# $\pi$ et $\lambda$

## Une étude sur la traduction des $\lambda$-termes dans le $\pi$-calcul

Damiano Mazza*

30 Juin 2003

Mémoire de DEA de
Mathématiques Discrètes et Fondements de l'Informatique
sous la direction de Laurent Regnier

Institut de Mathématiques de Luminy, Marseille, France

*mazza@iml.univ-mrs.fr

# Introduction

Ce mémoire a pour but l'étude du rapport entre le $\pi$-calcul, un formalisme introduit à la fin des années '80 pour la modélisation des systèmes mobiles, et le $\lambda$-calcul, l'un des modèles de calcul les plus connus et utilisés en informatique théorique.

L'exposition sera articulée en trois chapitres. Le premier chapitre sera consacré à la présentation (bien sûr au niveau introductif !) du $\pi$-calcul. Dans le deuxième chapitre on se concentrera sur la définition des principales stratégies de réduction du $\lambda$-calcul ; en fait, ce n'est pas possible de parler du rapport entre les deux formalismes sans considérer une stratégie d'évaluation bien précise pour le $\lambda$-calcul, car le $\pi$-calcul n'arrive pas à simuler la dynamique entière de ceci. Finalement, le troisième chapitre fera face au problème de la représentation du $\lambda$-calcul à l'intérieur du $\pi$-calcul. La traduction de Milner pour le $\lambda$-calcul *call by name* sera présentée, et sur celle-ci on démontrera un résultat la reliant à la *réduction linéaire de tête faible*, une stratégie d'évaluation du $\lambda$-calcul qui a été prouvé être corrélée à l'exécution des $\lambda$-termes au moyen de la machine de Krivine (KAM).

L'annexe qui conclut ce rapport contient une preuve alternative de la correspondance entre la traduction de Milner et la réduction linéaire de tête faible, basée sur l'introduction d'une machine abstraite à la Krivine, que l'on appellera *machine de Milner*. Cette démonstration, même si indirecte, a l'avantage d'être sans doute plus simple que celle fournie dans le troisième chapitre.

## Remerciements

Laurent Regnier m'a strictement interdit de le remercier pour avoir accepté d'être le directeur de mon stage et pour m'avoir suivi pendant son déroulement et aidé à développer les idées qui sont à la base de ce mémoire ; donc je ne vais pas le faire ici. Je remercie alors Thomas Ehrhard pour sa participation et contribution aux discussions entre moi et Laurent. Je suis aussi beaucoup reconnaissant à tout le monde à l'équipe LDP de Luminy, et en particulier aux thésards et aux autres étudiants du DEA qui ont contribué à créer une ambiance stimulante pour le travail et en même temps sûrement de bonne humeur.

# Table des matières

# Chapitre 1

# Le $\pi$-calcul

Le $\pi$-calcul (voir [SW01] pour une présentation extensive) est un formalisme qui a été introduit par Robin Milner avec le but de fournir une présentation mathématique complète de ce qu'on appelle généralement « processus mobiles ».

A la base de l'idée de mobilité il y a les deux concepts fondamentaux de *parallélisme* et de *concurrence*. Le parallélisme peut prendre des formes différentes, mais en général on pourrait dire que la caractéristique clé des mécanismes parallèles est la possibilité d'exécuter plusieurs tâches en même temps, en contraste avec l'exécution séquentielle, qui prévoit le développement d'une seule tâche à la fois. La concurrence fait appel au concept de *ressource* ; dans un contexte concurrent, les processus demandent pendant leur exécution l'utilisation d'une ou plusieurs ressources, lesquelles sont partagées entre les processus eux-mêmes. Ce partage implique la possibilité de collisions, c'est à dire de situations dans lesquelles une même ressource est demandée par plusieurs processus. C'est justement là qu'on a la concurrence : les processus sont en compétition pour l'accès aux ressources. Puisqu'en général on peut imaginer que l'accès à une ressource est lui-même géré par un certain processus, la concurrence comporte la présence d'*interactions* entre les processus. On pourrait bien imaginer une exécution parallèle sans interaction, mais cette dernière est absolument nécessaire pour qu'on puisse parler de concurrence.

Un processus mobile est donc un processus qui agit dans un contexte parallèle et concurrent, i.e. son exécution procède en parallèle avec celle d'autres processus et il existe la possibilité qu'il y ait des interactions concurrentes avec ces processus. En outre, la mobilité est liée à l'idée de *localisation* : chaque processus est *localisé* dans le cadre de l'environnement où il se trouve, et l'interaction dépend de façon fondamentale de cette localisation, car les processus peut « bouger » à l'intérieur de l'environnement et ils ont besoin que leur « position » soit connue pour que les autres puissent communiquer avec eux.

L'interaction est modélisée dans le $\pi$-calcul par le passage de *noms*. Les noms sont les unités fondamentales du $\pi$-calcul ; en gros, ils représentent des « canaux de communication » lesquels sont utilisé par les processus pour interagir l'un avec l'autre. La subtilité du $\pi$-calcul est dans le fait que ces interactions n'ont elles-mêmes comme seul objet que des noms. Autrement dit, les noms sont à la fois moyens de communication et objets des communications.

Tout cela veut dire que les processus du $\pi$-calcul interagissent entre eux d'une manière qui peut sembler assez bizarre : la seule chose qu'ils sont capables de

se communiquer est la possibilité même de communiquer ! La leçon fondamentale que le $\pi$-calcul et les autre calculs de passage de noms nous enseignent est que cette interaction, aussi simple qu'elle peut être, est en vérité suffisante pour modéliser en plein la mobilité des processus. D'ailleurs tout ça n'est pas si étonnant quand on pense que la substitution d'un terme à une variable est tout ce qu'il faut pour calculer n'importe quoi...

Passons donc à la définition formelle du $\pi$-calcul[1]. Comme on l'a déjà dit, on supposera avoir à notre disposition un ensemble dénombrable de *noms*, qu'on appellera par exemple $x, y, z, \ldots$ Avec ces noms, on va construire des objets syntaxiques qu'on appelle *préfixes* engendrés par la grammaire ci-dessous :

$$\pi ::= \overline{x}y \mid x(y)$$

Une brève explication de la signification des préfixes peut se donner de la façon suivante :

- $\overline{x}y$ est le *préfixe d'émission*, ou *émetteur* ; un terme contenant ce préfixe a la capacité d'envoyer le nom $y$ par le nom $x$.
- $x(y)$ est le *préfixe de réception*, ou *récepteur* ; un terme contenant ce préfixe a la capacité de recevoir un nom quelconque par le nom $x$ (la signification de $y$ sera expliquée ultérieurement).

Maintenant, les termes du $\pi$-calcul sont engendrés par la grammaire qui suit :

$$P ::= \mathbf{0} \mid \pi.P \mid P \mid P \mid \boldsymbol{\nu} z.P \mid !P$$

En général, on appellera les termes du $\pi$-calcul tout simplement « processus ». Voici une petite description de leur signification informelle :

- $\mathbf{0}$ est le processus nul ; c'est un processus qui ne peut rien faire.
- $\pi.P$ est un processus qui a la capacité donnée par le préfixe $\pi$, et il est en attente de réaliser celle-ci. Tant qu'il n'arrive pas à la réaliser, il est dans un état de « blocage ». Lorsqu'un processus ait cette forme, on dira que $\pi$ est son *préfixe gardien*.
- $P \mid Q$ est un processus dans lequel les processus $P$ et $Q$ sont libre d'interagir entre eux. L'opérateur $\mid$ (composition, ou « parallèle ») est donc la base du parallélisme dans le $\pi$-calcul, et il joue aussi un rôle fondamental dans la représentation de la concurrence, car il n'y a pas d'interaction possible entre deux processus du $\pi$-calcul sans qu'ils soient amenés ensemble par $\mid$ (on verra la règle d'interaction un peu plus loin).
- $\boldsymbol{\nu} z.P$ est un processus dans lequel le nom $z$ est « réservé », c'est à dire $z$ n'est connu qu'à l'intérieur de $P$. L'opérateur $\boldsymbol{\nu}$ s'appelle *restriction*, car son action est celle de limiter le champ d'action d'un nom. Dans l'opérateur $\boldsymbol{\nu}$ il y a la concrétisation de l'idée de localisation dont on a parlé tout à l'heure.
- $!P$ est un processus qui peut être copié un nombre illimité de fois. L'opérateur !, qui est nommé *réplication*, rappelle l'exponentiel de la logique linéaire ; c'est justement cette dernière qui a inspiré la notation utilisée.

[1] En vérité on en presentera ici une version restreinte, limitée au fragment utilisé pour représenter le $\lambda$-calcul. Le lecteur est invité à consulter [SW01] pour une présentation plus générale.

Une fois qu'on a introduit la syntaxe du $\pi$-calcul, il faut lui donner une *dynamique*, c'est à dire un ensemble de règles qui permettent aux termes d'interagir et donc de représenter un processus de calcul. Pour faire cela, on va d'abord éliminer de la syntaxe les distinctions inutiles. En fait, la réduction du $\pi$-calcul sera du même style que celle du $\lambda$-calcul, dans le sens qu'elle sera basée sur des substitutions. Chaque fois qu'on se trouve en face d'une notion de substitution, on doit être capable de gérer dans notre syntaxe le fait qu'il existe des occurrences de variables (ou des noms, dans le cas du $\pi$-calcul) dont la seule utilisation sera celle d'être remplacées par quelque chose d'autre (par des termes dans le $\lambda$-calcul, tandis qu'on verra plus loin que dans le $\pi$-calcul on n'a que le droit de remplacer un nom par un autre nom). Dans le $\pi$-calcul, comme dans le $\lambda$-calcul, il y a donc des occurrences de noms dont la seule chose qui importe est la *position* à l'intérieur du processus, et non pas le « nom » effectif. En parfaite analogie avec le $\lambda$-calcul, on dira alors que certaines occurrences des noms d'un processus sont *liées*, et puis on définira une relation d'équivalence (l'$\alpha$-équivalence) qui nous permettra d'oublier les différences syntaxiques inutiles.

Il existe une autre différence entre le $\pi$-calcul et le $\lambda$-calcul : dans ce dernier, le seul lieur de variables est le $\lambda$ ; dans le $\pi$-calcul, il en existe deux :

**Définition 1.1 (Lieurs)** *Dans le processus $x(y).P$, toute occurrence de $y$ dans $P$ est liée par le préfixe de réception, c'est à dire $x(y)$ est un lieur pour $y$ (l'occurrence de $y$ dans $x(y)$ est dite* liante*) et sa portée est $P$.*
*De même, dans le processus $\boldsymbol{\nu} z.P$, toute occurrence de $z$ dans $P$ est liée par la restriction, c'est à dire $\boldsymbol{\nu} z$ est un lieur pour $z$ (l'occurrence de $z$ dans $\boldsymbol{\nu} z$ est dite* liante*) et sa portée est $P$.*

Une fois qu'on a défini les lieurs, on peut définir l'ensemble des *noms libre* d'un processus $P$, qu'on notera $\mathsf{fn}(P)$ (c'est l'équivalent de l'ensemble des variables libres d'un $\lambda$-terme $T$, que l'on appellera $\mathsf{fv}(T)$).

Maintenant on peut définir l'$\alpha$-équivalence, exactement comme dans le $\lambda$-calcul :

**Définition 1.2 ($\alpha$-équivalence)** *Soient $P$ et $Q$ deux processus. On dira que $P$ est $\alpha$-équivalent à $Q$ (et vice-versa), et on écrira $P \simeq_\alpha Q$, s'ils ne diffèrent que pour un renommage d'occurrences liantes de noms et de toutes les occurrences liées correspondants.*

A partir de ce moment on considérera, sans aucune exception, les processus du $\pi$-calcul à $\alpha$-équivalence près.

Après avoir défini l'$\alpha$-équivalence, on est déjà en position de traiter la dynamique du $\pi$-calcul de façon intuitive ; plus loin on s'occupera de la définir formellement. D'abord, on introduira quelques conventions syntaxiques qui vont simplifier l'exposition, dont la justification formelle sera aussi donnée un peu plus loin :

(a) On supposera que l'opérateur de réplication a priorité syntaxique sur n'importe quel autre opérateur.

(b) On va regarder les processus du $\pi$-calcul et l'opérateur « parallèle » comme formant un monoïde commutatif avec élément neutre $\mathbf{0}$ ; en conséquence, on oubliera sans soucis toute parenthèses dans des expressions comme $P \mid Q \mid R$.

(c) On abrégera une expression comme $\boldsymbol{\nu} x_1.\boldsymbol{\nu} x_2 \ldots \boldsymbol{\nu} x_n.P$ par $\boldsymbol{\nu}(x_1, x_2, \ldots, x_n).P$ ou, encore plus simplement, $\boldsymbol{\nu}\vec{x}.P$.

(d) On notera la substitution des occurrences libres du nom $x$ avec le nom $y$ dans le processus $P$ par $P\{y/x\}$.

(e) Un processus qui ne contient que des préfixes est forcement de la forme $\pi_1 \ldots \pi_n.\mathbf{0}$ ; dans la suite, on oubliera souvent le $\mathbf{0}$ terminal, en sachant qu'il est toujours là sans exception, et on écrira par exemple $x(z).\overline{z}y$ au lieu que $x(z).\overline{z}y.\mathbf{0}$.

Avec ces conventions (et en vérité avec des autres outils formels qu'on n'a pas encore vus), on peut montrer que tout processus du $\pi$-calcul prend la forme

$$\boldsymbol{\nu}(z_1, \ldots, z_m).\left(\pi_1.P_1 \mid \ldots \mid \pi_k.P_k \mid !P_{k+1} \mid \ldots \mid !P_n\right) \qquad (*)$$

où les $P_i$ sont des processus de la même forme que celle ci-dessus.

L'idée à la base de la réduction du $\pi$-calcul est intuitivement la suivante. Supposons que $P$ soit un processus du $\pi$-calcul, qui est donc dans la forme qu'on vient de donner ; supposons en outre qu'il y ait deux préfixes gardiens parmi les $\pi_i$ (appelons les $\pi_a$ et $\pi_b$) tels que, par exemple, $\pi_a = \overline{x}y$, et $\pi_b = x(z)$, où $x, y, z$ sont trois noms quelconques, mais bien entendu $x$ est le même nom soit dans le préfixe émetteur que dans le préfixe récepteur. Dans ce cas, ces deux préfixes peuvent réaliser leurs capacités, c'est à dire l'un peut envoyer $z$ et l'autre peut le recevoir, et on dit alors qu'il y a *synchronisation* entre $\pi_a.P_a$ et $\pi_b.P_b$.

Lorsqu'on a établi que la communication peut avoir lieu, la chose suivante va se passer : en ayant réalisé leurs capacités, les préfixes gardiens de $\pi_a.P_a$ et $\pi_b.P_b$ vont disparaître, et l'effet de la communication va être une substitution dans $P_b$, qui était le processus gardé par le préfixe récepteur. Plus précisément, on aura que $\pi_a.P_a$ va se transformer en $P_a$, tandis que $\pi_b.P_b$ va se transformer en $P_b\{y/z\}$. La réduction va donc avoir la forme suivante :

$$\boldsymbol{\nu}(z_1, \ldots, z_m).\left(\ldots \mid \overline{x}y.P_a \mid \ldots \mid x(z).P_b \mid \ldots\right) \longrightarrow$$

$$\longrightarrow \boldsymbol{\nu}(z_1, \ldots, z_m).\left(\ldots \mid P_a \mid \ldots \mid P_b\{y/z\} \mid \ldots\right)$$

Quelques remarques : en premier lieu, rien n'interdit que le noms non liés impliqués dans la communication (à savoir $x$ et $y$) soient en fait des noms privés, c'est à dire qu'ils apparaissent parmi les $z_i$ ; même dans ce cas, il n'y aurait aucune différence en ce qui concerne la réduction. Deuxièmement, on n'a pas encore considéré les sous-processus de $P$ qui éventuellement se trouvent sous une réplication. En effet, même si la dynamique de base reste la même (un émetteur et un récepteur qui se synchronisent), la définition de la réduction en présence de réplications n'est pas tout à fait simple ; elle est pratiquement impossible à présenter informellement dans sa généralité. On verra en suite qu'il faudra introduire un outil technique particulier (la *congruence structurelle*) pour la traiter rigoureusement.

Néanmoins, à partir de cette présentation intuitive de la réduction du $\pi$-calcul on peut déjà déduire quelques caractéristiques fondamentales de sa dynamique :

– La récriture du $\pi$-calcul n'est pas confluente. Il suffit de considérer l'exemple suivant :

$$\overline{x}y \mid x(a).P \mid x(b).Q$$

Ce processus admet deux réduits, à savoir $P\{y/a\}$ et $Q\{y/b\}$, qui n'ont aucune raison d'être égaux.

– En général, la réduction « consomme » les termes, c'est à dire elle produit des termes « plus petits » par rapport à ceux de départ. L'exception importante sont les termes qui se trouvent au dessous d'une réplication ; ce n'est que grâce a cet opérateur qu'on peut fabriquer des processus dont la réduction peut continuer en théorie à l'infini, comme $!\overline{x}y \mid !x(z)$.
– La réduction peut avoir lieu entre deux sous-processus situés n'importe où à l'intérieur du même processus, et cela concorde avec l'idée que le $\pi$-calcul doit modéliser la mobilité. Toutefois, il y a des endroits où la réduction est interdite même s'il existe deux sous-processus en théorie capables d'interagir. En particulier, au dessous d'un préfixe, comme dans le processus

$$\overline{w}u.\left(\overline{x}y.P \mid x(a).Q\right) \mid z(b).R$$

qui n'a aucune réduction admissible, même s'il contient un sous-processus émetteur et un sous-processus récepteur qui pourraient communiquer le long du canal $x$. Comme on a déjà souligné avant, les préfixes « bloquent » le processus dont ils sont gardiens jusqu'au moment où ils arrivent à exercer leur capacité. Dans ce cas, il n'y a personne capable de recevoir le nom $u$ le long du nom $w$, et donc la partie gauche du parallèle reste dans un état de blocage.

Procédons maintenant à la définition formelle de la dynamique du $\pi$-calcul. Alors que dans le $\lambda$-calcul il suffit de définir une règle de réduction simplifiée (la $\beta_0$-réduction) et puis de l'étendre par clôture réflexive, transitive et contextuelle à la règle générale, dans le $\pi$-calcul il faut suivre une approche un peu différente. En fait, à cause de la nature parallèle des processus, on vient de voir que l'interaction ne peut pas être restreinte aux termes qui se trouvent « à côté » l'un de l'autre, comme dans le $\lambda$-calcul, mais il faut qu'il soit possible pour deux sous-termes d'interagir même si l'un se trouve « loin » de l'autre à l'intérieur du terme qui les contient. Toutefois, le lecteur pourra bien imaginer qu'il serait incroyablement compliqué de définir une interaction « à distance » comme celle esquissée précédemment en utilisant la même méthode que pour le $\lambda$-calcul (en particulier parce que, comme on l'a bien pu constater dans l'exemple précédent, la réduction du $\pi$-calcul ne passe pas au contexte).

En conséquence, dans ce cadre l'approche « chimique » de Berry et Boudol introduit dans [BB92] est largement plus commode. La dynamique du calcul est donc décomposée en deux parties : une partie « structurelle » qui détermine comment les sous-termes d'un processus peuvent « bouger » à l'intérieur du même processus avec le but de se trouver en position de pouvoir communiquer, et une partie « d'interaction » qui définit la façon exacte avec laquelle les communications ont lieu (une fois qu'elles sont devenues possibles grâce aux mouvements structurels).

La première étape sera donc celle de définir une *congruence structurelle* sur les processus du $\pi$-calcul :

**Définition 1.3 (Congruence structurelle)** *La* congruence structurelle, *qu'on notera $\equiv$, est la plus petite relation d'équivalence sur les processus du $\pi$-calcul (à $\alpha$-équivalence près) qui satisfait les équations du tableau 1.1.*

On peut bien vérifier que la congruence structurelle capture parfaitement les « mouvements » que les sous-processus doivent faire pour garantir une interaction comme celle introduite informellement ci-dessus. Par ailleurs, grâce à la

| | | | |
|---|---|---|---|
| Sc-comp-assoc | $P \mid (Q \mid R)$ | $\equiv$ | $(P \mid Q) \mid R$ |
| Sc-comp-inact | $P \mid \mathbf{0}$ | $\equiv$ | $P$ |
| Sc-comp-comm | $P \mid Q$ | $\equiv$ | $Q \mid P$ |
| Sc-res | $\boldsymbol{\nu} x.\boldsymbol{\nu} y.P$ | $\equiv$ | $\boldsymbol{\nu} y.\boldsymbol{\nu} x.P$ |
| Sc-res-inact | $\boldsymbol{\nu} z.\mathbf{0}$ | $\equiv$ | $\mathbf{0}$ |
| Sc-res-comp | $\boldsymbol{\nu} z.(P \mid Q)$ | $\equiv$ | $P \mid \boldsymbol{\nu} z.Q,\ z \notin \mathsf{fn}(P)$ |
| Sc-rep | $!P$ | $\equiv$ | $P \mid !P$ |

Tab. 1.1 – Les équations de la congruence structurelle.

$$\frac{}{\overline{x}y.P \mid x(z).Q \longrightarrow P \mid Q\{y/z\}}\ \text{R-inter}$$

$$\frac{P \longrightarrow Q}{P \mid R \longrightarrow Q \mid R}\ \text{R-par} \qquad \frac{P \longrightarrow Q}{\boldsymbol{\nu} z.P \longrightarrow \boldsymbol{\nu} z.Q}\ \text{R-res}$$

$$\frac{P \equiv P' \quad P' \longrightarrow Q' \quad Q' \equiv Q}{P \longrightarrow Q}\ \text{R-struct}$$

Tab. 1.2 – Les règles de réduction du $\pi$-calcul.

congruence structurelle on arrive finalement à comprendre la raison des conventions syntaxiques qu'on a faites tout à l'heure. Par exemple, les Sc-comp-$*$ justifient la structure de monoïde que l'on a supposée pour la convention (b), alors que Sc-res est évidemment à la base de la convention (c), et ainsi de suite. Toutes ensemble, les équations de la congruence structurelle permettent aussi de démontrer que tout terme du $\pi$-calcul peut s'écrire dans la forme $(*)$ ; en fait, on peut maintenant formuler le résultat plus précisément en disant que tout terme est *structurellement équivalent* à un terme de la forme $(*)$. En outre, les règles de réduction qu'on verra un peu plus loin et l'équation Sc-rep formalisent le comportement de l'interaction en présence de réplications.

On peut maintenant définir formellement la réduction du $\pi$-calcul :

**Définition 1.4 (Réduction)** *La réduction du $\pi$-calcul, notée $\longrightarrow$, est la relation définie par les règles du tableau 1.2.*

La réduction du $\pi$-calcul est donc constituée par une règle de base et trois règles qui servent à « propager » l'action de celle-ci à des contextes plus grands. En particulier, dans la règle R-struct on voit bien comment la congruence structurelle joue un rôle fondamental dans la définition de la réduction. En plus, on remarque qu'il n'existe pas de règle qui permettent d'effectuer une

réduction au dessous d'un préfixe, et c'est justement cela le comportement que la réduction doit avoir.

Avec les quatre règles de réduction et la congruence structurelle, on peut ramener n'importe quelle interaction « à distance » à une interaction de type « local », dans le sens où les sous-termes qui interagissent sont syntaxiquement « à côté ». On obtient ainsi un calcul qui se comporte de la façon qu'on a décrite informellement, capable de représenter la dynamique des processus mobiles.

Avant de laisser (temporairement) le sujet, il faut brièvement introduire une petite variante du $\pi$-calcul, qui sera celle que l'on utilisera en pratique. Cette version du calcul s'appelle *$\pi$-calcul polyadique*, et la seule différence consiste dans la possibilité d'envoyer et recevoir plusieurs noms en même temps (c'est justement cette caractéristique qu'on appelle *polyadicité*).

La syntaxe du calcul est donc modifiée en étendant les préfixes d'émission et de réception pour qu'ils puissent envoyer et recevoir un nombre arbitraire de noms :

$$\pi ::= \overline{x}\langle y_1, \ldots, y_m \rangle \;\Big|\; x(y_1, \ldots, y_n)$$

La réduction reste définie de manière identique, à exception de la règle R-INTER, qui doit être substituée par une version polyadique :

$$\frac{}{\overline{x}\langle y_1, \ldots, y_n \rangle.P \mid x(z_1, \ldots, z_n).Q \longrightarrow P \mid Q\{y_1/z_1, \ldots, y_n/z_n\}}\ \text{R-INTER}$$

C'est implicite dans la règle que l'arité de l'émetteur et celle du récepteur doivent nécessairement coïncider ; si les deux arités sont différentes, il n'y a pas d'interaction.

# Chapitre 2

# Les stratégies de réduction dans le $\lambda$-calcul

Le $\lambda$-calcul, introduit par Alonzo Church dans les années '30 et pour cela bien connu et étudié depuis des dizaines d'années, est le formalisme d'excellence pour la modélisation de la programmation fonctionnelle. Les différences clé entre le $\lambda$-calcul et le $\pi$-calcul peuvent être synthétisées comme suit :

- La réduction du $\lambda$-calcul prévoit une substitution apparemment plus « forte » que celle du $\pi$-calcul : dans ce dernier, tout ce qu'on peut faire est substituer un nom par un autre nom, alors que dans le $\lambda$-calcul on peut remplacer une variable (l'équivalent d'un nom) par un *terme* quelconque.
- Pourtant, la réduction du $\lambda$-calcul est confluente, au contraire de celle du $\pi$-calcul, laquelle, comme on l'a vu, présente des situations de non-confluence.
- La programmation modélisée par le $\lambda$-calcul est strictement séquentielle, tandis que le $\pi$-calcul est expressément conçu pour représenter des processus parallèles.

Même si la propriété de Church-Rosser garantit qu'en présence de plusieurs redex on peut choisir n'importe lequel parmi eux sans changer le résultat final de la réduction, ce choix a bien sûr des conséquences si regardé d'autres points de vue. Par exemple, la longueur de la réduction peut évidemment être conditionnée par les choix faits pendant la réduction elle-même, jusqu'à arriver à des cas extrêmes où certains choix conduisent à la forme normal tandis que certaines autres engendrent une suite de réductions infinie.

Par ailleurs, puisque le $\lambda$-calcul est la base théorique de tous les langages de programmation fonctionnels, l'étude de la $\beta$-réduction devient extrêmement important pour l'implémentation efficace de ces langages. En particulier, les ordinateurs conventionnels étant séquentiels et déterministes (normalement il n'y a qu'un seul processeur qui ne peut exécuter qu'une seule instruction à la fois), l'exécution d'un programme fonctionnel correspondant à un $\lambda$-terme nécessite un choix à chaque moment où se présentent plusieurs possibilités de réduction.

En conséquence, une partie importante de la théorie du $\lambda$-calcul s'occupe d'étudier ce que l'on appelle *stratégies de réduction*, c'est à dire méthodes formelles qui établissent quel redex choisir en présence de plusieurs. Plus formelle-

ment, une stratégie de réduction est une sous-relation de la $\beta$-réduction.

Un exemple de stratégie de réduction, très importante du point de vue théorique, est la *réduction gauche*. Dans cette stratégie, le redex choisi est toujours celui qui apparaît le plus à gauche dans le terme à réduire. Si $T$ se réduit en $U$ par réduction gauche, on écrira $T \succ V$ pour souligner la stratégie utilisée (même si évidemment il reste toujours vrai que $T \to U$, où $\to$ est la $\beta$-réduction conventionnelle). Voici par exemple la réduction d'un terme selon cette stratégie :

$$\begin{aligned}(\lambda xyz.x(yz))(\lambda x.x)(\lambda k.a)z &\succ (\lambda yz.(\lambda x.x)(yz))(\lambda k.a)z \\ &\succ (\lambda z.(\lambda x.x)((\lambda k.a)z))z \\ &\succ (\lambda x.x)((\lambda k.a)z) \succ (\lambda k.a)z \succ a\end{aligned}$$

Le résultat théorique fondamental concernant la réduction gauche, que l'on énoncera sans le démontrer, est le suivant :

**Théorème 2.1** *La réduction gauche d'un $\lambda$-terme $T$ termine ssi $T$ est normalisable, c.à.d. il existe un terme normal $N$ tel que $T \twoheadrightarrow N$.*

La réduction gauche est donc une stratégie « gagnante » pour la recherche de la forme normale d'un terme ; si elle existe, la réduction gauche va sûrement la trouver.

Les stratégies d'intérêt pratique les plus connues sont sans doute la *call by name* (CBN) et la *call by value* (CBV) ; dans la suite, nous en considérerons aussi des autres, notamment la *réduction de tête* et la *réduction linéaire de tête*.

## 2.1 La réduction de tête

Pour définir la réduction de tête on a besoin d'abord du résultat suivant :

**Proposition 2.2** *Tout $\lambda$-terme a la forme*

$$\lambda x_1 \dots x_m.TU_1 \dots U_n$$

*avec $m \geq 0$ et où $T$ (qu'on appellera* sous-terme de tête*) est soit une variable (inclue ou non parmi les $x_i$), et alors $n \geq 0$, soit une abstraction, et alors $n \geq 1$ ; les $U_j$ sont par contre des termes quelconque.*

**Preuve**. La preuve est une simple induction sur les termes du $\lambda$-calcul, qu'on ne se donnera pas la peine de faire. □

Dans le cas où le sous-terme de tête est une variable, on dira que $T$ est en *forme normale de tête*. Il est possible de montrer qu'un terme du $\lambda$-calcul est normal si et seulement si il est en forme normale de tête et tous les $U_j$ sont eux-mêmes normaux. En conséquence, un terme normal est un terme « récursivement » en forme normale de tête.

En revanche, si le sous-terme de tête de $T$ est l'abstraction $\lambda x.V$, alors, puisque $n \geq 1$, $T$ contient forcement au moins le redex $(\lambda x.V)U_1$, et il n'est donc pas normal. On appellera ce redex le *redex de tête* de $T$.

On a maintenant tous les éléments pour définir la réduction de tête :

$$\frac{}{(\lambda x.T)U \to_{\mathrm{N}} T\{U/x\}}\,\beta$$

$$\frac{T \to_{\mathrm{N}} T'}{TU \to_{\mathrm{N}} T'U}\,\mu$$

TAB. 2.1 – Les règles de réduction pour la *call by name*

**Définition 2.1 (Réduction de tête)** *La* réduction de tête *est la stratégie qui réduit toujours le redex de tête ; s'il n'y a pas de redex de tête, la stratégie ne fait aucune réduction. Si $T \to T'$ par réduction du redex de tête de $T$, on écrira $T \to_{\mathrm{H}} T'$.*

Un terme en forme normale de tête est donc normal par rapport à cette stratégie de réduction.

La réduction de tête est évidemment une sous-stratégie de la réduction gauche, car si un terme $T$ contient un redex de tête, ceci est forcement le redex le plus à gauche de $T$ ; par contre, l'existence d'un redex (et donc l'existence du redex le plus à gauche) n'implique nullement la présence du redex de tête (il suffit de penser à un terme non-normal mais en forme normale de tête). En effet, on peut voir la réduction gauche comme une « itération » de la réduction de tête.

La *call by name* (CBN) n'est rien d'autre qu'une sous-stratégie de la réduction de tête, appelée aussi *réduction de tête faible*. Comme le dit ce dernier nom, la CBN est un affaiblissement de la réduction de tête, qui consiste à réduire le redex de tête seulement si $m = 0$ (voir proposition 2.2), c'est à dire si le redex de tête ne se trouve au dessous d'aucune abstraction. Par exemple, le terme $\lambda z.(\lambda x.x)y$ est normal du point de vue de la réduction de tête faible.

La *call by name* peut se définir formellement en introduisant des *règles de réduction* sur les termes du $\lambda$-calcul, pareillement à ce qu'on a vu dans le cadre du $\pi$-calcul :

**Définition 2.2 (Réduction de tête faible (*Call by Name*))** *La* réduction de tête faible, *ou* call by name, *est la relation sur les termes du $\lambda$-calcul définie par les règles du tableau 2.1 ; on la notera par $\to_{\mathrm{N}}$.*

Dans la réduction de tête faible, on considère les termes du $\lambda$-calcul comme des fonctions appliquées à des arguments ; le terme le plus à gauche est la fonction, les autres à droite sont les arguments. Si un terme est une abstraction, on le considère comme une fonction à laquelle personne n'a donné d'arguments, et donc on n'a rien à réduire. En fait, la seule chose qu'on peut faire pour réduire un terme est de soumettre à la fonction (c.à.d. au sous-terme le plus à gauche) ces arguments, *sans possibilité de les évaluer avant*. C'est justement cela qui donne le nom *call by name* à la stratégie : les arguments sont passés aux fonctions *par nom*, leur valeur n'étant pas considérée.

Dans les langages de programmation il existe une autre possibilité pour ce qui concerne le passage des arguments aux fonctions, ce qu'on appelle *call by value* (CBV). Dans la CBV, on ne passe pas aux fonctions le « nom » des

$$\frac{V \text{ valeur}}{(\lambda x.T)V \to_{\mathrm{V}} T\{V/x\}}\,\beta_V$$

$$\frac{T \to_{\mathrm{V}} T'}{TU \to_{\mathrm{V}} T'U}\,\mu \qquad \frac{U \to_{\mathrm{V}} U'}{VU \to_{\mathrm{V}} VU'}\,\nu_V \ (V \text{ valeur})$$

TAB. 2.2 – Les règles de réduction pour la *call by value*

arguments (qui dans les implémentations réelles est habituellement un pointeur) mais plutôt leur *valeur*. Cela implique que les arguments doivent être évalués avant d'être passés à la fonction qui les attend.

Il est bien possible de formaliser tout cela dans le λ-calcul, au moyen d'une stratégie de réduction que l'on appelle justement *call by value* :

**Définition 2.3 (*Call by Value*)** *Appelons* valeurs *les λ-termes qui sont soit des variables, soit des abstractions, c.à.d., si $x$ est une variable, alors $x$ est un valeur, et $\lambda x.T$ est un valeur quel que soit le λ-terme $T$.*

*La réduction* call by value *est la relation sur les termes du λ-calcul, que l'on notera $\to_{\mathrm{V}}$, définie par les règles du tableau 2.2.*

On pourrait se demander si, entre CBN et CBV, il existe une stratégie « meilleure » que l'autre, par exemple en termes de nombre d'étapes nécessaires pour arriver à la forme normale. En absolu, la réponse est négative; aucune stratégie n'est « mieux » que l'autre. En particulier, la CBN se révèle plus efficace lorsqu'il y ait des *effacements* dans la réduction, c'est à dire lorsqu'il y ait des arguments qui ne sont pas utilisés par une fonction; en revanche, la CBV est plus efficace lorsqu'il y ait des *duplications* à faire, c'est à dire s'il y a des arguments qui sont utilisés plusieurs fois par une même fonction.

Les exemples suivants montrent les deux différentes situations. Soit $I = \lambda x.x$. Dans $(\lambda x.y)(III)$ l'argument $III$ n'est pas utilisé par la fonction $\lambda x.y$ ; en conséquence, la stratégie CBN prend moins d'étapes que la CBV pour arriver à la forme normale :

$$(\lambda x.y)(III) \to_{\mathrm{N}} y$$
$$(\lambda x.y)(III) \to_{\mathrm{V}} (\lambda x.y)(II) \to_{\mathrm{V}} (\lambda x.y)I \to_{\mathrm{V}} y$$

Ce phénomène est encore plus évident dans le terme $(\lambda x.y)(\Delta\Delta)$, où $\Delta = \lambda x.xx$ ; dans ce cas, la stratégie CBN trouve la forme normale en une seule étape, alors que la CBV continue la réduction à l'infini.

Considérons maintenant le terme $\Delta(III)$. Dans ce cas, l'argument $III$ est utilisé deux fois par $\Delta$, et donc c'est plus efficace de l'évalué avant de le soumettre à la fonction :

$$\Delta(III) \to_{\mathrm{N}} III(III) \to_{\mathrm{N}} II(III) \to_{\mathrm{N}} I(III) \to_{\mathrm{N}} III \to_{\mathrm{N}} II \to_{\mathrm{N}} I$$
$$\Delta(III) \to_{\mathrm{V}} \Delta(II) \to_{\mathrm{V}} \Delta I \to_{\mathrm{V}} II \to I$$

En modifiant l'argument de $\Delta$ en $IIy$, on arrive aussi à construire un exemple qui inverse en quelque sens la situation de l'exemple de non-termination que

l'on a donné ci-dessus ; en fait, dans ce cas la stratégie CBV arrive bien à la forme normale $yy$, tandis que la stratégie CBN s'arrête à une forme normale de tête qui n'est pas normale, à savoir $y(Iy)$.

## 2.2 La réduction linéaire de tête

La réduction linéaire de tête (rlt), introduite dans [DR96] et traitée de manière plus détaillée dans [DR03], est une stratégie de réduction inspirée par la normalisation des réseaux de preuve de la logique linéaire ([Gir87]). En vérité, la réduction linéaire de tête n'est pas une vraie stratégie de réduction, car ses étapes élémentaires ne sont pas des $\beta$-réductions. En particulier, la rlt n'arrive pas à trouver de termes normaux, et même pas de termes en forme normale de tête.

Pour introduire formellement la rlt il faut d'abord donner quelques définitions. Dans la suite, on notera par $[a_1, \ldots, a_n]$ la liste contenant les éléments $a_1, \ldots, a_n$ et dont l'élément de tête est $a_1$ ; la liste vide sera notée $[]$ et le constructeur de listes sera notée ::, c.à.d. $a :: [b] = [a, b]$. En outre, on supposera que chaque variable liée a un nom différent ; par exemple le terme $\Delta\Delta$ sera supposé être écrit $(\lambda x.xx)\lambda y.yy$. Ceci nous permettra d'indiquer avec des indices les différentes occurrences des variables liées ; par exemple, pour $\Delta\Delta$ on a $(\lambda x.x_0x_1)\lambda y.y_0y_1$. Finalement, par $T\langle U/x_i\rangle$ on notera le terme $T$ dans lequel on n'a substitué le terme $U$ qu'à la seule occurrence $x_i$ de $x$ ; par exemple

$$((\lambda xy.x_0(x_1y))\lambda z.z_0)\langle \lambda z.z/x_0\rangle = (\lambda xy.(\lambda z'.z'_0)(x_1y))\lambda z.z_0$$

où on a aussi effectué de l'$\alpha$-conversion automatiquement.

**Définition 2.4 (Sous-termes spinaux)** *Les* sous-termes spinaux *d'un $\lambda$-terme sont des sous-termes de $T$ définis récursivement de la façon suivante :*

- *$T$ est un sous-terme spinal de lui-même.*
- *Si $T = UV$ ou $T = \lambda x.U$, alors les sous-termes spinaux de $U$ sont des sous-termes spinaux de $T$ aussi.*

**Définition 2.5 (Liste des lambda de tête et redex premiers)** *La* liste des lambda de tête *d'un $\lambda$-terme $T$, dénotée $\lambda_h(T)$, est une liste $[\lambda x_1, \ldots, \lambda x_n]$ d'abstractions spinales de $T$ ; les* redex premiers *de $T$ sont des couples $(\lambda x, A)$ où $\lambda x$ est une abstraction spinale de $T$ et $A$ est un sous-terme argument de $T$. Les deux sont définis par induction sur $T$ :*

- *Si $T$ est une variable alors $\lambda_h(T)$ est vide et $T$ n'a aucun redex premier.*
- *Si $T = UV$ alors on a deux sous-cas :*
  - *Si $\lambda_h(U) = []$ alors $\lambda_h(T)$ est vide aussi et les redex premiers de $T$ sont ceux de $U$.*
  - *Si $\lambda_h(U) = \lambda x :: \rho$ alors $\lambda_h(T) = \rho$ et les redex premiers de $T$ sont ceux de $U$ avec l'ajout de $(\lambda x, V)$.*
- *Si $T = \lambda x.U$ alors $\lambda_h(T) = \lambda x :: \lambda_h(U)$ et les redex premiers de $T$ sont ceux de $U$.*

**Définition 2.6 (Ovette et hoc redex)** *Chaque $\lambda$-terme $T$ a exactement un sous-terme spinal qui est une occurrence de variable ; on dénotera cette occurrence $v_h(T)$ et on l'appellera* ovette *(en anglais* head occurrence*, hoc). Soit maintenant $T$ un terme tel que $v_h(T) = x_0$ ; alors le redex premier $(\lambda x, A)$, s'il existe, sera appelé* hoc redex.

Chaque terme contient au plus un hoc redex ; si $T$ ne contient pas d'hoc redex, on dira que $T$ est en *forme quasiment-normale de tête.*

On est enfin en position de définir la réduction linéaire de tête :

**Définition 2.7 (Réduction linéaire de tête)** *Soit $T$ un $\lambda$-terme n'étant pas en forme quasiment-normale de tête, et soient $(\lambda x, A)$ et $x_0$ resp. l'hoc redex et l'ovette de $T$. Alors, le réduit $T'$ de $T$ par réduction linéaire de tête est le terme*

$$T' = T\langle A/x_0\rangle$$

*Autrement dit, on substitue $A$ à l'ovette et on laisse le reste de $T$ inchangé. On notera cette opération avec $T \to_{\mathrm{HL}} T'$.*

Voici la réduction du terme $(\lambda f.f(fx))\lambda y.y$ par réduction linéaire de téte :

$$\begin{array}{rcl}(\lambda f.f(fx))\lambda y.y & \to_{\mathrm{HL}} & (\lambda f.(\lambda y'.y')(fx))\lambda y.y \\ & \to_{\mathrm{HL}} & (\lambda f.(\lambda y'.fx)(fx))\lambda y.y \\ & \to_{\mathrm{HL}} & (\lambda f.(\lambda y'.(\lambda y''.y'')x)(fx))\lambda y.y \\ & \to_{\mathrm{HL}} & (\lambda f.(\lambda y'.(\lambda y''.x)x)(fx))\lambda y.y\end{array}$$

Remarquons d'abord que la taille des termes ne décroît jamais par rlt ; elle peut à la limite rester la même lorsqu'on substitue l'ovette par une autre occurrence de variable. En outre, on voit bien qu'aucune étape de la réduction ne correspond à une étape de $\beta$-réduction, mais pourtant les réduits sont tous $\beta$-équivalents.

On peut en effet montrer le résultat suivant, qui relie la réduction linéaire de tête à la réduction de tête :

**Théorème 2.3 (Danos-Regnier)** *Si $T'$ est le réduit de $T$ par rlt, alors $T$ et $T'$ sont $\beta$-équivalents.*

*Si $T$ est en forme quasimant-normale de tête et $n$ est le nombre de ses redex premiers, alors la réduction de tête conduira à une forme normale de tête dans exactement $n$ étapes.*

*Si $T$ est un terme quelconque, la réduction linéaire de tête de $T$ termine si et seulement si la réduction de tête de $T$ termine.*

**Preuve**. Voir le théorème 1 de [DR03]. □

La dynamique de la réduction linéaire de tête est capturée par sa *suite de substitutions* :

**Définition 2.8 (Suite de substitutions)** *Soit $T = T_0$ un terme et $T_1, T_2, \ldots$ la suite (éventuellement infinie) de ses réduits successifs par rlt. On appelle* suite de substitutions *de $T$, dénotée $Sub_\lambda(T)$, la liste $[(z_0, A_1), (z_1, A_2), \ldots]$ où les $z_i$ et les $A_i$ sont respectivement des occurrences de variables de $T$ et des sous-termes argument de $T$ tels que pour tout $i$ l'ovette de $T_i$ est un résidu de $z_i$ et l'argument de l'hoc redex de $T_i$ est un résidu de $A_{i+1}$.*

Dans l'exemple ci-dessus, si on écrit le terme à réduire comme $(\lambda f.f_0(f_1x))\lambda y.y_0$, la suite de substitutions est

$$[(f_0, \lambda y.y), (y_0, fx), (f_1, \lambda y.y), (y_0, x)]$$

Comme le montre le théorème 2.3, la rlt est donc beaucoup relié à la réduction de tête. Toutefois, si on considère la version faible de la réduction

de tête (i.e. la stratégie CBN), ce n'est pas difficile de montrer que les deux sont orthogonales sous certains aspects. Par exemple, considérons le terme $T = \lambda x.(\lambda yz.yz)A$ ; on a

$$\lambda x.(\lambda yz.yz)A \rightarrow_{\mathrm{HL}} \lambda x.(\lambda yz.Az)A$$

et donc il y a au moins $(y, A)$ dans la suite de substitutions de $T$. Par contre, puisque $T$ est une abstraction, la réduction de tête faible ne fait aucune substitution.

On peut alors définir la version de la réduction linéaire de tête correspondant à la *call by name* :

**Définition 2.9 (Réduction linéaire de tête faible)** *La* réduction linéaire de tête faible*, que l'on notera* $\rightarrow_{\mathrm{WHL}}$*, est une sous-relation de* $\rightarrow_{\mathrm{HL}}$ *tel que* $T \rightarrow_{\mathrm{WHL}} T'$ *ssi* $T \rightarrow_{\mathrm{HL}} T'$ *et* $\lambda_h(T) = []$.

Donc si la liste des lambda de tête d'un terme $T$ n'est pas vide, $T$ est normal par rapport à la réduction linéaire de tête faible, et cela implique évidemment que $Sub_\lambda(T) = []$ (dans la suite on utilisera la notation $Sub_\lambda(\cdot)$ aussi pour la suite de substitutions de la réduction linéaire de tête *faible* d'un terme, car à partir de ce moment on ne considérera pratiquement que cette version de la stratégie).

# Chapitre 3

# Traduire le $\lambda$-calcul dans le $\pi$-calcul

## 3.1 La traduction de Milner

Dans [Mil92], Robin Milner a introduit la première traduction du $\lambda$-calcul dans le $\pi$-calcul, en montrant que ce dernier, même si équipé d'une règle de réduction dont la substitution semble plus « faible » que celle du $\lambda$-calcul, est en vérité de la même puissance expressive.

Il faut cependant dire que le $\pi$-calcul ne peut pas, au moins apparemment, simuler la dynamique de réduction du $\lambda$-calcul de façon complète ; en fait, quand on parle de « traduire » le $\lambda$-calcul dans le $\pi$-calcul, on est vraiment en train de dire que c'est possible de simuler dans le $\pi$-calcul *une certaine stratégie de réduction* du $\lambda$-calcul. En particulier, les stratégies considerées au debut par Milner furent les deux les plus connues, c'est à dire la *call by name* et la *call by value*. Aujourd'hui, on sait que pratiquement toute stratégie définissable dans le $\lambda$-calcul peut être simulée dans le cadre du $\pi$-calcul (voir [SW01] pour une exposition détaillée des différents résultats trouvés jusqu'à présent). Des deux traductions fournies dans [Mil92], on est intéressés à la traduction de la stratégie *call by name*, car cette dernière est très liée à la réduction linéaire de tête et à la machine de Krivine, lien qui sera établi aussi pour le $\pi$-calcul dans la prochaine section.

On introduira alors dans la suite la traduction de Milner CBN, que l'on notera $[\![\cdot]\!]$, i.e. si $T$ est un $\lambda$-terme, on définira une procédure pour construire un processus $[\![T]\!]$ du $\pi$-calcul dont la réduction simule l'évaluation de $T$ sous la stratégie *call by name*.

Les processus que l'on obtiendra seront en vérité dépendants d'un paramètre ; plus formellement, si $\mathsf{fv}(T) = A$, on aura $\mathsf{fn}([\![T]\!]) = A \cup \{u\}$, où $u$ est un nom correspondant à une variable du $\lambda$-calcul que l'on suppose jamais utilisé dans aucun $\lambda$-terme. En fait, à partir de maintenant on supposera que l'ensemble des noms du $\pi$-calcul est partitionné en deux sous-ensembles, l'un contenant des noms qu'on notera $x, y, z, \ldots$ et qu'on considérera coïncidant avec l'ensemble des variables du $\lambda$-calcul, l'autre contenant des noms qu'on notera $u, v, w, \ldots$ et qui évidemment sera complètement disjoint de l'ensemble des $\lambda$-variables. A ce moment, pour souligner la présence de ce paramètre dans les traductions des

$\lambda$-termes, on ajoutera souvent le nom qui représente le paramètre juste à côté du processus, comme par exemple $[\![T]\!]u$ ; dans ce cas, l'écriture $[\![T]\!]v$ dénotera le même processus auquel on a remplacé toute occurrence de $u$ avec $v$. Ce paramètre peut être vu comme le « canal de communication » qui est l'objet de l'interaction du processus avec les autres processus ; dans $[\![T]\!]u$, soit $u$ est le nom par lequel $[\![T]\!]$ peut envoyer ou recevoir quelque chose, soit il est l'objet même de la communication, c'est à dire $u$ est le nom envoyé par $[\![T]\!]$ lorsqu'il y a la possibilité pour $[\![T]\!]$ d'interagir.

Commençons par définir un processus particulier, qui n'est la traduction d'aucun $\lambda$-terme mais qui sera fondamental pour la définition de la traduction. On appellera ce processus *assignation* :

$$[x := T] \stackrel{\text{def}}{=} !x(u).[\![T]\!]u$$

où $T$ est un $\lambda$-terme et $x$ une variable. La signification de l'assignation est la suivante : la traduction du terme $T$ est mise à disposition des autres processus au moyen d'une « requête » envoyée sur le canal $x$. Le terme $T$ est donc devenu une espèce de ressource assignée (ou « localisée ») à $x$, c'est à dire une ressource dont l'utilisation peut être demandée en envoyant un nom le long du canal $x$ ; ce nom sera en suite le moyen de communication utilisé par le processus qui a eu accès à $T$ et par $T$ lui-même.

Ceci est un exemple typique du genre de mobilité exprimable dans le $\pi$-calcul : un processus serveur est en attente d'une requête par un processus client ; ce dernier, en connaissant le canal de communication sur lequel le serveur attend un signal, envoie sur ce canal une requête à laquelle le serveur répond avec une sorte de « clé » (un autre canal de communication) qui donne accès à une copie de la ressource fourni par le serveur lui-même. A ce moment, le client peut exploiter la ressource en utilisant le canal reçu, et quelqu'un d'autre peut envoyer une autre requête au serveur, qui doit donc toujours être capable de fournir des copie de sa ressource ; ceci justifie la présence de la réplication dans la définition du processus assignation[1].

La traduction de Milner CBN $[\![T]\!]u$ d'un $\lambda$-terme $T$ est à ce moment définissable par induction sur $T$ :

$$\begin{aligned}
[\![x]\!]u &\stackrel{\text{def}}{=} \overline{x}u \\
[\![\lambda x.T]\!]u &\stackrel{\text{def}}{=} u(x,v).[\![T]\!]v \\
[\![TU]\!]u &\stackrel{\text{def}}{=} \boldsymbol{\nu}(z,v).\left([\![T]\!]v \mid \overline{v}\langle z,u\rangle.[z := U]\right)
\end{aligned}$$

Dans la traduction de l'application, $z$ est un nom « frais », n'apparaissant libre ni dans $T$ ni dans $U$ ($v$ l'est automatiquement à cause de la partition conventionnelle que l'on à faite sur les noms du $\pi$-calcul).

La raison à la base de ce choix pour la traduction des $\lambda$-termes sera plus claire après les remarques suivants. Considérons le redex $(\lambda x.T)U$. Sa traduction est

$$[\![(\lambda x.T)U]\!]u = \boldsymbol{\nu}(z,v).\left(v(x,w).[\![T]\!]w \mid \overline{v}\langle z,u\rangle.[z := U]\right)$$

Après une étape d'interaction on arrive à

$$\boldsymbol{\nu}z.\left([\![T]\!]\{z/x\}u \mid [z := U]\right)$$

[1] Si on n'utilise pas la réplication dans l'assignation, on obtient une traduction du $\lambda$-calcul *affine*.

et donc, à $\alpha$-équivalence près, on a

$$[\![(\lambda x.T)U]\!]u \longrightarrow \boldsymbol{\nu} x.\,\big([\![T]\!]u \;\big|\; [x := U]\big)$$

La traduction du redex $(\lambda x.T)U$ se réduit donc en un processus dans lequel la réduction peut continuer dans $T$ et en même temps il y a eu la création d'une nouvelle ressource, qui a assigné l'argument $U$ à la variable $x$. Autrement dit, la substitution de $U$ à toute occurrence liée de $x$ qui aurait eu lieu avec la $\beta$-réduction est simulée par l'apparition de l'assignation $[x := U]$. Jusqu'à ici, on n'a encore fait aucune vraie substitution ; on a tout simplement « enregistré » le lien entre le terme $U$ et la variable qui le représentera à partir de ce moment.

La vraie substitution a lieu lorsqu'on tombe sur la traduction de la variable $x$ elle-même. Celle-ci est, à l'intérieur de $[\![T]\!]u$, le sous-processus $\overline{x}v$ (où $v$ sera un nom privé introduit par une application ou bien le nom $u$ lui-même) : il s'agit donc d'un processus qui envoie un nom le long du nom $x$. Cela peut être vue comme une « recherche » de la valeur de $x$ ; en fait, si pendant l'évaluation *call by name* d'un terme on tombe sur une variable, on en doit forcement connaître la valeur, sinon l'exécution va s'arrêter[2]. C'est exactement cela qui se passe dans la traduction de Milner : $\overline{x}v$ essaie d'envoyer un nom sur le canal $x$ ; s'il n'y a personne à l'écoute, il n'y a pas d'interaction possible et l'exécution termine. Sinon, s'il y a eu (comme dans l'exemple) une interaction qui a « créé » l'assignation $[x := U]$, alors il y a bien quelqu'un qui peut recevoir le nom envoyé par $\overline{x}v$ ; ce quelqu'un là est justement le processus d'assignation $!x(w).[\![U]\!]w$, qui interagit en mettant à disposition une copie de $[\![U]\!]$, laquelle peut utiliser le nom $v$ pour continuer l'exécution du terme.

Il n'y a donc que deux types d'interaction possibles dans un processus du $\pi$-calcul qui résulte de la traduction d'un $\lambda$-terme :

**Définition 3.1 ($\lambda$-assignation et $\lambda$-substitution)** *Dans un processus du $\pi$-calcul qui est la traduction d'un $\lambda$-terme, les deux types d'interaction que sa réduction comporte seront appelés comme suit :*

**$\lambda$-assignation** *C'est l'étape de réduction dans lequel la traduction d'une abstraction interagit avec la traduction de l'argument d'une application :*

$$\boldsymbol{\nu}(z, v, \ldots).\,\big(v(x,w).P \;\big|\; \overline{v}\langle z, u\rangle.\,[z := A] \;\big|\; \ldots\big) \longrightarrow$$

$$\longrightarrow \boldsymbol{\nu}(z, \ldots).\,\big(P\{z/x, u/w\} \;\big|\; [z := A] \;\big|\; \ldots\big)$$

*Après cette interaction, on dira que $z$ est le* résidu *de $x$, et toute occurrence de $z$ sera donc un* résidu *d'une occurrence de $x$.*

**$\lambda$-substitution** *C'est l'étape de réduction dan lequel la traduction d'une variable interagit avec un processus d'assignation :*

$$\boldsymbol{\nu}(z, \ldots).\,\big(\overline{z}u \;\big|\; [z := A] \;\big|\; \ldots\big) \longrightarrow \boldsymbol{\nu}(z, \ldots).\,\big([\![A]\!]u \;\big|\; [z := A] \;\big|\; \ldots\big)$$

*A chaque étape de $\lambda$-substitution qui a lieu à l'intérieur de la traduction d'un $\lambda$-terme $T$ on peut associer un couple formée par une occurrence de variable de $T$ et par un sous-terme argument de $T$, de la manière suivante : si l'occurrence de $z$ dans $\overline{z}u$ est le résidu de l'occurrence $x_0$ de $x$ dans $T$, on assignera à cette $\lambda$-substitution le couple $(x_0, A)$.*

[2] Rappelons qu'un terme du genre $xU_1 \ldots U_n$ est normal du point de vue de la CBN, car la valeur de $x$ est inconnue et on ne sait pas comment se comporter par rapport aux arguments $U_1, \ldots, U_n$.

*Si le processus $P$ se réduit en $Q$ au moyen d'un nombre quelconque de $\lambda$-assignations, mais sans effectuer aucune $\lambda$-substitution, on écrira $P \rightsquigarrow Q$.*

L'opération de substitution du $\lambda$-calcul est donc remplacée ici par deux opérations plus élémentaires : la $\lambda$-assignation, qui prévoit des substitutions de noms par des autres noms (et non pas de variables par des termes) et qui cause l'assignation d'un terme à un nom, et la $\lambda$-substitution, qui récupère le terme assigné pour son utilisation ultérieure.

La relation entre la traduction $[\![\cdot]\!]$ et le $\lambda$-calcul *call by name* est donnée par le résultat suivant :

**Théorème 3.1 (Milner)** *Pour tout $\lambda$-terme $T$, la réduction de $[\![T]\!]u$ est* déterministe *(c.à.d. il y a au plus une réduction possible à chaque étape), et on a l'un des deux cas suivants :*

**i.** $T \to_{\mathrm{N}} T'$ *et* $[\![T]\!]u \longrightarrow P'$, *où*

$$T' \simeq_\alpha U\{A_1/z_1, \ldots, A_n/z_n\}$$

*et*

$$P' \equiv \boldsymbol{\nu}(z_1, \ldots, z_n).\,\big([\![U]\!]u \mid [z_1 := A_1] \mid \ldots \mid [z_n := A_n]\big)$$

**ii.** *La réduction CBN de $T$ et la réduction de $[\![T]\!]u$ divergent.*

**Preuve**. Voir le théorème 4.6 de [Mil92]. □

La correspondance avec la stratégie CBN n'est donc pas tout à fait exacte. En général, la simulation de l'exécution *call by name* d'un terme $T$ n'arrive pas à sa forme normale $U$ (bien entendu, normale par rapport toujours à la CBN) ; au contraire, elle laisse des substitution à faire, en faisant lesquelles on peut finalement obtenir $U$. La correspondance aurait été exacte si on avait $[\![T]\!]u \longrightarrow [\![U]\!]u$, mais cela n'est pas garanti par le théorème 3.1, et ce n'est pas difficile de vérifier que ce n'est pas vrai en général.

En revanche, la dynamique qui se déroule dans le $\pi$-calcul lorsqu'on simule la réduction d'un $\lambda$-terme avec la traduction de Milner est beaucoup plus proche d'une autre stratégie d'exécution que l'on a vue : la réduction linéaire de tête. En fait, dans la prochaine section on s'occupera de montrer l'existence d'une correspondance exacte entre l'exécution des $\lambda$-termes simulée par la traduction de Milner et la réduction linéaire de tête faible.

## 3.2 Le $\pi$-calcul et la réduction linéaire de tête

On va d'abord définir l'analogue dans le $\pi$-calcul de la suite de substitutions de la réduction linéaire de tête faible :

**Définition 3.2 (Suite de $\lambda$-substitutions)** *La* suite de $\lambda$-substitutions *d'un processus $[\![T]\!]$, dénotée $Sub_\pi(T)$ et souvent appelée tout simplement* suite de substitutions, *est une liste $[(z_0, A_0), (z_1, A_1), \ldots]$ contenant les couples associées à chaque étape de $\lambda$-substitution effectuée dans l'exécution de $[\![T]\!]$ (définition 3.1), dans l'ordre où ces substitutions ont lieu.*

Remarquons que la suite de $\lambda$-substitutions est bien définie grâce au théorème 3.1, qui garantit que la réduction de la traduction d'un $\lambda$-terme est déterministe, et donc il n'y qu'un seul ordre possible pour les $\lambda$-substitutions.

Dans la suite, on s'occupera de montrer que, pour n'importe quel $\lambda$-terme $T$, $Sub_\lambda(T) = Sub_\pi(T)$ (où $Sub_\lambda(T)$ est la suite de substitutions de $T$ par réduction de tête *faible*).

Commençons par une définition :

**Définition 3.3 (Hauteur et sous-termes de hauteur $i$)** *On définit la hauteur d'un $\lambda$-terme $T$ (notée $h(T)$) par récurrence sur $T$ :*

· *Si $T$ est une variable, $h(T) = 0$.*

· *Si $T$ est une abstraction, c'est à dire $T = \lambda x.U$, alors*

$$h(T) = \begin{cases} h(U) & \text{si } U = \lambda y.V \\ h(U) + 1 & \text{si } U = VW \text{ ou } U = y \end{cases}$$

· *Si $T$ est une application, c'est à dire $T = UV$, alors $h(T) = h(U)$.*

*Si $i$ est un entier non négatif plus petit ou égal à $h(T)$, on appellera* sous-terme de hauteur $i$ *de $T$ le plus grand sous-terme spinal $U$ de $T$ tel que $h(U) = i$ ; on notera ce terme $T^{(i)}$.*

Intuitivement, la hauteur d'un terme $T$ est le nombre d'alternances $\lambda$-@ ou $\lambda$-variable contenues dans l'épine dorsale de $T$. Sur la structure des sous-termes de hauteur $i$ d'un $\lambda$-terme on peut montrer la proposition suivante :

**Proposition 3.2** *Soit $T$ un $\lambda$-terme tel que $h(T) = n$. Alors on a*

$$T^{(0)} = xA_{0,1} \dots A_{0,k_0}$$

*et, pour tout entier $i$ positif plus petit ou égal à $n$,*

$$T^{(i)} = (\lambda x_{i,1} \dots x_{i,p_i}.T^{(i-1)})A_{i,1} \dots A_{i,k_i}$$

*où les $p_i$ et les $k_i$ sont tous non-nuls sauf éventuellement $k_0$ et $k_n$.*

**Preuve**. Par récurrence sur la hauteur de $T$. Si $h(T) = 0$, c'est clair que la forme de $T^{(0)}$ (qui dans ce cas est $T$ lui-même) ne peut qu'être celle donnée dans la thèse, avec $k_0$ éventuellement nul. En revanche, si $h(T) = n+1$, on sait par hypothèse de récurrence que $T^{(n)}$ est de la forme voulue ; or $T$ ne peut pas être une variable, donc il n'y a que les deux possibilités suivantes :

$\lambda$. $T$ commence par une abstraction, et donc sa forme la plus générale est $T = T^{(n+1)} = \lambda x_1 \dots x_p.U$, où $U$ est un terme qui ne commence pas par une abstraction ; mais alors $U$ est évidemment le plus grand sous-terme spinal de $T$ tel que $h(U) = n$, donc $U = T^{(n)}$, et la thèse est vérifiée avec $k_{n+1} = 0$ (rappelons que $k_{h(T)}$ peut bien être nul).

@. $T$ commence par une application, et il est donc de la forme $(\lambda x_1 \dots x_p.U)A_1 \dots A_k$, avec $p, k > 0$ et où $U$ est encore une fois un terme qui ne commence pas par une abstraction ; mais alors on a toujours, par définition, $U = T^{(n)}$.

□

**Corollaire 3.3** *Tout terme du $\lambda$-calcul s'écrit de la forme $T^{(n)}$, où $T^{(n)}$ est un terme défini par récurrence de la façon suivante :*

$$T^{(0)} = xA_{0,1} \dots A_{0,k_0}$$
$$T^{(i+1)} = (\lambda x_{i+1,1} \dots x_{i+1,p_{i+1}}.T^{(i)})A_{i+1,1} \dots A_{i+1,k_{i+1}}$$

*où $n$ est la hauteur du terme, $x$ une variable, les $A_{i,j}$ des termes quelconques et les $p_i$ et les $k_i$ des entiers strictement positif sauf éventuellement $k_0$ et $k_n$.*

Les suites d'entiers que l'on a jusqu'à ce moment appelées $p_i$ et $k_i$ caractérisent la structure de l'épine dorsale des termes du $\lambda$-calcul ; on va donc définir une manière synthétique pour les indiquer :

**Définition 3.4** *Soient $T$ un terme de hauteur $n$, $1 \leq i \leq n$ et*

$$T^{(0)} = xA_{0,1} \dots A_{0,k_0}$$

*et*

$$T^{(i)} = (\lambda x_{i,1} \dots x_{i,p_i}.T^{(i-1)})A_{i,1} \dots A_{i,k_i}$$

*respectivement les sous-termes de hauteur zéro et $i$ de $T$. On pose, par définition,*

$$\sharp^0_\lambda(T) = 0$$
$$\sharp^i_\lambda(T) = p_i$$

*et*

$$\sharp^0_@(T) = k_0$$
$$\sharp^i_@(T) = k_i$$

*En outre, pour tout $j$ tel que $0 \leq j \leq n$, on pose*

$$\sharp^j_\Delta(T) = \sharp^j_@(T) - \sharp^j_\lambda(T)$$

Cette notation a été choisie en considérant le fait que $\sharp^i_\lambda(T)$ (resp. $\sharp^i_@(T)$) est le nombre de $\lambda$ (resp. @) consécutifs qui se trouvent dans l'épine dorsale de $T$ à la hauteur $i$.

Normalement, pour un terme quelconque $T$, tout ce qu'on peut dire sur les suites $\sharp_\lambda$ et $\sharp_@$ est que, pour $1 \leq i \leq h(T)$, $\sharp^i_\lambda(T) \geq 1$, tandis que pour $1 \leq i \leq h(T)-1$, $\sharp^i_@(T) \geq 1$ mais par contre $\sharp^0_@(T), \sharp^{h(T)}_@(T) \geq 0$. Au contraire, on va voir maintenant que si $T$ est un terme dont la liste des lambda de tête est vide, les deux suites ont une propriété bien particulière :

**Lemme 3.4** *Soit $T$ un $\lambda$-terme. Alors, $\lambda_h(T)$ est vide ssi pour tout entier $i$ tel que $0 \leq i \leq h(T)$, on a*

$$\sum_{j=i}^{h(T)} \sharp^j_\Delta(T) \geq 0$$

**Preuve**. Par induction sur la hauteur de $T$. Si $h(T) = 0$, $T$ ne contient pas d'abstractions, donc sa liste des lambda de tête est toujours vide, et comme on a de même toujours $\sharp^0_\Delta(T) \geq 0$, la thèse est vérifiée de façon triviale. Soit maintenant $h(T) = n+1$, et considérons $T^{(n)}$, dont la hauteur est par définition $n$. Si $\lambda_h(T^{(n)})$ est vide, alors par hypothèse d'induction on a que quel que soit $0 \leq i \leq n$, $\sum_{j=i}^{n} \sharp^j_\Delta(T) \geq 0$, et puisque on sait que $\lambda_h(T)$ est aussi vide, on est sûr qu'au dessus de $T^{(n)}$ il y a au moins autant d'applications que d'abstractions, c'est à dire $\sharp^{n+1}_@(T) \geq \sharp^{n+1}_\lambda(T)$, qui implique évidemment $\sharp^{n+1}_\Delta(T) \geq 0$, et donc, quel que soit $0 \leq i \leq n+1$,

$$\sum_{j=i}^{n+1} \sharp^j_\Delta(T) = \sharp^{n+1}_\Delta(T) + \sum_{j=i}^{n} \sharp^j_\Delta(T) \geq 0$$

Par contre, si $\lambda_h(T^{(n)})$ n'est pas vide, alors par hypothèse d'induction il existe un $i$ (ou bien plusieurs) tel que $\sum_{j=i}^{n} \sharp_\Delta^j(T) < 0$, c'est à dire il y a des endroit de l'épine dorsale de $T^{(n)}$ à partir desquels, en remontant l'épine elle-même, on trouve plus d'abstractions que d'applications. Les abstractions en excès sont exactement celles qui n'ont pas d'application correspondante, et qui sont donc dans la liste des lambda de tête de $T^{(n)}$. Evidemment la longueur de cette liste borne la différence entre les applications et les abstractions comptées à partir de n'importe quel endroit de l'épine dorsale de $T^{(n)}$. Si $m$ est la longueur de $\lambda_h(T^{(n)})$, on a alors, pour tout $i$,

$$\sum_{j=i}^{n} \sharp_\Delta^j(T) \geq -m$$

En revenant à notre terme de départ, comme $\lambda_h(T)$ est en tout cas bien vide, on déduit que dans l'épine dorsale de $T$ au dessus de $T^{(n)}$ il y a au moins une application pour chaque abstraction qui se trouve dans $\lambda_h(T^{(n)})$ et au moins une application pour chaque nouvelle abstraction; en termes des suites $\sharp_\lambda$ et $\sharp_@$, on a

$$\sharp_@^{n+1}(T) = \sharp_\lambda^{n+1}(T) + m + a$$

pour un entier $a$ non négatif. On a donc, pour tout $i$ (tel que $0 \leq i \leq n+1$),

$$\sum_{j=i}^{n+1} \sharp_\Delta^j(T) = \sharp_\Delta^{n+1}(T) + \sum_{j=i}^{n} \sharp_\Delta^j(T) \geq m + a - m = a \geq 0$$

qui vérifie notre thèse. □

**Remarque 3.1** *A partir de l'inégalité qu'on vient de montrer, on peut en déduire une autre :*

$$\sharp_\lambda^i(T) \leq \sharp_@^i(T) + \sum_{j=i+1}^{h(T)} \sharp_\Delta^j(T)$$

*qui est vraie pour tout terme $T$ vérifiant $\lambda_h(T) = []$ et pour tout $i$ tel que $0 \leq i \leq h(T)$. Celle-ci sera la version de l'inégalité qu'on utilisera dans la suite.*

Revenons à la traduction de Milner. Le corollaire 3.3 nous donne une nouvelle façon de regarder les processus du $\pi$-calcul qui dérivent d'un $\lambda$-terme; en fait, la décomposition récursive énoncée dans ce corollaire s'applique directement à la traduction, en produisant immédiatement un autre corollaire :

**Corollaire 3.5** *Si $T$ est un $\lambda$-terme de hauteur $n$, alors on peut décomposer sa traduction $[\![T]\!]u$ en exactement $n+1$ « niveaux » $P_T^{(0)}u_0, \ldots, P_T^{(n)}u_n$, avec $u_n = u$ et $P_T^{(n)}u = [\![T]\!]u$, définis de la façon suivante :*

$$\begin{aligned} P_T^{(0)}u_0 \quad = \quad & \boldsymbol{\nu}(\vec{z}_0, \vec{v}_0).(\overline{x}v_{0,1} \mid \overline{v_{0,1}}\langle z_{0,1}, v_{0,2}\rangle.[z_{0,1} := A_{0,1}] \mid \ldots \\ & \ldots \mid \overline{v_{0,k_0}}\langle z_{0,k_0}, u_0\rangle.[z_{0,k_0} := A_{0,k_0}]) \end{aligned}$$

$$\begin{aligned} P_T^{(i)}u_i \quad = \quad & \boldsymbol{\nu}(\vec{z}_i, \vec{v}_i).(v_{i,1}(x_{i,1}, w_2).w_2(x_{i,2}, w_3)\ldots w_{p_i}(x_{i,p_i}, u_{i-1}).P_T^{(i-1)}u_{i-1} \mid \\ & \mid \overline{v_{i,1}}\langle z_{i,1}, v_{i,2}\rangle.[z_{i,1} := A_{i,1}] \mid \ldots \mid \overline{v_{i,k_i}}\langle z_{i,k_i}, u_i\rangle.[z_{i,k_i} := A_{i,k_i}]) \end{aligned}$$

*où $1 \leq i \leq n$, $k_i = \sharp_@^i(T)$, $p_i = \sharp_\lambda^i(T)$ et les $A_{i,j}$ sont des $\lambda$-termes.*

**Remarque 3.2** *Le corollaire qu'on vient d'énoncer est valide à cause de la définition récursive de la traduction de Milner et du corollaire 3.3, qui donne aussi une décomposition récursive des $\lambda$-termes. C'est donc clair que le sous-processus au niveau $i$ de $[\![T]\!]$ correspond à la traduction du sous-terme de hauteur $i$, c'est à dire $P_T^{(i)} u = [\![T^{(i)}]\!] u$.*

Pour montrer finalement notre proposition, c'est à dire que la traduction de Milner en vérité simule la réduction linéaire de tête faible, démontrons d'abord quelques résultats préliminaires :

**Lemme 3.6** *Soit $T$ un $\lambda$-terme. Alors, si $\lambda_h(T) = []$, on a*

$$[\![T]\!] u \rightsquigarrow \boldsymbol{\nu}(\vec{z}, \vec{v}).\left([\![T^{(0)}]\!] v \mid \ldots \mid \overline{v_j}\langle z_j, w_j\rangle.\,[z_j := A_j] \mid \ldots \mid [z_l := A_l] \mid \ldots\right)$$

*où :*

- $1 \leq j \leq \sum_{m=0}^{h(T)} \sharp_{\Delta}^{m}(T)$*, tandis que dans le cas où la somme est nulle, il n'y a pas de sous-processus de la forme* $\overline{v_j}\langle z_j, w_j\rangle.\,[z_j := A_j]$
- $1 \leq l \leq \sum_{m=0}^{h(T)} \sharp_{\lambda}^{m}(T)$*, tandis que dans le cas où la somme est nulle, il n'y a pas de sous-processus de la forme* $[z_l := A_l]$
- $v \in \{u, \vec{v}\}$
- $w_j \in \{u, \vec{v}\} \setminus \{v, v_j\}$
- *Les $A_j$ et les $A_l$ sont des sous-termes argument de $T$*

*En revanche, si $\lambda_h(T) \neq []$, l'exécution de $[\![T]\!]$ n'arrive pas au niveau zéro, mais elle s'arrête avec un processus de la forme*

$$\boldsymbol{\nu}(\vec{z}, \vec{v}).\left(v(x, w).Q \mid \ldots \mid [z_l := A_l] \mid \ldots\right)$$

*où il n'existe pas de sous-processus préfixés par $\overline{v}\langle z, v'\rangle$ qui puissent interagir avec $v(x, w).Q$.*

**Preuve**. Supposons d'abord que la liste des lambda de tête soit vide. Par le corollaire 3.5, la réduction de chaque niveau $i$ de $[\![T]\!]u$ (pour $1 \leq i \leq h(T)$) commence avec un processus qui contient :

- $\sharp_{\lambda}^{i}(T)$ préfixes récepteurs sous lesquels on trouve $[\![T^{(i-1)}]\!]$ ;
- $\sharp_{@}^{i}(T) + \sum_{j=i+1}^{h(T)} \sharp_{\Delta}^{j}(T)$ sous-processus émetteurs ;
- $\sum_{j=i+1}^{h(T)} \sharp_{\lambda}^{j}(T)$ sous-processus d'assignation.

Sous l'hypothèse que la liste des lambda de tête de $T$ soit vide, par le lemme 3.4 on a que, pour tout $i$ (voir remarque 3.1),

$$\sharp_{\lambda}^{i}(T) \leq \sharp_{@}^{i}(T) + \sum_{j=i+1}^{h(T)} \sharp_{\Delta}^{j}(T)$$

et donc il y a toujours un nombre suffisant d'émetteurs capables d'interagir avec les récepteurs et de « libérer » $[\![T^{(i-1)}]\!]$ de ses préfixes gardiens, de manière telle que l'exécution puisse continuer au niveau successif (c'est à dire $i-1$). Le fait que la liste des lambda de tête soit vide garantit donc que la réduction arrive jusqu'au sous-processus de niveau zéro, en effectuant évidemment seulement des $\lambda$-assignations.

Au contraire, si on suppose que la liste des lambda de tête ne soit pas vide, encore par le lemme 3.4 on déduit qu'il existe un niveau $i$ tel que

$$\sharp^i_\lambda(T) > \sharp^i_@(T) + \sum_{j=i+1}^{h(T)} \sharp^j_\Delta(T)$$

Une fois que l'exécution arrive à ce niveau, on ne trouve pas assez d'émetteurs pour consommer tous les préfixes qui cachent $[\![T^{(i-1)}]\!]$; en conséquence, la réduction termine avec un processus de la forme donnée dans la thèse. □

**Lemme 3.7** *Si $(\lambda x, A)$ est un redex premier de $T$, alors $[\![T]\!]u$ contient un sous-processus structurellement équivalent à un processus de la forme*

$$\boldsymbol{\nu}(z, \vec{z}, v, \vec{v}).\big(v_1(y_1, w_2)\ldots w_k(y_k, w).w(x, s).P \mid \ldots$$

$$\ldots \mid \overline{v_i}\langle z_i, v_{i+1}\rangle.\,[z_i := A_i] \mid \ldots \mid \overline{v}\langle z, t\rangle.\,[z := A]\,\big)$$

*où :*

- *$k \geq 0$, $0 \leq i \leq k$.*
- *Si $k = 0$, alors $w = v$.*
- *$t$ est soit $u$, soit un nom privé de $[\![T]\!]$.*

**Preuve**. Par définition, $(\lambda x, A)$ est un redex premier de $T$ ssi il existe un sous-terme spinal de $T$ de la forme $UA$, avec $\lambda_h(U) = \lambda x :: \rho$; $[\![UA]\!]$ sera donc sûrement un sous-processus de $[\![T]\!]$. Le paramètre (nom libre) de ce sous-processus sera soit $u$, soit un nom privé de $[\![T]\!]$; en général, on peut l'appeler $t$ et dire donc que

$$[\![UA]\!]t = \boldsymbol{\nu}(z, v).\,\big([\![U]\!]v \mid \overline{v}\langle z, t\rangle.\,[z := A]\big)$$

est un sous-processus de $[\![T]\!]u$.

Or, puisque $\lambda_h(U)$ commence par $\lambda x$, $U$ ne peut pas être une variable; il ne peut donc qu'être de la forme $(\lambda y_1 \ldots y_k x.V)A_1 \ldots A_k$, avec $k \geq 0$. En substituant la traduction de ce terme dans $[\![UA]\!]v$ on obtient exactement le processus voulu. □

Le lemme 3.7 est fondamental parce qu'il a comme conséquence immédiate le corollaire suivant :

**Corollaire 3.8** *Soit $T$ un $\lambda$-terme dont l'hoc redex est $(\lambda x, A)$ et tel que $\lambda_h(T) = []$. Alors on a*

$$[\![T]\!]u \rightsquigarrow \boldsymbol{\nu}(z, v, \ldots).\,\Big([\![T^{(0)}]\!]\{z/x\}v \mid \ldots \mid [z := A] \mid \ldots\Big)$$

*où $z$ est le résidu de $x$ (selon la définition 3.1).*

**Preuve**. Grâce au lemme 3.6, puisque la liste des lambda de tête de $T$ vide, la réduction de $[\![T]\!]u$ arrive jusqu'au sous-processus $[\![T^{(0)}]\!]$, en effectuant toutes les étapes de $\lambda$-assignation possibles. Or, puisque $(\lambda x, A)$ est un redex premier, par le lemme 3.7, on sait que $[\![T]\!]u$ contient le processus

$$\boldsymbol{\nu}(z, \vec{z}, v, \vec{v}).\big(v_1(y_1, w_2)\ldots w_k(y_k, w).w(x, s).P \mid \ldots$$

$$\ldots \mid \overline{v_i}\langle z_i, v_{i+1}\rangle.\,[z_i := A_i] \mid \ldots \mid \overline{v}\langle z, t\rangle.\,[z := A]\,)$$

qui ne peut pas être sous-processus de $[\![T^{(0)}]\!]$, car il contient la traduction d'au moins une abstraction et $T^{(0)}$ n'en contient pas par définition. Mais alors la réduction de $[\![T]\!]u$ doit être passée par ce processus, en réduisant lequel on voit bien qu'on arrive à faire interagir $v(x,s).P$ avec $\overline{v}\langle z, t\rangle.\,[z := A]$, interaction qui « libère » l'assignation $[z := A]$ et avec laquelle $z$ devient le résidu de $x$. □

A l'aide de ces lemmes et de leur corollaire, on va montrer le résultat final de cette section :

**Théorème 3.9** *Soit $T$ un $\lambda$-terme. Alors, la suite de substitutions par réduction linéaire de tête faible de $T$ et la suite de substitutions de l'exécution de $[\![T]\!]u$ coïncident.*

**Preuve**. Soit $Sub_\lambda(T) = []$. Dans ce cas, $T$ est normal par rapport à la réduction linéaire de tête faible, et ceci nous donne deux possibilités : soit $\lambda_h(T) \neq []$, soit $\lambda_h(T) = []$ mais l'ovette de $T$ n'apparaît dans aucun de redex premier de $T$. Dans le premier cas, par le lemme 3.6 on a bien $Sub_\pi(T) = []$ ; dans le deuxième cas, $T^{(0)} = xA_{0,1}\ldots A_{0,\sharp^0_@(T)}$ et donc par le même lemme on arrive (bien entendu sans faire de $\lambda$-substitutions) à un processus de la forme

$$\boldsymbol{\nu}(\vec{z}, \vec{v}).\,(\overline{x}v \mid \ldots \mid [z_j := A_j] \mid \ldots)$$

avec $v \in \{u, \vec{v}\}$. Or puisque $x$ n'apparaît pas dans un sous-terme spinal de type $\lambda x.U$ de $T$ (sinon, comme $\lambda_h(T)$ est vide, ce sous terme aurait une application correspondante, et l'ovette serait donc la variable d'un redex premier, mais ceci n'est pas le cas par hypothèse), on déduit qu'aucune substitution d'un nom $z_j$ au nom $x$ n'a été effectuée pendant la réduction qui a amené au processus ci-dessus ; en conséquence, $x \notin \vec{z}$, et donc le sous-processus $\overline{x}v$ ne peut pas communiquer avec aucun des sous-processus d'assignation, lesquelles rappelons ont la forme $!z_j(w).[\![A_j]\!]w$. L'exécution donc s'arrête juste avant qu'il aurait été possible d'effectuer une $\lambda$-substitution, en obtenant ainsi $Sub_\pi(T) = []$.

Supposons maintenant que $Sub_\lambda(T) = (x_0, A) :: \rho$ ($x_0$ étant une occurrence de $x$), et que $T'$ soit le réduit de $T$ par réduction linéaire de tête faible. Evidemment $v_h(T) = x_0$, et $(\lambda x, A)$ est l'hoc-redex de $T$. On va montrer maintenant que $Sub_\pi(T) = (x_0, A) :: Sub_\pi(T')$, en concluant donc que les deux suites de substitution coïncident (car évidemment $\rho = Sub_\lambda(T')$).

D'après le lemme 3.6, on sait que l'exécution de $[\![T]\!]u$ arrive jusqu'à « faire sortir » au niveau d'interaction le sous-processus $[\![T^{(0)}]\!]$, qui contient la traduction de l'ovette, c'est à dire $\overline{x}v$, où $v$ est un nom lié par une restriction. Maintenant il y a deux remarques fondamentales à faire : d'abord, l'occurrence du nom $x$ dans $\overline{x}v$ est clairement celle qui correspond précisément à $x_0$ ; deuxièmement, comme $(\lambda x, A)$ est un redex premier de $T$, par le corollaire 3.8, le processus obtenu après la réduction contiendra le sous processus d'assignation $[z := A]$ et le nom $z$ sera le résidu du nom $x$. Evidemment l'occurrence de $x$ dans $\overline{x}v$ aura alors été substituée avec $z$, et le processus obtenu par réduction de $[\![T]\!]u$ prendra la forme

$$\boldsymbol{\nu}(z, \vec{z}, \vec{v}).\,(\overline{z}v \mid [z := A] \mid \ldots) \;\equiv\; \boldsymbol{\nu}(z, \vec{z}, \vec{v}).\,(\overline{z}v \mid z(w).[\![A]\!]w \mid [z := A] \mid \ldots)$$

avec $v \in \{u, \vec{v}\}$. Ce processus se réduit en

$$\boldsymbol{\nu}(z, \vec{z}, \vec{v}).\,([\![A]\!]v \mid [z := A] \mid \ldots)$$

en produisant précisément la $\lambda$-substitution $(x_0, A)$.

Il nous reste donc à vérifier que les $\lambda$-substitutions qui seront effectuées à partir de ce moment sont exactement celles de $[\![T']\!]u$. Pour montrer cela, considérons la réduction de $[\![T']\!]u$. La seule différence entre $T$ et $T'$ est que dans ce dernier terme $x_0$ a été remplacée par une copie du terme $A$. Or, si $n$ est la hauteur de $T$ et $m$ la hauteur de $A$, on a évidemment que $h(T') = h(T) + h(A) = n + m$ ; on a donc, vu que les épines dorsales des deux termes coïncident jusqu'à l'ovette de $T$ (cette dernière exclue), $T'^{(i)} = T^{(i-m)}$, pour tout $i$ tel que $m + 1 \leq i \leq n + m$. Mais alors, en commençant du niveau $n + m$ de $[\![T']\!]$ et en descendant jusqu'au niveau $m + 1$, on trouve que tous ces niveaux de $[\![T']\!]u$ coïncident parfaitement avec ceux de $[\![T]\!]u$, tandis que au niveau $m$, au lieu qu'avoir la traduction de $T^{(0)} = xA_{0,1} \ldots A_{0,\sharp^0_@(T)}$ on aura quelque chose comme

$$[\![AA_{0,1} \ldots A_{0,\sharp^0_@(T)}]\!] = \boldsymbol{\nu}(v, \ldots).\, ([\![A]\!]v \mid \ldots)$$

En conséquence, la réduction de $[\![T']\!]u$ se comporte au début exactement comme celle de $[\![T]\!]u$, jusqu'à arriver (sans avoir fait de $\lambda$-substitutions) au niveau $m$, où on trouvera un processus structurellement équivalent à ce qu'on a obtenue après la première $\lambda$-substitution de $[\![T]\!]u$. Comme le deux processus sont équivalents, ils engendreront en particulier la même suite de substitutions, et donc $Sub_\pi(T) = (x_0, A) :: Sub_\pi(T')$. □

# Annexe

# Le $\pi$-calcul et la machine de Krivine

Dans la suite on exposera une preuve alternative de la correspondance entre la dynamique de la traduction de Milner et la réduction linéaire de tête faible, en utilisant un résultat montré dans [DR03], qui relie la rlt faible à la machine de Krivine. Pour comprendre ce résultat, il faut d'abord introduire brièvement cette machine, appelée dans la suite KAM (*Krivine's Abstract Machine*).

La KAM est probablement le modèle opérationnel le plus simple du $\lambda$-calcul. Une configuration de la KAM est complètement définie par son état, qui consiste[3] en un triplet $\langle T, E, \sigma\rangle$, où $T$ est un $\lambda$-terme, $E$ est dit l'*environnement* et $\sigma$ est une pile de *clôtures*. Une clôture est un couple $(T, E)$, où $T$ est un $\lambda$-terme et $E$ un environnement ; un environnement est un ensemble de triplets $(x, T, E)$, où $x$ est une variable du $\lambda$-calcul, $T$ un terme et $E$ un environnement lui-même.

Le fonctionnement de la KAM est défini par les *transitions* suivantes :

$$\begin{array}{rccc} \textbf{push}: & \langle TU, E, \sigma\rangle & \to & \langle T, E, (U, E) :: \sigma\rangle \\ \textbf{pop}: & \langle \lambda x.T, E, (U, E') :: \sigma\rangle & \to & \langle T, \{(x, U, E')\} \cup E, \sigma\rangle \\ \textbf{jump}: & \langle x, \{(x, T, E')\} \cup E, \sigma\rangle & \to & \langle T, E', \sigma\rangle \end{array}$$

Pour ce qui concerne la transition *jump*, il faut préciser qu'elle est bien définie grâce à un résultat du à Sylvain Lippi, démontré dans [Lip99], qui garantit pour n'importe quelle variable $x$ la présence dans l'environnement d'au plus un triplet dont la première composante est $x$ elle-même. Ce qui est très remarquable, voire « miraculeux » est que l'$\alpha$-conversion n'est pas nécessaire au résultat ; la seule chose demandée est que le terme exécuté respecte la « convention de Barendregt », c'est à dire que toute variable liée ait un nom différent.

De toute façon, l'exécution d'un $\lambda$-terme $T$ sur la KAM peut être représentée par la suite de transitions engendrée à partir de l'état initial $\langle T, \emptyset, []\rangle$. On peut alors définir la suite de substitutions de l'exécution d'un terme sur la KAM de même que pour la rlt et la traduction de Milner :

[3] En vérité il y a plusieurs manières de présenter la machine de Krivine ; on a choisi celle donnée dans [Lip99], qui à notre avis est la plus simple à décrire. Voir par exemple [DR03] pour une version de la KAM utilisant les indices de de Bruijn.

**Définition A.1 (Suite de substitutions de la KAM)** *La* suite de substitutions *de l'exécution du terme $T$ sur la KAM est la liste (éventuellement infinie) $[(x_0, A_1), \ldots]$ où $x_i$ (resp. $A_{i+1}$) est l'occurrence de variable (resp. le terme) apparaissant dans l'état source (resp. l'état destination) de la $i$-ème transition* jump. *On dénotera cette suite par $Sub_{\mathrm{KAM}}(T)$.*

On peut maintenant énoncer la correspondance entre la rlt faible et la machine de Krivine de la façon suivante :

**Théorème A.1 (Danos-Regnier)** *Si $T$ est un $\lambda$-terme, alors on a $Sub_{\mathrm{KAM}}(T) = Sub_{\lambda}(T)$. Autrement dit, il existe une correspondance exacte entre les substitutions de la réduction linéaire de tête faible de $T$ et les transitions* jump *de l'exécution de $T$ avec la machine de Krivine.*

**Preuve**. Voir le corollaire 5 de [DR03]. □

Ce résultat nous fournit une autre voie pour montrer le lien entre la simulation du $\lambda$-calcul dans le $\pi$-calcul et la rlt faible ; en fait, si on arrive à trouver une correspondance entre la machine de Krivine (ou une machine équivalente) et la traduction de Milner, on trouve évidemment en même temps une preuve de la correspondance entre cette dernière et la rlt.

Pour commencer, introduisons quelques notations utiles :

**Définition A.2** *Etant donnés une liste des $\lambda$-termes et des noms $u, v, \vec{z}$ du $\pi$-calcul, on définit le processus*

$$\begin{array}{rcl} \mathsf{Stk}([], u, v, \vec{z}) & \stackrel{def}{=} & \mathbf{0} \\ \mathsf{Stk}([U_1, \ldots, U_n], u, v, \vec{z}) & \stackrel{def}{=} & \boldsymbol{\nu}\vec{v}.\big(\overline{v}\langle z_1, v_1\rangle.\,[z_1 := U_1] \;\big|\; \ldots \\ & & \ldots \;\big|\; \overline{v_{n-1}}\langle z_n, u\rangle.\,[z_n := U_n]\big) \end{array}$$

*On appelle* assignation *un couple $(z, A)$ où $z$ est un nom et $A$ un $\lambda$-terme. Alors, étant donné un ensemble d'assignations, on définit le processus*

$$\begin{array}{rcl} \mathsf{Env}(\emptyset) & \stackrel{def}{=} & \mathbf{0} \\ \mathsf{Env}(\{(z_1, A_1), \ldots, (z_n, A_n)\}) & \stackrel{def}{=} & [z_1 := A_1] \;\big|\; \ldots \;\big|\; [z_n := A_n] \end{array}$$

*Pour mettre en évidence les noms contenus dans l'ensemble d'assignations $\mathbf{E}$, on écrira $\mathbf{E}\vec{z}$.*

On peut maintenant démontrer le lemme qui suit :

**Lemme A.2** *Soit $P$ un processus qui est la traduction de Milner d'un $\lambda$-terme ou bien le réduit en un nombre quelconque d'étapes de la traduction de Milner d'un $\lambda$-terme. Alors, il existe un terme $T$ qui ne commence pas par une application, une liste de termes $\rho$ et un ensemble d'assignations $\mathbf{E}$ tels que*

$$P \equiv \boldsymbol{\nu}(\vec{z}, v).\,\big([\![T]\!]v \;\big|\; \mathsf{Stk}(\rho, u, v, \vec{z_1}) \;\big|\; \mathsf{Env}(\mathbf{E}\vec{z_2})\big)$$

*où $\vec{z} = \vec{z_1} \cup \vec{z_2}$ et, si $\rho = []$, $v = u$ et $u$ n'apparaît pas dans la restriction.*

**Preuve**. Par induction sur la longueur de la réduction qui a conduit à $P$. Si cette longueur est zéro, $P$ est la traduction du $\lambda$-terme $T$, dont on peut expliciter les applications en écrivant $T = F\vec{A}$, avec $F$ ne commençant pas une application, et donc la thèse est satisfaite trivialement avec $\rho = [\vec{A}]$ et $\mathbf{E} = \emptyset$.

Supposons maintenant la validité du lemme pour les réduits en $m$ étapes et analysons la $m+1$-ème étape de réduction, qui peut être soit une $\lambda$-assignation, soit une $\lambda$-substitution. Dans le premier cas, $T = (\lambda x.U)A$, et l'étape de réduction successive prend la forme

$$\begin{aligned} P &\equiv \boldsymbol{\nu}(\vec{z}, z, v, v').\,\big(v'(x,w).[\![U]\!]w \mid \overline{v'}\langle z, v\rangle.\,[z := A] \mid \mathsf{Stk}(\rho, u, v, \vec{z_1}) \mid \mathsf{Env}(\mathbf{E}\vec{z_2})\big) \\ &\longrightarrow \boldsymbol{\nu}(\vec{z}, z, v).\,\big([\![U\{z/x\}]\!]v \mid [z := V] \mid \mathsf{Stk}(\rho, u, v, \vec{z_1}) \mid \mathsf{Env}(\mathbf{E}\vec{z_2})\big) \\ &\equiv \boldsymbol{\nu}(\vec{z}, z, v).\,\big([\![U\{z/x\}]\!]v \mid \mathsf{Stk}(\rho, u, v, \vec{z_1}) \mid \mathsf{Env}(\{(z, A)\} \cup \mathbf{E}\vec{z_2})\big) \end{aligned}$$

En revanche, dans le cas d'une $\lambda$-substitution, on a $T = z$, et $z$ apparaît forcement dans une assignation de $\mathbf{E}$, sinon il n'y aurait pas d'interaction possible. On a alors $(z, A) \in \mathbf{E}$ et, toujours grâce à l'hypothèse d'induction,

$$\begin{aligned} P &\equiv \boldsymbol{\nu}(\vec{z}, z, v).\,\big(\overline{z}v \mid \mathsf{Stk}(\rho, u, v, \vec{z_1}) \mid \mathsf{Env}(\mathbf{E}\vec{z_2}z)\big) \\ &\equiv \boldsymbol{\nu}(\vec{z}, z, v).\,\big(\overline{z}v \mid z(w).[\![A]\!]w \mid \mathsf{Stk}(\rho, u, v, \vec{z_1}) \mid \mathsf{Env}(\mathbf{E}\vec{z_2}z)\big) \\ &\longrightarrow \boldsymbol{\nu}(\vec{z}, z, v).\,\big([\![A]\!]v \mid \mathsf{Stk}(\rho, u, v, \vec{z_1}) \mid \mathsf{Env}(\mathbf{E}\vec{z_2}z)\big) \end{aligned}$$

Jusqu'à ici, on a montré que tout réduit d'une traduction de Milner est congru à un processus de la forme donnée dans la thèse, mais où le terme $T$ peut aussi commencer par une application, ce qui est explicitement interdit par la thèse elle-même. Pour fixer ce dernier detail, il suffit de remarquer que, avec la notation introduite, la définition de la traduction de Milner d'une application devient

$$[\![TU]\!]u = \boldsymbol{\nu}(v, z).\,\big([\![T]\!]v \mid \mathsf{Stk}([U], u, v, z)\big)$$

En consequence, si l'on a un processus congru à

$$\boldsymbol{\nu}(\vec{z}, v).\,\big([\![T]\!]v \mid \mathsf{Stk}(\rho, u, v, \vec{z_1}) \mid \mathsf{Env}(\mathbf{E}\vec{z_2})\big)$$

avec $T$ arbitraire, on peut toujours expliciter les applications de $T$ en écrivant $T = FA_1 \ldots A_k$, où $F$ est un terme qui ne commence pas par une application. A ce moment, c'est clair que le processus ci-dessus est congru à

$$\boldsymbol{\nu}(\vec{z}, v').\,\big([\![F]\!]v' \mid \mathsf{Stk}([A_1, \ldots, A_k] :: \rho, u, v', \vec{z_3}) \mid \mathsf{Env}(\mathbf{E}\vec{z_2})\big)$$

où $\vec{z_1} \subset \vec{z_3}$ et $\vec{z} = \vec{z_2} \cup \vec{z_3}$, qui est bien de la forme désirée. □

A partir de la remarque qui conclut la preuve du lemme A.2, on pourrait reformuler ce même lemme en disant que tout réduit d'une traduction de Milner est structurellement congru à

$$\boldsymbol{\nu}\vec{z}.\,\big([\![T]\!]u \mid \mathsf{Env}(\mathbf{E}\vec{z})\big)$$

où $T$ est un $\lambda$-terme quelconque. Cela nous donne l'intuition que l'exécution d'un $\lambda$-terme dans le $\pi$-calcul est simulée en le plongeant dans un espèce d'*environnement*, qui contient les valeurs des variables libres du terme lui-même. Les sous-termes spinaux du terme en exécution sont ainsi successivement examinés, et les liens variable-argument peu à peu enregistrés dans l'environnement, jusqu'au moment où l'exécution tombe sur une variable, et il devient alors nécessaire de récupérer la valeur de cette variable, en la cherchant justement dans l'environnement.

Tout cela nous suggère la possibilité d'implémenter cette exécution au moyen d'une machine abstraite à la Krivine, dont les transitions simulent sur le $\lambda$-calcul la même dynamique qui se déroule dans le $\pi$-calcul. On appellera la machine qu'on va définir avec cette idée, *machine de Milner*, ou MAM (*Milner's Abstract Machine*). Une variante de cette machine a été introduite par Silvano Dal Zilio dans [Zil99] sous le nom de *machine abstraite fonctionnelle*.

Les états de la MAM sont des triplets $\langle T, \mathbf{E}, \rho\rangle$, où $T$ est un $\lambda$-terme, $\mathbf{E}$ l'environnement et $\rho$ une liste (éventuellement vide) de termes. Au contraire de la KAM, l'environnement de la MAM est tout simplement un ensemble (éventuellement vide) d'*assignations*, qui sont des couples $(z, A)$, où $z$ est un nom du $\pi$-calcul et $A$ un $\lambda$-terme. Les transitions sont définies comme suit :

$$\begin{array}{rccc} @: & \langle TU, \mathbf{E}, \rho\rangle & \rhd & \langle T, \mathbf{E}, U :: \rho\rangle \\ \lambda: & \langle \lambda x.T, \mathbf{E}, U :: \rho\rangle & \rhd & \langle T\{z/x\}, \mathbf{E} \cup \{(z, U)\}, \rho\rangle \\ \mathsf{var}: & \langle z, \{(z, T)\} \cup \mathbf{E}, \rho\rangle & \rhd & \langle T, \{(z, T)\} \cup \mathbf{E}, \rho\rangle \end{array}$$

Dans la transition $\lambda$, $z$ est un nom « frais », qui n'apparaît ni dans $T$ ni parmi les assignations de l'environnement. En outre, dans cette transition on dira que toute occurrence de $z$ qui substitue $x$ est un *résidu* de $x$ même, de sorte que, lorsqu'une transition $\mathsf{var}$ a lieu, on peut parler de $\lambda$-substitution pareillement au $\pi$-calcul, et définir en conséquence la suite de substitutions pour la MAM :

**Définition A.3 (Suite de substitutions de la MAM)** *La* suite de substitutions *de l'exécution d'un $\lambda$-terme $T$ sur la MAM, notée $Sub_{\mathrm{MAM}}(T)$, est la liste (éventuellement infinie) $[(x_0, A_1), \ldots]$, où $x_i$ est l'occurrence de variable dont $z_i$ est le résidu, et $z_i$ est le terme apparaissant dans l'état source de l'$i$-ème transition $\mathsf{var}$, $A_{i+1}$ étant le terme apparaissant dans l'état destination de la même transition.*

On peut montrer formellement que la MAM capture en effet la dynamique de la traduction de Milner. Pour faire cela, on introduit d'abord la notion de *dynamique*, qui sera définie par un couple $(\mathcal{A}, \succ)$ ou $\mathcal{A}$ est un ensemble et $\succ$ une relation réflexive et transitive sur $\mathcal{A}$. Les deux dynamiques qui nous intéressent sont les suivantes :

**Définition A.4 (Dynamique de la traduction de Milner)** *Soit $\tilde{\Pi}$ l'ensemble contenant tout processus du $\pi$-calcul (à $\alpha$-équivalence près) qui est la traduction de Milner d'un $\lambda$-terme ou bien le réduit (en un nombre non nul d'étapes) de la traduction de Milner d'un $\lambda$-terme. On pose $\Pi = \tilde{\Pi}/\equiv$, c'est à dire $\Pi$ est l'ensemble des traductions de Milner et de leurs réduits quotienté par la congruence structurelle (définition 1.3). Maintenant, si $\longrightarrow$ est la réduction du $\pi$-calcul (définition 1.4), on note par $\twoheadrightarrow$ sa clôture réflexive et transitive et on appelle $(\Pi, \twoheadrightarrow)$ la* dynamique de la traduction de Milner.

**Définition A.5 (Dynamique de la MAM)** *Soit $\Sigma$ l'ensemble des états de la MAM. Chaque transition de cette machine définit une relation binaire sur $\Sigma$ ; appelons ces trois relations $\rhd_@$, $\rhd_\lambda$ et $\rhd_{\mathsf{var}}$. On pose $\rhd_1 \stackrel{\text{def}}{=} \rhd_@ \cup \rhd_\lambda \cup \rhd_{\mathsf{var}}$, et on note par $\rhd$ la clôture réflexive et transitive de $\rhd_1$. On va alors appeler $(\Sigma, \rhd)$ la* dynamique de la machine de Milner.

Dans la suite on montrera que ces deux dynamiques se correspondent exactement l'une avec l'autre. En particulier, une $\lambda$-assignation dans le $\pi$-calcul va

correspondre à une étape $\rhd_\lambda$ et une $\lambda$-substitution va correspondre à une étape $\rhd_{\mathsf{var}}$ ; les étapes $\rhd_@$ vont par contre être absorbées par la congruence structurelle. Donnons d'abord la définition suivante :

**Définition A.6 (Codage et décodage)** *On définit les fonctions*

$$\begin{aligned} enc &: \Pi \to \Sigma \\ dec &: \Sigma \to \Pi \end{aligned}$$

*comme suit. Par le lemme A.2, on sait que tout processus $P \in \Pi$ s'écrit*

$$P = \boldsymbol{\nu}(\vec{z}, v).\ \big([\![T]\!]v \ \big|\ \mathsf{Stk}([U_1, \ldots, U_n], u, v, \vec{z_1}) \ \big|\ \mathsf{Env}(\mathbf{E}\vec{z_2})\big)$$

*où $T$ est un $\lambda$-terme qui ne commence pas pas une application. On pose alors*

$$enc(P) \stackrel{def}{=} \langle F, \mathbf{E}, [U_1, \ldots, U_n]\rangle$$

*Pour ce qui concerne la fonction dec, on pose*

$$dec\left(\langle T, \mathbf{E}\vec{z_2}, \rho\rangle\right) \stackrel{def}{=} \boldsymbol{\nu}(\vec{z}, v).\ \big([\![T]\!]v \ \big|\ \mathsf{Stk}(\rho, u, v, \vec{z_1}) \ \big|\ \mathsf{Env}(\mathbf{E})\big)$$

*où $\vec{z} = \vec{z_1} \cup \vec{z_2}$ et $v = u$ si $\rho = []$ (et dans ce cas $u$ n'apparaît pas dans la restriction). Evidemment, le décodage d'un état ainsi defini n'est pas en général de la forme donnée par le lemme A.2, car $T$ peut commencer par une application ; toutefois, vu que dans $\Pi$ on travaille à congruence structurelle près, le décodage d'un état tombe toujours dans une class d'équivalence de $\Pi$, et le représentant de cette classe (de la forme du lemme A.2) sera celui que l'on considerera dans la suite.*

A partir de maintenant, on supposera toujours que $\mathsf{fn}\left(\mathsf{Env}(\mathbf{E})\right) = \vec{z_2}$, de façon que, par exemple, $\mathsf{fn}\left(\mathsf{Env}(\mathbf{E}z)\right) = \vec{z_2} \cup z$ ; en outre, on abrégera $\mathsf{Stk}(\rho, u, v, \vec{z_1})$, $\mathsf{Stk}(\rho, u, v, z \cup \vec{z_1})$ respectivement par $\mathsf{Stk}(\rho)$, $\mathsf{Stk}(\rho, z)$, et $\vec{z}$ dénotera toujours la réunion des noms $\vec{z_1}$ et $\vec{z_2}$.

Comme on peut le deviner, le décodage est l'inverse du codage :

**Lemme A.3** *Soit $id_\Pi$ l'identité sur $\Pi$. Alors on a $dec \circ enc = id_\Pi$.*

**Preuve**. La preuve est une vérification triviale de l'égalité. Soit

$$P = \boldsymbol{\nu}(\vec{z}, v).\ \Big([\![F]\!]v \ \big|\ \mathsf{Stk}(\vec{U}) \ \big|\ \mathsf{Env}(\mathbf{E})\Big)$$

Alors on a

$$\begin{aligned} dec\left(enc\left(P\right)\right) &= dec\left(\langle F, \mathbf{E}, [\vec{U}]\rangle\right) \\ &= \boldsymbol{\nu}(\vec{z}, v).\ \Big([\![F]\!]v \ \big|\ \mathsf{Stk}([\vec{U}]) \ \big|\ \mathsf{Env}(\mathbf{E})\Big) \end{aligned}$$

□

Toutefois, la fonction de codage n'est pas surjective, donc on n'a pas $enc \circ dec = id_\Sigma$. A savoir, deux états de la MAM reliés par une étape $\rhd_@$ correspondent au même processus dans $\Pi$, car si $s \rhd_@ t$, on n'a pas $dec(s) \longrightarrow dec(t)$, mais on a plutôt $dec(s) \equiv dec(t)$. Ceci est la raison de l'inclusion de la congruence structurelle dans la définition de la dynamique de la traduction de Milner. En faisant attention à ce petit détail, on peut démontrer le résultat qui suit :

**Théorème A.4 (Correspondence entre $(\Pi, \twoheadrightarrow)$ et $(\Sigma, \rhd)$)** *Soient $P, Q \in \Pi$ et $s, t \in \Sigma$ tels que $s = enc(P)$ et $t = enc(Q)$. Alors, $P \twoheadrightarrow Q$ ssi $s \rhd t$.*

**Preuve**. Vérifions d'abord que $P \twoheadrightarrow Q \Longrightarrow enc(P) \rhd enc(Q)$. La réduction dans $\Pi$ comporte deux seules étapes possibles, la $\lambda$-assignation et la $\lambda$-substitution :

- Une $\lambda$-assignation a lieu lorsque la traduction d'une abstraction se synchronise avec la traduction de l'argument d'une application ; $P$ est donc égal à
$$\boldsymbol{\nu}(\vec{z}, v).\ \Big( [\![(\lambda x.T)U]\!]v \ \Big|\ \mathsf{Stk}([\vec{U}]) \ \Big|\ \mathsf{Env}(\mathbf{E}) \Big)$$
qui est congru à
$$\boldsymbol{\nu}(z, \vec{z}, v).\ \Big( [\![\lambda x.T]\!]v \ \Big|\ \mathsf{Stk}([U, \vec{U}], z) \ \Big|\ \mathsf{Env}(\mathbf{E}) \Big)$$
dont le codage est $s = enc(P) = \langle \lambda x.T, \mathbf{E}, [U, \vec{U}] \rangle$. En conséquence, on a
$$\begin{aligned} Q &= \boldsymbol{\nu}(z, \vec{z}, v).\ \Big( [\![T\{z/x\}]\!]v \ \Big|\ \mathsf{Stk}([\vec{U}]) \ \Big|\ \mathsf{Env}(\mathbf{E} \cup \{(z, U)\}) \Big) \\ &\equiv \boldsymbol{\nu}(z, \vec{z}', v).\ \Big( [\![F\{z/x\}]\!]v \ \Big|\ \mathsf{Stk}([\vec{A}\{z/x\}, \vec{U}]) \ \Big|\ \mathsf{Env}(\mathbf{E} \cup \{(z, U)\}) \Big) \end{aligned}$$
(où on a posé $T = F\vec{A}$, avec $F$ ne commençant pas par une application) et donc $t = \langle T\{z/x\}, \mathbf{E} \cup \{(z, U)\}, [\vec{U}] \rangle$. Or, comme $s = \langle \lambda x.F\vec{A}, \mathbf{E}, [U, \vec{U}] \rangle$, on a
$$s \rhd_\lambda \langle (F\vec{A})\{z/x\}, \mathbf{E} \cup \{(z, U)\}, [\vec{U}] \rangle \rhd_@ \ldots \rhd_@ t$$
et donc, à l'aide de quelques étapes $\rhd_@$ implicites dans la dynamique de la traduction de Milner mais que la MAM doit forcement expliciter, on obtient $s \rhd t$.
- Dans le cas de la $\lambda$-substitution, on a
$$P \equiv \boldsymbol{\nu}(z, \vec{z}, v).\ \big( \overline{z}v \ \big|\ \mathsf{Stk}(\rho) \ \big|\ \mathsf{Env}(\{(z, T)\} \cup \mathbf{E}) \big)$$
et donc
$$\begin{aligned} Q &= \boldsymbol{\nu}(z, \vec{z}, v).\ \big( [\![T]\!]v \ \big|\ \mathsf{Stk}(\rho) \ \big|\ \mathsf{Env}(\{(z, T)\} \cup \mathbf{E}) \big) \\ &\equiv \boldsymbol{\nu}(z, \vec{z}', v).\ \Big( [\![F]\!]v \ \Big|\ \mathsf{Stk}([\vec{A}] :: \rho) \ \Big|\ \mathsf{Env}(\{(z, T)\} \cup \mathbf{E}) \Big) \end{aligned}$$
où, comme d'habitude, on a posé $T = F\vec{A}$, avec $F$ ne commençant pas par une application. On obtient ainsi
$$\begin{aligned} s = enc(P) &= \langle z, \{(z, F\vec{A})\} \cup \mathbf{E}, \rho \rangle \\ &\rhd_{\mathsf{var}} \langle F\vec{A}, \{(z, T)\} \cup \mathbf{E}, \rho \rangle \\ &\rhd_@ \ \ldots \ \rhd_@ \langle F, \{(z, T)\} \cup \mathbf{E}, [\vec{A}] :: \rho \rangle = enc(Q) \end{aligned}$$
de sorte que l'on a justement $s \rhd t$.

Montrons maintenant l'implication $s \rhd t \Longrightarrow dec(s) \twoheadrightarrow dec(t)$ (rappelons que, grâce au lemme A.3, $dec(s) = P$ et $dec(t) = Q$). A cet effet, il suffit de vérifier les trois cas suivants :

@. On a $s \rhd_@ t$ ssi $s = \langle TU, \mathbf{E}, \rho\rangle$ et $t = \langle T, \mathbf{E}, U :: \rho\rangle$. Mais dans ce cas, comme $\Pi$ est défini à congruence structurelle près, $dec(s) = dec(t)$, et donc $P \twoheadrightarrow Q$.

$\lambda$. On a $s \rhd_\lambda t$ ssi $s = \langle \lambda x.T, \mathbf{E}, U :: \rho\rangle$ et $t = \langle T\{z/x\}, \mathbf{E} \cup \{(z, U)\}, \rho\rangle$, et alors

$$\begin{aligned} P = dec(s) &= \boldsymbol{\nu}(z, \vec{z}, v'). \left([\![\lambda x.T]\!]v' \mid \mathsf{Stk}(U :: \rho, u, v', z \cup \vec{z}_1) \mid \mathsf{Env}(\mathbf{E})\right) \\ &\equiv \boldsymbol{\nu}(\vec{z}, v). \left([\![(\lambda x.T)U]\!]v \mid \mathsf{Stk}(\rho) \mid \mathsf{Env}(\mathbf{E})\right) \\ &\longrightarrow \boldsymbol{\nu}(z, \vec{z}, v). \left([\![T\{z/x\}]\!]v \mid \mathsf{Stk}(\rho) \mid \mathsf{Env}(\{(z, U\} \cup \mathbf{E})\right) \\ &= dec(t) = Q \end{aligned}$$

$\mathsf{var}$. Dans ce cas, $s \rhd_{\mathsf{var}} t$ ssi $s = \langle z, \mathbf{E}, \rho\rangle$ et $s = \langle T, \mathbf{E}, \rho\rangle$, avec $(z, T) \in \mathbf{E}$, et donc

$$\begin{aligned} P = dec(s) &= \boldsymbol{\nu}(z, \vec{z}, v). \left(\overline{z}v \mid \mathsf{Stk}(\rho) \mid \mathsf{Env}(\mathbf{E}z)\right) \\ &\equiv \boldsymbol{\nu}(z, \vec{z}, v). \left(\overline{z}v \mid z(w).[\![T]\!]w \mid \mathsf{Stk}(\rho) \mid \mathsf{Env}(\mathbf{E}z)\right) \\ &\longrightarrow \boldsymbol{\nu}(z, \vec{z}, v). \left([\![T]\!]v \mid \mathsf{Stk}(\rho) \mid \mathsf{Env}(\mathbf{E}z)\right) \\ &= dec(t) = Q \end{aligned}$$

□

L'exécution d'un $\lambda$-terme $T$ sur la MAM est donc « isomorphe » à la réduction de $[\![T]\!]u$ dans le $\pi$-calcul, dans le sens qu'il existe une correspondance exacte entre les deux dynamiques.

Il nous reste maintenant à vérifier que l'exécution de la MAM correspond à la rlt faible. Cette correspondance peut s'établir en comparant les deux suites de substitutions :

**Théorème A.5 (MAM et réduction linéaire de tête faible)** *Si $T$ est un $\lambda$-terme, alors $Sub_{\mathrm{MAM}}(T) = Sub_\lambda(T)$.*

**Preuve**. Si $\lambda_h(T) \neq []$, la machine va s'arrêter, car elle va se trouver dans un état où le terme à exécuter est une abstraction, mais la pile est vide.

De même, si $\lambda_h(T) = []$ mais $T$ est en forme quasiment-normale de tête, la liste de substitutions va être également vide, en fait cette fois-ci l'exécution arrive à l'ovette mais cette dernière n'apparaît pas dans l'environnement, car elle n'a pas été impliquée dans une transition $\lambda$.

Le cas non trivial est donc celui d'un terme $T$ non-quasiment-normal de tête et dont la liste des lambda de tête est vide. Soit dans ce cas $(\lambda x, A)$ l'hoc redex, et $x_0$ (une occurrence de $x$) l'ovette. Le terme $T$ a alors la forme

$\ldots(\ldots\lambda x\ldots x_0\ldots)A\ldots$, et son exécution sur la MAM va être la suivante :

| | Terme | Env. | Pile |
|---|---|---|---|
| 1 | $T$ | $\emptyset$ | $[]$ |
| | $\vdots$ | | |
| 2 | $(\ldots\lambda x\ldots x_0\ldots)A_1$ | $\mathbf{E}_1$ | $\rho_1$ |
| 3 | $\ldots\lambda x\ldots x_0\ldots$ | $\mathbf{E}_1$ | $A_1 :: \rho_1$ |
| | $\vdots$ | | |
| 4 | $\lambda x\ldots x_0\ldots$ | $\mathbf{E}_2$ | $A_1 :: \rho_1$ |
| 5 | $\ldots z_0\ldots$ | $\mathbf{E}_2 \cup \{(z, A_1)\}$ | $\rho_1$ |
| | $\vdots$ | | |
| 6 | $z_0$ | $\mathbf{E}_3 \cup \{(z, A_1)\}$ | $\rho_2$ |
| 7 | $A_1$ | $\mathbf{E}_3 \cup \{(z, A_1)\}$ | $\rho_2$ |

Dans l'étape 2, $A_1$ est $A$ dans lequel il y a eu des substitutions de variables par des noms (précisément les noms qui se trouve dans les assignations de $\mathbf{E}_1$). A l'étape 4, on trouve la même pile qu'à l'étape 3 grâce à la supposition que $(\lambda x, A)$ soit un redex premier (en fait c'est le hoc-redex). On trouve donc que la première substitution dans $Sub_{\mathrm{MAM}}(T)$ est justement $(x_0, A)$ (vu que $A_1$ est le résidu de $A$).

Vérifions maintenant ce qui se passe dans l'exécution du réduit $T'$ de $T$, qui a la forme $\ldots(\ldots\lambda x\ldots A\ldots)A\ldots$ :

| | Terme | Env. | Pile |
|---|---|---|---|
| 1 | $T'$ | $\emptyset$ | $[]$ |
| | $\vdots$ | | |
| 2 | $(\ldots\lambda x\ldots A_1\ldots)A_1$ | $\mathbf{E}_1$ | $\rho_1$ |
| 3 | $\ldots\lambda x\ldots A_1\ldots$ | $\mathbf{E}_1$ | $A_1 :: \rho_1$ |
| | $\vdots$ | | |
| 4 | $\lambda x\ldots A_1\ldots$ | $\mathbf{E}_2$ | $A_1 :: \rho_1$ |
| 5 | $\ldots A_1\ldots$ | $\mathbf{E}_2 \cup \{(z, A_1)\}$ | $\rho_1$ |
| | $\vdots$ | | |
| 6 | $A_1$ | $\mathbf{E}_3 \cup \{(z, A_1)\}$ | $\rho_2$ |

Dans 2, toute les deux copies de $A$ ont été objet des mêmes substitutions que dans l'exécution de $T$, donc on trouve bien $A_1$. A partir de là, toute les autres transitions vont être les mêmes, avec la remarque fondamentale que, comme il est le résidu du sous-terme argument du redex premier $(\lambda x, A)$, $A_1$ ne contient pas de variables liées par les abstractions examinées après l'étape 2. $A_1$ n'est donc pas impliqué dans les substitutions qui sont effectuées entre les étapes 2 et 6, et c'est pour cela que l'on trouve $A_1$ à cette dernière étape.

L'exécution de $T'$ arrive donc au même état que celle de $T$, mais sans avoir fait aucune substitution. Evidemment la suite de substitutions va coïncider à partir de ce point, et alors on a $Sub_{\mathrm{MAM}}(T) = Sub_\lambda(T)$. □

Avec le théorème A.4, le résultat que l'on vient de démontrer complète la preuve que la dynamique de la traduction de Milner simule en vérité la réduction

linéaire de tête faible ; il a aussi comme conséquence immédiate l'équivalence entre MAM et KAM, au sens où les deux ont la même exécution sur les termes du $\lambda$-calcul.

# Bibliographie


[BB92] Gérard Berry and Gérard Boudol. **The Chemical Abstract Machine**. *Theoretical Computer Science*, 96 :217–248, 1992.

[DR96] Vincent Danos and Laurent Regnier. **Reversible, irreversible and optimal lambda-machines**. In *Proceedings of 1996 Workshop on Linear Logic*, volume 3 of *Electronic notes in Theoretical Computer Science*. Elsevier, 1996.

[DR03] Vincent Danos and Laurent Regnier. **How abstract machines implement head linear reduction**. *Preprint*, 2003.

[Gir87] Jean-Yves Girard. **Linear Logic**. *Theoretical Computer Science*, 50(1) :1–102, 1987.

[Lip99] Sylvain Lippi. *Théorie et pratique des réseaux d'interaction (Interaction Nets)*. Thèse de doctorat, Université Aix-Marseille II, 1999.

[Mil92] Robin Milner. **Functions as Processes**. *Journal of Mathematical Structures in Computer Science*, 2(2) :119–141, 1992. Previous version as Rapport de Recherche 1154, INRIA Sophia-Antipolis, 1990, and in *Proceedings of ICALP '91*, LNCS 443.

[SW01] Davide Sangiorgi and David Walker. *The $\pi$-calculus : a Theory of Mobile Processes*. Cambridge University Press, 2001.

[Zil99] Silvano Dal Zilio. *Calcul bleu : types et objets*. Thèse de doctorat, Université de Nice – Sophia-Antipolis, 1999.