages à faire suivre, il est possible que le même répéteur fasse suivre les deux messages, ce qui donne

$$P'' \rightarrow (\boldsymbol{\nu} vw)\big(u{\rightarrow}v \mid u{\rightarrow}w \mid \bar{v}\langle 1\rangle \mid \bar{v}\langle 2\rangle \mid v(y).Q \mid w(y).Q\big)$$

Il est clair que, dans la situation obtenue, la réception $w(y).Q$ ne pourra pas être débloquée, et l'un des deux messages ne sera pas consommé. Le codage du π-calcul asynchrone dans πI évite le problème en n'appliquant une telle transformation que dans des cas où ce conflit ne peut pas avoir lieu. À l'inverse, Gardner, Laneve et Wischik emploient des répéteurs linéaires [33] systématiquement associés à des réceptions, pour garantir qu'un répéteur consomme uniquement le message voulu, ce qui évite toute interférence.

Fusions L'exemple précédent illustre que l'emploi de connexions orientées implémentées par des processus normaux nécessite des précautions en raison des interférences possibles entre connexions. Lorsque les communications sont symétriques, le problème ne se pose plus puisque toute redirection d'un message par un répéteur devient réversible. C'est le principe des *equators* de Honda et Yoshida [43] : si l'on pose

$$u{\leftrightarrow}v := !u(\vec{x}).\bar{v}\langle\vec{x}\rangle \mid !v(\vec{x}).\bar{u}\langle\vec{x}\rangle$$

alors pour tout processus P asynchrone, $P \mid u{\leftrightarrow}v$ est équivalent à P dans lequel les noms u et v sont unifiés. Bien entendu, ce résultat n'est applicable qu'en calcul asynchrone, et l'équivalence considérée est une bisimulation faible qui doit ignorer les suites infinies de réductions.

Il n'existe pas de processus ayant cet effet de fusion dans les calculs synchrones où la communication se fait par substitution de noms. C'est la différence principale entre le π-calcul et les calculs de fusion [31, 69], qui disposent de réceptions non liantes. Dans le *fusion calculus*, on peut écrire simplement un processus qui égalise deux noms en posant par exemple $x{\leftrightarrow}y = (\boldsymbol{\nu} u)(\bar{u}\langle x\rangle \mid u\langle y\rangle)$.

L'inconvénient principal des calculs par fusion est que leur sémantique ne peut pas être formulée par une règle de réduction locale, puisque l'effet d'une unification de noms s'étend à toute la portée des noms considérés. C'est pour résoudre ce problème que Gardner et Wischik ont introduit la notion de fusion explicite [34]. Il s'agit de l'analogue des substitutions explicites dans le cadre des calculs à passage de noms : une fusion explicite est un terme $u{\leftrightarrow}v$ qui a pour effet de rendre les noms u et v équivalents. C'est la congruence structurelle

du π-calcul avec fusions explicites, appelé pi-F, qui réalise cet effet, au moyen d'un système de règles dont les principales sont

$$u\langle\vec{x}\rangle.P \mid u{\leftrightarrow}v \equiv v\langle\vec{x}\rangle.P \mid u{\leftrightarrow}v \qquad \alpha.P \mid u{\leftrightarrow}v \equiv \alpha.(P \mid u{\leftrightarrow}v) \mid u{\leftrightarrow}v$$

Une caractéristique importante de cette congruence structurelle est qu'elle engendre le comportement de substitution de noms du π-calcul au moyen d'étapes de réécriture purement locales.

La formulation du système de transition du π-calcul polarisé, et en particulier l'idée de faire apparaître la relation de connexion pour en déduire les transitions autorisées, est en partie inspirée de la démarche de Gardner et Wischik. Le calcul pi-F peut être considéré comme le fragment du π-calcul polarisé dans lequel tous les processus P ont une relation de connexion $\mathcal{F}(P)$ symétrique, et la fusion explicite est naturellement codée par $u{\leftrightarrow}v = (u{\rightarrow}v \mid v{\rightarrow}u)$. Dans ce cas $\mathcal{F}(P)$ est une relation d'équivalence sur les noms de P qui indique quels noms sont fusionnés. La symétrie de la relation de connexion implique que les transitions étiquetées $[u{\rightarrow}v]$ et $[v{\rightarrow}u]$ sont équivalentes, et le système de transition obtenu en identifiant ces transitions est équivalent au système de transition dit structuré du calcul pi-F. Par conséquent, la bisimilarité dans pi-F coïncide avec celle du π-calcul polarisé.

La seule différence entre pi-F et le fragment considéré du π-calcul polarisé est la congruence structurelle, plus large dans pi-F. En effet, il ne semble pas possible de formuler simplement le comportement des routeurs orientés du $\vec{\pi}$-calcul au moyen d'une congruence structurelle.

4.4.2 Autres systèmes de types

Le système d'interfaces utilisé ici peut se voir comme une restriction du système des types i/o du π-calcul [74, chapitre 7], avec la différence importante qu'ici on n'utilise que des capacités de lecture seule (i, notée ici $\downarrow$) ou d'écriture seule (o, notée ici $\uparrow$), et jamais de capacité mixte. Cependant, les conventions d'interprétation sont légèrement différentes. Considérons un processus de la forme $P = u(x).x(y)$, dans le système d'interfaces de la table 4.3 on dérive le type $P :: u : {\downarrow}{\downarrow}t$ pour tout type t, en effet P peut lire sur u un nom x sur lequel il lira un nom y de type t ; dans le système des types i/o on a le jugement $u : \mathrm{ii}T \vdash P$ qui exprime la même chose. Considérons maintenant le processus $Q = \bar{u}(x).\bar{x}(y)$: dans notre système, on type les noms par $Q :: u : {\uparrow}{\uparrow}t$ car Q émet sur u un nom sur lequel il va émettre un nom y de type t ; dans les types i/o le typage le plus strict s'écrira $u : \mathrm{oi}T \vdash Q$ car Q émet un nom x sur lequel le receveur est censé faire une réception, puisque Q y émet un nom.

Des travaux de Honda, Yoshida et Berger [11, 44, 76] visent à caractériser dans le π-calcul diverses classes de processus qui correspondent à différents traits des langages de programmation. Ils se fondent sur des systèmes de types qui peuvent être considéré comme un raffinement des types i/o ou de notre système d'interface. Leurs systèmes ajoutent deux aspects importants : d'une part des informations de linéarité qui contraignent le nombre d'actions présentes sur un canal donné, et d'autre part des caractérisations plus combinatoires des dépendances entre canaux. Les types spécifient ainsi assez précisément le comportement des processus.

Pour spécifier par typage le comportement des processus, nous employons une approche différente. Le système des interfaces est considéré comme un prototypage des processus dont le but est essentiellement de garantir localement la bonne forme des communications. Par dessus ce système, on développe un second niveau de typage pour décrire le comportement des processus d'une interface donnée.

Chapitre 5

Réalisabilité concurrente

Ce chapitre décrit les fondements d'une étude logique (ou devrait-on dire logicienne) des comportements de processus concurrents. La méthode employée consiste à faire le lien entre deux écoles d'esprit assez proches : la réalisabilité du point de vue logique, et les sémantiques de test du point de vue de la programmation.

La réalisabilité existe sous de nombreuses formes en théorie des modèles et en sémantique des langages de programmation, avec quelques principes caractéristiques. L'idée est de considérer une algèbre combinatoire dont les éléments servent de témoins pour la validité des formules logiques ; chaque formule est donc interprétée comme un ensemble d'éléments, et les combinateurs de l'algèbre considérée servent à implémenter les connecteurs logiques. Cette approche, qui remonte aux travaux de Kleene, a été largement étudiée pour la construction de modèles en logique intuitionniste et classique (avec le cas particulier du forcing), puis s'est montrée très adaptée à l'étude des langages de programmation fonctionnels qui peuvent être vus comme des algèbres combinatoires très concrètes. Appliquée à la logique linéaire, l'idée de la réalisabilité mène à celle de la sémantique de phases, qui en donne les modèles les plus simples. L'algèbre combinatoire est alors simplement un monoïde commutatif, dont la loi de composition sert à interpréter l'implication linéaire.

L'apport fondamental de la sémantique de phases réside dans l'interprétation de la négation. L'interprétation naïve de la négation classique consiste à dire que si une formule A est réalisable, elle est vraie et donc $\neg A$ est fausse et donc non réalisable. Comme la négation classique se doit d'être involutive, les seuls modèles qu'on obtient sont alors triviaux (dans le cas intuitionniste, la même interprétation est acceptable puisque la négation n'est plus censée être involutive). En sémantique de phases, l'interprétation de l'absurde (c'est-à-dire la constante $\bot$) n'est plus l'ensemble vide, mais un paramètre duquel se déduit la négation. On dit que x réalise $A^\bot$ si pour tout y réalisant A, xy réalise l'absurde, c'est-à-dire $xy \in \bot$. Cette relation entre x et y est qualifiée d'orthogonalité, et elle est le fondement de toute interprétation de la logique linéaire.

C'est cette orthogonalité qui permet de faire le lien avec la tradition des algèbres de processus. L'idée est d'interpréter $\bot$ autrement : bien qu'il représente logiquement l'absurde, on considère maintenant que ce paramètre définit un ensemble d'objets *valides*, c'est-à-dire qu'il définit la façon de *bien* se comporter. L'assertion $xy \in \bot$ signifie alors que la composition de x et y est valide, donc

que x et y interagissent correctement. L'orthogonal d'un élément x est donc l'ensemble des objets avec qui il interagit bien, c'est-à-dire l'ensemble des tests qu'il passe avec succès, et c'est $\bot$ qui définit la notion de test considérée. On retrouve là le principe des tests dans la sémantique des processus concurrents, où le fait qu'un processus P passe un test Q est défini par une condition uniforme sur $P \mid Q$.

Dans ce chapitre, on développe une réalisabilité pour la logique linéaire fondée sur l'algèbre des termes du $\vec{\pi}$-calcul, donnant corps à cette analogie. Les connecteurs logiques sont interprétés par des propriétés sémantiques élémentaires et implémentés par les constructions syntaxiques du langage, ce qui mène à la définition d'un système de types pour le calcul, avec une correspondance à la Curry-Howard entre calcul et logique.

5.1 Logique

On définit ici un système de typage des processus du π-calcul polarisé. Si le système d'interfaces introduit dans la section 4.3 a pour but de garantir la régularité des communications sur les canaux publics des processus, il ne donne aucune information sur la dynamique des processus. Le système de types développé dans ce chapitre a pour but de structurer l'ensemble des processus de chaque interface en fonction de leur comportement. Dans ce chapitre, on ne manipulera donc que des processus bien interfacés.

Définition 5.1. Les formules sont engendrées par la grammaire suivante :

$$\begin{array}{rl} A, B := & A \otimes B \mid \mathbf{1}_I \mid A \vee B \mid \mathbf{0}_I \mid [\downarrow]A \mid (\exists X : I)A \mid X \\ & A \parr B \mid \bot_I \mid A \wedge B \mid \top_I \mid [\uparrow]A \mid (\forall X : I)A \mid X^\bot \end{array}$$

où I désigne une sorte. La négation est l'involution sur les formules définie syntaxiquement par

$$\begin{array}{cc} (A \otimes B)^\bot := A^\bot \parr B^\bot & (\mathbf{1}_I)^\bot := \bot_{\neg I} \\ (A \vee B)^\bot := A^\bot \wedge B^\bot & (\mathbf{0}_I)^\bot := \top_{\neg I} \\ ((\exists X : I)A)^\bot := (\forall X : I)A^\bot & ([\downarrow]A)^\bot := [\uparrow](A^\bot) \end{array}$$

Pour toutes formules A et B on note $A \multimap B = A^\bot \parr B$.

Ces formules font intervenir des sortes au sens de la définition 4.19 page 62. Comme l'objectif est de décrire les comportements des processus d'une interface donnée, il convient de définir l'interface associée à chaque type.

Définition 5.2. Un jugement de sortage est de la forme $\Phi \Rightarrow A : I$ où A est une formule, I une sorte, et Φ un ensemble d'hypothèses de la forme $X : I$ appelé *environnement*, avec les variables X distinctes. Une formule A est de sorte I si le jugement $\Rightarrow A : I$ est dérivable par les règles de la table 5.1 page ci-contre.

En particulier on a toujours $\Phi \Rightarrow A : I$ si et seulement si $\Phi \Rightarrow A^\bot : \neg I$. Notons que, dans le cas de la quantification, on ne change pas la sorte du quantificateur, puisque la quantification ne change pas de domaine lors d'une négation, ainsi par exemple $((\forall X : I)([\uparrow]X^\bot \parr [\downarrow]X))^\bot = (\exists X : I)([\downarrow]X \otimes [\uparrow]X^\bot)$.

$$\frac{}{\Phi, X : I \Rightarrow X : I} \qquad \frac{c \in \{\mathbf{1}, \bot, \mathbf{0}, \top\}}{\Phi \Rightarrow c_I : I} \qquad \frac{\Phi \Rightarrow A : \varepsilon I, I}{\Phi \Rightarrow [\varepsilon]A : \varepsilon I}$$

$$\frac{\Phi \Rightarrow A : I \qquad \odot \in \{\otimes, \parr\} \qquad \Phi \Rightarrow B : J}{\Phi \Rightarrow A \odot B : I, J}$$

$$\frac{\Phi \Rightarrow A : I \qquad \odot \in \{\vee, \wedge\} \qquad \Phi \Rightarrow B : I}{\Phi \Rightarrow A \odot B : I}$$

$$\frac{\Phi, X : I \Rightarrow A : J}{\Phi \Rightarrow (\forall X : I)A : J} \qquad \frac{\Phi, X : I \Rightarrow A : J}{\Phi \Rightarrow (\exists X : I)A : J}$$

TAB. 5.1 – Sortage des types de processus.

Une formule bien sortée a donc une sorte, en fonction de la sorte attribuée aux variables propositionnelles. Il est facile de voir que pour un environnement Φ donné, pour chaque formule A il existe au plus une sorte I telle que $\Phi \Rightarrow A : I$ soit dérivable. On notera donc $\Phi(A)$ la sorte de A sous les hypothèses Φ, si cette sorte existe.

Définition 5.3. Un type Γ est une suite de la forme $\vec{x}_1 : A_1, \ldots, \vec{x}_n : A_n$, où tous les noms $x_{i,j}$ sont distincts. Un tel type est bien formé dans l'environnement Φ si, pour tout k, la sorte $\Phi(A_k)$ est définie et de même arité que $\vec{x}_k$; on note alors $\Phi \Rightarrow \Gamma$. L'interface d'un type Γ bien formé dans l'environnement Φ est l'interface $\Phi(\Gamma) = \vec{x}_1 : \Phi(A_1), \ldots, \vec{x}_n : \Phi(A_n)$.

Un jugement de typage est de la forme $\Phi \Rightarrow P \vdash \Gamma$. Un processus P est de type Γ dans l'environnement Φ si le jugement $\Phi \Rightarrow P \vdash \Gamma$ est dérivable selon les règles de la table 5.2.

Le sortage des types dans les règles de la table 5.2 peut essentiellement rester implicite, car toutes les règles, à l'exception de l'introduction du quantificateur universel, s'appliquent à environnement fixé. Dans la suite, on s'autorisera donc souvent à garder l'environnement Φ implicite, tout en vérifiant les conditions de bonne formation des types.

Proposition 5.4 (bonne formation)**.** *Pour tout jugement $\Phi \Rightarrow P \vdash \Gamma$ dérivable, on a $\Phi \Rightarrow \Gamma$ et $P :: \Phi(\Gamma)$.*

Démonstration. Découle immédiatement par induction des règles de typage des processus, car les règles de la table 5.2 contiennent toutes les hypothèses nécessaires. □

Remarque 5.5. La règle d'axiome, telle qu'elle est formulée ici, s'applique à une formule A supposée de sorte $\uparrow I$, c'est-à-dire qu'un processus d'un tel type A doit avoir un unique nom libre muni d'une capacité d'émission. C'est une légère restriction destinée à obtenir un système de typage calqué sur la syntaxe des termes du π-calcul polarisé, mais il est possible de généraliser cette règle d'axiome à des formules arbitraires, grâce aux routeurs généralisés de la définition 4.26 page 65 :

$$\frac{\Phi \Rightarrow A : I}{\Phi \Rightarrow \vec{x} \rightarrow^I \vec{y} \vdash \vec{x} : A^\bot, \vec{y} : A}$$

Actions :

$$\frac{}{\Phi \Rightarrow A : \uparrow I} \qquad \frac{\Phi \Rightarrow P \vdash \Gamma, u\vec{x} : A \qquad \Phi \Rightarrow A : \varepsilon I, I}{\Phi \Rightarrow u^{\varepsilon}(\vec{x}).P \vdash \Gamma, u : [\varepsilon]A}$$

Règle d'échange :

$$\frac{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A, \vec{y} : B, \Delta}{\Phi \Rightarrow P \vdash \Gamma, \vec{y} : B, \vec{x} : A, \Delta}$$

Multiplicatifs :

$$\frac{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A \qquad \Phi \Rightarrow Q \vdash \vec{y} : B, \Delta}{\Phi \Rightarrow P \mid Q \vdash \Gamma, \vec{x}\vec{y} : A \otimes B, \Delta} \qquad \frac{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A, \vec{y} : B}{\Phi \Rightarrow P \vdash \Gamma, \vec{x}\vec{y} : A \parr B}$$

Sous-typage :

$$\frac{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A \qquad \Phi(A) = \Phi(B)}{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A \vee B}$$

$$\frac{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A \qquad \Phi \Rightarrow P \vdash \Gamma, \vec{x} : B \qquad \Phi(A) = \Phi(B)}{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A \wedge B}$$

Quantificateurs :

$$\frac{\Phi \Rightarrow P \vdash \Gamma, \vec{u} : A[B/X] \qquad \Phi \Rightarrow B : I}{\Phi \Rightarrow P \vdash \Gamma, \vec{u} : (\exists X : I)A} \qquad \frac{\Phi, X : I \Rightarrow P \vdash \Gamma, \vec{u} : A}{\Phi \Rightarrow P \vdash \Gamma, \vec{u} : (\forall X : I)A}\ X \notin \mathrm{fv}(\Gamma)$$

Constantes :

$$\frac{}{\Phi \Rightarrow 1 \vdash \vec{x} : \mathbf{1}_I} \qquad \frac{\Phi \Rightarrow P \vdash \Gamma}{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : \bot_I} \qquad \frac{\Phi \Rightarrow \Gamma, \top_I \qquad P :: \Phi(\Gamma, \top_I)}{\Phi \Rightarrow P \vdash \Gamma, \top_I}$$

Coupure :

$$\frac{\Phi \Rightarrow P \vdash \Gamma, \vec{u} : A \qquad \Phi \Rightarrow Q \vdash \vec{u} : A^{\perp}, \Delta}{\Phi \Rightarrow (\boldsymbol{\nu}\vec{u})(P \mid Q) \vdash \Gamma, \Delta}$$

Tab. 5.2 – Règles de typage des processus.

5.2 Réalisabilité

Dans cette section, on définit par orthogonalité le genre d'observation qui sera considérée dans la suite. On en déduit une interprétation sémantique des types et des connecteurs, pour laquelle on obtiendra l'adéquation du typage. Dans toute la suite, on ne manipule que des termes bien interfacés, au sens de la définition 4.20.

Comme nous allons le voir, la structure d'orthogonalité permet de définir l'interprétation des types comme des ensembles de processus, appelés comportements, saturés par un certain préordre observationnel. En dehors de la structure de treillis complet de l'ensemble des comportements d'un sorte donnée, toute l'interprétation est fondée sur deux constructions :

- la composition de comportements, définie section 5.2.2, qui décrit les opérateurs de composition parallèle et de restriction,
- une forme de modalité, définie section 5.2.3, qui décrit la dynamique de communication induite par les préfixes dans le calcul.

Par la suite, on utilisera ces deux opérateurs pour construire des connecteurs qui décrivent des comportements plus spécifiques.

5.2.1 Orthogonalité, comportements

On commence par se donner un ensemble $\bot$ de processus, l'observation, qui définit l'orthogonalité.

Définition 5.6 (observation)**.** Une observation est un ensemble $\bot$ de processus d'interface vide clos par bisimulation.

Comme l'orthogonalité définit la validité de l'interaction entre deux processus, elle doit s'appliquer à des paires de processus d'interfaces opposées, c'est pourquoi l'observation contient des processus d'interface vide. En d'autres termes, elle définit une propriété des systèmes clos.

La clôture par bisimulation signifie qu'on ne veut pas distinguer de processus bisimilaires, puisque la bisimilarité est considérée comme l'équivalence sémantique la plus fine, les relations d'équivalence plus restreintes étant considérées comme syntaxiques.

Définition 5.7 (orthogonalité)**.** L'orthogonalité est définie entre processus et ensembles de processus d'interfaces opposées. Pour toute interface $\mathcal{I}$:

- $P :: \mathcal{I}$ et $Q :: \neg\mathcal{I}$ sont orthogonaux si $P \cdot_{\mathcal{I}} Q \in \bot$, on note alors $P \bot Q$;
- $\mathcal{A} :: \mathcal{I}$ et $\mathcal{B} :: \neg\mathcal{I}$ sont orthogonaux si $P \bot Q$ pour tous $P \in \mathcal{A}$ et $Q \in \mathcal{B}$;
- l'orthogonal de $\mathcal{A} :: \mathcal{I}$ est l'ensemble $\mathcal{A}^{\bot} = \{\, Q :: \neg\mathcal{I} \mid \forall P \in \mathcal{A}, P \bot Q \,\}$.

Nous allons voir que les notions classiques de may- et must-testing se définissent naturellement par orthogonalité, et elles illustrent bien le sens que l'on donne à l'observation. La définition standard de test [25, 14], héritée de CCS et étendue aux processus mobiles, consiste à choisir un nom de canal distingué ω et à tester la capacité à agir sur ce canal. Pour définir ce mécanisme formellement, on omet ω dans les interfaces de processus. On définit l'ensemble des processus acceptants comme l'ensemble de ceux capables d'effectuer un action ω, puis on déduit $\bot$ selon le style d'observation voulue : pour le may-testing, $\bot$ est l'ensemble des P tels qu'il existe une réduction $P \to^* Q$ avec Q acceptant ; pour le

must-testing, $\perp$ est l'ensemble des P tels que pour toute réduction $P \to^* Q$ il existe une réduction $Q \to^* R$ avec R acceptant. L'orthogonal d'un ensemble $\mathcal{A}$ est donc l'ensemble des tests passés avec succès par $\mathcal{A}$, ou de façon équivalente l'ensemble des processus qui réussissent les tests de $\mathcal{A}$. On reviendra en détail sur les propriétés de ce genre d'observation dans la section 5.3.

Cette définition d'orthogonalité est standard à ceci près qu'elle n'est définie qu'entre processus d'interfaces opposées. En particulier, elle satisfait les propriétés abstraites suivantes :

Proposition 5.8. *Pour tous ensembles de processus $\mathcal{A} :: \mathcal{I}$ et $\mathcal{B} :: \mathcal{I}$ on a :*

$$\mathcal{A} \subseteq \mathcal{B} \Rightarrow \mathcal{B}^\perp \subseteq \mathcal{A}^\perp \qquad \mathcal{A} \subseteq \mathcal{A}^{\perp\perp} \qquad \mathcal{A}^{\perp\perp\perp} = \mathcal{A}^\perp$$

Pour toute famille d'ensembles $\mathcal{A}_i :: \mathcal{I}$, on a :

$$(\bigcup_i \mathcal{A}_i)^\perp = \bigcap_i \mathcal{A}_i^\perp \qquad (\bigcup_i \mathcal{A}_i)^{\perp\perp} = (\bigcup_i \mathcal{A}_i^{\perp\perp})^{\perp\perp}$$

Démonstration. Par définition, l'orthogonal de $\bigcup_i \mathcal{A}_i$ est l'ensemble des Q tels que $P \perp Q$ pour tout $P \in \mathcal{A}_i$ pour tout i, c'est donc l'ensemble des éléments de chaque $\mathcal{A}_i^\perp$, donc $\bigcap \mathcal{A}_i^\perp$. En particulier, si $\mathcal{A} \subseteq \mathcal{B}$, on a $\mathcal{A} \cup \mathcal{B} = \mathcal{B}$ donc $\mathcal{A}^\perp \cap \mathcal{B}^\perp = \mathcal{B}^\perp$ et donc $\mathcal{B}^\perp \subseteq \mathcal{A}^\perp$. Par définition, pour tous $P \in \mathcal{A}$ et $Q \in \mathcal{A}^\perp$ on a $P \perp Q$ donc $P \perp \mathcal{A}^\perp$ et $\mathcal{A} \subseteq \mathcal{A}^{\perp\perp}$. Appliquée à $\mathcal{A}^\perp$ cette inégalité donne $\mathcal{A}^\perp \subseteq (\mathcal{A}^\perp)^{\perp\perp}$, d'autre part par décroissance on en déduit $(\mathcal{A}^{\perp\perp})^\perp \subseteq \mathcal{A}^\perp$, d'où l'égalité. Enfin, pour toute famille $\mathcal{A}_i$ on a $(\bigcup_i \mathcal{A}_i)^{\perp\perp} = (\bigcap_i \mathcal{A}_i^\perp)^\perp = (\bigcap_i \mathcal{A}_i^{\perp\perp\perp})^\perp = (\bigcup_i \mathcal{A}_i^{\perp\perp})^{\perp\perp}$ en appliquant alternativement la formule de l'union et celle du tri-orthogonal. □

Le passage à l'orthogonal est donc une opération involutive parmi les ensembles qui sont déjà l'orthogonal d'un certain ensemble, en d'autres termes le bi-orthogonal est un opérateur de clôture et le passage à l'orthogonal est involutif sur les ensembles clos. Comme l'orthogonalité correspond à la validité de l'interaction selon une définition paramétrée d'observation, le bi-orthogonal est une clôture par équivalence observationnelle, c'est pourquoi les ensembles clos sont appelés comportements, en accord avec la terminologie de la ludique [39]. Plus précisément, on parlera de comportements *localisés*, parce que chaque comportement est associé à une interface de la forme $x_1 : s_1, \ldots, x_n : s_n$ où les noms $x_1, \ldots, x_n$ sont explicites.

Définition 5.9 (comportement)**.** Soit $\mathcal{I}$ une interface. Un comportement d'interface $\mathcal{I}$ est un ensemble de termes d'interface $\mathcal{I}$ clos par bi-orthogonal. Pour toute interface $\mathcal{I}$, on note $\mathbf{C}_\mathcal{I}$ l'ensemble des comportements d'interface $\mathcal{I}$.

Définition 5.10 (équivalence observationnelle)**.** Soit $\mathcal{A}$ un ensemble de processus d'interface $\mathcal{I}$. Le comportement engendré par $\mathcal{A}$ est $\mathcal{A}^{\perp\perp}$. Le comportement engendré par un processus $P :: \mathcal{I}$ est $[\![P]\!] := \{P\}^{\perp\perp}$. Le préordre observationnel $\sqsubseteq$ est défini par $P \sqsubseteq Q$ si $[\![Q]\!] \subseteq [\![P]\!]$, l'équivalence observationnelle associée est notée $P \simeq Q$.

Dans la section 4.3.3, on a définit une notion de processus anonyme, qui permet de manipuler des processus à renommage près de leurs canaux publics. Pour la même raison, on introduit maintenant les comportements anonymes.

Définition 5.11. Un comportement anonyme de sorte $I = s_1 \dots s_k$ est un comportement d'interface $\{n_i : s_i\}_{1 \leqslant i \leqslant k}$. On note $\mathbf{C}_I$ l'ensemble des comportements anonymes de sorte I.

Notation 5.12. Les comportements anonymes et localisés sont essentiellement les mêmes objets, selon le contexte on utilisera la forme la plus adaptée parmi les deux. On les distingue par la convention typographique suivante : les versions localisées utilisent les lettre calligraphiées, $\mathcal{A}$ pour un comportement et $\mathcal{I}$ pour une interface, et les versions anonymes utilisent les lettres romaines, A pour les comportements et I pour les sortes.

On étend les notations utilisées pour les processus : $(\vec{x})\mathcal{A}$ est le comportement anonyme associé à $\mathcal{A}$ en faisant abstraction des noms libres selon $\vec{x}$, et si A est un comportement anonyme, on note $A[\vec{x}]$ son instanciation avec les noms $\vec{x}$. L'avantage de cette formulation est qu'elle permet de renommer les interfaces implicitement. Ainsi les comportements anonymes sont des ensembles de processus clos par équivalence observationnelle et sans référence à des noms de canaux précis. Ce sont donc les objets que l'on considérera comme des *types*, et qui serviront à interpréter les formules.

La proposition 5.8 montre que l'ensemble des comportements d'une interface donnée est un treillis complet pour l'inclusion, avec l'intersection comme borne inférieure et la clôture de l'union comme borne supérieure. Ainsi, pour toute sorte I et tous comportements $A : I$ et $B : I$ on pose

$$A \wedge B := A \cap B \qquad\qquad A \vee B := (A \cup B)^{\perp\perp} \tag{5.1}$$

Les deux dernières équations de la proposition 5.8 entraînent la dualité entre ces deux opérateurs et leur associativité. On a donc des bornes supérieures et inférieures pour toutes familles de comportements indexées sur des ensembles arbitraires. Pour une borne indexée sur l'ensemble des comportements d'une interface donnée, on obtient la quantification du second ordre :

Définition 5.13. Pour toutes sortes I et J et toute fonction $F : \mathbf{C}_I \to \mathbf{C}_J$, les quantificateurs du second ordre de sorte I sont définis comme

$$(\forall X : I)\, F(X) := \bigwedge_{X \in \mathbf{C}_I} F(X) \qquad (\exists X : I)\, F(X) := \bigvee_{X \in \mathbf{C}_I} F(X) \tag{5.2}$$

Dans chaque treillis de comportements, on définit les quatre constantes de la logique linéaire, comme on le fait dans les espaces de phases. Notons que cette définition nécessite l'utilisation du processus neutre 1 :

Définition 5.14. Pour toute sorte I, on pose

$$\begin{aligned} \top_I &= \emptyset^{\perp} = (\exists X : I)X & \qquad \bot_I &= \{1\}^{\perp} \\ \mathbf{0}_I &= \emptyset^{\perp\perp} = (\forall X : I)X & \qquad \mathbf{1}_I &= \{1\}^{\perp\perp} \end{aligned} \tag{5.3}$$

En d'autres termes, $\top_{\mathcal{I}}$ est l'ensemble des processus d'interface $\mathcal{I}$, élément maximum de $\mathbf{C}_{\mathcal{I}}$, et $\mathbf{0}_{\mathcal{I}}$ est le plus petit comportement d'interface $\mathcal{I}$, qui peut être vide ou non selon la définition de $\bot$. Le comportement $\mathbf{1}_{\mathcal{I}}$ est la clôture du processus neutre 1, son dual noté $\bot_{\mathcal{I}}$ est l'ensemble des processus P tels que $P \cdot 1 \in \bot$, qui est équivalent à $(\boldsymbol{\nu}\vec{x})P \in \bot$ si $\vec{x}$ est le support de l'interface $\mathcal{I}$. La notation n'est pas une coïncidence, car l'observation $\bot$ est effectivement

le comportement $\bot_\emptyset$, puisque $\bot$ est clos par congruence structurelle, ce qui implique que $P \in \bot$ est équivalent à $P|1 \in \bot$. On notera bien que, contrairement à la notation traditionnelle pour les ordres, mais conformément aux notations de la logique linéaire, $\bot_\mathcal{I}$ n'est donc pas en général le minimum du treillis $\mathbf{C}_\mathcal{I}$.

Exemple 5.15. Deux cas d'observations sont à noter : l'observation triviale est celle où $\bot$ est l'ensemble de tous les termes clos, dans ce cas la condition d'orthogonalité est toujours vérifiée et l'unique comportement de chaque interface $\mathcal{I}$ est $\top_\mathcal{I}$. L'autre extrême est le cas où $\bot$ est vide, et dans ce cas deux éléments ne sont jamais orthogonaux ; il reste alors exactement deux comportements dans chaque interface, le vide $\mathbf{0} = \bot$ et le plein $\mathbf{1} = \top$. Dans ce cas on obtient un modèle booléen, qui interprète les formules de la logique linéaire comme des formules classiques.

Proposition 5.16. *Soient une interface $\mathcal{I}$, un comportement $\mathcal{A} : \mathcal{I}$ et deux processus $P :: \mathcal{I}$ et $Q :: \mathcal{I}$ bisimilaires. Alors $P \in \mathcal{A}$ si et seulement si $Q \in \mathcal{A}$.*

Démonstration. Soit $\mathcal{A}$ un comportement d'une interface $\mathcal{I}$ donnée. Soit $P \in \mathcal{A}$ et soit $Q :: \mathcal{I}$ tel que $P \cong Q$. Soit $R \in \mathcal{A}^\bot$, alors comme la bisimulation est une congruence on a $(\boldsymbol{\nu}\mathcal{I})(P \,|\, R) \cong (\boldsymbol{\nu}\mathcal{I})(Q \,|\, R)$, donc $P \cdot R$ et $Q \cdot R$ sont bisimilaires. Par hypothèse $\bot$ est clos par bisimulation, or $P \cdot R \in \bot$ donc $Q \cdot R \in \bot$, et par conséquent $Q \perp \mathcal{A}^\bot$ et $Q \in \mathcal{A}^{\bot\bot} = \mathcal{A}$. □

Cette proposition montre donc qu'au sein d'une interface donnée on a une inclusion entre les trois principales équivalences sur les termes : la congruence structurelle $\equiv$ implique la bisimilarité $\cong$ qui implique l'équivalence observationnelle $\simeq$, dès que $\bot$ est clos par bisimulation.

La structure de treillis complet de l'ensemble des comportements d'une interface donnée fournit immédiatement une famille de connecteurs pour représenter les opérations de borne inférieure et de borne supérieure. Sous forme binaire, elles sont l'interprétation de $A \wedge B$ et $A \vee B$ respectivement, et la propriété 5.8 garantit qu'ils sont associatifs et commutatifs, avec $\top$ et $\mathbf{0}$ comme éléments neutres respectifs.

5.2.2 Multiplicatifs : connecteurs spatiaux

Dans cette section, on suppose choisie une observation $\bot$ quelconque. Bien que la définition 5.6 impose à $\bot$ d'être clos par bisimilarité, pour la plupart des énoncés de cette section il suffit que $\bot$ soit clos par congruence structurelle.

L'opérateur fondamental entre processus est la composition typée, au sens de la définition 4.23. Cette composition sert à définir l'orthogonalité quand elle s'applique à des interfaces opposées, mais elle définit plus généralement la composition de comportements.

Définition 5.17. Soient $\mathcal{I}, \mathcal{J}, \mathcal{K}$ trois interfaces disjointes et soient $\mathcal{A} : \mathcal{I} \to \mathcal{J}$ et $\mathcal{B} : \mathcal{J} \to \mathcal{K}$ deux ensembles de termes. La composition de $\mathcal{A}$ est $\mathcal{B}$ est le comportement

$$\mathcal{A} \cdot_\mathcal{J} \mathcal{B} := \{\, P \cdot_\mathcal{J} Q \mid P \in \mathcal{A}, Q \in \mathcal{B} \,\}^{\bot\bot} \tag{5.4}$$

L'interface de $\mathcal{A} \cdot_\mathcal{J} \mathcal{B}$ est $\mathcal{I} \to \mathcal{K}$.

L'orthogonalité est définie à partir de la composition de processus à canaux typés. L'associativité de cette composition entraîne en particulier la propriété d'adjonction suivante :

Lemme 5.18. *Soient $\mathcal{I}, \mathcal{J}, \mathcal{K}$ trois interfaces disjointes et soient $P :: \mathcal{I} \to \mathcal{J}$, $Q :: \mathcal{J} \to \mathcal{K}$ et $R :: \mathcal{K} \to \mathcal{I}$, alors $P \cdot Q \perp R$ si et seulement si $P \perp Q \cdot R$.*

Démonstration. Notons $\vec{x}$, $\vec{y}$ et $\vec{z}$ les supports respectifs des interfaces $\mathcal{I}$, $\mathcal{J}$ et $\mathcal{K}$, disjoints par hypothèse. On a alors $(P \cdot_{\mathcal{J}} Q) \cdot_{\mathcal{I}\to\mathcal{K}} R = (\boldsymbol{\nu}\vec{x}\vec{z})((\boldsymbol{\nu}\vec{y})(P \mid Q) \mid R) \equiv (\boldsymbol{\nu}\vec{x}\vec{y}\vec{z})(P \mid Q \mid R) \equiv (\boldsymbol{\nu}\vec{x}\vec{y})(P \mid (\boldsymbol{\nu}\vec{z})(Q \mid R)) = P \cdot_{\mathcal{I}\to\mathcal{J}} (Q \cdot_{\mathcal{K}} R)$, donc $(P \cdot_{\mathcal{J}} Q) \cdot_{\mathcal{I}\to\mathcal{K}} R \in \perp$ si et seulement si $P \cdot_{\mathcal{I}\to\mathcal{J}} (Q \cdot_{\mathcal{K}} R) \in \perp$ puisque $\perp$ est supposé clos par congruence structurelle. □

Lemme 5.19. *Pour tous ensembles de processus $\mathcal{A} : \mathcal{I} \to \mathcal{J}$ et $\mathcal{B} : \mathcal{J} \to \mathcal{K}$ on a $\mathcal{A} \cdot_{\mathcal{J}} \mathcal{B} = \mathcal{A}^{\perp\perp} \cdot_{\mathcal{J}} \mathcal{B}^{\perp\perp}$.*

Démonstration. Par construction l'opérateur $\cdot$ est monotone, donc l'inclusion directe est immédiate. La réciproque peut s'énoncer $(\mathcal{A} \cdot \mathcal{B})^{\perp} \subseteq (\mathcal{A}^{\perp\perp} \cdot \mathcal{B}^{\perp\perp})^{\perp}$. Pour tout $R \in (\mathcal{A} \cdot \mathcal{B})^{\perp}$ on a $R :: \mathcal{K} \to \mathcal{I}$, donc pour tous $P \in \mathcal{A}$ et $Q \in \mathcal{B}$, par le lemme 5.18, $R \perp P \cdot_{\mathcal{J}} Q$ implique $R \cdot_{\mathcal{K}} Q \perp P$ donc $R \cdot_{\mathcal{K}} Q \in \mathcal{A}^{\perp} = \mathcal{A}^{\perp\perp\perp}$, donc pour tout $P \in \mathcal{A}^{\perp\perp}$ on a $R \cdot_{\mathcal{K}} Q \perp P$. Par le lemme 5.18 on a donc $R \cdot_{\mathcal{I}} P \perp Q$, d'où on déduit de la même façon que pour tout $Q \in \mathcal{B}^{\perp\perp}$ on a $R \cdot_{\mathcal{I}} P \perp Q$ et donc $R \perp P \cdot_{\mathcal{J}} Q$, et donc $R \in (\mathcal{A}^{\perp\perp} \cdot \mathcal{B}^{\perp\perp})^{\perp}$. □

Cette proposition illustre une propriété importante de la composition : la positivité. Il s'agit d'une propriété des opérateurs qui sont définis comme la clôture d'un ensemble défini syntaxiquement. Les autres opérateurs positifs déjà définis sont la borne supérieure $\vee$ et la quantification existentielles, qui sont clôtures de l'union, et la dernière équation de la proposition 5.8 exprime que ces deux opérateurs sont aussi invariants par clôture observationnelle des arguments. Un intérêt de la positivité est que, lors de l'application de plusieurs opérateurs positifs, il suffit de calculer la clôture une fois après avoir raisonné syntaxiquement. C'est par ce moyen que l'on démontre les propriétés élémentaires de la composition :

Proposition 5.20. *La composition $\cdot$ est*
- *croissante en ses deux variables ;*
- *distributive sur $\vee$;*
- *commutative et associative.*

Démonstration. Pour tous $\mathcal{A} : \mathcal{I} \to \mathcal{J}$ et $\mathcal{B} : \mathcal{J} \to \mathcal{K}$, notons $f_{\mathcal{J}}(\mathcal{A}, \mathcal{B}) = \{ P \cdot_{\mathcal{J}} Q \mid P \in \mathcal{A}, Q \in \mathcal{B} \}$. Cette fonction $f_{\mathcal{J}}$ est l'extension de la composition de processus aux ensembles de processus, donc elle est croissante en ses deux variables et elle commute aux unions arbitraires. Par définition, on a $\mathcal{A} \cdot \mathcal{B} = f_{\mathcal{J}}(\mathcal{A}, \mathcal{B})^{\perp\perp}$, et l'opérateur de clôture est croissant, donc $\cdot$ est croissant. Soit $\mathcal{B}_i : \mathcal{J} \to \mathcal{K}$ une famille d'ensembles de processus, on a alors

$$\begin{array}{lll}
\mathcal{A} \cdot \bigvee_i \mathcal{B}_i & = \mathcal{A} \cdot \left(\bigcup_i \mathcal{B}_i\right)^{\perp\perp} & \text{par définition de } \vee \\
& = \mathcal{A} \cdot \bigcup_i \mathcal{B}_i & \text{par le lemme 5.19} \\
& = \left(\bigcup_i f_{\mathcal{J}}(\mathcal{A}, \mathcal{B}_i)\right)^{\perp\perp} & \text{par commutation de } f_{\mathcal{J}} \text{ sur l'union} \\
& = \left(\bigcup_i \mathcal{A} \cdot \mathcal{B}_i\right)^{\perp\perp} & \text{par la proposition 5.8} \\
& = \bigvee_i \mathcal{A} \cdot \mathcal{B}_i & \text{par définition de } \vee
\end{array}$$

Enfin, l'associativité et la commutativité découlent des mêmes propriétés de la composition des processus modulo congruence structurelle. □

La commutation avec les bornes supérieures s'étend au cas de la constante additive $\mathbf{0}$, qui est absorbante pour la composition, dans le sens que pour tout ensemble $\mathcal{A} : \mathcal{I} \to \mathcal{J}$ on a $\mathcal{A} \cdot \mathbf{0}_{\mathcal{J}\to\mathcal{K}} = \mathbf{0}_{\mathcal{I}\to\mathcal{K}}$. D'autre part la constante multiplicative $\mathbf{1}$ est neutre, au sens où $\mathcal{A} \cdot \mathbf{1}_\emptyset = \mathcal{A}$ pour tout comportement $\mathcal{A}$. En revanche, en général la composition ne commute pas avec les intersections. Étant donnée une famille de comportements $\mathcal{B}_i$, la monotonie permet de déduire $\mathcal{A} \cdot \bigcap_i \mathcal{B}_i \subseteq \mathcal{A} \cdot \mathcal{B}_j$ pour chaque j, donc $\mathcal{A} \cdot \bigcap_i \mathcal{B}_i \subseteq \bigcap_j \mathcal{A} \cdot \mathcal{B}_j$, mais l'inclusion peut être stricte pour certaines observations et certains comportements. Résumons donc, pour $\mathcal{A} : \mathcal{I} \to \mathcal{J}$ et $B_i : \mathcal{J} \to \mathcal{K}$:

$$\mathcal{A} \cdot \mathbf{0}_{\mathcal{J}\to\mathcal{K}} = \mathbf{0}_{\mathcal{I}\to\mathcal{K}} \qquad \mathcal{A} \cdot \mathbf{1}_\emptyset = \mathcal{A} \qquad \mathcal{A} \cdot \bigcap_i \mathcal{B}_i \subseteq \bigcap_j \mathcal{A} \cdot \mathcal{B}_j \tag{5.5}$$

Définition 5.21. Pour toute paire d'interfaces disjointes $\mathcal{I}$ et $\mathcal{J}$ et tous comportements $\mathcal{A} : \mathcal{I}$ et $\mathcal{B} : \mathcal{J}$, on pose

$$\mathcal{A} \otimes \mathcal{B} := \mathcal{A} \cdot_\emptyset \mathcal{B} \qquad \mathcal{A} \parr \mathcal{B} := (\mathcal{A}^\perp \otimes \mathcal{B}^\perp)^\perp \qquad \mathcal{A} \multimap \mathcal{B} := (\mathcal{A} \otimes \mathcal{B}^\perp)^\perp \tag{5.6}$$

Le tenseur est donc le cas particulier de la composition où les deux comportements ont des interfaces disjointes, c'est-à-dire qu'il correspond à la composition parallèle (sans restriction) de processus qui ne partagent aucun nom. Il s'agit en quelque sorte d'une conjonction séparante, comme il en existe dans les logiques spatiales [16], même si le lien précis avec la séparation géométrique de sous-processus dépend fortement de l'observation considérée. Son dual $\parr$ est donc une forme de disjonction, qu'il faut comprendre comme une forme symétrique du connecteur $\multimap$ d'implication spatiale, lequel se caractérise de la façon suivante :

Proposition 5.22. *Soient $\mathcal{A} : \mathcal{I}$ et $\mathcal{B} : \mathcal{J}$ deux comportements d'interfaces disjointes, alors pour tout $P :: \mathcal{I} \to \mathcal{J}$ on a $P \in \mathcal{A} \multimap \mathcal{B}$ si et seulement si pour tout $Q \in \mathcal{A}$, $P \cdot_\mathcal{I} Q \in \mathcal{B}$.*

Démonstration. $P \in \mathcal{A} \multimap \mathcal{B}$ est équivalent à $P \perp Q \cdot_\emptyset R$ pour tout $Q \in \mathcal{A}$ et $R \in \mathcal{B}^\perp$. D'après le lemme 5.18, $P \perp Q \cdot_\emptyset R$ si et seulement si $P \cdot_\mathcal{I} Q \perp R$, donc $P \in \mathcal{A} \multimap \mathcal{B}$ est équivalent à $P \cdot_\mathcal{I} Q \perp \mathcal{B}^\perp$, donc à $P \cdot_\mathcal{I} Q \in \mathcal{B}$. □

C'est à cause de cette propriété que la disjonction $\parr$ est la bonne forme de passage au contexte. En particulier, on a la règle suivante de passage au contexte de la composition :

Proposition 5.23. *Soient $\mathcal{I}$, $\mathcal{J}$, $\mathcal{K}$, $\mathcal{L}$ et $\mathcal{M}$ cinq interfaces disjointes. Soient quatre comportements $\mathcal{A} : \mathcal{I}$, $\mathcal{B} : \mathcal{K} \to \mathcal{L}$, $\mathcal{C} : \mathcal{J}$ et $\mathcal{D} : \mathcal{L} \to \mathcal{M}$, alors on a $(\mathcal{A} \multimap \mathcal{B}) \cdot_\mathcal{L} (\mathcal{C} \multimap \mathcal{D}) \subseteq (\mathcal{A} \otimes \mathcal{C}) \multimap (\mathcal{B} \cdot_\mathcal{L} \mathcal{D})$.*

Démonstration. Soient $P \in \mathcal{A} \multimap \mathcal{B}$ et $Q \in \mathcal{C} \multimap \mathcal{D}$. Pour tous $R \in \mathcal{A}$ et $S \in \mathcal{C}$ on a $P \cdot_\mathcal{I} R \in \mathcal{B}$ et $Q \cdot_\mathcal{J} S \in \mathcal{D}$ donc $(P \cdot_\mathcal{I} R) \cdot_\mathcal{L} (Q \cdot_\mathcal{J} S) \in \mathcal{B} \cdot \mathcal{D}$, ce qui permet de conclure par associativité des compositions grâce à la proposition 5.22. □

Cet énoncé contient comme cas particuliers le passage au contexte du tenseur et la règle de coupure :

Corollaire 5.24. *Soient $\mathcal{A} : \mathcal{I}$, $\mathcal{B} : \mathcal{J}$, $\mathcal{C} : \mathcal{K}$ et $\mathcal{D} : \mathcal{L}$ quatre comportements d'interfaces disjointes, alors on a les implications*

$$\frac{P \in \mathcal{A} \multimap \mathcal{B} \qquad Q \in \mathcal{C} \multimap \mathcal{D}}{P \mid Q \in (\mathcal{A} \otimes \mathcal{C}) \multimap (\mathcal{B} \otimes \mathcal{D})} \qquad \frac{P \in \mathcal{A} \multimap \mathcal{B} \qquad Q \in \mathcal{B} \multimap \mathcal{C}}{P \cdot_\mathcal{J} Q \in \mathcal{A} \multimap \mathcal{C}}$$

Le comportement de l'implication face aux constantes négatives se déduit de celui de la composition face aux constantes positives. Ainsi, pour tout comportement $\mathcal{A} : \mathcal{I}$, on a :

$$\mathcal{A} \parr \top_{\mathcal{J}} = \top_{\mathcal{I}\cup\mathcal{J}} \qquad\qquad \mathcal{A} \parr \bot_\emptyset = \mathcal{A} \tag{5.7}$$

Les opérateurs $\otimes$ et $\parr$ sont respectivement une conjonction et une disjonction, avec par construction une dualité à la de Morgan. La règle classique $A \wedge B \Rightarrow A \vee B$ se formule ici $A \otimes B \subseteq A \parr B$, elle est connue en logique linéaire sous le nom de règle *mix* [29], et son contenu calculatoire est considéré comme concurrent dans la mesure où il autorise des réseaux de preuve non connexes, dont plusieurs composantes se réduisent indépendamment. La validité de cette règle dépend de l'observation :

Proposition 5.25. *Les trois énoncés suivants sont équivalents :*
- *$A \otimes B \subseteq A \parr B$ pour tous A et B (règle mix),*
- *$\bot_\emptyset \cdot \bot_\emptyset = \bot_\emptyset \otimes \bot_\emptyset \subseteq \bot_\emptyset$ ($\bot$ est stable par composition),*
- *$\bot_\emptyset \subseteq \mathbf{1}_\emptyset$.*

Démonstration. Le fait que mix implique la stabilité de $\bot$ par composition est immédiat du fait que $\bot_\emptyset$ est neutre pour $\parr$, l'implication réciproque découle de la propriété 5.23 en écrivant $A \otimes B = (A \parr \bot_\emptyset) \otimes (B \parr \bot_\emptyset) \subseteq A \parr B \parr (\bot_\emptyset \otimes \bot_\emptyset) \subseteq A \parr B \parr \bot_\emptyset = A \parr B$. Le fait que $\bot_\emptyset \subseteq \mathbf{1}_\emptyset$ implique $\bot_\emptyset \otimes \bot_\emptyset \subseteq \bot_\emptyset$ découle du fait que $\mathbf{1}_\emptyset$ est neutre pour $\otimes$ et de la monotonie de ce connecteur, pour l'implication réciproque on remarque que $\bot_\emptyset \otimes \bot_\emptyset \subseteq \bot_\emptyset$ implique que pour tous P et $Q \in \bot_\emptyset$ on a $P \cdot Q \in \bot$, c'est-à-dire $\bot_\emptyset \subseteq (\bot_\emptyset)^\bot = \mathbf{1}_\emptyset$. □

Les multiplicatifs tels qu'ils sont définis ici sont des opérateurs partiels, puisqu'ils imposent à leurs deux opérandes d'avoir des interfaces disjointes. Ce petit problème peut être résolu par l'utilisation de comportements anonymes, le maigre prix à payer étant que $\otimes$ et $\parr$ ne sont plus commutatifs qu'à un isomorphisme trivial près. L'utilisation de comportements anonymes permet de plus de définir une structure de composition propre.

Définition 5.26. La catégorie des comportements a pour objets l'ensemble des sortes. Les morphismes de I vers J sont les comportements anonymes de sorte $I \to J$. La composition de $A : I \to J$ et $B : J \to K$ est le comportement anonyme défini par

$$(A;B)[\vec{x}\vec{z}] := A[\vec{x}\vec{y}] \cdot_{\vec{y}} B[\vec{y}\vec{z}]$$

où $\vec{x}$, $\vec{y}$, $\vec{z}$ sont des vecteurs de noms distincts de l'arité de I, J, K. Pour toute sorte I, l'identité sur I est le comportement

$$\mathrm{id}_I[\vec{x}\vec{y}] := \{\, \vec{x} \to^I \vec{y} \,\}^{\bot\bot}$$

L'associativité de la composition vient de la proposition 5.20, et les propriétés de l'identité découlent de la proposition 4.27 page 65 qui relie connexion et substitution des noms. Notons que le fait que id_I soit l'identité utilise la clôture des comportements par bisimulation ; c'est le seul point de cette section qui utilise plus que la congruence structurelle.

La définition des connecteurs multiplicatifs s'adapte en posant

$$(A \otimes B)[\vec{x}\vec{y}] := A[\vec{x}] \otimes B[\vec{y}] \qquad\qquad (A \parr B)[\vec{x}\vec{y}] := A[\vec{x}] \parr B[\vec{y}]$$

Donc $\otimes$ et $\parr$ associent à deux comportements de sorte I et J un comportement de sorte I, J, et pour $A : I$ et $B : J$ le comportement $A \multimap B$ est de sorte $I \to J$.

5.2.3 Modalités : connecteurs temporels

Les connecteurs multiplicatifs sont ceux qui décrivent l'interaction entre processus indépendants, c'est-à-dire le comportement de la composition, de façon très générale et indépendante de l'observation. Dans cette section, on définit des opérateurs qui décrivent les actions possibles d'un processus : il s'agit de modalités dans le style de la logique de Hennessy-Milner [64], adaptées au cas de notre calcul.

De même que la composition est associée aux opération de composition parallèle et de restriction, les modalités sont associées aux actions élémentaires du calcul, de la forme $u^{\varepsilon}(\vec{x})$. Un point important dans la définition qui suit est l'effet de ces modalités sur les interfaces : on impose en fait que l'interface d'un comportement $[u^{\varepsilon}(\vec{x})]\mathcal{A}$ soit limitée à l'unique nom u, avec la polarité ε. La modalité $[u^{\varepsilon}(\vec{x})]$ agit donc comme un lieur pour les noms $\vec{x}$, et elle s'applique à des comportements dont l'interface contient uniquement u et $\vec{x}$. Sous forme anonyme, on la note simplement $[\varepsilon]$, quelle que soit la sorte de la forme $\varepsilon I, I$ où elle s'applique.

Deux aspects du comportement d'un processus $u(x).P$ sont à considérer pour définir les modalités en conséquence : d'une part, bien entendu, un tel processus atteint P après une transition étiquetée $u(x)$, c'est-à-dire qu'il est orthogonal à tout $\bar{u}(x).Q$ avec $P \perp Q$. D'autre part, $u(x).P$ est inactif s'il n'interagit pas sur u, c'est-à-dire qu'il est indistinguable de 1 si u est rendu privé, donc $u(x).P \cdot (1 :: u) \in \mathbf{1}_{\emptyset}$. Cette deuxième condition est équivalente à $u(x).P \in \mathbf{1}_{\bar{u}} \multimap \mathbf{1}_{\emptyset}$ ou encore $u(x).P \in \perp_u \parr \mathbf{1}_{\emptyset}$, elle signifie que $u(x).P$ ne doit pas diverger, si l'on interprète intuitivement l'appartenance à $\perp$ comme une sorte de convergence.

Définition 5.27. Soient I une sorte et ε une polarité. La modalité de polarité ε et de sorte I est l'opérateur $[\varepsilon] : \mathbf{C}_{\varepsilon I, I} \to \mathbf{C}_{\varepsilon I}$ défini pour $A : \varepsilon I, I$ par

$$[\varepsilon]A = (u)\big\{\, \bar{u}^{\varepsilon}(\vec{x}).P \;\big|\; P \in A[u\vec{x}]^{\perp} \,\big\}^{\perp} \tag{5.8}$$

Ces modalités décrivent des étapes d'interaction, il faut donc que l'orthogonalité prenne en compte la réduction des processus ; plus précisément, il faut une condition analogue à la clôture par anti-réduction des $\perp\!\!\!\perp$ de la réalisabilité classique étudiée aux chapitres 2 et 3. À la différence du cas séquentiel, cette condition peut s'interpréter de différentes manières dans un cadre concurrent et non déterministe, selon le type de propriété à observer (may-testing, must-testing, etc.). On donne ici une condition abstraite sur $\perp$ qui garantit l'adéquation du typage :

Définition 5.28. Une observation $\perp$ est régulière si

- pour tous processus $P :: \mathcal{I}$ et $Q :: \neg\mathcal{I}$ tels que P a une unique transition $P \xrightarrow{\tau} P'$ avec $P' \perp Q$, on a $P \perp Q$;
- pour toute sorte I, toute polarité ε et tout comportement $A : \varepsilon I, I$, on a $[\varepsilon]A = [\neg\varepsilon]A^{\perp}$.

La première condition est la clôture par anti-réduction, sous une forme restreinte. Elle signifie que si un processus P ne peut que se réduire en un terme d'un comportement $\mathcal{A}$ alors P lui-même est élément du comportement $\mathcal{A}$. En d'autres termes, elle permet, lors du passage d'un test, de consommer les communications déterministes au sein d'un processus. La deuxième condition impose

la dualité entre modalités, elle signifie qu'un processus orthogonal à $u(x).P$ doit essentiellement se comporter comme un $\bar{u}(x).Q$ avec $P \perp Q$.

Proposition 5.29. *Si une observation $\perp$ vérifie*
- *pour tous $P :: \mathcal{I}$ et $Q :: \mathcal{I}$, $P \perp Q$ implique $\tau.P \perp Q$,*
- *pour tout ε et tout comportement $A : \varepsilon I, I$, $[\varepsilon]A \perp [\neg\varepsilon]A^{\perp}$,*

alors $\perp$ est régulière.

Démonstration. Soit $\perp$ une observation vérifiant les conditions voulues, et soit A un comportement de sorte $\varepsilon I, I$. Par définition de la régularité des observations, on a $[\varepsilon]A \perp [\neg\varepsilon]A^{\perp}$ donc $[\neg\varepsilon]A^{\perp} \subseteq ([\varepsilon]A)^{\perp}$. D'autre part, pour tout $P \in A[u\vec{x}]$ et tout $Q \in A[u\vec{x}]^{\perp}$ le processus $(\boldsymbol{\nu} u)(u^{\varepsilon}(\vec{x}).P \,|\, \bar{u}^{\varepsilon}(\vec{x}).Q)$ a une unique transition, étiquetée τ, qui mène à $(\boldsymbol{\nu} u\vec{x})(P \mid Q)$, or ce processus est élément de $\perp_{\emptyset}$ par hypothèse, donc $(\boldsymbol{\nu} u)(u^{\varepsilon}(\vec{x}).P \mid \bar{u}^{\varepsilon}(\vec{x}).Q)$ est aussi élément de $\perp_{\emptyset}$, d'après la première condition de validité de $\perp$. Ceci entraîne $u^{\varepsilon}(\vec{x}).P \perp \bar{u}^{\varepsilon}(\vec{x}).Q$, et donc $\{\, \bar{u}^{\varepsilon}(\vec{x}).Q \mid Q \in A[u\vec{x}]^{\perp} \,\} \subseteq \{\, u^{\varepsilon}(\vec{x}).P \mid P \in A[u\vec{x}] \,\}^{\perp}$. En passant au bi-orthogonal on déduit $([\varepsilon]A)^{\perp} \subseteq [\neg\varepsilon](A^{\perp})$, ce qui donne l'égalité. □

Proposition 5.30. *Pour toute observation régulière, les modalités distribuent sur les bornes inférieures et supérieures.*

Démonstration. Soient ε une polarité et I une sorte. Soit (A_i) une famille de comportements de sorte $\varepsilon I, I$. On a alors

$$\begin{aligned}[\varepsilon]\textstyle\bigcap_i A_i &= (u)\{\, \bar{u}^{\varepsilon}(\vec{x}).P \mid P \in \textstyle\bigvee_i A_i[u\vec{x}]^{\perp} \,\}^{\perp} \\ &= (u)\textstyle\bigcap_i \{\, \bar{u}^{\varepsilon}(\vec{x}).P \mid P \in A_i[u\vec{x}]^{\perp} \,\}^{\perp} = \textstyle\bigcap_i [\varepsilon]A_i\end{aligned}$$

Et l'auto-dualité permet d'en déduire la distribution sur la borne supérieure par passage au dual. □

La définition des modalités entraîne immédiatement que pour $P \in A[u\vec{x}]$ on a $u^{\varepsilon}(\vec{x}).P \in u : [\varepsilon]A$, de plus cette propriété passe au contexte :

Proposition 5.31. *Soient $A : I$ et $B : \varepsilon J, J$ deux comportements anonymes. Pour tout $P \in A[\vec{u}] \multimap B[v\vec{x}]$ on a $v^{\varepsilon}(\vec{x}).P \in A[\vec{u}] \multimap ([\varepsilon]B)[v]$.*

Démonstration. Soit $P \in A[\vec{u}] \multimap B[v\vec{x}]$, notons $P' = v^{\varepsilon}(\vec{x}).P$. Pour tout $R \in B[v\vec{x}]^{\perp}$, le processus $(\boldsymbol{\nu} v)(P' \mid v^{\neg\varepsilon}(\vec{x}).R)$ a une unique τ-transition qui mène à $P \cdot R$, et d'après la proposition 5.23 ce processus est élément de $A[\vec{u}]^{\perp}$. Pour tout $Q \in A[\vec{u}]$, on a donc $(\boldsymbol{\nu}\vec{u}v)(P' \mid Q \mid v^{\neg\varepsilon}(\vec{x}).R) \in \perp$ et donc $P' \cdot Q \perp \{\, v^{\neg\varepsilon}(\vec{x}).R \mid R \in B[v\vec{x}]^{\perp} \,\}$. Par conséquent on a $P' \cdot Q \in ([\varepsilon]B)[v]$ et $P' \in A[\vec{u}] \multimap ([\varepsilon]B)[v]$. □

5.2.4 Adéquation

Grâce à la définition des comportements et des connecteurs élémentaires, on peut formuler l'adéquation du système de types et de la sémantique des processus. L'énoncé obtenu est l'analogue du théorème 2.7 page 17 qui exprime l'adéquation dans le cadre séquentiel.

Définition 5.32. On suppose donnée une observation régulière. Une valuation de type $\Phi = X_1 : I_1, \ldots, X_n : I_n$ est la donnée, pour chaque variable propositionnelle X_k, d'un comportement anonyme $[\![X_k]\!] \in \mathbf{C}_{I_k}$. Une valuation s'étend

aux formules bien sortées dans Φ en interprétant chaque connecteur par l'opérateur entre comportements associé, avec

$$[\![(\forall X : I).A]\!] = \bigwedge_{[\![X]\!] \in \mathbf{C}_I} [\![A]\!] \qquad \text{et} \qquad [\![(\exists X : I).A]\!] = \bigvee_{[\![X]\!] \in \mathbf{C}_I} [\![A]\!]$$

L'interprétation d'un type bien formé $\Phi \Rightarrow \Gamma$ est le comportement d'interface $\Phi(\Gamma)$ défini par

$$[\![\vec{x}_1 : A_1, \ldots, \vec{x}_n : A_n]\!] = [\![A_1]\!][\vec{x}_1] \parr \cdots \parr [\![A_n]\!][\vec{x}_n]$$

Pour une valuation donnée, Un processus P réalise un type Γ s'il est élément de l'interprétation de Γ, on note alors $P \Vdash \Gamma$.

En rassemblant les nombreux lemmes et propositions de cette section, ainsi que les conditions qu'ils imposent sur $\bot$, on obtient le théorème d'adéquation suivant :

Théorème 5.33. *Si un jugement $\Phi \Rightarrow P \vdash \Gamma$ est dérivable, alors pour toute observation régulière et toute valuation de type Φ on a $P \Vdash \Gamma$.*

Démonstration. On raisonne par induction sur les règles de typage :
- La règle d'axiome s'écrit $\vec{x} \to^I \vec{y} \Vdash A[\vec{x}] \to A[\vec{y}]$ pour A de sorte I, sa validité est conséquence de la proposition 4.27.
- La règle d'introduction des actions est la proposition 5.31.
- La règle d'échange découle de la commutativité de $\parr$.
- Les règles tenseur et coupure sont valides d'après la proposition 5.23.
- L'adéquation de la borne supérieure et de la quantification existentielle sont conséquences de la croissance de $\parr$. Pour la quantification, il suffit d'instancier la variable X à la valeur de la formule qui lui est substituée.
- L'adéquation de l'intersection et de la quantification universelle sont assurées par le fait que $\parr$ commute avec l'intersection, dans le cas de la quantification on utilise le fait que l'hypothèse d'induction est vérifiée pour toute valuation, donc toute valeur de la variable sur laquelle on quantifie.
- La règle pour $\mathbf{1}$ est valide par définition de $\mathbf{1}$.
- La règle pour $\top$ découle du fait que $\top$ est absorbant pour $\parr$ et que $\top_I$ contient tous les processus de sorte I.
- Pour la règle du $\bot_I$, pour tout $P \Vdash \Gamma$ et tout vecteur de noms $\vec{x}$ indépendant de Γ on a $P \equiv (\boldsymbol{\nu}\vec{x})P \equiv P \cdot 1_I$ donc par la proposition 5.22 P est élément de $\mathbf{1}_I \multimap \Gamma$, c'est-à-dire de $\Gamma \parr \bot_I$.

Toutes les règles d'inférence sont donc valides. □

Comme on le verra dans la section 5.3.4, la régularité n'est pas une condition nécessaire pour obtenir l'adéquation. Plus généralement, on qualifiera de *valide* toute observation qui donne l'adéquation des règles de la table 5.2.

5.3 Observations

La condition de régularité des observations formulée dans la définition 5.28 est assez abstraite. Pour mieux comprendre sa signification, et ainsi comprendre à quelles sémantiques des processus peut s'appliquer notre système de types, on définit maintenant formellement plusieurs styles d'observations valides.

5.3.1 Démon

On commence par les notions classiques de may- et must-testing. Dans leur définition standard, elles utilisent un canal de référence ω et caractérisent le passage de test par la capacité des processus à agir sur ce canal. Ici on emploiera une formulation plus adaptée à la discipline stricte des interfaces et mathématiquement plus pure : on ajoute à l'algèbre des processus un terme particulier dont l'activation sert à caractériser l'acceptation.

Définition 5.34. L'algèbre des processus du π-calcul polarisé est étendue avec une constante notée $*$, appelée *démon*, sans aucune règle de transition. Un processus P est dit *acceptant* s'il existe un terme P' pour lequel $P \equiv * \mid P'$.

La définition de bisimulation est adaptée pour tenir compte de l'acceptation : une bisimulation est une relation binaire $\mathcal{S}$ qui vérifie les conditions de la définition 4.12 et telle que, pour tout $P \mathcal{S} Q$, P est acceptant si et seulement si Q est acceptant.

On constate sans difficulté que les propriétés de la bisimulation sont conservées dans ce calcul enrichi. Comme le terme $*$ n'a aucun nom libre et aucune transition, il respecte n'importe quelle interface, donc on ajoute la règle d'interfaçage triviale

$$\frac{}{* :: \mathcal{I}}$$

L'ensemble des acceptants est clos par réduction et c'est un idéal premier pour la composition : si P est acceptant alors $P \mid Q$ l'est aussi, et si $P \mid Q$ est acceptant alors soit P soit Q l'est ; on notera Acc cet ensemble. Le rôle du processus $*$ est de définir l'acceptation, c'est pourquoi on le qualifie de démon, car il est l'analogue du démon en ludique [39]. On définit ensuite $\perp$, selon le style d'observation voulue, à partir de la capacité des processus à se réduire vers cet ensemble.

Dans ces observations par test, le démon $*$ sera par construction élément de $\mathbf{0}$, donc de tout comportement. On peut donc admettre dans le système de types la nouvelle règle triviale

$$\frac{\Phi \Rightarrow \Gamma}{\Phi \Rightarrow * \vdash \Gamma}$$

Cette règle permet de construire des tests bien typés pour étudier les interactions possibles des processus. On trouvera des illustrations de cette technique dans le chapitre 7.

5.3.2 May testing

Définition 5.35. L'observation angélique, ou *may testing*, est l'ensemble $\perp_{\text{may}}$ des processus $P :: \emptyset$ tels qu'il existe une réduction $P \to^* Q \in Acc$.

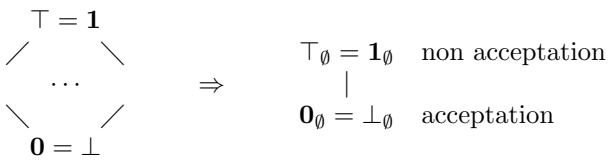
Proposition 5.36. *L'observation angélique est régulière.*

Démonstration. Comme la bisimulation, au sens de la définition 5.34, tient compte de la présence du démon, il est évident que $\perp$ est clos par bisimulation. Pour la condition d'anti-réduction, considérons deux termes P et Q d'interfaces opposées tels que P ait une unique transition $P \xrightarrow{\tau} P'$ avec $P' \perp Q$. Alors $P' \cdot Q$ peut se réduire vers Acc, donc $P \cdot Q$ le peut aussi, et donc $P \perp Q$.

Pour la dualité des modalités, étudions la structure d'un comportement $[\varepsilon]A$. Un terme P est orthogonal à $\bar{u}^{\varepsilon}(\vec{x}).Q$ lorsque $(\boldsymbol{\nu} u)(P \mid \bar{u}^{\varepsilon}(\vec{x}).Q)$ a une réduction qui atteint l'acceptation. Deux cas sont possibles : soit P a une réduction acceptante, soit il existe une suite de transitions $P \xrightarrow{\tau}{}^{*} \xrightarrow{u^{\varepsilon}(\vec{x})} P'$ avec $P' \perp Q$. Donc $P \in ([\varepsilon]A)[u]$ implique que P a une réduction acceptante ou qu'il a un réduit qui peut émettre $u^{\varepsilon}(\vec{x})$ avec un réduit élément de $A[u\vec{x}]$.

Soient $P \in u : [\varepsilon]A$ et $Q \in [\neg\varepsilon]A^{\perp}$. Si l'un de ces deux termes a une réduction acceptante, alors $P \cdot Q$ aussi. Sinon on sait donc que P peut émettre une action $u^{\varepsilon}(\vec{x})$ et arriver dans $A[u\vec{x}]$ et que Q peut émettre une action duale et arriver dans $A[u\vec{x}]^{\perp}$, et ces deux transitions peuvent se combiner en une réduction de $P \cdot Q$ vers un état acceptant. □

Il est intéressant de noter que, dans cette observation, si un processus $P :: I$ a une réduction qui atteint l'acceptation, alors pour tout $Q :: \neg I$ le processus $P \cdot Q$ peut accepter par la même réduction. En d'autres termes, on a $P \perp 1$ si et seulement si $P \perp \top$, donc pour toute interface on a $\perp = \mathbf{0}$, et par dualité $\mathbf{1} = \top$. En conséquence, à l'interface vide, on a exactement deux comportements :



Le comportement $\perp_{\emptyset}$ contient les processus capables d'atteindre un état acceptant, il est engendré par $*$; $\top_{\emptyset}$ contient tous les processus, et il est engendré par n'importe quel processus d'interface vide dont aucun réduit n'est acceptant. Cette égalité entre les constantes est caractéristique des modèles de la logique affine, et en effet pour le may testing la règle d'affaiblissement sur n'importe quelle formule est valide.

5.3.3 Fair testing

Définition 5.37. L'observation équitable, ou *fair testing*, est l'ensemble $\perp_{\text{fair}}$ des processus $P :: \emptyset$ tels que pour toute réduction $P \to^{*} Q$ il existe une réduction $Q \to^{*} R \in Acc$.

Proposition 5.38. *L'observation équitable est régulière.*

Démonstration. La démonstration est essentiellement la même que pour le may testing. Pour la première condition de régularité, si on a $P \perp Q$, alors pour toute réduction $\tau.P \mid Q \to^{*} R$, en consommant le préfixe τ s'il n'a pas été consommé par la réduction vers R, on déduit une réduction $R \to^{*} S$ telle que S soit réduit de $P \mid Q$, et S peut atteindre l'acceptation, donc $\tau.P \perp Q$.

La caractérisation des comportements $[\varepsilon]A$ change de signification par rapport au test angélique : on a maintenant $P \perp \bar{u}^{\varepsilon}(\vec{x}).Q$ si et seulement si (1) tout réduit de P peut soit se réduire en un état acceptant, soit se réduire en un processus qui peut émettre l'action $u^{\varepsilon}(\vec{x})$, et (2) tout réduit de P par une transition étiquetée $u^{\varepsilon}(\vec{x})$ est orthogonal à Q. Cette caractérisation donne bien la dualité entre modalités. □

La structure de cette observation est en fait très similaire au may testing, en particulier les treillis ont la même structure et les constantes sont identifiées de la même façon, ce qui donne également un modèle de la logique affine. Les deux observations sont tout de même différentes, par exemple les deux processus $u() \mid u().*$ et $\bar{u}()$ sont orthogonaux en observation angélique mais pas en observation équitable.

5.3.4 Terminaison

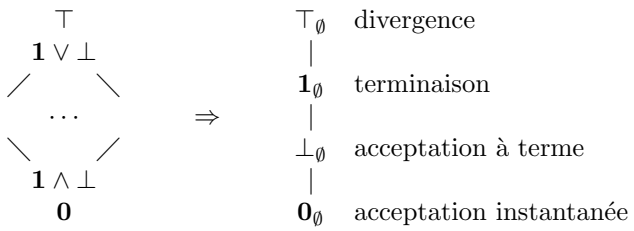
L'observation par terminaison est une technique habituelle dans l'étude des modèles de programmation séquentielle. Elle l'est un peu moins dans l'étude des processus concurrents, car il est souvent légitime de considérer qu'un processus qui diverge est équivalent à un processus inactif. Il s'agit en fait d'un choix de point de vue : considérer comme inactif un processus divergent est raisonnable quand on cherche à modéliser un système distribué dont les différents composants fonctionnent en parallèle ; en revanche si l'on cherche à modéliser l'exécution de plusieurs programmes dans un même système, on veut prendre en compte le fait que la divergence d'un processus bloque les autres.

Pour caractériser la terminaison dans notre formalisme, il paraît naturel de prendre pour $\bot$ l'ensemble des processus sans réduction infinie. Cette approche est valable mais pose un léger problème technique : comme aucun processus ne peut être orthogonal à un processus qui diverge, le comportement minimal $\mathbf{0}$ est vide. Si l'on applique la définition des modalités, on obtient alors $[\varepsilon]\top = \top$ alors que $[\varepsilon]\mathbf{0}$ n'est pas vide, ce qui fait échouer la condition de dualité des modalités, même si c'est pour une mauvaise raison. Il serait possible de résoudre le problème en adaptant la définition des modalités (c'était l'approche employée dans [6], qui est une version précédente de la théorie développée ici), mais l'emploi du démon permet de l'éviter.

Définition 5.39. L'observation stricte, ou *must testing*, est l'ensemble $\bot_{\text{must}}$ des processus $P :: \emptyset$ dont toute réduction maximale atteint l'acceptation.

Notation 5.40. Pour tout processus P, on note $\tau.P = (\boldsymbol{\nu} u)(u.P \mid \bar{u})$, de sorte que $\tau.P$ a une unique τ-transition qui mène à P. On note $\Omega = (\boldsymbol{\nu} u)(\bar{u} \mid !u.\bar{u})$, de sorte que Ω a une suite infinie de τ-transitions. Remarquons que, pour toute interface $\mathcal{I}$, on a $\Omega :: \mathcal{I}$ et que $P :: \mathcal{I}$ entraîne $\tau.P :: \mathcal{I}$.

Il s'agit donc d'une forme de test sensible à la possibilité de divergence, contrairement au test équitable. Un léger problème se pose : cette observation n'est pas régulière au sens de la définition 5.28, comme on va le voir. Les quatre constantes sont maintenant toutes distinctes :



L'acceptation instantanée contient tous les processus acceptants, qui contiennent donc $*$ en position active. L'acceptation à terme contient les processus dont

toute réduction maximale mène à l'acceptation, ce comportement est engendré par $\tau.*$, qui garantit l'acceptation face à tout processus non divergent. Si Ω est un processus qui diverge, on a donc $\Omega \perp *$ mais pas $\Omega \perp \tau.*$, car une réduction infinie de Ω empêche $\tau.*$ d'atteindre l'état $*$; c'est pour cette raison que $\perp_{\text{must}}$ n'est pas une observation régulière.

Le fait que tout processus non divergent, quelle que soit son interface, soit orthogonal à $\tau.*$ entraîne que $\{\tau.*\}^{\perp}$ est le comportement des processus sans réduction infinie non acceptante. De plus, si un processus a une réduction infinie non acceptante, il ne peut être orthogonal qu'à un processus acceptant, donc son orthogonal est $\mathbf{0}$ et il engendre $\top$. Ainsi $\{\tau.*\}^{\perp}$ est le plus grand comportement distinct de $\top$ et $\{\tau.*\}^{\perp\perp}$ est le plus petit comportement non nul.

Pour toute interface $\mathcal{I}$, le comportement $\perp_{\mathcal{I}}$ contient les processus dont toute réduction maximale est acceptante, sans information sur les transitions visibles. Ainsi pour toute action α de l'interface $\mathcal{I}$, $\alpha.\Omega \mid \tau.*$ est élément de $\perp_{\mathcal{I}}$. En conséquence, pour tout $P \in \mathbf{1}_{\mathcal{I}}$, s'il existe une transition $P \to^* \xrightarrow{\alpha} P'$ on a nécessairement $P' \in \mathbf{0}$. Les éléments de $\mathbf{1}_{\mathcal{I}} \cap \perp_{\mathcal{I}}$ sont donc les processus qui vérifient les deux conditions, c'est-à-dire les processus orthogonaux à tout processus qui n'a pas de réduction infinie non acceptante. C'est pourquoi $\mathbf{1} \cap \perp$ est le plus petit comportement non nul, et par dualité $\mathbf{1} \vee \perp$ est le plus grand comportement inférieur à $\top$.

Lemme 5.41. *Dans l'observation stricte, pour tout $P \perp Q$ tel que Q n'a pas de réduction infinie non acceptante, $\tau.P \perp Q$.*

Démonstration. Si Q n'a pas de réduction infinie non acceptante, toute réduction maximale de $\tau.P \mid Q$ qui n'est pas acceptante contient nécessairement la réduction $\tau.P \to P$, donc une telle réduction atteint un état qui est un réduit de $P \mid Q$. Par hypothèse on a $P \perp Q$ donc toute réduction maximale de $P \mid Q$ atteint l'acceptation, et il en est de même pour $\tau.P \mid Q$. □

Cette restriction a pour conséquence essentielle le fait que la règle d'introduction des modalités n'est valide que lorsque le comportement auquel la modalité est appliquée ne contient pas de divergence. On pose donc la notion suivante :

Définition 5.42. Un comportement $\mathcal{A}$ est non-divergent si pour tout $P \in \mathcal{A}$ toute réduction infinie de P atteint l'acceptation. Un comportement $\mathcal{A}$ est non-dégénéré si $\mathcal{A}$ et $\mathcal{A}^{\perp}$ sont non-divergents.

Lemme 5.43. $\mathbf{0}$ *et* $\top$ *sont les seuls comportements dégénérés. La non-dégénérescence est garantie par les modalités et préservée par tous les connecteurs.*

Démonstration. D'après la discussion qui suit la définition 5.39, un comportement A est non-dégénéré si et seulement si $\mathbf{1} \wedge \perp \subseteq A \subseteq \mathbf{1} \vee \perp$, donc les constantes $\mathbf{1}$ et $\perp$ sont non-dégénérées et la propriété est préservée par bornes supérieures et inférieures. Pour tout processus P et toute action α, le processus $\alpha.P$ n'est ni acceptant ni divergent, donc il est non-dégénéré, ce qui entraîne que pour tout A et tout ε le comportement $[\varepsilon]A$ est non-dégénéré. Enfin si P et Q sont deux processus d'interfaces disjointes qui ne sont ni acceptants ni divergents, leur composition parallèle $P \mid Q$ n'est ni acceptante ni divergente, donc $\otimes$ préserve la non-dégénérescence, et $⅋$ aussi par dualité. □

Ce lemme permet de garantir syntaxiquement que certains comportements sont non-dégénérés. En particulier, le fait que les modalités garantissent la non-dégénérescence entraîne que toute formule où les variables propositionnelles et les constantes $\mathbf{0}$ et $\top$ apparaissent derrière une modalité ont une valuation non-dégénérée. on obtient ainsi l'adéquation pour un sous-système :

Théorème 5.44. *Soit un processus typé $\Phi \Rightarrow P \vdash \Gamma$ dont le typage vérifie les deux contraintes suivantes :*

- $\mathbf{0}$, $\top$ *ne sont introduits (par la règle d'axiome) que sous une modalité,*
- *les quantificateurs $\forall X$ et $\exists X$ ne sont appliqués qu'à des formules où X apparaît sous une modalité.*

Alors pour toute valuation de type Φ on a $P \Vdash \Gamma$.

Démonstration. L'orthogonal de l'ensemble $\left\{ \bar{u}^{\varepsilon}(\vec{x}).P \;\middle|\; P \in A[u\vec{x}]^{\perp} \right\}$ est l'ensemble des processus vérifiant les deux conditions suivantes : (1) toute réduction maximale atteint Acc ou est finie et se termine dans un état où l'action $u^{\varepsilon}(\vec{x})$ est possible, (2) tout réduit par une transition $u^{\varepsilon}(\vec{x})$ est élément de $A[u\vec{x}]$. Il est facile de voir que l'orthogonal de cet ensemble se décrit de la même façon avec l'action $\bar{u}^{\varepsilon}(\vec{x})$ et le comportement $A[u\vec{x}]^{\perp}$, donc les modalités sont duales l'une de l'autre.

D'après le lemme 5.43, pour toute valuation de type Φ, l'interprétation de toute formule qui vérifie la restriction imposée par l'énoncé du théorème est non-dégénérée. Grâce au lemme 5.41 on en déduit la validité de la règle d'introduction des modalités. Pour toutes les autres règles, la démonstration du théorème 5.33 s'applique. □

Les deux conditions imposées par le théorème servent à éviter les cas dégénérés. En effet, si les constantes additives sont autorisées arbitrairement, la règle d'introduction des modalités est invalide. Considérons par exemple la dérivation suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Omega \vdash \top, \mathbf{0}}}{u.\Omega \vdash u : \downarrow\top, \mathbf{0}} \quad \dfrac{}{\Omega \vdash \top}}{u.\Omega \mid \Omega \vdash u : \downarrow\top} \quad \dfrac{\dfrac{}{* \vdash \mathbf{0}}}{\bar{u}.* \vdash u : \uparrow\mathbf{0}}}{(\boldsymbol{\nu} u)(u.\Omega \mid \Omega \mid \bar{u}.*) \vdash \bot}$$

Or le processus $(\boldsymbol{\nu} u)(u.\Omega \mid \Omega \mid \bar{u}.*)$ a une réduction infinie sans acceptation, ce qui contredit le fait qu'il soit de type $\bot$. La contrainte sur les quantifications peut s'illustrer par le cas suivant :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{X : \uparrow I \Rightarrow x{\rightarrow}y \vdash x : X^{\perp}, y : X}}{X : \uparrow I \Rightarrow u(x).x{\rightarrow}y \vdash u : \downarrow X^{\perp}, y : X}}{X : \uparrow I \Rightarrow u(x).x{\rightarrow}y \vdash uy : \downarrow X^{\perp} \parr X}}{u(x).x{\rightarrow}y \vdash uy : (\forall X : \uparrow I)(\downarrow X^{\perp} \parr X)}$$

En instanciant la variable X avec le comportement $\mathbf{0}_{\uparrow I}$, on en déduit que $u(x).x{\rightarrow}y$ doit être élément de $\downarrow\top \parr \mathbf{0}$, or ce comportement est égal à $\mathbf{0}$. En effet, pour A non dégénéré, on a toujours $A \otimes \top = \top$ car un processus divergent reste divergent en parallèle avec tout processus de A, non divergent et non acceptant ; par dualité on en déduit $A \parr \mathbf{0} = \mathbf{0}$. En conséquence on a $(\forall X : \uparrow I)(\downarrow X^{\perp} \parr X) = \mathbf{0}$, ce qui est contradictoire car tout élément de $\mathbf{0}$ est acceptant alors que $u(x).x{\rightarrow}y$ ne l'est pas.

Le problème vient donc du fait la quantification peut introduire des comportements dégénérés à la place des variables. Une autre approche, pour éviter d'avoir à imposer une restriction syntaxique sur la forme des quantifications, consiste à donner une sémantique différente aux quantificateurs : si l'on décide que $(\forall X : I)\, A$ est une quantification indexée par les comportements non-dégénérés, on a à nouveau l'adéquation pour toute valuation non dégénérée. Cette technique n'est pas totalement satisfaisante car elle elle rend moins propre l'interprétation des formules, mais elle permet d'obtenir des résultats plus généraux, comme le théorème de normalisation des termes typés :

Théorème 5.45. *Un processus typable sans les constantes additives n'a pas de réduction infinie.*

Démonstration. Considérons un processus typé $\Phi \Rightarrow P \vdash \Gamma$ dont le typage n'utilise pas la règle $\top$. On se place dans l'observation stricte, et on interprète les quantificateurs sous forme restreinte : $[\![(\forall X : I)A]\!]$ est l'intersection des $[\![A]\!]$ pour toute valuation non-dégénérée de X, de même pour la définition de $[\![(\exists X : I)A]\!]$. Le théorème 5.44 s'adapte pour montrer que, pour toute valuation non dégénérée de type Φ, on a $P \Vdash \Gamma$. Les seuls cas de la démonstration qui changent sont ceux des quantificateurs : pour la quantification universelle on utilise l'hypothèse d'induction, pour la quantification existentielle on utilise le fait que toute formule sans constante additive a une valeur non dégénérée si c'est le cas des variables, par le lemme 5.43. Pour une valuation arbitraire non dégénérée des variables propositionnelles on a donc $P \Vdash \Gamma$, or l'interprétation de Γ est non-dégénérée, donc P n'a pas de réduction infinie non acceptante. Comme P est un terme typé et qu'aucune règle de typage de la table 5.2 n'introduit le démon, P n'a donc aucune réduction infinie. □

Notons que cette preuve de terminaison des termes typés est une variante sur la technique des candidats de réductibilité [40]. En effet, en prenant pour $\bot$ l'ensemble des processus qui n'ont pas de réduction infinie, on retrouverait la définition classique des candidats. La technique de réalisabilité développée ici peut d'ailleurs se voir comme une généralisation de la technique des candidats pour d'autres propriétés que la terminaison.

5.4 Connecteurs construits

Les connecteurs primitifs que nous venons de définir sont les éléments fondamentaux qui servent à décrire des propriétés spatiales et temporelles. Dans cette section, on en déduit des connecteurs plus élaborés, d'une part pour représenter des mécanismes présents dans le calcul comme la réplication et le choix, et d'autre part pour définir l'ensemble des connecteurs de la logique linéaire.

5.4.1 Modalités linéaires

Les modalités introduites dans la section 5.2.3 sont de sorte $\varepsilon I, I \to \varepsilon I$, c'est-à-dire que les processus de type $[\varepsilon]A$ ont une action de plus sur leur unique canal que les processus de type A. Il est utile, en particulier pour modéliser la logique linéaire, de pouvoir décrire un comportement explicitement linéaire, c'est-à-dire la capacité à émettre exactement une action.

On pose donc les définitions suivantes :

Définition 5.46. Pour tout comportement $A : I$, on pose

$$\varepsilon A := [\varepsilon](\mathbf{1}_{\varepsilon I} \otimes A) \qquad \varepsilon^* A := [\varepsilon](\bot_{\varepsilon I} \parr A) \tag{5.9}$$

La combinaison d'un comportement $A : I$ avec $\mathbf{1}_J$ correspond à étendre la sorte des processus de A à $I \cup J$ sans modifier leur fonctionnement. En effet, par définition, on a $A \otimes \mathbf{1}_J = \{\, p \mid p \in A \,\}^{\bot\bot}$, puisque $\mathbf{1}_J$ est engendré par le processus neutre 1. Réaliser $A \otimes \mathbf{1}_J$ signifie donc réaliser A et être inactif sur la sorte J. À l'inverse, l'opération $A \parr \bot_J$ (c'est-à-dire $\mathbf{1}_{\neg J} \multimap A$) caractérise le comportement des processus qui réalisent A à condition de ne pas interagir sur la sorte J, en particulier envoyer une action sur J à un tel processus peut déclencher une divergence.

Cette dualité entre faire exactement une action et bien réagir face à une seule action explique que la modalité linéaire εA ne peut pas bénéficier de la même auto-dualité que la modalité générale $[\varepsilon]A$. Néanmoins, avec ces définitions, on a immédiatement les règles de typage suivantes :

$$\frac{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A}{\Phi \Rightarrow u^{\varepsilon}(\vec{x}).P \vdash \Gamma, u : \varepsilon A} \qquad \frac{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A}{\Phi \Rightarrow u^{\varepsilon}(\vec{x}).P \vdash \Gamma, u : \varepsilon^* A} \tag{5.10}$$

Pour étendre le langage des types avec ces nouvelles modalités, il faut donc poser $(\varepsilon A)^{\bot} = \bar{\varepsilon}^*(A^{\bot})$, cependant on a évidemment pour tout A l'inclusion $\varepsilon A \subseteq \varepsilon^* A$, et on vérifie aisément qu'on a toujours l'égalité $\varepsilon A \cdot \bar{\varepsilon} A^{\bot} = \bot_{\emptyset}$, donc la règle de coupure appliquée aux modalités linéaires est valide.

5.4.2 Choix et gardes

Les opérateurs $\wedge$ et $\vee$ sont associés à l'ordre observationnel, ordre ensembliste sur les comportements, et ils s'appliquent à interface fixée. Ils parlent de façon naturelle de sous-typage, et aussi de non-déterminisme. En effet, raisonner sur un processus de type $A \vee B$ impose de raisonner sur un processus qui pourra se révéler être élément de l'un ou l'autre type, éventuellement après des étapes de réduction. Par exemple, dans les observations considérées précédemment, si on a $P \in \mathcal{A}$ et $Q \in \mathcal{B}$, où $\mathcal{A}$ et $\mathcal{B}$ sont des comportements de même interface, on a $P \oplus Q \in \mathcal{A} \vee \mathcal{B}$, si $\oplus$ désigne un opérateur de choix non déterministe.

L'inconvénient de cette forme de choix est qu'elle n'est pas détectable par interaction : il n'est pas possible en général d'écrire un processus qui se comportera différemment selon que son interlocuteur est du comportement $\mathcal{A}$ ou du comportement $\mathcal{B}$. Un tel choix n'est possible que si les deux comportements sont distinguables par communication, le cas le plus simple étant celui où les deux comportements n'utilisent pas les mêmes canaux. Pour décrire cette forme de choix, on introduit donc les connecteurs additifs séparants suivants :

Définition 5.47. Pour tous comportements $A : I$ et $B : J$, on pose

$$A \oplus B := (A \otimes \mathbf{1}_J) \vee (\mathbf{1}_I \otimes B) \qquad A \,\&\, B := (A \parr \bot_J) \wedge (\bot_I \parr B) \tag{5.11}$$

On constate immédiatement que $\oplus$ et & ainsi définis sont associatifs, que $\mathbf{0}_{\emptyset}$ est neutre pour $\oplus$ et que $\top_{\emptyset}$ est neutre pour &. Les versions localisées sont en plus commutatives.

Remarque 5.48. Il n'y pas de distributivité stricte entre les multiplicatifs et les additifs séparants : si A, B et C sont des comportements anonymes de sortes non vides, $A \otimes (B \oplus C)$ et $(A \otimes B) \oplus (A \otimes C)$ ne sont pas de même sorte. Il n'est même pas possible de démontrer d'équivalence entre ces deux comportements dans le système de types, même si l'équivalence est réalisable.

Par construction, tous processus $P \in \mathcal{A}$ et $Q \in \mathcal{B}$ sont éléments de $\mathcal{A} \oplus \mathcal{B}$, et ce comportement est effectivement engendré par l'ensemble des éléments de $\mathcal{A}$ et $\mathcal{B}$ (après changement d'interface). Les règles suivantes sont en effet démontrables dans notre système de types :

$$\frac{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A}{\Phi \Rightarrow P \vdash \Gamma, \vec{x}\vec{y} : A \oplus B} \qquad \frac{\Phi \Rightarrow P \vdash \Gamma, \vec{y} : B}{\Phi \Rightarrow P \vdash \Gamma, \vec{x}\vec{y} : A \oplus B} \tag{5.12}$$

Le fait de manipuler des processus non déterministes a l'effet intéressant que, dans les observations étudiées plus haut, $\mathcal{A} \oplus \mathcal{B}$ est aussi engendré par l'ensemble des $P \oplus Q$, si l'opérateur $\oplus$ sur les processus agit comme un choix interne, par exemple en posant $P \oplus Q = (\boldsymbol{\nu} u)(u.P \mid u.Q \mid \bar{u})$. En effet, tout réduit d'un tel processus réalise soit $\mathcal{A}$ soit $\mathcal{B}$, donc $\mathcal{A} \oplus \mathcal{B}$.

Cherchons maintenant à décrire le comportement d'un processus $P \in \mathcal{A} \,\&\, \mathcal{B}$. L'assertion $P \in \mathcal{A} \parr \bot_{\mathcal{J}}$ peut s'interpréter de deux façons : en lisant $P \in \mathcal{A}^{\bot} \multimap \bot_{\mathcal{J}}$ on déduit que P composé avec tout processus $Q \in \mathcal{A}^{\bot}$ doit être élément de $\bot_{\mathcal{J}}$, en lisant $P \in \mathbf{1}_{\neg \mathcal{J}} \multimap \mathcal{A}$ on déduit que P restreint à l'interface $\mathcal{I}$ doit se comporter comme un processus de $\mathcal{A}$. En d'autres termes, P doit se comporter comme un processus de $\mathcal{A}$ ou de $\mathcal{B}$ lorsqu'il est connecté à un processus qui n'interagit que sur l'une des deux interfaces. Ce comportement est naturellement celui du choix externe dans les processus.

Rappelons que les processus séquentiels (notés $S, T \ldots$) sont les combinaisons de préfixes $\alpha.P$ par somme $S + T$ et réplication $!S$, c'est-à-dire qu'il s'agit de choix gardés. Pour un processus séquentiel S, le branchement se fait sur une certaine famille de canaux qui correspond aux différentes branches du choix. Si S est d'interface $\mathcal{I}$, on appelle interface active de S la partie de $\mathcal{I}$ restreinte à ces canaux. Par exemple, si $S = u(x).P + v(y).Q$ est d'interface $u : \downarrow I, v : \downarrow J, w : \uparrow K$, l'interface active de S est $u : \downarrow I, v : \downarrow J$.

Proposition 5.49. *Soient $\mathcal{A} : \mathcal{I}$, $\mathcal{B} : \mathcal{J}$ et $\mathcal{C} : \mathcal{K}$ trois comportements d'interfaces disjointes. Soient $S \in \mathcal{A} \multimap \mathcal{B}$ et $T \in \mathcal{A} \multimap \mathcal{C}$ deux processus séquentiels d'interface active $\mathcal{J}$ et $\mathcal{K}$ respectivement. Alors on a $S + T \in \mathcal{A} \multimap \mathcal{B} \,\&\, \mathcal{C}$.*

Démonstration. Il s'agit de démontrer que pour tout $P \in \mathcal{A}$ on a $P \cdot (S + T) \in \mathcal{B} \,\&\, \mathcal{C}$, c'est-à-dire $P \cdot (S + T) \bot \mathcal{B}^{\bot} \otimes \mathbf{1}_{\mathcal{K}} \parr \mathbf{1}_{\mathcal{J}} \otimes \mathcal{C}^{\bot}$, ou encore que pour tous $Q \in \mathcal{B}^{\bot}$ et $R \in \mathcal{C}^{\bot}$ on a $S + T \bot P \mid Q$ et $S + T \bot P \mid R$, par définition de $\otimes$ et $\parr$. Par hypothèse on a $S \bot P \mid Q$, or S et $(\boldsymbol{\nu}\mathcal{K})(S + T)$ sont bisimilaires, donc $(\boldsymbol{\nu}\mathcal{K})(S + T) \bot P \mid Q$ puisque $\bot$ est clos par bisimulation et que la bisimulation est une congruence. On a $(\boldsymbol{\nu}\mathcal{K})(S + T) \equiv (S + T) \cdot_{\mathcal{K}} 1$, donc on en déduit $S + T \bot P \mid Q \mid 1$ or le membre de droite engendre $\mathcal{A} \otimes \mathcal{B}^{\bot} \otimes \mathbf{1}_{\mathcal{K}}$, donc on a $S + T \in \mathcal{A} \multimap (\mathcal{B} \parr \bot_{\mathcal{K}})$. De la même façon, on prouve $S + T \in \mathcal{A} \multimap (\bot_{\mathcal{J}} \parr \mathcal{C})$, et comme $\multimap$ commute avec l'intersection à droite, on en conclut $S + T \in \mathcal{A} \multimap (\mathcal{B} \,\&\, \mathcal{C})$. □

Un problème se pose pour le typage des choix gardés : la proposition 5.49 permet de construire des représentants d'un type $A \multimap B \,\&\, C$ à partir de

Action et focalisation :

$$\frac{\Phi \Rightarrow P \vdash \Gamma, u\vec{x} : A}{\Phi \Rightarrow u^{\varepsilon}(\vec{x}).P \vdash \Gamma \mid u : [\varepsilon]A} \qquad \frac{\Phi \Rightarrow S \vdash \Gamma \mid \vec{x} : A}{\Phi \Rightarrow S \vdash \Gamma, \vec{x} : A}$$

Choix externe :

$$\frac{\Phi \Rightarrow S \vdash \Gamma \mid \vec{x} : A \qquad \Phi \Rightarrow T \vdash \Gamma \mid \vec{y} : B}{\Phi \Rightarrow S + T \vdash \Gamma \mid \vec{x}\vec{y} : A \,\&\, B}$$

TAB. 5.3 – Règles de typage des branchements.

représentants *séquentiels* de $A \multimap B$ et $A \multimap C$. On résout ce problème en définissant un système de types spécifique pour les processus séquentiels, en isolant une formule active qui décrit la partie active de l'interface du processus.

Définition 5.50. Un type de processus séquentiel est de la forme $\Gamma \mid \vec{x} : A$, de telle sorte que $\Gamma, \vec{x} : A$ soit un type de processus. Les mêmes notions de bonne formation s'appliquent. Un processus séquentiel S est de type $\Gamma \mid \Delta$ dans l'environnement Φ si $\Phi \Rightarrow S \vdash \Gamma \mid \Delta$ est dérivable par les règles des tables 5.2 et 5.3.

Étant donnée une observation et une valuation, l'interprétation de $\Gamma \mid \vec{x} : A$ est l'ensemble des processus séquentiels d'interface active $\vec{x}$ qui réalisent $\Gamma, \vec{x} : A$.

Proposition 5.51. *Si $\Phi \Rightarrow S \vdash \Gamma \mid \Delta$ est dérivable, alors pour toute observation valide et toute valuation de type Φ on a $S \Vdash \Gamma \mid \Delta$.*

Démonstration. Il suffit de vérifier la validité des règles de la table 5.3. Les règles d'action et de focalisation sont immédiates d'après la définition de $S \Vdash \Gamma \mid \Delta$, et le cas de l'introduction de & est le sens de la proposition 5.49. □

5.4.3 Points fixes

Le fait d'avoir toutes les bornes supérieures et inférieures permet de garantir l'existence de plus petits et plus grands points fixes pour tous les opérateurs croissants. On définit donc des opérateurs de point fixe sur les types, grâce auxquels on définira dans la section suivante des modalités exponentielles pour décrire la réplication gardée.

Proposition 5.52. *Toute fonction $F : \mathbf{C}_I \to \mathbf{C}_I$ croissante a un plus petit et un plus grand points fixes. On les note respectivement $\mu X.F(X)$ et $\nu X.F(X)$.*

Démonstration. Notons $F^{\vee}$ l'ensemble des points préfixes de F, c'est-à-dire $F^{\vee} = \{ X \mid F(X) \leqslant X \}$. On a nécessairement $F(\top) \leqslant \top$ donc cet ensemble n'est pas vide. De plus $F^{\vee}$ est stable par bornes inférieures arbitraires, en effet pour toute famille (X_i) dans $F^{\vee}$ on a $F(\bigwedge_i X_i) \leqslant F(X_i)$ pour chaque i par croissance de F, or pour chaque i on a par hypothèse $F(X_i) \leqslant X_i$ donc $F(\bigwedge_i X_i) \leqslant \bigwedge_i X_i$ et $\bigwedge_i X_i \in F^{\vee}$. Notons alors μF la borne inférieure de $F^{\vee}$, on a donc $\mu F \in F^{\vee}$ et $F(\mu F) \leqslant \mu F$, de plus μF minore tout point préfixe de F, et donc tout point fixe. Par croissance de F on en déduit $F^2(\mu F) \leqslant F(\mu F)$, donc $F(\mu F) \in F^{\vee}$, or μF est par construction le minimum de $F^{\vee}$ donc $\mu F \leqslant F(\mu F)$,

Sortage des types de points fixes :

$$\frac{\Phi, X : I \Rightarrow A : I}{\Phi \Rightarrow (\mu X : I)A : I} \qquad \frac{\Phi, X : I \Rightarrow A : I}{\Phi \Rightarrow (\nu X : I)A : I}$$

Axiome d'ordre supérieur :

$$\frac{}{\Phi, P \vdash X \Rightarrow P \vdash X}$$

Plus petit point fixe :

$$\frac{\Phi \Rightarrow P \vdash \Gamma, \vec{u} : A[\mathbf{0}_I/X]}{\Phi \Rightarrow P \vdash \Gamma, \vec{u} : (\mu X : I)A} \qquad \frac{\Phi \Rightarrow P \vdash \Gamma, \vec{u} : A[(\mu X : I)A/X]}{\Phi \Rightarrow P \vdash \Gamma, \vec{u} : (\mu X : I)A}$$

Plus grand point fixe :

$$\frac{\Phi, X : I, P \vdash \Gamma, \vec{u} : X \Rightarrow Q \vdash \Gamma, \vec{u} : A \qquad \Phi, X : I \Rightarrow A : I \qquad P \equiv Q}{\Phi \Rightarrow P \vdash \Gamma, \vec{u} : (\nu X : I)A}$$

Tab. 5.4 – Typage des points fixes.

d'où $\mu F = F(\mu F)$ et μF est donc le plus point fixe de F. De la même façon, on montre que F a un plus grand point postfixe νF, qui est nécessairement son plus grand point fixe. □

En particulier, on a $\mu F \geqslant \bigvee_{n\in\mathbb{N}} F^n(\mathbf{0})$ avec égalité si F commute aux bornes supérieures, et $\nu F \leqslant \bigwedge_{n\in\mathbb{N}} F^n(\top)$ avec égalité si F commute aux bornes inférieures. Pour montrer l'appartenance à un plus grand point fixe, l'énoncé suivant donne une approche plus systématique :

Proposition 5.53. *Soient I une sorte, F une fonction croissante sur $\mathbf{C}_I$ et Γ un comportement anonyme. Soit p un processus tel que, pour tout comportement $X : I$, $p \in \Gamma \parr X$ implique $p \in \Gamma \parr F(X)$, alors $p \in \Gamma \parr \nu F$.*

Démonstration. Soit C_p l'ensemble des comportements X tels que $p \in \Gamma \parr X$. Par distribution de $\parr$ sur la borne inférieure, si X_0 est la borne inférieure de C_p alors $p \in \Gamma \parr X_0$, donc X_0 est le minimum de C_p. Par hypothèse on a donc $p \in \Gamma \parr F(X_0)$, d'où $X_0 \subseteq F(X_0)$. X_0 est donc un point postfixe de F, il est donc inclus dans νF, et $p \in \Gamma \parr \nu F$. □

Pour faire de cette proposition une véritable règle de typage, il est nécessaire d'étendre le langage des séquents pour pouvoir formuler l'hypothèse « $p \in \Gamma \parr X$ implique $p \in \Gamma \parr F(X)$ ». Dans cette forme étendue, un jugement de typage est de la forme $\Phi \Rightarrow P \vdash \Gamma$ où l'environnement Φ est composée d'hypothèses $X : I$ pour l'interfaçage des variables propositionnelles et d'hypothèses $P \vdash \Gamma$ pour autoriser des hypothèse sur le typage des sous-termes. On ajoute donc une règle d'axiome d'ordre supérieur pour utiliser ces hypothèses, ce qui mène au système de règles de la table 5.4.

Notons que si le calcul est équipé d'un opérateur de définition récursive, par exemple de la forme $\mathrm{rec}\, Z.S[Z]$ où $S[\,]$ est un contexte de sorte $\Pi \to \Sigma$ (pour

représenter une récursion gardée), on peut en déduire la règle suivante :

$$\frac{\Phi, (X : I), (Z \vdash \Gamma, \vec{u} : X) \Rightarrow S[Z] \vdash \Gamma, \vec{u} : A}{\Phi \Rightarrow \mathrm{rec}\, Z.S[Z] \vdash \Gamma, \vec{u} : (\nu X : I)A}$$

Quelle que soit la façon de formuler ces règles de typage pour les points fixes, le système obtenu est très lourd à manipuler. D'autre part, le π-calcul polarisé tel qu'il est présenté dans le chapitre 4 ne contient pas d'opérateur de récursion mais un opérateur de réplication gardée. Il s'agit d'une forme particulière de récursion pour laquelle on se contentera d'un point fixe particulier, comme nous allons le voir maintenant.

5.4.4 Répétitions

Avec les différents connecteurs définis jusqu'ici, il n'est pas possible de décrire des comportements infinis, or c'est dans sa capacité à parler d'infini que réside l'essentiel de l'expressivité d'une logique. En logique linéaire, ce sont les opérateurs exponentiels qui donnent ce pouvoir. Dans la technologie des réseaux de preuves, ils se traduisent par des opérations de duplication de sous-réseaux, qui correspondent exactement à la pratique de la réplication gardée dans les calculs de processus. Notre interprétation des connecteurs exponentiels va donc naturellement consister à décrire la dynamique de réplication des processus.

On ne manipulera ici que de la réplication gardée, car cette contrainte est nécessaire à la bonne définition des connecteurs logiques associés. En effet, désigner explicitement une garde dans un processus comme $!u(x).P$ est le seul moyen de séparer les canaux qui déclenchent une réplication (ici simplement u) de ceux dont les actions sont répliquées (les autres noms libres de $!u(x).P$). Cette distinction est cruciale pour définir deux opérateurs duaux, correspondant au client (modalité $?A$, qui correspond à plusieurs exemplaires de A) et au serveur (modalité $!A$, qui fournit autant de copie de A que nécessaire). En conséquence, on définit la modalité $?A$ par deux propriétés caractéristiques :

- une action sur u libère un comportement A, et éventuellement d'autres requêtes sur u, donc $[\uparrow](?A \parr A) \subseteq ?A$;
- un client contient un nombre de requêtes arbitraire, autrement dit plusieurs clients de même type, arbitrairement corrélés, peuvent être fusionnés en un client de ce type, donc $(?A \parr \cdots \parr ?A) \cdot \delta_n \subseteq ?A$ où $\delta_n[u_1 \dots u_n v] = u_1{\to}v \mid \cdots \mid u_n{\to}v$.

De ces remarques, on déduit la définition de $?A$ comme un point fixe. Pour la deuxième condition, il suffit d'imposer les cas $n = 0$ et $n = 2$ pour avoir la contraction de tout nombre fini de clients.

Définition 5.54. Pour toute sorte I, on pose

$$W_I := \bot_\emptyset \otimes \mathbf{1}_I \quad C_I(X) := (X \parr X) \cdot \delta \quad \text{avec } \delta[uvw] = u{\to}w \mid v{\to}w \tag{5.13}$$

Soit $A : I$ un comportement, soit f_A la fonction sur $\mathbf{C}_{\uparrow I}$ définie par

$$f_A(X) := W_{\uparrow I} \vee C_{\uparrow I}(X) \vee [\uparrow](X \parr A) \tag{5.14}$$

On pose alors $?A := \mu f_A$ et $!A := (?A^\perp)^\perp$.

L'existence de ce plus petit point fixe est garantie par la proposition 5.52, puisque la fonction f_A est évidemment croissante. De plus, la définition de cette fonction garantit par construction les propriétés attendues de la modalité :

Proposition 5.55. *On a les règles suivantes pour toute observation valide :*

$$\frac{P \vdash \Gamma}{P \vdash \Gamma, u : ?A} \qquad \frac{P \vdash \Gamma, u : ?A, v : ?A}{P[w/u, v] \vdash \Gamma, w : ?A} \qquad \frac{P \vdash \Gamma, u : \uparrow A}{P \vdash \Gamma, u : ?A}$$

Démonstration. Par construction on a $W_{\uparrow I} \subseteq ?A$, c'est-à-dire que pour tous $P \in \bot_\emptyset$ on a $P \in ?A$, ce qui entraîne $1 \in \bot_\emptyset \multimap ?A$. Comme $\bot_\emptyset$ est le neutre de $⅋$, on en déduit que pour $P \Vdash \Gamma$ on a $P \Vdash \Gamma, \bot_\emptyset$ et donc $P \equiv P \cdot 1 \Vdash \Gamma, ?A$.

De la même façon, pour tout $P \Vdash \Gamma, u : ?A, v : ?A$ on a $P \cdot \delta[uvw] \Vdash \Gamma, w : (?A ⅋ ?A) \cdot \delta$, or d'après les règles de renommage des canaux par les routeurs (proposition 4.27 page 65) on a $P \cdot \delta[uvw] = (\boldsymbol{\nu} uv)(P \mid u{\to}w \mid v{\to}w) \cong P[w/u, v]$ donc $P[w/u, v] \Vdash \Gamma, w : C(?A)$ et $C(?A) \subseteq ?A$ par construction donc $P \Vdash \Gamma, w : ?A$.

Enfin, par la définition 5.46, on a $\uparrow A = [\uparrow](\mathbf{1}_{\uparrow I} \otimes A)$, or $\mathbf{1}_{\uparrow I} \otimes A$ est engendré par les éléments de A, et pour tout $Q \in A$ on a clairement $Q \in \bot_\emptyset ⅋ A$ et donc $Q \equiv Q \cdot 1 \in (\bot_\emptyset \otimes \mathbf{1}_{\uparrow I}) ⅋ A$ d'après l'adéquation de la règle du tenseur, donc $Q \in W_{\uparrow I} ⅋ A$, et par construction $W_I \subseteq ?A$ donc $Q \in ?A ⅋ A$. On en déduit $\mathbf{1}_{\uparrow I} \otimes A \subseteq ?A ⅋ A$ et donc $\uparrow A \subseteq [\uparrow](?A ⅋ A) \subseteq ?A$. Par conséquent, pour tout P, $P \Vdash \Gamma, u : \uparrow A$ entraîne $P \Vdash \Gamma, u : ?A$. □

La modalité duale $!A$ décrit donc des processus non divergents (orthogonaux à $W_{\uparrow I}$), susceptibles d'être dupliqués (par l'opérateur dual de $C_{\uparrow I}$), et fournissant des processus indépendants qui réalisent A. Ce comportement est naturellement implémenté par la réplication gardée, au moyen du lemme de duplication suivant :

Lemme 5.56. *Soit P un processus et soient u, v et w trois noms frais. On a alors $!u(\vec{x}).P \cdot (v{\to}u \mid w{\to}u) \cong !v(\vec{x}).P \mid !w(\vec{x}).P$.*

Démonstration. Soit $\delta = v{\to}u \mid w{\to}u$, notons $P_1 = (\boldsymbol{\nu} u)(!u(\vec{x}).P \mid \delta)$ et $P_2 = !v(\vec{x}).P \mid !w(\vec{x}).P$. On pose la relation $\mathcal{S} = \{(P_1 \mid Q), (P_2 \mid Q)\}$ avec Q arbitraire. Il est évident qu'on a toujours $\mathcal{F}(P_1) = \mathcal{F}(P_2) = \emptyset$. Considérons alors une transition d'un $P_1 \mid Q$: soit la transition est issue de Q, auquel cas elle est simulée par la même transition dans $P_2 \mid Q$, soit elle est issue de P_1 et on a une dérivation de la forme

$$\dfrac{\dfrac{\dfrac{!u(\vec{x}).P \xrightarrow{u(\vec{x})} !u(\vec{x}).P}{!u(\vec{x}).P \mid \delta \xrightarrow{u(\vec{x})} !u(\vec{x}).P \mid P \mid \delta}\,(v,u) \in \mathcal{F}(!u(\vec{x}).P \mid \delta)}{!u(\vec{x}).P \mid \delta \xrightarrow{v(\vec{x})} !u(\vec{x}).P \mid P \mid \delta}}{P_1 \mid Q \xrightarrow{v(\vec{x})} (\boldsymbol{\nu} u)(!u(\vec{x}).P \mid P \mid \delta) \mid Q}$$

ou de même avec une transition $w(\vec{x})$. Cette transition est simulée par

$$\frac{!v(\vec{x}).P \xrightarrow{v(\vec{x})} !v(\vec{x}).P \mid P}{P_2 \mid Q \xrightarrow{v(\vec{x})} !v(\vec{x}).P \mid P \mid !w(\vec{x}).P \mid Q}$$

Notons $P_1' \mid Q$ le réduit de $P_1 \mid Q$ et $P_2' \mid Q$ le réduit de $P_2 \mid Q$. Par hypothèse les noms u, v et w n'apparaissent pas dans P donc on a $P_1' \equiv P_1 \mid P$, d'où $P_1' \mid Q \equiv P_1 \mid (P \mid Q)\ \mathcal{S}\ P_2 \mid (P \mid Q) \equiv P_2' \mid Q$. La relation $\mathcal{S}$ est donc une bisimulation modulo $\equiv$, donc une bisimulation. En l'appliquant à $P_1 \mid 1$ et $P_2 \mid 1$ on en déduit $P_1 \cong P_2$. □

De ce lemme on peut déduire l'adéquation de la règle de promotion, avec la contrainte habituelle sur le contexte.

Proposition 5.57. *Pour tous comportements $A_1, \dots, A_n$ et B, pour toute observation valide, la règle suivante est valide :*

$$\frac{P \vdash u_1 : ?A_1, \dots, u_n : ?A_n, \vec{x} : B}{!v(\vec{x}).P \vdash u_1 : ?A_1, \dots, u_n : ?A_n, v : !B}$$

Démonstration. Notons $?\Gamma = u_1 : ?A_1, \dots, u_n : ?A_n$. Soit un processus $P \Vdash ?\Gamma, \vec{x} : B$. Par construction, pour $B : I$, $!B$ est donc le plus grand point fixe de l'opérateur

$$g_B(X) := W^{\perp}_{\uparrow\neg I} \cap C_{\uparrow\neg I}(X^{\perp})^{\perp} \cap [\downarrow](X \otimes B)$$

donc grâce à la proposition 5.53 il suffit pour montrer le résultat voulu de montrer que pour tout comportement X tel que $!u(\vec{x}).P \Vdash ?\Gamma, u : X$ on a $!u(\vec{x}).P \Vdash ?\Gamma, u : g_B(X)$. Soit donc un tel X.

- on a $!u(\vec{x}).P \cdot 1_u \equiv (\boldsymbol{\nu} u)!u(\vec{x}).P \cong 1$ or $1 \Vdash \mathbf{1}_{\emptyset}$ donc par affaiblissement (proposition précédente) on a $1 \Vdash ?\Gamma, \mathbf{1}_{\emptyset}$, et par conséquent $!u(\vec{x}).P \Vdash ?\Gamma, \mathbf{1}_u \multimap \mathbf{1}_{\emptyset}$, c'est-à-dire $!u(\vec{x}).P \Vdash ?\Gamma, W^{\perp}_{\uparrow\neg I}$;
- soit $\delta = v{\to}u \mid w{\to}u$, on a donc $!u(\vec{x}).P \cdot \delta \cong !v(\vec{x}).P \mid !w(\vec{x}).P$ d'après le lemme 5.56, or par hypothèse $!v(\vec{x}).P \Vdash ?\Gamma, v : X$ et $!w(\vec{x}).P \Vdash ?\Gamma, w : X$, et le contexte $?\Gamma$ supporte la contraction d'après la proposition précédente, donc $!u(\vec{x}).P \cdot \delta \Vdash ?\Gamma, vw : X \otimes X$ et donc $!u(\vec{x}).P \Vdash ?\Gamma, u : C_{\uparrow\neg I}(X^{\perp})^{\perp}$;
- comme $!u(\vec{x}).P \Vdash ?\Gamma, u : X$ et $P \Vdash ?\Gamma, \vec{x} : B$ on a $!u(\vec{x}).P \mid P \Vdash ?\Gamma, u\vec{x} : X \otimes B$ puisque $?\Gamma$ supporte la contraction, et par conséquent $u(\vec{x}).(!u(\vec{x}).P \mid P) \Vdash u : [\downarrow](X \otimes B)$. Par la règle de congruence structurelle $!u(\vec{x}).P \equiv u(\vec{x}).(!u(\vec{x}).P \mid P)$ on en déduit donc $!u(\vec{x}).P \Vdash u : [\downarrow](X \otimes B)$.

En combinant les trois points précédents on déduit donc $!u(\vec{x}).P \Vdash ?\Gamma, g_B(X)$, ce qui permet de conclure. □

Ce qui fait que la réplication gardée implémente correctement la modalité $!A$ est donc la propriété de duplication énoncée par le lemme 5.56, qui s'énonce $(\boldsymbol{\nu} u)(!u(\vec{x}).P \mid \delta[uvw]) \cong !v(\vec{x}).P \mid !w(\vec{x}).P$. Il s'agit d'une bisimulation, mais en la lisant de gauche à droite on retrouve la dynamique de duplication des réseaux de preuve, entre une boîte exponentielle $!u(\vec{x}).P$ et un duplicateur δ (les noms ne servent qu'à localiser l'opération sur des canaux particuliers). Il est intéressant de considérer la dynamique de l'opérateur dual $\gamma[uvw] = u{\to}v \mid u{\to}w$, qui est une co-duplication, ou co-contraction. On vérifie sans problème la bisimilarité $\gamma[uvw] \cdot \delta[u'vw] \cong u{\to}u'$. L'autre composition, par le canal u, donne $\delta[uvw] \cdot \gamma[uv'w'] \cong (v{\to}v' \mid v{\to}w' \mid w{\to}v' \mid w{\to}w')$, qui peut se reformuler comme la règle $\gamma\delta$ des combinateurs d'interaction de Lafont [54] ou la règle contraction/co-contraction des réseaux d'interaction différentiels [27] :

$$(\boldsymbol{\nu} u)(\delta[uvw] \mid \gamma[uv'w']) \cong (\boldsymbol{\nu}\vec{u})(\gamma[vu_1u_2] \mid \gamma[wu_3u_4] \mid \delta[v'u_1u_3] \mid \delta[w'u_2u_4])$$

Cette équivalence, en apparence plus compliquée que nécessaire, donne l'argument principal pour la validité de la règle de co-contraction :

Proposition 5.58. *Les comportements $!A$ sont stables par composition. En d'autres termes, $\gamma[uvw] \Vdash v : !A \otimes w : !A \multimap u : !A$ avec $\gamma[uvw] = u{\to}v \mid u{\to}w$.*

Démonstration. On reprend les notations de la proposition 5.57. Posons $X = (!A \otimes !A) \cdot \gamma$, il s'agit donc de montrer $X \subseteq g_A(X)$.

On a évidemment $\gamma[uvw] \cdot 1_u \cong 1_v \mid 1_w$, donc pour tous $P \Vdash v : !A$ et $Q \Vdash w : !A$, on a $((P \mid Q) \cdot \gamma[uvw]) \cdot 1_u \cong (P \cdot 1_v) \mid (Q \cdot 1_w)$, or par hypothèse $P \cdot 1_v \Vdash \mathbf{1}_\emptyset$ et $Q \cdot 1_w \Vdash \mathbf{1}_\emptyset$, donc $(P \cdot 1_v) \mid (Q \cdot 1_w) \Vdash \mathbf{1}_\emptyset$ et $(P \mid Q) \cdot \gamma[uvw]$ réalise $\mathbf{1}_u \multimap \mathbf{1}_\emptyset = W^\perp_{\uparrow\neg I}$.

Pour montrer $X \subseteq C_{\uparrow\neg I}(X^\perp)^\perp$ il suffit de montrer $X \perp C(X^\perp) = (X^\perp \parr X^\perp) \cdot \delta$ ou encore $X \cdot \delta \subseteq X \otimes X$. Pour tous $P \Vdash v : !A$ et $Q \Vdash w : !A$, d'après la remarque précédente on a

$$\begin{aligned}((P \mid Q) \cdot \gamma[uvw]) \cdot \delta[uv'w'] &\cong (\boldsymbol{\nu}\vec{u})(P \mid Q \mid \gamma[v'u_1u_2] \mid \gamma[w'u_3u_4] \mid \delta[vu_1u_3] \mid \delta[wu_2u_4]) \\ &\equiv (P \cdot \delta[vu_1u_3] \mid Q \cdot \delta[wu_2u_4]) \cdot (\gamma[v'u_1u_2] \mid \gamma[w'u_3u_4])\end{aligned}$$

Par définition de $!A$ on a $P \cdot \delta[vu_1u_3] \Vdash u_1 : !A \otimes u_3 : !A$ et de même pour Q, donc $P \cdot \delta[vu_1u_3] \mid Q \cdot \delta[wu_2u_4]$ réalise $\vec{u} : !A \otimes !A \otimes !A \otimes !A$. D'autre part, par définition de X, on a $\gamma[v'u_1u_2] \Vdash u_1 : !A \otimes u_2 : !A \multimap v' : X$ et de même pour $\gamma[w'u_3u_4]$, donc $\gamma[v'u_1u_2] \mid \gamma[w'u_3u_4]$ réalise $\vec{u} : !A \otimes !A \otimes !A \otimes !A \multimap v' : X \otimes w' : X$. En composant les deux processus on obtient par bisimilarité $((P \mid Q) \cdot \gamma[uvw]) \cdot \delta[uv'w'] \Vdash v' : X \otimes w' : X$, ce qui prouve $X \cdot \delta \subseteq X \otimes X$ et donc $X \subseteq C_{\uparrow\neg I}(X^\perp)^\perp$.

Reste à montrer $X \subseteq [\downarrow](X \otimes A)$, c'est-à-dire que $X[u]$ est orthogonal à $\left\{ \bar{u}(\vec{x}).R \;\middle|\; R \Vdash u : X^\perp \parr \vec{x} : A^\perp \right\}$. Pour tout $R \Vdash u : X^\perp \parr \vec{x} : A^\perp$, par définition de X on a $R \cdot \gamma \Vdash vw\vec{x} : ?A^\perp \parr ?A^\perp \parr A^\perp$, donc $\bar{u}(\vec{x}).(R \cdot \gamma) \Vdash vwu : ?A^\perp \parr ?A^\perp \parr ?A^\perp$. On montre facilement $\gamma \cdot \bar{u}(\vec{x}).(R \cdot \gamma) \cong \bar{v}(\vec{x}).(R \cdot \gamma) + \bar{w}(\vec{x}).(R \cdot \gamma)$, et par contraction chaque branche réalise $vw : ?A^\perp \parr ?A^\perp$, donc la somme réalise aussi ce comportement. De plus on a $\gamma \cdot (\bar{u}(\vec{x}).R) \cong \gamma \cdot \bar{u}(\vec{x}).(R \cdot \gamma)$, donc $\gamma \cdot (\bar{u}(\vec{x}).R)$ le réalise aussi, autrement dit $\bar{u}(\vec{x}).R \perp u : (!A \otimes !A) \cdot \gamma = X$, ce qui permet de conclure. □

5.4.5 Synthèse

En utilisant les deux constructions primitives que sont la composition et les modalités, on a défini l'ensemble des connecteurs de la logique linéaire, et ces connecteurs sont représentés syntaxiquement dans le calcul. Le langage des types est donc enrichi des constructions suivantes :

$$A, B := \cdots \mid \varepsilon A \mid \varepsilon^* A \mid A \oplus B \mid A \,\&\, B \mid !A \mid ?A$$

La dualité entre connecteurs s'étend en posant

$$(\varepsilon A)^\perp = \bar{\varepsilon}^*(A^\perp) \qquad (A \oplus B)^\perp = A^\perp \,\&\, B^\perp \qquad (!A)^\perp = ?A^\perp \tag{5.15}$$

Les règles de typage de ces connecteurs sont rassemblées dans la table 5.5 page suivante. On ne mentionne pas dans ce résumé le typage des points fixes, ceux-ci étant plutôt considérés comme des outils pour construire des connecteurs.

Comme tous ces connecteurs sont définis à partir des connecteurs primitifs du système présenté en table 5.2 page 74, le théorème d'adéquation s'étend aux les nouvelles règles grâce aux résultats de cette section.

Modalités linéaires :

$$\frac{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A}{\Phi \Rightarrow u^{\varepsilon}(\vec{x}).P \vdash \Gamma \mid u : \varepsilon A} \qquad \frac{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A}{\Phi \Rightarrow u^{\varepsilon}(\vec{x}).P \vdash \Gamma \mid u : \varepsilon^* A}$$

Modalités exponentielles :

$$\frac{\Phi \Rightarrow P \vdash \Gamma \mid u : \uparrow A}{\Phi \Rightarrow P \vdash \Gamma \mid u : ?A} \qquad \frac{\Phi \Rightarrow P \vdash ?\Gamma, \vec{x} : A}{\Phi \Rightarrow !u(\vec{x}).P \vdash ?\Gamma \mid u : !A}$$

$$\frac{\Phi \Rightarrow P \vdash \Gamma}{\Phi \Rightarrow P \vdash \Gamma, u : ?A} \qquad \frac{\Phi \Rightarrow P \vdash \Gamma, u : ?A, v : ?A}{\Phi \Rightarrow P[w/u, v] \vdash \Gamma, w : ?A}$$

Additifs :

$$\frac{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A}{\Phi \Rightarrow P \vdash \Gamma, \vec{x}\vec{y} : A \oplus B} \qquad \frac{\Phi \Rightarrow P \vdash \Gamma, \vec{y} : B}{\Phi \Rightarrow P \vdash \Gamma, \vec{x}\vec{y} : A \oplus B}$$

$$\frac{\Phi \Rightarrow S \vdash \Gamma \mid \vec{x} : A \qquad \Phi \Rightarrow T \vdash \Gamma \mid \vec{y} : B}{\Phi \Rightarrow S + T \vdash \Gamma \mid \vec{x}\vec{y} : A \,\&\, B}$$

Focalisation :

$$\frac{\Phi \Rightarrow S \vdash \Gamma \mid \vec{x} : A}{\Phi \Rightarrow S \vdash \Gamma, \vec{x} : A}$$

Règles concurrentes :

$$\frac{\Phi \Rightarrow P \vdash \Gamma \qquad \Phi \Rightarrow Q \vdash \Delta}{\Phi \Rightarrow P \mid Q \vdash \Gamma, \Delta} \qquad \frac{\Phi \Rightarrow P \vdash \Gamma, u : !A \qquad \Phi \Rightarrow Q \vdash \Delta, u : !A}{\Phi \Rightarrow P \mid Q \vdash \Gamma, \Delta, u : !A}$$

Tab. 5.5 – Typage des connecteurs construits.

Théorème 5.59. *Si un jugement* $\Phi \Rightarrow P \vdash \Gamma$ *est dérivable par les règles des tables 5.2 et 5.5, alors pour toute observation valide et toute valuation de type* Φ *on a* $P \Vdash \Gamma$*.*

Toutes les conséquences de l'adéquation sont donc préservées, y compris le théorème de terminaison des termes typés, puisque les points fixes préservent la non-dégénérescence des comportements. C'est dans ce cas que le théorème est intéressant, puisque les processus construits uniquement par les règles de la table 5.2 ont évidemment tous un comportement fini.

La réalisabilité concurrente construite dans ce chapitre permet donc d'interpréter toute la logique linéaire du second ordre. Les développements techniques que nous avons détaillés ici utilisent le π-calcul polarisé comme modèle calculatoire sous-jacent. Néanmoins, la structure précise du calcul n'est pas cruciale pour construire une réalisabilité et obtenir l'adéquation du système de typage.

- La composition de comportements utilise une forme de composition de processus $(\boldsymbol{\nu}\vec{x})(P|Q)$, constituée de composition parallèle et de restriction ; la seule contrainte est que cette composition (à congruence structurelle près) soit associative et commutative pour les termes bien interfacés.
- La constante $\mathbf{1}$ nécessite l'existence d'un processus 1, inactif et neutre pour la composition.
- Les modalités utilisent des préfixes de communication de la forme $u^{\varepsilon}(\vec{x}).P$. La communication de noms est le point important ; le fait que les noms soient privés semble être principalement une simplification du formalisme, et le préfixage n'est qu'une capacité particulière du calcul employé.

Ces trois éléments suffisent à définir toute la logique. La règle d'axiome nécessite en plus l'existence de processus ayant l'effet des routeurs. Comme on l'a vu dans la section 4.4.1, de tels processus peuvent exister dans d'autres calculs concurrents, par exemple en π-calcul asynchrone. Il serait donc possible de reprendre la démarche exposée ici avec une approche abstraite, où le calcul sous-jacent ne serait pas totalement spécifié. Les éléments de syntaxe supplémentaires employés dans les règles de typage, comme la réplication ou le choix gardé, se verront alors comme des constructions symboliques, pas nécessairement présentes dans la syntaxe précise du langage.

Chapitre 6

Traduction de systèmes classiques

Le système de typage du π-calcul polarisé développé dans le chapitre 5 contient la logique linéaire. En conséquence, toute démonstration de logique linéaire peut être interprétée comme une dérivation de typage, et donc comme la définition d'un processus. Tout système logique qui peut être traduit en logique linéaire peut ainsi être systématiquement traduit en π-calcul polarisé.

Une restriction s'applique cependant à cette observation : dans la logique de comportements développée pour décrire les processus, toute formule a une sorte fixée, donc la traduction d'un système logique dans la logique des comportements n'est applicable qu'aux formules bien sortées. Les seuls connecteurs de la logique linéaire qui sont affectés par cette restriction sont les quantificateurs, car la quantification sur les comportements se fait à sorte fixée. Dans la traduction de systèmes de types, ceci se manifeste par une restriction sur le degré de polymorphisme autorisé.

Partant de ces remarques, on définit ici des traductions systématiques du λ-calcul et de ses extensions avec contrôle dans le π-calcul polarisé, au moyen de traductions des logiques intuitionniste et classique dans la logique linéaire. On s'intéresse dans un premier temps à la décomposition usuelle de la logique intuitionniste $A \to B = !A \multimap B$, qui mène à une traduction minimaliste du λ-calcul simplement typé, mais sans le polymorphisme du système F. On passe ensuite aux traductions modales de la logique classique, lesquelles mènent à des traductions pour différentes stratégies d'évaluation dans lesquelles le polymorphisme du système F est préservé.

6.1 λ-calcul simplement typé

Le système F a un plongement standard dans la la logique linéaire, et donc dans les types de processus, ce qui induit une traduction du λ-calcul typé dans le π-calcul polarisé. Un point important est à noter : dans le typage des processus, chaque formule, et en particulier chaque variable propositionnelle, a une interface fixée, et les quantifications sont de la forme $(\forall X : I)\, F(X)$ où X est donc d'interface I, et cette contrainte pose donc une restriction sur le polymorphisme.

Dans cette section, on développe une traduction pour le λ-calcul simplement typé, afin d'acquérir une intuition sur le sens calculatoire des traductions. Dans

la section suivante, on définira d'autres traductions qui permettent le polymorphisme général du système F et de ses extensions à la logique classique.

Définition 6.1. On considère un λ-calcul simplement typé avec une famille de types de base notés X, $Y\ldots$ La traduction de ce système est définie par une interprétation de chaque type de base X par une variable propositionnelle (notée aussi X par abus de langage) dont l'interface est de la forme $\downarrow I$. La traduction des types simples se déduit par

$$(X)^* := X \qquad\qquad (A \to B)^* := !A^* \multimap B^* \tag{6.1}$$

L'interface d'un type fonctionnel $X \to Y$ contient donc une capacité négative (donc d'émission) pour l'argument de type X et une capacité positive (donc de réception) pour la sortie de type Y. Plus généralement un type $A_1, \ldots, A_n \to X$ a une interface d'arité $n+1$, avec n capacités négatives et un positive. Par conséquent un terme typé

$$x_1 : A_1, \ldots, x_n : A_n \vdash t : B_1, \ldots, B_k \to X$$

se traduit en un processus connaissant les noms $x_1, \ldots, x_n$ et localisé en un vecteur $\vec{\alpha}$ de longueur $k+1$. Un tel processus a alors un type de la forme suivante :

$$[\![t]\!]_{\vec{\alpha}} \vdash x_1 : ?A_1^{*\perp}, \ldots, x_n : ?A_n^{*\perp}, \vec{\alpha} : (!B_1^* \multimap \cdots \multimap !B_k^* \multimap X)$$

La règle d'axiome appliquée aux types de base se traduit donc de la façon suivante :

$$\frac{}{x : X \vdash x : X} \quad \to \quad \frac{\dfrac{}{\alpha{\to}y \vdash y : X^\perp, \alpha : X}}{\bar{x}(y).(\alpha{\to}y) \vdash x : ?X^\perp, \alpha : X} \tag{6.2}$$

L'écriture du terme obtenu peut se simplifier en $\bar{x}\langle\alpha\rangle$, selon la notation 4.28 page 66. On peut déduire une traduction l'axiome général par η-expansion, au moyen de la traduction des abstractions et applications. On peut également s'autoriser la règle d'axiome généralisée (voir remarque 5.5 page 73) et écrire la règle suivante, avec $B = !A_1^*, \ldots, !A_n^* \multimap X$:

$$\frac{\dfrac{}{\alpha'_1{\to}\alpha_1 \mid \cdots \mid \alpha'_n{\to}\alpha_n \mid \beta{\to}\beta' \vdash \alpha'_1 \ldots \alpha'_n\beta' : B^\perp, \alpha_1 \ldots \alpha_n\beta : B}}{\bar{x}\langle\vec{\alpha}\beta\rangle \vdash x : ?B^\perp, \vec{\alpha}\beta : B} \tag{6.3}$$

L'abstraction consiste simplement à introduire un connecteur $\parr$, ce qui ne change pas le terme :

$$\frac{P \vdash ?\Gamma, x : ?A^{*\perp}, \vec{\alpha} : B^*}{P \vdash ?\Gamma, x\vec{\alpha} : !A^* \multimap B^*} \tag{6.4}$$

Quant à l'application, elle ajoute naturellement une promotion sur l'argument :

$$\frac{P \vdash ?\Gamma, x\vec{\beta} : !A^* \multimap B^* \qquad \dfrac{\dfrac{Q \vdash ?\Delta, \vec{\gamma} : A^*}{!x(\vec{\gamma}).Q \vdash ?\Delta, x : !A^*} \qquad \dfrac{}{\vec{\alpha}{\to}\vec{\beta} \vdash \vec{\beta} : B^{*\perp}, \vec{\alpha} : B^*}}{!x(\vec{\gamma}).Q \mid \vec{\alpha}{\to}\vec{\beta} \vdash ?\Delta, x\vec{\beta} : !A^* \otimes B^{*\perp}, \vec{\alpha} : B^*}}{(\boldsymbol{\nu} x\vec{\beta})(P \mid !x(\vec{\gamma}).Q \mid \vec{\alpha}{\to}\vec{\beta}) \vdash ?\Gamma \cup ?\Delta, \vec{\alpha} : B^*} \tag{6.5}$$

La fin de la dérivation est une coupure sur $x\vec{\beta}$ suivie d'une suite de contractions qui unifient les variables de ?Γ et ?Δ ; pour être tout à fait précis il aurait fallu séparer ces variables et appliquer des substitutions dans les termes impliqués. Le terme $\vec{\alpha}{\to}\vec{\beta}$ est la η-expansion de l'axiome $B^* \multimap B^*$, comme illustré plus haut. Ce routeur apparaît sous une restriction et toutes les autres occurrences des noms de $\vec{\alpha}$ sont négatives, donc par la proposition 4.27 on peut simplifier le terme obtenu en substituant $\vec{\alpha}$ pour $\vec{\beta}$, ce qui donne en définitive la traduction suivante du λ-calcul simplement typé :

$$\begin{aligned} [\![x]\!]_\alpha &:= \bar{x}\langle\alpha\rangle \\ [\![\lambda x.t]\!]_{x\vec{\alpha}} &:= [\![t]\!]_{\vec{\alpha}} \\ [\![t\,u]\!]_{\vec{\alpha}} &:= (\boldsymbol{\nu} x)([\![t]\!]_{x\vec{\alpha}} \mid !x(\vec{\beta}).[\![u]\!]_{\vec{\beta}}) \end{aligned} \tag{6.6}$$

Il apparaît donc que la traduction obtenue est invariante par permutation des radicaux, au sens où on a toujours $[\![(\lambda xy.t)uv]\!]_\alpha \equiv [\![(\lambda yx.t)vu]\!]_\alpha$. Il s'agit là en fait de la σ-équivalence [71], dont on rappelle la définition :

Définition 6.2. La σ-équivalence est l'équivalence $=_\sigma$ engendrée par les règles suivantes par passage aux contextes arbitraires :

$$(\lambda x.t)uv =_\sigma (\lambda x.tv)u \qquad\qquad (\lambda xy.t)u =_\sigma \lambda y.(\lambda x.t)u \tag{6.7}$$

avec $x \notin \mathrm{fv}(v)$ et $y \notin \mathrm{fv}(u)$.

Proposition 6.3. *Pour tous termes typés $t =_\sigma u$ on a $[\![t]\!]_{\vec{\alpha}} \equiv [\![u]\!]_{\vec{\alpha}}$.*

Démonstration. Remarquons que le typage est préservé par σ-réduction. Il suffit donc de prouver que pour chacune des deux règles, les deux membres sont structurellement équivalents. Dans la suite, on notera $[x = t]$ le processus $!x(\vec{\alpha}).[\![t]\!]_{\vec{\alpha}}$ où $\vec{\alpha}$ est un vecteur de noms frais dont la longueur est supposée connue en fonction du contexte. Dans le cas de la première règle, on a

$$\begin{aligned} [\![(\lambda x.t)uv]\!]_{\vec{\alpha}} &= (\boldsymbol{\nu} y)([\![(\lambda x.t)u]\!]_{y\vec{\alpha}} \mid [y = v]) \\ &= (\boldsymbol{\nu} y)((\boldsymbol{\nu} x)([\![t]\!]_{y\vec{\alpha}} \mid [x = u]) \mid [y = v]) \\ &\equiv (\boldsymbol{\nu} x)((\boldsymbol{\nu} y)([\![t]\!]_{y\vec{\alpha}} \mid [y = v]) \mid [x = u]) \\ &= (\boldsymbol{\nu} x)([\![tv]\!]_{\vec{\alpha}} \mid [x = u]) \\ &= [\![(\lambda x.tv)u]\!]_{\vec{\alpha}} \end{aligned}$$

en utilisant l'hypothèse $x \notin \mathrm{fv}(v)$. Dans le cas de la deuxième règle, comme on a $y \notin \mathrm{fv}(u)$, on a

$$[\![(\lambda xy.t)u]\!]_{y\vec{\alpha}} = (\boldsymbol{\nu} x)([\![t]\!]_{\vec{\alpha}} \mid [x = u]) = [\![(\lambda x.t)u]\!]_{\vec{\alpha}} = [\![\lambda y.(\lambda x.t)u]\!]_{y\vec{\alpha}}$$

La σ-réduction des termes laisse donc leur traduction invariante à congruence structurelle près. □

Considérée à bisimilarité près, la traduction quotiente même plus que par σ-équivalence. En effet, considérons un terme de la forme $(\lambda x.t)u$ où x n'apparaît pas dans t. On a alors

$$[\![(\lambda x.t)u]\!]_{\vec{\alpha}} = (\boldsymbol{\nu} x)([\![t]\!]_{\vec{\alpha}} \mid [x = u]) \equiv [\![t]\!]_{\vec{\alpha}} \mid (\boldsymbol{\nu} x)[x = u] \cong [\![t]\!]_{\vec{\alpha}} \mid 1 \equiv [\![t]\!]_{\vec{\alpha}} \tag{6.8}$$

c'est-à-dire que l'effacement d'un argument non utilisé n'est pas une étape d'interaction dans le processus obtenu mais une simplification par bisimilarité. En étudiant la forme des traductions de termes modulo σ-équivalence, on constate en fait qu'une traduction a toujours au plus une τ-réduction, et qu'elle correspond précisément à une étape de réduction linéaire de tête [19, 24]. Rappelons brièvement la définition de cette forme de réduction : par σ-équivalence, tout λ-terme t peut se normaliser sous la forme

$$t =_\sigma \lambda x_1 \dots x_k(\lambda y_1 \dots y_n.(x)v_1 \dots v_p)u_1 \dots u_n \tag{6.9}$$

Ici chaque variable y_i est associée à l'argument u_i. La réduction linéaire de tête consiste alors, si x est l'une des variables y_i, à remplacer l'occurrence de tête de x, et uniquement celle-ci, par le terme u_i :

$$\begin{aligned}\lambda x_1 \dots x_k(\lambda y_1 \dots y_n.(y_i)v_1 \dots v_p)u_1 \dots u_n \\ \to \lambda x_1 \dots x_k(\lambda y_1 \dots y_n.(u_i)v_1 \dots v_p)u_1 \dots u_n\end{aligned} \tag{6.10}$$

Il n'y a pas de réduction dans le cas où x n'est pas l'un des y_i. La définition précise de cette stratégie de réduction peut s'étendre aux λ-termes généraux, sans recours à la σ-équivalence. On se passera ici d'une telle définition générale, précisément parce que la traduction en π-calcul polarisé quotiente par σ-équivalence.

Proposition 6.4. *Pour tout λ-terme simplement typé $\Gamma \vdash t : A$, le processus $[\![t]\!]_{\vec{\alpha}}$ est bisimilaire à t pour la réduction linéaire de tête.*

Démonstration. Comme il est démontré dans [71], tout terme t peur être normalisé par σ-équivalence en $t =_\sigma \lambda x_1 \dots x_k(\lambda y_1 \dots y_n.(x)v_1 \dots v_p)u_1 \dots u_n$. Par hypothèse on considère un terme typé, et selon la forme de ce terme le typage est de la forme $\Gamma \vdash t : A_1, \dots, A_k \to C$ où les A_i sont les types des x_i, et la variable x est d'un type $B_1, \dots, B_p \to C$, où les B_j sont les types des termes v_j. La traduction de t est donc paramétrée par un vecteur $x_1 \dots x_k\vec{\alpha}$ où $\vec{\alpha}$ correspond au type C, et elle s'écrit

$$[\![t]\!]_{\vec{x}\vec{\alpha}} \equiv (\boldsymbol{\nu}\vec{y}\vec{z})\big(\bar{x}\langle\vec{z}\vec{\alpha}\rangle \mid [z_1 = v_1] \mid \cdots \mid [z_p = v_p] \mid [y_1 = u_1] \mid \cdots \mid [y_n = u_n]\big)$$

pour une suite de variables fraîches $\vec{z}$, en particulier x ne peut pas être l'un des z_i. La seule émission active est donc celle sur x, et deux cas sont possibles : si x n'est pas l'un des y_j, le processus et bloqué et il a une unique transition étiquetée $\bar{x}$. S'il existe un j tel que $x = y_j$, il y a une unique transition

$$\begin{aligned}[\![t]\!]_{\vec{x}\vec{\alpha}} &= [\![\lambda\vec{x}.(\lambda\vec{y}.(y_j)\vec{v})\vec{u}]\!]_{\vec{x}\vec{\alpha}} \\ &\equiv (\boldsymbol{\nu}\vec{y})\big(\bar{y}_j\langle\vec{z}\vec{\alpha}\rangle \mid [y_1 = u_1] \mid \cdots \mid [y_n = u_n]\big) \\ &\to (\boldsymbol{\nu}\vec{y}\vec{z})\big([\![u_j]\!]_{\vec{z}\vec{\alpha}} \mid [z_1 = v_1] \mid \cdots \mid [z_p = v_p] \mid [y_1 = u_1] \mid \cdots \mid [y_n = u_n]\big) \\ &\equiv (\boldsymbol{\nu}\vec{y})\big([\![(u_j)\vec{v}]\!]_{\vec{\alpha}} \mid [y_1 = u_1] \mid \cdots \mid [y_n = u_n]\big) \\ &\equiv [\![(\lambda\vec{y}.(u_j)\vec{v})\vec{u}]\!]_{\vec{\alpha}} = [\![\lambda\vec{x}.(\lambda\vec{y}.(u_j)\vec{v})\vec{u}]\!]_{\vec{x}\vec{\alpha}}\end{aligned}$$

Par définition de la réduction linéaire de tête, le terme t est réductible si et seulement si x est l'un des y_j, et la réduction dans ce cas correspond bien à cette règle. □

6.2 Logique classique : LKT et LKQ

La traduction précédente du λ-calcul simplement typé est minimale au sens où elle introduit le moins possible d'actions, puisque la traduction des types n'ajoute que les modalités exponentielles strictement nécessaires. L'inconvénient qu'il y a à économiser les modalités est que la traduction d'un terme a une interface qui dépend de son arité en tant que fonction (et récursivement de l'arité de ses arguments), ce qui empêche d'étendre la traduction à un système de types polymorphe raisonnable. Pour retrouver le polymorphisme, on va maintenant introduire une modalité (éventuellement linéaire) sur chaque implication, ce qui permet d'obtenir des traductions du λ-calcul très systématiques et régulières dans lesquelles tous les types et processus ont la même interface.

6.2.1 Traductions

L'étude générale des traductions modales des logiques intuitionniste et classique dans la logique linéaire a été menée en détail par Danos, Joinet et Schellinx afin d'étudier l'élimination des coupures en logique classique [20, 21], et on adapte ici leur technique pour en extraire le contenu calculatoire sous forme de processus. Les systèmes logiques considérés sont tous construits sur le langage de formules du système F, donc avec uniquement l'implication et la quantification universelle du second ordre.

Définition 6.5. Une modalité généralisée est un mot sur $\{\downarrow, \uparrow, !, ?\}$. Pour une telle modalité μ, pour tout type A on note μA le type obtenu en appliquant les modalités de μ à A, et si I est la sorte de A on note μI la sorte de μA.

La définition précédente emploie les lettres μ et ν pour désigner les modalités généralisées, on prendra bien garde à ne pas confondre ces notations avec celles des points fixes. Comme ce chapitre n'emploie pas d'opérateurs de point fixe, il n'y a pas d'ambiguïté.

Définition 6.6. Une traduction modale $(\cdot)^*$ est définie par deux modalités généralisées μ et ν non vides. Soit T l'unique sorte telle que $T = \mu T \to \nu T$. Les types se traduisent par

$$(X)^* := X \qquad (A \to B)^* := \mu A^* \multimap \nu B^* \qquad (\forall X\, A)^* := (\forall X : T)\, A^* \tag{6.11}$$

Un séquent $x_1 : A_1, \ldots, x_n : A_n \vdash \alpha_1 : B_1, \ldots, \alpha_p : B_p$ se traduit en

$$\vdash x_1 : (\mu A_1^*)^\perp, \ldots, x_n : (\mu A_n^*)^\perp, \alpha_1 : \nu B_1^*, \ldots, \alpha_p : \nu B_p^*$$

À une traduction $(\cdot)^*$ est associée une traduction des règles d'inférences de la logique considérée (intuitionniste ou classique) en schémas de dérivation dans la logique des types.

Dans l'interprétation des types, les modalités linéaires $\uparrow$ et $\downarrow$ correspondent à des émissions et réceptions linéaires, ? correspond à une ensemble fini d'émissions et ! correspond à un serveur pouvant produire autant de réceptions que nécessaire. Les modalités construites induisent donc des protocoles d'interaction élémentaires.

Définition 6.7. Étant donnée une modalité μ, un nom u et une famille de noms distincts $\vec{x}$, le procotole $u^\mu(\vec{x})$ est défini par

$$u^{\uparrow}(\vec{x}).P := \bar{u}(\vec{x}).P \qquad u^{?}(\vec{x}).P := \bar{u}(\vec{x}).P \tag{6.12}$$

$$u^{\downarrow}(\vec{x}).P := u(\vec{x}).P \qquad u^{!}(\vec{x}).P := !u(\vec{x}).P \tag{6.13}$$

et récursivement $u^{\mu\dagger}(\vec{x}).P = u^\mu(v).v^\dagger(\vec{x}).P$ pour chaque $\dagger \in \{\downarrow, \uparrow, !, ?\}$, où v est un nom frais.

Notons que ces protocoles élémentaires sont bien typés, puisqu'on a toujours la règle d'inférence suivante :

$$\frac{P \vdash ?\Gamma, \vec{x} : A}{u^\mu(\vec{x}).P \vdash ?\Gamma, u : \mu A} \tag{6.14}$$

qui généralise les règles d'introduction des modalités élémentaires. Cette règle doit supposer que le contexte est de la forme $?\Gamma$ dès que la modalité μ contient une occurrence de !.

Les systèmes classiques considérés manipulent la logique propositionnelle minimale construite sur l'implication et la quantification du second ordre. Pour la partie calculatoire des preuves, on emploie donc le formalisme du λ-calcul dans le cadre intuitionniste et celui du $\lambda\mu$-calcul dans le cadre classique. La quantification est inchangée par traduction, et comme on se place dans un système à la Curry elle reste une annotation logique sans influence sur la structure calculatoire des preuves.

Les seules règles à considérer sont donc l'axiome et l'introduction et l'élimination de la flèche, qui correspondent au fragment multiplicatif de la logique linéaire, ou calculatoirement aux routeurs et aux compositions parallèles. La traduction consiste à insérer des modalités sur les formules pour autoriser les règles de contraction et affaiblissement sur les hypothèses et conclusions, c'est-à-dire à insérer des protocoles d'interaction entre routeurs. Pour pouvoir traduire les règles d'inférence, il faut donc poser quelques conditions supplémentaires :

- pour l'application, il faut pouvoir couper $\nu(\mu A \multimap \nu B)$ contre νA ;
- l'application entraîne des contractions, il faut donc que la modalité $\bar{\mu}$ (duale de μ) autorise la contraction, donc que μ commence par ! ;
- dans le cas classique, pour la même raison, il faut en plus que ν commence par ?.

Comme expliqué dans [20], on se ramène à deux cas symétriques : soit ν est suffixe de μ, soit μ est suffixe de ν. Ces deux cas correspondent respectivement aux systèmes LKT et LKQ, qui ont été identifiés comme le pendant logique des stratégies d'évaluation du λ-calcul en appel par nom et appel par valeur respectivement [17].

On va donc construire des traductions du $\lambda\mu$-calcul dans le π-calcul polarisé, en étudiant les traductions de ces systèmes logique dans la logique linéaire. Dans chaque cas, un terme t est traduit en un processus $[\![t]\!]_\alpha$ paramétré par un nom de canal α qui correspond à sa sortie. Ce canal est la conclusion active du terme, et le changement de conclusion active ne change pas le processus :

$$\frac{\dfrac{\Gamma \vdash t : B \mid \alpha : A, \Delta}{\Gamma \vdash [\beta]t \mid \alpha : A, \beta : B, \Delta}}{\Gamma \vdash \mu\alpha[\beta]t : A \mid \beta : B, \Delta} \quad \rightarrow \quad \frac{\dfrac{[\![t]\!]_\beta \vdash \Gamma, \alpha : \nu A, \beta : \nu B, \Delta}{[\![[\beta]t]\!] \vdash \Gamma, \alpha : \nu A, \beta : \nu B, \Delta}}{[\![\mu\alpha[\beta]t]\!]_\alpha \vdash \Gamma, \alpha : \nu A, \beta : \nu B, \Delta} \tag{6.15}$$

on obtient donc la règle triviale suivante, grâce à l'α-conversion :

$$[\![\mu\alpha[\beta]t]\!]_\alpha = [\![t]\!]_\beta \tag{6.16}$$

Appel par valeur Le cas de LKQ correspond à $\mu = !$ et $\nu = ?!$. Sans obscurcir le propos, on peut considérer que μ est une modalité qui commence par ! et que ν s'écrit $\xi\mu$ pour un ξ qui commence par ?. Dans ce cas l'axiome donne

$$\dfrac{\dfrac{}{y{\to}x \vdash x : \bar{\mu}A^\perp, y : \mu A}}{\alpha^\xi(y).(y{\to}x) \vdash x : \bar{\mu}A^\perp, \alpha : \nu A} \tag{6.17}$$

Comme de coutume, on s'autorise à abréger $\alpha^\xi(y).(y{\to}x)$ en une action non liante $\alpha^\xi\langle x\rangle$. L'abstraction consiste simplement à ajouter une modalité :

$$\dfrac{\dfrac{P \vdash ?\Gamma, x : \bar{\mu}A^\perp, \alpha : \nu B}{P \vdash ?\Gamma, x\alpha : \mu A \multimap \nu B}}{\beta^\nu(x\alpha).P \vdash ?\Gamma, \beta : \nu(\mu A \multimap \nu B)} \tag{6.18}$$

Pour l'application, on déconstruit le type de P au moyen de la dérivation suivante, avec l'abréviation naturelle $\beta^{\bar\nu}\langle y\alpha\rangle = \beta^{\bar\nu}(xz).(x{\to}y \mid z{\to}\alpha)$:

$$\dfrac{\dfrac{P \vdash ?\Gamma, \beta : \nu(\mu A \multimap \nu B) \qquad \dfrac{\dfrac{x{\to}y \vdash x : \mu A, y : \bar{\mu}A^\perp \qquad z{\to}\alpha \vdash z : \bar{\nu}B^\perp, \alpha : \nu B}{x{\to}y \mid z{\to}\alpha \vdash xz : \mu A \otimes \bar{\nu}B^\perp, y : \bar{\mu}A^\perp, \alpha : \nu B}}{\beta^{\bar\nu}\langle y\alpha\rangle \vdash \beta : \bar{\nu}(\mu A \otimes \bar{\nu}B^\perp), y : \bar{\mu}A^\perp, \alpha : \nu B}}{(\boldsymbol{\nu}\beta)(P \mid \beta^{\bar\nu}\langle y\alpha\rangle) \vdash ?\Gamma, y : \bar{\mu}A^\perp, \alpha : \nu B}}{\gamma^{\bar\xi}(y).(\boldsymbol{\nu}\beta)(P \mid \beta^{\bar\nu}\langle y\alpha\rangle) \vdash ?\Gamma, \gamma : \bar{\nu}A^\perp, \alpha : \nu B} \tag{6.19}$$

L'introduction du $\bar\nu$ est valide car $\bar\mu$ et ν commencent par ?, de même pour l'introduction du $\bar\xi$. En coupant le résultat contre $Q \vdash ?\Delta, \gamma : \nu A$ et en appliquant les contractions nécessaires au contexte, on obtient finalement la traduction de l'application :

$$[\![t\,u]\!]_\alpha := (\boldsymbol{\nu}\gamma)\big(\gamma^{\bar\xi}(y).(\boldsymbol{\nu}\beta)([\![t]\!]_\beta \mid \beta^{\bar\nu}\langle y\alpha\rangle) \;\big|\; [\![u]\!]_\gamma\big) \tag{6.20}$$

La traduction de LKQ avec $\mu = !$ et $\xi = ?$ induit donc une traduction du $\lambda\mu$-calcul pour une stratégie d'évaluation en appel par nom qui s'écrit :

$$\begin{aligned} [\![x]\!]^q_\alpha &:= \bar\alpha\langle x\rangle \\ [\![\lambda x.t]\!]^q_\alpha &:= \bar\alpha(y).!y(x\beta).[\![t]\!]^q_\beta \\ [\![t\,u]\!]^q_\alpha &:= (\boldsymbol{\nu}\gamma)\big(!\gamma(x).(\boldsymbol{\nu}\beta)([\![t]\!]^q_\beta \mid !\beta(w).\bar w\langle x\alpha\rangle) \;\big|\; [\![u]\!]^q_\gamma\big) \end{aligned} \tag{6.21}$$

Sans surprise, il s'agit exactement de la traduction formulée par Honda, Yoshida et Berger, dans leur étude du contrôle en π-calcul [44], pour le $\lambda\mu$-calcul en appel par nom de Ong et Stewart [66]. Leur traduction est aussi issue du typage de $\lambda\mu$, mais par des arguments différents liés à la nature du calcul fonctionnel avec contrôle.

Notons que le choix de $\xi = ?$ (ou $\nu = ?!$), qui nécessite l'introduction de réplications dans l'application, vient du fait que LKQ est un système classique, où la contraction doit être autorisée sur les conclusions. En se restreignant à la

logique intuitionniste ξ peut être linéaire, le cas extrême étant la modalité vide, c'est-à-dire qu'on peut prendre $\mu = \nu = !$, ce qui donne après simplification :

$$\begin{aligned} [\![x]\!]_\alpha &:= \alpha{\to}x \\ [\![\lambda x.t]\!]_\alpha &:= !\alpha(x\beta).[\![t]\!]_\beta \\ [\![t\,u]\!]_\alpha &:= (\boldsymbol{\nu}\beta\gamma)([\![t]\!]_\beta \mid [\![u]\!]_\gamma \mid \bar\beta\langle\gamma\alpha\rangle) \end{aligned} \tag{6.22}$$

Cette traduction évalue donc exactement une fois la fonction et l'argument puis branche les résultats l'un sur l'autre. Une fonction est un serveur qui attend des requêtes composées d'un pointeur sur l'argument x et d'un canal de sortie β et qui écrit sur β le résultat. Si on développe la traduction d'un terme comme f^2x, on obtient par exemple

$$\begin{aligned} [\![f^2x]\!]_\alpha &\equiv (\boldsymbol{\nu}\beta\gamma\delta\varepsilon)(\beta{\to}f \mid \delta{\to}f \mid \varepsilon{\to}x \mid \bar\delta\langle\varepsilon\gamma\rangle \mid \bar\beta\langle\gamma\alpha\rangle) \\ &\cong (\boldsymbol{\nu}\gamma)(\bar f\langle x\gamma\rangle \mid \bar f\langle\gamma\alpha\rangle) \end{aligned} \tag{6.23}$$

qui correspond bien au protocole expliqué. Il s'agit d'appel par valeur dans un cadre sans contrôle : l'évaluation de $t\,u$ consiste à évaluer t et u en parallèle, sans jamais réduire sous un λ, puis à appliquer le résultat de t à celui de u.

Appel par nom Le cas symétrique de traduction modale des preuves classiques est celui de LKT, plus généralement le cas où ν est suffixe de μ, c'est-à-dire $\mu = \xi\nu$ où ξ commence par ! pour autoriser la promotion dans un contexte de la forme $?\Gamma$, et où ν commence par ? pour autoriser la contraction de conclusions dans le cas classique. L'axiome s'écrit maintenant

$$\dfrac{\dfrac{}{y{\to}\alpha \vdash y : \bar\nu A^\perp, \alpha : \nu A}}{x^{\bar\xi}(y).(y{\to}\alpha) \vdash x : \bar\mu A^\perp, \alpha : \nu A} \tag{6.24}$$

où le processus se simplifie en $x^{\bar\xi}\langle\alpha\rangle$. L'abstraction est inchangée, et dans l'application c'est maintenant l'argument qui reçoit une modalité :

$$\dfrac{\dfrac{\dfrac{Q \vdash ?\Delta, \gamma : \nu A}{x^\xi(\gamma).Q \vdash ?\Delta, x : \mu A} \quad \dfrac{}{y{\to}\alpha \vdash y : \bar\nu B^\perp, \alpha : \nu B}}{x^\xi(\gamma).Q \mid y{\to}\alpha \vdash ?\Delta, xy : \mu A \otimes \bar\nu B^\perp, \alpha : \nu B}}{\beta^{\bar\nu}(xy).(x^\xi(\gamma).Q \mid y{\to}\alpha) \vdash ?\Delta, \beta : \bar\nu(\mu A \otimes \bar\nu B^\perp), \alpha : \nu B} \tag{6.25}$$

Le résultat est coupé contre $P \vdash ?\Gamma, \nu(\mu A \multimap \nu B)$ puis les contextes sont contractés, ce qui donne la traduction générique :

$$[\![t\,u]\!]_\alpha := (\boldsymbol{\nu}\beta)\big([\![t]\!]_\beta \mid \beta^{\bar\nu}(xy).(x^\xi(\gamma).[\![u]\!]_\gamma \mid y{\to}\alpha)\big) \tag{6.26}$$

Le cas de LKT s'obtient en instanciant ξ par ! et ν par ?, ce qui donne la traduction suivante après simplification :

$$\begin{aligned} [\![x]\!]^t_\alpha &:= \bar x\langle\alpha\rangle \\ [\![\lambda x.t]\!]^t_\alpha &:= \bar\alpha(x\beta).[\![t]\!]^t_\beta \\ [\![t\,u]\!]^t_\alpha &:= (\boldsymbol{\nu}\beta)\big([\![t]\!]^t_\beta \mid !\beta(xy).(!x(\gamma).[\![u]\!]^t_\gamma \mid y{\to}\alpha)\big) \end{aligned} \tag{6.27}$$

C'est-à-dire que l'application sur une pile α construit une pile β, réplicable puisque qu'une continuation peut être utilisée plusieurs fois, avec l'argument

(lui aussi réplicable) en tête et la pile α en queue. On obtient une version intuitionniste si on instancie la dérivation avec $\nu = \downarrow$, et après simplification et reformulation, on obtient

$$\begin{aligned} [\![x]\!]_\alpha &:= \bar{x}\langle\alpha\rangle \\ [\![\lambda x.t]\!]_\alpha &:= \alpha(x\beta).[\![t]\!]_\beta \\ [\![t\,u]\!]_\alpha &:= (\boldsymbol{\nu}\beta x)\big([\![t]\!]_\beta \mid !x(\gamma).[\![u]\!]_\gamma \mid \bar{\beta}\langle x\alpha\rangle\big) \end{aligned} \tag{6.28}$$

Le choix de la modalité $\downarrow$ pour ν, au lieu de $\uparrow$ qui est la version linéaire de ?, est arbitraire mais il permet de retomber sur la traduction ci-dessus qui est précisément le codage du λ-calcul dans le π-calcul étudié par Milner et Sangiorgi [62, 72]. Notons également qu'en instanciant la traduction avec le cas dégénéré $\nu = \varepsilon$, on retombe sur la traduction du λ-calcul simplement typé de la section 6.1, mais en perdant l'équation sur les sortes $T = \mu T \to \nu T$ n'a plus de solution.

Cette traduction identifie donc des $\lambda\mu$-termes s'ils construisent les mêmes résultats sur les mêmes conclusions. Il s'agit d'un analogue de la σ-équivalence mais appliqué aux μ-abstractions. Remarquons en revanche que les règles de la σ-équivalence ne s'appliquent plus dans le cas présent.

6.2.2 Simulation en appel par nom

La traduction du système LKT nous donne donc une traduction des λ-termes en processus, en tant qu'arbres de preuves. Cependant, le comportement calculatoire de ces processus n'est pas évident, c'est pourquoi on se penche ici plus précisément sur leurs réductions.

Définissons dès maintenant le langage effectivement simulé par la traduction LKT avant de montrer la bisimulation entre les deux systèmes. Il s'agit d'une légère variation sur le principe du λc-calcul étudié dans les chapitres 2 et 3.

Définition 6.8. Les termes et piles du langage λc^* sont définis par la grammaire suivante, étant donné une famille dénombrable de variables de termes notées $x, y, \ldots$ et une famille de variables de piles notées ρ :

$$\text{termes} \qquad t, u ::= x \mid *t \mid \lambda x.t \mid t\,u \mid k_\pi \mid \mathrm{cc} \tag{6.29}$$

$$\text{piles} \qquad \pi ::= \rho \mid t\,\pi \tag{6.30}$$

Un exécutable $(t)\,\pi$ est la donnée d'un terme t appliqué à une pile π, modulo l'associativité $(t\,u)\,\pi \equiv (t)\,u\,\pi$. La relation de réduction sur les exécutables est définie par les règles suivantes :

$$\begin{aligned} (\lambda x.t)\,u\,\pi &\to (t[*u/x])\,\pi & (\mathrm{cc})\,t\,\pi &\to (*t)\,k_\pi\,\pi \\ (*t)\,\pi &\to (t)\,\pi & (k_\pi)\,t\,\pi' &\to (*t)\,\pi \end{aligned} \tag{6.31}$$

Les variables de termes et de piles seront qualifiées de λ-variables et de μ-variables, car il s'agit exactement de ces notions en $\lambda\mu$-calcul, même si le langage considéré ne fournit pas de lieur μ pour les variables de continuation. Le terme $*t$ est à comprendre comme un pointeur sur le terme t, et la règle de réduction $(*t)\,\pi \to t\,\pi$ correspond au déréférencement du pointeur. Ce terme est donc essentiellement équivalent à t, on l'introduit ici pour obtenir une bisimulation entre la réduction des exécutables et celle de leur traduction.

Définition 6.9. La traduction des exécutables de λc^* est une extension de la traduction LKT du λ-calcul :

$$\begin{array}{ll} [x = t] := !x(\alpha).[\![t]\!]_\alpha & [\alpha = t\,\rho] := !\alpha(xy).([x = t] \mid y{\rightarrow}\rho) \\ [\![x]\!]_\alpha := \bar{x}\langle\alpha\rangle & [\![\rho]\!]_\alpha := \alpha{\rightarrow}\rho \\ [\![*t]\!]_\alpha := (\boldsymbol{\nu}x)(\bar{x}\langle\alpha\rangle \mid [x = t]) & \\ [\![t\,u]\!]_\alpha := (\boldsymbol{\nu}\beta)([\![t]\!]_\beta \mid [\beta = u\,\alpha]) & [\![t\,\pi]\!]_\alpha := (\boldsymbol{\nu}\beta)([\alpha = t\,\beta] \mid [\![\pi]\!]_\beta) \end{array} \tag{6.32}$$

Et pour les termes actifs :

$$\begin{aligned} [\![\lambda x.t]\!]_\alpha &:= \bar{\alpha}(x\beta).[\![t]\!]_\beta \\ [\![k_\pi]\!]_\alpha &:= \bar{\alpha}(x\beta).\bar{x}(\gamma).[\![\pi]\!]_\gamma \\ [\![\mathrm{cc}]\!]_\alpha &:= \bar{\alpha}(x\beta).\bar{x}(\gamma).[\gamma = k_\beta\,\beta] \end{aligned} \tag{6.33}$$

La traduction des exécutables est donnée par $[\![t\,\pi]\!] = (\boldsymbol{\nu}\alpha)([\![t]\!]_\alpha \mid [\![\pi]\!]_\alpha)$.

Remarquons que $[\![t\,u]\!]_\alpha = (\boldsymbol{\nu}\beta)([\![t]\!]_\beta \mid [\beta = u\,\alpha])$, entraîne l'équivalence voulue sur les exécutables : $[\![(t\,u)\,\pi]\!] \equiv [\![(t)\,u\,\pi]\!]$. Les deux processus additionnels $[x = t]$ et $[\alpha = t\,\rho]$ représentent naturellement des affectations de variables, respectivement de terme et de pile, on a en effet le lemme de substitution suivant :

Lemme 6.10. *Pour tout terme (ou pile ou exécutable) t, toute variable de terme x et tout terme u, on a $(\boldsymbol{\nu}x)([\![t]\!]_\alpha \mid [x = u]) \cong [\![t[*u/x]]\!]_\alpha$. Pour toute variable de pile ρ et toute pile π, on a $(\boldsymbol{\nu}\rho)([\![t]\!]_\alpha \mid [\![\pi]\!]_\rho) \cong [\![t[\pi/\rho]]\!]_\alpha$.*

Démonstration. Dans toute traduction de terme, de pile ou d'exécutable, toute occurrence d'une variable x est traduite en un $\bar{x}\langle\beta\rangle$ où β pointe sur la traduction du contexte où apparaît x. De plus, toute les occurrences du nom de canal x correspondent évidemment à des variables, donc dans le processus $(\boldsymbol{\nu}x)([\![t]\!]_\alpha \mid !x(\gamma).[\![u]\!]_\gamma)$ seules les traductions de variables peuvent interagir avec la réplication sur x. Comme cette réplication est l'unique capacité positive sur x, il est garanti que chaque émission interagira uniquement avec $[x = u]$, et ce uniquement lorsqu'elle sera rendue active par la consommation de tous les préfixes derrière laquelle elle se trouve. Ainsi pour tout contexte de processus $C[\,]$ qui ne capture pas x et ne contient x qu'en capacité négative, on a $(\boldsymbol{\nu}x)(C[\bar{x}\langle\beta\rangle] \mid !x(\gamma).[\![u]\!]_\gamma) \cong (\boldsymbol{\nu}x)(C[(\boldsymbol{\nu}x)(\bar{x}\langle\beta\rangle \mid !x(\gamma).[\![u]\!]_\gamma)])$. Le processus obtenu est donc $(\boldsymbol{\nu}x)(C[[\![*u]\!]_\beta] \mid [x = u])$, et en appliquant l'opération successivement à toutes les occurrences de x dans le terme considéré $[\![t]\!]_\alpha$ on obtient par transitivité $(\boldsymbol{\nu}x)([\![t]\!]_\alpha \mid [x = u]) \cong [\![t[*u/x]]\!]_\alpha \mid (\boldsymbol{\nu}x)[x = t] \cong [\![t[*u/x]]\!]_\alpha$.

Dans le cas de substitution de variables de pile, on remarque de même qu'un nom libre ρ n'apparaît dans un processus $[\![t]\!]_\alpha$ qu'en position négative (plus précisément, il apparaît toujours sous la forme $y{\rightarrow}\rho$). Dans $(\boldsymbol{\nu}\rho)([\![t]\!]_\alpha \mid [\![\pi]\!]_\rho)$, l'unique capacité positive sur ρ est donc fournie par $[\![\pi]\!]_\rho$, ce qui garantit que chaque occurrence de ρ dans t y sera connectée. Le processus $[\![\pi]\!]_\rho$ est une composition parallèle de processus répliqués, et il est bloqué par l'attente d'une action sur ρ, par conséquent on peut appliquer la même technique que précédemment et distribuer une copie de $[\![\pi]\!]_\rho$ à chaque occurrence de ρ dans $[\![t]\!]_\alpha$. Étant donnée la traduction des termes et des piles, ceci revient donc à substituer π pour chaque occurrence de ρ dans le terme considéré. □

Grâce à ce lemme, qui explique le rôle des pointeurs $*t$, on peut démontrer qu'il y a une bisimulation entre la réduction des exécutables de λc^* et la réduction de leur traduction. La formulation précise s'écrit modulo bisimilarité :

Théorème 6.11. *Pour tout exécutable $(t)\,\pi$, il existe une réduction $(t)\,\pi \to (t')\,\pi'$ si et seulement s'il existe un P tel que $[\![(t)\,\pi]\!] \to P$ et $P \cong [\![(t')\,\pi']\!]$.*

Démonstration. Par définition, un exécutable de λc^* a toujours au plus une réduction possible, et on remarque sans peine qu'il en est de même pour les traductions d'exécutables. Il suffit donc, pour conclure, de montrer que ces réductions se correspondent.

Une traduction de pile $[\![\pi]\!]_\alpha$ est un processus irréductible dont les noms libres sont les variables (de termes et de piles) libres et le canal principal α. La traduction d'une constante de pile n'a pas de transition et celle d'une pile $t\,\pi$ a une unique transition de la forme

$$[\![t\,\pi]\!]_\alpha \xrightarrow{\alpha(x\beta)} [x = t] \mid [\alpha = t\,\beta] \mid [\![\pi]\!]_\beta$$

De même, la traduction d'une abstraction $[\![\lambda x.t]\!]_\alpha$ est un processus irréductible avec pour noms libres les variables libres et la sortie α, et avec une unique transition

$$[\![\lambda x.t]\!]_\alpha \xrightarrow{\bar\alpha(x\beta)} [\![t]\!]_\beta$$

Par conséquent la traduction d'un exécutable $(\lambda x.t)\,u\,\pi$ n'a pas de transition et a une unique réduction qui s'écrit

$$\begin{aligned}[\![(\lambda x.t)u\pi]\!] &= (\boldsymbol{\nu}\alpha)([\![\lambda x.t]\!]_\alpha \mid [\![u\,\pi]\!]_\alpha)\\ &\to (\boldsymbol{\nu}x\alpha\beta)([\![t]\!]_\beta \mid [x = u] \mid [\alpha = u\,\beta] \mid [\![\pi]\!]_\beta)\end{aligned}$$

Grâce au lemme de substitution, comme la variable x n'apparaît que dans t, le réduit est bisimilaire à $(\boldsymbol{\nu}\alpha\beta)([\![t[*u/x]]\!]_\beta \mid [\alpha = u\,\beta] \mid [\![\pi]\!]_\beta)$, or α n'apparaît nulle part dans t, u et π puisqu'il est introduit par la traduction de l'exécutable $(\lambda x.t)u\pi$, donc le terme $[\alpha = u\,\beta]$ est bloqué et le réduit est bisimilaire à $[\![(t[*u/x])\pi]\!]$.

Dans le cas d'un exécutable $(*t)\pi$, comme $[\![*t]\!]_\alpha = (\boldsymbol{\nu}x)(\bar x\langle\alpha\rangle \mid !x(\beta).[\![t]\!]_\beta)$, le nom α n'apparaît pas en position active, donc la pile reste bloquée et l'unique réduction a lieu sur x, menant à $(\boldsymbol{\nu}\alpha)([\![t]\!]_\alpha \mid (\boldsymbol{\nu}x)!x(\beta).[\![t]\!]_\beta \mid [\![\pi]\!]_\alpha)$ qui est bisimilaire à $(\boldsymbol{\nu}\alpha)([\![t]\!]_\alpha \mid [\![\pi]\!]_\alpha) = [\![(t)\,\pi]\!]$.

Pour la réduction de cc et k_π, on constate que $[\![\mathrm{cc}]\!]_\alpha$ n'a pas d'autre nom libre que α et que $[\![k_\pi]\!]_\alpha$ a pour noms libres α et les noms libres de π. De plus, la sémantique opérationnelle de k_π s'écrit comme suit :

$$\begin{aligned}[\![(k_\pi)\,t\,\pi']\!] &\equiv (\boldsymbol{\nu}\alpha\beta)([\![k_\pi]\!]_\alpha \mid [\alpha = t\,\beta] \mid [\![\pi']\!]_\beta)\\ &\to (\boldsymbol{\nu}x)(\bar x(\gamma).[\![\pi]\!]_\gamma \mid [x = t]) \mid (\boldsymbol{\nu}\alpha\beta)([\![\pi']\!]_\beta \mid [\alpha = t\,\beta])\\ &\cong (\boldsymbol{\nu}x\gamma)(\bar x\langle\gamma\rangle \mid [x = t] \mid [\![\pi]\!]_\gamma)\\ &\equiv (\boldsymbol{\nu}\gamma)([\![(*t)]\!]_\gamma \mid [\![\pi]\!]_\gamma) = [\![(*t)\pi]\!]\end{aligned}$$

et on vérifie qu'il s'agit de la seule réduction possible. De plus, pour cc, on a l'unique réduction suivante :

$$\begin{aligned}[\![\mathrm{cc}\,t\,\pi]\!] &\equiv (\boldsymbol{\nu}\alpha\beta)([\![\mathrm{cc}]\!]_\alpha \mid [\alpha = t\,\beta] \mid [\![\pi']\!]_\beta)\\ &\to (\boldsymbol{\nu}x\beta)(\bar x(\gamma).[\gamma = k_\beta\,\beta] \mid [x = t] \mid [\![\pi]\!]_\beta \mid (\boldsymbol{\nu}\alpha)[\alpha = t\,\beta])\\ &\cong (\boldsymbol{\nu}x\beta\gamma)([\![x]\!]_\gamma \mid [x = t] \mid [\gamma = k_\beta\,\beta] \mid [\![\pi]\!]_\beta)\\ &\equiv (\boldsymbol{\nu}\beta)([\![(*t)\,k_\beta\,\beta]\!] \mid [\![\pi]\!]_\beta)\end{aligned}$$

Et le lemme de substitution permet de conclure en montrant que le réduit est bisimilaire à $[\![(*t)\, k_\pi\, \pi]\!]$.

Dans tous les cas où un exécutable a une transition, nécessairement unique, sa traduction a donc une transition unique qui lui correspond. Les cas où un exécutable n'a pas de réduction sont de deux espèces :

- une variable arrive en tête sous forme $(x)\,\pi$, dans ce cas la pile ne peut interagir et le processus $[\![(x)\,\pi]\!] = (\boldsymbol{\nu}\alpha)(\bar{x}\langle\alpha\rangle \mid [\![\pi]\!]_\alpha)$ n'a qu'une transition étiquetée $\bar{x}(\alpha)$;
- un terme $\lambda x.t$, cc ou k_π arrive en tête face à une variable de pile ρ, dans ce cas l'exécutable n'a pas de réduction mais une transition étiquetée $\bar{\rho}(x\beta)$ qui correspond à l'action de ce terme.

Les τ-transitions des traductions d'exécutables correspondent donc exactement aux réductions d'exécutables, à bisimilarité près. □

Corollaire 6.12. *Tout exécutable est fortement bisimilaire à sa traduction.*

D'autre part, les traductions de cc et k_π sont cohérentes avec le typage. En effet, remarquons déjà que le constructeur de pile $[\![t\,\rho]\!]$ a le type dual de l'abstraction, dans le sens suivant :

$$\dfrac{\dfrac{\dfrac{[\![t]\!]_\gamma \vdash ?\Gamma, \gamma : \nu A}{[x = t] \vdash ?\Gamma, x : \mu A} \qquad y{\rightarrow}\rho \vdash y : \bar{\nu}B^\perp, \rho : \nu B}{[x = t] \mid y{\rightarrow}\rho \vdash ?\Gamma, xy : \mu A \otimes \bar{\nu}B^\perp, \rho : \nu B}}{[\alpha = t\,\rho] \vdash ?\Gamma, \alpha : \bar{\nu}(\mu A \multimap \nu B)^\perp, \rho : \nu B}$$

Par suite, les continuations peuvent se typer :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[\![\pi]\!]_\gamma \vdash \gamma : \bar{\nu}A^\perp, \Delta}{\bar{x}(\gamma).[\![\pi]\!]_\gamma \vdash x : \bar{\mu}A^\perp, \Delta}}{\bar{x}(\gamma).[\![\pi]\!]_\gamma \vdash x : \bar{\mu}A^\perp, \beta : \nu B, \Delta}}{\bar{x}(\gamma).[\![\pi]\!]_\gamma \vdash x\beta : \mu A \multimap \nu B, \Delta}}{[\![k_\pi]\!]_\alpha \vdash \alpha : \nu(\mu A \multimap \nu B), \Delta}$$

d'où on déduit le typage de cc :

$$\dfrac{\dfrac{\dfrac{\dfrac{[\![k_\beta]\!]_\delta \vdash \delta : \nu(\mu A \multimap \nu B), \beta : \nu A}{[\gamma = k_\beta\,\beta] \vdash \gamma : \bar{\nu}(\mu(\mu A \multimap \nu B) \multimap \nu A)^\perp, \beta : \nu A}}{\bar{x}(\gamma).[\gamma = k_\beta\,\beta] \vdash x : \bar{\mu}(\mu(\mu A \multimap \nu B) \multimap \nu A)^\perp, \beta : \nu A}}{\bar{x}(\gamma).[\gamma = k_\beta\,\beta] \vdash x\beta : \mu(\mu(\mu A \multimap \nu B) \multimap \nu A) \multimap \nu A}}{[\![\mathrm{cc}]\!]_\alpha \vdash \alpha : \nu(\mu(\mu(\mu A \multimap \nu B) \multimap \nu A) \multimap \nu A)}$$

On retrouve donc bien la traduction de $\vdash \mathrm{cc} : \forall X \forall Y (((X \to Y) \to X) \to X)$.

6.2.3 Simulation en appel par valeur

Dans la section précédente, on a défini une légère variante du λc-calcul pour laquelle la traduction LKT donne une parfaite bisimulation entre réduction des exécutables et réduction des processus traduits. On applique maintenant la même méthode à la traduction LKQ pour dériver un calcul déterministe en appel par valeur.

Définition 6.13. Soient un ensemble dénombrable de variables de termes notées $x, y, \ldots$ et un ensemble dénombrable le variables de contextes notées α, β. Les termes, valeurs et piles du λcv-calcul sont définis par la grammaire suivante :

$$\text{valeurs} \qquad v, w ::= x \mid \lambda x.t \mid k_\pi \mid \mathrm{cc} \tag{6.34}$$

$$\text{termes} \qquad t, u ::= v \mid t\,u \mid v * w \tag{6.35}$$

$$\text{piles} \qquad \pi ::= \alpha \mid \pi\,t^f \mid \pi\,v^a \tag{6.36}$$

Un exécutable est un couple composé d'un contexte et d'un terme, noté $\pi(t)$, modulo l'équivalence $\pi(t\,u) \equiv \pi\,t^f(u)$. La réduction des exécutables est définie par les règles suivantes :

$$\begin{array}{cc} \pi\,t^f(v) \to \pi\,v^a(t) & \pi\,(\lambda x.t * v) \to \pi\,(t[v/x]) \\ \pi\,w^a(v) \to \pi\,(v * w) & \pi\,(\mathrm{cc} * v) \to \pi\,k_\pi^a(v) \\ & \pi'\,(k_\pi * v) \to \pi\,(v) \end{array} \tag{6.37}$$

Le rôle du terme $v * w$ est similaire au rôle du pointeur $*t$ dans le cas de λc^* : il s'agit d'un état intermédiaire de l'application, destiné à obtenir la bisimilarité avec les processus. Les deux règles de gauche définissent la stratégie de réduction, et les règles de droite définissent la sémantique des valeurs. Les piles ont une structure plus complexe que dans le cas de LKT, car il s'agit maintenant de contextes d'évaluations composés d'applications à gauche et à droite : t^f représente le terme t en position de fonction, alors que v^a représente la valeur v en position d'argument.

Définition 6.14. La traduction des valeurs, termes et piles de λcv est définie inductivement par les règles de la table 6.1 page suivante.

On vérifie sans peine qu'en développant la traduction des termes on retombe exactement sur la traduction LKQ telle qu'on l'a dérivée dans la section 6.2, les autres éléments servent à identifier les étapes précises de réduction. De même que pour la variante en appel par nom, on a ici une propriété de substitution pour les affectations de valeurs et de variables :

Lemme 6.15. *Soit e un exécutable de λcv. Pour toute variable de terme x et toute valeur v sans occurrence de x, on a $(\boldsymbol{\nu} x)(\llbracket e \rrbracket \mid [x = v]) \cong \llbracket e[v/x] \rrbracket$. Pour toute variable de contexte α et toute pile π sans occurrence de α on a $(\boldsymbol{\nu}\alpha)(\llbracket e \rrbracket \mid \llbracket \pi \rrbracket_\alpha) \cong \llbracket e[\pi/\alpha] \rrbracket$.*

Démonstration. Notons que, dans toute traduction d'exécutable, les seules occurrences de canaux libres ont une polarité d'émission et elles correspondent aux occurrences de variables libres. Dans le cas de la composition de $\llbracket e \rrbracket$ avec une valeur $[x = y]$ ou une pile vide $\llbracket \beta \rrbracket_\alpha$, il s'agit d'une composition avec un routeur, et la propriété de substitution 4.25 s'applique.

Dans les autres cas, $[x = v]$ est une réplication de la forme $!x(y\alpha).P$ sans occurrence de x dans P et $\llbracket \pi \rrbracket_\alpha$ est une réplication de la forme $!\alpha(x).Q$ sans occurrence de α dans Q. Par les mêmes arguments que dans le lemme 6.10, le processus $(\boldsymbol{\nu} x)(\llbracket e \rrbracket \,|\, [x = v])$ est donc bisimilaire au même processus où une copie indépendante de $[x = v]$ est connectée à chaque occurrence négative de x dans $\llbracket e \rrbracket$. D'après la définition 6.14, ces occurrences sont toutes de la forme $[y = x] = y{\to}x$, donc on a pour chaque occurrence $(\boldsymbol{\nu} x)([y = x] \,|\, [x = v]) = (\boldsymbol{\nu} x)(y{\to}x \,|\, [x = v]) \cong [y = v])$. En conséquence, par congruence de $\cong$, $(\boldsymbol{\nu} x)(\llbracket e \rrbracket \mid [x = v])$ est bisimilaire à $\llbracket e[v/x] \rrbracket$. La cas des substitutions de piles est analogue. □

valeurs :	$[x = y] := x{\to}y$
	$[x = \lambda y.t] := !x(y\alpha).[\![t]\!]_\alpha$
	$[x = \mathrm{cc}] := !x(y\alpha).[\![y * k_\alpha]\!]_\alpha$
	$[x = k_\pi] := !x(y\alpha).[\![\pi(y)]\!]$
termes :	$[\![v]\!]_\alpha := \bar{\alpha}(x).[x = v]$
	$[\![t\,u]\!]_\alpha := (\boldsymbol{\nu}\beta)([\beta = \alpha\,t^f] \mid [\![u]\!]_\beta)$
	$[\![v * w]\!]_\alpha := (\boldsymbol{\nu}xy)([x = v] \mid \bar{x}\langle y\alpha\rangle \mid [y = w])$
éléments de pile :	$[\alpha = \beta\,t^f] := !\alpha(x).(\boldsymbol{\nu}\gamma)([\![t]\!]_\gamma \mid !\gamma(y).\bar{y}\langle x\beta\rangle)$
	$[\alpha = \beta\,v^a] := (\boldsymbol{\nu}x)([x = v] \mid !\alpha(y).\bar{y}\langle x\beta\rangle)$
piles :	$[\![\beta]\!]_\alpha := \alpha{\to}\beta$
	$[\![\pi\,t^f]\!]_\alpha := (\boldsymbol{\nu}\beta)([\alpha = \beta\,t^f] \mid [\![\pi]\!]_\beta)$
	$[\![\pi\,v^a]\!]_\alpha := (\boldsymbol{\nu}\beta)([\alpha = \beta\,v^a] \mid [\![\pi]\!]_\beta)$
exécutables :	$[\![\pi\,(t)]\!] := (\boldsymbol{\nu}\alpha)([\![\pi]\!]_\alpha \mid [\![t]\!]_\alpha)$

TAB. 6.1 – Traduction de λcv en π-calcul polarisé.

Théorème 6.16. *Pour tout exécutable $\pi\,(t)$, il existe une réduction $\pi\,(t) \to \pi'\,(t')$ si et seulement s'il existe un P tel que $[\![\pi\,(t)]\!] \to P$ et $P \cong [\![\pi'\,(t')]\!]$.*

Démonstration. Notons que l'équivalence $\pi\,(t\,u) \equiv \pi\,t^f(u)$ est une congruence structurelle dans la traduction des exécutables, il suffit donc d'étudier le cas des exécutables de la forme $\pi\,(v)$ et $\pi\,(v * w)$, puisque tout exécutable peut s'écrire sous l'une de ces deux formes.

Toute valeur v est traduite par un processus $\bar{\alpha}(x).[x = v]$ donc toute valeur $[\![v]\!]_\alpha$ a une unique transition, étiquetée $\bar{\alpha}(x)$, qui mène à $[x = v]$ pour une variable fraîche x. On raisonne donc sur les transitions de $[\![\pi]\!]_\alpha$ en fonction de la forme de π :

- Dans le cas d'une pile vide $\pi = \beta$ on a $[\![\pi]\!]_\alpha = \alpha{\to}\beta$ donc $[\![\pi\,(t)]\!]$ n'a pas de réduction et une unique transition étiquetée $\bar{\beta}(x)$.
- Pour une pile $\pi\,t^f$ on a l'unique réduction suivante :

$$\begin{aligned}[\![\pi\,t^f(v)]\!] &\equiv (\boldsymbol{\nu}\alpha\beta)([\![\pi]\!]_\beta \mid [\alpha = \beta\,t^f] \mid [\![v]\!]_\alpha)\\ &\to (\boldsymbol{\nu}x\alpha\beta\gamma)([\![\pi]\!]_\beta \mid [\alpha = \beta\,t^f] \mid [\![t]\!]_\gamma \mid !\gamma(y).\bar{y}\langle x\beta\rangle \mid [x = v])\\ &\equiv (\boldsymbol{\nu}\beta\gamma)([\![\pi]\!]_\beta \mid (\boldsymbol{\nu}\alpha)[\alpha = \pi\,t^f] \mid [\![t]\!]_\gamma \mid [\gamma = \beta\,v^a]) \cong [\![\pi\,v^a(t)]\!]\end{aligned}$$

- Pour une pile $\pi\,w^a$ on a l'unique réduction suivante :

$$\begin{aligned}[\![\pi\,w^a(v)]\!] &\equiv (\boldsymbol{\nu}\alpha\beta)([\![\pi]\!]_\beta \mid [\alpha = \beta\,w^a] \mid [\![v]\!]_\alpha)\\ &\to (\boldsymbol{\nu}xy\alpha\beta)([\![\pi]\!]_\beta \mid [\alpha = \beta\,w^a] \mid [x = w] \mid \bar{y}\langle x\beta\rangle \mid [y = v])\\ &\equiv (\boldsymbol{\nu}\beta)([\![\pi]\!]_\beta \mid (\boldsymbol{\nu}\alpha)[\alpha = \beta\,w^a] \mid [\![v * w]\!]_\beta) \cong [\![\pi\,(v * w)]\!]\end{aligned}$$

Une traduction de pile $[\![\pi]\!]_\alpha$ a donc au plus une transition, sur son canal de référence α. D'autre part, une traduction de terme $[\![v{*}w]\!]_\alpha$ n'a pas de transition

étiquetée α mais une τ-transition unique, donc $[\![\pi\,(v*w)]\!]$ a toujours une unique réduction qui est celle de $[\![v*w]\!]_\alpha = (\boldsymbol{\nu} xy)([x=v] \mid [y=w] \mid \bar{x}\langle y\alpha\rangle)$, et cette transition dépend donc uniquement de la forme de v. Dans chaque, le lemme de substitution s'applique :

– Pour l'abstraction, on a la réduction

$$\begin{aligned}[\![\pi\,(\lambda z.t*w)]\!] &\equiv (\boldsymbol{\nu} xy\alpha)([\![\pi]\!]_\alpha \mid !x(z\beta).[\![t]\!]_\beta \mid \bar{x}\langle y\alpha\rangle \mid [y=w])\\ &\to (\boldsymbol{\nu} xyz\alpha\beta)([\![\pi]\!]_\alpha \mid [x=\lambda z.t] \mid [\![t]\!]_\beta \mid z{\to}y \mid \beta{\to}\alpha \mid [y=w])\\ &\cong (\boldsymbol{\nu} z)([\![\pi\,(t)]\!] \mid [z=w]) \mid (\boldsymbol{\nu} x)[x=\lambda z.t] \cong [\![\pi\,(t[w/z])]\!]\end{aligned}$$

– Pour la sauvegarde de continuations, on a

$$\begin{aligned}[\![\pi\,(\mathrm{cc}*w)]\!] &\equiv (\boldsymbol{\nu} xy\alpha)([\![\pi]\!]_\alpha \mid !x(z\beta).[\![z*k_\beta]\!]_\beta \mid \bar{x}\langle y\alpha\rangle \mid [y=w])\\ &\to (\boldsymbol{\nu} xyz\alpha\beta)([\![\pi]\!]_\alpha \mid [x=\mathrm{cc}] \mid [\![z*k_\beta]\!]_\beta \mid z{\to}y \mid \beta{\to}\alpha \mid [y=w])\\ &\cong (\boldsymbol{\nu} z\beta)([\![\beta\,(z*k_\beta)]\!] \mid [\![\pi]\!]_\beta \mid [z=w]) \cong [\![\pi\,(w*k_\pi)]\!]\end{aligned}$$

– Enfin, pour les continuations sauvegardées on a

$$\begin{aligned}[\![\pi'\,(k_\pi*w)]\!] &\equiv (\boldsymbol{\nu} xy\alpha)([\![\pi']\!]_\alpha \mid !x(z\beta).[\![\pi\,(z)]\!] \mid \bar{x}\langle y\alpha\rangle \mid [y=w])\\ &\to (\boldsymbol{\nu} xyz\alpha\beta)([\![\pi']\!]_\alpha \mid [x=k_\pi] \mid [\![\pi\,(z)]\!] \mid z{\to}y \mid \beta{\to}\alpha \mid [y=w])\\ &\cong (\boldsymbol{\nu} z)([\![\pi\,(z)]\!] \mid [z=w]) \mid (\boldsymbol{\nu}\alpha)[\![\pi']\!]_\alpha \cong [\![\pi\,(w)]\!]\end{aligned}$$

Les réductions des exécutables correspondent donc exactement aux réductions de leur traduction, à bisimulation près. □

6.3 Correction sémantique

Les traductions du λ-calcul de la section précédentes sont issues de traductions entre systèmes logique. La correction opérationnelle de ces traductions est énoncée sous la forme des théorèmes de simulation entre un exécutable, pour une stratégie de réduction donnée, et sa traduction. Le fait que la traduction soit typée par construction peut être considéré comme la correction logique des traductions :

Proposition 6.17. *Pour toute traduction modale $(\cdot)^*$ et tout λ-terme t, $\vdash t : A$ est dérivable si et seulement si $[\![t]\!]^*_\alpha \vdash \alpha : A^*$ est dérivable.*

Le théorème d'adéquation de la logique des comportements entraîne donc que, pour toute observation, le processus $[\![t]\!]_\alpha$ réalise la traduction du type A. Par conséquent, on obtient gratuitement le fait que la réduction des termes typés termine dans les stratégies de réductions induites par les traductions :

Théorème 6.18. *Soit un λ-terme typé $\Gamma \vdash t : A$. Pour toute constante de pile ρ, la réduction de $(t)\rho$ en appel par nom et la réduction de $\rho(t)$ en appel par valeur terminent.*

En anticipant sur les résultats de spécification du chapitre 7, le théorème 7.2 garantit que la traduction d'un exécutable bien typé et irréductible est capable d'interagir sur l'un de ses noms libres. En d'autres termes, dans toute stratégie issue d'une traduction logique, les seules situations de blocage dans la réduction

d'un exécutable correspondent à l'arrivée d'une variable en tête ou à l'application d'un terme actif (de la forme $\lambda x.t$, cc ou k_π) sur une pile vide.

On cherche maintenant à établir un argument de correction sémantique de la traduction. Tant du côté classique que du côté linéaire, l'adéquation du typage s'exprime par une notion de réalisabilité fondée sur des principes similaires dans les deux cas. On se propose donc de mettre en correspondance les deux notions.

6.3.1 Observations et environnements

La traduction LKT impose l'interface des traductions de λ-termes clos : à partir de la traduction $(A \to B)^* = !?A^* \multimap ?B^*$ on sait qu'un terme typé $\vdash t : A$ sera traduit en un $[\![t]\!]_\alpha \vdash \alpha : ?A^*$, où A ne peut être qu'un type fonctionnel. Notons Λ l'interface des traductions de termes clos, on a donc $\Lambda = \uparrow(\uparrow\bar{\Lambda}, \Lambda)$; de même l'interface des traductions de piles closes se définit par $\Pi = \downarrow(\downarrow\bar{\Pi}, \Pi)$.

Dans la réalisabilité concurrente, l'orthogonalité est définie entre processus d'interfaces opposées, au moyen d'un $\perp$ qui est un ensemble de processus clos, à la différence du cas classique où aucune restriction n'est imposée. De fait, l'utilisation de variables libres et de constantes de piles dans $\perp\!\!\!\perp$ fait partie de la technique employée en réalisabilité classique (voir par exemple la démonstration de la proposition 3.25 page 48). Dans la suite, on va donc supposer qu'un ensemble fini de variables libres x_i et ρ_j est fixé, et on note $E = x_1 : \downarrow\Lambda, \ldots, x_n : \downarrow\Lambda, \rho_1 : \Pi, \ldots, \rho_k : \Pi$ l'interface sur laquelle l'environnement doit instancier ces variables. Avec ce choix de variables libres, la traduction d'un terme sur une sortie α est donc un processus d'interface $E \to \alpha : \Lambda$ et la traduction d'une pile sur une sortie α est un processus d'interface $E \to \alpha : \Pi$. Un exécutable se traduit donc en un processus d'interface $\neg E$.

Étant donnée une observation $\perp$ sur les processus, on a donc on notion naturelle d'orthogonalité entre un exécutable et un environnement qui en instancie les variables libres. Comme illustré plus haut, le comportement d'un exécutable se décompose en des réductions internes et des transitions observables qui correspondent aux requêtes sur la valeur des variables libres. L'observation d'un exécutable par un environnement consiste donc à instancier ces variables par des processus particuliers, qui ne seront pas nécessairement des λ-termes. En quelque sorte, l'accès aux variables libres correspond à des appels système, et le système ne peut observer un programme que lorsque celui-ci lui donne la main.

Définition 6.19. Soit $\perp$ une observation sur les processus et soit $\mathcal{E}$ un comportement d'interface E. Le couple $(\perp, \mathcal{E})$ induit une observation sur λc^* définie comme $t \perp\!\!\!\perp \pi$ si et seulement si $[\![(t)\,\pi]\!] \in \mathcal{E}^\perp$.

L'observation sur λc^* se compose donc de $\perp$ qui détermine le genre de propriété à observer (terminaison, may-testing, must-testing, etc) et de l'environnement $\mathcal{E}$ qui détermine le rôle des variables dans cette observation. Bien entendu, comme la réduction des exécutables est déterministe, les différents styles de tests deviennent équivalents, à moins que l'environnement n'introduise du non-déterminisme.

Exemple 6.20. On se place dans le cas habituel où $\perp$ est l'observation par test strict. Soit x_i l'une des variables libres, et posons $\mathcal{E} = \{x_i(\alpha).*\}$. Comme une traduction de λ-terme ne peut jamais contenir $*$, il est évident que $[\![(t)\pi]\!] \perp \mathcal{E}$ implique que $[\![(t)\pi]\!]$ doit atteindre un état où il a une transition $\bar{x}_i(\alpha)$, et d'après

l'étude précédente du comportement des termes ceci revient à dire qu'il existe une pile π' telle que $(t)\pi$ se réduit en $(x_i)\pi'$. Ainsi on a dans ce cas $t \perp\!\!\!\perp \pi$ si et seulement si $(t)\pi \to (x_i)\pi'$ pour π' quelconque.

Exemple 6.21. On peut raffiner l'exemple précédent en observant la structure de la pile. Remarquons que l'unique transition d'un processus $[\![(x)\pi]\!]$ est étiquetée $\bar{x}(\alpha)$ et mène à un processus P irréductible. De plus, on a $P \cong \alpha{\to}\rho$ si π est une variable ρ, et si π n'est pas vide on a toujours $\mathcal{F}(P) = \mathrm{id}$ et P a une transition sur α. Les deux cas peuvent être identifiés par le test $\bar{\alpha}(y\beta) \mid \rho(y\beta).*$ qui fait accepter dans le premier cas et bloque l'évaluation dans le second. Ainsi, en posant $\mathcal{E} = \{x_i(\alpha).(\bar{\alpha}(y\beta) \mid \rho_j(y\beta).*)\}$, on a $t \perp\!\!\!\perp \pi$ si et seulement si $(t)\pi \to (x_i)\rho_j$.

À partir de ce genre d'observation, il est possible de retrouver des théorèmes de spécification avec des techniques similaires à celles des chapitres 2 et 3, en faisant varier l'observateur $\mathcal{E}$. Reprenons le cas élémentaire de l'identité :

Proposition 6.22. *Soit t un λc-terme clos de type $\forall X\,(X \to X)$. Pour tout terme u et toute pile π on a la réduction $(t)\,u\,\pi \to (u)\,\pi$.*

Démonstration. Soient x et ρ deux variables fraîches. Par construction on a donc $[\![(t)\,x\,\rho]\!] \vdash x : \bar{\mu}X^{\perp}, \rho : \nu X$. Notons P ce processus. Prenons pour $\perp$ une observation par test et posons $X = \mathbf{0}$. On a alors

$$\dfrac{P \vdash x : \bar{\mu}X^{\perp}, \rho : \nu X \qquad \dfrac{* \vdash y\beta : \mathbf{0}}{!x(\alpha).\bar{\alpha}(y\beta).* \vdash x : \mu X} \qquad \dfrac{\Omega \vdash y\beta : \top}{!\rho(y\beta).\Omega \vdash \rho : \bar{\nu}X^{\perp}}}{(\boldsymbol{\nu} x\rho)(P \mid !x(\alpha).\bar{\alpha}(y\beta).* \mid !\rho(y\beta).\Omega) \vdash \perp}$$

Donc toute réduction du processus obtenu atteint un état acceptant, c'est-à-dire libère $*$. Comme P est la traduction d'un λ-terme, sa réduction est déterministe, et comme $*$ est libéré il existe un réduit de P qui émet une action sur x. Cette réduction correspond donc à $(t)\,x\,\rho \to (x)\,\pi$ pour un certain π.

Soit Q le processus réduit de P qui correspond à $(x)\,\pi$. Les comportements sont clos par réduction, donc Q réalise aussi $x : \bar{\mu}X^{\perp}, \rho : \nu X$ pour tout X. Posons maintenant $X = \top$, on montre comme au-dessus que le processus $(\boldsymbol{\nu} x\rho)(Q \mid !x(\alpha).\bar{\alpha}(y\beta).\Omega \mid !\rho(y\beta).*)$ réalise $\perp$, or Q émet des transitions $\bar{x}(\alpha)$ puis $\alpha(y\beta)$. Cette action sur α est nécessairement synchronisée contre $!\rho(y\beta).*$, sinon elle déclencherait une divergence dans un état non acceptant. Par conséquent, π est nécessairement la pile vide ρ, et on a donc $(t)\,x\,\rho \to (x)\,\rho$.

Les règles de réduction de λc^* sont invariantes par substitution, donc en substituant u pour x et π pour ρ on en déduit $(t)\,u\,\pi \to (u)\,\pi$. □

6.3.2 D'une réalisabilité à l'autre

Il faut donc introduire une nouvelle notion d'orthogonalité entre processus, plus complexe que la définition naturelle du chapitre 5. On suppose choisie une observation $\perp$ au sens standard. L'idée est de choisir un environnement de référence, sous forme d'un comportement $\mathcal{E}$ donné, et de définir l'orthogonalité modulo $\mathcal{E}$. On dit alors que deux processus sont orthogonaux si leur composition est orthogonale à ce comportement.

Définition 6.23. Soit E une interface et soient deux processus $P :: E \to I \to J$ et $Q :: E \to J \to K$. La composition de P et Q modulo E est le processus $P \cdot_{E,J} Q = (\boldsymbol{\nu} J)(P \mid Q)$ d'interface $E \to I \to K$.

Définition 6.24. Une observation relative d'interface E est un couple $(\bot, \mathcal{E})$ où $\bot$ est une observation au sens de la réalisabilité concurrente et $\mathcal{E}$ est un comportement d'interface E, appelé environnement.

Pour toute interface I, deux processus $P :: E \to I$ et $Q :: E \to \neg I$ sont $\mathcal{E}$-orthogonaux si $(\boldsymbol{\nu} I)(P \mid Q) \ \bot \ \mathcal{E}$, on note alors $P \perp_{\mathcal{E}} Q$. Le $\mathcal{E}$-orthogonal d'un ensemble $\mathcal{A} : E \to I$ est l'ensemble $\mathcal{A}^{\mathcal{E}} = \{\, Q :: E \to \neg I \mid P \perp_{\mathcal{E}} Q \,\}$. Un $\mathcal{E}$-comportement d'interface I est un ensemble $\mathcal{A} : E \to I$ tel que $(\mathcal{A}^{\mathcal{E}})^{\mathcal{E}} = \mathcal{A}$.

Proposition 6.25. *Les $\mathcal{E}$-comportements sont des comportements.*

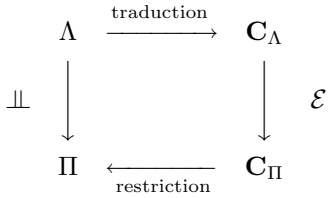
Démonstration. Soit $\vec{x}$ le support de E. Notons δ le processus $\vec{x}{\to}^{E}\vec{y} \,|\, \vec{x}{\to}^{E}\vec{z}$, en utilisant les notations de la définition 4.26 page 65, de sorte qu'on a $\delta :: (\vec{x} : E) \to (\vec{y} : E), (\vec{z} : E)$. Notons $(\cdot)'$ l'opération de renommage de $\vec{x}$ en $\vec{y}$ et $(\cdot)''$ l'opération de renommage de $\vec{x}$ en $\vec{z}$. La composition modulo E se réécrit alors comme $P \cdot_{E,I} Q = \delta \cdot_{E',E''} (P' \cdot_I Q'')$. Pour tout ensemble $\mathcal{A} : E \to I$ on a alors $P \in \mathcal{A}^{\mathcal{E}}$ si et seulement si, pour tout $Q \in \mathcal{A}$ et tout $R \in \mathcal{E}$, $\delta \cdot (P' \cdot Q'') \perp R$, qui est équivalent à $P' \perp Q'' \cdot (\delta \cdot R)$, donc on a $(\mathcal{A}^{\mathcal{E}})' = (\mathcal{A}'' \cdot (\delta \cdot \mathcal{E}))^{\perp}$. Les $\mathcal{E}$-orthogonaux sont donc des comportements. □

L'orthogonalité modulo un environnement correspond précisément à la notion d'orthogonalité utilisée en réalisabilité classique. Pour tout ensemble X de termes ou de piles, notons X^* l'ensemble des traductions d'éléments de X. Notons Λ^* l'ensemble des traductions de termes et Π^* celui des traductions de piles. On a alors la caractérisation suivante :

Théorème 6.26. *Soit $(\bot, \mathcal{E})$ une observation relative, soit $\perp\!\!\!\perp$ l'observation induite sur λc^*. Pour tout ensemble de termes X on a $(X^{\perp\!\!\!\perp})^* = (X^*)^{\mathcal{E}} \cap \Pi^*$.*

Démonstration. Par définition de $\perp\!\!\!\perp$, pour tout terme t et toute pile π on a $t \perp\!\!\!\perp \pi$ si et seulement si $[\![(t)\pi]\!] \perp \mathcal{E}$. D'autre part, on a $[\![(t)\pi]\!] = (\boldsymbol{\nu}\alpha)([\![t]\!]_\alpha \mid [\![\pi]\!]_\alpha) = [\![t]\!]_{\alpha \cdot_{E,\alpha}} [\![\pi]\!]_\alpha$, donc $[\![(t)\pi]\!]$ est orthogonal à $\mathcal{E}$ si et seulement si $[\![t]\!]_\alpha$ est orthogonal à $[\![\pi]\!]_\alpha$ modulo $\mathcal{E}$. L'ensemble des piles orthogonales à tout terme de X est donc l'ensemble des π telles que $[\![\pi]\!]_\alpha$ soit $\mathcal{E}$-orthogonal à $\{\, [\![t]\!]_\alpha \mid t \in X \,\}$, c'est-à-dire élément $(X^*)^{\mathcal{E}}$. Les traductions de piles de $X^{\perp\!\!\!\perp}$ sont donc exactement les traductions de piles qui sont éléments de $(X^*)^{\mathcal{E}}$, c'est-à-dire que $(X^{\perp\!\!\!\perp})^*$ est égal à $(X^*)^{\mathcal{E}} \cap \Pi^*$. □

En d'autres termes, on a la commutation suivante entre l'orthogonal en réalisabilité classique et l'orthogonal concurrent :



L'orthogonal séquentiel d'un ensemble de termes X est la traduction inverse de l'orthogonal concurrent de la traduction de X. Notons qu'il est nécessaire d'appliquer une traduction inverse pour avoir une égalité. Pour un comportement de termes X, les traductions X^* et $(X^{\perp\!\!\!\perp})^*$ sont évidemment orthogonales, par construction de $\perp\!\!\!\perp$, mais a priori l'un n'est pas l'orthogonal de l'autre. La raison

est que dans un tel comportement X^* ne se trouvent que des processus très réguliers issus des traductions de λ-termes.

En conséquence, pour les observations considérées, un comportement de termes est toujours la traduction inverse d'un comportement concurrent. Pour les observations par test équitable ou strict les comportements sont clos par réduction, c'est donc le cas pour tout $\perp\!\!\!\perp$ issu d'une observation relative.

Cette décomposition permet de retrouver la définition des connecteurs en réalisabilité classique. Pour un type A du système F, notons A^* sa traduction LKT, notons $[A]$ sa valuation en tant qu'ensemble de piles et $[\![A]\!] = [A]^{\perp\!\!\!\perp}$ sa valeur en tant qu'ensemble de termes. Pour tout ensemble de processus X de sorte Π (resp. Λ), notons X^p (resp. X^t) l'ensemble des piles (resp. des termes) dont la traduction est élément de X, de sorte que $[A] = (!{A^*}^{\perp})^p$ et $[\![A]\!] = (?A^*)^t$. La traduction LKT est définie par $(A \to B)^* = !?A^* \multimap ?B^*$, d'où

$$\begin{aligned}
[A \to B] &= (!(!?A^* \multimap ?B^*)^{\perp})^p \\
&= (!(!?A^* \otimes !{B^*}^{\perp}))^p \\
&= (?A^*)^t \cdot (!{B^*}^{\perp})^p \qquad \text{par définition de la traduction} \\
&= [\![A]\!] \cdot [B]
\end{aligned}$$

L'influence de l'observation concurrente sur l'observation induite pour les exécutables séquentiels est à clarifier. En effet, les exécutables traduits du λc-calcul sont toujours déterministes, donc pour eux may, must et fair testing sont équivalents, et c'est en présence de combinateurs capables de créer du parallélisme que $\perp$ prendra son sens. L'analogie informatique illustre le type de propriété qu'il serait possible de décrire : chaque exécutable traduit du λc-calcul représente un thread, et le système d'exploitation est un processus concurrent qui établit des connexions entre ces threads. Par suite, le choix de $\perp$ dépend du genre de système à modéliser : le test équitable correspond au parallélisme, situation où la divergence d'un processus n'empêche pas le reste du système de tourner ; le test strict représente plutôt l'exécution de plusieurs programmes sur une seule machine, où la divergence de l'un des threads empêche les autres de s'exécuter. Le choix de l'observation détermine donc quels combinateurs il serait possible de décrire sous forme de processus, par exemple l'observation équitable devrait permettre l'écriture d'un combinateur comme le ou parallèle, ou l'opérateur *amb* de McCarthy [60]. À l'heure actuelle, ces considérations sont essentiellement des pistes pour de futures applications du cadre de la réalisabilité concurrente.

Chapitre 7

Exemples d'applications

7.1 Spécification de processus

L'une des applications de la réalisabilité classique introduite dans le chapitre 2 est la spécification de processus séquentiels au moyen du typage, il paraît donc envisageable d'appliquer les mêmes idées au cas concurrent. L'objectif de ce chapitre est donc d'explorer les possibilités de spécification de processus au moyen de la réalisabilité concurrente. Le principe est le même que celui utilisé dans le chapitre 3 : étant donné un processus typé, on choisit une observation pour laquelle l'interprétation sémantique des types est valide, puis on construit des environnements de test afin d'en déduire les interactions possibles du processus considéré. Comme le formalisme de la réalisabilité concurrente est moins mature que celui de la réalisabilité classique, il est évident que des techniques et des intuitions restent à établir pour en tirer des spécifications aussi développées que dans le cas séquentiel. Les quelques résultats présentés ici donnent néanmoins une idée des techniques de preuve sémantique applicables aux processus typés.

7.1.1 Construction de tests

Dans toute la suite, on utilisera des observations par test telles qu'elles sont décrites dans la section 5.3.

Définition 7.1. Un terme propre (ou quasi-preuve) est un processus dans lequel n'apparaît pas la constante $*$. Dans une observation donnée, un type est dit proprement réalisable si son interprétation contient un terme propre.

Cette définition, très naturelle, identifie les processus pour lesquels se feront toutes les démonstrations de spécification. Les autres processus sont ceux qui jouent le rôle de tests, on considère en effet que c'est aux tests de décréter si une expérience est réussie ($*$ est apparu en position active) ou pas.

Dans les observations par tests, certains termes jouent naturellement un rôle particulier. Par construction $*$ fait partie de tout comportement, et c'est naturellement l'ingrédient de base de n'importe quel test. On ajoute donc la règle de typage suivante :

$$\frac{\Phi \Rightarrow \Gamma}{\Phi \Rightarrow * \vdash \Gamma}$$

On utilisera aussi le test strict, qui est sensible à la divergence, dans lequel l'utilisation d'un processus divergent Ω permet de raffiner la construction des tests. Il faut ici rappeler que l'adéquation du typage dans le cas du test strict (voir théorèmes 5.44 page 89 et 5.45 page 90) impose des contraintes : les constantes additives ne doivent être introduites que sous une modalité, et les quantifications sont indexées par les comportements non divergents. On ne peut donc pas employer de règle de typage $\Omega \vdash \top, \Gamma$ car elle viole cette contrainte. On pose donc les notations suivantes :

$$\Omega := (\boldsymbol{\nu} u)(\bar{u} \mid !u.\bar{u}) \qquad B := \tau.* = (\boldsymbol{\nu} x)(\bar{x} \mid x.*) \qquad B_{u^\varepsilon} := u^\varepsilon(\vec{x}).\Omega \mid B \tag{7.1}$$

de sorte que Ω diverge, B réalise $\bot_\emptyset$ et B_{u^ε} réalise $\bot_{u:\varepsilon I}$. En particulier B_u a la propriété de terminer mais de diverger face à toute action sur le canal u, ce qui entraîne que $B_u \cdot B_{\bar{u}}$ diverge. on peut donc poser la règle

$$\frac{}{\Phi \Rightarrow B_{u^\varepsilon} \vdash u : \bot_{\varepsilon I}}$$

et cette règle est compatible avec toutes les observations considérées.

Enfin, dans toutes les observations par test, $\bot$ est stable par composition parallèle, donc dans toutes ces observations la règle mix est valide (voir proposition 5.25 page 81). Par conséquent, pour tous comportements $\mathcal{A}$ et $\mathcal{B}$ on a $\mathcal{A} \otimes \mathcal{B} \subseteq \mathcal{A} \parr \mathcal{B}$, et on s'autorise la règle de typage associée :

$$\frac{\Phi \Rightarrow P \vdash \Gamma \qquad \Phi \Rightarrow Q \vdash \Delta}{\Phi \Rightarrow P \mid Q \vdash \Gamma, \Delta}$$

Dans le cas de l'observation angélique et de l'observation équitable, on a l'égalité $\bot_I = \mathbf{0}_I$ pour toute sorte I, or $\mathbf{0}_I$ est par définition inclus dans tout comportement et $\bot_I$ peut être introduit dans n'importe quel séquent. En conséquence, ces observations sont valides pour la logique affine, c'est-à-dire qu'elles autorisent la règle d'affaiblissement sur n'importe quelle formule :

$$\frac{\Phi \Rightarrow P \vdash \Gamma}{\Phi \Rightarrow P \vdash \Gamma, \vec{x} : A}$$

7.1.2 Bon comportement des termes typés

Le système de types, sans les extensions évoquées plus haut, interprète la logique linéaire. Il s'agit d'un système très régulier, par conséquent les processus typés dans ce système sont extrêmement civilisés. Une première conséquence est que tous ces processus terminent, comme on l'a montré dans le théorème 5.45. Il serait possible également de montrer la confluence des termes typés. Une autre propriété importante pour des processus concurrents, à laquelle on s'intéresse maintenant, est l'absence de blocage.

Formuler l'absence de blocage en général n'est pas évident, car il s'agit d'une propriété subjective : le fait qu'une partie d'un processus n'agisse pas peut être considéré comme une erreur dans certains programmes, alors que dans d'autres situations ce peut être le comportement attendu. Par exemple si l'on écrit $(\boldsymbol{\nu} u)(\bar{u} \mid u.P \mid u.Q)$ pour représenter un choix non déterministe entre P et Q, après réduction on obtient soit $P \mid (\boldsymbol{\nu} u)u.Q$ soit $(\boldsymbol{\nu} u)u.P \mid Q$, et dans chaque cas il existe une action bloquée, or c'est précisément l'effet recherché. Un autre cas usuel est celui d'un serveur : si l'on écrit $(\boldsymbol{\nu} u)(!u(x).P \mid Q)$, il est raisonnable

que l'interaction avec Q fasse un certain nombre de requêtes sur u puis n'utilise plus le serveur, ce qui mène à une situation comme $(\boldsymbol{\nu} u)!u(x).P \,|\, Q'$ où le serveur est bloqué.

L'énoncé qui suit est une façon d'exprimer le fait qu'un processus bien typé n'atteint pas de situation de blocage du point de vue de son environnement. On ne considère pas le blocage éventuel d'actions au sein d'un processus, mais uniquement sa capacité à interagir.

Théorème 7.2 (non-blocage)**.** *Soit un terme typé $\Phi \Rightarrow P \vdash \Gamma$ tel que, dans le type Γ, toutes les constantes et variables propositionnelles apparaissent sous une modalité. Pour toute réduction $P \to^* P'$ il existe une réduction $P' \to^* P''$ telle que P'' a une transition visible.*

Démonstration. On se place dans l'observation équitable, pour une valuation quelconque des variables propositionnelles. La contrainte imposée sur Γ signifie que ce type est une combinaison, par les connecteurs $\otimes$, $\parr$, $\vee$ et $\wedge$, de formules $[\varepsilon_i]A_i$, avec possiblement des quantificateurs. Notons $\mathcal{I} = u_1 : \varepsilon_1 I_1, \ldots, u_n : \varepsilon_n I_n$ l'interface associée à $\Gamma^\perp$, et posons $T_\mathcal{I} := u_1^{\varepsilon_1}(\vec{x}_1).*|\cdots|u_n^{\varepsilon_n}(\vec{x}_n).*$. Montrons $T_\mathcal{I} \Vdash \Gamma^\perp$ par induction sur la formule $\Gamma^\perp$.

Le cas de base est celui d'une formule $u : [\varepsilon]A$. Quelque soit la formule A on a $* \Vdash \vec{x} : A$, donc $T_\mathcal{I} = u^\varepsilon(\vec{x}).* \Vdash u : [\varepsilon]A$ se déduit par la règle des modalités. Pour le tenseur, en supposant $T_\mathcal{I} \Vdash \vec{x} : A$ et $T_\mathcal{J} \Vdash \vec{y} : B$, on a par la règle du tenseur $T_{\mathcal{I},\mathcal{J}} = T_\mathcal{I} \,|\, T_\mathcal{J} \vdash \vec{x}\vec{y} : A \otimes B$. Cette construction est valide également pour le $\parr$ puisque le règle mix est satisfaite dans l'observation considérée. Pour le cas de l'intersection, remarquons que si le séquent $\vdash \vec{x} : A \wedge B$ est bien formé alors A et B sont de même sorte, donc par hypothèse d'induction on a $T_\mathcal{I} \Vdash \vec{x} : A$ et $T_\mathcal{I} \vdash \vec{x} : B$ avec le même terme, donc $T_\mathcal{I} \vdash \vec{x} : A \wedge B$. L'argument s'applique aussi au connecteur $\vee$. Pour les quantificateurs, $(\forall X : I)A$, $(\exists X : I)A$ et A ont toujours la même sorte donc le même argument s'applique aussi.

Le processus $T_\mathcal{I}$ réalise donc $\Gamma^\perp$, et par hypothèse P réalise Γ, donc on a $(\boldsymbol{\nu}\mathcal{I})(P \,|\, T_\mathcal{I}) \in \perp$. Par définition de l'observation équitable, cela signifie que tout réduit de $P \,|\, T_\mathcal{I}$ peut se réduire en un état acceptant. Soit alors une réduction $P \to^* P'$. Par hypothèse P est un terme propre donc il ne contient pas $*$ et P' non plus. Le fait que $P' \,|\, T_\mathcal{I}$ puisse atteindre un état acceptant entraîne donc qu'il existe un réduit P'' de P' capable de produire une action $\bar{u}_i^{\varepsilon_i}(\vec{x}_i)$. □

Cette spécification est très superficielle, comme l'illustre la démonstration où l'on construit un test qui ne dépend que de la sorte du type considéré. Il est possible de raffiner le résultat pour prendre en compte les connecteurs employés. Illustrons cela dans un cas simple :

Proposition 7.3. *Soit un processus typé $\Phi \Rightarrow P \vdash u_1 u_2 : \uparrow A_1 \otimes \uparrow A_2$ qui n'utilise pas le démon. Pour tout $i \in \{1, 2\}$, pour toute réduction $P \to^* Q$ il existe une réduction $Q \to^* R$ telle que R peut émettre une action sur u_i.*

Démonstration. On se place dans l'observation équitable, pour une valuation arbitraire de type Φ. Soit la dérivation suivante, utilisant la règle du démon :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{* \vdash \vec{x} : A_1^\perp, \vec{y} : A_2^\perp}}{u_2(\vec{y}).* \vdash \vec{x} : A_1^\perp, u_2 : \downarrow A_2^\perp}}{u_1(\vec{x}).u_2(\vec{y}).* \vdash u_1 : \downarrow A_1^\perp, u_2 : \downarrow A_2^\perp}}{u_1(\vec{x}).u_2(\vec{y}).* \vdash u_1 u_2 : \downarrow A_1^\perp \parr \downarrow A_2^\perp}$$

Notons $T_1 = u_1(\vec{x}).u_2(\vec{y}).*$ ce processus de test, on a alors $(\boldsymbol{\nu} u_1 u_2)(P \mid T_1) \Vdash \bot$. Pour toute réduction $P \to^* Q$, on a la réduction $P \mid T_1 \to^* Q \mid T_1$ donc $Q \mid T_1$ peut atteindre l'acceptation, par définition de $\bot$. Comme P est bien typé, il ne contient pas d'occurrence de $*$, donc Q non plus, et la seule façon d'atteindre l'activation est de consommer le préfixe dans $T_1 = u_1(\vec{x}).u_2(\vec{y}).*$, ce qui prouve que Q a un réduit susceptible d'émettre l'action $\bar{u}_1(\vec{x})$. En considérant le test $T_2 = u_2(\vec{y}).u_1(\vec{x}).*$ on montre de même que Q a un réduit susceptible d'émettre l'action $\bar{u}_2(\vec{y})$. □

Ce résultat illustre le sens calculatoire des connecteurs $\otimes$ et $⅋$: lorsque deux conclusions d'un terme typé sont connectées par un tenseur, les actions présentes sur les canaux correspondants doivent être indépendantes. En revanche, si deux conclusions sont reliées par $⅋$, les actions correspondantes peuvent être corrélées, notamment par une relation de préfixage, comme illustré dans la preuve précédente. Les règles de construction et de composition des processus typés servent alors à garantir que les diverses corrélations ne sont pas contradictoires, c'est-à-dire que les situations d'interblocage sont évitées.

7.1.3 Schémas de routage

On s'intéresse ici au fragment le plus simple du système de types : le calcul propositionnel en logique linéaire multiplicative. C'est-à-dire que l'on considère uniquement des combinaisons multiplicatives de variables propositionnelles, toujours quantifiées universellement. On se place dans l'observation par test strict.

Définition 7.4. Le langage MLL_0 est le langage de formules $\forall X_1 \dots \forall X_n F$ où F est engendré par $\otimes$ et $⅋$ sur les variables X_i.

Remarquons déjà que l'interprétation de toute formule de MLL_0 est non-divergente, en effet on peut choisir pour les X_i des valeurs non dégénérées et cette propriété est conservée par tous les connecteurs. Dans le cas de l'observation par test strict, la non-dégénérescence implique en particulier l'absence de réduction infinie non acceptante. Dans la suite, on se restreint à des variables dont l'interface est de la forme $\uparrow I$, c'est-à-dire avec un seul nom positif avec une capacité d'émission ; cette convention ne constitue pas une perte de généralité, et les résultats qui suivent s'adapteraient sans peine à des variables d'interface non vide arbitraire.

Proposition 7.5. *Pour toute formule $F \in \mathrm{MLL}_0$, pour tout processus propre $P \Vdash F$, il n'existe pas de τ-réduit de P capable de faire une transition autre que τ.*

Démonstration. Notons que, puisque la règle mix est vérifiée, il suffit pour conclure de démontrer le résultat sur les formules construites uniquement avec $⅋$. On considère donc que F est une disjonction multiplicative de formules élémentaires $u : X$. Soit $P \Vdash F$, supposons qu'il existe un réduit $P \to^* P'$ où P' est capable d'émettre une action $u^{\varepsilon}(\vec{x})$, par exemple avec $\varepsilon = \uparrow$. Comme $⅋$ est associatif et commutatif on peut alors écrire $F = \forall \vec{X}\, (u : X ⅋ G)$. En instanciant X à $\mathbf{1}_{\uparrow I}$ on a donc $P \Vdash u : \mathbf{1}_{\uparrow I}, G$, or on a $B_u \Vdash u : \bot_{\downarrow \neg I}$ donc $Q = (\boldsymbol{\nu} u)(P \mid B_u)$ réalise G. Comme P a un réduit P' qui peut émettre $\bar{u}(\vec{x})$, on en déduit une réduction de Q qui mène à libérer Ω et peut donc diverger, or Q réalise G qui

est un comportement non dégénéré (puisque X est instancié par un comportement non dégénéré), ce qui est contradictoire. Aucun réduit de P ne peut donc émettre d'action. □

Cette proposition montre donc que tout processus qui réalise un type de MLL_0 est capable uniquement de faire du routage. À cause des contraintes sur l'interface des processus, le routage se fait nécessairement entre les formules élémentaires positives et négatives (entre les $X^\perp$ et les Y). Le but des propositions qui suivent est d'identifier le genre de connexion effectivement admissible.

Proposition 7.6. *Pour tout P réalisant une formule de* MLL_0*, la relation $\mathcal{F}(P)$ est une bijection partielle.*

Démonstration. Une fois de plus, il suffit pour conclure de démontrer le résultat sur les formules construites uniquement avec ⅋. Considérons donc une formule F, disjonction multiplicative de formules atomiques, et soit $P \Vdash F$. D'après la proposition précédente P ne peut pas émettre d'action, on se limite donc à étudier $\mathcal{F}(P)$.

Pour tout $(u, v) \in \mathcal{F}(P)$, d'après les règles sur les interfaces, u est associé dans F à une variable $X^\perp$ et v à une variable Y. De plus on a nécessairement $X = Y$, sans quoi il serait possible d'instancier X à $\perp$ et Y à $\mathbf{1}$ de sorte qu'on ait $P \Vdash u : \mathbf{1}_{\downarrow I}, v : \mathbf{1}_{\uparrow J}, G$, par conséquent $(\boldsymbol{\nu} uv)(P \mid B_{\bar{u}} \mid B_v)$ réaliserait le comportement non dégénéré G, or $(u, v) \in \mathcal{F}(P)$ impliquerait que $B_{\bar{u}}$ et B_v puissent interagir et donc diverger.

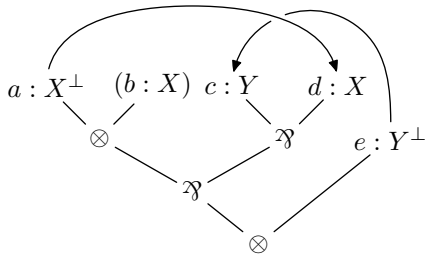
On montre à présent que la relation $\mathcal{F}(P)$ est une bijection partielle. Supposons qu'il existe trois canaux distincts u, v et w tels que (u, v) et (u, w) soient éléments de $\mathcal{F}(P)$. D'après la remarque précédente ils concernent donc la même variable propositionnelle, c'est-à-dire qu'on a $F = u : X^\perp, v : X, w : X, G$. En posant $X = [\uparrow]\perp$ on a donc $P \Vdash u : [\downarrow]\mathbf{1}, v : [\uparrow]\perp, w : [\uparrow]\perp$, or clairement $v(\vec{x}) \Vdash v : [\downarrow]\mathbf{1}$ et de même pour $w(\vec{x})$, et d'autre part $\bar{u}(\vec{x}).B_{\bar{u}} \Vdash u : [\uparrow]\perp$, donc $(\boldsymbol{\nu} uvw)(P \mid \bar{u}(\vec{x}).B_{\bar{u}} \mid v(\vec{x}) \mid w(\vec{x}))$ réalise le comportement non dégénéré G, or ce processus peut diverger en présence de la connexion $\{(u, v), (u, w)\}$, si v consomme la réception sur u puis w fait diverger $B_{\bar{u}}$. La même méthode s'applique pour prouver qu'il n'existe pas de u, v et w distincts tels qu'on ait la connexion $\{(v, u), (w, u)\}$, ainsi $\mathcal{F}(P)$ est toujours une bijection partielle. □

Les processus qui réalisent les formules de MLL_0 sont donc des processus sans divergence et donc la relation de connexion est toujours une bijection partielle. Comme cette relation est croissante au long de toute réduction, un tel processus est donc entièrement caractérisé par les connexions de ses réduits irréductibles, c'est-à-dire qu'il est équivalent à un choix non déterministe entre des compositions parallèles de routeurs.

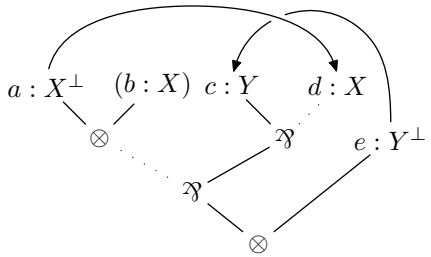
Cherchons maintenant à décrire la structure possible de ces routeurs en fonction de la formule considérée. On utilise pour cela des critères de correction pour les réseaux de preuves multiplicatifs. Les réseaux considérés sont affines, c'est-à-dire que toute formule peut y être introduite par affaiblissement. On pourra se référer à [23, 37, 38] pour la définition précise des réseaux de preuve.

Définition 7.7. Soit F une formule de MLL_0 et soit $P \Vdash F$. La structure de preuve pour F associée à P est la structure de preuve affine sans coupure de conclusion F et dont les axiomes sont données par $\mathcal{F}(P)$. Les variables qui ne sont pas reliées à un axiome sont introduites par affaiblissement.

Exemple 7.8. Considérons la formule $F = \forall XY\,((X^\perp \otimes X) \parr (Y \parr X)) \otimes Y^\perp)$. Cette formule a cinq occurrences de variables, elle est donc d'arité 5. Supposons qu'on ait un processus $P \Vdash abcde : F$ avec $\mathcal{F}(P) = \mathrm{id} \cup \{(a,d),(e,c)\}$, par exemple $P = a{\to}d \mid e{\to}c$, la structure de preuve induite est alors la suivante, qui n'est pas un réseau de preuve correct de MLL :



L'idée est de démontrer que P ne peut pas réaliser F, en utilisant le fait que cette structure n'est pas un réseau de preuve. Dans les observations considérées la règle mix est valide, donc le critère de correction se limite à un argument d'acyclicité : pour tout switching, la structure doit être acyclique [23, 4]. Dans l'exemple ci-dessus, la condition n'est pas satisfaite à cause du cycle suivant :



La présence de ce cycle permet d'exhiber un test qui réalise $F^\perp$ mais face auquel l'interaction ne peut pas mener à un état acceptant. Pour cela on instancie X et Y à la valeur $[\uparrow]\mathbf{1}$ et on construit le terme suivant (on garde implicite la sorte des constantes) :

$$\dfrac{\dfrac{\dfrac{}{b{\to}a \vdash a : [\uparrow]\mathbf{1}, b : [\downarrow]\perp} \quad \dfrac{\dfrac{}{* \vdash x : \perp}}{c(x).* \vdash c : [\downarrow]\perp} \quad \dfrac{\dfrac{}{1 \vdash x : \mathbf{1}}}{\bar{d}(x) \vdash d : [\uparrow]\mathbf{1}}}{b{\to}a \vdash ab : [\uparrow]\mathbf{1} \parr [\downarrow]\perp \quad c(x).* \mid \bar{d}(x) \vdash cd : [\downarrow]\perp \otimes [\uparrow]\mathbf{1}} \quad \dfrac{}{1 \vdash x : \mathbf{1}}}{b{\to}a \mid c(x).* \mid \bar{d}(x) \vdash abcd : ([\uparrow]\mathbf{1} \parr [\downarrow]\perp) \otimes ([\downarrow]\perp \otimes [\uparrow]\mathbf{1}), x : \mathbf{1}}}{Q := \bar{e}(x).(b{\to}a \mid c(x).* \mid \bar{d}(x)) \vdash abcd : ([\uparrow]\mathbf{1} \parr [\downarrow]\perp) \otimes ([\downarrow]\perp \otimes [\uparrow]\mathbf{1}), e : [\uparrow]\mathbf{1}}$$

Donc on a $Q \Vdash F^\perp$, et comme P réalise F par hypothèse on a $P \cdot Q \in \perp$, c'est-à-dire que $P \mid Q$ doit atteindre un état acceptant. Or il est facile de constater que $P \mid Q$ est bloqué : comme P est un terme propre, la seule façon d'atteindre l'acceptation est de faire agir Q, donc l'interaction doit commencer par $\bar{e}(x)$. Comme le seul effet de P est de fournir le routeur $e{\to}c$, la seule action susceptible de consommer $\bar{e}(x)$ est $c(x)$, qui est justement préfixée par $\bar{e}(x)$.

Le processus Q est donc un contre-exemple qui montre que P ne réalise pas la formule F. Remarquons que dans toute structure de preuve de type F le même

cycle apparaît, donc Q sert en fait de contre-exemple à tout processus propre susceptible de réaliser F. En conséquence, F n'est pas proprement réalisable.

Comme l'illustre cet exemple, c'est la présence du tenseur comme connecteur principal de la formule considérée qui permet d'obtenir une situation de blocage. En effet, le type dual a un $\parr$ comme connecteur principal, ce qui permet d'introduire le préfixage qui bloque la réduction. Il est possible de définir ainsi une méthode générale pour construire un contre-exemple comme ci-dessus, dès qu'il existe un switching pour lequel chaque composante connexe de la structure de preuve est cyclique.

Conjecture 7.9. *Dans les observations angélique et équitable, une formule de* MLL_0 *est réalisable si et seulement si elle est démontrable en logique affine.*

Le fait que chaque formule démontrable soit réalisable est évident puisque toutes les règles de déduction la logique affine sont valides dans les observations considérées. Pour la réciproque, il serait nécessaire d'établir un critère de correction pour les structures de preuve affines et de démontrer que pour toute structure mettant en défaut ce critère il est possible de construire un contre-exemple selon la technique précédente.

7.2 Contrôle et concurrence

Dans le chapitre 6, on a établi une traduction du λc-calcul dans le π-calcul polarisé qui respecte la réduction, le typage et la réalisabilité. Ceci donne un moyen puissant d'étendre le calcul séquentiel : tout processus P qui réalise la traduction A^* d'un certain type classique A peut être considéré comme une nouvelle instruction, avec des règles d'évaluation qui peuvent être concurrentes, et le typage garantit que les programmes typés qui utilisent cette instruction seront valides.

On illustre ici cette idée au moyen d'un λ-calcul enrichi d'un type de base, sur lequel on synthétisera des combinateurs concurrents. Pour permettre l'écriture de combinateurs réellement concurrents, on utilise un extension du calcul avec une application linéaire, à la manière du λ-calcul avec multiplicités de Boudol et Laneve [15].

7.2.1 Linéarité en λ-calcul

Le calcul que l'on définit ici est une variante du λc-calcul, tel que le définit la traduction LKT, sans opérateur de contrôle mais muni d'un type de base booléen.

Définition 7.10. Étant donné un ensemble dénombrable de variables de terme (notées $x, y \ldots$) et de variables de pile (notées ρ), l'ensemble des termes et piles du λb-calcul est engendré par la grammaire suivante :

$$t, u ::= x \mid *t \mid \lambda x.t \mid t\,u \mid t\,u^1 \mid 0 \mid (t\ ?\ u : v) \mid \mathbf{t} \mid \mathbf{f}$$
$$\pi ::= \rho \mid t\,\pi \mid t^1\,\pi \mid \mathrm{test}(t, u)\,\pi \mid \pi\,\{x{=}t^1\}$$

Les piles sont considérées modulo les équivalences $t\,(\pi\{x{=}u^1\}) \equiv (t\,\pi)\{x{=}u^1\}$ et $\pi\{x{=}t^1\}\{y{=}u^1\} = \pi\{y{=}u^1\}\{x{=}t^1\}$. La sémantique opérationnelle est donnée en table 7.1 page suivante.

β-réduction :

push	$(t\,u)\,\pi \equiv (t)\,u\,\pi$
pop	$(\lambda x.t)\,u\,\pi \to (t[*u/x])\,\pi$
use	$(*t)\,\pi \to (t)\,\pi$

β-réduction linéaire :

push1	$(t\,u^1)\,\pi \equiv (t)\,u^1\,\pi$
pop1	$(\lambda x.t)\,u^1\,\pi \to (t)\,\pi\,\{x{=}u^1\}$
use1	$(x)\,\pi\,\{x{=}t^1\} \to (t)\,\pi[0/x]$

calcul booléen :

push-if	$((t\ ?\ u : v))\,\pi \equiv (t)\,\mathrm{test}(u,v)\,\pi$
if-true	$(\mathbf{t})\,\mathrm{test}(t,u)\,\pi \to (*t)\,\pi$
if-false	$(\mathbf{f})\,\mathrm{test}(t,u)\,\pi \to (*u)\,\pi$

TAB. 7.1 – Sémantique opérationnelle de λb.

La constante 0 représente une occurrence d'un variable qui n'a pas été affectée par manque de ressource, sont arrivée en tête lors de la réduction provoque un blocage. Par exemple :

$$(\lambda f.f(fx))I^1\rho \to (f)(fx)\rho\{f{=}I^1\} \to (I)(0x)\rho \to (0)x\rho$$

Comme système de types pour ce calcul, on emploie les types simples du λ-calcul avec un type de base Bool, enrichis avec une implication linéaire $A \multimap B$ pour désigner les termes qui utilisent au plus une fois leur argument. On ne se préoccupe pas ici de polymorphisme.

Définition 7.11. Étant donné un ensemble dénombrable de variables propositionnelles (notées $X, Y \ldots$), le langage des types du λb-calcul est engendré par la grammaire

$$A, B ::= X \mid \mathrm{Bool} \mid A \to B \mid A \multimap B$$

Un jugement de typage est de la forme $\Gamma; \Delta \vdash t : A$ où Γ et Δ sont des ensembles d'hypothèses de la forme $x : A$, avec des x tous distincts. Les règles de typage sont données en table 7.2 page ci-contre.

Dans un jugement $\Gamma; \Delta \vdash t : A$, la partie Γ représente donc les variables utilisées un nombre arbitraire de fois, alors que Δ représente les variables utilisées linéairement. Notons qu'on se place dans un cadre affine car la règle d'axiome autorise un contexte arbitraire, même pour les variables linéaires.

Définition 7.12. La traduction en π-calcul polarisé des termes et piles du λb-calcul est donnée en table 7.3 page suivante.

Proposition 7.13. *Tout exécutable est bisimilaire à sa traduction.*

Axiome, déréliction :

$$\frac{}{\Gamma;\Delta,x:A\vdash x:A} \qquad \frac{\Gamma;x:A,\Delta\vdash t:B}{\Gamma,x:A;\Delta\vdash t:B}$$

Abstraction :

$$\frac{\Gamma;\Delta,x:A\vdash t:B}{\Gamma;\Delta\vdash \lambda x.t:A\multimap B} \qquad \frac{\Gamma,x:A;\Delta\vdash t:B}{\Gamma;\Delta\vdash \lambda x.t:A\to B}$$

Application :

$$\frac{\Gamma;\Delta_1\vdash t:A\multimap B \qquad \Gamma;\Delta_2\vdash u:A}{\Gamma;\Delta_1,\Delta_2\vdash t\,u^1:B} \qquad \frac{\Gamma;\Delta\vdash t:A\to B \qquad \Gamma;\vdash u:A}{\Gamma;\Delta\vdash t\,u:B}$$

Calcul booléen :

$$\frac{\dfrac{}{\Gamma;\vdash \mathbf{t}:\mathrm{Bool}} \qquad \dfrac{}{\Gamma;\vdash \mathbf{f}:\mathrm{Bool}} \qquad \Gamma;\Delta_1\vdash t:\mathrm{Bool} \qquad \Gamma;\Delta_2\vdash u:A \qquad \Gamma;\Delta_2\vdash v:A}{\Gamma;\Delta_1,\Delta_2\vdash (t\ ?\ u:v):A}$$

TAB. 7.2 – Typage du λb-calcul.

termes

$$\begin{aligned}
[\![x]\!]_\alpha &:= \bar{x}\langle\alpha\rangle\\
[\![*t]\!]_\alpha &:= (\boldsymbol{\nu} x)(\bar{x}\langle\alpha\rangle \mid [x=t^1])\\
[\![\lambda x.t]\!]_\alpha &:= \bar{\alpha}(x\beta).[\![t]\!]_\beta\\
[\![t\,u]\!]_\alpha &:= (\boldsymbol{\nu}\beta)([\![t]\!]_\beta \mid [\![u\,\alpha]\!]_\beta)\\
[\![t\,u^1]\!]_\alpha &:= (\boldsymbol{\nu}\beta)([\![t]\!]_\beta \mid [\![u^1\,\alpha]\!]_\beta)\\
[\![0]\!]_\alpha &:= 1\\
[\![\mathbf{t}]\!]_\alpha &:= \bar{\alpha}(xy).\bar{x}\\
[\![\mathbf{f}]\!]_\alpha &:= \bar{\alpha}(xy).\bar{y}\\
[\![(t\ ?\ u:v)]\!]_\alpha &:= (\boldsymbol{\nu}\beta)([\![t]\!]_\beta \mid [\![\mathrm{test}(u,v)\,\alpha]\!]_\beta)
\end{aligned}$$

piles

$$\begin{aligned}
[\![\rho]\!]_\alpha &:= \alpha{\to}\rho\\
[\![t\,\pi]\!]_\alpha &:= \alpha(x\beta).([x=t] \mid [\![\pi]\!]_\beta)\\
[\![t^1\,\pi]\!]_\alpha &:= \alpha(x\beta).([x=t^1] \mid [\![\pi]\!]_\beta)\\
[\![\mathrm{test}(t,u)\,\pi]\!]_\alpha &:= (\boldsymbol{\nu}\beta)(\alpha(xy).(x.[\![t]\!]_\beta + y.[\![u]\!]_\beta) \mid [\![\pi]\!]_\beta)
\end{aligned}$$

$$\begin{aligned}
[\![\pi\,\{x{=}t^1\}]\!] &:= [\![\pi]\!]_\alpha \mid [x=t^1]\\
[x=t] &:= !x(\alpha).[\![t]\!]_\alpha\\
[x=t^1] &:= x(\alpha).[\![t]\!]_\alpha
\end{aligned}$$

exécutables

$$[\![(t)\pi]\!] := (\boldsymbol{\nu}\alpha)([\![t]\!]_\alpha \mid [\![\pi]\!]_\alpha)$$

TAB. 7.3 – Traduction du λb-calcul en π-calcul polarisé.

Démonstration. Les équivalences sur les piles et les exécutables correspondent à des congruences structurelles dans les traductions, il suffit donc de considérer les exécutables de la forme $(t)\pi\, e$ où t n'est pas une application et e est un ensemble d'affectations.

Les règles pop et use sont les mêmes que pour le λc-calcul (voir théorème 6.11 page 111), et la règle pop1 est évidente. Pour la règle use1, remarquons que l'unique transition étiquetée $x(\beta)$ de $[\![\pi\{x{=}t^1\}]\!]_\alpha$ mène à $[\![\pi]\!]_\alpha | [\![t]\!]_\beta$, donc l'unique réduction de $[\![(x)\pi\{x{=}t^1\}]\!]$ mène à $(\boldsymbol{\nu}x)[\![(t)\pi]\!]$. Toutes les occurrences de x sont alors bloquées, donc le processus $(\boldsymbol{\nu}x)[\![(t)\pi]\!]$ est bisimilaire à celui dans lequel chaque $\bar{x}\langle\alpha\rangle$ est remplacé par 1, c'est-à-dire $[\![(t)\pi[0/x]]\!]$.

Dans le cas des booléens, considérons le cas $(\mathbf{t})\mathrm{test}(t,u)\pi$. La traduction s'écrit $(\boldsymbol{\nu}\alpha\beta)(\bar{\alpha}(xy).\bar{x} \mid \alpha(xy).(x.[\![t]\!]_\beta + y.[\![u]\!]_\beta) \mid [\![\pi]\!]_\beta)$ dont l'unique réduction mène à $(\boldsymbol{\nu}xy\beta)(\bar{x} \mid x.[\![t]\!]_\beta + y.[\![u]\!]_\beta \mid [\![\pi]\!]_\beta)$ qui est bisimilaire à $(\boldsymbol{\nu}\beta)(\tau.[\![t]\!]_\beta \mid [\![\pi]\!]_\beta)$ donc à $[\![(*t)\pi]\!]$. Le cas du booléen $\mathbf{f}$ est similaire. □

Proposition 7.14. *La traduction $(A)^*$ des types du λb-calcul est définie inductivement comme*

$$(X)^* := X \qquad (A \to B)^* := !{\uparrow}A^* \multimap {\uparrow}B^*$$
$$(\mathrm{Bool})^* := {\uparrow}\mathbf{1}_\emptyset \oplus {\uparrow}\mathbf{1}_\emptyset \qquad (A \multimap B)^* := {\downarrow}{\uparrow}A^* \multimap {\uparrow}B^*$$

Pour tout terme typé $x_1 : A_1 \dots x_n : A_n; y_1 : B_1, \dots, y_k : B_k \vdash t : C$ on a

$$[\![t]\!]_\alpha \vdash x_1 : ?{\downarrow}(A_1^*)^\perp, \dots, x_n : ?{\downarrow}(A_n^*)^\perp, y_1 : {\uparrow}{\downarrow}(B_1^*)^\perp, \dots, y_k : {\uparrow}{\downarrow}(k_n^*)^\perp, \alpha : {\uparrow}C^*$$

dans le système de type des processus augmenté de la règle d'affaiblissement sur les formules εA.

Démonstration. La traduction est une simple extension de la traduction LKT en version intuitionniste. La seule utilisation de la règle de promotion se trouve dans la traduction de l'application non linéaire, c'est pourquoi seule cette règle impose un contexte linéaire vide. □

L'utilisation de l'affaiblissement sur les formules de la forme εA correspond à la possibilité d'affaiblissement pour les variables linéaires. Cette possibilité n'est pas cruciale pour la définition du calcul mais on l'utilisera dans la suite. Notons que cet affaiblissement est valide dans les observations par test, y compris pour le test strict :

Lemme 7.15. *Dans l'observation stricte, la règle suivante est valide :*

$$\frac{\Phi \Rightarrow P \vdash \Gamma}{\Phi \Rightarrow P \vdash \Gamma, u : \varepsilon A}$$

Démonstration. Soient un type Γ d'interface $\mathcal{I}$, une valuation de type Φ, et soit un processus $P \Vdash \Gamma$. Par définition, pour tout processus $Q \Vdash \Gamma^\perp$ on a $(\boldsymbol{\nu}\mathcal{I})(P \mid Q) \in \perp$. Quel que soit $R \Vdash A^\perp$, le processus $(\boldsymbol{\nu}\mathcal{I}, u)(P \mid Q \mid \bar{u}^\varepsilon(\vec{x}).R)$ ne contient aucune interaction sur u, puisque ce nom n'apparaît pas dans l'interface de P ni celle de Q, donc ce processus est bisimilaire à $(\boldsymbol{\nu}\mathcal{I})(P \mid Q)$, et il est donc élément de $\perp$ puisque cet ensemble est clos par bisimulation. On a donc $P \perp Q \mid \bar{u}^\varepsilon(\vec{x}).R$, et donc $P \Vdash \Gamma, u : \varepsilon A$. □

Corollaire 7.16. *Soit $\vdash t : A$ un terme typé du λb-calcul. Dans les observations par test, on a $[\![t]\!]_\alpha \Vdash \alpha : A^*$.*

Corollaire 7.17. *Soit un terme typé* $\vdash t : \mathrm{Bool}$ *et soit* ρ *une constante de pile. Alors* $(t)\rho$ *se réduit en* $(\mathbf{t})\rho$ *ou* $(\mathbf{f})\rho$.

Démonstration. On se place dans l'observation par test strict. Par le corollaire précédent on a $[\![t]\!]_\rho \Vdash \rho : \uparrow(\uparrow\mathbf{1} \oplus \uparrow\mathbf{1})$, donc $[\![t]\!]_\rho \perp \rho(xy).(x.* + y.*)$. Ceci signifie que $[\![t]\!]_\rho$, qui est déterministe, se réduit en un terme capable d'émettre un action $\bar{\rho}(xy)$. Comme les exécutables sont bisimilaires à leur traduction, $(t)\rho$ se réduit donc en un $(t')\rho$ où t' est un terme tel que $[\![t']\!]_\rho$ émet une action $\bar{\rho}(xy)$. D'après la traduction de la table 7.3, t' est donc une constante ou un abstraction, or il ne peut s'agir d'une abstraction en raison de l'arité des canaux x et y imposée par le test $x.* + y.*$. □

7.2.2 Linéarité et parallélisme

La traduction de l'implication $(A \to B)^* = !\uparrow A^* \multimap \uparrow B^*$ entraîne que, dans la traduction d'un terme $t\,u$, l'argument u devienne un processus répliqué, de la forme $!x(\alpha).[\![u]\!]_\alpha$, donc sans réduction interne. C'est cette restriction qui impose que les traductions de λ-termes soient déterministes est sans réduction à droite d'une application.

Dans le cas d'une application linéaire $t\,u^1$, cette contrainte n'est plus nécessaire. Étant donnés trois canaux α, β, γ, on peut en effet définir un applicateur $Ap[\alpha\beta\gamma]$ par la dérivation suivante :

$$\dfrac{\dfrac{\dfrac{}{x\langle\beta\rangle \vdash x : \downarrow\uparrow A, \beta : \downarrow A^\perp} \qquad \dfrac{}{\delta{\to}\gamma \vdash \delta : \downarrow B^\perp, \gamma : \uparrow B}}{x\langle\beta\rangle \mid \delta{\to}\gamma \vdash x\delta : \downarrow\uparrow A \otimes \downarrow B^\perp, \beta : \downarrow A^\perp, \gamma : \uparrow B}}{Ap[\alpha\beta\gamma] := \alpha(x\delta).(x\langle\beta\rangle \mid \delta{\to}\gamma) \vdash \alpha : \downarrow(\downarrow\uparrow A \otimes \downarrow B^\perp), \beta : \downarrow A^\perp, \gamma : \uparrow B}$$

On en déduit une nouvelle traduction de l'application $t\,u^1$ en posant

$$\dfrac{P \vdash ?\Gamma, \Delta_1, \alpha : \uparrow(\uparrow\downarrow A^\perp \parr \uparrow B) \qquad Q \vdash ?\Gamma, \Delta_2, \beta : \uparrow A}{(\boldsymbol{\nu}\alpha\beta)(P \mid Q \mid Ap[\alpha\beta\gamma]) \vdash ?\Gamma, \Delta_1, \Delta_2, \gamma : \uparrow B}$$

Ainsi, en définissant $[\![t\,u^1]\!]_\gamma := (\boldsymbol{\nu}\alpha\beta)([\![t]\!]_\alpha \mid [\![u]\!]_\beta \mid Ap[\alpha\beta\gamma])$ on obtient une traduction du λb-calcul dont l'évaluation est plus libre. Elle est maintenant augmentée de la règle

$$\dfrac{u \to v}{(t)\,u^1\,\pi \to (u)\,v^1\,\pi}$$

qui autorise l'évaluation en parallèle de la fonction et de l'argument dans le cas des applications linéaires.

Grâce au parallélisme autorisé par la linéarité, il est donc naturel de chercher à implémenter des mécanismes de communication entre processus, à la manière de ceux que suggérait l'étude menée dans le chapitre 3. On s'intéresse ici au cas particulier du tiers-exclu, que l'on reformule dans une version linéarisée. Lorsque l'affaiblissement est autorisé, on peut écrire la dérivation suivante :

$$\dfrac{\dfrac{\dfrac{}{\bar{x}\langle\beta\rangle \vdash x : ?\downarrow A^\perp, \beta : \uparrow A}}{\bar{x}\langle\beta\rangle \vdash x : ?\downarrow A^\perp, v : \uparrow\mathbf{0}, \beta : \uparrow A}}{[\![k_\beta]\!]_\alpha = \bar{\alpha}(xv).\bar{x}\langle\beta\rangle \vdash \alpha : \uparrow(?\downarrow A^\perp \parr \uparrow\mathbf{0}), \beta : \uparrow A}$$

Le terme obtenu est la traduction LKT du terme k_β du λc-calcul (voir définition 6.9 page 110), il a donc le type $\alpha : \uparrow(\neg A)^*, \beta : \uparrow A$, c'est-à-dire qu'il a le

type d'un tiers-exclu. Soient alors deux λ-termes typés $\Gamma; \Delta_1, x : \neg A \vdash t : B$ et $\Gamma; \Delta_2, y : A \vdash u : B$. On a la dérivation suivante :

$$\frac{[\![t]\!]_\gamma \vdash ?\Gamma, \Delta_1, x : \uparrow\downarrow(\neg A)^{*\perp}, \gamma : \uparrow B}{(\boldsymbol{\nu} x)([\![t]\!]_\gamma \mid x\langle\alpha\rangle) \vdash ?\Gamma, \Delta_1, \alpha : (\uparrow(\neg A)^*)^\perp, \gamma : \uparrow B}$$

Et de même $(\boldsymbol{\nu}\delta)([\![u]\!]_\delta \mid y\langle\beta\rangle) \vdash ?\Gamma, \Delta_2, \beta : (\uparrow A)^\perp, \delta : \uparrow B$, d'où

$$(\boldsymbol{\nu} xy\alpha\beta)([\![t]\!]_\gamma \mid [\![u]\!]_\delta \mid x\langle\alpha\rangle \mid y\langle\beta\rangle \mid [\![k_\beta]\!]_\alpha) \vdash ?\Gamma, \Delta_1, \Delta_2, \gamma : \uparrow B, \delta : \uparrow B \qquad (7.2)$$

Le processus obtenu se comporte donc comme la composition parallèle des exécutables $(t)\gamma$ et $(u)\delta$, avec une corrélation entre les variables x et y donnée par le terme $(\boldsymbol{\nu}\alpha\beta)(x\langle\alpha\rangle \mid y\langle\beta\rangle \mid [\![k_\beta]\!]_\alpha)$. Les actions $x\langle\alpha\rangle$ et $y\langle\beta\rangle$ servent à nommer des continuations, en effet on a la réduction

$$\begin{aligned}(\boldsymbol{\nu} x)([\![(x)\pi]\!] \mid x\langle\alpha\rangle) &\equiv (\boldsymbol{\nu} x\gamma)(\bar{x}(z).z{\rightarrow}\gamma \mid [\![\pi]\!]_\gamma \mid x(\alpha').\alpha{\rightarrow}z) \\ &\to (\boldsymbol{\nu}\gamma z)(\alpha{\rightarrow}z \mid z{\rightarrow}\gamma \mid [\![\pi]\!]_\gamma) \cong [\![\pi]\!]_\alpha\end{aligned}$$

L'action de $[\![k_\beta]\!]_\alpha$ s'écrit alors $(\boldsymbol{\nu}\alpha)([\![t\,\pi]\!]_\alpha \mid [\![k_\beta]\!]_\alpha) \equiv [\![(k_\beta)t\,\pi]\!] \to\!\!\cong [\![(*t)\beta]\!]$. La continuation β est définie par l'autre processus : si $(u)\delta$ se réduit en un certain $(y)\pi$, alors on a la réduction $(\boldsymbol{\nu} y)([\![(y)\pi]\!] \mid y\langle\beta\rangle) \to [\![\pi]\!]_\beta$.

En résumé, la sémantique opérationnelle du terme k_β est définie par les deux règles suivantes :

$$(k_\beta)\,t\,\pi \to (*t)\,\beta \qquad\qquad (t)\beta \mid (\beta)\pi \equiv (t)\pi \qquad (7.3)$$

La seconde règle est formulée comme une équivalence plutôt que comme une réduction pour décrire les étapes de réduction des processus. On peut l'interpréter comme l'appel système `wait` dans les systèmes Unix : le processus $(\beta)\pi$ est bloqué jusqu'à ce que le processus β termine en renvoyant un valeur t, et l'exécution continue alors avec cette valeur de retour.

Cette façon de réaliser un tiers exclu utilise donc une continuation réifiée k_π, de la même forme que celle utilisée en λc-calcul. Pour effectivement réaliser le tiers exclu, il faudrait maintenant construire à partir de ce terme une fonction de type $(\neg A \multimap B) \multimap (A \multimap B) \multimap B$. Le problème vient du fait que le processus $[\![k_\beta]\!]_\alpha$ a deux sorties α et β, et que l'application de $f : \neg A \multimap B$ et $g : A \multimap B$ donne toujours un processus à deux sorties. Puisqu'on est dans un cadre intuitionniste, la contraction des deux sorties est impossible à typer. Or il est important de rester dans le cadre intuitionniste car le cas classique interdit l'écriture d'une application avec parallélisme comme on le fait plus haut.

7.2.3 Choix et contraction

La contraction de formules arbitraires n'est pas démontrable dans la logique des comportements, et par la méthode de la section 7.1.3 on peut montrer que la formule $\forall X\, X \parr X \multimap X$ n'est même pas réalisable. Cependant, pour certains types, il est possible d'écrire un processus particulier qui réalise la contraction. C'est le cas en particulier de la représentation des booléens.

Proposition 7.18. *Soit un type de processus* Γ *et soient* $P \Vdash \Gamma$ *et* $Q \Vdash \Gamma$. *Soit le processus* $\mathrm{Amb}(P,Q)[abcd] := a.P + b.Q + c.P + d.Q$. *Pour l'observation stricte, on a* $\mathrm{Amb}(P,Q)[abcd] \Vdash (ab : \mathrm{Bool}^*) \parr (cd : \mathrm{Bool}^*) \multimap \Gamma$.

Démonstration. Soit un processus $R \Vdash ab : \mathrm{Bool}^*, cd : \mathrm{Bool}^*$. En reprenant les notations de l'équation 7.1 page 122, on a $a.B_a + b.B_b \Vdash ab : \mathrm{Bool}^{*\perp}$ donc $(\boldsymbol{\nu} abcd)(R|a.B_a+b.B_b|c.B_c+d.B_d)$ est élément de $\perp$. Ceci entraîne que toute réduction maximale de R est acceptante ou est finie et termine dans un état R' capable d'émettre une action parmi $\bar{a}, \bar{b}, \bar{c}, \bar{d}$. Supposons qu'il existe une transition $R \to^* R' \xrightarrow{\bar{a}} R''$, alors ce même test prouve que $(\boldsymbol{\nu} abcd)(R'' \,|\, B_a \,|\, c.B_c + d.B_d)$ est élément de $\perp$, et donc R'' ne diverge pas ; en conséquence, on a $(\boldsymbol{\nu} abcd)R'' \in \mathbf{1}_\emptyset$. On peut ainsi décrire le comportement de $(\boldsymbol{\nu} abcd)(R \,|\, \mathrm{Amb}(P,Q)[abcd])$: toute réduction maximale non acceptante de ce terme contient nécessairement une interaction sur un canal parmi a, b, c, d. Si cette réduction a lieu sur a, elle s'écrit donc

$$\begin{aligned}&(\boldsymbol{\nu} abcd)(R \mid \mathrm{Amb}(P,Q)[abcd])\\ &\quad \to^* (\boldsymbol{\nu} abcd)(R' \mid \mathrm{Amb}(P,Q)[abcd]) \to (\boldsymbol{\nu} abcd)(R'' \mid P) \equiv (\boldsymbol{\nu} abcd)R'' \mid P\end{aligned}$$

Le réduit réalise donc $\mathbf{1}_\emptyset \otimes \Gamma = \Gamma$. Il en est de même si la première interaction a lieu sur b, c ou d, ainsi toute réduction maximale de $(\boldsymbol{\nu} abcd)(R|\mathrm{Amb}(P,Q)[abcd])$ atteint le comportement Γ donc ce terme lui-même réalise Γ. Par conséquent, on a bien $\mathrm{Amb}(P,Q)[abcd] \Vdash abcd : (\mathrm{Bool}^{*\perp} \otimes \mathrm{Bool}^{*\perp}), \Gamma$. □

Ce qui fait fonctionner le processus $\mathrm{Amb}(P,Q)$ est que toute l'information contenue dans un processus de type Bool^* peut être extraite par interaction. Un processus $P \Vdash ab : \mathrm{Bool}^*, cd : \mathrm{Bool}^*$ a deux sorties corrélées de type booléen, l'une sur ab et l'autre sur cd. Il est impossible de prédire quelle sortie agira la première, en revanche un action unique sur le canal a signifie que la sortie ab a renvoyé la valeur $\mathbf{t}$, ce qui permet le choix entre P et Q.

Dans le cas où P et Q réalisent le type Bool, on obtient un opérateur de contraction sur le type des booléens. Posons

$$B(0) := \mathbf{1}_{\uparrow\emptyset} \otimes \uparrow\mathbf{1}_\emptyset \qquad\qquad B(1) := \uparrow\mathbf{1}_\emptyset \otimes \mathbf{1}_{\uparrow\emptyset} \tag{7.4}$$

de sorte que $\mathrm{Bool}^* = B(0) \vee B(1)$, $\bar{x} \vdash xy : B(1)$ et $\bar{y} \vdash xy : B(0)$. Par la même technique que dans la proposition 7.18, on peut démontrer

$$\mathrm{Amb}(\bar{x},\bar{y})[abcd] \Vdash (ab : B(i)) \parr (cd : B(j)) \multimap (xy : B(i) \vee B(j)) \tag{7.5}$$

pour tous $i, j \in \{0,1\}$. C'est-à-dire que $\mathrm{Amb}(\bar{x},\bar{y})$ fait un choix non déterministe entre les deux conclusions du processus auquel il est connecté, et il répond sur xy la première valeur qu'il reçoit.

Définition 7.19. Un système S est un multi-ensemble fini d'exécutables, noté $e_1 | \cdots | e_n$ à permutation près. Les règles de réduction des exécutables s'étendent aux systèmes en posant $e \mid S \to e' \mid S$ si $e \to e'$.

On ajoute une constante de pile $\perp$ et, pour tous canaux distincts α, β et γ, un exécutable $\alpha\beta{\to}\gamma$. Ces opérateurs sont munis des règles de réductions suivantes, avec $b \in \{\mathbf{t}, \mathbf{f}\}$:

amb-L	$(b)\,\alpha \mid \alpha\beta{\to}\gamma \to (b)\gamma \mid (\beta)\perp$
amb-R	$(b)\,\beta \mid \alpha\beta{\to}\gamma \to (b)\gamma \mid (\alpha)\perp$
drop	$(b)\perp \to \emptyset$

Typage du tiers-exclu :

$$\frac{\Gamma;\Delta_1, x : \neg A \vdash t : \mathrm{Bool} \qquad \Gamma;\Delta_2, y : A \vdash u : \mathrm{Bool}}{\Gamma;\Delta_1,\Delta_2 \vdash Txy.(t,u) : \mathrm{Bool}}$$

Règles de réduction :

fork	$(Txy.(t,u))\,\pi \equiv (t[*k_\alpha/x])\beta \mid (u[*\alpha/y])\gamma \mid \beta\gamma{\to}\delta \mid (\delta)\pi$	
return	$(k_\alpha)\,t\,\pi \to (*t)\,\alpha$	
wait	$(t)\,\alpha \mid (\alpha)\,\pi \equiv (t)\,\pi$	
amb-L	$(b)\,\alpha \mid \alpha\beta{\to}\gamma \to (b)\gamma \mid (\beta)\bot$	$b \in \{\mathbf{t},\mathbf{f}\}$
amb-R	$(b)\,\beta \mid \alpha\beta{\to}\gamma \to (b)\gamma \mid (\alpha)\bot$	$b \in \{\mathbf{t},\mathbf{f}\}$
drop	$(b)\bot \to \emptyset$	$b \in \{\mathbf{t},\mathbf{f}\}$

Tab. 7.4 – Contrôle en λb-calcul.

La sémantique opérationnelle des exécutables $(\alpha)\pi$ est donnée par l'équation 7.3. Pour étendre la traduction des exécutables et des systèmes, on pose naturellement $[\![e_1 \mid e_2]\!] = [\![e_1]\!] \mid [\![e_2]\!]$ et

$$\begin{aligned}[\![\alpha\beta{\to}\gamma]\!] &:= (\boldsymbol{\nu} abcd)(\alpha\langle ab\rangle \mid \beta\langle cd\rangle \mid \mathrm{Amb}([\![\mathbf{t}]\!]_\gamma, [\![\mathbf{f}]\!]_\gamma)[abcd]) \\ [\![\bot]\!]_\alpha &:= \alpha(xy)\end{aligned}$$

La bisimilarité entre un système et sa traduction s'obtient en fait en décomposant les règles amb-L et amb-R en deux τ-transitions, on se passera ici de la formulation précise de ces transitions pour éviter d'accumuler les notations.

Grâce au contracteur $[\![\alpha\beta{\to}\gamma]\!]$, on peut à présent définir l'opérateur qui réalise le tiers exclu, lorsqu'il est appliqué à des fonctions dont le type de retour est Bool. Soient deux termes typés $; x : \neg A \vdash t : \mathrm{Bool}$ et $; y : A \vdash u : \mathrm{Bool}$, d'après la règle 7.2 on a

$$(\boldsymbol{\nu} xy\alpha\beta)([\![t]\!]'_\alpha \mid [\![u]\!]'_\beta \mid x\langle\alpha\rangle \mid y\langle\beta\rangle \mid [\![k_\beta]\!]_\alpha) \vdash \alpha' : {\uparrow}\mathrm{Bool}^*, \beta' : {\uparrow}\mathrm{Bool}^*$$

Par conséquent, on posant

$$[\![Txy.(t,u)]\!]_\gamma := (\boldsymbol{\nu} xy\alpha\beta)([\![t]\!]'_\alpha \mid [\![u]\!]'_\beta \mid x\langle\alpha\rangle \mid y\langle\beta\rangle \mid [\![k_\beta]\!]_\alpha \mid [\![\alpha'\beta'{\to}\gamma]\!]) \qquad (7.6)$$

on obtient $[\![Txy.(t,u)]\!]_\gamma \Vdash \gamma : {\uparrow}\mathrm{Bool}$, dans l'observation stricte. En somme, on obtient un opérateur de contrôle dont la sémantique opérationnelle et le typage sont résumés dans la table 7.4.

En utilisant cette construction $Txy.(t,u)$ il est possible de déduire un terme T_B de type $(\neg A \multimap B) \multimap (A \multimap B) \multimap B$ pour chaque type B de la forme $B_1 \to \cdots \to B_n \to \mathrm{Bool}$. En effet, il suffit de poser

$$T_B := \lambda fg.\lambda x_1 \ldots x_n.Tyz.(fyx_1 \ldots x_n, gzx_1 \ldots x_n) \qquad (7.7)$$

La sémantique opérationnelle de ce combinateur s'écrit alors

$$(T_B)\,t\,u\,v_1 \ldots v_n\,\pi \to (t)\,(*k_\alpha)^1\,v_1 \ldots v_n\,\beta \mid (u)\,(*\alpha)^1\,v_1 \ldots v_n\,\gamma \mid \beta\gamma{\to}\delta \mid (\delta)\pi$$

C'est-à-dire que T_B duplique explicitement la partie de la pile qui contient les arguments à appliquer à la valeur de retour de f et g.

La validité du calcul typé obtenu s'exprime par le théorème suivant :

Théorème 7.20. *Pour tout terme typé* $\vdash t : \mathrm{Bool}$*, utilisant éventuellement les règles de la table 7.4, l'exécutable* $(t)\rho$ *termine et ses réduits irréductibles sont de la forme* $(b)\rho \mid S$ *avec* $b \in \{\mathbf{t}, \mathbf{f}\}$ *et où* S *est un système irréductible.*

Démonstration. Ce théorème est une reformulation du corollaire 7.17 pour le calcul enrichi des combinateurs concurrents. L'argument est le même, la validité des règles de typage est donnée par la proposition 7.18 et les dérivations développées dans les sections précédentes. □

Chapitre 8

Réseaux concurrents

Le π-calcul est un formalisme élémentaire largement accepté comme référence parmi les calculs pour la concurrence et la mobilité, et il sert de référence à de nombreuses restrictions et extensions pour modéliser divers aspects des systèmes concurrents. Dans une démarche visant à la fois à simplifier la syntaxe et à gagner en expressivité sont apparus les calculs de fusion tels que le χ-calcul de Fu [32] et le *fusion calculus* de Parrow et Victor [69], puis le calcul de fusions explicites de Gardner et Wischik [34]. Une variation sur ce principe est étudiée en détail dans le chapitre 4 de cette thèse. L'idée de ces calculs est de rendre symétriques les opérations d'émission et de réception, en fondant la communication non plus sur la substitution de noms mais sur l'unification (ou explicitement la connexion entre canaux), ce qui fait gagner en expressivité, parfois même en simplifiant le formalisme.

La plupart des modèles de calcul concurrents et mobiles sont par nature géométriques, et même les systèmes à réduction de termes comme le π-calcul ont une forte intuition spatiale. En effet, à chaque calcul de processus est associé un système de règles structurelles de commutation et de réécriture de portées destiné à définir la notion appropriée de localité au sein des processus. Cette intuition géométrique a d'ailleurs mené à la définition de plusieurs formalismes graphiques, soit comme représentation de calculs existants comme dans le cas des π-nets [63] ou des *solo diagrams* [55], soit comme de nouveaux modèles du calcul concurrent comme dans les travaux de Milner sur les bigraphes [47].

On introduit dans ce chapitre le calcul des réseaux concurrents comme un pas de plus vers une formulation géométrique des processus concurrents. En effet, si le passage de la substitution à la fusion rend la communication très géométrique, la forme d'ordonnancement imposée dans ces calculs par le préfixage reste très syntaxique car elle est directement issue des arbres de synchronisation de CCS. Notre approche consiste à remplacer ce préfixage syntaxique par une relation d'activation (ou *enabling*) dans le style des structures d'événements [75], approche plus géométrique et plus adaptée à l'étude algébrique.

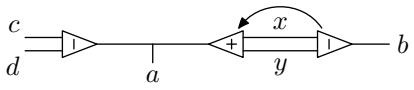
Ce chapitre se concentre sur l'étude du formalisme des réseaux concurrents et de son système de préfixage. On montre en particulier que la communication par fusion suffit à exprimer toute garde monotone, généralisant en cela le résultat principal du calcul des solos dû à Laneve et Victor [56, 57]. On montre également que l'introduction dans le calcul de sommes externes est comparable à l'introduction de gardes non monotones.

8.1 Termes et graphes

On présente les réseaux concurrents d'une part comme une algèbre de termes munie d'un système de transitions étiquetées (on appelle ce système le πg-calcul), et d'autre part comme un formalisme de réduction d'hypergraphes.

8.1.1 Exemples introductifs

Informellement, un réseau concurrent est un réseau d'émissions et de réceptions reliées par des canaux. Chaque action a un port principal (le sujet) et un certain nombre de ports auxiliaires (les objets). Deux actions peuvent interagir si elles sont reliées par leur port principal. Par exemple, le processus qui s'écrit $(\boldsymbol{\nu} xy)(\bar{a}\langle cd\rangle \mid a\langle xy\rangle.\bar{b}\langle xy\rangle)$ en πg-calcul est représenté par le réseau



Formellement, l'ensemble des canaux dans ce processus est $\{a, b, c, d, x, y\}$, parmi lesquels a, b, c, d sont considérés comme visibles, ce que l'on représente par les arêtes pendantes, alors que x et y sont privés. Les deux actions de port principal a forment une paire active dont la réduction aura pour effet d'unifier x à c et y à d. La flèche représente le fait que l'action de droite est préfixée par celle du milieu, c'est-à-dire que $\bar{b}\langle xy\rangle$ ne pourra agir qu'après l'action $a\langle xy\rangle$. Ce préfixage est étendu par conjonction et par disjonction. Considérons les deux exemples suivants :

Dans les deux cas, on a deux réceptions $a\langle x\rangle$ et $b\langle y\rangle$ et une émission $\bar{c}\langle xy\rangle$. Dans le premier cas, la flèche à deux pointes exprime que cette émission est préfixée par les deux réceptions, c'est-à-dire que l'émission n'est possible que lorsque $a\langle x\rangle$ et $b\langle y\rangle$ ont agi, mais celles-ci peuvent agir dans n'importe quel ordre. Ce mécanisme n'est pas directement exprimable en π-calcul, mais certains modèles (comme par exemple le join-calcul [30]) disposent de ce genre de synchronisation. Dans l'autre cas, les deux flèches disjointes expriment une disjonction, c'est-à-dire que l'émission est possible dès lors que l'une des deux réceptions a agi. Ce mécanisme n'est pas non plus exprimable dans les calculs existants, sauf au moyen de codages. D'autre part, notons que dans ce cas $\bar{c}\langle xy\rangle$ peut « envoyer » les noms x et y avant qu'ils n'aient été « reçus » effectivement, ce qui est un phénomène typique des calculs de fusion.

Dans la section 8.3, on étendra ce formalisme pour y inclure un mécanisme de réplication. Les gardes construites par conjonction et disjonction sur les actions élémentaires peuvent exprimer toute contrainte monotone, et on ajoutera ensuite la possibilité d'exprimer des gardes non monotones, ce qui mettra en évidence le lien avec le mécanisme de choix externe bien connu dans les algèbres de processus.

8.1.2 Termes et transitions

On présente d'abord les réseaux concurrents comme un calcul de termes qui étend le calcul des solos [57]. Il serait possible de le concevoir comme une extension du $\vec{\pi}$-calcul développé au chapitre 4, mais le lien formel entre les deux formalismes n'est pas crucial. On suppose donnés deux ensembles de noms $\mathbf{N}$ et $\mathbf{L}$ disjoints, représentant respectivement les canaux et les lieux. Les noms de lieux servent à identifier des occurrences d'actions.

Définition 8.1. Les termes du πg-calcul sont définis par la grammaire :

actions	α	$:= u^{\varepsilon}\langle\vec{x}\rangle$	
gardes	π, ρ	$:= 0 \mid 1 \mid \ell \mid \pi + \rho \mid \pi\rho$	
termes	P, Q	$:= \ell : \alpha$	action
		$\langle\pi\rangle P$	préfixage
		$1 \mid (P \mid Q)$	composition parallèle
		$(\boldsymbol{\nu} x)P \mid (\boldsymbol{\nu}\ell)P$	restriction

où u, x, x_i désignent des canaux et ℓ désigne un lieu. On note Π_g l'ensemble des termes et Υ l'ensemble des gardes.

Les termes sont considérés modulo substitution injective des noms liés. Un nom x (de canal ou de lieu) est lié par le $(\boldsymbol{\nu} x)$ environnant le plus proche. La polarité ε des actions se comprend comme dans le $\vec{\pi}$-calcul (voir définition 4.1 page 52), mais ici on considère toutes les actions $u^{\varepsilon}\langle\vec{x}\rangle$ comme non liantes. Dans cette partie, on ne considère que des termes finitaires, la réplication sera étudiée en détail dans la section 8.3.1.

Les préfixes s'interprètent comme des conditions : un processus $\langle\pi\rangle P$ est bloqué tant que la garde π n'est pas satisfaite. Une garde élémentaire ℓ représente la condition qui est fausse initialement et qui devient vraie dès qu'une action étiquetée ℓ a agi. La somme et le produit de gardes se comprennent respectivement comme une disjonction et une conjonction, et donc 0 et 1 représentent les conditions fausse et vraie respectivement.

Une localité ℓ est dite définie dans une terme P si une action $\ell : \alpha$ apparaît dans P avec ℓ non lié. On note $\mathrm{loc}(P)$ l'ensemble des localités de P, défini inductivement par

$$\begin{aligned} \mathrm{loc}(\ell : \alpha) &= \{\ell\} \\ \mathrm{loc}((\boldsymbol{\nu}\ell)P) &= \mathrm{loc}\, P \setminus \{\ell\} \\ \mathrm{loc}(P \mid Q) &= \mathrm{loc}\, P \cup \mathrm{loc}\, Q \\ \mathrm{loc}\, 1 = \mathrm{loc}\, \alpha &= \emptyset \\ \mathrm{loc}(\langle\pi\rangle P) = \mathrm{loc}((\boldsymbol{\nu} x)P) &= \mathrm{loc}\, P \end{aligned}$$

On suppose que chaque localité est définie au plus une fois dans un terme, cette restriction est une convenance notationnelle, on peut facilement voir que le système obtenu est équivalent à un système où chaque localité pourrait contenir plusieurs actions.

De façon similaire, on définit $\mathrm{FL}(P)$ comme l'ensemble des noms de lieux dont dépendent les gardes de P (il ne s'agit pas à proprement parler de l'ensemble des noms de lieux qui apparaissent libres dans les gardes car cet ensemble n'est pas

invariant par congruence structurelle ; il s'agit plutôt de l'ensemble minimal au sein d'une classe de congruence). Un terme P est dit *à préfixes clos* si $\text{FL}(P) \subseteq \text{loc}(P)$, c'est-à-dire si ses gardes ne dépendent que de ses propres actions.

Notation 8.2. On veut fréquemment écrire une action qui n'intervient dans aucune garde, dans ce cas on s'autorise à écrire α pour représenter $(\boldsymbol{\nu}\ell)(\ell : \alpha)$.

Exemple 8.3. Au moyen de gardes disjonctives, on peut exprimer une situation où plusieurs actions partagent une même continuation. Le processus

$$P := (\boldsymbol{\nu}\ell m)(\ell : u\langle\rangle \mid m : v\langle\rangle \mid \langle \ell + m \rangle Q)$$

est un cas typique qui ne peut pas s'exprimer directement dans les calculs standard : Q ne peut agir que lorsqu'au moins une des localités ℓ et m a agi. On peut par exemple coder P en un processus $P' := (\boldsymbol{\nu}s)(u\langle\rangle.\bar{s} \mid v\langle\rangle.\bar{s} \mid s.Q)$ qui a le comportement attendu : dès que $u\langle\rangle$ ou $v\langle\rangle$ agit, le signal $\bar{s}$ est lancé et Q est libéré.

Exemple 8.4. Considérons la situation plus complexe suivante :

$$\begin{aligned} P = (\boldsymbol{\nu}\ell_a\ell_b\ell_c)\big(\ & \ell_a : a_1\langle\rangle \mid \ell_b : b_1\langle\rangle \mid \ell_c : c_1\langle\rangle \\ & \mid \langle \ell_a \rangle \bar{a}_2\langle\rangle \mid \langle \ell_b \rangle \bar{b}_2\langle\rangle \mid \langle \ell_c \rangle \bar{c}_2\langle\rangle \mid \langle \ell_a\ell_b \rangle \bar{d}\langle\rangle \mid \langle \ell_a + \ell_b\ell_c \rangle \bar{e}\langle\rangle \ \big) \end{aligned}$$

Le processus P écoute sur trois ports a_1, b_1 et c_1. À une requête sur chacun de ces ports il répond respectivement sur a_2, b_2 et c_2. De plus, lorsque $a_1\langle\rangle$ et $b_1\langle\rangle$ ont agi, P envoie $\bar{d}\langle\rangle$, et lorsque soit $a_1\langle\rangle$ soit $b_1\langle\rangle$ et $c_1\langle\rangle$ on agi il envoie $\bar{e}\langle\rangle$. En π-calcul, on peut par exemple simuler ce processus de la façon suivante :

$$\begin{aligned} Q = (\boldsymbol{\nu}s_a s_{b1} s_{b2} s_c t)\big(\ & a_1\langle\rangle.\big(\bar{s}_a\langle\rangle \mid \bar{t}\langle\rangle \mid \bar{a}_2\langle\rangle\big) \\ & \mid b_1\langle\rangle.\big(\bar{s}_{b1}\langle\rangle \mid \bar{s}_{b2}\langle\rangle \mid \bar{b}_2\langle\rangle\big) \mid c_1\langle\rangle.\big(\bar{s}_c\langle\rangle \mid \bar{c}_2\langle\rangle\big) \\ & \mid s_a\langle\rangle.s_{b1}\langle\rangle.\bar{d}\langle\rangle \mid s_{b2}\langle\rangle.s_c\langle\rangle.\bar{t}\langle\rangle \mid t\langle\rangle.\bar{e}\langle\rangle \ \big) \end{aligned}$$

mais le processus obtenu est nettement plus complexe : en programmant de cette façon il faut être soigneux sur l'utilisation des signaux pour garantir que les signaux employés sont suffisants et ont le mécanisme voulu (sans possibilité de blocage, par exemple). De plus, ce codage contient trop d'information de séquentialité, par exemple l'ordre des signaux dans les préfixes $s_a\langle\rangle.s_{b1}\langle\rangle$ et $s_{b2}\langle\rangle.s_c\langle\rangle$ est indifférent, et cette dissymétrie artificielle rend moins naturelle l'étude de ce genre de processus.

Ces deux cas illustrent la situation générale où un processus a une expression simple et concise en πg-calcul et des codages techniques en π-calcul ou fusion. Néanmoins, toute forme d'ordonnancement monotone est exprimable au moyen de ce genre de codage, et l'objectif de la section 8.2 est d'en développer une démonstration précise. De ce fait le πg-calcul n'ajoute pas de pouvoir expressif par rapport à ses prédécesseurs, mais il peut plutôt être considéré comme une sorte de schéma de programmation de plus haut niveau. L'étude systématique des codages permet de plus de mieux cerner la nature de l'ordonnancement d'un point de vue sémantique.

Définition 8.5 (équivalence structurelle)**.** L'équivalence structurelle $\equiv$ sur les Π_g est la plus petite congruence qui contient l'α-équivalence et qui vérifie les axiomes de la composition et de la restriction de la table 4.1 page 54 et les axiomes supplémentaires de la table 8.1 page suivante.

Lois de monoïde commutatif sur les gardes :

$$\begin{array}{ccc} 0+\pi \equiv \pi & \pi_1+\pi_2 \equiv \pi_2+\pi_1 & \pi_1+(\pi_2+\pi_3) \equiv (\pi_1+\pi_2)+\pi_3 \\ 1\cdot\pi \equiv \pi & \pi_1\cdot\pi_2 \equiv \pi_2\cdot\pi_1 & \pi_1\cdot(\pi_2\cdot\pi_3) \equiv (\pi_1\cdot\pi_2)\cdot\pi_3 \end{array}$$

Inter-distributivité des lois sur les gardes :

$$\begin{array}{ll} 0\cdot\pi \equiv 0 & \pi_1\cdot(\pi_2+\pi_3) \equiv (\pi_1\cdot\pi_2)+(\pi_1\cdot\pi_3) \\ 1+\pi \equiv 1 & \pi_1+(\pi_2\cdot\pi_3) \equiv (\pi_1+\pi_2)\cdot(\pi_1+\pi_3) \end{array}$$

Règles de portée des noms de lieux :

$$\begin{array}{rl} (\boldsymbol{\nu}\ell)\langle\pi\rangle P \equiv \langle\pi\rangle(\boldsymbol{\nu}\ell)P & \text{préfixage (avec } \ell \notin \mathrm{FL}(\pi)) \\ P \mid (\boldsymbol{\nu}\ell)Q \equiv (\boldsymbol{\nu}\ell)(P \mid Q) & \text{extension de portée (avec } \ell \notin \operatorname{loc} P) \\ (\boldsymbol{\nu}\ell)(\boldsymbol{\nu}m)P \equiv (\boldsymbol{\nu}m)(\boldsymbol{\nu}\ell)P & \text{commutation} \end{array}$$

Action des préfixes :

$$\begin{array}{rl} \langle 1\rangle P \equiv P & \text{libération} \\ \langle\pi_1\rangle\langle\pi_2\rangle P \equiv \langle\pi_1\pi_2\rangle P & \text{composition} \\ \langle\pi\rangle(P \mid Q) \equiv \langle\pi\rangle P \mid \langle\pi\rangle Q & \text{distribution} \end{array}$$

TAB. 8.1 – Règles structurelles additionnelles du πg-calcul.

Par la suite on appellera CN (pour *Concurrent Nets*) l'ensemble des termes du πg-calcul quotienté par équivalence structurelle. La sémantique opérationnelle du πg-calcul se définit, à équivalence structurelle près, par un système de transitions étiquetées, ou LTS (pour *labelled transition system*). Le choix d'un LTS par rapport à un système par réduction est imposé par le fait que chaque interaction peut avoir des effets qui se propagent dans l'environnement. Il y a deux sortes de tels effets : les fusions de noms et les mises à jour de préfixes, en conséquence les étiquettes des transitions sont des couples. Les transitions sont de deux espèces :

$((\boldsymbol{\nu}\vec{x})\alpha, L)$ pour les actions, c'est-à-dire qu'un processus P a une transition vers P' étiquetée ainsi s'il peut émettre l'action α en ouvrant la portée des noms dans $\vec{x}$ et en consommant les actions de l'ensemble L,

(φ, L) pour les interactions, où φ est une relation d'équivalence sur $\mathbf{N}$ et L une partie de $\mathbf{L}$, c'est-à-dire que P a une transition vers P' étiquetée (φ, L) s'il peut se réduire en P' en fusionnant des canaux selon la relation φ et en consommant les actions étiquetées par L.

On notera (e, L) pour désigner une étiquette de transition de n'importe lequel de ces deux types.

Définition 8.6 (fusion, substitution)**.** Une fusion est une relation d'équivalence φ sur $\mathbf{N}$. Le *domaine* de φ est l'ensemble des noms x dont la classe d'équivalence par φ n'est pas réduite à un singleton. Pour un tel nom, on note $x \in \varphi$. On note $\{\vec{x} = \vec{y}\}$ la plus petite fusion φ par laquelle $x_i \; \varphi \; y_i$ pour chaque i (en supposant que $\vec{x}$ et $\vec{y}$ sont de même longueur). On note $\varphi \backslash \{x\}$ la fusion obtenue en oubliant

Émission et synchronisation :

$$\frac{}{\ell : \alpha \xrightarrow{\alpha,\{\ell\}} 1} \qquad \frac{P \xrightarrow{(\boldsymbol{\nu}\vec{x}_1)\bar{a}\langle\vec{y}_1\rangle,L} P' \qquad Q \xrightarrow{(\boldsymbol{\nu}\vec{x}_2)a\langle\vec{y}_2\rangle,M} Q' \qquad |\vec{y}_1| = |\vec{y}_2|}{P \mid Q \xrightarrow{\{\vec{y}_1=\vec{y}_2\}\setminus\vec{x}_1\vec{x}_2,L\cup M} (\boldsymbol{\nu}\vec{x}_1\vec{x}_2)(P' \mid Q')\sigma[1/L, M]}$$

où σ implémente $\{\vec{y}_1 = \vec{y}_2\}$ et $z \notin \vec{x}_1\vec{x}_2 \Rightarrow \sigma(z) \notin \vec{x}_1\vec{x}_2$.

Composition :

$$\frac{P \xrightarrow{e,L} P'}{P \mid Q \xrightarrow{e,L} P' \mid Q[1/L]} \qquad \frac{P \xrightarrow{e,L} P'}{(\boldsymbol{\nu}\ell)P \xrightarrow{e,L\setminus\{\ell\}} (\boldsymbol{\nu}\ell)P'}$$

Règles de portée :

$$\frac{P \xrightarrow{(\boldsymbol{\nu}\vec{x})a^\varepsilon\langle\vec{y}\rangle,L} P' \qquad z \notin a\vec{y}}{(\boldsymbol{\nu}z)P \xrightarrow{(\boldsymbol{\nu}\vec{x})a^\varepsilon\langle\vec{y}\rangle,L} (\boldsymbol{\nu}z)P'} \qquad \frac{P \xrightarrow{\varphi,L} P' \qquad z \notin \varphi}{(\boldsymbol{\nu}z)P \xrightarrow{\varphi,L} (\boldsymbol{\nu}z)P'}$$

$$\frac{P \xrightarrow{(\boldsymbol{\nu}\vec{x})a^\varepsilon\langle\vec{y}\rangle,L} P' \qquad z \in \vec{y},\ z \neq a}{(\boldsymbol{\nu}z)P \xrightarrow{(\boldsymbol{\nu}z\vec{x})a^\varepsilon\langle\vec{y}\rangle,L} P'} \qquad \frac{P \xrightarrow{\varphi,L} P' \qquad z\ \varphi\ y,\ z \neq y}{(\boldsymbol{\nu}z)P \xrightarrow{\varphi\setminus\{z\},L} P'[y/z]}$$

TAB. 8.2 – Système de transitions étiquetées du πg-calcul.

x dans φ, c'est-à-dire $\varphi \cap (\mathbf{N} \setminus \{x\})^2 \cup \{(x, x)\}$.

Une substitution de canaux est une application idempotente de $\mathbf{N}$ dans $\mathbf{N}$. Une substitution σ *implémente* une équivalence φ si pour tout couple (x, y) on a $x\ \varphi\ y$ si et seulement si $\sigma(x) = \sigma(y)$.

Définition 8.7 (sémantique opérationnelle)**.** La sémantique opérationnelle du πg-calcul est le système de transitions étiquetées engendré, à équivalence structurelle près, par les règles de la table 8.2. La notation $P\sigma$ représente l'application de la substitution de canaux σ au terme P, et $P[1/L]$ représente la substitution dans P de chaque occurrence libre d'un nom de L par 1.

Notons qu'aucune règle ne fait intervenir les préfixes. En effet, comme le système de transition est défini modulo congruence structurelle, c'est cette équivalence qui se charge de libérer les actions préfixées. Quand une action $\ell : \alpha$ est consommée par une transition, l'étiquette de la transition conserve l'information que ℓ doit être remplacé par 1 dans les préfixes. Dans le réduit, la congruence structurelle permet de simplifier les gardes et de libérer des processus au moyen de la règle $\langle 1\rangle P \equiv P$. Un système de transition sans congruence structurelle aurait aussi bien pu être défini, et la règle pour les préfixes s'écrirait alors

$$\frac{P \xrightarrow{e,L} P' \qquad \pi \equiv 1}{\langle\pi\rangle P \xrightarrow{e,L} P'}$$

La contrainte $\pi \equiv 1$ est équivalente au fait que π vaut 1 avec toutes les variables mises à 0, en particulier elle a pour effet de bloquer un processus de la forme $\langle\ell\rangle\ell : \alpha$, car c'est la situation canonique de blocage où une action attend sa propre

exécution ; en d'autres termes, le préfixe a priorité sur l'effet d'une éventuelle transition. Il n'est pas difficile de montrer l'équivalence entre les deux systèmes, la formulation à congruence structurelle près souligne le fait que la libération des préfixes s'apparente à une règle de *garbage collection*. Cette règle sera employée dans une forme généralisée dans la section 8.3.2 pour manipuler des gardes non monotones.

Définition 8.8 (bisimulation)**.** Une simulation est une relation binaire $\mathcal{S}$ sur CN telle que $P \mathrel{\mathcal{S}} Q$ implique que pour toute transition $P \xrightarrow{e,L} P'$ il existe un processus Q' tel que $Q \xrightarrow{e,L} Q'$ et $P'\sigma \mathrel{\mathcal{S}} Q'\sigma$ pour une substitution σ qui implémente l'effet de e. Une bisimulation est une simulation dont la réciproque est une simulation. Deux processus sont bisimilaires s'ils sont en relation par une bisimulation.

À cause des effets que peut produire l'environnement, la bisimilarité ainsi définie n'est pas une congruence. En effet, une transition dans le contexte peut fusionner des noms ou libérer des gardes, ce qui peut faire apparaître des transitions possibles qui ne sont pas prises en compte dans la définition de la bisimulation. Considérons par exemple les termes suivants :

$$P := \ell : u\langle x\rangle \mid \bar{u}\langle y\rangle \qquad\qquad Q := \langle \ell\rangle \bar{x} \mid y.R$$

Le terme Q n'a pas de transition, puisque la garde ℓ est bloquée et que x et y sont distincts, donc il est bisimilaire à 1. En revanche, la transition de P a deux effets : elle remplace ℓ par 1 et elle fusionne x et y, donc l'unique transition de $P \mid Q$ mène à $\bar{x} \mid x.R$, qui peut agir en libérant R, alors que $P \mid 1$ se réduit en 1 qui est inactif. Pour résoudre ce problème, on introduit la notion suivante :

Définition 8.9 (stabilité)**.** Une relation binaire $\mathcal{S}$ sur CN est *stable* si pour tout couple $P \mathrel{\mathcal{S}} Q$ en relation :
- pour toute substitution σ, $P\sigma \mathrel{\mathcal{S}} Q\sigma$,
- pour tout ensemble de lieux $L \subset \mathbf{L} \setminus (\operatorname{loc} P \cup \operatorname{loc} Q)$, $P[1/L] \mathrel{\mathcal{S}} Q[1/L]$.

La bisimilarité stable est définie comme la bisimilarité en imposant à $\mathcal{S}$ d'être stable (cette relation est appelée hyperéquivalence dans le calcul de fusion). La définition de stabilité tient compte de tous les effets de l'environnement, c'est-à-dire de toute l'information contenue dans les transitions, ce qui permet de démontrer facilement le résultat voulu :

Théorème 8.10. *La bisimilarité stable est une congruence.*

D'autre part, pour toute définition de bisimulation on peut définir une notion associée de bisimulation faible, où des transitions invisibles sont autorisées gratuitement. La notion de stabilité ci-dessus fait référence aux effets que peuvent produire les interactions, et une transition est invisible si elle ne produit pas de tel effet, en d'autres termes les τ-transitions s'obtiennent en posant $\tau = (\mathrm{id}, \emptyset)$:

Définition 8.11 (transition interne)**.** La relation de transition interne $\to$ sur CN est définie par $P \to Q$ si $P \xrightarrow{\mathrm{id},\emptyset} Q$. On note $\to^*$ la clôture réflexive et transitive de $\to$. On appelle *transition faible* la relation de transition étiquetée définie par $P \overset{e}{\twoheadrightarrow} Q$ s'il existe deux termes P' et Q' pour lesquels $P \to^* P' \xrightarrow{e} Q' \to^* Q$.

La bisimilarité faible se définit de la même façon que la bisimilarité (qualifiée dans la suite de *forte*) avec la relation de transition faible.

Dans la section 8.2 on s'intéresse aux liens entre la communication par fusion et les gardes monotones et entre les sommes externes et les gardes non monotones, du point de vue de l'expressivité. Cette étude se fait au moyen de codages qui influent sur l'arité des communications, et dans ce cadre on a recours à une notion de bisimulation barbelée, qui ne prend en compte qu'une forme un peu restreinte d'observable :

Définition 8.12 (observation)**.** Un processus P a une barbe sur (u, L) s'il existe $\vec{x}$, $\vec{y}$, ε et P' tels qu'il y ait une transition $P \xrightarrow{(\boldsymbol{\nu}\vec{x})u^{\varepsilon}\langle\vec{y}\rangle, L} P'$. On note alors $P \downarrow (u, L)$. La relation d'observation faible $\Downarrow$ est définie par $P \Downarrow (u, L)$ s'il existe un P' pour lequel $P \to^* P'$ et $P' \downarrow (u, L)$.

Définition 8.13 (bisimulation barbelée)**.** Une simulation barbelée est une relation binaire $\mathcal{S}$ sur CN telle que $P \mathcal{S} Q$ implique

- pour toute barbe (u, L), si $P \downarrow (u, L)$ alors $Q \downarrow (u, L)$;
- pour toute transition $P \xrightarrow{\varphi, L} P'$ il existe un Q' tel que $Q \xrightarrow{\varphi, L} Q'$ et une substitution σ qui implémente φ pour laquelle $P'\sigma \mathcal{S} Q'\sigma$.

Une bisimulation barbelée est une relation $\mathcal{S}$ telle que $\mathcal{S}$ et $\mathcal{S}^{-1}$ soient des simulations barbelées.

Une bisimulation faible barbelée est une bisimulation barbelée pour les transitions faibles, en remplaçant l'observation par l'observation faible. La notion de stabilité s'applique aux bisimulations barbelées comme aux bisimulations standard.

8.1.3 Graphes et réductions

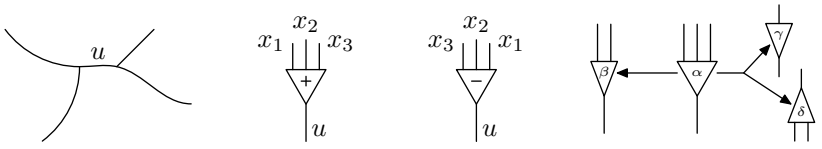
Le πg-calcul, s'il est construit sur des idées relativement naturelles, souffre d'une certaine lourdeur notationnelle, comme c'est le cas de beaucoup de calculs de processus. En particulier, il est nécessaire de munir l'algèbre des termes d'un grand nombre de règles d'équivalence structurelle afin de définir proprement les notions pertinentes de portée et de localité. Afin de réduire ce problème, on introduit maintenant une formulation graphique, au moyen de la propriété de normalisation suivante :

Proposition 8.14 (énumération)**.** *Tout terme P du πg-calcul à préfixes clos est structurellement équivalent à un terme de la forme suivante, où le produit représente une composition parallèle :*

$$P \equiv (\boldsymbol{\nu}\vec{w})(\boldsymbol{\nu}\vec{m}) \prod_{i=1}^{n} \langle\pi_i\rangle \ell_i : u_i^{\varepsilon_i}(\vec{x}_i) \quad \textit{avec} \quad \pi_i \equiv \sum_{j=1}^{p_i} \ell_{i,j,1} \cdots \ell_{i,j,q_{i,j}}$$

pour un certain $n \geqslant 0$ et des $p_i \geqslant 0$ et $q_{i,j} \geqslant 0$. Les $\ell_{i,j,k}$ sont éléments de l'ensemble $\{ \ell_i \mid 1 \leqslant i \leqslant n \}$. Les familles $\vec{w}$ et $\vec{m}$ représentent respectivement les canaux et localités privés. Une telle formulation est appelée une énumération de P.

Démonstration. Par extension de portée, tous les lieurs peuvent être rassemblés à l'extérieur du terme, et les règles de distribution et de composition des préfixes



canal u | réception $u\langle x_1 x_2 x_3\rangle$ | émission $\bar{u}\langle x_1 x_2 x_3\rangle$ | flèches d'activation $\beta \vdash \alpha\,;\, \gamma, \delta \vdash \alpha$

FIG. 8.1 – Syntaxe graphique des réseaux concurrents.

donnent l'écriture de P comme un produit d'actions élémentaires gardées. La forme canonique des gardes est obtenue par distributivité de la conjonction sur la disjonction. □

Par conséquent un processus peut être décrit comme un ensemble de localités, chacune associée à une action élémentaire et un préfixe. Les actions sont construites sur l'ensemble des noms de canaux, dont certains sont liés. Les canaux libres forment donc l'interface. Les préfixes peuvent être soit 1 soit une disjonction non vide d'ensembles de lieux. Ils définissent donc une relation de préfixage entre les ensembles non vides de lieux et les lieux. On obtient donc la définition algébrique suivante, où $\mathcal{C}^*$ représente les suites finies sur $\mathcal{C}$ et $\mathcal{P}_0(\mathcal{A})$ les parties non vides de $\mathcal{A}$.

Définition 8.15. Un réseau concurrent est la donnée de
- un ensemble $\mathcal{C}$ de canaux,
- une partie $\mathcal{I}$ de $\mathcal{C}$ appelée interface,
- un ensemble $\mathcal{A}$ d'actions étiquetées sur $\mathbf{P} \times \mathcal{C} \times \mathcal{C}^*$,
- une relation d'activation $\vdash$ entre $\mathcal{P}_0(\mathcal{A})$ et $\mathcal{A}$.

Dans la suite, en conformité avec les notations pour les termes, on désignera les canaux par les minuscules latines et les actions par des minuscules grecques. Une action positive $(\downarrow, u, \vec{x})$ sera notée $u(\vec{x})$ et une action négative $(\uparrow, u, \vec{x})$ sera notée $\bar{u}(\vec{x})$, en notant de la même façon les étiquettes et les actions (ou occurrences d'étiquettes) s'il n'y a pas d'ambiguïté. Dans de telles actions, u est le port principal et les éléments de $\vec{x}$ sont les ports auxiliaires.

La figure 8.1 illustre les conventions graphiques utilisées par la suite pour représenter les réseaux concurrents : les canaux sont les arêtes d'un hypergraphe dont les sommets sont les actions, les canaux de l'interface sont représentés avec des arêtes pendantes. Une action est représentée par un triangle avec la polarité au milieu, le port principal connecté à un sommet et les ports principaux ordonnés sur le côté opposé. Par convention, les ports auxiliaires des actions positives sont ordonnés de gauche à droite (en regardant depuis le port principal) et ceux des actions négatives de droite à gauche, ce qui permet d'avoir des figures plus propres en préservant la planarité par réduction (la formulation graphique de la réduction, figure 8.2 page suivante, en est donne illustration).

La relation d'activation est représentée au moyen de flèches : pour chaque élément $\beta_1, \ldots, \beta_n \vdash \alpha$ de cette relation, on dessine une flèche à plusieurs pointes partant de α et pointant sur chacun des β_i. Les différents éléments de $\vdash$ sont représentés par des flèches disjointes. Ces flèches représentent des gardes, donc une action n'est active que si aucune flèche n'en part, et les communications n'ont lieu qu'entre actions actives.

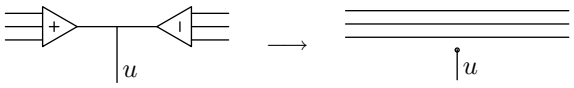


FIG. 8.2 – Réduction d'une paire active.

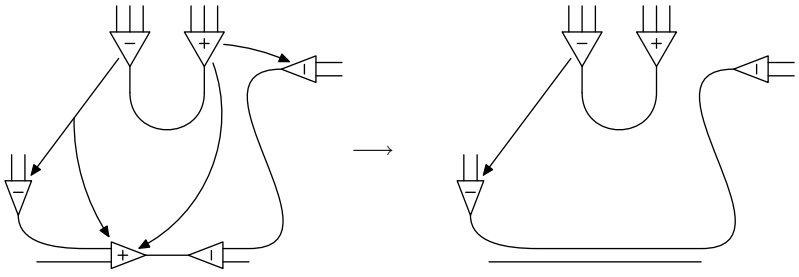


FIG. 8.3 – Réduction d'une paire active avec mise à jour des gardes.

Définition 8.16. Soit $P = (\mathcal{C}, \mathcal{I}, \mathcal{A}, \vdash)$ un réseau concurrent.
- Une action α est active s'il n'existe aucun $X \subset \mathcal{A}$ tel que $X \vdash \alpha$;
- Une paire active est une paire d'actions $\{\alpha, \beta\}$ de polarités opposées de même sujet et de même arité ;
- P se réduit selon une paire active $\{\alpha, \beta\}$ en le réseau P' obtenu en retirant α et β du quotient de P par l'équivalence $\vec{x} = \vec{y}$. S'il existe dans P une flèche $X \vdash \gamma$ avec $X \subseteq \{\alpha, \beta\}$, alors toute flèche $Y \vdash \gamma$ est retirée de P', sinon toute flèche $X \vdash \gamma$ est remplacée par $X \setminus \{\alpha, \beta\} \vdash \gamma$.

Le comportement sur les flèches illustre pourquoi la définition 8.15 impose aux $X \vdash \alpha$ d'avoir toujours X non vide : si une flèche $\emptyset \vdash \alpha$ apparaît dans la réduction, elle signifie que l'action α devient active, et donc toutes les autres contraintes d'activation sur α deviennent caduques, on supprime donc tous les $Y \vdash \alpha$, ce qui correspond à l'axiome $1 + \pi = 1$ dans les règles structurelles de la table 8.1. Cet axiome, sous forme générale, est précisément celui à ajouter dans l'équivalence de la définition suivante pour obtenir l'équivalence entre les réseaux concurrents et les termes du πg-calcul. Une approche essentiellement équivalente serait d'autoriser les flèches vides et de dire que α est actif si $\emptyset \vdash \alpha$.

Comme illustré dans la figure 8.2, la réduction consiste à supprimer les deux actions de la paire active et à connecter les canaux de leurs ports auxiliaires. Le port principal des deux actions est toujours présent dans le réseau, avec deux actions de moins. Toute flèche pointant sur l'une des deux actions est supprimée, ce qui peut éventuellement libérer d'autres actions.

Définition 8.17. La relation $\equiv$ sur les réseaux concurrents est la plus petite équivalence pour laquelle un réseau contenant une flèche $X \vdash \alpha$ est équivalent au même réseau augmenté de flèches $Y \vdash \alpha$ avec $X \subseteq Y$.

En d'autres termes, un réseau peut se simplifier en supprimant toutes les flèches $Y \vdash \alpha$ pour lesquelles il y a une flèche $X \vdash \alpha$ avec $X \subseteq Y$. Deux réseaux sont équivalents par $\equiv$ lorsque leurs formes ainsi simplifiées sont égales (ou isomorphes en tant que graphes étiquetés). À l'inverse, on peut considérer

que la relation $\vdash$ est saturée si $X \vdash \alpha$ et $X \subseteq Y$ impliquent $Y \vdash \alpha$ et que deux réseaux sont équivalents s'ils ont la même forme saturée ; notons que cette condition de régularité est exactement celle imposée sur l'activation dans les structures d'événements [75]. Cette simple condition, qui correspond en termes de gardes à $x + xy \equiv x$, suffit pour obtenir l'équivalence entre termes et réseaux.

Proposition 8.18. *Il y a isomorphisme entre les réseaux concurrents et les termes à préfixes clos du πg-calcul pour leurs réductions respectives.*

Démonstration. La propriété d'énumération de la proposition 8.14 fournit la traduction entre les deux formalismes : pour un terme P, le réseau associé $[\![P]\!]$ est $(\mathcal{C}, \mathcal{I}, \mathcal{A}, \vdash)$ où $\mathcal{C}$ est l'ensemble des noms de canaux (libres et liés), $\mathcal{I}$ est l'ensemble des noms libres, $\mathcal{A}$ l'ensemble des noms de lieux (libres et liés) $\{ \ell_i \mid 1 \leqslant i \leqslant n \}$ avec ℓ_i étiqueté $(\varepsilon_i, u_i, \vec{x}_i)$. On vérifie sans problème que cette traduction est une bijection entre les classes d'équivalence structurelle et qu'elle commute avec les transitions internes, au sens qu'on a $P \xrightarrow{\varphi, L} P'$ si et seulement si $[\![P]\!]$ se réduit en $[\![P']\!]$ avec l'équivalence φ. □

En conséquence, dans la suite, on parlera de réseaux concurrents tant pour désigner des réseaux au sens de la définition 8.15 que pour désigner des πg-termes à préfixes clos modulo congruence structurelle. On s'autorisera donc à faire référence à un même processus avec l'une ou l'autre notation, selon le contexte.

8.1.4 Fragments

Le πg-calcul est conçu essentiellement comme une extension du calcul de fusion, qui est lui-même une extension du π-calcul. Il est possible de caractériser précisément ces calculs et d'autres comme des fragments du calcul des réseaux concurrents. L'avantage de la formulation graphique des processus est qu'elle permet de formuler ces fragment par des critères très naturels sur la structure des processus.

Formellement, on voit un calcul C comme le système de transition dont les états sont les termes (à congruence structurelle près) et dont les transitions sont les réductions internes. On dit alors que C est un sous-calcul de CN si ses états s'injectent dans les réseaux de sorte que le plongement soit plein et fidèle pour les réductions. On cherchera en particulier des critères (que l'on peut qualifier de critères de correction par analogie avec l'habitude des réseaux de preuve) pour identifier l'image d'un tel plongement.

Dans cette section, on ne s'intéresse qu'aux fragments sans réplication de calculs connus. En effet, la façon d'implémenter la réplication est en général assez indépendante de la structure précise du calcul, et les critères énoncés ici s'étendraient donc au cas avec réplication à condition de la formuler de la même façon dans chaque cadre.

Restriction du préfixage

Le calcul des solos est le fragment le plus simple de CN. Son plongement consiste simplement à traduire une action α en un terme élémentaire $(\boldsymbol{\nu}\ell)(\ell : \alpha)$, selon la notation déjà employée. Son image se caractérise simplement par le fait que la relation d'activation est vide. Il est intéressant de remarquer que les *solo*

diagrams de Laneve, Parrow et Victor [55] sont très similaires à notre syntaxe, car le diagramme correspondant à un terme est précisément le graphe dual du réseau concurrent correspondant à sa traduction (les sommets du diagramme sont les canaux et les arêtes orientées sont les nœuds du réseau).

Le calcul de fusion correspond à la modification du π-calcul dans laquelle les réceptions ne sont pas des lieurs. En formulant la démarche à l'envers, il s'agit du calcul des solos enrichi de préfixage à la CCS. L'algèbre des termes (avec nos notations) est définie comme

$$P, Q := 1 \mid (P \mid Q) \mid (\boldsymbol{\nu} x)P \mid \bar{u}\langle\vec{x}\rangle.P \mid u\langle\vec{x}\rangle.P$$

Pour plonger ce calcul dans CN, définissons la traduction $[\![\cdot]\!]_\pi$ paramétrée par une garde π qui représente la garde du sous-terme considéré. On pose donc $[\![\alpha.P]\!]_\pi = (\boldsymbol{\nu}\ell)(\langle\pi\rangle\ell : \alpha \mid [\![P]\!]_\ell)$ où ℓ est un nom de lieu frais, et la traduction s'étend par commutation sur la composition et la restriction. La traduction d'un processus P est alors le réseau $[\![P]\!]_1$. On vérifie sans peine que cette traduction est bien un plongement de fusion dans CN. L'image de la traduction se caractérise par une contrainte sur la relation d'activation :

Définition 8.19. Soit P un réseau concurrent. On pose :
- le préfixe d'une action α est l'ensemble $(\_ \vdash \alpha) = \{\, X \mid X \vdash \alpha \,\}$,
- P est à préfixage simple si, pour toute action α, $(\_ \vdash \alpha)$ est soit vide soit de la forme $\{\{\beta\}\}$,
- si P est à préfixage simple, on note $\leftarrow$ la relation de préfixage telle que $\beta \leftarrow \alpha$ si $\beta \vdash \alpha$.

Proposition 8.20. *L'image de la traduction du calcul de fusion est l'ensemble des réseaux à préfixage simple dont la relation de préfixage est acyclique.*

Démonstration. Sous ces hypothèses, la relation $\leftarrow$ forme un graphe orienté acyclique, et tous les nœuds sont de degré sortant au plus 1, donc $\leftarrow$ est une forêt, qui correspond précisément à la structure syntaxique du terme. Comme les règles structurelles de composition et de portée sont les mêmes on reconstruit un terme de fusion à partir du réseau par induction sur la hauteur de cette forêt. □

Un autre restriction fréquemment employée dans la littérature est l'asynchronie. Dans le π-calcul asynchrone, seules les réceptions peuvent agir en préfixes, et cette contrainte se traduit immédiatement dans les réseaux en imposant qu'aucune action négative n'apparaisse à gauche de la relation d'activation. Notons que cette contrainte est indépendante de la condition de préfixage simple acyclique : le sens de l'asynchronie est que les émissions sont des messages envoyés sans garantie de réception (et le seul moyen de garantir cette réception est d'attendre un message de confirmation explicite). Cette absence d'information sur l'état d'une réception correspond à l'absence de flèche vers les émissions, puisque les flèches de synchronisation sont des formes de communications entre actions, comme on l'illustrera en détail dans la suite.

Restriction des gardes

Continuons de parcourir à rebours l'évolution des calculs de processus : si le *fusion calculus* est apparu comme une extension du π-calcul où seul le $(\boldsymbol{\nu} x)$

est lieur, on voit maintenant le π-calcul comme la restriction de fusion dans laquelle chaque réception est liante. Plus précisément, chaque réception doit avoir la forme $(\boldsymbol{\nu}\vec{x})u\langle\vec{x}\rangle.P$ où $\vec{x}$ est une suite de noms de canaux distincts. La traduction du π-calcul est donc un cas particulier de celle du calcul de fusion. Pour caractériser son image, il est nécessaire de reconstituer la relation de causalité entre actions imposée par cette restriction de portée :

Définition 8.21. La causalité est la relation binaire $\Leftarrow$ sur les actions d'un réseau concurrent telle qu'on ait $\beta \Leftarrow \alpha$ lorsque β est une action positive et au moins l'un des ports de α est relié à un port auxiliaire de β.

Proposition 8.22. *L'image de la traduction du π-calcul est l'ensemble des réseaux concurrents à préfixage simple dont la relation $\leftarrow$ est acyclique et tels que $\Leftarrow$ soit inclus dans la clôture transitive de $\leftarrow$.*

Démonstration. Notons que l'inclusion de $\Leftarrow$ dans la clôture de $\leftarrow$ impose que $\Leftarrow$ est acyclique et anti-réflexive. En particulier, ceci interdit la présence d'actions de la forme $\alpha = u\langle u\rangle$ car le port principal de α est relié par u à son port auxiliaire et donc $\alpha \Leftarrow \alpha$. De même cette condition garantit que les ports auxiliaires des réceptions sont distincts. L'inclusion de $\Leftarrow$ dans la clôture de $\leftarrow$ garantit ensuite que toute action dépendant de l'objet d'une réception α est préfixée, peut-être indirectement, par α, donc toutes les occurrences des objets de α se retrouvent sous son préfixe, ce qui permet de placer les restrictions aux endroits voulus lors de la traduction inverse d'un réseau en un terme. □

Dans la définition de $\Leftarrow$, l'action de gauche est supposée positive, puisque seules les réceptions ont une portée imposée en π-calcul. Si on étendait cette relation en une causalité $\Leftarrow$ entre actions de toutes polarités, on imposerait en plus aux émissions d'être liantes, ce qui mènerait exactement à capturer le calcul *private* π ou πI [73].

Un point à remarquer est que, malgré son utilisation dans la proposition précédente, la relation $\Leftarrow$ est définie sans référence au préfixage. On peut en effet formuler cette notion de communication privée indépendamment, en imposant à $\Leftarrow$ d'être anti-réflexive et acyclique. Cette contrainte associe effectivement une portée à chaque action positive, le fragment obtenu est au π-calcul ce que les réseaux concurrents sont au calcul fusion, en particulier la communication peut y être formulée simplement par substitution. Imposer la même contrainte sur $\Leftarrow$ mène une forme « privée » des réseaux concurrents où la communication se fait par α-conversion.

On a donc une caractérisation géométrique du préfixage à la CCS, une caractérisation indépendante de la communication par substitution de noms et de la communication par unification, et une façon simple d'associer les deux notions. En variant les combinaisons de contraintes, on capture diverses variantes connues sur le principe du π-calcul.

8.2 Expressivité des gardes monotones

En remplaçant le préfixage syntaxique du π-calcul par une relation d'activation dans le style des structures d'événements, il est naturel de penser qu'on a enrichi le calcul par rapport à ses prédécesseurs. Il apparaît néanmoins que les

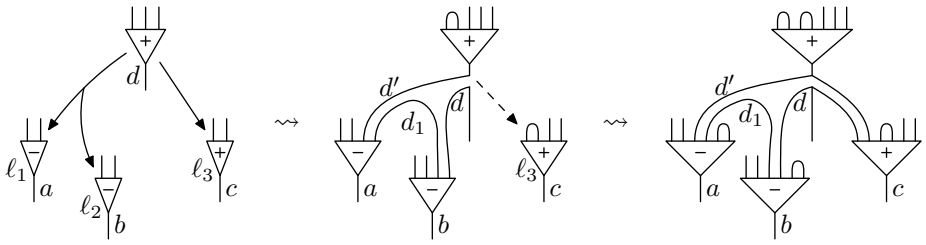


FIG. 8.4 – Exemple de traduction explosive.

gardes monotones arbitraires peuvent être codées sous forme de communications de canaux.

L'objectif des gardes est d'imposer un certain ordonnancement des actions d'un processus, c'est-à-dire en quelque sorte de restreindre sa stratégie de réduction. Une analogie peut d'ailleurs être utilement établie avec le λ-calcul : il y a plusieurs stratégies de réductions standard pour normaliser les λ-termes, mais la technique de traduction par passage de continuation [70] donne un moyen d'imposer une stratégie d'évaluation particulière, au prix d'une légère modification de l'interface du terme (car chaque terme traduit reçoit une continuation comme argument supplémentaire). L'approche employée ici est similaire, dans la mesure où le principe est d'ajouter des arguments supplémentaires aux communications afin que chaque interaction permette de mettre en contact des actions précédemment bloquées car indépendantes.

L'idée est donc d'utiliser la fusion pour transformer l'attente explicite d'une action en l'attente implicite de l'unification de noms. Une approche similaire a été employé par Laneve et Victor [56] pour coder le calcul de fusion dans les solos, par une technique très dépendante de la structure des termes. Les traductions développées ici emploient des arguments différents et plus géométriques, motivés par la présentation des processus sous forme de réseaux.

8.2.1 Traduction explosive

Dans cette section on définit une traduction des gardes qualifiée d'explosive car elle ajoute deux arguments à chaque communication pour chaque élément de la relation d'activation. L'intérêt de cette traduction est qu'elle fournit une bisimulation barbelée forte entre un processus et sa traduction, l'inconvénient étant que la traduction n'est pas compositionnelle.

Considérons l'exemple de la figure 8.4 : on part d'un processus P de la forme

$$P := \ell_1 : \bar{a}\langle\vec{x}\rangle \mid \ell_2 : \bar{b}\langle\vec{y}\rangle \mid \ell_3 : c\langle\vec{z}\rangle \mid \langle \ell_1\ell_2 + \ell_3\rangle d\langle\vec{w}\rangle \mid Q$$

où Q est l'environnement, non représenté sur la figure. L'action $d\langle\vec{w}\rangle$ est bloquée jusqu'à ce qu'agisse soit $c\langle\vec{z}\rangle$ soit $\bar{a}\langle\vec{x}\rangle$ et $b\langle\vec{y}\rangle$. On retarde donc l'action de $d\langle\vec{w}\rangle$ en remplaçant d par un canal frais d' qui sera unifié à d lorsque cette condition est remplie. On traduit la flèche $a\langle\vec{x}\rangle, \bar{b}\langle\vec{y}\rangle \vdash d\langle\vec{w}\rangle$ en posant

$$P_1 := (\boldsymbol{\nu} d' d_1)\big(\bar{a}\langle d_1 d'\vec{x}\rangle \mid \bar{b}\langle d d_1\vec{y}\rangle \mid (\boldsymbol{\nu} x)\ell_3 : c\langle x x\vec{z}\rangle \mid \langle\ell_3\rangle(\boldsymbol{\nu} x) d'\langle x x\vec{w}\rangle\big) \mid Q_1$$

où Q_1 représente Q dans lequel toute action $\ell : u^\varepsilon\langle\vec{v}\rangle$ est remplacée par $(\boldsymbol{\nu}x)\ell : u^\varepsilon\langle xx\vec{v}\rangle$, afin que toute action puisse effectuer la fusion voulue. La traduction de la flèche $c\langle\vec{z}\rangle \vdash d\langle\vec{w}\rangle$ donne ensuite

$$\begin{aligned} P_2 := (\boldsymbol{\nu}d'd_1)\big(\ & (\boldsymbol{\nu}y)\bar{a}\langle yyd_1d'\vec{x}\rangle \mid (\boldsymbol{\nu}x)\bar{b}\langle xxdd_1\vec{y}\rangle \\ & \mid (\boldsymbol{\nu}x)c\langle dd'xx\vec{z}\rangle \mid (\boldsymbol{\nu}xy)d'\langle xxyy\vec{w}\rangle\big) \mid Q_2 \end{aligned}$$

où Q_2 est obtenu en transformant Q_1 comme on a transformé Q à la première étape. Il apparaît clairement que d et d' seront unifiés exactement quand la garde qu'avait l'action avant traduction sera validée.

Définition 8.23. Soit P un réseau concurrent à préfixes clos. Supposons donnée une énumération de P (au sens de la proposition 8.14). Pour chaque i, posons $u'_i = u_i$ si $\pi_i \equiv 1$ et soit u'_i un nom frais sinon. Soit $(u_{i,j,k})$ une famille de noms telle que $u_{i,j,0} = u_i$ et $u_{i,j,q_{i,j}} = u'_i$ et où tous les autres $u_{i,j,k}$ sont frais. La traduction de P pour cette énumération est

$$(\boldsymbol{\nu}\vec{w}')\prod_{I=1}^{n}(\boldsymbol{\nu}\vec{z}_I)u'_I\langle\vec{y}_I\vec{x}_I\rangle \quad \text{où} \quad y_{I,\langle i,j,\eta\rangle} = \begin{cases} u_{i,j,k-\eta} & \text{si } \exists k,\ \ell_I = \ell_{i,j,k} \\ z_{I,\langle i,j\rangle} & \text{sinon} \end{cases}$$
$$\vec{w}' = \vec{w} \cup \{u_{i,j,k} \mid k \neq 0\}$$

où $\vec{z}_I$ est une famille de noms frais indexée sur $\{\,(i,j) \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant p_i\,\}$ et $\vec{y}_I$ est une famille indexée sur $\{\,(i,j,\eta) \mid 1 \leqslant i \leqslant n, 1 \leqslant j \leqslant p, \eta \in \{1,0\}\,\}$. La traduction explosive de P est l'ensemble $[\![P]\!]_e$ des traductions explosives pour toutes les énumérations possibles (on comprend la traduction comme une relation plutôt que comme une fonction).

Notons que, dans cette définition, les arguments supplémentaires $\vec{y}_I$ sont indexés par un ensemble de triplets, sans spécifier d'ordre précis. Il est entendu que cet ordre est sans importance, seul importe le fait que le même ordre soit utilisé pour toutes les actions d'un processus donné.

Théorème 8.24. *Tout processus P est fortement et stablement bisimilaire par barbelés à tout $Q \in [\![P]\!]_e$.*

Démonstration. L'idée est que la traduction commute avec la réduction, modulo l'introduction de paires d'arguments inutiles. La démonstration est technique mais pas spécialement difficile, on pourra la trouver dans [8]. □

8.2.2 Traduction par duos

On va maintenant définir une traduction sur le même principe mais en résolvant le problème de la compositionnalité de la traduction. Pour ce faire, on procède en deux étapes, illustrées par les figures 8.5 page suivante et 8.6 page 153.

La première étape (qualifiée de traduction en duos) consiste à transformer un processus muni d'un schéma de préfixage monotone quelconque en un processus où chaque action est au plus préfixe d'une paire active chargée d'unifier des noms. Le processus obtenu, élément d'un fragment du calcul de fusion (on parle de *duos* dans la terminologie de Parrow [68]), est faiblement bisimilaire au processus initial.

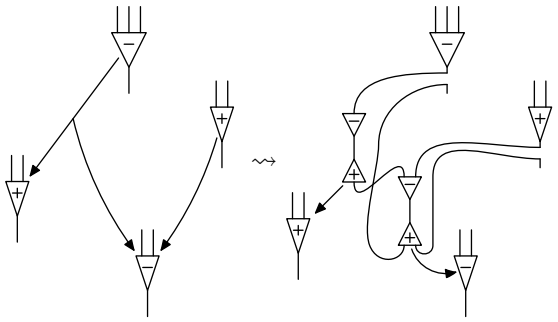


FIG. 8.5 – Exemple de traduction en duos.

La deuxième étape transforme ce processus à préfixage plus simple en un processus sans aucun préfixage explicite, par une méthode analogue à celle de la section précédente, en ajoutant à chaque communication des données chargées de libérer les paires introduites dans la première étape. Cette deuxième étape est donc une bisimulation barbelée forte.

Définition 8.25 (Traduction en duos)**.** Soit P un réseau concurrent à préfixes clos. Supposons choisie une énumération de P de la forme

$$P = \prod_{i=1}^{n} \langle \pi_i \rangle \ell_i : u_i \langle \vec{x}_i \rangle \quad \text{avec} \quad \pi_i = 1 \text{ ou } \pi_i = \sum_{j=1}^{p_i} \ell_{i,j,1} \cdots \ell_{i,j,q_{i,j}}$$

Pour chaque i, soit u'_i le canal u_i si $\pi_i \equiv 1$ ou un nom frais sinon. Soit $u_{i,j,k}$ une famille de noms de canaux telle que $u_{i,j,0} = u_i$ et $u_{i,j,q_{i,j}} = u'_i$, et où tous les autres éléments sont frais. Le codage de P pour cette énumération est

$$(\boldsymbol{\nu}\vec{w}') \prod_{I=1}^{n} (\boldsymbol{\nu} v)\big(\ell_I : u'^{\varepsilon_I}_I \langle \vec{x}_I \rangle \mid \langle \ell_I \rangle v \langle \vec{y}_I \rangle \mid \bar{v} \langle \vec{z}_I \rangle\big) \quad \text{avec} \quad \vec{w}' = \vec{w} \cup \{u_{i,j,k} \mid k \neq 0\}$$

où $\vec{y}_I$ et $\vec{z}_I$ sont les familles de noms de canaux indexées sur $\{(i,j,k) \,|\, \ell_{i,j,k} = \ell_I\}$ définies par

$$y_{I,(i,j,k)} = u_{i,j,k-1} \qquad \text{et} \qquad z_{I,(i,j,k)} = u_{i,j,k}$$

La traduction en duos est la relation qui associe P à l'ensemble $[\![P]\!]_d$ de ses traductions pour toutes ses énumérations possibles.

Théorème 8.26. *Tout processus P est faiblement et stablement bisimilaire à tout processus $Q \in [\![P]\!]_d$.*

Démonstration. L'argument principal de la démonstration de bisimulation est qu'il est possible de réduire les processus unificateurs $v\langle \vec{y}_I \rangle \mid \bar{v}\langle \vec{z}_I \rangle$ à tout moment, car ils ne peuvent pas interférer avec le reste des communications. Le développement technique détaillé se trouve dans [8]. □

Remarquons déjà que cette traduction commute avec la composition parallèle, au sens que si P et Q sont deux processus à préfixes clos, alors on a $[\![P]\!]_d \,|\, [\![Q]\!]_d \subseteq [\![P \,|\, Q]\!]_d$, ce qui n'était pas le cas pour la traduction explosive. De plus, les traduction ont une forme de préfixage extrêmement simple, qui rentre en particulier dans la définition suivante :

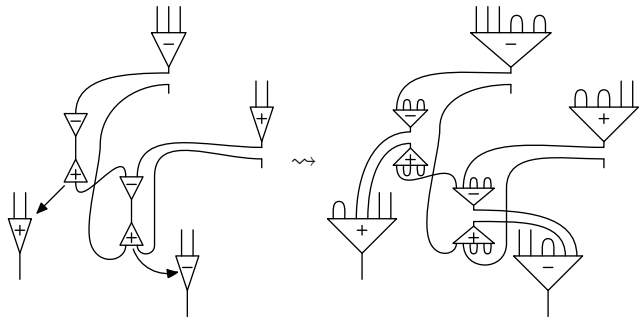


FIG. 8.6 – Exemple de codage des duos.

Définition 8.27. Un processus P est à préfixage linéaire s'il s'écrit

$$P \equiv (\boldsymbol{\nu}\vec{w}) \prod_{i=1}^{n} \langle \pi_i \rangle \ell_i : u_i^{\varepsilon_i} \langle \vec{x}_i \rangle \quad \text{avec} \quad \pi_i = \begin{cases} \ell_{p(i)} & \text{si } i \in \mathrm{dom}(p) \\ 1 & \text{sinon} \end{cases}$$

où p est une injection partielle sur $\{1, \ldots, n\}$.

Lemme 8.28. *Pour tout processus P, tout $Q \in [\![P]\!]_d$ est à préfixage linéaire.*

Le sens de ce préfixage « linéaire » est que chaque action agit comme garde d'au plus une autre action, donc chaque interaction doit libérer au plus deux actions. En conséquence, il est suffisant d'ajouter exactement quatre arguments à chaque action, deux pour chaque polarité, et la structure du préfixage garantit qu'il ne pourra pas y avoir d'interférence. Mise à part cette simplification, la traduction est essentiellement la même que celle de la définition 8.23.

Définition 8.29 (traduction linéaire)**.** Soit P un processus à préfixage linéaire. Supposons donnée une énumération de P selon les notations de la définition 8.27. Soit (u'_i) une famille de noms frais indexée sur $\mathrm{dom}(p)$. La traduction linéaire de P pour cette énumération est

$$(\boldsymbol{\nu}\vec{w}\vec{u}') \prod_{i=1}^{n} (\boldsymbol{\nu} yz) \ell_i : \alpha_i$$

où

$$\alpha_i = \begin{cases} u^{\varepsilon_i}(yyzz\vec{x}_i) & \text{si } i \notin \mathrm{rng}(p) \\ \bar{u}(yyu_{p^{-1}(i)}u'_{p^{-1}(i)}\vec{x}_i) & \text{si } \varepsilon_i = \uparrow \\ u(u_{p^{-1}(i)}u'_{p^{-1}(i)}zz\vec{x}_i) & \text{si } \varepsilon_i = \downarrow \end{cases} \quad \text{avec} \quad u = \begin{cases} u'_i & \text{si } i \in \mathrm{dom}(p) \\ u_i & \text{sinon} \end{cases}$$

La traduction linéaire est la relation qui associe chaque P à l'ensemble $[\![P]\!]_l$ de ses traductions linéaires pour toutes les énumérations possibles.

Théorème 8.30. *La traduction linéaire est une bisimulation barbelée forte stable.*

Démonstration. Les actions dans un processus et dans sa traduction sont en bijection, et les arguments supplémentaires de chaque action simulent précisément le déblocage des préfixes. La démonstration précise est essentiellement la même que pour la traduction explosive, elle se trouve détaillée dans [8]. □

Une fois encore, on peut remarquer que si P et Q sont deux processus à préfixes clos et à préfixage linéaire, alors $[\![P]\!]_l \mid [\![Q]\!]_l \subseteq [\![P \mid Q]\!]_l$, et donc la composée $[\![\cdot]\!]_{dl}$ des deux traductions distribue elle aussi su la composition parallèle de processus à préfixes clos. La validité de cette traduction (qualifiée de traduction par duos) dérive immédiatement des théorèmes 8.26 et 8.30.

Corollaire 8.31. *Tout processus P est faiblement et stablement bisimilaire par barbelés à toutes ses traductions par duos.*

8.3 Extensions

Le calcul présenté jusqu'ici ne contient aucun mécanisme de réplication ou de récursion, donc les termes et les réseaux (finis) ne peuvent représenter que des comportements finis ; d'autre part, on n'a défini aucun mécanisme apparenté à du choix externe. En d'autres termes, le calcul des réseaux concurrents est purement multiplicatif. Pour étendre l'étude géométrique des calculs de processus à ces deux aspects, on étudie maintenant des extensions possibles, d'abord en introduisant une forme de réplication, puis en montrant dans quelle mesure l'introduction d'un mécanisme de choix est comparable à l'extension du système des préfixes à des gardes non monotones.

8.3.1 Réplication

On enrichit donc le πg-calcul avec un opérateur de réplication noté $!P$. Comme la sémantique opérationnelle est un système de transition étiqueté, cette réplication est traitée de façon paresseuse, c'est-à-dire que des copies sont créées explicitement lors des communications. Le fait de dupliquer un processus impose une certaine discipline sur la manipulation des noms de localités, en particulier à cause de la contrainte selon laquelle chaque nom doit être défini au plus une fois : dans un processus comme $!(\ell : \alpha)$, le nom ℓ devrait apparaître une fois par copie, ce qui est interdit. Deux possibilités sont envisageables :

- Lever la contrainte sur l'unicité de définition de chaque nom. La théorie du calcul en serait légèrement différente, en particulier plusieurs actions seraient susceptibles de libérer une même garde, ce qui nuirait à la représentation algébrique issue de la proposition 8.14 et rendrait les codages plus pénibles à formuler.
- Faire de $!P$ un lieur sur tous les noms de localités. Cette vision souligne le fait qu'un processus $!P$ est un serveur qui attend des communications pour déclencher des copies, auquel cas le fait qu'une action extérieure au serveur soit gardée par l'activation d'une action dans le serveur n'a pas un sens bien défini.

C'est cette seconde approche qu'on emploie ici : dans $!P$, une copie de P est produite en cas de communication et aucune localité définie dans P n'est visible de l'extérieur. On a donc la règle suivante :

$$\frac{P \xrightarrow{(\boldsymbol{\nu}\vec{x})\alpha, L} P'}{!P \xrightarrow{(\boldsymbol{\nu}\vec{x})\alpha, \emptyset} !P \mid (\boldsymbol{\nu}\, \mathrm{loc}(P'))P'}$$

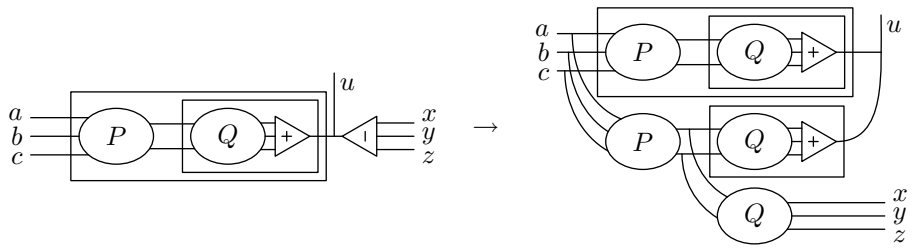


FIG. 8.7 – Réduction avec réplications.

à moins d'imposer par une règle de bonne formation que $\mathrm{loc}(P) = \emptyset$ dans $!P$. Pour conserver une propriété de normalisation dans le style de la proposition 8.14, on doit ajouter aux règles structurelles de la définition 8.5 une loi de distribution des préfixes sur les termes répliqués :

$$\langle\pi\rangle !P \equiv !\langle\pi\rangle P$$

grâce à laquelle on peut étendre la propriété d'énumération de la façon suivante : tout processus P à préfixes clos est structurellement équivalent à un terme de la forme

$$P \equiv (\boldsymbol{\nu}\vec{w})(\boldsymbol{\nu}\vec{m})\Big(\prod_{i=1}^{n}\langle\pi_i\rangle\ell_i : u_i^{\varepsilon_i}(\vec{x}_i) \;\Big|\; \prod_{i=1}^{b}!P_i\Big) \quad \text{avec} \quad \pi_i \equiv \sum_{j=1}^{p_i}\ell_{i,j,1}\cdots\ell_{i,j,q_{i,j}}$$

où les P_i sont tels que $\mathrm{loc}(P_i) = \emptyset$. Pour poursuivre la normalisation dans les sous-termes répliqués, il faut soit imposer à ces termes d'être eux-mêmes à préfixes clos (donc que leurs préfixes ne dépendent pas des ℓ_i, en particulier) soit autoriser des lieux libres dans les préfixes. Dans le premier cas, la représentation graphique s'étend naturellement au moyen de boîtes pour représenter les sous-termes répliqués, et la règle de réduction se traduit naturellement comme illustré figure 8.7. Dans ce cas, le codage des gardes par duos s'étend immédiatement en

$$\Big[\!\Big[P_0 \;\Big|\; \prod_{i=0}^{b}!P_i\Big]\!\Big]_{dl} := [\![P_0]\!]_{dl} \;\Big|\; \prod_{i=0}^{b}![\![P_i]\!]_{dl}$$

où $[\![\cdot]\!]_{dl}$ appliqué aux termes sans réplication est la traduction issue des définitions 8.25 et 8.29. Cette traduction est elle aussi une bisimulation barbelée faible, sa validité découle de celle de la traduction dans le cas fini grâce à la commutation du codage sur la composition parallèle.

Il est naturel de se demander si les codages s'étendent au cas où les préfixes des processus répliqués ne sont pas clos, et donc si le résultat d'expressivité associé entre communication par fusion et gardes monotones est préservé en présence de réplication.

Exemple 8.32. Il est intéressant de remarquer que les traductions développées plus haut sont invalides dans cas. Considérons le cas minimal suivant :

$$P := \ell : \bar{a}\langle b\rangle \mid !(\boldsymbol{\nu}x)(c\langle x\rangle \mid \langle\ell\rangle\bar{x}\langle y\rangle)$$

La traduction par duos définie plus haut ne peut s'appliquer ici, car elle nécessiterait d'étendre la portée de x à celle de ℓ, or il est évident qu'aucune règle d'extension de portée de peut autoriser à étendre une portée hors de la réplication. Ce cas met bien en évidence que le préfixage est une forme de communication, au même titre que le passage de noms, et que les mêmes notions de portée et de confidentialité s'y appliquent. Néanmoins, cet exemple peut se traduire en un processus faiblement bisimilaire dans lequel le préfixage sur ℓ se fait sur un nom public :

$$P' := (\boldsymbol{\nu} uv)\big(\ell : \bar{a}\langle b\rangle \mid \langle \ell\rangle\{u = v\} \mid !Q\big)$$
$$\text{avec} \qquad Q := (\boldsymbol{\nu} xx'm)\big(m : c\langle x\rangle \mid \bar{x}'\langle y\rangle \mid \langle m\rangle \bar{u}\langle xx'\rangle \mid (\boldsymbol{\nu} z)v\langle zz\rangle\big)$$

où $\{u = v\}$ est un processus qui unifie u et v, par exemple $(\boldsymbol{\nu} z)(z\langle u\rangle \mid \bar{z}\langle v\rangle)$. L'intérêt est qu'ici l'action privée est retardée par la fusion des noms x et x' qui restent privés, alors que leur unification se fait au moyen d'une communication sur les canaux publics u et v, une fois que ceux-ci sont unifiés par $\langle \ell\rangle\{u = v\}$. Le rôle du préfixage de $\bar{u}\langle xx'\rangle$ par m est d'interdire les communications sur u entre les copies de Q et $!Q$, sans quoi de telles communications déclencheraient des copies de Q à l'infini, faisant diverger le processus. La garde pointe sur $c\langle x\rangle$ qui est l'unique action publique de Q (en dehors de celles sur u et v bien entendu), afin que le déclenchement d'une copie de Q libère $\bar{u}\langle xx'\rangle$ dont l'effet de fusionner x et x' si la garde ℓ est libérée. Notons que les communications sur u entre différentes copies déclenchées de Q ne posent pas de problème car la seule action présente en réception reste $(\boldsymbol{\nu} z)v\langle zz\rangle$.

Cet exemple illustre comment des processus avec réplication et préfixage arbitraire peuvent être codés en des processus à préfixes clos, qui à leur tour peuvent être traduits en des processus sans préfixage. On se passera ici d'une définition précise d'un tel codage, car l'exemple ci-dessus donne l'essentiel du principe et une définition précise n'apporterait pas spécialement d'éclaircissements supplémentaires.

8.3.2 Sommes et gardes non monotones

Il est naturel de chercher à étendre les résultats d'expressivité de la section précédente au cas des gardes non-monotones. Comme on va le voir ici, il s'avère que l'introduction dans le calcul des gardes non-monotones est comparable à l'introduction des sommes externes.

Pour étendre le πg-calcul avec des gardes non-monotones, on étend simplement la syntaxe des gardes avec une construction $\neg\pi$ munie des règles de dualité classiques, par exemple engendrées par les règles

$$\neg(\pi + \rho) \equiv \neg\pi \cdot \neg\rho \qquad \neg 0 \equiv 1 \qquad \neg\neg\pi \equiv \pi$$

Le système de transition reste le même, enrichi d'une règle de réduction sous les préfixes. Cette règle doit supposer que la garde, avant transition, est libre, ce qui s'écrit :

$$\frac{P \xrightarrow{e,L} P' \qquad \pi[0/\mathbf{L}] \equiv 1}{\langle\pi\rangle P \xrightarrow{e,L} \langle\pi\rangle P'[1/L]}$$

Cette règle est une extension de celle évoquée page 142 comme présentation alternative de l'effet des gardes dans la sémantique opérationnelle du calcul. Notons CN_n le calcul enrichi de gardes non monotones.

Exemple 8.33. Une telle règle est maintenant nécessaire car une garde non-monotone peut être vraie initialement et devenir fausse lors d'une action, par exemple dans le processus $\ell : \alpha \mid \langle \neg \ell \rangle \beta$ l'action β ne peut agir qu'avant α, elle devient bloquée dès que l'action α agit.

On cherche à comparer l'expressivité de cette extension avec l'ajout d'une somme externe. Cette fonctionnalité peut s'introduire naturellement comme un opérateur binaire + associatif et commutatif. On étend le système de transitions avec la règle

$$\frac{P \xrightarrow{(\boldsymbol{\nu}\vec{x})\alpha, L} P'}{P + Q \xrightarrow{(\boldsymbol{\nu}\vec{x})\alpha, L} P'}$$

La règle symétrique se déduit de l'équivalence structurelle $P + Q \equiv Q + P$. Il s'agit d'une somme externe car le choix entre P et Q ne se fait que par communication avec un agent extérieur à la somme. Notons CN^+ le calcul à gardes monotones enrichi d'une somme externe.

Notation 8.34. Pour le codage qui suit, il est pratique de se placer dans la version du langage où une localité peut contenir plusieurs actions. On autorise donc la construction $\ell : P$, et ℓ est alors substitué par 1 dès qu'une action quelconque dans P est utilisée. Cette formulation est équivalente dans la mesure où un tel processus se traduit systématiquement en un processus propre en comprenant ce ℓ comme la somme des noms des actions présentes dans P :

$$\ell : (P \mid Q) \mid R := (\ell_1 : P \mid \ell_2 : Q \mid R)[\ell_1 + \ell_2 / \ell]$$

où ℓ_1 et ℓ_2 sont frais. On se ramène ainsi au cadre précédent. La sémantique de cette variante du calcul s'obtient simplement au moyen de la règle

$$\frac{P \xrightarrow{e, L} P'}{\ell : P \xrightarrow{e, L \cup \{\ell\}} \ell : P'}$$

et sans modification du reste du système. La démonstration précise de l'équivalence entre les deux formulations ne pose aucune difficulté. Dans ce système, on a en particulier les règles suivantes de bisimulation pour le cas où une localité ne contient aucune action :

$$\langle 0 \rangle P \simeq 1 \qquad\qquad (\boldsymbol{\nu}\ell) P \simeq P[0/\ell] \quad \text{si } \ell \notin \mathrm{loc}(P)$$

Définition 8.35. Le codage $[\![\cdot]\!]_g$ de CN^+ dans CN est défini par induction sur les termes en commutant avec toutes les constructions du langage à l'exception de la somme :

$$[\![P + Q]\!]_g := (\boldsymbol{\nu}\ell m)\big(\langle \neg m \rangle \ell : [\![P]\!]_g \mid \langle \neg \ell \rangle m : [\![Q]\!]_g\big)$$

où ℓ et m sont des noms qui n'apparaissent ni dans P ni dans Q.

Proposition 8.36. *Pour tout $P \in CN^+$, les processus P et $[\![P]\!]_g$ sont fortement bisimilaires.*

Démonstration. La preuve se fait par induction structurelle sur les termes. Comme la traduction n'affecte que l'opérateur +, seul ce cas est à prendre

en compte. Soit donc $P + Q$ un processus de CN^+. Toute transition de $P + Q$ est de la forme

$$\frac{P \xrightarrow{(\boldsymbol{\nu}\vec{x})\alpha, L} P'}{P + Q \xrightarrow{(\boldsymbol{\nu}\vec{x})\alpha, L} P'}$$

Par hypothèse d'induction, P et $[\![P]\!]_g$ sont fortement bisimilaires, donc il existe un processus P'' fortement bisimilaire à P' pour lequel on a

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{[\![P]\!]_g \xrightarrow{(\boldsymbol{\nu}\vec{x})\alpha, L} P''}{\ell : [\![P]\!]_g \xrightarrow{(\boldsymbol{\nu}\vec{x})\alpha, L\cup\{\ell\}} \ell : P''}}{\langle\neg m\rangle\ell : [\![P]\!]_g \xrightarrow{(\boldsymbol{\nu}\vec{x})\alpha, L\cup\{\ell\}} \langle\neg m\rangle\ell : P''}}{\langle\neg m\rangle\ell : [\![P]\!]_g \mid \langle\neg\ell\rangle m : [\![Q]\!]_g \xrightarrow{(\boldsymbol{\nu}\vec{x})\alpha, L\cup\{\ell\}} \langle\neg m\rangle\ell : P'' \mid \langle\neg 1\rangle m : [\![Q]\!]_g}}{[\![P + Q]\!]_g \xrightarrow{(\boldsymbol{\nu}\vec{x})\alpha, L} (\boldsymbol{\nu}\ell m)(\langle\neg m\rangle\ell : P'' \mid \langle 0\rangle m : [\![Q]\!]_g) \simeq P''}$$

Donc $[\![P+Q]\!]_g$ se réduit en un processus bisimilaire au réduit de $P+Q$ par une transition de même étiquette, donc $[\![P + Q]\!]_g$ simule $P + Q$.

Réciproquement, si $[\![P + Q]\!]_g$ émet une action, il l'émet forcément par le codage de P ou par celui de Q. Par symétrie, on peut supposer qu'il s'agit de P, dans ce cas la transition doit être issue d'une dérivation de la forme de celle écrite au-dessus, d'après la forme du terme. Par hypothèse d'induction, P est fortement bisimilaire à $[\![P]\!]_g$, donc il existe un processus P' réduit de P par une transition étiquetée $((\boldsymbol{\nu}\vec{x})\alpha, L)$ et fortement bisimilaire à P''. Par conséquent, $P + Q$ se réduit en P' par la même transition, ce qui prouve que $P + Q$ simule $[\![P + Q]\!]_g$. $P + Q$ et $[\![P + Q]\!]_g$ sont donc fortement bisimilaires. □

Il est possible de définir un codage réciproque, qui transforme tout processus de CN_n à gardes non-monotones en un processus de CN^+, avec gardes monotones et sommes. En effet, un préfixage $\langle\neg\ell\rangle\alpha$ signifie que α peut agir tant qu'aucune action étiquetée ℓ n'a agi, autrement dit soit l'action α n'agit pas, soit elle agit avant toutes les actions étiquetées ℓ. Il s'agit donc de traduire selon le schéma

$$\ell : P \mid \langle\neg\ell\rangle\alpha \quad \mapsto \quad \ell : P + (\boldsymbol{\nu}m)(m : \alpha \mid \langle m\rangle\ell : P)$$

Cette traduction simple doit être généralisée au cas général où $\langle\neg\ell\rangle\alpha$ est remplacé par un $\langle\pi\rangle P$, donc avec des gardes arbitraires portant sur des ensembles arbitraires d'actions. On se passera ici d'une définition générale, car une telle traduction serait visiblement très technique, sans être particulièrement instructive sur la nature du préfixage non-monotone.

S'il est une conclusion à tirer de cette brève étude du préfixage non monotone, c'est qu'il s'agit probablement d'une forme de contrôle trop peu structurée, et dont la nature est assez différente de celle du choix externe. Une façon plus pertinente d'implémenter le choix au moyen de mécanismes locaux comme les gardes présentées ici pourrait être d'employer des variables de choix distinctes des noms de lieux, à la manière des réseaux de preuves avec poids [38]. Un poids est une variable p, initialement indéterminée, et dont l'interaction va fixer la valeur à 0 ou 1. Un terme peut alors être pondéré par un choix, ce que l'on

notera $[p]P$ ou $[\neg p]P$, par exemple au moyen d'une règle

$$\frac{P \xrightarrow{\alpha} P'}{[p]P \xrightarrow{\alpha,p=1} P'}$$

Cet effet passe au contexte comme pour les préfixes monotones :

$$\frac{P \xrightarrow{\alpha,p=x} P'}{P \mid Q \xrightarrow{\alpha,p=x} P' \mid Q[x/p]}$$

et on ajoute des règles structurelles $[1]P \equiv P$ et $[0]P \equiv 1$. On ne se lancera pas ici dans l'étude précise du formalisme obtenu, il s'agit plutôt d'une piste pour exploration future. Comme la définition de gardes monotones se fait en remplaçant le préfixe par une mécanisme très comparable à l'activation telle qu'on la trouve dans les structures d'événements, il est probable que la forme géométrique appropriée pour représenter les choix soit une relation d'exclusion mutuelle entre actions.

Conclusion

Conclusions

Dans la première partie de cette thèse, nous avons mené une exploration du sens calculatoire du raisonnement classique. Le formalisme de la réalisabilité classique de Krivine, dont nous illustrons la généralité en l'appliquant à un langage de programmation minimal, donne des techniques puissantes de spécification pour le calcul avec contrôle. Par ce moyen, nous avons décrit avec un certaine précision le contenu opérationnel des formes normales disjonctives, décrivant ainsi le principe de raisonnement par contradiction en termes d'interaction et de communication entre plusieurs flots d'exécution.

Dans la seconde partie, nous avons développé un cadre de réalisabilité analogue pour le calcul concurrent. À partir d'une notion abstraite d'observation, formulée comme une relation d'orthogonalité entre processus, on construit la notion centrale de *comportement*. L'étude de la structure des comportements mène à la définition de connecteurs élémentaires qui décrivent les interactions entre processus. Le langage utilisé est une variante polarisée du π-calcul bénéficiant d'un bonne régularité qui permettent d'obtenir simplement les propriétés fondamentales attendues par la réalisabilité. La logique des comportements que l'on obtient se traduit par un système de typage des processus qui garantit de bonnes propriétés à l'exécution, telles que l'absence de divergence et de blocage.

La logique des comportements contient la logique linéaire, ce qui fournit une interprétation des démonstrations linéaires en tant que processus du π-calcul polarisé. Cette traduction est ensuite utilisée pour obtenir une reconstruction systématique de traductions typées de divers λ-calculs dans le π-calcul. Nous établissons un lien entre la réalisabilité classique et son analogue concurrente, ce qui permet de montrer sémantiquement la correction des traductions, puis d'enrichir les calculs fonctionnels par des combinateurs concurrents.

L'ordonnancement entre les actions d'un processus n'a pas un statut clair d'un point de vue logique. C'est pour mieux le comprendre que, dans la dernière partie, nous menons une étude du préfixage dans le calcul concurrent. Nous montrons en particulier qu'il est possible de traduire sous forme de causalité entre communications toute forme d'ordonnancement au sein d'un processus, de façon analogue aux traductions CPS qui transforment en causalité des stratégies de réduction du λ-calcul. Cette partie a été développée avant la réalisabilité classique, c'est pour cette raison qu'elle est présentée dans un formalisme différent et que l'interprétation logique des traductions n'est pas développée.

Perspectives

Extensions du typage Plusieurs extensions naturelles du système présenté ici sont à envisager. En premier lieu, l'extension du langage des types du calcul propositionnel au calcul des prédicats, avec l'introduction de quantification du premier ordre. Si la définition technique de cette extension est immédiate, c'est sa signification calculatoire qu'il faut comprendre ; c'est certainement un bon moyen pour représenter et typer des structures de données dans le calcul concurrent. À plus long terme, c'est également ce qui permettrait de définir une notion d'extraction de programmes concurrents à partir de la logique.

Une autre extension à considérer est le polymorphisme. Le système des sortes utilisé ici est statique, à la manière des types simples du λ-calcul, ce qui restreint la forme des quantifications sur les comportements. Il serait souhaitable d'ajouter au système un polymorphisme dans les interfaces de processus, et donc dans la logique des comportements. Dans le même ordre d'idées, il peut être intéressant de se demander à quel point les interfaces sont importantes dans la construction du système, et s'il serait possible d'appliquer la même démarche sans prototypage des processus. Les interfaces servent à garantir l'arité et la polarité des actions, donc il sera probablement nécessaire de modifier le calcul sous-jacent pour pouvoir s'en passer.

Contrôle et concurrence Le système de typage des processus est essentiellement une variante de la logique linéaire, c'est pourquoi sa première application concerne le calcul fonctionnel, et les traductions du λ-calcul dans le π-calcul. Au delà de ces traductions, il paraît naturel de poursuivre l'étude en étendant le λ-calcul par des mécanismes concurrents, afin de mieux comprendre la nature des mécanismes de contrôle mis en jeu dans l'interprétation calculatoire de la logique classique. Cette piste est explorée dans le chapitre 7 où la contraction appliquée aux types de base est vue comme un choix non déterministe, cependant l'emploi de types de base fait perdre le polymorphisme, et donc la quantification du second ordre. La poursuite de cette étude pourrait mener à une nouvelle interprétation calculatoire non-déterministe de la logique classique, fondée sur une forme de contrôle originale.

Spécification de processus Si la logique des comportements présentée dans cette thèse décrit des comportements essentiellement fonctionnels, la réalisabilité elle-même n'a pas de telles contraintes, comme on l'illustre à plusieurs reprises en montrant que des types dérivables sont réalisés par des processus significativement concurrents. Dans l'optique de la spécification de processus, un enjeu important est donc l'extension de cette logique, par des règles et des connecteurs capables de décrire des comportements typiques de l'interaction concurrente. Sur ce point, l'étude de l'ordonnancement menée dans le chapitre 8 a probablement des suggestions à apporter, car c'est en comprenant la structure de causalité dans les processus qu'il sera possible de déterminer quels sont les mécanismes fondamentaux à ajouter à la logique.

Les systèmes existants pour la spécification de processus concurrents utilisent des langages tels que les logiques temporelles (logiques LTL et CTL par exemple) qui sont fondés sur des principes assez différents de ceux employés ici. Pour comprendre si notre approche est applicable à la certification de processus, il

est important de comprendre les liens entre la réalisabilité concurrente et de tels formalismes. C'est probablement par une étude plus poussée de la structure des observations que de tels liens pourront être établis.

Sémantique de la concurrence La structure algébrique des comportements suggère la possibilité de définir des sémantiques plus dénotationnelles pour les processus, en s'inspirant notamment de la logique linéaire. Dans cette optique, il sera intéressant de développer une comparaison entre notre système et d'autres modèles de l'interaction, comme la ludique et ses extensions, les sémantiques de jeux ou les systèmes d'interaction.

En particulier, l'étude géométrique de calcul de processus menée dans le chapitre 8 suggère que la géométrie de l'interaction peut être un outil approprié pour l'étude de la structure des processus. Un point important à prendre en compte dans cette optique est le statut à donner au non-déterminisme.

Bibliographie

[1] Samson Abramsky. *Computational interpretations of linear logic.* Theoretical Computer Science, 111(1–2):3–57, 1993.

[2] Samson Abramsky. *Interaction categories.* In Geoffrey L. Burn, Simon J. Gay et Mark Ryan, éditeurs, Theory and Formal Methods, Workshops in Computing, pages 57–69. Springer Verlag, 1993.

[3] Samson Abramsky, Pasquale Malacaria et Radha Jagadeesan. *Full abstraction for PCF.* In International Symposium on Theoretical Aspects of Computer Science (TACS), pages 1–15, 1994.

[4] Andrea Asperti et Giovanna Dore. *Yet another correctness criterion for multiplicative linear logic with mix.* In Anil Nerode et Yuri Matiyasevich, éditeurs, Third international symposium on logical foundations of computer science, Lecture Notes in Computer Science, volume 813, pages 34–46. Springer Verlag, juillet 1994.

[5] Emmanuel Beffara. *Realizability with constants.* Prépublication PPS//03/04//n°19, Équipe PPS, mars 2003. International Workshop on Formal Methods and Security, Nanjing, China.

[6] Emmanuel Beffara. *A concurrent model for linear logic.* In 21st International Conference on Mathematical Foundations of Programming Semantics (MFPS), Electronic Notes in Theoretical Computer Science, 2005. To appear.

[7] Emmanuel Beffara et Vincent Danos. *Disjunctive normal forms and local exceptions.* In 8th ACM International Conference on Functional Programming (ICFP), pages 203–211. ACM Press, 2003.

[8] Emmanuel Beffara et François Maurel. *Concurrent nets: a study of prefixing in process calculi.* Theoretical Computer Science, 2005. Long version of [9], to appear.

[9] Emmanuel Beffara et François Maurel. *Concurrent nets: a study of prefixing in process calculi.* In 11th International Workshop on Expressiveness in Concurrency (EXPRESS), Electronic Notes in Theoretical Computer Science, volume 128, pages 67–86. Elsevier, avril 2005.

[10] Gianluigi Bellin et Philip J. Scott. *On the π-calculus and linear logic.* Theoretical Computer Science, 135(1):11–65, 1994.

[11] Martin Berger, Kohei Honda et Nobuko Yoshida. *Sequentiality and the π-calculus.* In 5th International Conference on Typed Lambda Calculi and Applications (TLCA), 2001.

[12] Gérard Berry et Gérard Boudol. *The chemical abstract machine.* In 17th ACM Symposium on Principles of Programming Languages (POPL), pages 81–94. ACM Press, 1990.

[13] Michele Boreale. *On the expressiveness of internal mobility in name-passing calculi.* Theoretical Computer Science, 195(2):205–226, mars 1998.

[14] Michele Boreale et Rocco de Nicola. *Testing equivalence for mobile processes.* Information and Computation, 120(2):279–303, 1995.

[15] Gérard Boudol et Cosimo Laneve. *The discriminating power of multiplicities in the λ-calculus.* Information and Computation, 126(1):83–102, avril 1996.

[16] Luís Caires et Luca Cardelli. *A spatial logic for concurrency (part II).* Theoretical Computer Science, 322(3):517–565, 2004.

[17] Pierre-Louis Curien et Hugo Herbelin. *The duality of computation.* In 5th ACM International Conference on Functional Programming (ICFP), pages 233–243. ACM Press, 2000.

[18] Mads Dam. *Process-algebraic interpretations of positive linear and relevant logics.* Journal of Logic and Computation, 4:939–973, 1994.

[19] Vincent Danos, Hugo Herbelin et Laurent Regnier. *Games semantics and abstract machines.* In 11th IEEE Symposium on Logic in Computer Science (LICS), 1996.

[20] Vincent Danos, Jean-Baptiste Joinet et Harold Schellinx. *LKQ and LKT: Sequent calculi for second order logic based upon linear decomposition of classical implication.* In Jean-Yves Girard, Yves Lafont et Laurent Regnier, éditeurs, Advances in Linear Logic, pages 211–224. Cambridge University Press, 1995.

[21] Vincent Danos, Jean-Baptiste Joinet et Harold Schellinx. *A new deconstructive logic: linear logic.* Journal of Symbolic Logic, 62:755–807, 1996.

[22] Vincent Danos et Jean-Louis Krivine. *Disjunctive tautologies as synchronisation schemes.* In Peter Clote et Helmut Schwichtenberg, éditeurs, 14th Annual Conference of the European Association for Computer Science Logic (CSL), numéro 1862 in Lecture Notes in Computer Science, pages 292–301. Springer Verlag, 2000.

[23] Vincent Danos et Laurent Regnier. *The structure of multiplicatives.* Archive for Mathematical Logic, 28:181–203, 1989.

[24] Vincent Danos et Laurent Regnier. *How abstract machines implement head linear reduction.* Submitted, 2003.

[25] Rocco de Nicola et Matthew Hennessy. *Testing equivalence for processes.* In 10th International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science, pages 548–560. Springer Verlag, 1983.

[26] Gilles Dowek, Thérèse Hardin et Claude Kirchner. *Theorem proving modulo.* Journal of Automated Reasoning, 2003. To appear.

[27] Thomas Ehrhard et Laurent Regnier. *Differential interaction nets.* In Workshop on Logic, Language, Information and Computation, 2004. Invited paper.

[28] Matthias Felleisen, Daniel P. Friedman, Eugene E. Kohlbecker et Bruce F. Duba. *A syntactic theory of sequential control.* Theoretical Computer Science, 52(3):205–237, 1987.

[29] Arnaud Fleury et Christian Rétoré. *The mix rule.* Mathematical Structures in Computer Science, 4(2):173–185, 1994.

[30] Cédric Fournet et Georges Gonthier. *The reflexive CHAM and the join-calculus.* In 23rd ACM Symposium on Principles of Programming Languages (POPL), pages 372–385. ACM Press, 1996.

[31] Yuxi Fu. *The χ-calculus.* In 1997 Advances in Parallel and Distributed Computing Conference (APDC), pages 74–81. IEEE, Computer Society Press, 1997.

[32] Yuxi Fu. *A proof-theoretical approach to communication.* In P. Degano, R. Gorrieri et A. Marchetti-Spaccamela, éditeurs, 24th International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science, volume 1256, pages 325–335. Springer Verlag, 1997.

[33] Philippa Gardner, Cosimo Laneve et Lucian Wischik. *Linear forwarders.* In Roberto Amadio et Denis Lugiez, éditeurs, Concur 2003, Lecture Notes in Computer Science, volume 2761, pages 415–430. Springer Verlag, 2003.

[34] Philippa Gardner et Lucian Wischik. *Explicit fusions.* In Mogens Nielsen et Branislav Rovan, éditeurs, 25th International Symposium on Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Computer Science, volume 1893, pages 373–382. Springer Verlag, 2000.

[35] Jean-Yves Girard. *Linear logic.* Theoretical Computer Science, 50:1–102, 1987.

[36] Jean-Yves Girard. *A new constructive logic: classical logic.* Mathematical Structures in Computer Science, 1(3):255–296, 1991.

[37] Jean-Yves Girard. *Linear logic: Its syntax and semantics.* In Jean-Yves Girard, Yves Lafont et Laurent Regnier, éditeurs, Advances in Linear Logic, pages 1–42. Cambridge University Press, 1995.

[38] Jean-Yves Girard. *Proof-nets: The parallel syntax for proof-theory.* In Paolo Agliano et Aldo Ursini, éditeurs, Logic and Algebra. M. Dekker, New York, 1996.

[39] Jean-Yves Girard. *Locus solum.* Mathematical Structures in Computer Science, 11(3):301–506, 2001.

[40] Jean-Yves Girard, Yves Lafont et Paul Taylor. *Proofs and types.* Cambridge University Press, 1989.

[41] Timothy G. Griffin. *A formulae-as-types notion of control.* In 17th ACM Symposium on Principles of Programming Languages (POPL), pages 47–58. ACM Press, janvier 1990.

[42] Matthew Hennessy et Robin Milner. *Algebraic laws for nondeterminism and concurrency.* Journal of the ACM, 32(1):137–161, janvier 1985.

[43] Kohei Honda et Nobuko Yoshida. *On reduction-based process semantics.* Theoretical Computer Science, 151(2):437–486, novembre 1995.

[44] Kohei Honda, Nobuko Yoshida et Martin Berger. *Control in the π-calculus.* In 4th ACM-SIGPLAN Continuation Workshop, 2004.

[45] J. Martin E. Hyland et Chih-Hao Luke Ong. *Pi-calculus, dialogue games and PCF.* In 7th ACM Conference on Functional Programming Languages and Computer Architecture, pages 96–107. ACM Press, 1995.

[46] J. Martin E. Hyland et Chih-Hao Luke Ong. *On full abstraction for PCF (parts I, II and III).* Information and Computation, 163(2):285–408, 2000.

[47] Ole Høgh Jensen et Robin Milner. *Bigraphs and transitions.* ACM SIGPLAN Notices, 38(1):38–49, 2003.

[48] Georg Kreisel. *On the interpretation of non-finitist proofs (part I).* Journal of Symbolic Logic, 16(4):241–267, décembre 1951.

[49] Jean-Louis Krivine. *A general storage theorem for integers in call-by-name λ-calculus.* Theoretical Computer Science, 129(1):79–94, juin 1994.

[50] Jean-Louis Krivine. *Typed λ-calculus in classical Zermelo-Frænkel set theory.* Archive in Mathematical Logic, 40(3):189–205, 2001.

[51] Jean-Louis Krivine. *Dependent choice, 'quote' and the clock.* Theoretical Computer Science, 308:259–276, 2003.

[52] Jean-Louis Krivine. *Realizability in classical logic.* Cours d'école doctorale, Université de Marseille Luminy, 2004. A paraître dans Panoramas et synthèses, Société Mathématique de France.

[53] Yves Lafont. *Interaction nets.* In 17th ACM Symposium on Principles of Programming Languages (POPL), pages 95–108. ACM Press, 1990.

[54] Yves Lafont. *Interaction combinators.* Information and Computation, 137(1):69–101, 1997.

[55] Cosimo Laneve, Joachim Parrow et Björn Victor. *Solo diagrams.* In Naoki Kobayashi et Benjamin C. Pierce, éditeurs, 4th International Symposium on Theoretical Aspects of Computer Science (TACS), Lecture Notes in Computer Science, volume 2215, pages 127–144. Springer Verlag, 2001.

[56] Cosimo Laneve et Björn Victor. *Solos in concert.* In Jiří Wiederman, Peter van Emde Boas et Mogens Nielsen, éditeurs, 26th International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Computer Science, volume 1644, pages 513–523. Springer Verlag, juillet 1999.

[57] Cosimo Laneve et Björn Victor. *Solos in concert.* Mathematical Structures in Computer Science, 13(5):657–683, octobre 2003.

[58] Olivier Laurent. *Étude de la polarisation en logique.* Thèse de doctorat, Université Aix-Marseille II, mars 2002.

[59] Xavier Leroy, Damien Doligez, Jacques Garrigue, Didier Rémy et Jérôme Vouillon. *The Objective Caml system.* INRIA, 2004.

[60] John McCarthy. *A basis for a mathematical theory of computation.* Computer Programming and Formal Systems, pages 33–70, 1963.

[61] Robin Milner. *Communication and concurrency.* Prentice Hall, 1989.

[62] Robin Milner. *Functions as processes.* In 17th International Colloquium on Automata, Languages and Programming (ICALP), pages 167–180. Springer Verlag, 1990.

[63] Robin Milner. *Pi-nets: A graphical form of π-calculus*. In Donald Sannella, éditeur, 5th European Symposium on Programming (ESOP), Lecture Notes in Computer Science, volume 788, pages 26–42. Springer Verlag, 1994.

[64] Robin Milner, Joachim Parrow et David Walker. *Modal logics for mobile processes*. Theoretical Computer Science, 114(1):149–171, 1993.

[65] Peter O'Hearn et David Pym. *The logic of bunched implication*. Bulletin of Symbolic Logic, 5(2):215–244, juin 1999.

[66] Chih-Hao Luke Ong et Charles Alexander Stewart. *A Curry-Howard foundation for functional computation with control*. In 24th ACM Symposium on Principles of Programming Languages (POPL), 1997.

[67] Michel Parigot. *$\lambda\mu$-calculus: an algorithmic interpretation of classical natural deduction*. In 3rd International Conference on Logic Programming and Automated Reasoning (LPAR), Lecture Notes in Computer Science, volume 624, pages 190–201. Springer Verlag, 1992.

[68] Joachim Parrow. *Trios in concert*. In Gordon Plotkin, Colin Stirling et Mads Tofte, éditeurs, Proof, Language and Interaction: Essays in Honour of Robin Milner, pages 621–637. MIT Press, 1998.

[69] Joachim Parrow et Björn Victor. *The fusion calculus: Expressiveness and symmetry in mobile processes*. In 13th IEEE Symposium on Logic in Computer Science (LICS), pages 176–185, 1998.

[70] Gordon Plotkin. *Call-by-name, call-by-value and the λ-calculus*. Theoretical Computer Science, 1(2):125–159, décembre 1975.

[71] Laurent Regnier. *Une équivalence sur les λ-termes*. Theoretical Computer Science, 126(2):281–292, 1994.

[72] Davide Sangiorgi. *An investigation into functions as processes*. In 9th International Conference on Mathematical Foundations of Programming Semantics (MFPS), pages 143–159, London, UK, 1994. Springer Verlag.

[73] Davide Sangiorgi. *π-calculus, internal mobility and agent-passing calculi*. Theoretical Computer Science, 167(2):235–274, 1996.

[74] Davide Sangiorgi et David Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.

[75] Glynn Winskel. *Event structures*. In Advances in Petri nets: applications and relationships to other models of concurrency, pages 325–392. Springer Verlag, 1987.

[76] Nobuko Yoshida, Martin Berger et Kohei Honda. *Strong normalisation in the π-calculus*. In 16th IEEE Symposium on Logic in Computer Science (LICS), pages 311–322, 2001.

Logique, Réalisabilité et Concurrence

Cette thèse se consacre à l'application de techniques de réalisabilité dans le cadre de l'étude du sens calculatoire de la logique. Dans une première partie, nous rappelons le formalisme de la réalisabilité classique de Krivine, dans lequel nous menons ensuite une étude du contenu opérationnel de tautologies purement classiques. Cette exploration du sens calculatoire de la disjonction classique révèle des comportements riches, avec une forte intuition interactive, qui s'interprètent avantageusement comme des structures de contrôle typées. Afin de mieux comprendre la nature de ces mécanismes, nous définissons ensuite une technique de réalisabilité à la Krivine pour un modèle de calcul concurrent, dans le but d'obtenir une notion de constructivité qui ne soit plus fondée sur l'idée de fonction, mais sur celle de processus interactif. Le cadre ainsi obtenu donne une interprétation réellement concurrente de la logique linéaire dans un calcul de processus dérivé du π-calcul, permettant d'appliquer au cas concurrent la méthode de spécification précédemment étudiée dans le cas séquentiel. Par la suite, l'étude des traductions de la logique classique vers la logique linéaire mène à reconstruire systématiquement des décompositions interactives du calcul fonctionnel, permettant ainsi de faire le lien au niveau logique entre les réalisabilités classique et concurrente. Dans une dernière partie, nous étudions plus en détail le mode de calcul issu des algèbres de processus, afin de comprendre son système d'ordonnancement. Cette étude mène à la définition d'un modèle de calcul plus géométrique qui permet une exploration formelle de la notion de causalité dans les calculs concurrents.

Logic, Realisability and Concurrency

This thesis is devoted to the application of realisability techniques in the study of the computational meaning of logic. In a first part, we recall the formalism of Krivine's classical realisability, in which we carry a study of the operational contents of purely classical tautologies. This exploration of the computational meaning of classical disjunction reveals rich behaviours, with strong interactive intuitions, that are profitably interpreted as typed control structures. In search of a better understanding of the nature of those mechanisms, we then define a similar realisability technique for a model of concurrent computation, in order to build a notion of constructivity that is not based on the idea of function, but rather on that of interactive process. The framework we get provides a genuinely concurrent interpretation of linear logic in a process calculus derived from the π-calculus, which allows us to apply in the concurrent case the specification methods previously studied in the sequential context. Subsequently, the study of translations of classical logic into linear logic provides systematic reconstructions of interactive decompositions of functional computation, which establishes a link at the logical level between the sequential and concurrent realisabilities. In the last part, we study in more detail the kind of computation inherited from process algebras, in order to understand its kind of scheduling. This study leads us to the definition of a more geometric computation model that allows a formal exploration of the notion of causality in concurrent calculi.