Encodage de la réécriture de termes en π-calcul

Rapport de stage de L2

Guillaume Chirache
Encadré par Romain Demangeon
Laboratoire d'informatique de Paris 6
Sorbonne Université

Septembre 2021

Table des matières

1 Introduction

On considère une algèbre, c'est-à-dire une structure algébrique dont l'ensemble T des termes est construit à partir d'une liste finie de constantes et d'opérateurs admettant un certain nombre de termes en arguments. On souhaite étudier la manipulation des éléments de T en programmation concurrente. Pour cela, il est utile de les représenter en π-calcul, qui est un langage formel servant à modéliser des systèmes concurrents. C'est ce qu'a fait Amel Chadda lors de son stage de L3 au LIP6 en juin-juillet 2019. Son rapport [Cha19] propose une manière d'encoder les termes de telles algèbres en π-calcul, dont on donne une version légèrement modifiée dans la définition 31. Pour poursuivre ce travail, on souhaite effectuer des calculs, dans un cadre le plus général possible, sur ces termes. Le problème qui se pose est représenté dans la figure 1 : comment représenter une application $f : T^n \to T$ en π-calcul pour pouvoir l'appliquer aux termes encodés ?

En mathématiques, une telle application est définie de façon ensembliste par une relation qui à tout élément de T^n associe un unique terme de T. Cette définition n'est pas utilisable directement en informatique,

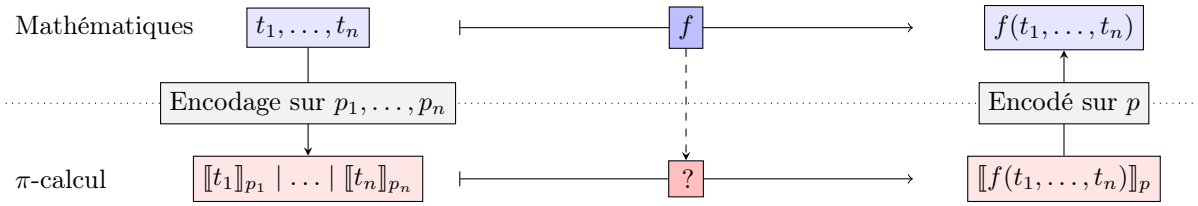

FIGURE 1 – Comment représenter $f : T^n \to T$ en π-calcul ?

car elle ne dit comment obtenir $f(t_1, \ldots, t_n)$ à partir de $t_1, \ldots, t_n$, ni même si ce calcul est réalisable par un ordinateur. Il est nécessaire de disposer d'un modèle de calcul indiquant comment passer d'un élément à son image par des opérations élémentaires. Pour cela, on fait le choix d'utiliser la **réécriture de termes**, sur laquelle l'ouvrage [BN98] constitue une référence. Plutôt que de définir une application $T^n \to T$, on ajoute un opérateur d'arité n, et on explicite sa manipulation à travers des règles de réécriture. On peut ainsi évaluer les termes en leur appliquant des règles de réécriture jusqu'à obtenir un terme irréductible appelé « forme normale ». L'intérêt de cette approche est double : d'une part, les règles de réécriture indiquent explicitement comment passer d'un terme à un autre, ce qui permet de les transcrire en algorithme. D'autre part, on ne perd pas en expressivité par rapport à ce qui est possible sur un ordinateur puisque la réécriture constitue un modèle de calcul Turing-complet. Cela signifie qu'elle permet de modéliser toute opération calculable par une machine de Turing, ce qui correspond à la notion « naturelle » de calculabilité dont la thèse de Church postule l'existence.

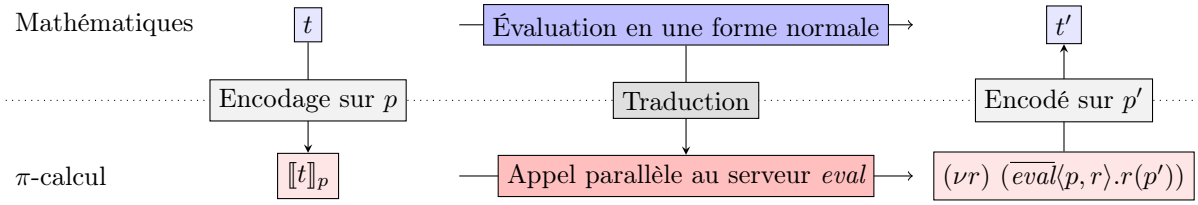

FIGURE 2 – L'implémentation réalisée

Mon travail, réalisé au LIP6 en septembre 2021 et encadré par Romain DEMANGEON, a consisté à m'approprier les notions abordées dans cette introduction (π-calcul, systèmes de réécriture et encodage des algèbres), à les adapter légèrement pour en produire une présentation cohérente, puis à implémenter le serveur *eval* décrit dans la figure 2. Afin d'ouvrir la voie à une future preuve de cette implémentation, j'ai proposé un énoncé de correction (conjectures 37 et 38), ce qui a nécessité la création d'une notion d'encodage inverse (définition 32). Pour illustrer les notions présentées, j'ai pris comme exemple tout au long de l'article la réécriture de l'arithmétique de Peano permettant d'en calculer les additions et multiplications, dont j'ai aussi prouvé la terminaison et la validité.

La section 2 introduit formellement les notions mathématiques d'algèbre et de réécriture sur ses termes, puis la section 3 présente le π-calcul et l'encodage des algèbres dans ce langage. Enfin, la section 4 présente la traduction des règles de réécriture en un service π-calcul.

2 Algèbres et réécriture

Dans cette section, on introduit les notions mathématiques d'algèbre et de réécriture, que nous allons ensuite modéliser en π-calcul. Une algèbre est un objet exclusivement syntaxique, tandis que la réécriture est un modèle de calcul permettant de lui associer une sémantique. Mon travail ici a consisté à m'approprier ces notions et à produire une définition formelle et cohérente des définitions associées, dans la mesure où celles-ci sont introduites différemment selon les références.

2.1 Algèbres de termes

Une algèbre de termes (ou, en informatique, simplement « algèbre ») est une structure algébrique dont les éléments (termes) sont construits à partir d'une liste bien définie d'opérateurs munis d'une arité (un nombre d'arguments attendus). Pour définir formellement les constructions autorisées, on introduit la notion de signature.

Définition 1 (Signature)**.**

(i) On dispose d'un ensemble dénombrable $\mathcal{S}$ de symboles.

(ii) Une signature Σ est un ensemble **fini** de couples $(f, a) \in \mathcal{S} \times \mathbb{N}$, constitués d'un symbole f appelé **opérateur** et d'un entier naturel a appelé **arité** de l'opérateur, et tel que pour tout $((f, a), (g, b)) \in \Sigma^2$, $a \neq b \implies f \neq g$.

(iii) Pour $a \in \mathbb{N}$, on note $\Sigma_a = \{f \in \mathcal{S} \mid (f, a) \in \Sigma\}$ l'ensemble des opérateurs de Σ d'arité a.

(iv) Les opérateurs de Σ_0, qui n'admettent aucun argument, sont appelés constantes.

Définition 2 (Algèbre)**.** On définit inductivement l'algèbre sur la signature Σ, dont l'ensemble des termes est noté $\mathrm{T}(\Sigma)$, par :

$$\forall (f, a) \in \Sigma,\ \forall (t_1, \ldots, t_a) \in \mathrm{T}(\Sigma)^a,\ f(t_1, \ldots, t_a) \in \mathrm{T}(\Sigma)\,.$$

Remarque 3. Informellement, cela signifie qu'un terme $t \in \mathrm{T}(\Sigma)$ est un opérateur appliqué à autant de termes que son arité le demande. Cette définition inclut le cas de base $\Sigma_0 \subseteq \mathrm{T}(\Sigma)$, qui correspond à $a = 0$. Le cas dégénéré où la signature ne comporte aucune constante n'est pas interdit, mais présente peu d'intérêt car $\Sigma_0 = \varnothing \implies \mathrm{T}(\Sigma) = \varnothing$.

Exemple 4 (Entiers naturels et arithmétique de Peano)**.** On peut construire l'ensemble des entiers naturels comme l'algèbre de signature $\{(0, 0), (s, 1)\}$, en considérant que la constante 0 désigne le zéro et que s est l'opérateur « suivant », de sorte que pour tout entier $n \geqslant 0$, n est désigné par le terme $s^n(0)$.

La signature $\mathcal{P} = \{(0, 0), (s, 1), (+, 2), (\cdot, 2)\}$, obtenue en ajoutant deux opérateurs $+$ et $\cdot$ d'arité 2, permet de construire l'ensemble des expressions de l'arithmétique de Peano, c'est-à-dire celles constituées d'additions et de multiplications sur des entiers naturels. Ainsi, le terme $s^2(0) \cdot (s(0) + s^3(0))$ représente l'expression $2 \cdot (1 + 3)$. La structure d'algèbre ne permet cependant pas de traduire le fait que $2 \cdot (1 + 3) = 8$, ce qui nous amènera à introduire la réécriture.

On peut concevoir un terme d'une algèbre comme une arborescence dont les feuilles sont des constantes, les nœuds internes des opérateurs d'arité non nulle, et la racine le dernier opérateur utilisé. Cela justifie l'introduction des définitions suivantes.

Définition 5. Soit Σ une signature. Pour tout $(f, a) \in \Sigma$ et tout terme $t \in \mathrm{T}(\Sigma)$ tel que $t = f(t_1, \ldots, t_a)$ avec $(t_1, \ldots, t_a) \in \mathrm{T}(\Sigma)^a$, on définit récursivement l'ensemble des symboles $\mathrm{S}(t)$, celui des sous-termes $\mathrm{ST}(t)$, la taille $\mathrm{T}(t)$ et la hauteur $\mathrm{H}(t)$ de t, ainsi que le nombre $\mathrm{O}(t, g)$ d'occurences d'un opérateur g de Σ dans t par les formules ci-dessous.

$$\mathrm{S}(t) = \{f\} \cup \left(\bigcup_{i=1}^{a} \mathrm{S}(t_i) \right) \qquad \mathrm{ST}(t) = \{t\} \cup \left(\bigcup_{i=1}^{a} \mathrm{ST}(t_i) \right) \qquad \mathrm{T}(t) = 1 + \sum_{i=1}^{a} \mathrm{T}(t_i)$$

$$\mathrm{H}(t) = 1 + \max_{i \in [\![1, a]\!]} \mathrm{H}(t_i) \qquad \mathrm{O}(t, g) = \begin{cases} 1 + \sum_{i=1}^{a} \mathrm{O}(t_i, g) & \text{si } f = g \\ \sum_{i=1}^{a} \mathrm{O}(t_i, g) & \text{sinon} \end{cases}$$

On appelle de plus racine de t et on note $\mathrm{R}(t)$ l'opérateur f. Enfin, on définit l'ensemble des constantes de t comme $\mathrm{C}(t) = \mathrm{S}(t) \cap \Sigma_0$.

2.2 Systèmes de réécriture

La réécriture (ou récriture) dans une algèbre est un cadre formel permettant d'en transformer les termes par l'application de règles précises et calculables. Elle permet de donner une sémantique à une structure

exclusivement syntaxique. Ainsi, en reprenant l'exemple 4, on souhaite par exemple évaluer $s^2(0)\cdot(s(0)+s^3(0))$ en $s^8(0)$.

Pour définir formellement ces règles et leurs applications, on doit introduire plusieurs objets. L'idée globale est la suivante : une variable permet de désigner un terme quelconque, une substitution permet de remplacer les variables par des termes, et un contexte permet de situer, parmi les sous-termes d'un élément, celui où l'on applique une règle de réécriture.

Définition 6 (Algèbre munie de variables, substitution)**.**

(i) On dispose d'un ensemble dénombrable $\mathcal{V}$ de « variables », tel que $\mathcal{S} \cap \mathcal{V} = \varnothing$.

(ii) L'algèbre munie de variables sur la signature Σ, dont l'ensemble des termes est noté $\mathrm{T}(\Sigma, \mathcal{V})$, est informellement l'algèbre obtenue en ajoutant les variables aux constantes de Σ. Formellement, on la définit inductivement par les formules ci-dessous.

$$\mathcal{V} \subseteq \mathrm{T}(\Sigma, \mathcal{V}) \qquad \forall (f, a) \in \Sigma,\ \forall (r_1, \dots, r_a) \in \mathrm{T}(\Sigma, \mathcal{V})^a,\ f(r_1, \dots, r_a) \in \mathrm{T}(\Sigma, \mathcal{V})$$

(iii) Pour $r \in \mathrm{T}(\Sigma, \mathcal{V})$, on définit $\mathrm{S}(r)$, $\mathrm{ST}(r)$, $\mathrm{T}(r)$, $\mathrm{H}(r)$, $\mathrm{O}(r, \cdot)$, $\mathrm{R}(r)$ et $\mathrm{C}(r)$ de la même manière que dans la définition 5, en considérant les variables comme des opérateurs d'arité nulle. On définit l'ensemble $\mathrm{V}(r)$ des variables de r par $\mathrm{V}(r) = \mathrm{C}(r) \cap \mathcal{V}$.

(iv) Une substitution est une application $\sigma : \mathcal{V} \to \mathrm{T}(\Sigma, \mathcal{V})$ dont le support $\mathrm{supp}(\sigma) = \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$ est fini et telle que $\sigma(\mathrm{supp}(\sigma)) \subseteq \mathrm{T}(\Sigma)$.

(v) Si $r \in \mathrm{T}(\Sigma, \mathcal{V})$ et que $\mathrm{V}(r) \subseteq \mathrm{supp}(\sigma)$, on note r^σ le terme de $\mathrm{T}(\Sigma)$ obtenu en remplaçant les variables de r selon σ. Formellement, si $r \in \mathrm{supp}(\sigma)$, $r^\sigma = \sigma(r)$, et si $r = f(r_1, \dots, r_a)$ avec $(f, a) \in \Sigma$ et $(r_1, \dots, r_a) \in \mathrm{T}(\Sigma, \mathcal{V})^a$, $r^\sigma = f({r_1}^\sigma, \dots, {r_a}^\sigma)$.

Définition 7 (Contexte, instance)**.**

(i) Un contexte $\mathscr{C}$ sur Σ est informellement un terme de $\mathrm{T}(\Sigma)$ auquel un unique sous-terme, noté $\blacktriangle$, est manquant. Formellement, un contexte est soit le symbole $\blacktriangle$, soit un objet de la forme $f(t_1, \dots, t_a)$, où $(f, a) \in \Sigma$ et il existe $i \in [\![1, a]\!]$ tel que t_i est un contexte et que $\forall j \neq i$, $t_j \in \mathrm{T}(\Sigma)$.

(ii) Si $\mathscr{C}$ est un contexte sur Σ et $u \in \mathrm{T}(\Sigma)$, on appelle instance de $\mathscr{C}$ de paramètre u et on note $\mathscr{C}[u]$ le terme obtenu en remplaçant $\blacktriangle$ dans $\mathscr{C}$ par u. Formellement, pour tout $u \in \mathrm{T}(\Sigma)$, on pose $\blacktriangle[u] = u$ et pour tout $(f, a) \in \Sigma$ et tout contexte $\mathscr{C}$ tel que $\mathscr{C} = f(t_1, \dots, t_a)$, où $i \in [\![1, a]\!]$ est tel que t_i est un contexte, $\mathscr{C}[u] = f(\dots, t_{i-1}, t_i[u], t_{i+1}, \dots)$.

Définition 8 (Système de réécriture)**.**

(i) Une règle de réécriture $r \longrightarrow r'$ sur la signature Σ est la donnée d'un couple de termes $(r, r') \in \mathrm{T}(\Sigma, \mathcal{V})^2$ tel que r n'est pas une variable, que toutes les variables de r y apparaissent une unique fois et que toutes les variables de r' apparaissent dans r, i.e. $r \notin \mathcal{V}$, $\forall x \in \mathrm{V}(r)$, $\mathrm{O}(r, x) = 1$ et $\mathrm{V}(r') \subseteq \mathrm{V}(r)$.

(ii) Un système de réécriture sur Σ est un ensemble fini de règles de réécriture sur Σ.

Définition 9 (Règle applicable, terme réductible, forme normale)**.**

(i) On dit que $t \in \mathrm{T}(\Sigma)$ se réécrit en $t' \in \mathrm{T}(\Sigma)$, ce que l'on note $t \longrightarrow t'$, s'il existe une règle $r \longrightarrow r'$, une substitution σ et un contexte $\mathscr{C}$ sur Σ tels que $t = \mathscr{C}[r^\sigma]$ et $t' = \mathscr{C}[{r'}^\sigma]$.

(ii) Un terme $t \in \mathrm{T}(\Sigma)$ est dit réductible s'il existe $t' \in \mathrm{T}(\Sigma)$ tel que $t \longrightarrow t'$, et irréductible dans le cas contraire.

(iii) On note $\xrightarrow{*}$ la clotûre symétrique transitive de $\longrightarrow$, c'est-à-dire que $t \xrightarrow{*} t'$ si et seulement s'il existe $n \in \mathbb{N}$ et $(t_k)_{k \in [\![0,n]\!]} \in \mathrm{T}(\Sigma)^{n+1}$ tels que $t_0 = t$, $t_n = t'$ et $\forall k \in [\![0, n-1]\!]$, $t_k \longrightarrow t_{k+1}$.

(iv) On dit que $t' \in \mathrm{T}(\Sigma)$ est une forme normale de $t \in \mathrm{T}(\Sigma)$ si $t \xrightarrow{*} t'$ et que t' est irréductible.

Évaluer un terme consiste donc à lui appliquer des règles de réécriture, de façon potentiellement non déterministe s'il y a plusieurs possibilités, jusqu'à obtenir une forme normale. Si le processus de réécriture aboutit toujours en un nombre fini d'étapes, le système de réécriture est dit terminant.

Définition 10 (Système de réécriture terminant)**.** Un système de réécriture sur Σ est terminant s'il n'existe aucune suite $(t_n)_{n \in \mathbb{N}} \in \mathrm{T}(\Sigma)^{\mathbb{N}}$ telle que pour tout $n \in \mathbb{N}$, $t_n \longrightarrow t_{n+1}$.

2.3 Cas de l'arithmétique de Peano

On reprend la signature $\mathcal{P} = \{(0,0),(s,1),(+,2),(\cdot,2)\}$ définie à l'exemple 4, et on la munit d'un système de réécriture pour pouvoir calculer la valeur des expressions de $\mathrm{T}(\mathcal{P})$.

Définition 11 (Réécriture sur l'arithmétique de Peano)**.** On munit la signature $\mathcal{P}$ des règles de réécriture ci-dessous, où $(x,y) \in \mathcal{V}^2$.

$$0+y \longrightarrow y \qquad s(x)+y \longrightarrow s(x+y) \qquad 0\cdot y \longrightarrow 0 \qquad s(x)\cdot y \longrightarrow y+(x\cdot y)$$

Ces quatre règles de réécriture sont compatibles avec les lois $+$ et $\cdot$ sur $\mathbb{N}$ puisqu'étant donné la description des opérateurs 0 et s dans l'exemple 4, elles s'interprètent respectivement comme les égalités $0+p=p$, $(n+1)+p=(n+p)+1$, $0\cdot p=0$ et $(n+1)\cdot p = p+(n\cdot p)$, qui sont bien vérifiées pour tout $(n,p)\in\mathbb{N}^2$. Ainsi, si $t \longrightarrow t'$ et, par récurrence immédiate, si $t \xrightarrow{*} t'$, les expressions modélisées par t et t' ont la même valeur.

Montrons maintenant que ce système de réécriture a de bonnes propriétés. En particulier, il est terminant, et calcule effectivement les opérations contenues dans les termes, de sorte que les termes irréductibles ne contiennent pas d'additions et de multiplications.

Proposition 12. *Si $t \in \mathrm{T}(\mathcal{P})$ est irréductible, alors $\mathrm{O}(t,+) = \mathrm{O}(t,\cdot) = 0$.*

Preuve. Montrons la contraposée : soit $t \in \mathrm{T}(\mathcal{P})$ tel que $\mathrm{O}(t,+) + \mathrm{O}(t,\cdot) \geqslant 1$. On pose :

$$h_{\min} = \min\{\mathrm{H}(t') \mid t' \in \mathrm{ST}(t),\, \mathrm{R}(t') = + \text{ ou } \mathrm{R}(t') = \cdot\}\,,$$

qui est bien défini comme plus petit élément d'une partie non vide de $\mathbb{N}$, et on se donne un $t' \in \mathrm{ST}(t)$ tel que $\mathrm{R}(t') \in \{+,\cdot\}$ et $\mathrm{H}(t') = h_{\min}$. Alors le premier argument de la racine de t' est de hauteur $h \leqslant h_{\min} - 1$, donc si sa propre racine était $+$ ou $\cdot$, on aurait $h_{\min} \leqslant h \leqslant h_{\min} - 1$... exclu ! Cette racine est donc soit 0, soit s, ce qui permet d'appliquer l'une des quatre règles de réécriture. □

La terminaison est plus délicate. Pour identifier un variant, on donne un poids à chaque terme de $\mathrm{T}(\mathcal{P})$, en s'inspirant de la notion d'interprétation polynomiale définie dans [BN98].

Définition 13. On définit le poids des termes de $\mathrm{T}(\mathcal{P})$ par les formules suivantes, pour tout $(x,y) \in \mathrm{T}(\mathcal{P})^2$.

$$\langle 0\rangle = 1 \qquad \langle s(x)\rangle = \langle x\rangle + 1 \qquad \langle x+y\rangle = 2\langle x\rangle + \langle y\rangle \qquad \langle x\cdot y\rangle = \langle x\rangle^2\langle y\rangle + 1$$

En particulier, $\langle t\rangle$ est un entier pour tout $t \in \mathrm{T}(\mathcal{P})$.

Lemme 14. *Pour tout $t \in \mathrm{T}(\mathcal{P})$, $\langle t\rangle \geqslant 1$.*

Preuve. Montrons par induction structurelle sur $t \in \mathrm{T}(\mathcal{P})$ la proposition $\mathcal{H}(t)$: « $\langle t\rangle \geqslant 1$ ». $\mathcal{H}(0)$ est claire. Soit $(x,y) \in \mathrm{T}(\mathcal{P})^2$ tel que $\mathcal{H}(x)$ et $\mathcal{H}(y)$. Alors $\langle s(x)\rangle = \langle x\rangle + 1 \geqslant 2 \geqslant 1$, $\langle x+y\rangle \geqslant 2\times 1 + 1 = 3 \geqslant 1$ et $\langle x\cdot y\rangle \geqslant 1^2\times 1 + 1 = 2 \geqslant 1$. On a donc $\mathcal{H}(s(x))$, $\mathcal{H}(x+y)$ et $\mathcal{H}(x\cdot y)$, ce qui conclut la récurrence. □

Proposition 15. *Pour chaque règle de réécriture $r \longrightarrow r'$ définie sur $\mathcal{P}$, et pour toute substitution σ telle que $\mathrm{V}(r) \subseteq \mathrm{supp}(\sigma)$, $\langle r'^{\sigma}\rangle < \langle r^{\sigma}\rangle$.*

Preuve. Soit σ une substitution dont le support contient $\{x,y\}$. On pose $t = \sigma(x)$ et $t' = \sigma(y)$. Vérifions les quatre règles.

$$\langle 0+t'\rangle = 2 + \langle t'\rangle > \langle t'\rangle \qquad \langle s(t)+t'\rangle = 2(\langle t\rangle + 1) + \langle t'\rangle = 2 + 2\langle t\rangle + \langle t'\rangle > 1 + 2\langle t\rangle + \langle t'\rangle = \langle s(\langle t\rangle + \langle t'\rangle)\rangle$$

$$\langle 0\cdot t'\rangle = 1^2\times\langle t'\rangle + 1 = \langle t'\rangle + 1 > 1 = \langle 0\rangle \quad \text{par le lemme 14}$$

Pour la règle $s(x)\cdot y \longrightarrow y + (x\cdot y)$, on a $\langle s(t)\cdot t'\rangle = (\langle t\rangle+1)^2\langle t'\rangle + 1 = \langle t'\rangle\langle t\rangle^2 + 2\langle t\rangle\langle t'\rangle + \langle t'\rangle + 1$ et $\langle t' + (t\cdot t')\rangle = 2\langle t'\rangle + \langle t\rangle^2\langle t'\rangle + 1$. Comme $2\langle t\rangle\langle t'\rangle + \langle t'\rangle > 2\langle t'\rangle$ par le lemme 14, on trouve bien $\langle s(t)\cdot t'\rangle > \langle t' + (t\cdot t')\rangle$. □

Lemme 16. *Soit $\mathscr{C}$ un contexte sur $\mathcal{P}$ et $(t,t') \in \mathrm{T}(\mathcal{P})^2$. Supposons que $\langle t' \rangle < \langle t \rangle$. Alors $\langle \mathscr{C}[t'] \rangle < \langle \mathscr{C}[t] \rangle$.*

Schéma de preuve. On se donne $(t,t') \in \mathrm{T}(\mathcal{P})^2$ tel que $\langle t' \rangle < \langle t \rangle$ et on montre par induction structurelle sur le contexte $\mathscr{C}$ la proposition $\langle \mathscr{C}[t'] \rangle < \langle \mathscr{C}[t] \rangle$. La base est immédiate et l'induction se déduit de la stricte croissance sur $\mathbb{N}^*$ de $x \mapsto x+1$, $x \mapsto 2x+y$ et $x \mapsto x^2y+1$ pour $y \geqslant 1$, ainsi que de $y \mapsto 2x+y$ et $y \mapsto x^2y+1$ pour $x \geqslant 1$. □

Théorème 17 (Terminaison)**.** *Le système de réécriture défini sur $\mathcal{P}$ termine.*

Preuve. Supposons avoir une suite $(t_n)_{n\in\mathbb{N}}$ de termes telle que $\forall n \in \mathbb{N}$, $t_n \longrightarrow t_{n+1}$, et fixons $n \in \mathbb{N}$. On note $r \longrightarrow r'$ et σ respectivement la règle et la substitution associés à la réécriture $t_n \longrightarrow t_{n+1}$. Par la proposition 15, $\langle r'^{\sigma} \rangle < \langle r^{\sigma} \rangle$, puis par le lemme 16, $\langle t_{n+1} \rangle < \langle t_n \rangle$. Ainsi, la suite $(\langle t_n \rangle)_{n\in\mathbb{N}}$ est strictement décroissante et à valeurs dans $\mathbb{N}^*$... absurde ! Donc le système de réécriture termine. □

3 Encodage des algèbres en π-calcul

3.1 Présentation du π-calcul et équivalence comportementale

Introduit en 1992 par l'informaticien britannique Robin Milner, le π-calcul (ou Pi-calcul) est une algèbre de processus, autrement dit un langage formel ne servant pas à programmer en tant que tel, mais à modéliser des processus pour en étudier des propriétés. Le π-calcul permet de représenter des **systèmes concurrents par passage de messages sur des canaux**, c'est-à-dire constitués de plusieurs tâches qui s'exécutent parallélement et peuvent s'échanger des messages sur des « canaux » dédiés. Il n'en modélise que la partie observable, à savoir l'envoi et la réception de messages. Le π-calcul est en outre Turing-complet. Il existe plusieurs variantes du π-calcul : celui que nous utilisons est le π-calcul polyadique avec réception répliquée, tel qu'introduit dans le cours [DA14]. Certaines définitions, notamment celle de l'α-équivalence, proviennent elles du livre [SW01], qui constitue l'ouvrage de référence en π-calcul.

Définition 18 (Noms)**.** On dispose d'un ensemble dénombrable de noms (aussi appelés canaux), noté $\mathcal{N} = \{a, b, c, \dots, x, y, \dots\}$.

Définition 19 (Syntaxe du π-calcul)**.** La syntaxe des processus en π-calcul est la suivante, écrite en forme de Backus-Naur :

$$P, Q ::= 0 \parallel \overline{a}\langle v_1, \dots, v_n \rangle.P \parallel a(x_1 \dots, x_n).P \parallel\ !a(x_1, \dots, x_n).P \parallel P \mid Q \parallel P + Q \parallel (\nu c)\ P\,,$$

où $n \in \mathbb{N}$ et a, c, v_i et x_i sont des canaux pour $i \in [\![1, n]\!]$. L'ensemble des processus est noté Π.

- 0 est le processus nul : il ne fait rien.
- $\overline{a}\langle v_1, \dots, v_n \rangle.P$ est le processus prêt à envoyer les noms $v_1, \dots, v_n$ sur le canal a, puis à continuer selon le processus P.
- $a(x_1, \dots, x_n).P$ est le processus prêt à recevoir des noms $x_1, \dots, x_n$ sur le canal a, puis à continuer selon le processus P.
- $!a(x_1, \dots, x_n).P$ (« réception répliquée ») est le processus prêt, une infinité de fois, à recevoir des noms $x_1, \dots, x_n$ sur le canal a pour ensuite continuer selon P.
- $P \mid Q$ est le processus qui exécute P parallélement à Q.
- $P + Q$ est le processus qui se poursuit soit comme P, soit comme Q, mais pas les deux en même temps.
- $(\nu c)\ P$ (restriction) est le processus P muni de la variable liée c.

Remarque 20.

(i) On peut parler de « terme » plutôt que de « processus ». Néanmoins, pour éviter les confusions, on réserve ici ce mot aux éléments des algèbres de termes modélisées, et on garde celui de « processus » pour les programmes écrits en π-calcul.

(ii) Les noms peuvent être liés à un processus de trois matières : par une réception, une réception répliquée et une restriction. La convention de Barendregt demande que les noms liés soient distincts deux à deux et distincts des noms libres, ce que permet l'α-équivalence des noms liés. Pour la définir, on introduit d'abord une notion de substitution.

Définition 21 (α-équivalence des noms liés)**.**

(i) Si, pour tout $i \in [\![1, n]\!]$, le processus $P \in \Pi$ ne contient aucune occurrence du nom x_i, et que de plus $x_i \neq x_j$ et $y_i \neq y_j$ pour tous $i \neq j$, on note $P[x_1, \ldots, x_n/y_1, \ldots, y_n]$ le processus obtenu à partir de P en remplaçant, pour tout $i \in [\![1, n]\!]$, toutes les occurrences du nom y_i par le nom x_i.

(ii) Un changement de noms liés dans un processus P est le remplacement d'un sous-terme $a(y_1, \ldots, y_n).Q$ de P par $a(x_1, \ldots, x_n).Q[x_1, \ldots, x_n/y_1, \ldots, y_n]$, le remplacement d'un sous-terme $!a(y_1, \ldots, y_n).Q$ de P par $!a(x_1, \ldots, x_n).Q[x_1, \ldots, x_n/y_1, \ldots, y_n]$ ou le remplacement d'un sous-terme $(\nu y)\ Q$ de P par $(\nu x)\ Q$,

(iii) On dit que P et Q sont α-équivalents si l'on obtient Q à partir de P par un nombre fini de changements de noms liés.

(iv) Par convention, on considère deux processus α-équivalents comme égaux. Autrement dit, l'égalité $=$ est en fait définie sur les classes d'équivalence de la relation d'α-équivalence.

Remarque 22. La notion de « remplacement » n'a pas été ici formellement définie, mais elle est tout à fait analogue à celle construite pour les systèmes de réécriture.

Notation 23. Soit $n \in \mathbb{N}$.

(i) On n'écrit pas les terminaisons nulles des processus, c'est-à-dire que l'on note $\overline{a}\langle v_1, \ldots, v_n\rangle$, $a(x_1, \ldots, x_n)$ et $!a(x_1, \ldots, x_n)$ respectivement pour $\overline{a}\langle v_1, \ldots, v_n\rangle.0$, $a(x_1, \ldots, x_n).0$ et $!a(x_1, \ldots, x_n).0$.

(ii) On pose $(\nu c_1 \ldots c_n)\ P = (\nu c_1)\ \ldots\ (\nu c_n)\ P$.

(iii) On pose $\sum_{k=1}^{n} P_k = P_1 + \cdots + P_n$ et $\prod_{k=1}^{n} P_k = P_1 \mid \ldots \mid P_n$. Par convention, ces écritures désignent le processus nul lorsque $n = 0$.

Les interprétations données dans la définition 19 sont une description informelle du sens des notations utilisées, mais ne constituent pas une sémantique formelle. Pour cela, on dispose d'une congruence structurelle, qui traduit l'égalité de deux processus, ainsi que de sémantiques de réduction et de transition.

Définition 24 (Congruence structurelle)**.** La congruence structurelle du π-calcul, notée $\equiv$, traduit l'égalité entre deux processus et est donnée par les règles ci-dessous. En particulier, les lois $+$ et $\mid$ sont associatives et commutatives sur l'ensemble quotient $\Pi/\equiv$, et elles admettent le processus nul 0 comme élément neutre.

$$P \mid 0 \equiv P \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad P + 0 \equiv P \qquad P + Q \equiv Q + P$$

$$P + (Q + R) \equiv (P + Q) + R \qquad (\nu a)\ (\nu b)\ P \equiv (\nu b)\ (\nu a)\ P \qquad (\nu c)\ 0 \equiv 0$$

$$(\nu a)\ (P \mid Q) \equiv (\nu a)\ P \mid Q \quad \text{s'il n'y a aucune occurrence de } a \text{ dans } Q \qquad \frac{P \equiv P'}{(\nu a)\ P \equiv (\nu a)\ P'}$$

$$\frac{P \equiv P'}{\overline{a}\langle v_1, \ldots, v_n\rangle.P \equiv \overline{a}\langle v_1, \ldots, v_n\rangle.P'} \qquad \frac{P \equiv P'}{a(x_1, \ldots, x_n).P \equiv a(x_1, \ldots, x_n).P'}$$

$$\frac{P \equiv P'}{!a(x_1, \ldots, x_n).P \equiv !a(x_1, \ldots, x_n).P'} \qquad \frac{P \equiv P' \qquad Q \equiv Q'}{P \mid Q \equiv P' \mid Q'} \qquad \frac{P \equiv P' \qquad Q \equiv Q'}{P + Q \equiv P' + Q'}$$

Définition 25 (Sémantique de réduction)**.** La sémantique de réduction modélise le comportement d'un processus « en vase clos », c'est-à-dire la façon dont il se réduit sans interagir avec l'extérieur. Pour éviter les confusions avec la relation $\longrightarrow$ de la définition 9, on note $P \longrightarrow_\pi P'$ lorsqu'il existe une réduction de P

vers P'. On note de plus $\xrightarrow{*}_\pi$ la clotûre symétrique transitive de $\longrightarrow_\pi$.

$$(\textbf{Com}) \quad a(x_1,\dots,x_n).P \mid \overline{a}\langle v_1,\dots,v_n\rangle.Q \longrightarrow_\pi P[v_1,\dots,v_n/x_1,\dots,x_n] \mid Q$$

$$(\textbf{Rep}) \quad !a(x_1,\dots,x_n).P \mid \overline{a}\langle v_1,\dots,v_n\rangle.Q \longrightarrow_\pi !a(x_1,\dots,x_n).P \mid P[v_1,\dots,v_n/x_1,\dots,x_n] \mid Q$$

$$(\textbf{Sum}) \quad \frac{P \longrightarrow_\pi P'}{P+Q \longrightarrow_\pi P'} \qquad (\textbf{Par}) \quad \frac{P \longrightarrow_\pi P'}{P \mid Q \longrightarrow_\pi P' \mid Q} \qquad (\textbf{Res}) \quad \frac{P \longrightarrow_\pi P'}{(\nu a)\ P \longrightarrow_\pi (\nu a)\ P'}$$

$$(\textbf{Cong}) \quad \frac{P \equiv Q \qquad P' \equiv Q' \qquad P \longrightarrow_\pi P'}{Q \longrightarrow_\pi Q'}$$

Définition 26 (Processus terminant, forme normale)**.** Soit P un processus. On introduit deux définitions proches de celles qui existent sur une algèbre de termes.

(i) On dit que P est terminant s'il n'existe aucune suite $(P_n)_{n\in\mathbb{N}}$ de processus telle que $P = P_0$ et $\forall n \in \mathbb{N}$, $P_n \longrightarrow_\pi P_{n+1}$.

(ii) On dit que P' est une forme normale de P si $P \xrightarrow{*}_\pi P'$ et que P' n'admet aucune réduction.

Définition 27 (Sémantique de transition)**.** La sémantique de transition décrit le comportement d'un processus interagissant avec l'extérieur. Pour décrire ces interactions, on introduit les étiquettes ci-dessous.

$$\alpha ::= \underbrace{av_1,\dots,v_n}_{\text{Réception sur } a \text{ de } v_1,\dots,v_n} \parallel \underbrace{\overline{a}v_1,\dots,v_n}_{\text{Émission sur } a \text{ de } x_1,\dots,x_n} \parallel \underbrace{\overline{a}(v_1,\dots,v_n)}_{\text{Émission de noms \textbf{liés}}} \parallel \underbrace{\tau}_{\text{Transition interne}}$$

On note $P \xrightarrow{\alpha}_\pi P'$ lorsque P admet une transition vers P' en effectuant l'action α. Les règles de transition sont données ci-dessous.

$$(\textbf{Out}) \quad \overline{a}\langle v_1,\dots,v_n\rangle.P \xrightarrow{\overline{a}v_1,\dots,v_n}_\pi P \qquad (\textbf{In}) \quad a(v_1,\dots,v_n).P \xrightarrow{av_1,\dots,v_n}_\pi P[v_1,\dots,v_n/x_1,\dots,x_n]$$

$$(\textbf{Rep}) \quad !a(v_1,\dots,v_n).P \xrightarrow{av_1,\dots,v_n}_\pi !a(v_1,\dots,v_n).P \mid P[v_1,\dots,v_n/x_1,\dots,x_n]$$

$$(\textbf{Com}) \quad \frac{P \xrightarrow{av_1,\dots,v_n}_\pi P' \qquad Q \xrightarrow{\overline{a}v_1,\dots,v_n}_\pi Q'}{P \mid Q \xrightarrow{\tau}_\pi P' \mid Q'} \qquad (\textbf{Par}) \quad \frac{P \xrightarrow{\alpha}_\pi P'}{P \mid Q \xrightarrow{\alpha}_\pi P' \mid Q} \qquad (\textbf{Sum}) \quad \frac{P \xrightarrow{\alpha}_\pi P'}{P+Q \xrightarrow{\alpha}_\pi P'}$$

$$(\textbf{Res}) \quad \frac{P \xrightarrow{\alpha}_\pi P' \qquad \text{aucune occurrence de } a \text{ dans } \alpha}{(\nu a)\ P \xrightarrow{\alpha}_\pi (\nu a)\ P'} \qquad (\textbf{Open}) \quad \frac{P \xrightarrow{\overline{a}v_1,\dots,v_n}_\pi P'}{(\nu v_1,\dots,v_n)\ P \xrightarrow{\overline{a}(v_1,\dots,v_n)}_\pi P'}$$

$$(\textbf{Close}) \quad \frac{P \xrightarrow{av_1,\dots,v_n}_\pi P' \qquad Q \xrightarrow{\overline{a}(v_1,\dots,v_n)}_\pi Q'}{P \mid Q \xrightarrow{\tau}_\pi (\nu v_1,\dots,v_n)\ (P' \mid Q')} \qquad (+ \text{ règles symétriques})$$

Pour $\alpha \neq \tau$, on note de plus $P \overset{\alpha}{\Longrightarrow}_\pi P'$ s'il existe $(n,p) \in \mathbb{N}^2$, $(P_i)_{i\in[\![0,n]\!]}$ et $(P'_j)_{j\in[\![0,p]\!]}$ tels que $P_0 = P$, $P'_p = P'$, $\forall i \in [\![0, n-1]\!]$, $P_i \xrightarrow{\tau}_\pi P_{i+1}$, $\forall j \in [\![0, p-1]\!]$, $P'_j \xrightarrow{\tau}_\pi P'_{j+1}$ et $P_n \xrightarrow{\alpha}_\pi P'_0$. Plus informellement, la relation $\overset{\alpha}{\Longrightarrow}_\pi$ correspond à $\xrightarrow{\tau}_\pi \dots \xrightarrow{\tau}_\pi \xrightarrow{\alpha}_\pi \xrightarrow{\tau}_\pi \dots \xrightarrow{\tau}_\pi$.

On introduit enfin une relation d'équivalence comportementale appelée « bisimilarité faible » : deux processus sont bisimilaires faibles lorsqu'ils interagissent de la même façon avec l'extérieur, aux transitions internes près. Cette définition nous permettra par la suite de caractériser le fait qu'un processus encode un terme d'une algèbre.

Définition 28 (Bisimilarité faible)**.**

(i) Une relation binaire $\mathcal{R} \subseteq \Pi^2$ est appelée **simulation faible** si, pour toute étiquette $\alpha \neq \tau$, pour tout $(P, Q) \in \mathcal{R}$ et pour tout $P' \in \Pi$ tel que $P \overset{\alpha}{\Longrightarrow}_\pi P'$, il existe $Q' \in \Pi$ tel que $Q \overset{\alpha}{\Longrightarrow}_\pi Q'$ et $P' \mathcal{R} Q'$.

(ii) Une relation binaire $\mathcal{B} \subseteq \Pi^2$ est appelée **bisimulation faible** si $\mathcal{B}$ et $\mathcal{B}^{-1} = \{(Q, P) \mid (P, Q) \in \mathcal{B}\}$ sont des simulations faibles.

(iii) On dit que deux processus P et Q sont **bisimilaires faibles**, ce que l'on note $P \sim Q$, s'il existe une bisimulation faible $\mathcal{B}$ telle que $P \mathcal{B} Q$.

La proposition suivante montre que la relation $\sim$ est la plus grande des bisimulations faibles, puisque c'en est l'union par définition.

Proposition 29. *La bisimilarité faible $\sim$ est une bisimulation faible.*

Preuve. Soit P et Q tels que $P \sim Q$ (i.e. $Q \sim^{-1} P$) et une étiquette $\alpha \neq \tau$. Alors on a une bisimulation faible $\mathcal{B}$ telle que $P \mathcal{B} Q$, i.e. $Q \mathcal{B}^{-1} P$. Supposons avoir P' tel que $P \overset{\alpha}{\Longrightarrow}_\pi P'$. Comme $\mathcal{B}$ est une simulation faible, il existe Q' tel que $Q \overset{\alpha}{\Longrightarrow}_\pi Q'$ et $P' \mathcal{B} Q'$, d'où $P' \sim Q'$, ce qui montre que $\sim$ est une simulation faible. Supposons maintenant avoir Q' tel que $Q \overset{\alpha}{\Longrightarrow}_\pi Q'$. Comme $\mathcal{B}^{-1}$ est une simulation faible, il existe alors P' tel que $P \overset{\alpha}{\Longrightarrow}_\pi P'$ et $Q' \mathcal{B}^{-1} P'$, d'où $Q' \sim^{-1} P'$, ce qui montre de même que $\sim^{-1}$ est une simulation faible et conclut la preuve. □

Proposition 30. *La bisimilarité faible est une relation d'équivalence.*

Preuve. La bisimilarité faible est réflexive car $=$ est une bisimulation faible, et symétrique car si $\mathcal{B}$ est une bisimulation faible, alors $\mathcal{B}^{-1}$ en est une également. Pour la transitivité, soit P, Q et R tels que $P \sim R$ et $R \sim Q$. On a donc deux bisimulations faibles $\mathcal{B}'$ et $\mathcal{B}''$ telles que $P \mathcal{B}' R$ et $R \mathcal{B}'' Q$, ce qui permet d'écrire $P \mathcal{B} Q$ en posant $\mathcal{B} = \{(P, Q) \in \Pi^2 \mid \exists R \in \Pi,\ P \mathcal{B}' R \text{ et } Q \mathcal{B}'' R\}$. Il reste à vérifier que $\mathcal{B}$ est une bisimulation faible. Soit donc $(P, Q) \in \mathcal{B}$ et une étiquette $\alpha \neq \tau$. Alors on a R tel que $P \mathcal{B}' R$ et $R \mathcal{B}'' Q$. Supposons avoir P' tel que $P \overset{\alpha}{\Longrightarrow}_\pi P'$. Comme $P \mathcal{B}' R$, on a R' tel que $R \overset{\alpha}{\Longrightarrow}_\pi R'$ et $P' \mathcal{B} R'$, et comme $R \mathcal{B}'' Q$, on a Q' tel que $Q \overset{\alpha}{\Longrightarrow}_\pi Q'$ et $R' \mathcal{B} Q'$. Alors, par définition, $P' \mathcal{B} Q'$, et $\mathcal{B}$ est une simulation faible. En utilisant $(\mathcal{B}')^{-1}$ et $(\mathcal{B}'')^{-1}$, on prouve de façon analogue que $\mathcal{B}^{-1}$ en est une également. □

3.2 Encodage d'une algèbre de termes

On souhaite maintenant encoder en π-calcul les algèbres telles qu'elles ont été définies dans la première section. L'intérêt est de pouvoir ensuite utiliser la puissance du π-calcul pour les manipuler, et en particulier pour en calculer les réécritures. Cette démarche est analogue à ce qui est couramment fait en λ-calcul, système formel également Turing-complet fondé sur les notions de fonction et d'application d'une fonction, où l'on dispose de constructions usuelles pour les entiers (entiers de Church), les couples, les listes, ou encore les arbres binaires. Le principe de la construction opérée vient de l'article [Cha19], avec deux légères différences : les constante n'émettent, conformément à leur arité nulle, aucun argument sur le canal de retour leur correspondant, et l'encodage d'un terme est défini comme un processus paramétré par un canal. On y ajoute une nouveauté : la notion d'encodage inverse, permettant de traduire le fait qu'un terme est encodé sur un canal.

Dans cette sous-section, on se donne une signature $\Sigma = \{(f_i, a_i)\}_{i \in [\![1,n]\!]}$.

Définition 31 (Encodage)**.** Soit $i \in [\![1, n]\!]$, $(t_1, \ldots, t_{a_i}) \in \mathrm{T}(\Sigma)^{a_i}$, et p un canal. L'encodage de $t = f_i(t_1, \ldots, t_{a_i}) \in \mathrm{T}(\Sigma)$ en π-calcul sur le canal p est défini récursivement comme le processus :

$$[\![t]\!]_p = !p(r_1, \ldots, r_n).(\nu p_1 \ldots p_{a_i})\ (\textstyle\prod_{k=1}^{a_i} [\![t_k]\!]_{p_k} \mid \overline{r_i}\langle p_1, \ldots, p_{a_i}\rangle)\,.$$

Ainsi, lorsque l'on dispose d'un canal p et que l'on souhaite décomposer le terme t encodé dessus, il faut procéder comme suit.

— Créer n canaux $r_1, \ldots, r_n$ et les envoyer sur p.

— Écouter sur ces canaux. Un seul d'entre eux, disons r_i, va renvoyer un message, ce qui indique que la racine de t est f_i.

— Les a_i canaux reçus sur r_i sont les arguments de la racine. Si $a_i \neq 0$, il faut répéter l'opération sur ces canaux pour poursuivre la décomposition.

Définition 32 (Encodage inverse)**.** On dit que le processus P encode le terme $t \in \mathrm{T}(\Sigma)$ sur le canal p si $P \sim [\![t]\!]_p$.

Comme $\sim$ est une relation d'équivalence d'après la proposition 30, cela entraîne en particulier le fait que $[\![t]\!]_p$ encode t sur p, et donc que les définitions 31 et 32 sont bien cohérentes entre elles.

Exemple 33 (Arithmétique de Peano)**.** On reprend la signature $\mathcal{P}$ de l'exemple 4. En renommant les canaux de retour par souci de lisibilité, on obtient, pour tout $(x, y) \in \mathrm{T}(\mathcal{P})^2$ et tout canal p, l'encodage ci-dessous.

$$[\![0]\!]_p = !p(z, s, a, m).\overline{z} \qquad [\![s(x)]\!]_p = !p(z, s, a, m).(\nu c)\ ([\![x]\!]_{p_1} \mid \overline{s}\langle p_1\rangle)$$

$$[\![x + y]\!]_p = !p(z, s, a, m).(\nu p_1 p_2)\ ([\![x]\!]_{p_1} \mid [\![y]\!]_{p_2} \mid \overline{a}\langle p_1, p_2\rangle)$$

$$[\![x \cdot y]\!]_p = !p(z, s, a, m).(\nu p_1 p_2)\ ([\![x]\!]_{p_1} \mid [\![y]\!]_{p_2} \mid \overline{m}\langle p_1, p_2\rangle)$$

À partir de là, on souhaite pouvoir réécrire, en π-calcul, les expressions modélisées afin d'obtenir des termes ne faisant intervenir que le zéro et l'opérateur « suivant ». Pour cela, on a besoin d'un service π-calcul qui applique les règles de réécriture de la définition 11.

4 Traduction d'un système de réécriture en service π-calcul

On souhaite désormais créer un service π-calcul permettant de réécrire automatiquement les termes d'une algèbre en fonction d'un système de réécriture donné. On se munit pour cette section d'une signature $\Sigma = \{(f_i, a_i)\}_{i \in [\![1,n]\!]}$ et d'un système de réécriture sur celle-ci. Pour tout $i \in [\![1, n]\!]$, on note m_i le nombre de règles de réécriture dont la racine à gauche est f_i, et ces règles sont notées $r_{ij} \longrightarrow r'_{ij}$, pour $j \in [\![1, m_i]\!]$. Pour $i \in [\![1, n]\!]$ et $j \in [\![1, m_i]\!]$, on note $x_1^{ij}, \ldots, x_{p_{ij}}^{ij}$ les éléments de $\mathrm{V}(r_{ij})$.

Dans la sous-section 4.1, on présente les différents processus de l'algorithme de réécriture sous la forme de schémas. Les processus en tant que tels sont donnés dans la sous-section 4.2. Un énoncé formel de correction est proposé dans la sous-section 4.3.

4.1 Principe général

On construit dans un premier temps, pour $i \in [\![1, n]\!]$ et $j \in [\![1, m_i]\!]$, un serveur app_{ij} (figure 3), chargé d'appliquer la règle $r_{ij} \longrightarrow r'_{ij}$ à la racine d'un terme donné, si cela est possible. Il prend en entrée un canal p sur lequel est encodé un terme $t \in \mathrm{T}(\Sigma)$ et deux canaux de retour p et q. Si $t = {r_{ij}}^{\sigma}$ pour une substitution σ telle que $\mathrm{V}(r_{ij}) \subseteq \mathrm{supp}(\sigma)$, il émet sur r un canal p' où est encodé ${r'_{ij}}^{\sigma}$. Sinon, il émet un message vide sur q.

On construit alors un serveur *rec* (figure 4), qui prend en entrée un canal p où est encodé un terme t, ainsi que deux canaux de retour r et q. Dans le cas où t est réductible, il émet sur r un canal où est encodé un terme t' tel que $t \longrightarrow t'$. Dans le cas où t est irréductible, il émet un message vide sur q. Son principe de fonctionnement est le suivant : on appelle récursivement *rec* sur les arguments de la racine de t, et on cherche une réécriture applicable à la racine. Si une réécriture a été possible quelque part (au niveau de la racine ou d'un sous-terme), on renvoie sur r le terme avec ses arguments réécrits. Sinon, on émet sur q.

Dans le cas où le système de réécriture est terminant, on peut en plus construire un serveur *eval* (figure 5) prenant en entrée un canal p sur lequel est encodé un terme t, ainsi qu'un canal de retour r sur lequel il renvoie un canal où est encodé une forme normale t' de t. Son fonctionnement est simple : appeler *rec* autant de fois que nécessaire, jusqu'à obtenir un terme irréductible, ce qui arrive toujours dans le cas d'un système de réécriture terminant.

4.2 Implémentation formelle

On définit ici formellement les processus π-calcul représentant les serveurs dont le fonctionnement est décrit dans la sous-section 4.1. On les a séparés en sous-processus pour faciliter la lecture et car certains sont définis récursivement.

Définition 34 (Serveur d'application d'une règle)**.** Soit $i \in [\![1, n]\!]$ et $j \in [\![1, m_i]\!]$.

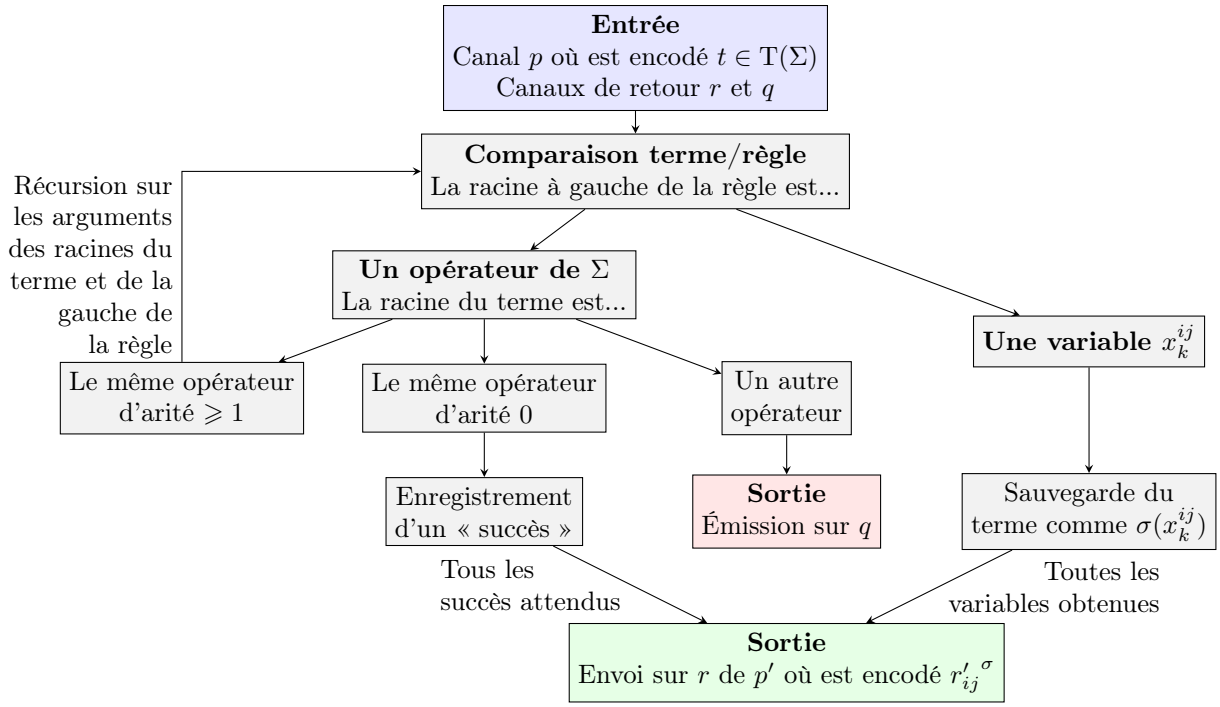

FIGURE 3 – Principe de l'algorithme exécuté par app_{ij}

(i) Pour tout $r \in \mathrm{ST}(r_{ij})$, le processus $\mathrm{Ident}_{ij}(r)$ est défini récursivement selon la racine de r.
— Si $\mathrm{R}(r) = x_k^{ij} \in \mathcal{V}$, $\mathrm{Ident}_{ij}(r) = \overline{x_k}\langle p\rangle$.
— Si $\mathrm{R}(r) = f_k \in \Sigma_0$, $\mathrm{Ident}_{ij}(r) = (\nu r_1 \dots r_n)\ (\overline{p}\langle r_1, \dots, r_n\rangle.(\sum_{k' \neq k} r_{k'}(p_1, \dots, p_{a_{k'}}).\overline{f} + r_k.\overline{s}))$.
— Si $\mathrm{R}(r) = f_k \notin \Sigma_0 \cup \mathcal{V}$ et $r = f_k(r_1, \dots, r_{a_k})$, alors :

$$\begin{aligned}\mathrm{Ident}_{ij}(r) = (\nu r_1 \dots r_n)\ (\overline{p}\langle r_1, \dots, r_n\rangle.(\textstyle\sum_{k' \neq k} r_{k'}(p_1, \dots, p_{a_{k'}}).\overline{f} \\ + r_k(p_1, \dots, p_{a_k}).\textstyle\prod_{k'=1}^{a_k} \mathrm{Ident}_{ij}(r_{k'})[p_{k'}/p]))\,.\end{aligned}$$

(ii) Pour tout $r \in \mathrm{ST}(r'_{ij})$ et tout canal c, on définit récursivement le processus $\mathrm{TermeRéécrit}_{ij}^{c}(r)$.
— Si $\mathrm{R}(r) = x_k^{ij} \in \mathcal{V}$, $\mathrm{TermeRéécrit}_{ij}^{c}(r) = x_k(x).!c(r_1, \dots, r_n).\overline{x}\langle r_1, \dots, r_n\rangle$.
— Si $r = f_k(r_1, \dots, r_{a_k})$ pour $k \in [\![1, n]\!]$, alors :

$$\mathrm{TermeRéécrit}_{ij}^{c}(r) = !c(r_1, \dots, r_n).(\nu c_1 \dots c_{a_k})\ (\textstyle\prod_{k'=1}^{a_k} \mathrm{TermeRéécrit}_{ij}^{c_{k'}}(r_{k'}) \mid \overline{r_k}\langle c_1, \dots, c_{a_k}\rangle)\,.$$

(iii) On définit alors le processus $\mathrm{Serv}_{app_{ij}}$ par :

$$\begin{aligned}\mathrm{Serv}_{app_{ij}} = {} & !app_{ij}(p, r, q).(\nu x_1 \dots x_{p_{ij}} s f c)\ (\mathrm{Ident}_{ij}(r_{ij}) \mid f.\overline{q} \\ & \mid \underbrace{s.\cdots.s}_{m \text{ fois}}.x_1(t_1).\cdots.x_{p_{ij}}(t_{p_{ij}}).(\mathrm{TermeRéécrit}_{ij}^{c}(r'_{ij}) \mid \overline{r}\langle c\rangle))\,,\end{aligned}$$

où $m = |\mathrm{C}(r_{ij})| - |\mathrm{V}(r_{ij})|$.

Définition 35 (Serveur de réécriture)**.**

(i) Soit $i \in [\![1, n]\!]$. Le processus RecOp_i est défini par :

$$\begin{aligned}\mathrm{RecOp}_i = {} & (\textstyle\prod_{j=1}^{m_i} \overline{app_{ij}}\langle p, r', q'\rangle \mid \prod_{k=1}^{a_i} \overline{rec}\langle p_k, r_k, q_k\rangle). \\ & ((\textstyle\sum_{k=1}^{a_i}(r_k(p'_k).(\nu p')\ (!p'(r'_1, \dots, r'_n).\overline{r'_i}\langle p_1, \dots, p_{k-1}, p'_k, p_{k+1}, \dots, p_{a_i}\rangle \mid \overline{r}\langle p'\rangle)) \\ & + r'(p').\overline{r}\langle p'\rangle) \mid \underbrace{q'.\cdots.q'}_{m_i \text{ fois}}.q_1.\cdots.q_{a_i}.\overline{q})\,.\end{aligned}$$

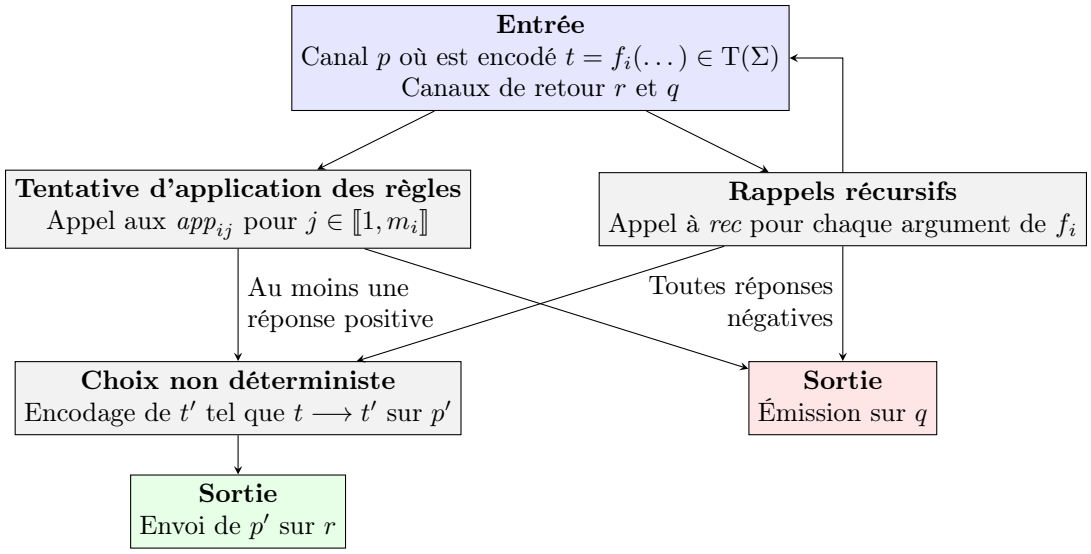

FIGURE 4 – Principe de l'algorithme exécuté par *rec*

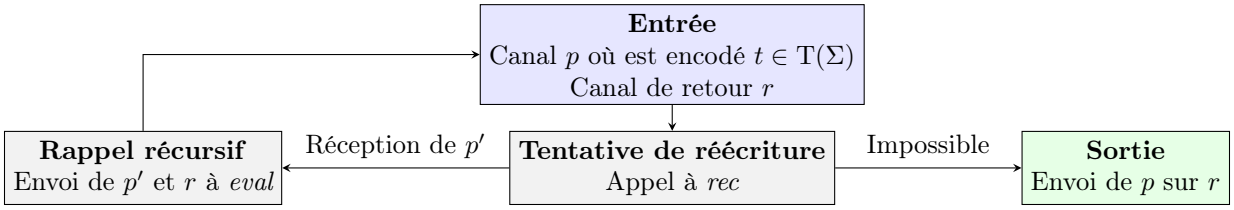

FIGURE 5 – Principe de l'algorithme exécuté par *eval*

(ii) On définit alors le processus Serv_{rec} par :

$$\begin{aligned}\mathrm{Serv}_{rec} = {} & !rec(p,r,q).(\nu c_1 \dots c_n)\ (\overline{p}\langle c_1,\dots,c_n\rangle.\textstyle\sum_{i=1}^{n}(c_i(p_1,\dots,p_{a_i}).(\nu r' q' r_1 q_1 \dots r_{a_i} q_{a_i})\ \mathrm{RecOp}_i)) \\ & \mid \textstyle\prod_{i=1}^{n}\prod_{j=1}^{m_i}\mathrm{Serv}_{app_{ij}}\,.\end{aligned}$$

Définition 36 (Serveur d'évaluation)**.** Si le système de réécriture défini sur Σ est terminant, on définit le processus Serv_{eval} par :

$$\mathrm{Serv}_{eval} = !eval(p,r).(\nu r' q)\ (\overline{rec}\langle p, r', q\rangle.(q.\overline{r}\langle p\rangle + r'(p').\overline{eval}\langle p', r\rangle)) \mid \mathrm{Serv}_{rec}\,.$$

4.3 Correction

Je n'ai pas eu le temps de proposer une preuve de correction au cours du stage. Cette sous-section vise uniquement à donner l'énoncé des théorèmes de terminaison et de validité des serveur *rec* et *eval*, en utilisant la notion d'encodage inverse donnée dans la définition 32. Cela ouvre la voie à une future preuve.

Conjecture 37 (Terminaison et validité du serveur *rec*)**.** *Soit P un processus encodant $t \in \mathrm{T}(\Sigma)$ sur le canal p. Alors il existe un unique processus Q tel que $P \mid Serv_{rec} \xrightarrow{recp,r,q}_{\pi} Q$. De plus, ce processus Q est terminant, et les trois points suivants sont vérifiés.*

(i) Si t est irréductible, toute forme normale Q' de Q est bisimilaire faible à $P \mid Serv_{rec} \mid \overline{q}$.

(ii) Si t est réductible, toute forme normale Q' de Q renvoie un terme $t' \in \mathrm{T}(\Sigma)$ tel que $t \longrightarrow t'$, c'est-à-dire qu'il existe un canal p' et un processus P' encodant t' sur p' tels que $Q' \xrightarrow{\overline{r}(p')}_{\pi} \equiv P \mid P' \mid Serv_{rec}$.

(iii) Pour tout $t' \in \mathrm{T}(\Sigma)$, $t \longrightarrow t'$ si et seulement s'il existe une forme normale Q' de Q renvoyant t' au sens ci-dessus.

Conjecture 38 (Terminaison et validité du serveur *eval*)**.** *Supposons le système de réécriture défini sur Σ terminant et soit P un processus encodant $t \in \mathrm{T}(\Sigma)$ sur le canal p. Alors il existe un unique processus Q tel que $P \mid Serv_{eval} \xrightarrow{evalp,r}_{\pi} Q$. De plus, ce processus Q est terminant et les deux points suivants sont vérifiés.*

(i) Toute forme normale Q' de Q renvoie une forme normale t' de t, c'est-à-dire qu'il existe un canal p' et un processus P' encodant t' sur p' tels que $Q' \xrightarrow{\overline{r}(p')}_{\pi} \equiv P \mid P' \mid Serv_{eval}$.

(ii) Pour tout $t' \in \mathrm{T}(\Sigma)$, t' est une forme normale de t si et seulement s'il existe une forme normale Q' de Q renvoyant t' au sens ci-dessus.

4.4 Application à l'évaluation de l'arithmétique de Peano

Considérons la signature $\mathcal{P}$ de l'exemple 4 muni du système de réécriture de la définition 11. Par les sous-sections précédentes, on a obtenu un serveur *eval* permettant de calculer les expressions de l'arithmétique de Peano encodées en π-calcul. Le cas général ayant été traité, on ne va pas réécrire le code dans ce cas particulier, mais l'utiliser pour implémenter de nouvelles fonctions sur les termes de $\mathrm{T}(\mathcal{P})$. On peut par exemple créer des serveurs *suiv* et *prec* donnant le successeur et le prédécesseur d'un terme (en considérant que le prédécesseur de 0 est 0).

$$\mathrm{Serv}_{suiv} = !suiv(n,r).(\nu pr')\ (\overline{eval}\langle n, r'\rangle.r'(n').(!p(z,s,a,m).\overline{s}\langle n'\rangle) \mid \overline{r}\langle p\rangle) \mid \mathrm{Serv}_{eval}$$

$$\mathrm{Serv}_{prec} = !prec(n,r).(\nu r' zsam)\ (\overline{eval}\langle n, r'\rangle.r'(n').\overline{n'}\langle z,s,a,m\rangle.(z.\overline{r}\langle n'\rangle + s(n'').\overline{r}\langle n''\rangle)) \mid \mathrm{Serv}_{eval}$$

Pour le serveur *prec*, l'idée est que la forme normale d'un terme ne peut pas contenir d'additions et de multiplications, et que seuls les cas où la racine est 0 ou s sont donc à traiter.

On peut également créer un serveur récursif permettant le calcul de la factorielle d'un entier.

$$\begin{aligned}\mathrm{Serv}_{fact} = {} & !fact(n,r).(\nu r' r'' pzsam)\ (\overline{eval}\langle n, r'\rangle.r'(n_1).\overline{n_1}\langle z,s,a,m\rangle.(z.(!p(z',s',a',m').\overline{s'}\langle n_1\rangle \mid \overline{r}\langle p\rangle) \\ & + s(n_2).\overline{fact}\langle n_2, r''\rangle.r''(n_3).(!p(z',s',a',m').\overline{m'}\langle n_1, n_3\rangle \mid \overline{eval}\langle p, r\rangle))) \mid \mathrm{Serv}_{eval}\end{aligned}$$

5 Conclusion

5.1 Bilan du travail réalisé

L'apport de ce travail est de proposer une implémentation de la théorie classique de la réécriture de termes, développée dans [BN98], en π-calcul. Pour cela, j'ai repris le travail d'Amel Chadda dans [Cha19] sur l'encodage des algèbres en π-calcul en y ajoutant une notion d'encodage inverse, puis j'ai écrit un serveur réécrivant un terme lorsque cela est possible suivant un système de réécriture donné. Ce serveur utilise la théorie de la concurrence pour tester toutes les règles en même temps et fait un choix non déterministe parmi les réécritures possibles, de sorte que chacune d'entre elles corresponde à une π-réduction réalisable. J'ai de plus introduit les notions nécessaires afin que la terminaison et la validité de cette implémentation puissent être énoncées formellement. En revanche, je n'ai pas eu le temps de me pencher sur sa preuve, et ce serait la première chose à faire pour poursuivre ce stage.

Par la suite, plusieurs applications de cet encodage peuvent être envisagées, comme la réécriture de l'arithmétique de Peano, exemple pris tout au long de l'article, mais aussi de manière plus générale la traduction de toute fonction calculable sur les algèbres de termes en π-calcul, dans la mesure où la réécriture de termes est Turing-complète. On pourrait également implémenter en Go les systèmes de réécriture. L'article [Cha19] propose une bibliothèque de services manipulant les algèbres en Go, à partir du compilateur de π-calcul créé par Marco Guinti. Le travail restant est donc l'écriture en Go des définitions de la sous-section 4.2.

Une dernière idée serait d'étendre le travail d'encodage des algèbres et de leurs réécritures à des structures plus complexes en ajoutant des conditions sur les arguments des opérateurs. Si Σ est une signature, on pourrait encoder des listes d'éléments de $\mathrm{T}(\Sigma)$ en considérant la signature $\Sigma \cup \{(v, 0), (\ell, 2)\}$, où v désignerait une liste vide et ℓ un opérateur d'ajout d'un élément prenant un premier paramètre de racine v ou ℓ et un second dans $\mathrm{T}(\Sigma)$. On exigerait de plus que les opérateurs de Σ ne prennent que des éléments de $\mathrm{T}(\Sigma)$ en arguments. Partitionner ainsi la signature ne pose pas de difficulté supplémentaire, et cette démarche ouvrirait une voie amusante : la possibilité d'encoder le π-calcul et sa sémantique de réduction en π-calcul.

5.2 Apport personnel du stage

Ce stage a été réalisé en septembre 2021, avant ma troisième année de licence de mathématiques à Berlin qui ne commençait que mi-octobre. Il n'était pas prévu dans mon cursus et je remercie tout particulièrement Romain Demangeon d'avoir accepté de l'encadrer. Pour cela, j'ai suivi durant l'été des enregistrements vidéo de ses cours donnés en master, afin d'avoir une certaine connaissance du sujet avant de commencer le stage.

Je suis intéressé aussi bien par les mathématiques que par l'informatique, donc la découverte d'un sujet – les algèbres de processus et plus généralement l'informatique théorique – croisant des deux matières m'a été très utile pour affiner mon projet d'orientation. Étant en licences de mathématiques et de sciences sociales dans le cadre d'un double cursus, je n'avais pas eu beaucoup de cours d'informatique et être dans ce cadre pendant un mois, lors duquel j'ai aussi assisté à des cours de master, m'a permis de fortement progresser. Enfin, ce stage dans un laboratoire a été une opportunité de découvrir ce qu'est vraiment la recherche, en échangeant avec mon maître de stage, d'autres chercheurs et doctorants du LIP6, et en assistant à une soutenance de thèse.

Mon projet de recherche, qui m'a amené à introduire de nouveaux objets ou à en réinventer certains existants (je n'ai pu me procurer un exemplaire de [BN98] qu'après deux semaines de stage), m'a fait prendre conscience de la difficulté de la définition formelle en mathématiques. La majorité de mon temps de réflexion a servi à trouver des définitions ou à revenir sur celles déjà écrites afin de les rendre suffisament rigoureuses, pertinentes mais aussi « canoniques ». Les cours de licence nous apprennent à démontrer des résultats, mais pas tellement à imaginer des définitions puisque nous étudions des théories déjà construites. Cette découverte a non seulement été une surprise pour moi, mais aussi une occasion de beaucoup progresser en formalisme. Je pense avoir amélioré ma perception du niveau de rigueur nécessaire dans un contexte assez éloigné de celui d'un exercice de mathématiques.

6 Références

[BN98] Franz Baader et Tobias Nipkow : *Term rewriting and all that.* Cambridge University Press, 1998.

[Cha19] Amel Chadda : Analyse d'efficacité de services en π-calcul. Rapport de stage au LIP6, 2019.

[DA14] Romain Demangeon et Carlos Agon : Paradigmes de programmation concurrente. Cours de M2 à Sorbonne Université, 2014.

[SW01] Davide Sangiorgi et David Walker : *The π-calculus : a theory of mobile processes.* Cambridge University Press, 2001.