Typage et déduction dans le calcul de réécriture

Benjamin Wack

Département de formation doctorale en informatique **École doctorale IAEM Lorraine**
UFR STMIA

Typage et déduction dans le calcul de réécriture

THÈSE

présentée et soutenue publiquement le 7 octobre 2005

pour l'obtention du

Doctorat de l'université Henri Poincaré – Nancy 1
(spécialité informatique)

par

Benjamin Wack

Composition du jury

Président :	Mariangiola Dezani	Professeur, Università di Torino, Italie
Rapporteurs :	Gilles Dowek	Professeur, École Polytechnique, Palaiseau, France
	Herman Geuvers	Professeur, Radboud Universiteit, Nijmegen, Pays-Bas
Examinateurs :	Adam Cichon	Professeur, Université Henri Poincaré, Nancy, France
	Claude Kirchner	Directeur de recherche, INRIA, Nancy, France
	Luigi Liquori	Chargé de recherche, INRIA, Sophia Antipolis, France

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thloria.

Remerciements

Claude Kirchner a eu la patience de diriger cette thèse et de discuter très régulièrement avec moi de l'avancée de mes travaux, et ce malgré son emploi du temps qui frôle les 35 heures par jour. Il a su écouter mes doutes et mes doléances, et me conseiller sans jamais rien m'imposer.

Luigi Liquori, bien qu'il ait quitté Nancy à la fin de ma première année de thèse, a su continuer à travailler avec moi et a patiemment relu et corrigé mes productions écrites. Mon obstination à ne pas tenir compte de certaines de ses remarques n'a eu d'égal que son entêtement à relever la moindre virgule qui ne lui convenait pas.

Horatiu Cirstea, sans être officiellement mon encadrant, fait certainement partie des personnes qui connaissent et comprennent le mieux le contenu de ce manuscrit. Je ne peux qu'espérer qu'il a apprécié autant que moi notre travail commun.

Gilles Dowek et Herman Geuvers me font le très grand honneur d'être rapporteurs de cette thèse, eux dont les manuscrits et exposés sont pour moi des modèles de pédagogie en plus d'être des mines d'information. La seule idée que ce manuscrit allait être relu par eux a certainement été pour beaucoup dans les efforts que j'ai faits pour écrire des démonstrations complètes et convaincantes. Je me surprends à espérer que l'emploi du français dans ce document a évité à Herman d'y voir toutes les imperfections dont il est certainement chargé, mais je me fais peu d'illusions.

Mariangiola Dezani, dont j'ai déjà pu apprécier la vivacité scientifique à plusieurs reprises, me fait le plaisir de présider le jury ; de plus, elle a pris le temps de lire ce manuscrit en détails, et ce malgré la barrière de la langue.

Adam Cichon a accepté d'être examinateur, ce qui me donne l'occasion de lui signifier le plaisir que j'ai eu à travailler avec lui pour différents enseignements.

Je dois avouer que, dans un premier jet de ces remerciements, j'avais oublié que le français n'est pas la langue maternelle d'Adam, Horatiu et Luigi. Je m'efforce de croire qu'ils le prendront comme un compliment.

Il semble d'usage chez les aspirants docteurs de remercier leurs parents pour leur soutien ; je suis pour ma part infiniment reconnaissant aux miens de m'avoir octroyé une paix royale pendant ces trois ans et même depuis bien plus longtemps.

En travaillant au Loria, j'ai eu presque quotidiennement l'occasion d'apprécier les conversations tour à tour politiques, scientifiques, sociologiques, artistiques ou loufoques mais toujours enrichissantes de Hacène, Huy, Jérôme, Joseph, Laïka et Sylvain, et ça méritait d'être dit.

Je remercie Serge Autexier et Christoph Benzmüller, et de façon plus générale le groupe Omega pour l'accueil très amical qu'ils m'ont fait à Saarbrücken en février 2005.

Je finirai par une pensée pour Bruno, Chloé, François, Hélène, Julien, Pascal, Pierre et les autres agrégatifs grâce à qui les mathématiques ont été distrayantes en plus d'être une distraction.

Logic is the art of thinking and reasoning in strict accordance with the limitations and incapacities of the human misunderstanding.
Ambrose Bierce, The Devil's Dictionary.

Table des matières

Introduction

Les premières utilisations intensives de l'informatique dans la démonstration mathématique remontent aux années 70. L'exemple le plus connu en est certainement la démonstration du théorème des quatre couleurs par K. Appel et W. Haken [AHK77] à l'aide d'un programme qui vérifia systématiquement quelques 1400 cas parmi lesquels devrait nécessairement se trouver un hypothétique contre-exemple. Si le nombre de cas à examiner a été réduit depuis, il est remarquable qu'à ce jour on ne connaisse toujours pas de démonstration vérifiable à la main de ce théorème.

Cependant, l'utilisation de logiciels écrits expressément pour traiter un problème mathématique donné est sévèrement limitée. D'une part, la correction de ces logiciels n'est pas garantie et la démonstration fournie n'est donc acceptable qu'après plusieurs vérifications indépendantes. D'autre part, le raisonnement proprement dit reste du ressort de l'humain, et on ne peut donc pas réellement parler de démonstration automatique. Les travaux sur les λ-calculs typés, initiés par A. Church [Chu40] et H. B. Curry [Cur34] remédient à ces problèmes en fournissant un cadre très général pour la démonstration assistée par ordinateur. En effet, le λ-calcul fournit une représentation linéaire des démonstrations, facile à manipuler par la machine indépendemment des mécanismes retenus pour la démonstration et l'interaction avec l'utilisateur. La seule procédure dont la correction est cruciale est la vérification des démonstrations, qui se ramène en fait à une vérification de typage pour un λ-terme ; ce *noyau* logiciel est en général réduit et donc facile à certifier. Dans ce cadre, l'intégralité du raisonnement est vérifiée par la machine, ce qui permet de corriger d'éventuelles erreurs de déduction dues à l'humain. Ainsi, la démonstration du théorème des quatre couleurs faite récemment à l'aide du logiciel Coq par G. Gonthier [Gon05] constitue pleinement une démonstration assistée par ordinateur, *a priori* plus fiable que les précédentes.

Deux écoles d'assistants à la démonstration ont évolué parallèlement. On trouve ainsi les logiciels fondés sur l'isomorphisme de Curry-Howard-de Bruijn [How80], les premiers ayant été développés dans le projet AUTOMATH de N. G. de Bruijn [Bru70]. Ces assistants reposent sur la correspondance entre un λ-terme et une démonstration en déduction naturelle, ainsi qu'entre un type et une proposition. Ils sont donc en quelque sorte confinés au paradigme de la déduction naturelle ; en revanche, leur puissance déductive peut être ajustée selon le système de types choisi, puisqu'une démonstration correspond directement à une dérivation de typage.

L'autre école rassemble les différentes extensions du *Logical Framework* d'Edinburgh, introduit par R. Harper, F. Honsell et G. Plotkin [HHP93]. L'idée centrale est ici de considérer un système de types relativement faible, dépourvu de constructeurs de types d'ordre supérieur. Par contre, ce système est suffisant pour manipuler des contextes et lier ou instancier des variables, ce qui est généralement primordial pour la spécification d'un système formel. Toutes les structures d'un système (expressions, assertions, règles de déduction) doivent alors être décrites dans ce métalangage au moyen de constantes dédiées. Par exemple, à une règle de déduction donnée correspond une constante dont le type exprime qu'elle prend en arguments des démonstrations des

prémisses de la règle et renvoie une démonstration de la conclusion. Les assertions elles-mêmes sont exprimées par des termes du métalangage, et on se donne généralement une constante qui exprime qu'une assertion est démontrable. Le Logical Framework est donc une utilisation moins directe mais plus souple du λ-calcul comme représentation des démonstrations.

Dès ces débuts, il est apparu qu'on ne pourrait construire des démonstrations de grande envergure que si le formalisme sous-jacent possédait une puissance calculatoire suffisante. L'évolution vers une automatisation des calculs nécessaires à la démonstration se voit particulièrement bien dans le projet Coq [Coq04]. À la base des premières versions on trouve le Calcul des Constructions, proposé par Th. Coquand et G. Huet [CH88], qui combine le pouvoir déductif de la logique d'ordre supérieur et des types dépendants avec l'expressivité du polymorphisme. Par la suite, Th. Coquand et Ch. Paulin-Mohring [CP90] ont étendu le formalisme avec des types primitifs inductifs pour généraliser la relation de convertibilité sur les types. Enfin, la possibilité d'ajouter une relation de réécriture à cette relation de conversion, ce qui ajoute encore à la capacité calculatoire du système, a été étudiée successivement par J. Gallier et V. Breazu-Tannen [GBT89], F. Barbanera [Bar90], J.-P. Jouannaud et M. Okada [JO95], M. Fernández [Fer93] et F. Blanqui [Bla05b].

D'un point de vue plus fondamental, l'idée de considérer une démonstration comme indépendante des diverses étapes de calcul qui peuvent y advenir est connue sous le nom de *principe de Poincaré* [BB02]. G. Dowek, Th. Hardin et C. Kirchner ont traduit ce principe pour les systèmes de déduction formels tels que la déduction naturelle ou le calcul des séquents, obtenant ainsi le concept de *déduction modulo* [DHK03]. Dans ce système formel, les règles d'inférence sont toutes définies modulo une congruence sur les propositions, elle-même généralement donnée sous forme d'un système de réécriture.

Il est intéressant de noter une certaine uniformité de conception dans tous ces travaux : les auteurs partent d'un système purement déductif existant, auquel ils ajoutent une forme de réécriture qui se charge du calcul. Si on a alors bien les deux aspects à disposition, ils restent relativement disjoints, et il faut parler de *coopération* entre calcul et déduction plutôt que d'une réelle intégration. Une question qui se pose donc naturellement est de déterminer les résultats que l'on pourrait obtenir à partir d'un unique formalisme dans lequel calcul et déduction sont uniformément représentés.

Dans cette thèse, nous apportons des éléments de réponse à cette question, puisque nous étudions en quoi l'introduction du filtrage dans un calcul typé peut influer sur la logique et les démonstrations qu'il permet de représenter. Plus particulièrement, nous nous intéressons au calcul de réécriture (ou ρ-calcul) et à ses diverses variantes. Ce formalisme, qui étend le λ-calcul par des motifs et une théorie de filtrage *a priori* arbitraire, a été introduit par H. Cirstea et C. Kirchner [CK01], initialement afin de décrire une sémantique du langage de règles et de stratégies ELAN [BCD$^+$00].

Nous proposons plusieurs versions typées du ρ-calcul. Les premières sont fortement inspirées du λ-calcul simplement typé et du système de types polymorphe introduit par J.-Y. Girard [GLT89]. Les propriétés usuelles de ce genre de systèmes de types restent valides : la préservation du type affirme qu'un terme bien typé conserve le même type au cours de ses réductions ; l'unicité du typage assure qu'il n'y a pas d'ambiguïté dans la façon de construire un type pour un terme donné ; la décidabilité du typage garantit que le calcul d'un type peut être effectué et vérifié automatiquement. En revanche, nous verrons que, grâce au filtrage, ces systèmes de type ne permettent que d'assurer un typage faible, au sens où ils autorisent la définition de points fixes (ce qui n'est pas le cas pour le λ-calcul).

Il est alors possible de représenter assez facilement des programmes. D'une part, on dispose presque directement d'un paradigme fonctionnel avec motifs à la ML, pour lequel nous détaillerons l'inférence de types polymorphes. D'autre part, on peut modéliser des programmes définis par des règles de réécriture ainsi que certaines stratégies guidant l'application de ces règles. Par contre, la terminaison est souvent très liée à la consistance, et le ρ-calcul n'échappe pas à la règle : les systèmes de déduction correspondant à ces premiers systèmes de types sont logiquement incohérents si l'on ne restreint pas fortement les constantes utilisées dans les motifs.

Une autre famille de systèmes de types a donc été établie, sur la base des Systèmes de Types Purs (PTS) de S. Berardi [Ber88] et J. Terlouw [Ter89], qui sont d'autres systèmes de types pour le λ-calcul. Dans l'adaptation pour le ρ-calcul, appelée P^2TS, les types sont plus riches que dans le système simplement typé, car les abstractions, et donc les motifs, y sont propagés sous forme d'un autre opérateur de liaison qui permet de définir des types dépendants.

Encore une fois les propriétés usuelles sont vérifiées, mais il apparaît que, contrairement au cas du λ-calcul, la formulation usuelle des types simples et son expression dans le cadre des P^2TS ne sont pas équivalentes. En effet, nous verrons que l'on peut démontrer la normalisation forte des termes acceptés dans les P^2TS $\rho_{\rightarrow}$ et ρP, ce qui exclut la formation des points fixes mentionnés précédemment. La démonstration de normalisation forte proprement dite se fait au moyen d'une réduction de $\rho_{\rightarrow}$ vers le système $\lambda\omega$ du λ-cube, et d'une réduction de ρP vers $\rho_{\rightarrow}$.

Enfin, munis du résultat de forte normalisation dans ρP, nous verrons comment le calcul de réécriture peut être utilisé dans le contexte des assistants à la démonstration. Encore une fois, deux approches sont possibles. La première consiste à utiliser les P^2TS pour définir un Logical Framework avec des capacités de filtrage, ce qui permet une représentation plus concise et plus naturelle des structures que l'on veut décrire dans ce cadre formel. Par ailleurs, la possibilité de représenter des règles de réécriture permet de définir naturellement des termes de preuve pour la déduction naturelle modulo dans lesquels les étapes de conversion définies par réécriture sont explicitement prises en compte. La représentation ainsi obtenue a l'avantage de rassembler dans un même ρ-terme la partie calculatoire et la partie déductive d'une démonstration, l'une ou l'autre pouvant être effacée à volonté.

La seconde approche, plus prospective, consiste à étendre la déduction naturelle de sorte qu'il existe des règles d'introduction et d'élimination non seulement pour les connecteurs logiques, mais aussi pour chaque symbole de prédicat défini par la théorie dans laquelle on se place. Nous généraliserons la notion de coupure pour tenir compte de ces nouvelles règles, et nous démontrerons la correction et la complétude de ce nouveau système de déduction par rapport à la logique du premier ordre munie d'axiomes adéquats. Nous proposerons un sous-langage du ρ-calcul comme langage de termes de preuve. L'isomorphisme de Curry-Howard-de Bruijn est alors étendu de sorte qu'une introduction correspond à une abstraction sur un motif qui comporte le symbole de prédicat introduit ; similairement, une élimination correspond à l'application d'une fonction à un argument dont le symbole de tête est le symbole éliminé. La théorie des types résultante pour le calcul de réécriture est alors complètement différente, et nous en étudierons quelques aspects métathéoriques.

Itinéraires de lecture Le chapitre 1 rappelle les définitions et propriétés de certains formalismes que nous serons amenés à utiliser par la suite. Le chapitre 2 présente brièvement trois directions de recherche parallèles à la nôtre, qui s'attachent toutes plus ou moins à mieux comprendre les relations possibles entre un système de déduction (ou de typage) et la réécriture ou le filtrage. Le chapitre 3 fait un état de l'art du calcul de réécriture, le formalisme autour

duquel l'essentiel de cette thèse est construite. Toute la suite du manuscrit est dédiée aux différents résultats issus de mes travaux. Le chapitre 4 consiste en l'étude d'une première famille de systèmes de types, dont on verra qu'ils sont trop permissifs pour définir un cadre logique, mais qui permettent de représenter de nombreux aspects de la programmation par règles. Le chapitre 5 fait l'étude d'une autre famille de disciplines de typage, plus adaptées à la déduction. L'essentiel de ce chapitre est consacré à une démonstration de normalisation forte pour deux de ces systèmes, par le biais d'une réduction vers un λ-calcul typé. Pour finir, le chapitre 6 pose les bases de l'utilisation du calcul de réécriture en tant que langage de termes de preuve pour des systèmes déductifs, en lien étroit avec la déduction modulo.

Si les sections 1.1 et 1.2 peuvent aisément être omises par le lecteur confirmé, il est recommandé de lire la section 1.3 pour se familiariser avec la déduction modulo.

Le lecteur intéressé par les aspects les plus programmatoires pourra se diriger directement vers les chapitres 3 et 4. Il trouvera également une formalisation intéressante du λ-calcul par valeurs dans la section 5.2.4.

L'amateur de théorie des types pourra se mettre en condition avec les sections 2.2 et 2.3. À la lumière de la discussion en section 4.3, il pourra étudier les systèmes proposés au chapitre 5, puis s'intéresser au système décrit en section 6.2.

Enfin, l'automatisation et l'introduction de la réécriture dans les assistants à la démonstration sont couvertes principalement par les sections 2.1 et 6.1, cette dernière s'appuyant très largement sur les résultats du chapitre 5.

La lecture intégrale du manuscrit n'est évidemment proscrite pour personne.

1

Notions préliminaires

Ce premier chapitre rassemble quelques notions auxquelles nous ferons appel, ou desquelles nous nous inspirerons par la suite. Nous traiterons notamment le λ-calcul et la réécriture, les deux formalismes à l'origine du calcul de réécriture. Nous présenterons également quelques systèmes de déduction naturelle et leurs liens avec le λ-calcul.

1.0 Notions générales

Nous allons travailler sur différents formalismes. La plupart d'entre eux incluront une algèbre de termes, qui se définit à partir d'un ensemble de constantes munies d'arités.

Définition 1 (Signature et arité)

1. *Une* signature Σ *est une fonction d'un ensemble de constantes (généralement fini) vers les entiers naturels.*
2. *L'entier* $\Sigma(f)$ *est appelé* arité *de* f*. On écrira parfois* f_n *pour signifier que* f *est d'arité* n*.*

On notera en général les constantes a, b, c si leur arité est nulle, et f, g, h sinon.

On écrira parfois abusivement Σ au lieu de $\mathcal{D}om(\Sigma)$. Ainsi l'expression $f_n \in \Sigma$ signifie que $f \in \mathcal{D}om(\Sigma)$ et $\Sigma(f) = n$.

Définition 2 (Algèbre de termes)

Étant donnés une signature Σ *et un ensemble de variables* $\mathcal{X}$*, l'*algèbre de termes *engendrée par* Σ *(sur* $\mathcal{X}$*), notée* $\mathcal{T}(\Sigma, \mathcal{X})$*, est le plus petit ensemble tel que :*

- *toute variable de* $\mathcal{X}$ *est dans* $\mathcal{T}(\Sigma, \mathcal{X})$ *;*
- *si* $f_n \in \Sigma$ *et si les termes* $t_1, \ldots, t_n \in \mathcal{T}(\Sigma, \mathcal{X})$*, alors* $f(t_1, \ldots, t_n) \in \mathcal{T}(\Sigma, \mathcal{X})$*.*

Les variables de $\mathcal{X}$ seront notées x, y, z ; les termes algébriques seront notés s, t, u. Lorsqu'il n'y a pas d'ambiguïté sur Σ et $\mathcal{X}$, on appellera *termes algébriques* l'ensemble des termes de $\mathcal{T}(\Sigma, \mathcal{X})$.

Pour transformer des termes, que ce soit dans le λ-calcul ou pour la réécriture, le mécanisme élémentaire est la substitution.

Définition 3 (Substitution)

Étant donné un ensemble de termes $\mathcal{T}$ *pouvant comporter des variables prises dans un ensemble* $\mathcal{X}$*, une* substitution θ *est une application de* $\mathcal{X}$ *dans* $\mathcal{T}$ *qui ne diffère de l'identité que sur une partie de* $\mathcal{X}$*.*

L'ensemble des variables x de $\mathcal{X}$ telles que $\theta(x) \neq x$ est appelé domaine *de θ, noté $\mathcal{D}om(\theta)$. Si $\mathcal{D}om(\theta) = \{x_1, \ldots, x_n\}$ et pour tout i on a $\theta(x_i) = t_i$, on pourra écrire θ comme $[x_1 := t_1 \ldots x_n := t_n]$.*

Étant donnée une notion de variable libre $\mathcal{FV}$ qui à un terme de $\mathcal{T}$ associe un ensemble de variables, la substitution θ se prolonge en une application de $\mathcal{T}$ dans $\mathcal{T}$. Cette application agit sur un terme en remplaçant chacune de ses variables libres par son image par θ.

On notera cette application de façon postfixée : son action sur un terme t est $t\theta$.

Exemple 1 (Substitutions sur les termes algébriques)

Dans un terme algébrique t, on considérera généralement que toutes les variables sont libres. Par conséquent, l'action d'une substitution sur un terme algébrique peut s'écrire

$$\begin{array}{rcll} x\theta & = & \theta(x) & \text{si } x \in \mathcal{D}om(\theta) \\ x\theta & = & x & \text{si } x \notin \mathcal{D}om(\theta) \\ f(t_1, \ldots, t_n) & = & f(t_1\theta, \ldots, t_n\theta) & \end{array}$$

Une notion liée à celle de substitution est la notion de contexte, qui peut être vu comme un terme paramétré.

Définition 4 (Contexte)

On se donne une variable spéciale $\Box \notin \mathcal{X}$.

Un terme $C\Box$ est un contexte *relativement à un ensemble de termes $\mathcal{T}$ si*

- *la variable $\Box$ apparaît une et une seule fois dans $C\Box$;*
- *la variable $\Box$ est libre dans $C\Box$.*

Pour tout terme t de $\mathcal{T}$, on note $C[t]$ le terme $C\Box[\Box := t]$.

Exemple 2 (Contexte algébrique)

Lorsque $\mathcal{T}$ est une algèbre de termes on obtient :

1. *Le terme $\Box$ est un contexte algébrique.*
2. *Si t_i est un contexte algébrique et si les termes t_j sont algébriques (pour tout $j \neq i$), alors $f_n(t_1, \ldots, t_n)$ est un contexte algébrique.*

Par exemple $C\Box \triangleq f(a, g(\Box, b))$ est un contexte algébrique et pour tout terme t on a $C[t] \equiv f(a, g(t, b))$.

Nous définirons également diverses relations binaires sur les termes. Les différentes égalités seront données de la façon suivante.

Définition 5 (Théorie équationnelle)

Une théorie équationnelle *sur un ensemble de termes $\mathcal{T}$ est la donnée d'un ensemble $\mathbb{T}$ d'axiomes de la forme $t =_{\mathbb{T}} u$.*

L'égalité $=_{\mathbb{T}}$ est alors la plus petite congruence, c.-à-d. la relation réflexive, symétrique, transitive et close par contexte (si $t =_{\mathbb{T}} u$ alors $C[t] =_{\mathbb{T}} C[u]$ pour tout $C\Box$) qui vérifie les axiomes de $\mathbb{T}$.

Une équivalence particulière est l'équivalence syntaxique.

Définition 6 (Équivalence syntaxique)

*L'*équivalence syntaxique $\equiv$ *est l'égalité obtenue en prenant $\mathbb{T} = \emptyset$.*

Par exemple, sur une algèbre de termes $\mathcal{T}$:

– $x \equiv y$ *si et seulement si* x *et* y *sont la même variable de* $\mathcal{X}$;
– $f(t_1, \ldots, t_n) \equiv g(u_1, \ldots, u_m)$ *si et seulement si* $f = g$ *(d'où* $n = m$*) et si* $\forall i \leq n,\ t_i \equiv u_i$.
C'est la relation d'équivalence la plus fine possible sur $\mathcal{T}$.

Remarque 1 (Équivalence définitionnelle)
On écrira parfois $e \triangleq t$*, ce qui signifie qu'on* définit *une expression* e *comme le terme* t.

On définit également la notion de sous-terme, qui est un ordre sur les termes.

Définition 7 (Sous-terme)
Le seul sous-terme d'une variable x *est* x *elle-même.*
Les sous-termes de $f_n(t_1, \ldots, t_n)$ *sont :*
– $f_n(t_1, \ldots, t_n)$ *lui-même ;*
– *chacun des sous-termes de* t_i*, pour tout* $1 \leq i \leq n$.
Un terme u *est un* sous-terme propre *de* t *si et seulement si* u *est un sous-terme de* t *et* $u \not\equiv t$.

On définit d'autres relations binaires non symétriques, avec en général l'intention de les considérer comme des méthodes de calcul, c'est pourquoi on choisira la notation $\rightarrow$ plutôt que $>$ qui est à réserver pour un ordre.

Définition 8 (Relations binaires dérivées)
Étant donnés un ensemble de termes $\mathcal{T}$ *et une relation binaire* $\rightarrow \subseteq \mathcal{T} \times \mathcal{T}$*, on définira les relations suivantes (inductivement pour les clôtures transitives) :*
– *la relation* $\mapsto$ *est la clôture contextuelle de* $\rightarrow$*, c.-à-d.* $t \mapsto t'$ *si et seulement si il existe un contexte* $C\Box$ *et des termes* s *et* s' *tels que* $t \equiv C[s]$ *et* $t' \equiv C[s']$ *et* $s \rightarrow s'$;
– *la relation* $\twoheadrightarrow$ *est la clôture réflexive transitive de* $\rightarrow$*, c.-à-d.* $t \twoheadrightarrow t'$ *si et seulement si* $\begin{cases} t \equiv t' \\ ou \\ \exists t'',\ t \rightarrow t'' \wedge t'' \twoheadrightarrow t' \end{cases}$;
– *la relation* $\mapsto\!\!\!\!\twoheadrightarrow$ *est la clôture réflexive transitive de* $\mapsto$;
– *la relation* $=$ *est la clôture réflexive transitive symétrique de* $\mapsto$ *(souvent appelée conversion), c.-à-d.* $t = t'$ *si et seulement si* $t \equiv t'$ *ou* $\exists t'',\ t = t'' \wedge \begin{cases} t'' \mapsto\!\!\!\!\twoheadrightarrow t' \\ ou \\ t' \mapsto\!\!\!\!\twoheadrightarrow t'' \end{cases}$

Définissons deux propriétés abstraites essentielles de ces relations.

Définition 9 (Confluence, propriété de Church-Rosser)

1. *Une relation* $\rightarrow$ *est dite* confluente *sur un ensemble de termes* $\mathcal{T}$ *si et seulement si*

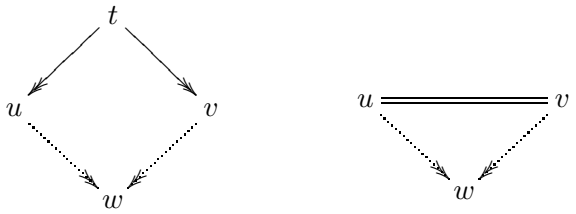
$$\forall t, u, v,\ (t \twoheadrightarrow u \text{ et } t \twoheadrightarrow v) \ \Rightarrow\ \exists w,\ (u \twoheadrightarrow w \text{ et } v \twoheadrightarrow w)$$

2. *Une relation* $\rightarrow$ *vérifie la* propriété de Church-Rosser *si et seulement si :*

$$\forall u, v,\ u = v \ \Rightarrow\ \exists w,\ (u \twoheadrightarrow w \text{ et } v \twoheadrightarrow w)$$

On peut démontrer que la confluence est équivalente à la propriété de Church-Rosser.

Pour fixer les idées, on peut donner une représentation graphique de ces deux propriétés : un trait plein dénote une hypothèse et des pointillés dénotent une conclusion.



confluence *Church-Rosser*

Définition 10 (Terminaison)
Une relation $\rightarrow$ *est* terminante *sur un ensemble de termes* $\mathcal{T}$ *si pour tout* $t \in \mathcal{T}$*, toute chaîne de réductions* $t \rightarrow t_1 \rightarrow t_2 \ldots$ *est finie.*

Remarque 2
Cette définition équivaut à dire que $\rightarrow$ *est un* préordre bien fondé *sur* $\mathcal{T}$*.*

Enfin, lorsqu'on veut démontrer des propriétés de façon systématique, il est souvent commode de décrire par un système formel les règles qui gouvernent ces démonstrations.

Définition 11 (Règle et système d'inférence)
Une règle d'inférence *est la donnée d'un ensemble éventuellement vide d'hypothèses (ou prémisses)* $H_1, \ldots, H_n$ *; d'une conclusion* C *; d'une condition d'application* ϕ *; éventuellement d'une étiquette* ℓ*. On note une telle règle*

$$(\ell)\ \frac{H_1 \qquad \ldots \qquad H_n}{C} \quad (\phi)$$

La condition d'application rassemble des prémisses dont la validité n'est pas décidée par le système d'inférence.

Un système d'inférence *est un ensemble de règles d'inférences. Le plus souvent, on donnera des schémas de règles, autrement dit des règles valables pour toute instanciation de leurs paramètres. Une conclusion* C *est* dérivable *dans un système donné si :*

- *il existe une règle n'ayant aucune hypothèse, dont la condition d'application est vérifiée, et dont la conclusion est* C *;*
- *ou il existe une règle dont la conclusion est* C*, dont la condition d'application est vérifiée et dont toutes les hypothèses sont dérivables.*

Un enchaînement de règles permettant de dériver une conclusion C *est appelé* arbre de dérivation *pour cette conclusion.*

1.1 Démonstrations et programmes

Rappelons les notions fondamentales des λ-calculs typés et des systèmes de déduction naturelle, et voyons en quoi ils sont intrinsèquement liés par l'isomorphisme de Curry-Howard-de Bruijn. La discussion de cette section est largement inspirée par les notes de M. H. B. Sørensen et P. Urzyczyn [SU98].

1.1.1 Lambda-calculs typés

Le λ-calcul, introduit par A. Church [Chu33], constitue un fondement de la théorie des fonctions, puisqu'il permet de représenter explicitement l'abstraction d'une variable et l'application d'un terme à un autre. Il s'est rapidement avéré que ce formalisme très simple était également très puissant, puisqu'il permet de représenter toutes les fonctions calculables au sens de Turing. D'un point de vue sémantique, il est apparu que cette généralité pose problème, puisque tout terme peut être appliqué à tout autre, y compris lui-même, sans considération de domaine sur lequel une fonction pouvait opérer.

Des systèmes de types ont donc été définis pour le λ-calcul, avec comme idée de base qu'une fonction a pour type $A \to B$ si, appliquée à un argument de type A, elle renvoie toujours une valeur de type B. Plusieurs extensions visant à rendre plus flexibles ces schémas de types ont été proposées, puis unifiées dans un unique formalisme, le Calcul des Constructions, par Th. Coquand et G. Huet. Par la suite et d'après les travaux indépendants de S. Berardi et J. Terlouw, la structure composite de ce calcul et les généralisations possibles ont été mises en évidence par H. Barendregt. C'est sa présentation, tirée de [Bar92], que nous reprenons ici.

Définition 12 (Syntaxe et sémantique opérationnelle du λ-calcul)
Les pseudo-termes du λ-calcul typé sont définis par la grammaire

$$\mathcal{T}_\lambda \ ::= \ \mathcal{S} \mid \mathcal{X} \mid \mathcal{T}_\lambda\,\mathcal{T}_\lambda \mid \lambda\mathcal{X}{:}\mathcal{T}_\lambda.\mathcal{T}_\lambda \mid \Pi\mathcal{X}{:}\mathcal{T}_\lambda.\mathcal{T}_\lambda$$

où $\mathcal{X}$ est un ensemble infini de variables et $\mathcal{S}$ un ensemble de sortes. On utilisera les lettres A, B, C pour dénoter des pseudo-termes arbitraires, et s, s_1, s_2 pour des sortes.

Les variables libres $\mathcal{FV}$ d'un pseudo-terme sont définies par

$$\begin{aligned}
\mathcal{FV}(x) &= \{x\} \\
\mathcal{FV}(AB) &= \mathcal{FV}(A) \cup \mathcal{FV}(B) \\
\mathcal{FV}(\lambda x{:}A.B) &= (\mathcal{FV}(A) \cup \mathcal{FV}(B)) \setminus \{x\} \\
\mathcal{FV}(\Pi x{:}A.B) &= (\mathcal{FV}(A) \cup \mathcal{FV}(B)) \setminus \{x\}
\end{aligned}$$

et ses variables liées sont définies de façon complémentaire

$$\begin{aligned}
\mathcal{BV}(x) &= \emptyset \\
\mathcal{BV}(AB) &= \mathcal{BV}(A) \cup \mathcal{BV}(B) \\
\mathcal{BV}(\lambda x{:}A.B) &= \mathcal{BV}(A) \cup \mathcal{BV}(B) \cup \{x\} \\
\mathcal{BV}(\Pi x{:}A.B) &= \mathcal{BV}(A) \cup \mathcal{BV}(B) \cup \{x\}
\end{aligned}$$

Sur cet ensemble de termes on définit la relation de β-réduction.

$$(\lambda x{:}A.B)C \ \to_\beta \ B[x := C]$$

On notera $\mapsto_\beta$, $\twoheadrightarrow_\beta$ et $=_\beta$ les relations dérivées de $\to_\beta$.

Exemple 3 *Appelons ω le terme $\lambda x.xx$. Alors*

$$\omega\omega \equiv (\lambda x.xx)\omega \mapsto_\beta \omega\omega \ldots$$

Ce terme admet une chaîne infinie (stationnaire) de β-réductions.

Remarque 3 (Notations)

Pour faciliter la lecture des λ-termes, nous suivrons plusieurs conventions :

1. *L'application est prioritaire sur les abstractions. Par exemple $\lambda x.xy$ se lit $\lambda x.(xy)$*
2. *L'associativité par défaut est à droite pour les abstractions, et à gauche pour les applications. Par exemple $\lambda x.\lambda y.xyz$ se lit $\lambda x.(\lambda y.(xy)z)$*
3. *Nous utiliserons parfois la notation $\overline{t}_{(1..n)}$ pour désigner un «vecteur» de termes $t_1, \ldots, t_n$, qui sera utilisée en combinaison avec les conventions de notations pour les abstractions et les applications. Par exemple $\lambda\overline{x}_{(1..n)}.y\overline{x}_{(1..n)}$ se lit $\lambda x_1 \ldots \lambda x_n.yx_1 \ldots x_n$.*

Il est rare que l'on compare deux λ-termes de façon purement syntaxique. En effet, les substitutions n'opérant que sur des variables libres, les noms utilisés pour les variables liées ne jouent aucun rôle. On définira donc l'α-conversion comme la plus fine relation de congruence vérifiant les équations suivantes :

$$\frac{A =_\alpha A' \qquad B[x := y] =_\alpha B'}{\lambda x{:}A.B =_\alpha \lambda y{:}A'.B'} \text{ (POUR } y \notin B) \qquad \frac{A =_\alpha A' \qquad B[x := y] =_\alpha B'}{\Pi x{:}A.B =_\alpha \Pi y{:}A'.B'} \text{ (POUR } y \notin B)$$

Étant donné un λ-terme A, on pourra donc toujours considérer que $\mathcal{FV}(A)$ et $\mathcal{BV}(A)$ sont des ensembles disjoints (condition d'hygiène de Barendregt), quitte à considérer un autre λ-terme α-équivalent.

De cet ensemble de termes, on ne veut cependant pas tout conserver. D'une part, on éliminera certains termes, par exemple ceux de la forme $(\Pi x{:}A.B)C$ parce que dans les λ-calculs standard il ne représentent ni une fonction ni un type ; d'autre part on interdira certains termes, par exemple xx, parce qu'ils ne respectent pas les considérations de domaine évoquées plus haut. Les termes légaux sont caractérisés par un système de types.

Définition 13 (Systèmes de Types Purs (ou Pure Type Systems, PTS))

1. *Un* contexte Γ *est une liste finie ordonnée, éventuellement vide, de déclarations de types de la forme $x{:}A$ où $x \in \mathcal{X}$ et $A \in \mathcal{T}_\lambda$. Si $x{:}A \in \Gamma$ on notera aussi $\Gamma(x) = A$.*
   La notion de variable libre s'étend aux contextes par $\mathcal{FV}(\Gamma, x{:}A) = \mathcal{FV}(\Gamma) \cup \mathcal{FV}(A)$.
2. *Un* jugement de typage *est un triplet $\Gamma \vdash A : B$ signifiant que A admet pour type B sous les hypothèse de typage déclarées dans Γ.*
3. *Un* système de types purs *est un système d'inférence pour des jugements de typage, dont les règles sont données dans la figure 1.1. Ces règles sont paramétrées par trois ensembles $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ qui sont respectivement un ensemble de sortes, de paires de sortes et de triplets de sortes.*
4. *Un pseudo-terme A est dit* légal *s'il existe un contexte Γ et un pseudo-terme B tels que l'un des jugements $\Gamma \vdash A : B$ ou $\Gamma \vdash B : A$ est dérivable.*

Dans un tel système, il faut comprendre les sortes de $\mathcal{S}$ comme différents univers dans lesquels sont classés les termes légaux. Les règles de typage se comprennent alors ainsi :

(AXIOM) et l'ensemble $\mathcal{A}$ décrivent la hiérarchie entre les sortes ; on notera que c'est la seule règle sans hypothèse, et donc celle par laquelle toutes les dérivations de types doivent se terminer ;

(START) exprime que toute variable apparaissant dans un terme doit avoir été déclarée dans le contexte, et vérifie la bonne formation du type attribué à cette variable ;

$$(\text{AXIOM})\ \frac{}{\vdash s_1 : s_2}\ ((s_1,s_2) \in \mathcal{A})$$

$$(\text{START})\ \frac{\Gamma \vdash A : s}{\Gamma, x{:}A \vdash x : A}\ \left(\begin{array}{c} x \notin \Gamma \\ s \in \mathcal{S} \end{array}\right)$$

$$(\text{WEAK})\ \frac{\Gamma \vdash A : B \qquad \Gamma \vdash C : s}{\Gamma, x{:}C \vdash A : B}\ \left(\begin{array}{c} x \notin \Gamma \\ s \in \mathcal{S} \end{array}\right)$$

$$(\text{PROD})\ \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x{:}A \vdash B : s_2}{\Gamma \vdash \Pi x{:}A.B : s_3}\ ((s_1,s_2,s_3) \in \mathcal{R})$$

$$(\text{APPL})\ \frac{\Gamma \vdash F : \Pi x{:}A.B \qquad \Gamma \vdash a : A}{\Gamma \vdash Fa : B[x := a]}$$

$$(\text{ABS})\ \frac{\Gamma, x{:}A \vdash b : B \qquad \Gamma \vdash \Pi x{:}A.B : s}{\Gamma \vdash \lambda x{:}A.b : \Pi x{:}A.B}\ (s \in \mathcal{S})$$

$$(\text{CONV})\ \frac{\Gamma \vdash A : C \qquad \Gamma \vdash B : s}{\Gamma \vdash A : B}\ \left(\begin{array}{c} B =_\beta C \\ s \in \mathcal{S} \end{array}\right)$$

FIG. 1.1 – Règles d'inférence d'un système de types purs

(WEAK) assure la correction du contexte, en vérifiant qu'une même variable n'y est déclarée qu'une seule fois et que son type est bien formé ;

(PROD) gouverne la formation des Π-abstractions, que l'on peut en première approximation assimiler à des types, en fonction des règles autorisées dans l'ensemble $\mathcal{R}$;

(APPL) permet de typer une application, en vérifiant que l'argument fourni a est bien du type attendu A et propage la valeur de cet argument dans le type du résultat ;

(ABS) permet de typer une abstraction et déclenche la vérification de conformité de son type relativement aux règles de produit autorisées ;

(CONV) identifie tous les types bien formés β-convertibles entre eux.

Les cas les plus connus et les plus utilisés de ces systèmes de types purs sont certainement ceux du λ-cube de Barendregt, qui correspondent pour la plupart à des systèmes étudiés indépendamment sous une formulation différente mais équivalente.

Exemple 4 (Les systèmes du λ-cube)

On se place dans le cas où $\mathcal{S} = \{*, \Box\}$ et où le seul axiome est $\mathcal{A} = \{(*, \Box)\}$. De plus on imposera que pour tout triplet (s_1, s_2, s_3) de $\mathcal{R}$ on ait $s_2 = s_3$, et on se contentera donc d'écrire $(s_1, s_2) \in \mathcal{R}$. Enfin, on considérera que la règle $(*, *)$ est toujours présente dans $\mathcal{R}$.

Selon lesquelles des trois autres règles possibles $(*, \Box)$, $(\Box, *)$ et $(\Box, \Box)$ apparaissent dans $\mathcal{R}$, on peut ainsi définir une collection de huit systèmes de types, que l'on représente traditionnellement sur les sommets d'un cube, les faces décrivant quelles règles ont été retenues.

Le Calcul des Constructions, qui correspond à λC, est obtenu en prenant les quatre règles dans $\mathcal{R}$, et on comprend ainsi mieux sa structure.

– Le système $\lambda_\rightarrow$ correspond au λ-calcul simplement typé [Chu40], avec $\mathcal{R} = \{(*, *)\}$.

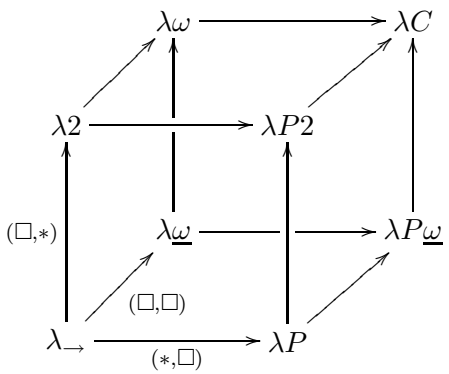


FIG. 1.2 – Le λ-cube de Barendregt

- *La règle $(\Box,*)$ permet de construire des termes qui dépendent de types. On obtient ainsi la notion de type polymorphe, et le système $\lambda 2$ correspond au système* F *de J.-Y. Girard [Gir72, GLT89].*
- *La règle $(\Box,\Box)$ permet de construire des types qui dépendent de types. Le système $\lambda\omega$, qui inclut également la règle $(\Box,*)$, correspond au système d'ordre supérieur* Fω *de Girard [Gir72, GLT89].*
- *La règle $(*,\Box)$ permet de construire des types qui dépendent de termes [Bru70, HHP93]. Nous verrons dans la prochaine section le sens qu'on peut lui donner en logique.*

Remarque 4 (Le λ-calcul simplement typé)

On présente habituellement le λ-calcul simplement typé sous une forme plus directe : les types (qu'on notera τ,σ) sont de la forme

$$\mathcal{T}^* \ ::= \ \mathcal{K}^* \mid \mathcal{T}^* \rightarrow \mathcal{T}^*$$

où $\mathcal{K}^*$ est un ensemble de types atomiques (on notera ι ou κ un tel type) et les termes sont donnés par

$$\mathcal{T}_{\lambda\rightarrow} \ ::= \ \mathcal{X} \mid \lambda\mathcal{X}{:}\mathcal{T}^*.\mathcal{T}_{\lambda\rightarrow} \mid \mathcal{T}_{\lambda\rightarrow}\,\mathcal{T}_{\lambda\rightarrow}$$

Un contexte Γ est alors un ensemble de déclarations de types $x{:}\tau$ où une même variable n'apparaît pas plus d'une fois, et les règles de typage sont celles de la figure 1.3.

$$(\text{VAR})\ \frac{}{\Gamma, x{:}\tau \vdash x : \tau} \qquad (\text{ABS})\ \frac{\Gamma, x : \tau \vdash b : \sigma}{\Gamma \vdash \lambda x{:}\tau.b : \tau \rightarrow \sigma} \qquad (\text{APPL})\ \frac{\Gamma \vdash F : \tau \rightarrow \sigma \qquad \Gamma \vdash a : \tau}{\Gamma \vdash Fa : \sigma}$$

FIG. 1.3 – Le λ-calcul simplement typé

On peut montrer que ce système est équivalent au système $\lambda\!\rightarrow$ du λ-cube, avec la traduction

ci-dessous. Pour toute variable x telle que $\Gamma(x) = *$, on se donne un type atomique ι_x.

$$\begin{array}{rcll} |x| & = & x \in \mathcal{X} & \textit{si } \Gamma \vdash x : A : * \\ |x| & = & \iota_x \in \mathcal{K}^* & \textit{si } \Gamma \vdash x : * \\ |AB| & = & |A|\,|B| & \\ |\lambda x{:}A.B| & = & \lambda x{:}|A|.|B| & \\ |\Pi x{:}A.B| & = & |A| \to |B| & \\ |\Gamma, x{:}*| & = & |\Gamma| & \\ |\Gamma, x{:}A| & = & |\Gamma|, x{:}|A| & \end{array}$$

qui est réversible à condition de bien choisir la variable x lorsqu'on produit une Π-abstraction à partir d'un type $\tau \to \sigma$.

Nous utiliserons indifféremment l'une ou l'autre présentation, dans l'idée de rendre l'exposé plus compréhensible.

Rappelons enfin quelques propriétés de ces λ-calculs typés. La confluence n'est pas à proprement parler une propriété du système de types, mais comme elle est équivalente à la propriété de Church-Rosser, elle légitime la règle (CONV) en donnant une caractérisation pratique de la condition $B{=_\beta}C$.

Théorème 1 (Confluence [Bar84])
La β-réduction $\mapsto_\beta$ est confluente sur l'ensemble des pseudo-termes $\mathcal{T}_\lambda$.

Le théorème de correction valide les règles de formation des types comme (PROD).

Théorème 2 (Correction [GN91])
Dans tout PTS, si $\Gamma \vdash A : B$ alors $\exists s \in \mathcal{S}$, $(B \equiv s$ ou $\Gamma \vdash B : s)$.

On pourra alors classer les termes légaux selon la sorte affectée à leur type. Par exemple, dans les systèmes du λ-cube, on utilise parfois la nomenclature suivante :
- A est un *objet* s'il existe Γ et B tels que $\Gamma \vdash A : B$ et $\Gamma \vdash B : *$;
- A est un *type* s'il existe Γ tel que $\Gamma \vdash A : *$;
- A est un *constructeur* s'il existe Γ et B tels que $\Gamma \vdash A : B$ et $\Gamma \vdash B : \Box$;
- A est un *genre* (en anglais *kind*) s'il existe Γ tel que $\Gamma \vdash A : \Box$.

La propriété suivante donne tout son sens à un système de types en tant que mécanisme d'analyse statique. Elle exprime qu'un terme qui vérifie un jugement de typage continue à le vérifier après des réductions arbitraires.

Théorème 3 (Préservation du type [GN91])
Dans tout PTS, si $\Gamma \vdash A : B$ et $A \mapsto\!\!\!\!\to_\beta A'$ alors $\Gamma \vdash A' : B$.

Enfin, la normalisation forte n'est pas une propriété valide dans tous les PTS. Elle a été étudiée pour de nombreux systèmes en théorie des types, qui ne sont pas tous des instances de PTS ; nous nous limiterons ici au cas des systèmes du λ-cube.

Théorème 4 (Normalisation forte [Coq85, Gir72, GN91])
Dans tous les PTS du λ-cube, si $\Gamma \vdash A : B$, alors il n'existe pas de chaîne de β-réduction infinie partant de A ou de B. Autrement dit, la β-réduction est terminante sur l'ensemble des termes légaux.

À titre indicatif, donnons les grandes lignes de la démonstration.

1. On définit les *candidats de réductibilité* comme des ensembles de pseudo-termes fortement normalisants, vérifiant certaines propriétés comme la stabilité par réduction ou par une forme d'expansion (la relation symétrique de $\mapsto_\beta$).
2. On se donne une interprétation $[\![\ ]\!]$ qui à chaque type associe un candidat de réductibilité.
3. On démontre que pour tout terme A typable (c.-à-d. $\Gamma \vdash A : B$) et pour toute instanciation θ de ses variables libres, $A\theta$ appartient à $[\![B]\!]$.
4. On en conclut que tout terme typable est fortement normalisant (puisque $[\![B]\!] \subseteq SN$).

Exemple 5 *Le terme $\omega\omega$ de l'exemple 3 n'est typable dans aucun système du λ-cube. Par exemple, en λ-calcul simplement typé, une dérivation de type pour ω serait de la forme*

$$\dfrac{\dfrac{(\text{VAR})\ \dfrac{}{x : \tau \vdash x : \tau' \to \sigma} \qquad (\text{VAR})\ \dfrac{}{x : \tau \vdash x : \tau'}}{(\text{APPL})\ x : \tau \vdash xx : \sigma}}{(\text{ABS})\ \vdash \lambda x{:}\tau.xx : \tau \to \sigma}$$

mais alors la règle (VAR) *imposerait $\tau' \to \sigma \equiv \tau \equiv \tau'$ ce qui est impossible : aucun type τ' ne peut être un sous-terme de lui-même. Par conséquent, ω n'est pas simplement typable (et donc à plus forte raison $\omega\omega$ non plus).*

À tout terme légal A on peut donc associer une forme normale en effectuant autant de β-réductions que possible. De plus, la confluence assure que cette forme normale est unique, et on la notera $\mathsf{Nf}(A)$. On en déduit deux corollaires essentiels, valables dans chaque système du λ-cube.

Corollaire 5 (Décidabilité [Ben93])
Dans tout PTS fortement normalisant, les deux problèmes suivants sont décidables :

1. *Étant donnés un contexte Γ et un pseudo-terme A, existe-t-il un pseudo-terme B tel que $\Gamma \vdash A : B$ soit dérivable ?*
2. *Étant donnés un contexte Γ et deux pseudo-termes A et B, le jugement $\Gamma \vdash A : B$ est-il dérivable ?*

En effet le calcul d'un type admissible pour un pseudo-terme A peut s'effectuer en choisissant les règles d'inférence selon la structure de A. Les seules règles non guidées par la syntaxe sont (WEAK), qui peut être retardée jusqu'au typage des variables, et (CONV), dont l'applicabilité repose sur la condition $B{=_\beta}C$. Pour évaluer cette dernière, il suffit de calculer les formes normales de B et de C et de vérifier si elles sont α-convertibles.

Corollaire 6 (Consistance [GN91])
Dans tout PTS fortement normalisant étendant $\lambda 2$, il existe des pseudo-termes B tels que $\vdash B : *$ et tels qu'aucun pseudo-terme A ne vérifie $\vdash A : B$.

On parlera dans ce cas de types *non habités*. En effet on peut par exemple montrer que, dans tous les systèmes qui étendent $\lambda 2$, un terme A tel que $\vdash A : \Pi\alpha{:}*.*$ n'admettrait aucune forme normale, ce qui est contradictoire avec le théorème 4. On peut d'ailleurs montrer que l'habitabilité de ce dernier type est équivalente à l'inconsistance du système.

1.1.2 Déduction naturelle pour les logiques intuitionnistes

Nous allons décrire différentes logiques intuitionnistes, par le biais de systèmes de *déduction naturelle*, introduits par D. Prawitz [Pra65]. Procédons de façon incrémentale : nous commencerons par la logique propositionnelle minimale, puis nous verrons comment ajouter d'autres connecteurs et passer à la logique des prédicats ou aux logiques d'ordre supérieur.

Définition 14 (Logique propositionnelle minimale $\mathbf{PROP}_{min}$)

1. *Le langage de* formules *de $PROP_{min}$ est*

$$\Phi_{min} \ ::= \ \mathcal{VP} \mid \Phi_{min} \Rightarrow \Phi_{min}$$

   où $\mathcal{VP}$ est un ensemble infini de variables propositionnelles *qu'on notera p, q. Les formules seront notées ϕ, ψ, ϑ.*

2. *Un* contexte Γ *est un ensemble fini non ordonné, éventuellement vide, de formules.*

3. *Un* jugement *est une paire $\Gamma \vdash \phi$ signifiant que la proposition ϕ est démontrable sous les hypothèses énoncées dans Γ.*

4. *Les jugements dérivables sont donnés par les règles d'inférence de la figure 1.4. Un arbre de dérivation pour $\Gamma \vdash \phi$ est aussi appelé une* démonstration *de ce jugement.*

$$(Ax)\ \frac{}{\Gamma, \phi \vdash \phi} \qquad (\Rightarrow I)\ \frac{\Gamma, \psi \vdash \phi}{\Gamma \vdash \psi \Rightarrow \phi} \qquad (\Rightarrow E)\ \frac{\Gamma \vdash \psi \Rightarrow \phi \qquad \Gamma \vdash \psi}{\Gamma \vdash \phi}$$

FIG. 1.4 – Logique propositionnelle minimale

Exemple 6 *Un théorème de $PROP_{min}$ est $p \Rightarrow (p \Rightarrow q) \Rightarrow q$:*

$$\dfrac{\dfrac{\dfrac{(Ax)\ \dfrac{}{p, p \Rightarrow q, q \vdash q}}{(\Rightarrow I)\ \dfrac{}{p, p \Rightarrow q \vdash q \Rightarrow q}} \qquad \dfrac{(Ax)\ \dfrac{}{p, p \Rightarrow q \vdash p \Rightarrow q} \qquad (Ax)\ \dfrac{}{p \Rightarrow q, p \vdash p}}{(\Rightarrow E)\ \dfrac{}{p, p \Rightarrow q \vdash q}}}{(\Rightarrow I)\ \dfrac{}{p, p \Rightarrow q \vdash q}}}{(\Rightarrow I)\ \dfrac{}{p \vdash (p \Rightarrow q) \Rightarrow q}}}{\vdash p \Rightarrow (p \Rightarrow q) \Rightarrow q}$$

Ce système est très simple et les propositions qu'il permet de formuler et de démontrer ne sont intéressantes que pour une logique très formelle et abstraite. Cependant, il nous servira de système jouet pour exposer les concepts de base que nous manipulerons ensuite pour des systèmes plus complexes.

On peut d'ores et déjà remarquer qu'il existe une règle d'introduction $(\Rightarrow I)$ et une règle d'élimination $(\Rightarrow E)$ pour le connecteur logique $\Rightarrow$ que nous avons utilisé dans notre langage de formules. La règle (Ax) est celle qui permet d'utiliser directement les hypothèses énoncées dans Γ. Nous pouvons maintenant définir le concept de coupure.

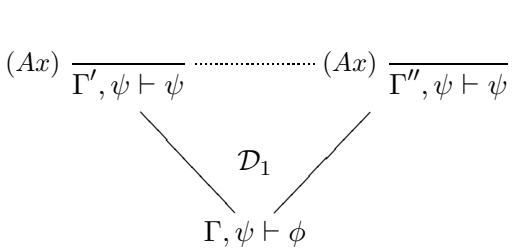
Définition 15 (Coupure)

Une coupure *dans une démonstration en déduction naturelle consiste en une règle d'introduction d'un connecteur immédiatement suivie d'une règle d'élimination pour le même connecteur et à la même position.*

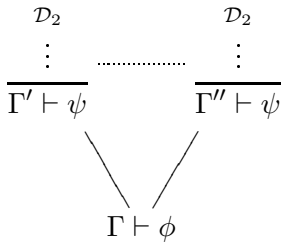
Ainsi, dans PROP_{min}, une coupure se présente sous la forme

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}_1\\ \vdots\\ \Gamma, \psi \vdash \phi\end{array}}{\Gamma \vdash \psi \Rightarrow \phi}\,(\Rightarrow I) \qquad \begin{array}{c}\mathcal{D}_2\\ \vdots\\ \Gamma \vdash \psi\end{array}}{\Gamma \vdash \phi}\,(\Rightarrow E)$$

qui correspond typiquement à introduire un lemme ψ, le prouver séparément par la dérivation $\mathcal{D}_2$, puis montrer ϕ sous l'hypothèse ψ dans la dérivation $\mathcal{D}_1$. Il est assez facile de voir que ce détour n'est pas nécessaire : en effet, pour démontrer $\Gamma \vdash \phi$, on peut reprendre $\mathcal{D}_1$ et la compléter par des copies de $\mathcal{D}_2$ partout où l'hypothèse ψ est utilisée. On obtient alors une démonstration de cette forme :



Dérivation $\mathcal{D}_1$ originale



Dérivation sans la coupure

Exemple 7 *Dans l'exemple 6, la sous-dérivation la plus à gauche comporte une coupure puisque $q \Rightarrow q$ est introduit puis éliminé aussitôt. Par le mécanisme ci-dessus on obtient la démonstration sans coupure*

$$\dfrac{\dfrac{\dfrac{(Ax)\ \overline{} \qquad (Ax)\ \overline{}}{p, p \Rightarrow q \vdash p \Rightarrow q}\,(\Rightarrow E)}{p, p \Rightarrow q \vdash q}\,(\Rightarrow I)}{p \vdash (p \Rightarrow q) \Rightarrow q}\,(\Rightarrow I)$$
$$\vdash p \Rightarrow (p \Rightarrow q) \Rightarrow q$$

Il n'est cependant pas évident que cette méthode suffise à éliminer toutes les coupures apparaissant dans une démonstration, ne serait-ce que parce que $\mathcal{D}_2$ peut elle-même contenir des coupures qui risquent d'être dupliquées au cours du procédé. Nous verrons dans la prochaine section comment l'élimination des coupures dans une démonstration peut s'étudier comme la β-réduction dans un λ-terme.

Introduisons maintenant les autres connecteurs usuels de la logique propositionnelle intuitionniste ainsi que les règles de déduction associées.

Définition 16 (Logique propositionnelle intuitionniste PROP)
Le langage de formules de PROP est donné par

$$\Phi \ ::= \ \mathcal{VP} \mid \bot \mid \Phi \Rightarrow \Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi$$

et les règles d'inférence du système sont celles des figures 1.4 et 1.5.

$$(\wedge I)\ \frac{\Gamma \vdash \phi \qquad \Gamma \vdash \psi}{\Gamma \vdash \phi \wedge \psi} \qquad (\wedge E_l)\ \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \phi} \qquad (\wedge E_r)\ \frac{\Gamma \vdash \phi \wedge \psi}{\Gamma \vdash \psi}$$

$$(\vee I_l)\ \frac{\Gamma \vdash \phi}{\Gamma \vdash \phi \vee \psi} \qquad (\vee I_r)\ \frac{\Gamma \vdash \psi}{\Gamma \vdash \phi \vee \psi} \qquad (\vee E)\ \frac{\Gamma \vdash \phi \vee \psi \qquad \Gamma, \phi \vdash \vartheta \qquad \Gamma, \psi \vdash \vartheta}{\Gamma \vdash \vartheta}$$

$$(\bot E)\ \frac{\Gamma \vdash \bot}{\Gamma \vdash \phi}$$

FIG. 1.5 – Les autres règles de la logique propositionnelle intuitionniste

On remarquera que la négation n'apparaît pas dans notre langage de formules ; en effet on écrira $\phi \Rightarrow \bot$ plutôt que $\neg\phi$, ce qui est strictement équivalent dans toutes les logiques que nous étudierons. On remarquera également qu'un même connecteur peut avoir plusieurs variantes de règles d'introduction ou d'élimination, ce qui traduit leur symétrie. Enfin $\bot$ n'a pas de règle d'introduction : une fois démontrée l'élimination des coupures, c'est ce qui nous permettra de vérifier la consistance de la logique. Il paraît d'ailleurs raisonnable de ne pas fournir de règle de déduction dont la conclusion est systématiquement l'absurde.

Ces logiques sont dites *intuitionnistes* par opposition aux logiques *classiques*, que l'on obtient en ajoutant par exemple l'axiome du tiers exclu $\phi \vee (\phi \Rightarrow \bot)$. C'est ce qui explique pourquoi nous avons pu nous passer de la négation mais pas de l'implication par exemple, puisque dans un cadre intuitionniste $\phi \Rightarrow \psi$ n'est pas équivalent à $\neg\phi \vee \psi$. Le principal intérêt est que les démonstrations produites sont constructives : ainsi, si l'on prouve une disjonction, il faut préciser lequel de ses facteurs est valide.

Exemple 8 *Dans PROP, on peut démontrer $p \Rightarrow \neg\neg p$ mais pas $\neg\neg p \Rightarrow p$. En effet, avec l'abréviation vue plus haut, on a*

$$\cfrac{\cfrac{\cfrac{(Ax)\ \overline{p, p \Rightarrow \bot \vdash p \Rightarrow \bot} \qquad (Ax)\ \overline{p, p \Rightarrow \bot \vdash p}}{(\Rightarrow E)\ p, p \Rightarrow \bot \vdash \bot}}{(\Rightarrow I)\ p \vdash (p \Rightarrow \bot) \Rightarrow \bot}}{(\Rightarrow I)\ \vdash p \Rightarrow ((p \Rightarrow \bot) \Rightarrow \bot)}$$

mais pour démontrer $\neg\neg p \Rightarrow p$ il faut utiliser par exemple $p \vee (p \Rightarrow \bot)$.

On peut ensuite passer à la logique du premier ordre, qui ne manipule plus seulement des propositions, mais aussi une algèbre de termes et des prédicats sur ces termes.

Définition 17 (Logique du premier ordre intuitionniste PRED)

On se donne un ensemble $\mathcal{X}$ de variables algébriques, une signature Σ qui associe à chaque symbole de fonction f une unique arité $n \in \mathbb{N}$, et enfin un ensemble R de symboles de prédicats eux aussi dotés d'une arité.

1. *Une* formule atomique *est une expression $r(t_1, \ldots, t_n)$ où $r \in R$ est un symbole de prédicat d'arité n et $t_1, \ldots, t_n$ sont des termes algébriques.*

2. *Le langage des* formules du premier ordre *de PRED est donnée par*

$$\Phi_P \ ::= \ FA \mid \bot \mid \Phi_P \Rightarrow \Phi_P \mid \Phi_P \wedge \Phi_P \mid \Phi_P \vee \Phi_P \mid \forall \mathcal{X}.\Phi_P \mid \exists \mathcal{X}.\Phi_P$$

où FA est l'ensemble des formules atomiques.
On définit une notion de variables libres sur les termes et les formules :

$$\begin{aligned}
\mathcal{FV}(x) &= \{x\} \\
\mathcal{FV}(f(t_1,\ldots,t_n)) &= \mathcal{FV}(r(t_1,\ldots,t_n)) \ = \ \mathcal{FV}(t_1) \cup \ldots \cup \mathcal{FV}(t_n) \\
\mathcal{FV}(\phi \Rightarrow \psi) &= \mathcal{FV}(\phi \wedge \psi) \ = \ \mathcal{FV}(\phi \vee \psi) \ = \ \mathcal{FV}(\phi) \cup \mathcal{FV}(\psi) \\
\mathcal{FV}(\forall x.\phi) &= \mathcal{FV}(\exists x.\phi) \ = \ \mathcal{FV}(\phi) \setminus \{x\} \\
\mathcal{FV}(\Gamma, \phi) &= \mathcal{FV}(\Gamma) \cup \mathcal{FV}(\phi)
\end{aligned}$$

Les règles d'inférence du système sont celles des figures 1.4, 1.5 et 1.6.

$$(\forall I)\ \frac{\Gamma \vdash \phi}{\Gamma \vdash \forall x.\phi}\ (x \notin \mathcal{FV}(\Gamma)) \qquad (\forall E)\ \frac{\Gamma \vdash \forall x.\phi}{\Gamma \vdash \phi[x := t]}$$

$$(\exists I)\ \frac{\Gamma \vdash \phi[x := t]}{\Gamma \vdash \exists x.\phi} \qquad (\exists E)\ \frac{\Gamma \vdash \exists x.\phi \qquad \Gamma, \phi \vdash \psi}{\Gamma \vdash \psi}\ (x \notin \mathcal{FV}(\Gamma,\psi))$$

FIG. 1.6 – Les règles spécifiques aux quantificateurs

Définition 18 *Pour un prédicat unaire P et si la signature est telle qu'il existe au moins un terme algébrique t, la proposition $\forall x.P(x) \Rightarrow \exists y.P(y)$ est démontrable :*

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\forall x.P(x) \vdash \forall x.P(x)}\,(Ax)}{\forall x.P(x) \vdash P(t)}\,(\forall E)}{\forall x.P(x) \vdash \exists x.P(x)}\,(\exists I)}{\vdash \forall x.P(x) \Rightarrow \exists x.P(x)}\,(\Rightarrow I)$$

L'idée de démonstration constructive prend ici tout son sens : pour démontrer une formule existentiellement quantifiée $\exists x.\phi$ il faut pouvoir fournir un témoin pour la valeur de la variable x. Ainsi, dans l'exemple précédent, il a fallu choisir un terme t, même s'il était totalement arbitraire et n'apparaît plus dans la proposition démontrée. On commence également à voir en quoi une démonstration peut s'apparenter à une fonction (et donc à un λ-terme) : être capable de produire un témoin pour la démonstration d'une proposition revient bien à connaître une fonction qui prend en paramètres les hypothèses de cette proposition et retourne le témoin attendu.

Enfin, on peut s'intéresser aux logiques d'ordre supérieur, qui sont des extensions de PROP et PRED dans lesquelles on s'autorise à quantifier (existentiellement et universellement) sur des propositions. On obtient ainsi dans un premier temps la logique propositionnelle de second ordre PROP2, dans laquelle on ne peut quantifier que sur des variables propositionnelles :

$$\Phi_2 \ ::= \ \mathcal{VP} \mid \bot \mid \Phi_2 \Rightarrow \Phi_2 \mid \Phi_2 \wedge \Phi_2 \mid \Phi_2 \vee \Phi_2 \mid \forall \mathcal{VP}.\Phi_2 \mid \exists \mathcal{VP}.\Phi_2$$

$$(\forall I)\ \frac{\Gamma \vdash \phi}{\Gamma \vdash \forall p.\phi}\ (p \notin \mathcal{FV}(\Gamma)) \qquad (\forall E)\ \frac{\Gamma \vdash \forall p.\phi}{\Gamma \vdash \phi[p := \vartheta]}$$

$$(\exists I)\ \frac{\Gamma \vdash \phi[p := \vartheta]}{\Gamma \vdash \exists p.\phi} \qquad (\exists E)\ \frac{\Gamma \vdash \exists p.\phi \qquad \Gamma, \phi \vdash \psi}{\Gamma \vdash \psi}\ (p \notin \mathcal{FV}(\Gamma,\psi))$$

FIG. 1.7 – Les règles spécifiques à la logique propositionnelle du second ordre

L'instanciation de ces variables propositionnelles liées se fait par des règles d'introduction et d'élimination très similaires à celles de PRED, données en figure 1.7

On peut ainsi démontrer des théorèmes comme $\forall p.(p \Rightarrow p)$. En logique propositionnelle, on pouvait déjà démontrer $\phi \Rightarrow \phi$ pour toute proposition ϕ mais on ne disposait pas d'un moyen d'exprimer la généralité d'une telle démonstration.

Il est alors naturel de vouloir définir des fonctions sur les propositions, par exemple une fonction AND d'arité 2 qui prend deux propositions et renvoie leur conjonction. On peut ensuite définir des fonctions sur les fonctions, quantifier sur les fonctions, et ainsi de suite. La clôture de toutes ces opérations amène à la logique propositionnelle d'ordre supérieur, dans laquelle on peut par exemple représenter fidèlement (et en conservant un point de vue intuitionniste) tous les connecteurs uniquement au moyen de $\Rightarrow$ et de $\forall$.

Les mêmes extensions peuvent être obtenues pour PRED, en quantifiant sur des symboles de prédicats et en définissant des fonctions sur les prédicats.

1.1.3 Isomorphisme de Curry-Howard-de Bruijn

Correspondance entre $\lambda_{\rightarrow}$ et PROP$_{min}$

Reprenons notre système de logique propositionnelle minimale PROP$_{min}$ et comparons-le avec les règles (VAR), (ABS) et (APPL) du λ-calcul simplement typé.

$$(\text{VAR})\ \frac{}{\Gamma, x{:}\tau \vdash x : \tau} \qquad (Ax)\ \frac{}{\Gamma, \phi \vdash \phi}$$

$$(\text{ABS})\ \frac{\Gamma, x : \tau \vdash b : \sigma}{\Gamma \vdash \lambda x{:}\tau.b : \tau \rightarrow \sigma} \qquad (\Rightarrow I)\ \frac{\Gamma, \psi \vdash \phi}{\Gamma \vdash \psi \Rightarrow \phi}$$

$$(\text{APPL})\ \frac{\Gamma \vdash F : \tau \rightarrow \sigma \qquad \Gamma \vdash a : \tau}{\Gamma \vdash Fa : \sigma} \qquad (\Rightarrow E)\ \frac{\Gamma \vdash \psi \Rightarrow \phi \qquad \Gamma \vdash \psi}{\Gamma \vdash \phi}$$

FIG. 1.8 – Comparaison entre déduction et typage

Il apparaît assez clairement une correspondance entre les types que l'on produit et les formules que l'on démontre, à condition d'assimiler un type flèche à une implication. Cette correspondance est même plus profonde, puisqu'elle permet également d'identifier un λ-terme à une démonstration, une introduction de $\Rightarrow$ correspondant à une abstraction et une élimination correspondant à une application.

Théorème 7 (Isomorphisme de Curry-Howard-de Bruijn [CF58, How80])

Étant donnée la traduction | | des types et contextes en propositions et contextes :

$$\begin{aligned} |\iota| &= p \\ |\tau \to \sigma| &= |\tau| \Rightarrow |\sigma| \\ |\emptyset| &= \emptyset \\ |\Gamma, x{:}\tau| &= |\Gamma|, |\tau| \end{aligned}$$

Il existe une bijection de l'ensemble des λ-termes simplement typés vers l'ensemble des démonstrations de $PROP_{min}$, telle que

1. *si $\Gamma \vdash A : \tau$ alors l'image de A est une démonstration de $|\Gamma| \vdash |\tau|$ dans $PROP_{min}$;*
2. *toute démonstration de $\Gamma \vdash \phi$ dans $PROP_{min}$ admet pour image réciproque un terme A tel que $\Delta \vdash A : |\phi|^{-1}$ avec $\mathcal{FV}(A) \subseteq \mathcal{D}om(\Delta)$ et $|\Delta| = \Gamma$.*

On dira alors que A est un terme de preuve *pour la proposition ϕ ou pour sa démonstration.*

Exemple 9 *La démonstration de $p \Rightarrow (p \Rightarrow q) \Rightarrow q$ donnée à l'exemple 6 est isomorphe au λ-terme $\lambda x{:}p.\lambda y{:}(p \to q).((\lambda z{:}q.z)(yx))$.*

Les λ-termes fournissent donc une façon pratique de représenter des démonstrations, et on voit en quoi un raisonnement intuitionniste est comparable à une fonction : prouver une implication $\phi \Rightarrow \psi$ revient à transformer une démonstration de ϕ en une démonstration de ψ.

La β-réduction détermine la sémantique opérationnelle du λ-calcul, et à ce titre on pourrait supposer qu'elle n'a pas réellement de sens au niveau des démonstrations. Il est donc remarquable de constater qu'un radical $(\lambda x.A)B$ traduit exactement une coupure, et qu'un pas de β-réduction correspond à l'élimination de cette coupure telle que présentée à la section précédente.

Exemple 10 *Le λ-terme donné à l'exemple 9, et qui correspond à une démonstration avec coupure, contient effectivement le radical $(\lambda z.z)(xy)$. Sa forme normale est $\lambda x{:}p.\lambda y{:}(p \to q).(yx)$, qui correspond bien à la démonstration sans coupure de l'exemple 7.*

Ainsi, les propriétés connues sur les λ-termes ont des conséquences immédiates sur les démonstrations :

- un λ-terme en forme normale représente une démonstration sans coupure ;
- la préservation du type implique que l'élimination d'une coupure dans une démonstration ne modifie pas la proposition démontrée ;
- la normalisation forte implique que toute démonstration admet une unique version sans coupure qui peut s'obtenir par itération finie du processus d'élimination des coupures ;
- la décidabilité du typage implique que la vérification des démonstrations est décidable.

On commence à voir en quoi cet isomorphisme nous permet de démontrer la cohérence des logiques que nous étudions. En effet, s'il existe une démonstration de $\vdash \bot$, elle admet une forme normale. Celle-ci ne peut alors se terminer ni par la règle $(\Rightarrow I)$ puisque $\bot$ n'est pas une implication, ni par plusieurs applications de la règle $(\Rightarrow E)$ puisque celles-ci devraient être précédées d'une règle d'introduction, formant ainsi une coupure. Le même genre de raisonnement peut être mené pour PROP, PRED et les logiques d'ordre supérieur.

On peut ensuite traiter les autres connecteurs de la logique propositionnelle en étendant la syntaxe du λ-calcul avec des constructions simples, et la β-réduction avec le comportement

attendu pour l'élimination des coupures. La conjonction induit ainsi des paires ordonnées et des projections, alors que la disjonction induit un mécanisme de branchement :

$$\begin{array}{rcl} \mathcal{T}_\lambda & ::= & \mathcal{X} \mid \lambda\mathcal{X}.\mathcal{T}_\lambda \mid \mathcal{T}_\lambda\mathcal{T}_\lambda \mid (\mathcal{T}_\lambda, \mathcal{T}_\lambda) \mid \pi_1(\mathcal{T}_\lambda) \mid \pi_2(\mathcal{T}_\lambda) \\ & & \mid \mathsf{in}_l(\mathcal{T}_\lambda) \mid \mathsf{in}_r(\mathcal{T}_\lambda) \mid \mathsf{case}(\mathcal{T}_\lambda, \mathcal{X}.\mathcal{T}_\lambda, \mathcal{X}.\mathcal{T}_\lambda) \mid \mathit{botelim}(\mathcal{T}_\lambda) \end{array}$$

avec les règles de réduction additionnelles

$$\begin{array}{rcl} \pi_1(A_1, A_2) & \to_\beta & A_1 \\ \pi_2(A_1, A_2) & \to_\beta & A_2 \\ \mathsf{case}(\mathsf{in}_l(A), x.B_1, y.B_2) & \to_\beta & B_1[x := A] \\ \mathsf{case}(\mathsf{in}_r(A), x.B_1, y.B_2) & \to_\beta & B_2[y := A] \end{array}$$

Les règles de typage, données en figure 1.9, reflètent de très près les règles associées aux connecteurs. Comme pour l'implication, on peut constater que chacune des nouvelles règles de β-réduction traduit l'élimination de la coupure correspondante.

$$(\textsc{pair})\ \frac{\Gamma \vdash A_1 : \phi \qquad \Gamma \vdash A_2 : \psi}{\Gamma \vdash (A_1, A_2) : \phi \wedge \psi}$$

$$(\textsc{proj}_1)\ \frac{\Gamma \vdash A : \phi \wedge \psi}{\Gamma \vdash \pi_1(A) : \phi} \qquad (\textsc{proj}_2)\ \frac{\Gamma \vdash A : \phi \wedge \psi}{\Gamma \vdash \pi_2(A) : \psi}$$

$$(\textsc{case}_1)\ \frac{\Gamma \vdash A : \phi}{\Gamma \vdash \mathsf{in}_l(A) : \phi \vee \psi} \qquad (\textsc{case}_2)\ \frac{\Gamma \vdash A : \psi}{\Gamma \vdash \mathsf{in}_r(A) : \phi \vee \psi}$$

$$(\textsc{switch})\ \frac{\Gamma \vdash A : \phi \vee \psi \qquad \Gamma, x{:}\phi \vdash B_1 : \vartheta \qquad \Gamma, y{:}\psi \vdash B_2 : \vartheta}{\Gamma \vdash \mathsf{case}(A, x.B_1, y.B_2) : \vartheta}$$

$$(\textsc{absurd})\ \frac{\Gamma \vdash A : \bot}{\Gamma \vdash \mathit{botelim}(A) : \phi}$$

FIG. 1.9 – Règles de typage pour les autres connecteurs de la logique propositionnelle

Montrons ensuite comment les autres systèmes du λ-cube correspondent aux diverses extensions de PROP_{min}.

Correspondance entre λP et PRED

Le système λP permet de construire des types dépendant de termes, et a été souvent utilisé depuis le projet AUTOMATH [Bru80] pour construire et vérifier des démonstrations formelles. En effet, si l'on s'autorise à utiliser les λ-termes pour représenter non seulement les démonstrations mais aussi les termes algébriques, on obtient ainsi des propositions dépendant de termes algébriques, autrement dit des prédicats. Ainsi, un symbole de relation n-aire P aura un type de la forme $\Pi x_1{:}\iota \ldots \Pi x_n{:}\iota.*$ où ι est une constante de type réservée pour les termes algébriques. On vérifie aisément que

$$\iota{:}* \vdash \Pi x_1{:}\iota \ldots \Pi x_n{:}\iota.* : \Box$$

c.-à-d. que P est un constructeur de types, en utilisant n fois la règle de produit $(*, \Box)$. Alors pour tous termes algébriques $t_1, \ldots, t_n$ on aura $\vdash Pt_1 \ldots t_n : *$ ce qui exprime bien que c'est un type, autrement dit une proposition.

On voit ici que des termes appartenant au niveau objet peuvent apparaître dans un type, ce qui justifie la notation des Π-abstractions employée dans les PTS, puisque le nom d'une variable abstraite peut être significatif même dans le type (contrairement au cas simplement typé). Ce même mécanisme nous permet également d'exprimer la quantification universelle : si on a

$$\Gamma, x{:}\iota \vdash A : \phi : *$$

autrement dit si ϕ est une proposition dans laquelle apparaît (éventuellement) la variable algébrique x et si A est une démonstration de ϕ, alors

$$\Gamma \vdash \lambda x{:}\iota.A : \Pi x{:}\iota.\phi$$

ce qui exprime que $\lambda x.A$ est une démonstration de $\forall x.\phi$. L'élimination de $\forall$ se fait par application d'un tel terme à un terme algébrique. Remarquons que nous sommes loin d'avoir utilisé tout ce que λP permet d'exprimer : il est par exemple possible de définir plusieurs types pour les termes algébriques (on parlera de logique multisortée), ou encore quantifier sur des fonctions. C'est cette grande expressivité dans un langage pourtant assez simple qui explique que λP soit à la base du Logical Framework [HHP93].

Correspondance entre $\lambda 2$, $\lambda\omega$, λC et les logiques d'ordre supérieur

Le langage $\lambda 2$ est symptomatique de l'isomorphisme de Curry-Howard-de Bruijn, au sens où il a été introduit simultanément par J.-Y. Girard comme langage de termes de preuve [Gir72] et par J. C. Reynolds comme langage de programmation polymorphe [Rey74]. En effet, dans $\lambda 2$, on peut définir des termes dépendant de types, autrement dit on peut quantifier sur des variables propositionnelles. On obtient donc naturellement un langage de types qui correspond directement au langage de formules de PROP2. Ainsi, le type $\Pi\alpha{:}*.\Pi x{:}\alpha.\alpha$ représente la proposition $\forall\alpha(\alpha \Rightarrow \alpha)$. Le λ-terme $\lambda\alpha{:}*.\lambda x{:}\alpha.x$ (l'identité polymorphe) représente une démonstration de cette proposition, et en l'appliquant à une proposition quelconque ϕ on obtient une démonstration de $\phi \Rightarrow \phi$. Le mécanisme d'instanciation des variables quantifiées universellement est donc ici aussi complètement explicite.

De façon remarquable, la Π-abstraction permet de représenter tous les connecteurs dans $\lambda 2$. Par exemple, $\bot$ s'écrit $\forall\alpha.\alpha$, et la règle d'élimination $(\bot E)$ se traduit par le fait que, si $\Gamma \vdash A : \Pi\alpha{:}*.\alpha$ alors $\Gamma \vdash A\phi : \phi$ pour toute proposition ϕ. Cette remarque explique également pourquoi la consistance des PTS qui étendent $\lambda 2$ (corollaire 6) entraîne immédiatement la cohérence des systèmes de déduction naturelle correspondants.

Pour finir, les systèmes $\lambda\omega$ et λC, avec leurs types dépendant de types, permettent de manipuler des fonctions sur les propositions, et donc de passer à l'ordre supérieur. Par exemple, dans λC, on peut fournir une définition uniforme du quantificateur universel : on pose

$$\vdash ALL \equiv \lambda\alpha{:}*.\lambda p{:}(\Pi\alpha{:}*.*).\Pi a{:}\alpha.(p\ a) \ : \ \Pi\alpha{:}*.\Pi p{:}(\Pi\alpha{:}*.*).*$$

Autrement dit, pour tout type A et pour tout prédicat P sur A, on a $ALL\ P\ A =_\beta \Pi a{:}A.(P\ a)$, ce qui représente bien $\forall a.P(a)$.

Pour conclure cette section, établissons un récapitulatif des correspondances mises en évidence par l'isomorphisme de Curry-Howard-de Bruijn.

λ-calcul	déduction naturelle
λ-terme	démonstration
type	proposition
habitabilité	prouvabilité
β-radical	coupure
normalisation	élimination des coupures
λ-abstraction	implication
paire	conjonction
case	disjonction
Π-abstraction	quantification universelle
consistance	cohérence

1.2 Réécriture

Si le λ-calcul constitue un formalisme dans lequel s'expriment bien les notions de programmation fonctionnelle, il reste un paradigme de calcul assez minimaliste, au sens où tout (jusqu'aux entiers et booléens) doit être représenté. Nous utiliserons donc surtout les aspects déductifs du λ-calcul ; pour tout ce qui est plutôt calculatoire, nous nous appuierons sur la réécriture et sur le filtrage. La présentation faite ici s'appuie essentiellement sur les ouvrages de F. Baader et T. Nipkow [BN98], de C. et H. Kirchner [KK99], et du collectif Terese [Ter02].

1.2.1 Généralités

Dans toute cette section on se donne une signature Σ et un ensemble de variables algébriques $\mathcal{X}$ qui engendrent une algèbre de termes $\mathcal{T}(\Sigma, \mathcal{X})$.

Définition 19 (Filtrage, subsomption, équation de filtrage)

1. *Étant donnée une théorie équationnelle $\mathbb{T}$ sur $\mathcal{T}(\Sigma, \mathcal{X})$, pour deux termes s et t pris dans $\mathcal{T}(\Sigma, \mathcal{X})$, on dit que t* subsume *s modulo $\mathbb{T}$, noté $t \ll_{\mathbb{T}} s$, si et seulement s'il existe une substitution θ telle que $s =_{\mathbb{T}} t\theta$. On appelle alors θ un* $\mathbb{T}$-filtre *de t à s.*

   Le plus souvent $\mathbb{T}$ est la théorie vide et $=_{\mathbb{T}}$ est en fait $\equiv$. On omettra alors $\mathbb{T}$ et on parlera de filtrage syntaxique.

2. *Un* système d'équations de filtrage *S est un ensemble de paires de termes s_i, t_i que l'on note $\{t_i \ll^?_{\mathbb{T}} s_i\}_{i \in I}$. Une* solution *d'un tel système est une substitution θ telle que :*

   – $\mathcal{D}om(\theta) = \bigcup_{i \in I} \mathcal{FV}(t_i)$;
   – *θ est un $\mathbb{T}$-filtre de t_i à s_i pour tout $i \in I$.*

   On notera $\mathcal{S}ol(t \ll^?_{\mathbb{T}} s)$ l'ensemble des solutions de $\{t \ll^?_{\mathbb{T}} s\}$.

On appellera $\mathbb{T}$ la *théorie de filtrage* utilisée.

Exemple 11 *Dans la théorie vide :*
- *$f(x, y)$ subsume $f(a, a)$, avec le filtre $[x := a, y := a]$;*
- *$g(x)$ ne subsume pas $h(a)$ car les symboles de tête g et h ne coïncident pas ;*
- *$g(a)$ ne subsume pas $g(x)$ (si $a \in \Sigma$) car le filtre ne peut agir que sur $g(a)$ qui ne comporte aucune variable (c'est l'unification et non pas le filtrage qui permet d'instancier des variables dans les deux termes) ;*

- *$f(x,x)$ ne subsume pas $f(a,b)$ car aucun des deux filtres $[x := a]$ ou $[x := b]$ ne convient ;*
- *t subsume t pour tout terme t de $\mathcal{T}(\Sigma,\mathcal{X})$;*
- *x subsume tout terme de $\mathcal{T}(\Sigma,\mathcal{X})$.*

Le filtrage syntaxique est relativement facile à décider : pour résoudre un système d'équations de filtrage $\{t_i \ll^? s_i\}_{i\in I}$, il suffit d'appliquer, dans n'importe quel ordre et tant que possible, les transformations de la figure 1.10 (dans le style de l'algorithme de filtrage de G. Huet [Hue76]). Si l'on obtient **F**, alors le système est irrésoluble ; sinon, on obtient un système d'équations de la forme $\{x_i \ll^? s'_i\}$ où une même variable x_i est toujours associée au même terme s'_i. On en déduit alors immédiatement un filtre de t à s.

Par la même occasion, on peut voir que ces transformations donnent toujours le même résultat pour une équation de filtrage donnée. On constate donc que le filtrage syntaxique est *unitaire* : s'il existe un filtre θ de t à s, alors il est unique (au sens des valeurs prises par θ sur les variables de t), et on le notera donc $\theta_{(t\ll s)}$; autrement dit $\mathcal{S}ol(t \ll^?_\emptyset s) = \{\theta_{(t\ll s)}\}$.

$$
\begin{array}{rcll}
\{f(t_1,\ldots,t_n) \ll^? f(s_1,\ldots,s_n)\} \cup S & \Longrightarrow & \{t_1 \ll^? s_1, \ldots, t_n \ll^? s_n\} \cup S & \\
\{f(t_1,\ldots,t_n) \ll^? g(s_1,\ldots,s_n)\} \cup S & \Longrightarrow & \mathbf{F} & \text{si } f \not\equiv g \\
\{f(t_1,\ldots,t_n) \ll^? x\} \cup S & \Longrightarrow & \mathbf{F} & \\
\{x \ll^? s\} \cup S & \Longrightarrow & \mathbf{F} & \text{si } x \ll^? s' \in S \text{ pour un certain } s' \not\equiv s
\end{array}
$$

FIG. 1.10 – Filtrage syntaxique

Pour d'autres théories, on pourra par exemple s'intéresser à la décidabilité du filtrage ou au nombre de solutions potentielles d'une équation. On dira d'une théorie équationnelle $\mathbb{T}$ qu'elle est une théorie de filtrage *finitaire* si toute équation de filtrage modulo $\mathbb{T}$ admet un nombre fini de solutions. Les théories les plus usitées sont empruntées au domaine de l'algèbre : on choisit dans la signature un ou plusieurs symboles associatifs, commutatifs, à élément neutre, etc.

Nous pouvons maintenant présenter les notions de règle, de pas, de système et de relation de réécriture.

Définition 20 (Réécriture)

1. *Une* règle de réécriture *est une paire de termes de $\mathcal{T}(\Sigma,\mathcal{X})$, notée $l \to r$ et éventuellement assortie d'une étiquette ℓ, telle que :*
   - *le* membre gauche *l n'est pas une simple variable ;*
   - *toute variable du* membre droit *r apparaît également dans l.*

   Une règle est dite linéaire gauche *si une même variable n'apparaît pas deux fois dans l.*
2. *Le terme t* se réécrit en un pas *en t' selon la règle $\ell : l \to r$ si et seulement si $t \equiv C[l\theta]$ pour un certain contexte $C\Box$ (autrement dit si l subsume un sous-terme de t) et si $t' \equiv C[r\theta]$. Ceci définit une relation $\to_\ell$ sur $\mathcal{T}(\Sigma,\mathcal{X})$.*
3. *Un* système de réécriture (ou TRS) *$\mathcal{R}$ est un ensemble de règles de réécriture, éventuellement explicitement accompagné de la signature Σ.*
4. *La* relation de réécriture *$\to_\mathcal{R}$ associée au système $\mathcal{R}$ est l'union des relations $\to_\ell$ pour toutes les règles ℓ de $\mathcal{R}$. On pourra considérer les relations dérivées $\twoheadrightarrow_\mathcal{R}$ et $=_\mathcal{R}$ (par définition $\to_\mathcal{R}$ est déjà close par contexte).*

Une partie des symboles de la signature Σ sur laquelle on travaille ne sont jamais utilisés en tête des membres gauches de règles de réécriture. Par conséquent, un terme ne contenant que ces symboles est obligatoirement en forme normale (par rapport à $\rightarrow_{\mathcal{R}}$) : on considère donc que ces symboles sont des *constructeurs* des valeurs vers lesquelles on veut évaluer les termes.

Exemple 12 (Entiers de Peano) *On peut définir les entiers avec deux constructeurs seulement : 0 (d'arité 0) et S (d'arité 1). Un entier n est alors représenté par $S(\ldots S(0))$ (avec n occurrences de S). C'est le plus simple ensemble infini de termes que l'on peut définir par constructeurs.*

Ces constructeurs déterminent donc largement la structure des objets que l'on va manipuler. S'ils n'apparaissent pas en tête des membres gauches, en revanche, on va souvent discriminer les arguments des fonctions selon leurs constructeurs. Ainsi, une fonction f définie de façon récursive sur les entiers est typiquement représentée par un système de réécriture de la forme

$$\begin{cases} f(0) & \rightarrow & a \\ f(S(x)) & \rightarrow & g(f(x)) \end{cases}$$

Par opposition, on dira des symboles apparaissant en tête des membres gauches qu'ils sont *définis*, dans la mesure où ils représentent des fonctions dont les règles de réécriture explicitent la sémantique.

Par sa nature même, la réécriture fournit un paradigme de calcul facile à manipuler car très déclaratif : tout est donné sous forme de règles qui peuvent s'appliquer n'importe où dans les termes, et on obtient ainsi une présentation d'une congruence complexe $=_{\mathcal{R}}$ uniquement en décrivant des manipulations locales.

Dans une moindre mesure, on peut également partir d'une congruence $=_{\mathcal{E}}$ pour obtenir un système de réécriture. En effet, si $=_{\mathcal{E}}$ est spécifiée par un ensemble d'équations $\{l = r\}$, le système de réécriture $\{l \rightarrow r, r \rightarrow l \mid l = r \in \mathcal{E}\}$ engendre une relation de conversion $=_{\mathcal{R}}$ identique à $=_{\mathcal{E}}$. On peut même se contenter de considérer une seule des deux «orientations» pour chaque équation, étant donné le caractère symétrique de $=_{\mathcal{R}}$. En revanche, ce procédé n'est pas systématique puisqu'un axiome dans lequel un des membres est réduit à une variable ou dont les deux membres n'ont pas les mêmes variables ne donne pas immédiatement une règle de réécriture.

On a donc un moyen agréable de présenter une théorie équationnelle. Cependant, rien ne garantit *a priori* que cette présentation fournisse un moyen plus effectif de décider de l'égalité de deux termes. Le seul problème de la bonne orientation d'un système d'équations est non trivial et même souvent sans solution satisfaisante. Dans la section suivante, nous présentons les propriétés de confluence et de terminaison, qui donne tout leur sens aux systèmes de réécriture en tant que présentation des théories équationnelles.

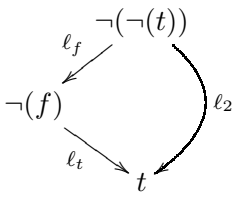
1.2.2 Quelques propriétés et exemples importants

Une première étape pour qu'un TRS soit utilisable comme mécanisme de décision pour sa fermeture réflexive symétrique transitive est de pouvoir se contenter d'appliquer les règles dans un seul sens. Pour cela, il faut pouvoir réduire la relation de conversion $=_{\mathcal{R}}$ à une relation orientée.

Définition 21 (Confluence)
Un système de réécriture $\mathcal{R}$ est dit confluent *(ou* Church-Rosser*) si la relation de réécriture associée $\rightarrow_{\mathcal{R}}$ est confluente (ou vérifie la propriété de Church-Rosser).*

Exemple 13 *Avec la signature* $\{t_0, f_0, \neg_1\}$*, le système*

$$\begin{cases} \ell_2 & : & \neg(\neg(x)) & \to & x \\ \ell_f & : & \neg(t) & \to & f \\ \ell_t & : & \neg(f) & \to & t \end{cases}$$



est confluent : le seul risque est que la règle ℓ_2 *interfère avec l'application successive des deux autres, mais dans tous les cas le résultat est le même, comme montré ci-dessus pour* $\neg(\neg(t))$*.*

Dans un tel système, pour décider de l'égalité de deux termes, il suffit donc de leur trouver un réduit commun. Cela dit, même ce problème est indécidable en général.

Exemple 14 *Avec la signature* $\{a_0, f_1, g_1\}$*, le système*

$$\begin{cases} f(x) & \to & f(f(x)) \\ g(x) & \to & g(g(x)) \\ f(a) & \to & a \\ g(a) & \to & a \end{cases}$$

est confluent mais les termes $f(a)$ *et* $g(a)$*, qui se réduisent tous les deux à* a*, ont chacun une infinité de réduits différents (respectivement* $f^n(a)$ *et* $g^n(a)$ *pour tout* $n \in \mathbb{N}$*), qu'il faudrait potentiellement tous examiner pour trouver le réduit commun.*

Pour rendre un système de réécriture utilisable, on ajoutera donc une seconde condition :

Définition 22 (Terminaison) *Un système de réécriture* $\mathcal{R}$ *est dit* terminant *si et seulement si la relation de réécriture associée* $\to_{\mathcal{R}}$ *est terminante sur l'ensemble des termes de* $\mathcal{T}(\Sigma, \mathcal{X})$*.*

Pour un tel système, on a donc une notion de forme normale selon $\to_{\mathcal{R}}$, qui de plus est unique si la propriété de Church-Rosser est vérifiée. Par conséquent, si on parvient à orienter les équations de définition d'une congruence $=_{\mathcal{E}}$ de façon à former un système de réécriture $\mathcal{R}$ confluent et terminant, on obtient une procédure de décision pour le *problème du mot* $t =^?_{\mathcal{E}} s$:

1. calculer *la* forme normale t' de t en appliquant des règles de $\mathcal{R}$ autant que possible ;
2. calculer *la* forme normale s' de s de même (par terminaison ces deux étapes terminent) ;
3. alors par Church-Rosser, $s =_{\mathcal{E}} t$ si et seulement si $s' \equiv t'$.

Le confluence et la terminaison fournissent donc un critère de décidabilité très commode ; on se doutera cependant que la plupart des systèmes de réécriture ne possèdent pas l'une ou l'autre de ces propriétés. Plutôt que de transformer le système jusqu'à en obtenir une version utilisable et qui engendre la même relation de conversion, on préfère parfois imposer un certain contrôle sur l'application des diverses règles de $\mathcal{R}$. Pour cela, C. Kirchner, H. Kirchner et M. Vittek ont défini la notion de *stratégie* [KKV95, Vit94].

Définition 23 (Stratégie, Système de calcul)

- *Un* pas de réécriture simple *est la donnée d'une étiquette de règle et d'une position dans un terme.*
- *Une* stratégie S *est un ensemble de séquences de pas de réécriture simples.*

- *Un* système de calcul *est la donnée de :*
  1. *un système de réécriture $\mathcal{R}$ dont toutes les règles sont étiquetées ;*
  2. *une* stratégie S.
- *Le terme t se réécrit en t en suivant la stratégie S si et seulement s'il existe une séquence de pas de réécriture dans S qui réécrit t en t'.*

Exemple 15 *Pour le système de réécriture de l'exemple 13, la stratégie $S = \{\ell_2^n \ell_f\}_{n\in\mathbb{N}}$ toujours appliquée en tête des termes réécrit (et normalise) tous les termes convertibles à f. En revanche, elle ne s'applique sur aucun terme convertible à t puisque pour ceux-ci il est impossible d'appliquer la règle ℓ_f en fin de réécriture.*

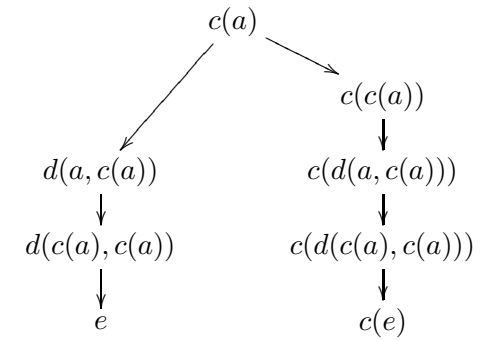
On décrira généralement les stratégies sous une forme plus pratique ; par exemple, pour le langage ELAN, P. Borovanský a défini un ensemble de *stratégies atomiques* [Bor98] qui peuvent être combinées pour en construire des plus élaborées :
- une étiquette de règle est une stratégie ;
- la stratégie $S_1; S_2$ désigne la concaténation des stratégies S_1 et S_2 ;
- la stratégie $\texttt{one}(S_1, \ldots, S_n)$ donne le premier résultat de la première stratégie applicable parmi ses arguments ;
- la stratégie $\texttt{repeat*}(S)$ désigne l'itération de la stratégie S tant qu'elle est applicable ;
- etc.

On peut également décrire la position à laquelle une règle s'applique de façon plus abstraite. On parle par exemple de stratégie de réduction du sous-terme le plus à gauche (*leftmost*), du sous-terme le plus interne (*innermost*) ou le plus externe (*outermost*).

Pour conclure cette section, étudions un cas de non-confluence qui sera particulièrement intéressant lorsque nous étudierons le calcul de réécriture.

Exemple 16 *Le système de réécriture ci-dessous n'est pas confluent.*

$$\begin{cases} d(x,x) & \to & e \\ c(x) & \to & d(x,c(x)) \\ a & \to & c(a) \end{cases}$$



En effet, le diagramme de droite ne peut être «fermé» : e est en forme normale, et une plus courte réduction de $c(e)$ à e devrait passer par $d(e, c(e))$, qui lui-même ne peut être réduit qu'après une réduction de $c(e)$ à e.

Cet exemple a été proposé par J. W. Klop [Klo80]. Il est particulièrement intéressant dans la mesure où les membres gauches des règles ne partagent aucun symbole (on parle de système *sans paire critique*). Il apparaît assez facilement que la non-confluence est ici due à la non-linéarité de la règle $d(x,x) \to e$.

On peut rendre la responsabilité de cette règle encore plus flagrante si on s'autorise à réécrire des λ-termes. En effet, en λ-calcul (non typé), on peut définir un combinateur de point fixe,

autrement dit un terme Y tel que pour tout terme M, on ait $\mathsf{Y}M \mapsto\!\!\!\!\to_\beta M(\mathsf{Y}M)$. On peut alors définir les termes suivants, dont on montre quelques réductions intéressantes :

$$\begin{array}{rcl} C & \equiv & \mathsf{Y}\big(\lambda y.\lambda x.d(x, y\,x)\big) \\ C & \mapsto\!\!\!\!\to_\beta & \big(\lambda y.\lambda x.d(x, y\,x)\big)\ C \\ & \mapsto_\beta & \lambda x.d(x, C\,x) \\ A & \equiv & \mathsf{Y}C \\ A & \mapsto\!\!\!\!\to_\beta & C\,A \end{array}$$

On voit que le λ-terme C reproduit le comportement de la constante c et de la seconde règle de réécriture, tandis que A reproduit le comportement de la constante a et de la troisième règle. On peut donc se passer de ces deux règles, d'où la variante ci-dessous de l'exemple 16.

Exemple 17 *Pour un λ-calcul non typé, l'ajout d'une unique règle de réécriture non linéaire gauche $\mathcal{R} = \{d(x, x) \to e\}$ à la β-réduction produit un calcul non confluent. En effet, le λ-terme A défini ci-dessus se réduit selon $\mapsto_{\beta\cup\mathcal{R}}$, d'une part à e, d'autre part à $C\,e$, et ces deux termes n'ont pas de réduit commun.*

Avec cette formulation on voit un peu plus clairement que c'est en fait la présence *simultanée* de non-terminaison et de non linéarité qui induit la non-confluence de la réduction étudiée. On peut d'ailleurs démontrer que tout TRS sans paire critique, et vérifiant la terminaison ou la linéarité, est confluent [Hue80, Toy87].

1.3 Déduction modulo

La déduction modulo a été proposée par G. Dowek, Th. Hardin et C. Kirchner [DHK03] comme un moyen de définir des démonstrations dans lesquelles les arguments calculatoires sont occultés. La vérification et la recherche d'une démonstration requièrent alors plus de puissance de calcul, mais les démonstrations obtenues sont plus compactes, et surtout les étapes de raisonnement y sont mises en évidence. Les travaux exposés dans le chapitre 6 ont été largement influencés par ce formalisme.

1.3.1 Principes

Un premier constat à l'origine de la déduction modulo est que, en logique du premier ordre, il est fastidieux de définir et d'utiliser des règles de calcul sur les termes algébriques. Prenons pour exemple le cas de l'arithmétique, traité exhaustivement par G. Dowek et B. Werner [DW05]. Une façon de définir l'addition sur les entiers de Peano est de considérer les deux axiomes suivants :

$$\begin{array}{c} \forall y.(0 + y = y) \\ \forall x.\forall y.(s(x) + y = x + s(y)) \end{array}$$

Les démonstrations se feront donc toutes dans un contexte Γ non vide. Celui-ci devra également contenir les axiomes concernant l'égalité, comme la réflexivité $\forall x.(x = x)$ ou la transitivité $\forall x.\forall y.\forall z.((x = y \wedge y = z) \Rightarrow x = z)$.

Une propriété aussi simple que $1+1=2$ (qui s'écrit ici $s(0)+s(0)=s(s(0))$) requiert alors la démonstration suivante :

$$\dfrac{\dfrac{\dfrac{\dfrac{(Ax)\ \overline{\Gamma \vdash \forall x.\forall y.\forall z.(x=y \wedge y=z \Rightarrow x=z)}}{\Gamma \vdash \forall y.\forall z.(s(0)+s(0)=y \wedge y=z \Rightarrow s(0)+s(0)=z)}(\forall E)}{\Gamma \vdash \forall z.(s(0)+s(0)=0+s(s(0)) \wedge 0+s(s(0))=z \Rightarrow s(0)+s(0)=z)}(\forall E)}{\Gamma \vdash s(0)+s(0)=0+s(s(0)) \wedge 0+s(s(0))=s(s(0)) \Rightarrow s(0)+s(0)=s(s(0))}(\forall E)}$$
$$(a)$$

$$\dfrac{\dfrac{\dfrac{(Ax)\ \overline{\Gamma \vdash \forall x.\forall y.(s(x)+y=x+s(y))}}{\Gamma \vdash \forall y.(s(0)+y=0+s(y))}(\forall E)}{\Gamma \vdash s(0)+s(0)=0+s(s(0))}(\forall E) \qquad \dfrac{(Ax)\ \overline{\Gamma \vdash \forall y.(0+y=y)}}{\Gamma \vdash 0+s(s(0))=s(s(0))}(\forall E)}{\Gamma \vdash s(0)+s(0)=0+s(s(0)) \wedge 0+s(s(0))=s(s(0))}(\wedge I)$$
$$(b)$$

$$\dfrac{(a) \qquad (b)}{\Gamma \vdash s(0)+s(0)=s(s(0))}(\Rightarrow E)$$

On est aisément convaincu qu'une démonstration devient plus lisible et que le raisonnement reste tout aussi valide et convaincant si on peut se passer de telles étapes. Cette idée de s'abstraire des étapes calculatoires dans une démonstration est connue sous le nom du *principe de Poincaré* [BB02].

En déduction modulo, on choisit pour cela d'identifier les propositions selon une congruence $\cong$, qui fera partie intégrante de la théorie dans laquelle on raisonne. Toutes les règles d'inférence sont alors définies avec la possibilité de choisir un quelconque représentant de la classe de congruence des propositions sur lesquelles on travaille. Par exemple, pour la logique minimale du premier ordre, les règles deviennent celles de la figure 1.11. On notera qu'il faut annoter les règles du quantificateur avec l'information nécessaire pour reconstruire la formule $\forall x.\phi$.

$$(Ax)\ \frac{}{\Gamma, \phi \vdash_{\cong} \psi} \quad \text{si } \phi \cong \psi$$

$$(\Rightarrow I)\ \frac{\Gamma, \psi \vdash_{\cong} \phi}{\Gamma \vdash_{\cong} \vartheta} \quad \text{si } \vartheta \cong (\psi \Rightarrow \phi) \qquad (\Rightarrow E)\ \frac{\Gamma \vdash_{\cong} \vartheta \qquad \Gamma \vdash_{\cong} \psi}{\Gamma \vdash_{\cong} \phi} \quad \text{si } \vartheta \cong (\psi \Rightarrow \phi)$$

$$(\forall I)\langle x,\ \phi\rangle\ \frac{\Gamma \vdash_{\cong} \phi}{\Gamma \vdash_{\cong} \psi} \quad \text{si } \psi \cong \forall x.\phi \text{ et } x \notin \mathcal{FV}(\Gamma)$$

$$(\forall E)\langle x,\ \phi,\ t\rangle\ \frac{\Gamma \vdash_{\cong} \psi}{\Gamma \vdash_{\cong} \vartheta} \quad \text{si } \psi \cong \forall x.\phi \text{ et } \vartheta \cong \phi[x := t]$$

FIG. 1.11 – Logique minimale du premier ordre modulo

Ainsi, les deux axiomes précédents seront supprimés du contexte, et on déclarera plutôt que

$$\begin{aligned} 0 + y &\cong y \\ s(x) + y &\cong x + s(y) \end{aligned}$$

Par conséquent il est immédiat que $s(0)+s(0) \cong s(s(0))$: dans une proposition, on peut donc instancier une même variable x indifféremment par l'un ou l'autre de ces termes. La démonstration précédente se réduit désormais à

$$(\forall E)\langle x,\ x{=}x,\ s(s(0))\rangle \dfrac{(Ax)\ \dfrac{}{\Gamma \vdash_{\cong} \forall x.(x = x)}}{\Gamma \vdash_{\cong} s(0) + s(0) = s(s(0))}$$

qui n'utilise plus que l'axiome de réflexivité. La congruence utilisée ici est décrite uniquement au niveau des termes algébriques, mais il est possible également de considérer directement des équations sur les propositions, par exemple

$$x * y \geq 0 \quad \cong \quad (x \geq 0 \wedge y \geq 0) \vee (x \leq 0 \wedge y \leq 0)$$

Encore une fois, les règles d'introduction et d'élimination s'appliquent quel que soit le représentant choisi dans une classe de congruence ; on peut donc avoir des démonstrations comme

$$(\vee I_l)\ \dfrac{\dfrac{\vdots}{\Gamma \vdash_{\cong} x \geq 0 \wedge y \geq 0}}{\Gamma \vdash_{\cong} x * y \geq 0}$$

Il apparaît assez rapidement que la congruence $\cong$ doit être définie avec soin, d'une part pour que la logique définie soit cohérente, d'autre part pour que la recherche de démonstrations reste réalisable. Comme on l'a vu à la section précédente, il est souvent commode de présenter une congruence sous la forme d'un système de réécriture. Dans toute la suite, on considérera que la congruence $\cong$ utilisée en déduction modulo est la relation de conversion $=_{\mathcal{R}}$ associée à un système de réécriture $\mathcal{R}$ composé :

- d'un ensemble de règles réécrivant des termes algébriques en termes algébriques ;
- d'un ensemble de règles réécrivant des propositions *atomiques* en propositions quelconques.

Par ailleurs on demandera généralement que le système $\mathcal{R}$ soit terminant et confluent, pour assurer que la validité d'une démonstration soit toujours vérifiable.

Le résultat primordial de la déduction modulo est le lemme d'équivalence, qui en énonce la correction et la complétude par rapport à la logique du premier ordre.

Lemme 8 (Équivalence [DHK03])
Pour toute congruence $=_{\mathcal{R}}$ définie par un système de réécriture qui réécrit des termes en termes et des propositions atomiques en propositions, il existe une théorie $\mathcal{T}h$ telle que

$$\Gamma \vdash_{=_{\mathcal{R}}} P \quad \textit{si et seulement si} \quad \mathcal{T}h, \Gamma \vdash P$$

Lorsque la congruence est décidable, on a le résultat supplémentaire suivant :

Lemme 9 (Décidabilité [DHK03, DW03])
Pour toute congruence $=_{\mathcal{R}}$ décidable et définie par un système de réécriture qui réécrit des termes en termes et des propositions atomiques en propositions, la vérification des démonstrations en déduction modulo $=_{\mathcal{R}}$ est également décidable.

Les règles de réécriture dont le membre gauche est une proposition atomique peuvent constituer une définition pour le symbole de prédicat de cette proposition, par exemple

$$X \subseteq Y \quad \rightarrow \quad \forall x.(x \in X \Rightarrow x \in Y)$$

donne bien une sémantique à $\subseteq$. Cependant, une telle règle a souvent un contenu déductif. Par exemple, on peut caractériser un anneau intègre par la règle

$$x * y = 0 \quad \rightarrow \quad x = 0 \vee y = 0$$

alors que l'égalité et la loi multiplicative sont définies par leurs propres systèmes de règles. Une règle peut même sembler n'être que définitionnelle, et pourtant être essentielle dans la présentation d'une théorie. Par exemple, pour définir l'arithmétique comme une théorie modulo, il faut introduire un symbole de prédicat N pour les entiers naturels, entièrement décrit par la seule règle

$$N(n) \quad \rightarrow \quad \forall p.(0 \in p \Rightarrow \forall y.(N(y) \Rightarrow y \in p \Rightarrow s(y) \in p) \Rightarrow n \in p)$$

qui est elle-même une traduction du schéma de récurrence.

On prendra garde à ne pas se laisser abuser par la notation $\rightarrow$ des règles de réécriture, qui n'a rien à voir avec une implication : les propositions des membres gauche et droit doivent être équivalentes, puisque l'intention est toujours de définir une congruence.

1.3.2 Démonstrations normales en déduction naturelle modulo

Une fois défini un cadre de déduction aussi général, il est intéressant de chercher des critères de correction sur les systèmes ainsi définis. Cette section s'inspire des travaux de G. Dowek et B. Werner [DW03], qui ont étudié l'élimination des coupures et la cohérence au moyen de termes de preuve très semblables à ceux qu'on a vu pour la logique du premier ordre. On pourra comparer ce langage de termes avec ceux proposés aux sections 6.1 et 6.2.

Dans un premier temps, montrons qu'une règle de réécriture mal choisie peut remettre en cause la cohérence de la théorie. Par exemple avec l'unique règle

$$R \quad \rightarrow \quad R \Rightarrow \bot$$

on obtient la démonstration de $\vdash_{\cong} \bot$ suivante :

$$\dfrac{\dfrac{\dfrac{(Ax)\ \overline{R \vdash_{\cong} R \Rightarrow \bot} \qquad (Ax)\ \overline{R \vdash_{\cong} R}}{(\Rightarrow E)\ R \vdash_{\cong} \bot}}{(\Rightarrow I)\ \vdash_{\cong} R \Rightarrow \bot} \qquad \dfrac{\dfrac{(Ax)\ \overline{R \vdash_{\cong} R \Rightarrow \bot} \qquad (Ax)\ \overline{R \vdash_{\cong} R}}{(\Rightarrow E)\ R \vdash_{\cong} \bot}}{(\Rightarrow I)\ \vdash_{\cong} R}}{(\Rightarrow E)\ \vdash_{\cong} \bot}$$

On remarquera que les deux sous-démonstrations sont exactement identiques, mais qu'on a choisi deux représentants distincts pour leurs conclusions (de même pour les différentes utilisations de la règle (Ax)).

On a vu à la section 1.1 que normalisation et cohérence étaient souvent intimement liées, et on pourrait donc croire que le problème posé par cette règle est dû à sa non-terminaison, ce qui a été initialement conjecturé. Cependant, Dowek et Werner en ont également donné une

présentation terminante (et confluente) en imposant quelques étapes de déduction sur le membre droit :

$$R \in R \quad \rightarrow \quad \forall y.(\ \forall x.(y \in x \Rightarrow R \in x) \ \Rightarrow\ y \in R \ \Rightarrow\ \bot\ )$$

En effet ici la proposition $R \in R$ ne réapparaît pas immédiatement, mais on peut la retrouver en instanciant y par R grâce à la règle $(\forall E)$. On obtient alors une démonstration de $\vdash_{\cong} \bot$ très similaire.

On remarquera que la démonstration de $\vdash_{\cong} \bot$ ci-dessus comporte une règle $(\Rightarrow I)$ suivie d'une règle $(\Rightarrow E)$ pour le même connecteur $\Rightarrow$. Si on tente de supprimer cette coupure par la même méthode que dans la section 1.1.2, on retrouve exactement la même démonstration.

Comme pour la déduction naturelle, il semble donc pratique de chercher à éliminer les coupures des démonstrations, puis de montrer la cohérence de ces démonstrations en forme normale. On remarque cependant qu'il faut imposer une condition supplémentaire sur la congruence utilisée :

Théorème 10 (Cohérence des démonstrations normales [DW03])
Si la congruence $\cong$ est définie par un système de réécriture confluent*, alors il n'existe aucune démonstration sans coupure de $\vdash_{\cong} \bot$.*

En effet, la confluence garantit que $\bot$ n'est convertible à aucune autre proposition non atomique. Il suffit alors de montrer que si $\vdash_{\cong} \phi$ est démontrable, alors ϕ n'est ni atomique, ni $\bot$.

Comme on pourra s'en douter, les deux règles de réécriture vues plus haut donnent un système de déduction qui ne vérifie pas l'élimination des coupures. G. Dowek et B. Werner ont caractérisé une famille de systèmes de réécriture pour lesquelles l'élimination des coupures (et donc la cohérence) sont vérifiées.

Définition 24 (Prémodèle)
Un prémodèle *pour la congruence $\cong$ est un modèle pour la signature Σ (au sens usuel des modèles de la logique du premier ordre) tel que, pour toute paire de formules convertibles $\phi \cong \psi$ et pour toute valuation de leurs variables libres, les interprétations de ϕ et de ψ dans ce modèle sont identiques.*

Théorème 11 (Normalisation modulo [DW03])
Pour toute congruence $\cong$ admettant un prémodèle, la déduction naturelle modulo $\cong$ vérifie l'élimination des coupures.

Nous ne rentrerons pas dans les détails de cette démonstration. Cependant, il est intéressant de constater qu'elle utilise comme termes de preuve des λ-termes, dont les règles de typage sont les règles usuelles où les types sont considérés modulo la congruence $\cong$. Par exemple, pour les variables, la λ-abstraction et l'application, on a les règles

$$(Ax)\ \frac{}{\Gamma, x : \phi \vdash_{\cong} x : \psi} \quad \text{si } \phi \cong \psi$$

$$(\Rightarrow I)\ \frac{\Gamma, x : \psi \vdash_{\cong} M : \phi}{\Gamma \vdash_{\cong} \lambda x.M : \vartheta} \quad \text{si } \vartheta \cong (\psi \rightarrow \phi) \qquad (\Rightarrow E)\ \frac{\Gamma \vdash_{\cong} M : \vartheta \quad \Gamma \vdash_{\cong} N : \psi}{\Gamma \vdash_{\cong} MN : \phi} \quad \text{si } \vartheta \cong (\psi \rightarrow \phi)$$

La β-réduction, quant à elle, doit être légèrement assouplie sur les constructions correspondant à la disjonction et le quantificateur existentiel.

La construction d'un prémodèle dépend largement du système de réécriture considéré. Cependant, il est possible de la faire systématiquement pour certaines classes de systèmes de réécriture, entre autres :

- les TRS qui ne réécrivent que des termes en termes ;
- les TRS confluents et terminants sans quantificateurs (dans les membres droits).

Maintenant que nous avons rappelés les différentes notions que nous allons utiliser tout au long de ce manuscrit, et qui sont largement à la base des travaux effectués, nous allons évoquer trois formalismes voisins de nos recherches, dans la mesure où ils sont consacrés à l'utilisation de réécriture (ou de filtrage) dans un calcul typé avec un sens logique fort.

2

Autour de l'interaction entre types, logique et réécriture

Ce chapitre présente trois directions de recherche dont les principes et les motivations sont très similaires aux nôtres : ajouter une forme de réécriture à un calcul typé, et étudier les propriétés de la logique sous-jacente. Pour éviter de nous égarer dans des subtilités techniques, nous commettrons parfois quelques simplifications par rapport aux travaux originaux, tout en tâchant d'en conserver les idées principales.

Dans la section 2.1, nous présentons des variantes du calcul des constructions où la relation de conversion $=_\beta$ est étendue par une relation de réécriture sur les objets ou sur les types. Ceci modifie donc le comportement du système de types, tout en conservant le système de réécriture considéré comme un objet à part, externe au formalisme.

Dans la section 2.2, nous expliquons comment le fait d'ajouter des capacités de filtrage très précises au λ-calcul permet de définir de nouvelles règles de typage qui mettent en évidence le lien entre les motifs acceptables et le traitement des différents connecteurs logiques dans un calcul des séquents.

Dans la section 2.3, nous évoquons une approche assez différente puisqu'elle ne s'appuie pas directement sur un λ-calcul typé : on part d'une sémantique en logique linéaire pour une forme généralisée de la réécriture, puis on définit un langage de termes de preuves qui s'avère très similaire au calcul de réécriture, le formalisme que nous étudierons à partir du prochain chapitre.

2.1 Calcul des Constructions Algébriques (CAC) et Higher-Order Recursive Path Ordering (HORPO)

L'idée de considérer un λ-calcul (typé) dans lequel on peut ajouter à la β-réduction une relation de réécriture $\rightarrow_{\mathcal{R}}$ n'est pas nouvelle. Dans le cas des λ-calculs simplement typé et polymorphe, M. Okada d'une part [Oka89] et V. Breazu-Tannen et J. Gallier d'autre part [GBT89] ont démontré que tout système $\mathcal{R}$ confluent et terminant préservait la normalisation forte du système (la confluence ayant déjà été montrée par Breazu-Tannen [BT88]).

Ce n'est cependant que récemment que cette étude a été étendue au cas des systèmes avec types dépendants, dont le calcul des constructions est un cas typique. En effet, dans ce cadre, la nouvelle relation de réduction $\mapsto_{\beta\cup\mathcal{R}}$ modifie non seulement la sémantique opérationnelle des

termes, mais aussi la relation de typage, car il faut considérer une règle de conversion adaptée :

$$(\text{CONV})\ \frac{\Gamma \vdash A : C \qquad \Gamma \vdash B : s}{\Gamma \vdash A : B} \quad \left(\begin{array}{c} B =_{\beta\cup\mathcal{R}} C \\ s \in \mathcal{S} \end{array} \right)$$

Les travaux de D. Walukiewicz-Chrząszcz [WC03] et F. Blanqui [Bla05b] couvrent deux approches de ce problème.

- La première prolonge les travaux de J.-P. Jouannaud et A. Rubio sur l'extension du Recursive Path Ordering (RPO) à l'ordre supérieur [JR99]. Elle considère donc des règles de réécriture dont le membre gauche peut être un λ-terme quelconque, mais qui ne s'appliquent qu'au niveau objet (c.-à-d. aux termes t tels que $\vdash t : T : *$).
- Le second, à l'opposé, s'appuie sur le *Schéma général*, initialement défini par J.-P. Jouannaud et M. Okada [JO95], et considère des règles de réécriture s'appliquant à tous les niveaux (y compris dans les types) mais dont le membre gauche est algébrique uniquement.

La technique employée pour démontrer la normalisation forte est similaire dans les deux cas : comme dans le cas du calcul des constructions, il s'agit de définir une interprétation des types sous forme d'ensemble de termes fortement normalisants, mais en tenant compte des définitions par réécriture. Comparons les conditions qui sont exigées sur la signature et sur le système de réécriture $\mathcal{R}$ considéré afin de garantir la bonne formation des candidats de réductibilité.

Confluence de $\rightarrow_{\beta\cup\mathcal{R}}$ et préservation du typage Pour tenir compte de l'ajout de la réécriture, la règle de conversion doit utiliser l'égalité $=_{\beta\cup\mathcal{R}}$, qui n'est pas forcément décidable et peut même poser des problèmes au niveau du typage : rien ne garantit *a priori* que deux termes convertibles soient du même type ni même qu'ils soient typables. La condition simple consistant à imposer que les membres gauche et droit de chaque règle soient bien typés et aient le même type conduirait à des règles non linéaires (à cause de multiples occurrences des variables de types notamment), ce qui n'est pas souhaitable notamment pour les raisons vues à la section 1.2.2.

Dans le calcul des constructions algébriques, on résout ce problème par la méthode usuelle : on exige que la relation $\rightarrow_{\beta\cup\mathcal{R}}$ soit confluente, et on peut se contenter d'exprimer la conversion par $B \rightarrow\!\!\leftarrow C$ (c.-à-d. B et C ont un réduit commun). Comme on n'utilise plus la réduction $\rightarrow_{\mathcal{R}}$ que dans le sens direct, la préservation du typage posera peu de problèmes : il suffit d'assurer que, si un membre gauche est typable, alors le membre droit l'est aussi (c'est le principe de base du Schéma général). Par contre, on peut signaler que, pour certains systèmes de réécriture, il est plus difficile de démontrer que $\rightarrow_{\beta\cup\mathcal{R}}$ est confluente lorsqu'on n'a pas encore démontré sa normalisation forte. Notons enfin que, dans le cas où il n'y a pas de règles au niveau des types, il n'est pas nécessaire de démontrer la confluence de $\rightarrow_{\beta\cup\mathcal{R}}$.

Pour HORPO les choses sont différentes : les règles de réécriture utilisées ne sont pas fixées *a priori*, et dans la règle de conversion, on remplace la condition $B =_{\beta\cup\mathcal{R}} C$ par $B \twoheadrightarrow_{\beta\cup\mathcal{R}} C$ ou $C \twoheadrightarrow_{\beta\cup\mathcal{R}} B$, ce qui permet de simuler une conversion par plusieurs applications successives de cette règle. De plus, pour décider de la condition $B \twoheadrightarrow_{\mathcal{R}} C$ il existe une règle d'inférence supplémentaire (REW) dont voici une version simplifiée :

$$(\text{REW})\ \frac{\Gamma \vdash l \succ r \qquad \Gamma \vdash l : T \qquad \Gamma \vdash r : T}{\Gamma \vdash C[l\theta] \rightarrow_{\mathcal{R}} C[r\theta]}$$

où $\succ$ est l'ordre qui définit HORPO. Toutes les paires de termes (l, r) telles que $l \succ r$ sont donc éligibles en tant que règles de réécriture. On constate que, d'une part, la réécriture des types

qui est rendue possible par (CONV) est entièrement retranscrite dans la dérivation de typage ; d'autre part, à chaque pas de réécriture, il est vérifié que les membres gauche et droit considérés ont un type commun. La correction de la conversion vis-à-vis du typage est alors assurée en un sens très fort : si $\Gamma \vdash A : B$ et $\Gamma \vdash A =_{\beta\cup\mathcal{R}} A'$ alors $\Gamma \vdash A' : B$.

Conditions sur les symboles constructeurs Les symboles constructeurs disponibles influencent largement le genre de fonctions récursives que l'on peut définir. Il a notamment été montré par N. P. Mendler [Men87] que, si les types des symboles constructeurs ne vérifient pas une condition de *positivité*, on peut en général obtenir facilement des points fixes et des incohérences. Par exemple, un type T avec le seul constructeur $c : (T \to U) \to T$ n'est pas admissible. On pourrait croire qu'il est impossible de produire un terme de ce type puisque c attend un argument utilisant lui-même T ; cependant, il est possible de définir le terme

$$\vdash \lambda x.f(x)x : T \to U$$

si le symbole f est de type $T \to (T \to U)$. Alors $c(\lambda x.f(x)x)$ est de type T et on peut lui appliquer le terme précédent. Pour se débarrasser de f et de c, il suffit de se donner la règle de réécriture $f(c(x)) \to x$, et le terme obtenu $(\lambda x.f(x)x)c(\lambda x.f(x)x)$ n'est pas normalisant selon $\to_{\beta\cup\mathcal{R}}$. C'est la présence de T à gauche de la flèche dans le type de l'argument de f qui rend ce « paradoxe » possible, et c'est ce genre de position (pour le type dont on donne un constructeur) qui est interdite par la condition de positivité.

Une condition similaire est reprise dans CAC comme dans HORPO ; de plus on utilise une condition d'*étoile-dépendance*. Celle-ci consiste à distinguer parmi les arguments d'un constructeur des *paramètres* et des *opérandes*, les premiers pouvant apparaître dans les types des seconds (ce qui a du sens dès qu'on utilise des types dépendants). L'exemple typique est celui des listes polymorphes : on se donne la constante $list : \Pi A{:}*.*$, de sorte que pour tout type τ, on a un type $list(\tau)$. Pour le constructeur $cons : \Pi A{:}*.\Pi x{:}A.\Pi l{:}list(A).list(A)$, il faut alors que A soit un paramètre puisqu'il apparaît dans les types de x et l, alors que x et l peuvent être des opérandes puisqu'ils n'apparaissent dans aucun type. On demandera par la suite que, dans tout membre gauche de règle, les paramètres soient des variables distinctes, et on aura donc tout intérêt à considérer un maximum d'arguments comme des opérandes pour accepter plus de règles.

Dans le calcul des constructions algébriques, on ajoute encore une notion d'accessibilité sur les différents arguments de chaque constructeur. Un constructeur peut donc prendre des arguments à peu près quelconques tant qu'il n'est pas permis de discriminer sur ces arguments. Ceci permet de décrire très précisément sur quels sous-termes la récursion est susceptible de porter, et donc d'avoir une notion assez fine de fonction récursive dont la terminaison est garantie.

Conditions sur les symboles définis Dans tous les cas, on se donne une précédence $>_F$ sur les symboles et un ordre sur les arguments de chaque symbole, cet ordre étant défini par une combinaison d'ordres lexicographiques et multiset. Dans HORPO, ceci induit directement un ordre $\succ$ sur tous les termes ; dans CAC, on définit la *clôture calculable* qui impose, lors du typage d'un membre droit de règle, que celui-ci soit plus petit que le membre gauche, et on peut prolonger cette clôture en un ordre sur tous les termes. On peut alors montrer la stabilité des candidats de réductibilité par rapport à la conversion : si $\Gamma \vdash A =_{\beta\cup\mathcal{R}} A'$ alors $[\![A]\!] = [\![A']\!]$.

Pour HORPO, on a vu que la décroissance des règles était directement imposée dans la dérivation de typage par la règle (REW). Pour CAC, une décroissance assez stricte est demandée pour les règles de réécriture définies au niveau des types ; par ailleurs, sur l'ensemble du système de réécriture (à part éventuellement sur une partie totalement algébrique), on demande que les

règles satisfassent le schéma général (c.-à-d. décroissance et typabilité). Signalons qu'en ajoutant des annotations de taille sur les types, on obtient une caractérisation plus générale des systèmes de réécriture $\mathcal{R}$ pour lesquels $\rightarrow_{\beta\cup\mathcal{R}}$ est normalisante, et de plus le typage est décidable [Bla04, Bla05a].

Instances de ces calculs Il est intéressant de voir dans quelle mesure ces résultats subsument des travaux précédents, notamment la démonstration de normalisation forte du calcul des constructions inductives de B. Werner [Wer94].

Le CAC permet de simuler une version faible des constructions inductives [Bla05c], dans laquelle on ne considère pas d'η-réduction, et où tout pas de calcul inductif $\rightarrow_\iota$ est suivi d'une normalisation selon $\mapsto_\beta$ (ce qui correspond d'ailleurs à la stratégie de réduction implantée dans Coq [Coq04]). De plus, les types inductifs doivent être *admissibles*, ce qui revient plus ou moins à vérifier la condition d'étoile-dépendance.

Similairement, dans HORPO, on ne peut considérer que des types inductifs *plats* (c.-à-d. tous leurs opérandes sont au niveau objet, ce qui assure que leurs règles d'élimination sont acceptées) et *compatibles* avec HORPO (c.-à-d. il y a une décroissance entre le type formé et les types des arguments de ses constructeurs).

Notons enfin que la déduction naturelle modulo peut être vue comme un calcul des constructions algébriques : les règles de déduction de la logique du premier ordre se traduisent en règles de typage par l'isomorphisme de Curry-Howard-de Bruijn, et la conversion des propositions modulo $=_\mathcal{R}$ se traduit immédiatement en une règle de conversion dans CAC.

Pour montrer la correction d'une instance de la déduction modulo, on a donc le choix entre construire un prémodèle comme on l'a vu à la section 1.3.2, ou bien vérifier les conditions de CAC ou de HORPO pour le système de réécriture utilisé. En particulier, on a vu que la déduction modulo est toujours correcte si on se limite à de la réécriture au niveau des termes, et ce résultat se retrouve immédiatement avec les conditions de normalisation forte dans CAC.

2.2 Un λ-calcul avec motifs et substitutions explicites

Nous venons de voir qu'il est possible de modifier radicalement les règles d'évaluation et de typage d'un λ-calcul simplement en considérant un système de réécriture externe, dont la relation de réduction va s'additionner à la β-réduction. On peut prendre le parti-pris inverse, qui consiste à inclure des capacités de filtrage directement dans le calcul considéré. Il devient alors plus difficile d'utiliser les résultats usuels sur la réécriture ; par contre, on obtient un calcul plus « expressif », au sens où tout n'a pas à être codé comme dans le λ-calcul. Par ailleurs, on peut espérer avoir un meilleur contrôle sur la forme de réécriture que l'on manipule.

C'est le choix qu'ont fait D. Kesner, L. Puel et V. Breazu-Tannen dans leur *Typed Pattern Calculus* (TPC) [KPT96], où la λ-abstraction se fait non plus sur une variable mais sur un motif. La famille de motifs utilisés est volontairement restreinte afin d'obtenir des règles de typage en correspondance étroite avec le calcul des séquents. Par la suite, S. Cerrito et D. Kesner ont proposé une version appelée TPC_{ES} [CK04] où le filtrage et la quasi-totalité des substitutions sont traités explicitement, et où un isomorphisme de Curry-Howard-de Bruijn avec le calcul des séquents est clairement établi. C'est cette version que nous présenterons ici. On peut noter que J. Forest a prolongé ces travaux avec le λP_w-calcul [For02, For03], toujours basé sur les mêmes λ-termes avec motifs, mais où toutes les substitutions sont traitées explicitement et où les constructions du genre $let \ldots in$ ne sont plus nécessaires. La correspondance avec le calcul des séquents est cependant moins immédiate.

La grammaire des pseudo-termes de TPC_{ES} est donnée dans la figure 2.1. On retrouve la λ-abstraction, mais qui porte sur un motif P. Celui-ci peut être une simple variable, mais aussi une paire $\langle P_1, P_2\rangle$ (l'argument doit être une paire), à laquelle correspond le type produit, une somme $(P_1 \mid_\xi P_2)$ annotée par une variable particulière ξ (l'argument doit être une injection), à laquelle correspond le type somme, ou une variable préfixée par l'opérateur $\sharp$, qui impose qu'elle ait un type fonctionnel. La contraction @ (l'argument doit être subsumé par chacun des deux motifs) et le joker _ (l'argument est quelconque) ne correspondent pas à un type en particulier puisqu'ils serviront à rendre compte d'opérations structurelles sur les séquents.

Pour que le filtrage ait un sens, on trouve les opérateurs réciproques des paires et sommes dans les termes eux-mêmes (paire, injections), ainsi qu'un opérateur de choix [|] et un opérateur de substitution explicite. Les variables ξ servent à transmettre le résultat d'un filtrage sur une somme à la construction [|]. Il n'y a pas de notion d'application comme en λ-calcul : celle-ci est décomposée en plusieurs constructions qui permettent de faire le filtrage explicitement. Le terme $\mathsf{let}\ M\ \mathsf{be}\ P\ \mathsf{in}\ N$ a pour sémantique « filtrer M selon P pour instancier les variables de N » ; le terme $(\lambda P.J)\ \mathsf{of}\ M\ \mathsf{is}\ Q\ \mathsf{in}\ N$ signifie « appliquer $\lambda P.J$ à M, puis filtrer le résultat selon Q pour instancier les variables de N » ; enfin le terme $z\ \mathsf{of}\ N\ \mathsf{is}\ P\ \mathsf{in}\ M$ attend une instanciation de z pour se réduire au précédent. Pour représenter une application MN on écrira alors $\mathsf{let}\ M\ \mathsf{be}\ \sharp z\ \mathsf{in}\ (z\ \mathsf{of}\ N\ \mathsf{is}\ y\ \mathsf{in}\ y)$. On pourra vérifier que, si $M \equiv \lambda P.M'$, alors ce terme se réduit bien à $\mathsf{let}\ N\ \mathsf{be}\ P\ \mathsf{in}\ M'$.

$$
\begin{array}{rrcl}
\text{(Pseudo-termes)} & T & ::= & x \mid \lambda P{:}\tau.T \mid \langle T, T\rangle \mid \mathsf{inl}_\sigma(T) \mid \mathsf{inr}_\tau(T) \mid [T \mid_\xi T] \mid T[x/T] \\
 & & & \mid \mathsf{let}\ T\ \mathsf{be}\ P{:}\tau\ \mathsf{in}\ T \mid z\ \mathsf{of}\ T\ \mathsf{is}\ P{:}\tau\ \mathsf{in}\ T \mid \lambda P{:}\tau.T\ \mathsf{of}\ T\ \mathsf{is}\ P{:}\tau\ \mathsf{in}\ T \\
\text{(Types)} & \tau & ::= & \iota \mid \tau \to \tau \mid \tau \times \tau \mid \tau + \tau \\
\text{(Motifs)} & P & ::= & x \mid \sharp x \mid \langle P, P\rangle \mid (P \mid_\xi P) \mid @(P, P) \mid \_
\end{array}
$$

FIG. 2.1 – Grammaire de TPC_{ES}

Les règles de réductions permettent de bien comprendre le sens de ces différentes constructions, et ce d'autant plus que le filtrage est explicite, et donc décomposé en opérations atomiques. Nous ne donnerons pas ici les règles $\to_{es}$ qui décrivent la propagation, l'évaluation et la composition de ces substitutions et qui sont assez standard.

Initialisation		
$(\lambda P.J)\ \mathsf{of}\ M\ \mathsf{is}\ Q\ \mathsf{in}\ N$	$\to_P$	$\mathsf{let}\ (\mathsf{let}\ M\ \mathsf{be}\ P\ \mathsf{in}\ J)\ \mathsf{be}\ Q\ \mathsf{in}\ N$
Filtrage		
$\mathsf{let}\ \langle M_1, M_2\rangle\ \mathsf{be}\ \langle P_1, P_2\rangle\ \mathsf{in}\ N$	$\to_P$	$\mathsf{let}\ M_1\ \mathsf{be}\ P_1\ \mathsf{in}\ (\mathsf{let}\ M_2\ \mathsf{be}\ P_2\ \mathsf{in}\ N)$
$\mathsf{let}\ M\ \mathsf{be}\ @(P_1, P_2)\ \mathsf{in}\ N$	$\to_P$	$\mathsf{let}\ M\ \mathsf{be}\ P_1\ \mathsf{in}\ (\mathsf{let}\ M\ \mathsf{be}\ P_2\ \mathsf{in}\ N)$
$\mathsf{let}\ \mathsf{inl}(M)\ \mathsf{be}\ (P_1 \mid_\xi P_2)\ \mathsf{in}\ N$	$\to_P$	$\mathsf{let}\ M\ \mathsf{be}\ P_1\ \mathsf{in}\ N[\xi := \mathtt{L}]$
$\mathsf{let}\ \mathsf{inr}(M)\ \mathsf{be}\ (P_1 \mid_\xi P_2)\ \mathsf{in}\ N$	$\to_P$	$\mathsf{let}\ M\ \mathsf{be}\ P_2\ \mathsf{in}\ N[\xi := \mathtt{R}]$
$\mathsf{let}\ (\lambda P.M)\ \mathsf{be}\ \sharp z\ \mathsf{in}\ N$	$\to_P$	$N[z/\lambda P.M]$
$\mathsf{let}\ M\ \mathsf{be}\ x\ \mathsf{in}\ N$	$\to_P$	$N[x/M]$
$\mathsf{let}\ M\ \mathsf{be}\ \_\ \mathsf{in}\ N$	$\to_P$	N

On notera que les variables de choix ξ ne sont pas traitées par des substitutions explicites. En effet, l'action de la « substitution » $[\xi := \mathtt{K}]$ (pour K valant L ou R) est assez particulière sur les

constructions de choix : elle leur communique les résultats des filtrages effectués sur les sommes.

$$\begin{array}{lcll} [M \mid_\xi N][\xi := \mathtt{L}] & = & M[\xi := \mathtt{L}] & \\ [M \mid_\xi N][\xi := \mathtt{R}] & = & N[\xi := \mathtt{R}] & \\ [M \mid_\zeta N][\xi := \mathtt{K}] & = & [M[\xi := \mathtt{K}] \mid_\zeta N[\xi := \mathtt{K}]] & \text{si } \zeta \neq \xi \end{array}$$

L'étude d'un calcul avec autant de règles est souvent fastidieuse : ici, même le simple fait que toute réduction d'un pseudo-terme donne à nouveau un pseudo-terme n'est pas immédiat. On peut cependant démontrer la confluence du calcul :

Théorème 12 (Confluence de TPC_{ES} [CK04])

La réduction $\mapsto$ définie comme la clôture par contexte de $\rightarrow_P \cup \rightarrow_{es}$ est confluente sur les pseudo-termes de TPC_{ES}.

Cette propriété s'obtient par l'étude de plusieurs sous-systèmes de réduction :

1. La réduction $\mapsto_{es}$ est fortement normalisante.
2. La réduction $\mapsto_{es}$ est confluente.
3. La réduction $\mapsto_P \mapsto_{es}^*$ sur les formes es-normales est confluente.
4. Si $e \mapsto e'$ alors leurs formes es-normales vérifient la relation $(\mapsto_P \mapsto_{es}^*)^*$.
5. Par le lemme d'interprétation [CHL96], la réduction $\mapsto$ est confluente.

Le calcul des séquents, comme la déduction naturelle, est un paradigme de déduction qui peut être adapté à plusieurs logiques différentes. Au lieu de considérer des règles d'élimination et d'introduction pour chaque connecteur, on ne considère plus que des règles d'introduction, mais qui peuvent également porter sur les formules du contexte. La notion de coupure est ici primitive au calcul : il existe une règle de coupure, et le théorème d'élimination des coupures dit qu'on peut démontrer les mêmes propositions avec ou sans cette règle. Voyons quelques-unes des règles de typage de TPC_{ES}, que nous allons mettre en regard des règles qui leur correspondent dans le calcul des séquents intuitionniste propositionnel.

$$(\textsc{var})\ \frac{}{\Gamma, x:\tau \vdash x:\tau} \qquad (Ax)\ \frac{}{\Gamma, A \vdash A}$$

$$(\to R)\ \frac{\Gamma, P:\tau \vdash M:\sigma}{\Gamma \vdash \lambda P:\tau.M : \tau \to \sigma} \qquad (\Rightarrow R)\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}$$

$$(\to L)\ \frac{\Gamma \vdash M:\tau \qquad \Gamma, P:\sigma \vdash N:\rho}{\Gamma, \sharp z:\tau\to\sigma \vdash z\ \mathsf{of}\ M\ \mathsf{is}\ P{:}\sigma\ \mathsf{in}\ N:\rho} \qquad (\Rightarrow L)\ \frac{\Gamma \vdash A \qquad \Gamma, B \vdash C}{\Gamma, A \Rightarrow B \vdash C}$$

$$(\times L)\ \frac{\Gamma, P:\tau, Q:\sigma \vdash M:\rho}{\Gamma, \langle P, Q\rangle : \tau\times\sigma \vdash M:\rho} \qquad (\wedge L)\ \frac{\Gamma, A, B \vdash C}{\Gamma, A \wedge B \vdash C}$$

$$(\times R)\ \frac{\Gamma \vdash M:\tau \qquad \Gamma \vdash N:\sigma}{\Gamma \vdash \langle M, N\rangle : \tau\times\sigma} \qquad (\wedge R)\ \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}$$

$$(+L)\ \frac{\Gamma, P:\tau \vdash M:\rho \qquad \Gamma, Q:\sigma \vdash N:\rho}{\Gamma, (P \mid_\xi Q):\tau+\sigma \vdash [M \mid_\xi N]:\rho} \qquad (\vee L)\ \frac{\Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma, A \vee B \vdash C}$$

$$(+R_1)\ \frac{\Gamma \vdash M:\tau}{\Gamma \vdash \mathsf{inl}_\sigma(M):\tau+\sigma} \quad (+R_2)\ \frac{\Gamma \vdash M:\sigma}{\Gamma \vdash \mathsf{inr}_\tau(M):\tau+\sigma} \qquad (\vee R)_1\ \frac{\Gamma \vdash A}{\Gamma \vdash A \vee B} \quad (\vee R)_2\ \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}$$

$$(sub)\ \frac{\Gamma \vdash M:\tau \qquad \Gamma, x:\tau \vdash N:\sigma}{\Gamma \vdash N[x/M]:\sigma}$$

$$(let)\ \frac{\Gamma \vdash M:\tau \qquad \Gamma, P:\tau \vdash N:\sigma}{\Gamma \vdash \mathsf{let}\ M\ \mathsf{be}\ P{:}\tau\ \mathsf{in}\ N:\sigma} \qquad (cut)\ \frac{\Gamma \vdash A \qquad \Gamma, A \vdash B}{\Gamma \vdash B}$$

Une correspondance immédiate apparaît entre abstraction, produit et somme et respectivement implication, conjonction et disjonction. On voit par ailleurs l'intérêt d'avoir des motifs dans les λ-abstractions : la décomposition d'un motif dans le contexte permet de rendre compte des règles d'introduction gauche des connecteurs. Enfin, le traitement des substitutions explicites et des let donne directement un mécanisme de coupure. On peut voir de même que les motifs de joker et de contraction correspondent respectivement aux règles d'affaiblissement et de contraction, qui sont des règles structurelles du calcul des séquents. On voit que la construction z of ... is ... in ... permet de représenter la règle $(\Rightarrow L)$; en revanche, le terme $\lambda P.M$ of ... is ... in ... n'est que le précédent dans lequel la variable z a été instanciée, et son typage donne lieu à une règle dérivée (comportant une coupure), ce qui s'explique par le fait qu'il n'est là que pour clore l'ensemble des pseudo-termes par réduction.

Comme pour la déduction naturelle et le λ-calcul, on a un véritable isomorphisme de Curry-Howard-de Bruijn : non seulement les termes représentent des démonstrations, mais leurs réductions reproduisent le mécanisme d'élimination des coupures. En effet, les règles d'initialisation et de filtrage explicite consistent à décomposer la formule de coupure selon ses connecteurs. Une fois la formule décomposée au maximum (on a atteint des variables donc des axiomes), les règles de propagation des substitutions explicites permettent d'échanger une coupure avec une règle d'introduction d'un connecteur pour la rapprocher des feuilles de l'arbre de dérivation. Lorsqu'elle atteint une variable, la substitution explicite s'évalue si nécessaire et disparaît, de même que la coupure correspondante.

On peut enfin montrer les bonnes propriétés de ce calcul typé.

Théorème 13 (Préservation du type dans TPC_{ES} [CK04])
Si $\Gamma \vdash T : \tau$ *et* $T \mapsto T'$ *alors* $\Gamma \vdash T' : \tau$.

La démonstration de ce théorème est longue en raison du nombre de cas à étudier, mais le seul cas qui semble vraiment poser problème est celui du filtrage sur les sommes. En effet, comme nous l'avons vu, les variables de choix ξ sont les seules à ne pas être traitées par des substitutions explicites, et donc le filtrage sur les sommes modifie instantanément la structure interne du terme alors que les autres constructions ne font que des décompositions et instanciations locales.

Théorème 14 (Normalisation forte de TPC_{ES} [CK04])
Si $\Gamma \vdash T : \tau$ *alors il n'existe pas de chaîne infinie de réductions* $\mapsto$ *à partir de* T.

Comme pour la confluence, on étudie séparément les systèmes $\mapsto_P$ et $\mapsto_{es}$, et on utilise les formes es-normales pour montrer la normalisation de l'ensemble des termes typés.

Concluons par une rapide discussion sur les motifs acceptés : ici, ils sont très fortement conditionnés par les règles de typage inspirées du calcul des séquents. On obtient un paradigme de calcul fonctionnel avec motifs et dans lequel on peut exprimer facilement des branchements ; il peut donc sembler déroutant de ne pas pouvoir exprimer des motifs à la ML comme par exemple

```
#let rec plus = function
  (0, y) -> y
 |(S x, y) -> S (plus x y)
```

Le problème vient du fait que le type des arguments de `plus` est *récursif*, dans le sens où le constructeur `S` prend un entier et renvoie un entier. Il est possible d'ajouter ces mécanismes à TPC_{ES}, au prix d'une construction de motif supplémentaire $fold$, de sa contrepartie dans les termes et d'un opérateur de récursion explicite. Cependant, dans un tel cadre, la définition de types récursifs reste assez fastidieuse, et la normalisation forte n'est évidemment plus valide.

2.3 Réécriture de forme supérieure

Dans la section 1.1, nous avons vu qu'un λ-terme pouvait représenter une démonstration dans une logique donnée. C. Schürmann et A. Stump ont proposé de voir un chemin de réécriture comme une démonstration dans un fragment de la logique linéaire [SS04]. Les « termes de preuve » obtenus dépassent largement le cadre de la réécriture du premier ordre, et on peut en déduire une notion de *réécriture de forme supérieure* qui permet de réécrire des règles (à ne pas confondre avec la *réécriture d'ordre supérieur*, qui s'attache essentiellement à réécrire des λ-termes).

Voyons tout d'abord la logique considérée et le lien avec la réécriture. Les règles données sont celles de la logique linéaire : elles sont très semblables à celles de la logique du second ordre, à ceci près que les hypothèses du contexte ne peuvent être utilisées qu'une fois et une seule. C'est ce qui explique que, dans la règle $(\multimap E)$, on partitionne le contexte en Γ et Γ'. Ainsi, dans cette logique, on peut démontrer $p \multimap (p \multimap q) \multimap q$ mais pas $p \multimap (p \multimap q) \multimap (p \multimap q \multimap r) \multimap r$ car il faudrait utiliser deux fois l'hypothèse p. La seule règle supplémentaire ici est (Cong), qui permet de travailler à l'intérieur d'un contexte algébrique $C\Box$, ce qui est souhaitable dès lors que l'on veut utiliser la réécriture.

L'idée est alors de voir une règle de réécriture $A \to B$ comme une trace de la démonstration de la proposition $A \multimap B$. La conjonction & permet de rendre compte des différents chemins de réduction possibles à partir d'un même terme, et le contexte Γ rassemble les différentes règles

$$(Ax)\ \frac{}{A \vdash A} \qquad (\top I)\ \frac{}{\Gamma \vdash \top} \qquad (\textsc{Cong})\ \frac{\Gamma \vdash A \multimap B}{\Gamma \vdash C[A] \multimap C[B]}$$

$$(\multimap I)\ \frac{\Gamma, A \vdash B}{\Gamma \vdash A \multimap B} \qquad (\multimap E)\ \frac{\Gamma \vdash A \multimap B \qquad \Gamma' \vdash A}{\Gamma, \Gamma' \vdash B}$$

$$(\& I)\ \frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \& B} \qquad (\& E_l)\ \frac{\Gamma \vdash A \& B}{\Gamma \vdash A} \qquad (\& E_r)\ \frac{\Gamma \vdash A \& B}{\Gamma \vdash B}$$

$$(\forall I)\ \frac{\Gamma \vdash A}{\Gamma \vdash \forall x.A}\ (x \notin \mathcal{FV}(\Gamma)) \qquad (\forall E)\ \frac{\Gamma \vdash \forall x.A}{\Gamma \vdash A[x := t]}$$

FIG. 2.2 – Logique linéaire + congruence

de réécriture utilisées (vu le cadre linéaire, Γ contient autant de fois chaque règle que celle-ci est utilisée).

On recherche alors une traduction $[\![\ ]\!]$ de la réécriture vers la logique linéaire telle que :
- si $t \twoheadrightarrow_{\mathcal{R}} t_i$ pour tout $i \in [1..n]$, alors $[\![\mathcal{R}]\!] \vdash [\![t]\!] \multimap [\![t_1]\!] \& \ldots \& [\![t_n]\!]$;
- si $[\![\mathcal{R}]\!] \vdash [\![t]\!] \multimap [\![t']\!]$, alors $t \twoheadrightarrow_{\mathcal{R}} t'$.

Il faut pour cela traduire à la fois les termes, les ensembles de termes et les règles de réécriture en logique linéaire :

$$\begin{array}{rcll} [\![x]\!] & = & x & \\ [\![f(t_1, \ldots, t_n)]\!] & = & f([\![t_1]\!], \ldots, [\![t_n]\!]) & \\ [\![\{t_1, \ldots, t_n\}]\!] & = & [\![t_1]\!] \& \ldots \& [\![t_n]\!] & \\ [\![l \to r]\!] & = & \forall x_1 \ldots \forall x_n [\![l]\!] \multimap [\![r]\!] & \text{où } \mathcal{FV}(l) = \{x_1, \ldots, x_n\} \end{array}$$

On voit bien comment une occurrence d'une règle $[\![l \to r]\!]$ dans le contexte permet de réécrire un terme : l'application successive de règles $(\forall E)$ permet d'instancier les différentes variables de l et de r, puis la règle (CONG) permet de déterminer à quelle position la réécriture a lieu, et enfin la règle $(\multimap E)$ effectue la réduction proprement dite.

Cependant, la correspondance obtenue n'est pas parfaite, et notamment beaucoup de dérivations valides en logique intuitionniste ne représentent pas clairement une réduction de réécriture. On a donc le théorème suivant :

Théorème 15 (Correction et complétude [SS04])
- *Complétude : si $t \twoheadrightarrow_{\mathcal{R}} t_i$ pour tout $i \in [1..n]$, alors $[\![\mathcal{R}]\!] \vdash [\![t]\!] \multimap [\![t_1]\!] \& \ldots \& [\![t_n]\!]$.*
- *Correction restreinte : si $[\![\mathcal{R}]\!] \vdash [\![t]\!] \multimap [\![t']\!]$ est dérivable avec au plus une seule utilisation de la règle* (CONG) *en fin de dérivation, alors $t \twoheadrightarrow_{\mathcal{R}} t'$.*

La restriction sur la correction doit se comprendre ainsi : les règles usuelles de la logique linéaire permettent de décrire le mécanisme de réécriture en tête des termes, à condition de considérer une démonstration en forme normale (au sens de l'élimination des coupures). La règle (CONG) revient à placer cette réécriture à l'intérieur d'un terme, mais avec cette règle la notion de démonstration normale n'est plus la même, et donc on ne peut pas toujours se ramener à un chemin de réécriture au sens usuel du terme. Le fait de lever la restriction sur (CONG) est justement ce qui permet de définir la réécriture de forme supérieure.

Une fois ce cadre défini, il est intéressant de ne pas se limiter à la réécriture du premier ordre, dans la mesure où on peut définir des règles comme $(p \multimap q) \multimap (q \multimap p)$, qui réécrit une règle en une autre règle. Il apparaît assez vite qu'on peut conserver les règles de la figure 2.2 à la seule condition de considérer les contextes comme des listes ordonnées. Ainsi, dans la règle $(\multimap E)$, il faut nécessairement utiliser les hypothèses dans un ordre bien précis.

Exemple 18 (Logique linéaire ordonnée) *En logique linéaire ordonnée, on peut démontrer* $(p \multimap q) \multimap p \multimap q$ *:*

$$\dfrac{\dfrac{\dfrac{(Ax)\ \overline{}}{p \multimap q \vdash p \multimap q} \qquad (Ax)\ \dfrac{}{p \vdash p}}{(\multimap E)\ \dfrac{}{p \multimap q, p \vdash q}}}{(\multimap I)\ \dfrac{}{p \multimap q \vdash p \multimap q}}}{(\multimap I)\ \dfrac{}{\vdash (p \multimap q) \multimap (p \multimap q)}}$$

mais pas $p \multimap (p \multimap q) \multimap q$ *: on ne peut pas trouver une formule de coupure* A *telle que*

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma \vdash A \multimap q} \qquad \dfrac{\vdots}{\Gamma' \vdash A}}{(\multimap E)\ \dfrac{}{p, p \multimap q \vdash q}}}{(\multimap I)\ \dfrac{}{p \vdash (p \multimap q) \multimap q}}}{(\multimap I)\ \dfrac{}{\vdash p \multimap (p \multimap q) \multimap q}}$$

Le problème de la seconde formule se comprend mieux si on regarde les buts intermédiaires du point de vue de la réécriture. Par exemple, $p \vdash (p \multimap q) \multimap q$ n'a pas une interprétation naturelle car p ne représente pas une règle.

On définit donc la *réécriture de forme supérieure* par simple extension des propriétés données au premier ordre : $t \to_{\mathcal{R}} t'$ si et seulement si $[\![\mathcal{R}]\!] \vdash [\![t]\!] \multimap [\![t']\!]$. Ceci permet d'exprimer assez naturellement des mécanismes de haut niveau sur des fonctions, comme par exemple la génération de programmes ou des opérations sur les variables liées.

Pour obtenir un *calcul de réécriture* (au sens de Stump et Schürmann), il suffit d'ajouter aux termes de la logique linéaire ordonnée une application @ avec la réduction

$$t \mathbin{@} u \ \mapsto\ u' \ \text{ ssi } \ u\theta \to_{t\theta} u'\theta \ \text{ (c.-à-d. } u\theta \text{ se réécrit en } u'\theta \text{ selon } t\theta) \text{ pour toute } \theta$$

Pour les termes clos, la condition de réduction est donc simplement $u \to_t u'$, ce qui correspond bien à l'idée d'appliquer la règle t au terme u.

Le calcul de réécriture que nous présenterons au prochain chapitre a une syntaxe très similaire, et ses règles de réduction s'écrivent ici (la troisième règle caractérisant le cas où les structures sont des ensembles) :

$$\begin{aligned}
(l \to r) \mathbin{@} t &\mapsto \{r\theta_1, \ldots, r\theta_n\} \qquad \text{où } \mathcal{S}ol(l \ll t) = \{\theta_1, \ldots, \theta_n\} \\
\{t_1, \ldots, t_n\} \mathbin{@} u &\mapsto \{t_1 \mathbin{@} u, \ldots, t_n \mathbin{@} u\} \\
\{t_1, \ldots, \{u_1, \ldots, u_n\}, \ldots, t_n\} &\mapsto \{t_1, \ldots, u_1, \ldots, u_n, \ldots, t_n\}
\end{aligned}$$

On peut constater qu'elles sont admissibles, au sens où leur membre gauche implique effectivement leur membre droit. Il serait intéressant de voir si elles peuvent constituer (quitte à en ajouter d'autres du même genre) un ensemble complet de règles pour la logique de la figure 2.2.

Pour conclure ce chapitre, notons quelques points communs et différences entre les trois formalismes évoqués et le calcul de réécriture sur lequel nous allons travailler :

1. Dans CAC ou dans HORPO, c'est bien de la réécriture, au sens le plus général du terme, qui est ajoutée au λ-calcul ; par contre, elle est encore considérée comme un objet externe, et il n'est par exemple pas possible de passer une règle en argument d'une fonction. Dans le ρ-calcul, les règles comme les termes seront des citoyens de première classe, qui peuvent être pris en argument et modifiés.
2. Dans TPC_{ES} et les autres calculs avec filtrage explicite, la situation est inverse : le filtrage est réellement intégré au calcul, mais par contre la classe de motifs considérés est relativement restreinte. En ρ-calcul, nous considérerons des motifs plus généraux, la contrepartie étant que nous ne pourrons pas définir des règles de typage aussi spécifiques.
3. La réécriture de forme supérieure est un peu en marge des deux précédents, mais on peut tout de même remarquer qu'elle ne nous donne pas un mécanisme de *typage* pour le ρ-calcul, puisqu'une règle se traduit en une formule, et non pas en une démonstration d'une formule. De plus la correspondance exacte entre notre calcul de réécriture et celui défini ici reste à préciser.

Nous n'avons quasiment pas parlé ici de la *réécriture d'ordre supérieure*, qui consiste à réécrire des λ-termes et constitue donc une autre approche de la combinaison de λ-calcul et de réécriture. Les formulations les plus connues sont les *Combinatory Reduction Systems (CRS)* [KOR93], les *Higher-Order Rewrite Systems (HRS)* [Nip91] et les *Expression Reduction Systems (ERS)* [Kha90].

Cependant, à notre connaissance, les aspects logiques de ces différents formalismes n'ont pas été étudiés à ce jour. Pour une comparaison approfondie avec le ρ-calcul, le lecteur intéressé pourra se référer aux travaux de C. Bertolissi [BCK05, Ber05].

3

Calcul de réécriture

Dans ce chapitre, nous présentons le calcul de réécriture (ou ρ-calcul), sur lequel nous baserons toute la suite de notre étude. La première version, introduite par H. Cirstea et C. Kirchner [CK01] pour donner une sémantique à ELAN [BCD$^+$00], est très semblable au calcul vu en section 2.3, notamment dans son traitement de la distributivité des ensembles. Cependant, nous nous focaliserons sur une version mise au point avec L. Liquori [CKL01], qui met moins l'accent sur la modélisation de la réécriture et qui est plus proche du λ-calcul pur. Nous ferons quelques rappels concernant la confluence du calcul et nous donnerons quelques exemples montrant son expressivité.

3.1 Définitions

Comme pour TPC_{ES} [CK04], on peut voir le calcul de réécriture comme un λ-calcul où l'abstraction porte sur un motif. Celui-ci peut être quasi arbitraire, et donc la sémantique du calcul dépend largement de la façon dont le filtrage est évalué. Rappelons que, pour cela, on se donne une théorie équationnelle $\mathbb{T}$ qui détermine une égalité $=_{\mathbb{T}}$ entre les termes (voir déf. 5). Les solutions d'un problème de filtrage $P \ll^?_{\mathbb{T}} A$ sont l'ensemble des substitutions $\{\theta_i\}_{i\in I}$ telles que $P\theta_i =_{\mathbb{T}} A$. Pour certaines théories $\mathbb{T}$, un même problème de filtrage peut donc avoir plusieurs solutions, et pour éviter le non-déterminisme, notre calcul avec motifs doit donc disposer d'un moyen de représenter un ensemble de résultats.

En plus des λ-termes et des motifs, on introduit donc une notion de *structure* $T \wr T$ qui permet de rassembler plusieurs termes en un seul. En première approche, elle peut être vue comme une paire, mais on l'utilisera parfois comme un ensemble ou une liste ordonnée, en se donnant une théorie équationnelle appropriée sur le symbole $\wr$. Comme les règles sont elles-mêmes des termes, la structure permet de représenter non seulement des ensembles de résultats, mais aussi des ensembles de règles, ce qui sera utile lorsque nous voudrons encoder la réécriture.

On en arrive donc à la syntaxe suivante :

$$\begin{array}{lrcl} \text{(Motifs)} & \mathcal{P} & \subseteq & \mathcal{T}_\rho \\ \text{(Termes)} & \mathcal{T}_\rho & ::= & \mathcal{X} \mid \mathcal{K} \mid \lambda\mathcal{P}.\mathcal{T}_\rho \mid \mathcal{T}_\rho\,\mathcal{T}_\rho \mid \mathcal{T}_\rho \wr \mathcal{T}_\rho \end{array}$$

pour un ensemble (infini) de variables $\mathcal{X}$ et un ensemble (infini) de constantes $\mathcal{K}$. On utilisera les mêmes conventions de notations et de priorité des opérateurs qu'en λ-calcul. On notera qu'il n'y a *pas de signature* à proprement parler, au sens où les constantes de $\mathcal{K}$ ne sont pas munies d'une arité particulière. En effet, les termes algébriques sont représentés de façon currifiée au moyen

de l'application : si f est une constante, on considère la représentation

$$\begin{aligned} [\![x]\!] &= x \\ [\![f(t_1,\ldots,t_n)]\!] &= ((f\,[\![t_1]\!])\ldots)[\![t_n]\!] \end{aligned}$$

Par conséquent, une même constante est susceptible d'être appliquée à un nombre arbitraire d'arguments. L'ensemble des motifs admissibles n'est pas fixé *a priori*, et la sémantique du calcul dépendra donc largement de son choix. On peut cependant en citer quelques exemples.

Exemple 19 (Classes de motifs pour le calcul de réécriture)

1. *La classe de motifs la plus simple est $\mathcal{P} = \mathcal{X}$ qui, comme on le verra, redonne le λ-calcul. Il est souhaitable que $\mathcal{X}$ au moins soit inclus dans toute classe de motifs.*
2. *La classe des motifs algébriques*

$$\mathcal{P} \;::=\; \mathcal{X} \mid f\,\overline{\mathcal{P}}$$

   simule exactement l'union de toutes les algèbres de termes $\mathcal{T}(\mathcal{K},\mathcal{X})$ que l'on peut construire en assignant tous les choix d'arités possibles aux constantes de $\mathcal{K}$. C'est une des classes que nous utiliserons le plus souvent.
3. *Pour une classe de motifs donnée, on pourra se restreindre aux motifs linéaires.*
4. *En toute généralité, on peut choisir $\mathcal{P} = \mathcal{T}_\rho$. Il paraît alors naturel de considérer un filtrage d'ordre supérieur (modulo β-conversion), mais il est possible également de vouloir discriminer sur la forme d'une fonction, comme nous le verrons à la section 3.2.*

L'abstraction étant généralisée, il faut revoir la notion de variable libre.

Définition 25 (Variables libres dans le calcul de réécriture)

$$\begin{aligned} \mathcal{FV}(x) &= \{x\} \\ \mathcal{FV}(a) &= \emptyset \\ \mathcal{FV}(A\,B) &= \mathcal{FV}(A) \cup \mathcal{FV}(B) \\ \mathcal{FV}(\lambda P.A) &= \mathcal{FV}(A) \setminus \mathcal{FV}(P) \\ \mathcal{FV}(A \wr B) &= \mathcal{FV}(A) \cup \mathcal{FV}(B) \end{aligned}$$

et ses variables liées sont définies de façon duale

$$\begin{aligned} \mathcal{BV}(x) &= \emptyset \\ \mathcal{BV}(a) &= \emptyset \\ \mathcal{BV}(A\,B) &= \mathcal{BV}(A) \cup \mathcal{BV}(B) \\ \mathcal{BV}(\lambda P.A) &= \mathcal{BV}(A) \cup \mathcal{BV}(P) \cup \mathcal{FV}(P) \\ \mathcal{BV}(A \wr B) &= \mathcal{BV}(A) \cup \mathcal{BV}(B) \end{aligned}$$

On vérifie aisément que, pour les motifs réduits à une variable, cette définition est conservative par rapport au λ-calcul. Encore une fois, on raisonnera généralement modulo α-conversion.

On peut alors définir les règles de réduction suivantes, paramétrées par la théorie équationnelle modulo laquelle on fait le filtrage (rappelons que $\theta \in \mathcal{S}ol(P \ll_{\mathbb{T}} B)$ si et seulement si $P\theta =_{\mathbb{T}} B$).

Définition 26 (Sémantique opérationnelle du calcul de réécriture)
Étant donnée une théorie équationnelle $\mathbb{T}$, on définit deux réductions sur les ρ-termes :

$$\begin{array}{rcll}(\lambda P.A)\,B & \to_\rho & A\theta_1 \wr \ldots \wr A\theta_n & \text{si } \mathcal{S}ol(P \ll_{\mathbb{T}} B) = \{\theta_1, \ldots, \theta_n\} \text{ avec } n > 0\\ (A_1 \wr A_2)\,B & \to_\delta & A_1\,B \wr A_2\,B & \end{array}$$

On notera $\to_{\rho\delta}$ la relation $\to_\rho \cup \to_\delta$, et $\mapsto_{\rho\delta}$, $\mapsto\!\!\!\!\to_{\rho\delta}$, $=_{\rho\delta}$ ses relations dérivées.

Soulignons immédiatement la condition $n > 0$ dans la ρ-réduction : s'il n'existe aucun filtre de P à B, cette réduction n'a pas lieu. En effet, comme montré à l'exemple 21, une instanciation ou une réduction dans B peut donner lieu à de nouveaux filtres. Une réduction prématurée vers une structure vide (qui d'ailleurs n'est pas présente dans la grammaire des termes) compromettrait alors directement la confluence du calcul. Il nous faut donc distinguer deux notions :

Définition 27 (Radical, préradical)
En ρ-calcul, on appellera préradical *tout terme $(\lambda P.A)\,B$.*
On appellera un tel terme radical *seulement si $P \ll^?_{\mathbb{T}} B$ admet au moins une solution.*

Exemple 20 (Termes et réductions)

1. *Avec un filtrage syntaxique ($\mathbb{T} = \emptyset$), on a*

$$\begin{array}{rcl}(\lambda x.A)\,B & \mapsto_\rho & A[x := B]\\ (\lambda a.A)\,a & \mapsto_\rho & A\\ (\lambda a.a)\,b & & \text{est en forme normale, bien qu'il soit un préradical.}\\ (\lambda(f\,x).(g\,x))\,(f\,a) & \mapsto_\rho & (g\,a)\\ (\lambda(f\,x\,x).x)\,(f\,a\,a) & \mapsto_\rho & a\\ (\lambda(f\,x\,x).x)\,(f\,a\,b) & & \text{est en forme normale.}\\ (\lambda a.b \wr \lambda a.c)\,a & \mapsto_\delta & (\lambda a.b)\,a \wr (\lambda a.c)\,a \;\mapsto_\rho\; b \wr (\lambda a.c)\,a \;\mapsto_\rho\; b \wr c\end{array}$$

2. *Dans une théorie où $f\,x\,y =_{\mathbb{T}} f\,y\,x$, on a*

$$\begin{array}{rcl}(\lambda x.x)\,(f\,a\,b) & \mapsto_\rho & f\,a\,b\\ (\lambda(f\,x\,y).x)\,(f\,a\,b) & \mapsto_\rho & a \wr b\end{array}$$

On notera qu'on ne considère qu'un filtre par classe d'équivalence, autrement dit on fait du filtrage de classes plutôt que du filtrage modulo.

Exemple 21 (Un préradical qui devient un radical)
Dans le terme $(\lambda(f\,x).(\lambda a.b)\,x)\,(f\,a)$, le préradical $(\lambda a.b)\,x$ est en forme normale. Cependant, la réduction externe le transforme en radical par instanciation de x :

$$(\lambda(f\,x).(\lambda a.b)\,x)\,(f\,a) \mapsto_\rho (\lambda a.b)a \mapsto_\rho b$$

Remarque 5 (Théorie sur le symbole de structure $\wr$)
Lorsque le symbole $\wr$ est purement syntaxique, la structure associée d'un arbre binaire orienté qui porte les termes au niveau des feuilles. Il est souvent commode de considérer une structure de données plus souple, obtenue en considérant une autre théorie équationnelle sur $\wr$:
– si $\wr$ est commutatif, les structures sont des arbres binaires non orientés ;

– *si $\wr$ est associatif, les structures sont des listes ordonnées ;*
– *si $\wr$ est associatif et commutatif, les structures sont des multiensembles ;*
– *si $\wr$ est associatif, commutatif et idempotent ($A \wr A =_{\mathbb{T}} A$), les structures sont des ensembles.*

On notera que dans les cas où $\wr$ n'apparaît pas dans les motifs (en particulier quand les motifs sont algébriques), cette théorie ne modifie pas le filtrage.

Sauf mention explicite du contraire, on considérera généralement les structures comme des ensembles.

3.2 Confluence

On peut voir assez rapidement que, selon la classe de motifs autorisés et la théorie de filtrage choisie, le calcul peut ou non être confluent. Nous allons ici donner un panel de contre-exemples à la confluence, et de conditions qui l'assurent.

À propos du filtrage d'ordre supérieur

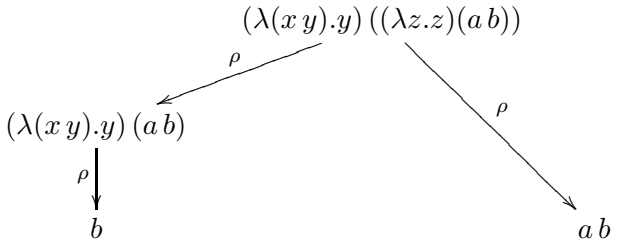
Dans le λ-calcul, le filtrage d'ordre supérieur se fait en général dans un cadre typé. Par ailleurs, il n'est pas encore totalement compris : les résultats les plus précis à ce jour sont la décidabilité du filtrage modulo $\beta\eta$ à l'ordre 4, par V. Padovani [Pad00], et l'indécidabilité du filtrage modulo β à l'ordre 6, par R. Loader [Loa03].

En imposant des restrictions sur la forme des motifs, il est possible de se détacher partiellement du typage, comme cela est fait pour les *Higher-Order Patterns* de D. Miller [Mil91].

Dans le ρ-calcul, le filtrage d'ordre supérieur est actuellement à l'étude et aucun résultat à ce sujet n'est connu pour l'instant ; par la suite nous éviterons donc de considérer des motifs trop généraux, notamment des motifs comportant une λ-abstraction.

Cas du filtrage syntaxique

Une première précaution à prendre, même dans le cas simple du filtrage syntaxique, concerne l'utilisation de l'application. En effet, considérons l'exemple suivant :



Selon qu'on choisit de résoudre d'abord le radical $(\lambda z.z)\,(a\,b)$ (réduction de gauche) ou le plus externe (réduction de droite), on obtient deux résultats complètement différents et pour lesquels on ne trouvera pas de réduit commun (ils sont tous deux en forme normale). Le problème est dû ici à la variable x du motif $(x\,y)$, qui est située à gauche de l'application (on dira qu'elle est en position *active* ou *fonctionnelle*) et qui permet donc de faire du filtrage sur les fonctions.

On peut vouloir définir un filtrage syntaxique sur les fonctions pour discriminer selon leur *expression* et non pas leur valeur, comme cela a été proposé dans [BCKL03]. Dans ce cas, il faut donc n'avoir que des constantes (et pas de variables) en position active. Ce filtrage est très prometteur dans le cadre de la transformation de programmes, mais son intérêt pour la déduction automatique est moins clair. Si l'on interdit à la fois les abstractions et les variables

actives dans les motifs, on retrouve la classe des motifs algébriques (éventuellement étendue par les structures), à laquelle nous restreindrons notre étude à partir de maintenant.

Pour montrer rapidement les problèmes posés par le filtrage d'ordre supérieur en ρ-calcul, voyons les solutions du problème $x\,y \ll^? (\lambda z.z)\,(a\,b)$ avec un filtrage du second ordre modulo $\beta\eta$-conversion :

$$\left\{\begin{array}{l} x =_{\beta\eta} \lambda z.z \\ y =_{\beta\eta} a\,b \end{array}\right. \qquad \left\{\begin{array}{l} x =_{\beta\eta} \lambda z.az \\ y =_{\beta\eta} b \end{array}\right. \qquad \left\{\begin{array}{l} x =_{\beta\eta} \lambda z.a\,b \\ y \text{ quelconque} \end{array}\right.$$

On constate que la dernière solution n'était même pas envisagée dans nos différentes réductions. Un filtrage modulo ρ devrait considérer des solutions supplémentaires comme par exemple $x =_\rho \lambda P.a\,z$ et $y =_\rho P[z := b]$ qui est une généralisation de la seconde solution.

Même dans le cas des motifs algébriques, un autre problème peut venir de la non-linéarité (c.-à-d. quand une variable apparaît deux fois dans un même motif). Comme on l'a vu dans l'exemple 17, la présence simultanée de λ-calcul et de filtrage non linéaire compromet très facilement la confluence ; le calcul de réécriture n'échappe pas à ce travers, comme le montre l'exemple suivant.

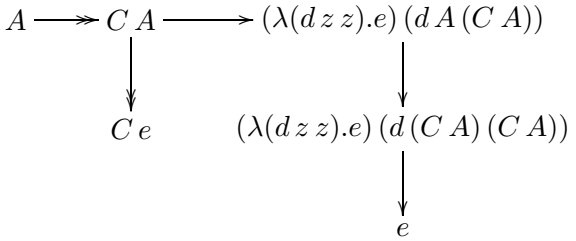
Exemple 22 *On définit le combinateur de point fixe* Y *exactement comme en* λ*-calcul :*

$$\mathsf{Y} \triangleq \Big(\lambda y.\lambda x.(x\,(y\,y\,x))\Big)\,\Big(\lambda y.\lambda x.(x\,(y\,y\,x))\Big)$$

et on a bien, pour tout ρ*-terme* A*, la réduction* $\mathsf{Y}\,A \mapsto\!\!\!\!\to_\rho A\,(\mathsf{Y}\,A)$*. On définit alors les termes suivants, qui sont très similaires à ceux de l'exemple 17, à ceci près que la règle non linéaire* $d(z,z) \to e$ *est intégrée dans* C*.*

$$\begin{array}{rcl} C & \equiv & \mathsf{Y}\,\big(\lambda y.\lambda x.(\lambda(d\,z\,z).e)\,(d\,x\,(y\,x))\big) \\ C & \mapsto\!\!\!\!\to_\rho & \big(\lambda y.\lambda x.(\lambda(d\,z\,z).e)\,(d\,x\,(y\,x))\big)\;C \\ & \mapsto_\rho & \lambda x.(\lambda(d\,z\,z).e)\,(d\,x\,(C\,x)) \\ A & \equiv & \mathsf{Y}\,C \end{array}$$

On a alors les réductions suivantes :



La constante e *est en forme normale, et dans la plus courte réduction de* $C\,e$ *à* e*, il faut forcément réduire ses radicaux de tête. Au bout de quelques* ρ*-réductions on obtient le terme* $(\lambda(d\,z\,z).e)\,(d\,e\,(C\,e))$*, qui ne peut être réduit en tête qu'après une réduction de* $C\,e$ *à* e*. Comme on a supposé la minimalité de la réduction, on conclut par l'absurde que* $C\,e$ *ne se réduit pas à* e*, et donc les deux réductions amorcées ne confluent pas.*

De manière générale, on supposera donc les motifs linéaires. Par la suite, nous donnerons des systèmes de types vérifiant la propriété de normalisation forte ; nous pourrons alors considérer à nouveau des motifs non linéaires puisque le terme Y n'est pas typable.

Une fois ces contre-exemples mis en évidence, on peut s'attacher à trouver une condition suffisante de confluence assez générale. Dans son étude sur le λ-calcul avec motifs [Oos90], V. van Oostrom définit la condition suivante :

Définition 28 (Rigid Pattern Condition (RPC))
On dit qu'un motif P est rigide *(ou qu'il vérifie RPC) si*

$$\forall \theta,\ \forall A,\quad P\theta \mapsto\!\!\!\!\!\rightarrow_{\rho\delta} A \ \Rightarrow\ A \equiv P\theta' \text{ avec } \theta \mapsto\!\!\!\!\!\rightarrow_{\rho\delta} \theta'$$

En particulier, tout motif algébrique linéaire vérifie RPC.

On a alors le résultat général suivant :

Théorème 16 (Confluence par RPC [BCKL03])
Avec un filtrage syntaxique, la relation $\mapsto_{\rho\delta}$ est confluente sur l'ensemble des ρ-termes dont les motifs vérifient RPC.

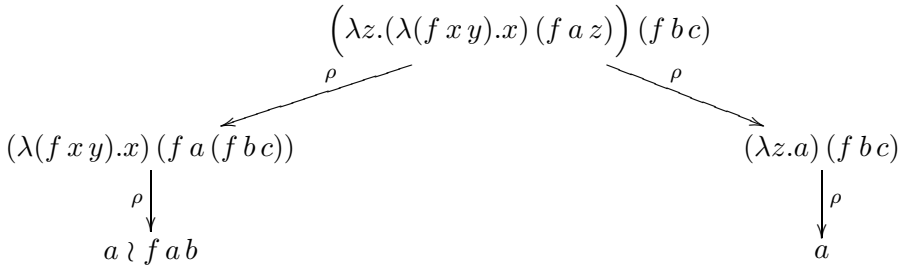
Sauf mention explicite du contraire, à partir de la section 3.3, nous utiliserons donc uniquement des motifs algébriques linéaires et le calcul sera confluent.

Autres théories de filtrage

La discussion du paragraphe précédent est valable dans le cas du filtrage syntaxique ; il est assez facile de voir que même une théorie de filtrage très naturelle peut invalider la confluence du calcul. Prenons par exemple un symbole f associatif :

$$f(x, f(y, z)) =_{\mathbb{T}} f(f(x, y), z)$$

On a alors les réductions suivantes :



La non-confluence vient ici d'un « oubli » dans la réduction de droite : comme le radical interne est réduit avant instanciation de z, il n'est pas possible de réarranger l'argument $f\,a\,z$ pour trouver les deux solutions du problème de filtrage modulo l'associativité qui apparaît dans la réduction de gauche. C'est donc cette dernière qu'il faudrait considérer comme la bonne réduction ; de fait, nous allons même voir que, pour obtenir un calcul confluent dans les théories de filtrage non syntaxiques en général, il faut fixer une *stratégie d'évaluation*, autrement dit un ordre bien précis de choix des radicaux dans un terme donné.

Stratégies de réduction

Nous avons vu qu'une façon d'assurer la confluence est de restreindre l'ensemble des motifs acceptés. Cependant, dans tous les contre-exemples que nous avons vu, un des chemins de réduction semblait préférable aux autres. On peut donc plutôt choisir d'imposer des conditions supplémentaires pour l'application de la règle $\rightarrow_\rho$.

Définition 29 (Stratégie de réduction dans le ρ-calcul) *Une* stratégie de réduction *dans le ρ-calcul est la donnée d'un prédicat $\mathcal{C}$ sur les ρ-termes, et on modifie alors la règle $\rightarrow_\rho$ comme suit :*

$$(\lambda P.A)\, B \quad \rightarrow_\rho \quad A\theta_1 \wr \ldots \wr A\theta_n \qquad \begin{array}{l} si\ \ \mathcal{S}ol(P \ll_{\mathbb{T}} B) = \{\theta_1, \ldots, \theta_n\}\ avec\ n > 0 \\ et\ si\ B\ v\acute{e}rifie\ \mathcal{C} \end{array}$$

La condition $\mathcal{C}$ est souvent décrite par l'appartenance à un ensemble de termes.

Par exemple, on peut autoriser des variables à gauche de l'application dans les motifs tout en conservant un algorithme de filtrage très simple, à condition de s'assurer que ces variables ne peuvent être instanciées que par des constantes. La forme des arguments autorisés rappelle alors le λ-calcul par valeurs de G. Plotkin [Plo75], adapté pour tenir compte du filtrage. Une étude poussée de la confluence avec ce genre de motifs a été faite par B. Jay [Jay04].

Définition 30 (Valeurs dans le ρ-calcul) *L'ensemble des valeurs $\mathcal{V}$ est défini par*

$$\mathcal{V} \quad ::= \quad \mathcal{K} \mid \lambda\mathcal{P}.\mathcal{T}_\rho \mid \mathcal{K}\,\overline{\mathcal{V}} \mid \mathcal{V} \wr \mathcal{V}$$

Plutôt que de s'intéresser à la forme des arguments, on peut tout simplement demander qu'ils ne soient pas susceptibles d'évoluer.

Définition 31 (Valeurs rigides)

Un ρ-terme A est une valeur rigide *s'il est clos ($\mathcal{FV}(A) = \emptyset$) et en forme normale selon $\mapsto_{\rho\delta}$.*

On écarte ainsi le problème vu dans le cas d'un symbole associatif, où l'instanciation d'une variable est susceptible de produire plus de solutions. Le résultat se généralise d'ailleurs pour toute théorie de filtrage.

Théorème 17 (Confluence par valeurs rigides [Cir00, Hou05])

Quelle que soit la théorie de filtrage choisie, la relation $\mapsto_{\rho\delta}$ est confluente sous la stratégie d'appel par valeurs rigides.

Par la suite nous préférerons assurer la confluence par RPC, d'une part parce que nous utiliserons essentiellement un filtrage syntaxique, d'autre part pour pouvoir facilement raisonner sur la forme des motifs. Une partie de nos résultats restent valables quelle que soit la méthode retenue pour assurer la confluence, d'autres demanderaient d'adapter la démonstration, ce que nous signalerons lorsqu'il faudrait faire des changements importants.

3.3 Expressivité

Un premier constat, facile à faire mais fondamental, est que le ρ-calcul avec filtrage syntaxique est une extension du λ-calcul :

Théorème 18 (Simulation du λ-calcul en ρ-calcul [Cir00])
Soit ρ_λ la classe des ρ-termes sans structures et dont tous les motifs sont des variables.

1. *La traduction des termes de ρ_λ en λ-termes est triviale (ce sont déjà des λ-termes).*
2. *La classe ρ_λ est stable par $\mapsto_\rho$ et ne contient que des formes δ-normales.*
3. *Pour tous $A, A' \in \rho_\lambda$, on a $A \mapsto_\rho A' \Leftrightarrow A \mapsto_\beta A'$.*

Cette propriété assure immédiatement que le ρ-calcul possède au minimum l'expressivité des machines de Turing, autrement dit qu'on peut y représenter toutes les fonctions calculables. Par la suite, lorsque nous définirons des systèmes de types pour le ρ-calcul, nous montrerons qu'ils constituent également une extension conservative des λ-calculs typés correspondants.

Le lien avec la réécriture est un peu moins immédiat. En effet, on peut représenter une règle $l \to r$ par l'abstraction $\lambda l.r$, mais contrairement à la réécriture, une «règle» est consommée par la ρ-réduction correspondante. Par exemple, dans le système de réécriture composé de l'unique règle $f(x) \to x$, on a la réduction

$$f(f(f(a))) \to_{\mathcal{R}} f(f(a)) \to_{\mathcal{R}} f(a) \to_{\mathcal{R}} a$$

alors qu'en calcul de réécriture on a

$$(\lambda(f\,x).x)\,(f\,(f\,(f\,a))) \mapsto_\rho f\,(f\,a)$$

et $f\,(f\,a)$ est une forme normale. Pour obtenir a, il aurait fallu appliquer trois instances de $\lambda(f\,x).x$ au lieu d'une. Ceci explique qu'on peut autoriser des membres gauches réduits à une variable sans que cela entraîne immédiatement des réductions non terminantes :

$$(\lambda x.(f\,x))\,a \mapsto_\rho f\,a$$

alors qu'avec une règle de réécriture $x \to f(x)$ on aurait

$$a \to_{\mathcal{R}} f(a) \to_{\mathcal{R}} f(f(a)) \to_{\mathcal{R}} f(f(f(a))) \mapsto_\rho \ldots$$

La condition de régularité $\mathcal{FV}(r) \subseteq \mathcal{FV}(l)$ n'est plus nécessaire non plus, dans la mesure où les variables libres de r sont susceptibles d'être liées par un autre λ et instanciées par d'autres réductions, par exemple

$$\big(\lambda(f\,x).\lambda(g\,y).(h\,x\,y)\big)\,(f\,a)\,(g\,b) \;\mapsto_\rho\; \big(\lambda(g\,y).(h\,a\,y)\big)\,(g\,b) \;\mapsto_\rho\; h\,a\,b$$

Les règles d'évaluation en réécriture et en ρ-calcul sont donc sensiblement différentes ; cependant, on peut établir des correspondances assez fortes. Reprenons la traduction $[\![\ ]\!]$ des termes algébriques donnée au début de ce chapitre et voyons les aspects de la réécriture que l'on peut encoder. Dans un premier temps, on peut toujours représenter une réduction si on connaît les règles utilisées.

Théorème 19 (Représentation d'un chemin de réécriture [CK01])
Étant donnés un système de réécriture $\mathcal{R}$ et deux termes algébriques t, t', si $t \twoheadrightarrow_{\mathcal{R}} t'$, alors il existe un ρ-terme A tel que $A\,[\![t]\!] \twoheadrightarrow_\rho [\![t']\!]$.

De façon plus générale, il est possible de simuler tout le comportement de certains systèmes de réécriture :

Théorème 20 (Représentation d'un système de réécriture [CLW03])
Étant donné un système de réécriture $\mathcal{R}$, il existe un ρ-terme A tel que pour tous termes algébriques t, t' :

1. *Si $A\,[\![t]\!] \mapsto\!\!\!\!\rightarrow_{\beta\delta} [\![t']\!]$, alors $t \twoheadrightarrow_{\mathcal{R}} t'$.*
2. *Si $\mathcal{R}$ est convergent et si $t \twoheadrightarrow_{\mathcal{R}} t'$, alors $A\,[\![t]\!] \mapsto\!\!\!\!\rightarrow_{\beta\delta} [\![t']\!]$.*

Enfin, on peut même préciser certaines stratégies d'évaluation dans la façon dont on simule le système de réécriture.

Théorème 21 (Représentation d'un système de calcul [CLW03, CKLW03])
Étant donné un système de réécriture $\mathcal{R}$ assorti d'une stratégie S suffisamment simple, il existe un ρ-terme A tel que pour tous termes algébriques t, t' :

1. *Si $A\,[\![t]\!] \mapsto\!\!\!\!\rightarrow_{\beta\delta} [\![t']\!]$, alors $t \twoheadrightarrow_{\mathcal{R}} t'$ suivant S.*
2. *Si $t \twoheadrightarrow_{\mathcal{R}} t'$ suivant S, alors $A\,[\![t]\!] \mapsto\!\!\!\!\rightarrow_{\beta\delta} [\![t']\!]$.*

Nous ne justifions pas cette représentation de la réécriture pour l'instant ; nous y reviendrons plus en détails dans la section 4.2.2, où nous expliciterons la classe de stratégies que l'on peut représenter.

Autres travaux Pour conclure ce chapitre, énumérons quelques directions de recherche autour du calcul de réécriture qui ne seront *pas* traitées dans ce manuscrit.

1. La relation avec la réécriture d'ordre supérieur, et plus particulièrement avec les *Combinatory Reduction Systems* [KOR93] a été étudiée par C. Bertolissi, H. Cirstea et C. Kirchner [BCK05].
2. Un mécanisme d'exceptions assurant la confluence a été introduit par G. Faure et C. Kirchner pour une version antérieure du calcul [FK02].
3. Une extension du calcul en direction de la réécriture de graphes a été proposée par P. Baldan, C. Bertolissi, H. Cirstea et C. Kirchner [BBCK05].
4. La sémantique dénotationnelle du calcul est étudiée par G. Faure et A. Miquel [FM05].
5. Plusieurs interpréteurs sont en cours d'implantation : celui de L. Liquori et B. Serpette est conçu en Scheme de façon certifiée [LS04], et celui de G. Faure [Fau05] utilise un filtrage explicite étudié dans [CFK05].

4

Types pour la programmation

Dans ce chapitre, nous allons étudier une première série de systèmes de types pour le calcul de réécriture, directement inspirés du λ-calcul simplement typé. Nous commencerons par les systèmes dits *à la Church* (avec annotations de types) puis nous verrons les systèmes *à la Curry*, dans lesquels les types des variables liées doivent être inférés. Nous verrons que la plupart des propriétés d'un calcul typé sont vérifiées à l'exception de la normalisation forte, c'est pourquoi nous utiliserons ces types de préférence lorsque nous représenterons des programmes en ρ-calcul. Nous montrerons comment exprimer une forme de récursion contrôlée, ce qui nous permettra notamment de représenter les systèmes de réécriture comme annoncé au chapitre précédent. Nous conclurons ce chapitre par une étude de ces systèmes de types du point de vue logique. Les travaux présentés ici ont été publiés dans [CKLW03, CLW03, LW05].

4.1 Types simples et polymorphes

4.1.1 Systèmes de types

Le premier système de types que nous allons définir est l'adaptation la plus immédiate possible des types simples du λ-calcul (voir déf. 4) au ρ-calcul. Le langage de types utilisé est toujours

$$\mathcal{T}^* \quad ::= \quad \mathcal{K}^* \mid \mathcal{T}^* \to \mathcal{T}^*$$

et un contexte Γ associe aux variables de son domaine un unique type simple. De plus, le typage est paramétré par une signature Σ qui à chaque constante de $\mathcal{K}$ associe un unique type simple (ce qui est plus précis que les arités utilisées pour les termes algébriques). Enfin, en λ-calcul, à chaque λ-abstraction est associée le type de la variable. Comme on a ici un motif, il faut associer à toute abstraction un contexte Δ donnant les types des variables sur lesquelles on abstrait. On modifie donc la syntaxe des termes comme suit :

$$\mathcal{T}_{\rho 1} \quad ::= \quad \mathcal{X} \mid \mathcal{K} \mid \lambda \mathcal{P}{:}\Delta.\mathcal{T}_{\rho 1} \mid \mathcal{T}_{\rho 1}\,\mathcal{T}_{\rho 1} \mid \mathcal{T}_{\rho 1} \wr \mathcal{T}_{\rho 1}$$

Exemple 23 (Contexte de typage d'un motif)
Dans le ρ-terme $\lambda(f\,x\,y).(g\,x\,y)$, les variables x et y sont liées.
Une version typée de ce terme s'écrit donc $\lambda(f\,x\,y){:}(x{:}\tau, y{:}\sigma).(g\,x\,y)$.

Les règles de typage sont alors celles de la figure 4.1.

La règle pour l'abstraction ne se contente plus de copier le type de la variable abstraite à gauche de la flèche, elle doit calculer un type pour le motif P et charger tout Δ dans le contexte.

$$(\text{VAR})\ \frac{}{\Gamma, x{:}\tau \vdash_\Sigma x : \tau} \qquad (\text{CONST})\ \frac{(f{:}\tau) \in \Sigma}{\Gamma \vdash_\Sigma f : \tau} \qquad (\text{STRUCT})\ \frac{\Gamma \vdash_\Sigma A : \tau \qquad \Gamma \vdash_\Sigma B : \tau}{\Gamma \vdash_\Sigma A \wr B : \tau}$$

$$(\text{ABS})\ \frac{\Gamma, \Delta \vdash_\Sigma P : \sigma \qquad \Gamma, \Delta \vdash_\Sigma A : \tau}{\Gamma \vdash_\Sigma \lambda P{:}\Delta.A : \sigma \to \tau}\ (\mathcal{D}om(\Delta) = \mathcal{FV}(P))$$

$$(\text{APP})\ \frac{\Gamma \vdash_\Sigma A : \sigma \to \tau \qquad \Gamma \vdash_\Sigma B : \sigma}{\Gamma \vdash_\Sigma A\,B : \tau}$$

FIG. 4.1 – Système de types simples $\rho 1$ pour le calcul de réécriture

La règle (STRUCT) impose que tous les membres d'une même structure aient le même type ; on pourrait étudier une règle plus générale inspirée de la règle $(\wedge I)$ de la déduction naturelle :

$$(\text{STRUCT'})\ \frac{\Gamma \vdash_\Sigma A : \sigma \qquad \Gamma \vdash_\Sigma B : \tau}{\Gamma \vdash_\Sigma A \wr B : \sigma \wedge \tau}$$

en lien avec une notion de sous-typage pour éliminer les $\wedge$. Cette notion de «type conjonction» ne doit pas être confondue avec les types intersections définis par H. Barendregt, M. Coppo et M. Dezani [BCDC83], dans lesquels σ et τ doivent être deux types admissibles pour un *même* terme. Une notion basique de type conjonction pour le ρ-calcul a été étudiée dans [LS04]. Dans ce manuscrit, nous nous restreindrons à la règle (STRUCT).

Dans un second temps, on peut étendre ce système avec des types polymorphes, qui comportent des variables de type. Ils correspondent aux types du système $\lambda 2$ du λ-cube, mais on peut les formuler sans utiliser la Π-abstraction, à condition d'ajouter un lieur dans le langage de types :

$$\mathcal{T}^*_\forall \quad ::= \quad \mathcal{X}^* \mid \mathcal{K}^* \mid \mathcal{T}^*_\forall \to \mathcal{T}^*_\forall \mid \forall \mathcal{X}^*.\mathcal{T}^*_\forall$$

où $\mathcal{X}^*$ est un ensemble (infini dénombrable) de variables de types qu'on notera α ou β. On a maintenant une notion de variable *de type* libre ou liée (qu'on notera $\mathcal{FV}^*$ et $\mathcal{BV}^*$), qui s'étend aux contextes.

$$\begin{array}{rcl@{\qquad}rcl}
\mathcal{FV}^*(\iota) & = & \emptyset & \mathcal{BV}^*(\iota) & = & \emptyset \\
\mathcal{FV}^*(\sigma \to \tau) & = & \mathcal{FV}^*(\sigma) \cup \mathcal{FV}^*(\tau) & \mathcal{BV}^*(\sigma \to \tau) & = & \mathcal{BV}^*(\sigma) \cup \mathcal{BV}^*(\tau) \\
\mathcal{FV}^*(\alpha) & = & \{\alpha\} & \mathcal{BV}^*(\alpha) & = & \emptyset \\
\mathcal{FV}^*(\forall \alpha.\tau) & = & \mathcal{FV}^*(\tau) \setminus \{\alpha\} & \mathcal{BV}^*(\forall \alpha.\tau) & = & \mathcal{BV}^*(\tau) \cup \{\alpha\} \\
\mathcal{FV}^*(\Gamma, x{:}\tau) & = & \mathcal{FV}^*(\Gamma) \cup \mathcal{FV}^*(\tau) & \mathcal{BV}^*(\Gamma, x{:}\tau) & = & \mathcal{BV}^*(\Gamma) \cup \mathcal{BV}^*(\tau)
\end{array}$$

La grammaire des termes est également étendue avec la possibilité d'abstraire sur des variables de type et de passer des types en argument. Les motifs peuvent également comporter des types.

$$\begin{array}{rcl}
\mathcal{Q}_\forall & ::= & \mathcal{K} \mid \mathcal{Q}_\forall\, \mathcal{P}_\forall \mid \mathcal{Q}_\forall\, \mathcal{T}^*_\forall \\
\mathcal{P}_\forall & ::= & \mathcal{X} \mid \mathcal{Q}_\forall \\
\mathcal{T}_{\rho_\forall} & ::= & \mathcal{X} \mid \mathcal{K} \mid \lambda \mathcal{P}_\forall{:}\Delta.\mathcal{T}_{\rho_\forall} \mid \mathcal{T}_{\rho_\forall}\, \mathcal{T}_{\rho_\forall} \mid \mathcal{T}_{\rho_\forall} \wr \mathcal{T}_{\rho_\forall} \mid \lambda \mathcal{X}^*.\mathcal{T}_{\rho_\forall} \mid \mathcal{T}_{\rho_\forall}\, \mathcal{T}^*_\forall
\end{array}$$

Les variables libres et liées des termes sont étendues en conséquence ; on distingue les variables de type des variables de terme.

$$\begin{array}{rcl} \mathcal{FV}(x) & = & \{x\} \\ \mathcal{FV}(a) & = & \emptyset \\ \mathcal{FV}(A\,B) & = & \mathcal{FV}(A) \cup \mathcal{FV}(B) \\ \mathcal{FV}(\lambda P.A) & = & \mathcal{FV}(A) \setminus \mathcal{FV}(P) \\ \mathcal{FV}(A \wr B) & = & \mathcal{FV}(A) \cup \mathcal{FV}(B) \\ \mathcal{FV}(\lambda\alpha.A) & = & \mathcal{FV}(A) \\ \mathcal{FV}(A\,\tau) & = & \mathcal{FV}(A) \end{array} \qquad \begin{array}{rcl} \mathcal{FV}^*(x) & = & \emptyset \\ \mathcal{FV}^*(a) & = & \emptyset \\ \mathcal{FV}^*(A\,B) & = & \mathcal{FV}^*(A) \cup \mathcal{FV}^*(B) \\ \mathcal{FV}^*(\lambda P.A) & = & \mathcal{FV}^*(A) \cup \mathcal{FV}^*(P) \\ \mathcal{FV}^*(A \wr B) & = & \mathcal{FV}^*(A) \cup \mathcal{FV}^*(B) \\ \mathcal{FV}^*(\lambda\alpha.A) & = & \mathcal{FV}^*(A) \setminus \{\alpha\} \\ \mathcal{FV}^*(A\,\tau) & = & \mathcal{FV}^*(A) \cup \mathcal{FV}^*(\tau) \end{array}$$

$$\begin{array}{rcl} \mathcal{BV}(x) & = & \emptyset \\ \mathcal{BV}(a) & = & \emptyset \\ \mathcal{BV}(A\,B) & = & \mathcal{BV}(A) \cup \mathcal{BV}(B) \\ \mathcal{BV}(\lambda P.A) & = & \mathcal{BV}(A) \cup \mathcal{BV}(P) \cup \mathcal{FV}(P) \\ \mathcal{BV}(A \wr B) & = & \mathcal{BV}(A) \cup \mathcal{BV}(B) \\ \mathcal{BV}(\lambda\alpha.A) & = & \mathcal{BV}(A) \\ \mathcal{BV}(A\,\tau) & = & \mathcal{BV}(A) \end{array} \qquad \begin{array}{rcl} \mathcal{BV}^*(x) & = & \emptyset \\ \mathcal{BV}^*(a) & = & \emptyset \\ \mathcal{BV}^*(A\,B) & = & \mathcal{BV}^*(A) \cup \mathcal{BV}^*(B) \\ \mathcal{BV}^*(\lambda P.A) & = & \mathcal{BV}^*(A) \cup \mathcal{BV}^*(P) \\ \mathcal{BV}^*(A \wr B) & = & \mathcal{BV}^*(A) \cup \mathcal{BV}^*(B) \\ \mathcal{BV}^*(\lambda\alpha.A) & = & \mathcal{BV}^*(A) \cup \{\alpha\} \\ \mathcal{BV}^*(A\,\tau) & = & \mathcal{BV}^*(A) \cup \mathcal{BV}^*(\tau) \end{array}$$

Notons que dans une abstraction $\lambda P.A$ (où P est un motif différent de α), les variables de type apparaissant dans P ne sont pas liées par l'abstraction. Seule la λ-abstraction sur les variables de types est susceptible d'instancier ces variables.

$$\begin{array}{rcll} (\lambda P.A)\,B & \to_\rho & A\theta_1 \wr \ldots \wr A\theta_n & \text{si } \mathcal{S}ol(P \ll_{\mathbb{T}} B) = \{\theta_1, \ldots, \theta_n\} \text{ avec } n > 0 \\ & & & \text{et } \forall i, \mathcal{D}om(\theta_i) = \mathcal{FV}(P) \\ (\lambda\alpha.A)\tau & \to_\rho & A[\alpha := \tau] & \\ (A_1 \wr A_2)\,B & \to_\wr & A_1\,B \wr A_2\,B & \end{array}$$

Enfin, aux règles de typage de $\rho 1$ s'ajoutent celles de la figure 4.2 pour ces deux nouvelles constructions.

$$(\text{ABSTYPE})\ \frac{\Gamma \vdash_\Sigma A : \tau}{\Gamma \vdash_\Sigma \lambda\alpha.A : \forall\alpha.\tau}\ (\alpha \notin \mathcal{FV}^*(\Gamma)) \qquad (\text{APPTYPE})\ \frac{\Gamma \vdash_\Sigma A : \forall\alpha.\tau}{\Gamma \vdash_\Sigma A\,\sigma : \tau[\alpha := \sigma]}$$

FIG. 4.2 – Règles additionnelles pour le ρ-calcul polymorphe $\rho_\forall$

Exemple 24 (Quelques fonctions polymorphes)

- *L'exemple le plus simple est l'identité polymorphe :*

$$\vdash_\Sigma \lambda\alpha.\lambda x{:}\alpha.x : \forall\alpha.(\alpha \to \alpha)$$

Pour tout A de type τ, on a donc

$$(\lambda\alpha.\lambda x{:}\alpha.x)\,\tau\,A \mapsto_\rho (\lambda x{:}\tau.x)\,A \ \mapsto_\rho\ A$$

- *On peut définir des constructeurs polymorphes, par exemple*

$$cons : \forall\alpha.(\alpha \to list \to list)$$

En utilisant le filtrage, on peut alors définir une fonction qui récupère la tête d'une liste quel que soit le type des éléments :

$$\vdash_\Sigma \lambda\beta.\lambda(cons\,\beta\,x\,l){:}(x{:}\beta, l{:}list).x : \forall\beta.(list \to \beta)$$

Donnons enfin une variante de ce système polymorphe. Dans le contexte des langages de programmation fonctionnelle, le polymorphisme est souhaitable dans la mesure où il permet de définir des fonctions très générales. Cependant, le fait de devoir manipuler des types explicitement peut rendre un tel langage inutilisable en pratique. On peut donc considérer une version dans le style *à la Curry*, où les types n'apparaissent pas du tout au niveau des termes (pour le λ-calcul le polymorphisme à la Curry a été initialement introduit par R. Milner [Mil78]). Dans ce cadre, il n'est plus nécessaire de raffiner la réduction $\mapsto_\rho$ puisque seules des variables de termes apparaissent dans les motifs. On retrouve ainsi le langage de termes non typés de la section 3.1 :

$$\mathcal{T}_{\rho_C} \quad ::= \quad \mathcal{X} \mid \mathcal{K} \mid \lambda\mathcal{P}.\mathcal{T}_{\rho_C} \mid \mathcal{T}_{\rho_C}\,\mathcal{T}_{\rho_C} \mid \mathcal{T}_{\rho_C} \wr \mathcal{T}_{\rho_C}$$

Par contre, le langage de types $\mathcal{T}^*_\forall$ n'est plus suffisant. En effet, lorsque le type d'une constante contient des lieurs et que cette constante est appliquée à des arguments, les types de ces arguments ne peuvent pas toujours être inférés à partir du type du terme, ce qui peut mettre en péril la préservation du type.

Exemple 25 (Typage implicite et non-préservation du type)
Avec la signature $\Sigma = \{a : \iota, l : list, cons : \forall\alpha.(\alpha \to list \to list)\}$, pour tout terme typable A et pour tout B de type $list$, on aurait

$$\dfrac{\dfrac{\dfrac{(\text{CONST})\ \overline{\Gamma \vdash_\Sigma cons : \forall\alpha.(\alpha \to list \to list)}}{(\text{APPTYPE})\ \Gamma \vdash_\Sigma cons : \tau \to list \to list} \qquad \vdots}{(\text{APP})\ \Gamma \vdash_\Sigma cons\,A : list \to list} \qquad \vdots}{(\text{APP})\ \Gamma \vdash_\Sigma cons\,A\,B : list}$$

Par conséquent on aurait la dérivation de type

$$\dfrac{\dfrac{\dfrac{\vdots}{x{:}\kappa, y{:}list \vdash_\Sigma cons\,x\,y : list} \qquad (\text{VAR})\ \overline{x : \kappa, y : list \vdash_\Sigma x : \kappa}}{(\text{ABS})\ \vdash_\Sigma \lambda(cons\,x\,y).x : list \to \kappa} \qquad \dfrac{\vdots}{\vdash_\Sigma cons\,a\,l : list}}{(\text{APP})\ \vdash_\Sigma (\lambda(cons\,x\,y).x)\,(cons\,a\,l) : \kappa}$$

Ce terme a donc pour type κ (arbitraire !), or le terme se réduit à a qui a pour type ι.

Pour éviter ce problème, dans le type de chaque constante, il faut faire apparaître toutes les variables de type liées dans le type de retour. Ainsi le type proposé pour *cons* n'est pas admissible car α n'apparaît pas dans *list*. On obtient donc la grammaire de types

$$\mathcal{T}^*_C \quad ::= \quad \mathcal{X}^* \mid \mathcal{K}^*(\overline{\mathcal{T}^*_C}) \mid \mathcal{T}^*_C \to \mathcal{T}^*_C \mid \forall\mathcal{X}^*.\mathcal{T}^*_C$$

Pour caractériser les types corrects parmi ceux-ci, on ajoute un jugement de bonne formation $\overline{\alpha} \vdash_\Sigma \tau$ qui fera partie des hypothèses de la règle de typage des constantes, et on étend le système de types avec les règles d'inférence pour ce nouveau jugement. Les règles additionnelles ou modifiées par rapport à $\rho_\forall$ sont celles de la figure 4.3.

Exemple 26 (Fonctions polymorphes à la Curry)
Reprenons les termes de l'exemple 24.

Formation des types des constantes

$$(\text{CONSTR})\ \frac{}{\overline{\alpha} \vdash_\Sigma \iota(\overline{\alpha})} \qquad (\mathcal{FV})\ \frac{}{\vdash_\Sigma \alpha}$$

$$(\text{GEN})\ \frac{\overline{\alpha}, \beta \vdash_\Sigma \tau}{\overline{\alpha} \vdash_\Sigma \forall\beta.\tau} \qquad (\text{FUNCT})\ \frac{\overline{\alpha} \vdash_\Sigma \tau_2}{\overline{\alpha} \vdash_\Sigma \tau_1 \to \tau_2}$$

Règles modifiées

$$(\text{CONST})\ \frac{\vdash_\Sigma \tau}{\Gamma \vdash_\Sigma f : \tau}\ (f{:}\tau) \in \Sigma$$

$$(\text{ABS})\ \frac{\Gamma, \Delta \vdash_\Sigma P : \sigma \qquad \Gamma, \Delta \vdash_\Sigma A : \tau}{\Gamma \vdash_\Sigma \lambda P.A : \sigma \to \tau}\ (\mathcal{D}om(\Delta) = \mathcal{FV}(P))$$

$$(\text{ABSTYPE})\ \frac{\Gamma \vdash_\Sigma A : \tau}{\Gamma \vdash_\Sigma A : \forall\alpha.\tau}\ (\alpha \notin \mathcal{FV}(\Gamma,\Sigma)) \qquad (\text{APPTYPE})\ \frac{\Gamma \vdash_\Sigma A : \forall\alpha.\tau}{\Gamma \vdash_\Sigma A : \tau[\alpha := \sigma]}$$

FIG. 4.3 – Règles spécifiques au système polymorphe à la Curry $\rho_{\forall C}$

- *L'identité devient naturellement polymorphe :* $\vdash_\Sigma \lambda x.x : \forall\alpha.(\alpha \to \alpha)$ *et il n'est plus nécessaire de préciser le type de l'argument : pour tout* A *(typable) on a* $(\lambda x.x)\, A \mapsto_p A$.
- *La déclaration de cons devient cons :* $\forall\alpha.(\alpha \to list(\alpha) \to list(\alpha))$*, et notre fonction de tête de liste polymorphe devient* $\vdash_\Sigma \lambda(cons\, x\, l).x : \forall\beta.(list(\beta) \to \beta)$.

Pour finir cette section, voyons le genre de théorie de filtrage que nous pourrons utiliser en accord avec les types.

Définition 32 (Substitution bien typée)
Dans un contexte Γ*, une substitution* θ *est dite* bien typée relativement à un contexte Δ *si*

1. $\mathcal{D}om(\theta) = \mathcal{D}om(\Delta)$;
2. *pour tout* $(x{:}\tau) \in \Delta$*, on a* $\Gamma \vdash_\Sigma x\theta : \tau$.

Définition 33 (Théorie bien typée)
Une théorie équationnelle $\mathbb{T}$ *est une théorie de filtrage* bien typée *si, pour tous termes* P *et* A *tels que* $\Gamma, \Delta \vdash_\Sigma P : \tau$ *et* $\Gamma \vdash_\Sigma A : \tau$ *avec* $\mathcal{D}om(\Delta) = \mathcal{FV}(P)$*, toute solution de l'équation de filtrage* $P \ll^?_{\mathbb{T}} A$ *est bien typée relativement à* Δ.

Dans le cadre des ρ-calculs typés, nous ne considérerons que des théories de filtrage bien typées et finitaires (c.-à-d. tout problème de filtrage admet un nombre fini de solutions).

Proposition 1 (Filtrage syntaxique et typage)
La théorie syntaxique est une théorie de filtrage finitaire et bien typée.

Démonstration : On a déjà vu que le filtrage syntaxique était unitaire, et cela reste vrai dans un cadre typé.

Le bon typage se démontre par récurrence sur la structure de P. Traitons le cas de $\rho_{\forall C}$, qui est le plus difficile.

Si le motif est réduit à une variable x alors par hypothèse on a $\Gamma, \Delta \vdash_\Sigma x : \tau$ et $\Gamma \vdash_\Sigma A : \tau$, ce qui assure immédiatement que $\Gamma \vdash_\Sigma x\theta : \tau$ puisque $x\theta = A$.

Si le motif est un certain $f\overline{P}$ alors le terme A est de la forme $f\overline{A}$. D'après les règles de formation des types des constantes, le type de f est de la forme

$$\forall\overline{\alpha_1}.(\tau_1 \to \forall\overline{\alpha_2}.(\tau_2 \to \ldots \forall\overline{\alpha}_n.\iota(\overline{\alpha_1}, \ldots, \overline{\alpha_n}) \ldots))$$

où toutes les variables liées apparaissent dans le type de retour $\iota(\overline{\alpha_1}, \ldots, \overline{\alpha_n})$. Par conséquent, le type τ commun à $f\overline{P}$ et $f\overline{A}$ est de la forme

$$\forall\overline{\alpha_{k+1}}.(\tau_{k+1} \to \ldots \forall\overline{\alpha}_n.\iota(\overline{\sigma_1}, \ldots, \overline{\sigma_k}, \overline{\alpha_{k+1}}, \ldots \overline{\alpha_n}) \ldots))$$

obtenu en instanciant les premiers $\overline{\alpha_i}$ par des types $\overline{\sigma_i}$ et en tronquant les k premiers τ_i qui correspondent aux k sous-termes $\overline{P}$ (resp. $\overline{A}$).

Comme toutes les variables liées apparaissent dans le type de retour de f et comme les variables libres dans Σ ne peuvent être liées par une règle (ABSTYPE), toutes les instanciations de variables de type par la règle (APPTYPE) sont retranscrites dans les types $\overline{\sigma_1}, \ldots, \overline{\sigma_k}$. Une même variable de type est donc nécessairement instanciée par le même type lors du typage de $f\overline{P}$ ou de $f\overline{A}$. Ceci assure immédiatement que, pour tout $i \leq k$, on a $\Gamma, \Delta \vdash P_i : \tau_i[\overline{\alpha := \sigma}]$ et $\Gamma \vdash A_i : \tau_i[\overline{\alpha := \sigma}]$.

Par hypothèse de récurrence, chacune des substitutions $\theta_{P_i \ll A_i}$ est bien typée, et donc la substitution $\theta_{P \ll A}$ (qui s'écrit par exemple comme la composition de toutes ces substitutions) est bien typée.

□

Pour terminer cette section, comparons ces systèmes aux λ-calculs typés correspondants.

Remarque 6 (Conservativité par rapport au λ-calcul)
Pour les trois systèmes $\rho 1$, $\rho_\forall$ et $\rho_{\forall C}$, un λ-terme représenté en ρ-calcul admet un type τ si et seulement s'il admet le même type dans le système correspondant en λ-calcul.

En effet, il n'est pas difficile de voir que les règles (VAR), (APP) *et les différentes versions de* (ABSTYPE) *et* (APPTYPE) *sont identiques à celles du λ-calcul. La seule règle qui diffère est* (ABS), *mais lorsque le motif est une variable, elle devient*

$$\frac{\Gamma, x : \sigma \vdash x : \sigma \qquad \Gamma, x : \sigma \vdash A : \tau}{\Gamma \vdash \lambda x{:}(x{:}\sigma).A : \sigma \to \tau}$$

La prémisse de gauche est trivialement vérifiée par la règle (VAR), *et on a donc bien une règle équivalente à* (ABS) *du λ-calcul, avec le léger abus de notation consistant à remplacer le contexte $(x{:}\sigma)$ par le type σ dans $\lambda x.A$.*

Les autres règles de $\rho 1$, $\rho_\forall$ et $\rho_{\forall C}$ ne sont jamais utilisées pour typer un λ-terme en ρ-calcul.

4.1.2 Premières propriétés

Voyons les propriétés vérifiées par ces systèmes de types, adaptées des λ-calculs correspondants. En premier lieu, voici un lemme technique qui nous sera utile pour les démonstrations suivantes.

Lemme 22 (Génération dans $\rho 1$ et $\rho_\forall$)
Dans $\rho 1$ et $\rho_\forall$, la dernière règle utilisée dans une dérivation de typage ayant pour conclusion $\Gamma \vdash_\Sigma A : \tau$ dépend uniquement de la forme de A :

si $A \equiv x$ *alors la dernière règle utilisée est* (VAR) *et* $(x{:}\tau) \in \Gamma$;

si $A \equiv f$ *alors la dernière règle utilisée est* (CONST) *et* $(f{:}\tau) \in \Sigma$;

si $A \equiv A_1 \wr A_2$ *alors la dernière règle utilisée est* (STRUCT) *avec* $\Gamma \vdash_\Sigma A_1 : \tau$ *et* $\Gamma \vdash_\Sigma A_2 : \tau$;

si $A \equiv \lambda P{:}\Delta.A_1$ *alors la dernière règle utilisée est* (ABS) *et* $\tau \equiv \sigma \to \tau_1$ *avec* $\Gamma, \Delta \vdash_\Sigma P : \sigma$ *et* $\Gamma, \Delta \vdash_\Sigma A_1 : \tau_1$;

si $A \equiv A_1\, A_2$ *alors la dernière règle utilisée est* (APP) *et il existe un type* σ *tel que* $\Gamma \vdash_\Sigma A_1 : \sigma \to \tau$ *et* $\Gamma \vdash_\Sigma A_1 : \sigma$;

si $A \equiv \lambda\alpha.A_1$ *alors la dernière règle utilisée est* (ABSTYPE) *et* $\tau \equiv \forall\alpha.\tau_1$ *avec* $\Gamma \vdash_\Sigma A_1 : \tau_1$ *et* $\alpha \notin \mathcal{FV}(\Gamma)$;

si $A \equiv A_1\, \sigma$ *alors la dernière règle utilisée est* (APPTYPE) *et* $\tau \equiv \tau_1[\alpha := \sigma]$ *avec* $\Gamma \vdash_\Sigma A_1 : \forall\alpha.\tau_1$.

Lemme 23 (Génération dans $\rho_{\forall C}$)
Dans $\rho_{\forall C}$, toute dérivation de typage ayant pour conclusion $\Gamma \vdash_\Sigma A : \tau$ se termine par un certain nombre de règles (ABSTYPE) *et* (APPTYPE) *précédées d'une règle dépendant uniquement de la forme de A, et il existe un type τ', des variables de type $\overline{\alpha}$ et une substitution ϑ sur des variables de type tels que $\tau = \forall\overline{\alpha}.(\tau'\vartheta)$ et :*

si $A \equiv x$ *alors la dernière règle utilisée est* (VAR) *avec* $(x{:}\tau') \in \Gamma$;

si $A \equiv f$ *alors la dernière règle utilisée est* (CONST) *avec* $(f{:}\tau') \in \Sigma$;

si $A \equiv A_1 \wr A_2$ *alors la dernière règle utilisée est* (STRUCT) *avec* $\Gamma \vdash_\Sigma A_1 : \tau'$ *et* $\Gamma \vdash_\Sigma A_2 : \tau'$;

si $A \equiv \lambda P.A_1$ *alors la dernière règle utilisée est* (ABS) *et il existe un contexte* Δ *tel que* $\tau' \equiv \sigma \to \tau_1$ *avec* $\Gamma, \Delta \vdash_\Sigma P : \sigma$ *et* $\Gamma, \Delta \vdash_\Sigma A_1 : \tau_1$;

si $A \equiv A_1\, A_2$ *alors la dernière règle utilisée est* (APP) *et il existe un type* σ *tel que* $\Gamma \vdash_\Sigma A_1 : \sigma \to \tau'$ *et* $\Gamma \vdash_\Sigma A_1 : \sigma$.

Démonstration : Par simple inspection des règles de typage. □

Un autre lemme technique essentiel est la clôture du typage par substitution :

Lemme 24 (Substitution)
Pour tous contextes Γ, Δ, pour tout terme A et tout type τ, on a :

1. *Si θ est une substitution bien typée relativement à Δ, si $\Gamma, \Delta \vdash_\Sigma A : \tau$, alors $\Gamma \vdash_\Sigma A\theta : \tau$.*
2. *Si $\Gamma \vdash_\Sigma A : \tau$, alors pour tout type σ on a $\Gamma[\alpha := \sigma] \vdash_\Sigma A[\alpha := \sigma] : \tau[\alpha := \sigma]$.*

Démonstration : Par récurrence sur la structure de A dans les deux cas. □

On peut maintenant démontrer une première propriété fondamentale : la préservation du type, qui assure qu'un objet ne change pas de type au cours de ses réductions.

Théorème 25 (Préservation du type [CLW03])
Si $\Gamma \vdash_\Sigma A : \tau$ et si $A \mapsto_{\beta\delta} A'$ alors $\Gamma \vdash_\Sigma A' : \tau$.

Nous démontrons ce théorème directement dans $\rho_\forall$ puisque toute dérivation dans $\rho 1$ est une dérivation dans $\rho_\forall$; le seul point délicat est de s'assurer qu'on n'utilise pas les règles de $\rho_\forall$ dans les cas qui peuvent concerner $\rho 1$.

Le cas du typage polymorphe à la Curry est très similaire, à ceci près qu'il faut à chaque fois gérer les variables liées $\overline{\alpha}$ et la substitution de types ϑ du lemme de génération.

Démonstration : Dans un premier temps, procédons par récurrence sur la structure de A pour déterminer où a lieu la réduction.

Si $A \equiv \lambda P{:}\Delta.A_1$ **avec** $A_1 \mapsto_{\not\delta} A_1'$ alors $\tau \equiv \sigma \to \tau_1$ avec $\Gamma, \Delta \vdash_\Sigma P : \sigma$ et $\Gamma, \Delta \vdash_\Sigma A_1 : \tau_1$.
Par hypothèse de récurrence sur A_1, on a $\Gamma, \Delta \vdash_\Sigma A_1' : \tau_1$, d'où

$$(\text{ABS})\ \frac{\Gamma, \Delta \vdash_\Sigma P : \sigma \qquad \Gamma, \Delta \vdash_\Sigma A_1' : \tau_1}{\Gamma \vdash_\Sigma \lambda P{:}\Delta.A_1' : \sigma \to \tau_1}$$

Si $A \equiv A_1 \wr A_2$ **avec** $A_1 \mapsto_{\not\delta} A_1'$ **(resp.** $A_2 \mapsto_{\not\delta} A_2'$**)** alors $\Gamma \vdash_\Sigma A_1 : \tau$ et $\Gamma \vdash_\Sigma A_2 : \tau$.
Par hypothèse de récurrence sur A_1 (resp. A_2) on a $\Gamma \vdash_\Sigma A_1' : \tau$ (resp. $\Gamma \vdash_\Sigma A_2' : \tau$) d'où

$$(\text{STRUCT})\ \frac{\Gamma \vdash_\Sigma A_1' : \tau \qquad \Gamma \vdash_\Sigma A_2 : \tau}{\Gamma \vdash_\Sigma A' : \tau} \quad \text{resp.} \quad (\text{STRUCT})\ \frac{\Gamma \vdash_\Sigma A_1 : \tau \qquad \Gamma \vdash_\Sigma A_2' : \tau}{\Gamma \vdash_\Sigma A' : \tau}$$

Si $A \equiv \lambda\alpha.A_1$ **avec** $A_1 \mapsto_{\not\delta} A_1'$ alors $\tau \equiv \forall\alpha.\tau_1$ avec $\Gamma \vdash_\Sigma A_1 : \tau_1$ et $\alpha \notin \mathcal{FV}(\Gamma)$.
Par hypothèse de récurrence sur A_1, on a $\Gamma \vdash_\Sigma A_1' : \tau_1$, d'où

$$(\text{ABSTYPE})\ \frac{\Gamma \vdash_\Sigma A_1' : \tau_1}{\Gamma \vdash_\Sigma \lambda\alpha.A_1' : \forall\alpha.\tau_1}\ (\alpha \notin \mathcal{FV}(\Gamma))$$

Si $A \equiv A_1\, A_2$ **avec** $A_1 \mapsto_{\not\delta} A_1'$ **(resp.** $A_2 \mapsto_{\not\delta} A_2'$**)** alors $\Gamma \vdash_\Sigma A_1 : \sigma \to \tau$ et $\Gamma \vdash_\Sigma A_2 : \sigma$.
Par hypothèse de récurrence sur A_1 (resp. A_2) on a $\Gamma \vdash_\Sigma A_1' : \sigma \to \tau$ (resp. $\Gamma \vdash_\Sigma A_2' : \sigma$) d'où

$$(\text{APP})\ \frac{\Gamma \vdash_\Sigma A_1' : \sigma \to \tau \qquad \Gamma \vdash_\Sigma A_2 : \sigma}{\Gamma \vdash_\Sigma A' : \tau} \quad \text{resp.} \quad (\text{APP})\ \frac{\Gamma \vdash_\Sigma A_1 : \sigma \to \tau \qquad \Gamma \vdash_\Sigma A_2' : \sigma}{\Gamma \vdash_\Sigma A' : \tau}$$

Si $A \equiv A_1\, \sigma$ **avec** $A_1 \mapsto_{\not\delta} A_1'$ alors $\Gamma \vdash_\Sigma A_1 : \forall\alpha.\tau_1$ et $\tau \equiv \tau_1[\alpha := \sigma]$.
Par hypothèse de récurrence sur A_1 on a $\Gamma \vdash_\Sigma A_1' : \forall\alpha.\tau_1$ d'où

$$(\text{APPTYPE})\ \frac{\Gamma \vdash_\Sigma A_1' : \forall\alpha.\tau_1}{\Gamma \vdash_\Sigma A' : \tau}$$

Si $A \equiv (A_{11} \wr A_{12})\, A_2$ **avec** $A' = A_{11}\, A_2 \wr A_{12}\, A_2$ alors $\Gamma \vdash_\Sigma A_{11} \wr A_{12} : \sigma \to \tau$ et $\Gamma \vdash_\Sigma A_2 : \sigma$, d'où $\Gamma \vdash_\Sigma A_{11} : \sigma \to \tau$ et $\Gamma \vdash_\Sigma A_{12} : \sigma \to \tau$. Par conséquent

$$(\text{STRUCT})\ \frac{(\text{APP})\ \dfrac{\Gamma \vdash_\Sigma A_{11} : \sigma \to \tau \qquad \Gamma \vdash_\Sigma A_2 : \sigma}{\Gamma \vdash_\Sigma A_{11}\, A_2 : \tau} \qquad (\text{APP})\ \dfrac{\Gamma \vdash_\Sigma A_{12} : \sigma \to \tau \qquad \Gamma \vdash_\Sigma A_2 : \sigma}{\Gamma \vdash_\Sigma A_{12}\, A_2 : \tau}}{\Gamma \vdash_\Sigma A' : \tau}$$

Si $A \equiv (\lambda P{:}\Delta.A_1)\, A_2$ **avec** $A' \equiv A_1\theta_1 \wr \ldots \wr A_1\theta_n$ **où** $\{\theta_1, \ldots, \theta_n\} = \mathcal{S}ol(P \ll_{\mathbb{T}} A_2)$ alors $\Gamma, \Delta \vdash_\Sigma P : \sigma$ et $\Gamma, \Delta \vdash_\Sigma A_1 : \tau$ et $\Gamma \vdash_\Sigma A_2 : \sigma$. Par bon typage de la théorie de filtrage et par application du point 1 du lemme de substitution 24, on a $\Gamma \vdash_\Sigma A_1\theta_i : \tau$ pour tout i. Par application de $n-1$ occurrences de la règle (STRUCT), on a $\Gamma \vdash_\Sigma A' : \tau$.

Si $A \equiv (\lambda\alpha.A_1)\,\sigma$ **avec** $A' \equiv A_1[\alpha := \sigma]$ alors $\tau = \tau_1[\alpha := \sigma]$ avec $\Gamma \vdash_\Sigma A_1 : \tau_1$. De plus $\alpha \notin \mathcal{FV}(\Gamma)$ donc $\Gamma[\alpha := \sigma] = \Gamma$. Par application du point 2 du lemme de substitution 24, on a donc $\Gamma \vdash_\Sigma A_1[\alpha := \sigma] : \tau_1[\alpha := \sigma]$.

□

Corollaire 26

Si $\Gamma \vdash_\Sigma A : \tau$ *et si* $A \mapsto\!\!\!\!\!\rightarrow_{\rho\delta} A'$ *alors* $\Gamma \vdash_\Sigma A' : \tau$.

Démonstration : On se ramène au cas d'un unique pas de réduction, par récurrence sur la longueur de $A \mapsto\!\!\!\!\!\rightarrow_{\rho\delta} A'$.

Si $A \equiv A'$ alors trivialement $\Gamma \vdash_\Sigma A' : \tau$.

Si $\exists A'',\ A \mapsto_{\rho\delta} A'' \wedge A'' \mapsto\!\!\!\!\!\rightarrow_{\rho\delta} A'$ alors par préservation du type, $\Gamma \vdash_\Sigma A'' : \tau$. Par hypothèse de récurrence sur $A'' \mapsto\!\!\!\!\!\rightarrow_{\rho\delta} A'$, on a donc également $\Gamma \vdash_\Sigma A' : \tau$.

□

Une seconde propriété importante est l'unicité du type, qui pourra faciliter les questions d'inférence et de vérification.

Théorème 27 (Unicité du type dans $\rho 1$ et $\rho_\forall$ [CLW03, LW05])

Pour tout contexte Γ, *pour tout terme* A, *si* $\Gamma \vdash_\Sigma A : \tau$ *et* $\Gamma \vdash_\Sigma A : \tau'$ *dans* $\rho_\forall$ *(resp.* $\rho 1$*) alors* $\tau \equiv \tau'$.

Démonstration : Par récurrence sur la structure de A en utilisant le lemme de génération. Les cas de base des variables et des constantes sont déterminés de façon unique, respectivement par le contexte Γ et la signature Σ. □

Évidemment, dans le cas du typage à la Curry, il n'y a pas unicité :

Exemple 27 (Non-unicité du type à la Curry)

Comme le type des variables n'est pas précisé, dès qu'il y a une abstraction, on peut choisir une infinité dénombrable de types : pour tout τ, *on a*

$$\dfrac{\dfrac{}{x{:}\tau \vdash_\Sigma x : \tau}\,(\text{VAR})}{\vdash_\Sigma \lambda x.x : \tau \to \tau}\,(\text{ABS})$$

Le mécanisme d'abstraction et d'instanciation implicite montre encore mieux la versatilité du système : pour tout type τ, *on a*

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x{:}\alpha \vdash_\Sigma x : \alpha}\,(\text{VAR})}{\vdash_\Sigma \lambda x.x : \alpha \to \alpha}\,(\text{ABS})}{\vdash_\Sigma \lambda x.x : \forall\alpha.(\alpha \to \alpha)}\,(\text{ABSTYPE})}{\vdash_\Sigma \lambda x.x : \tau \to \tau}\,(\text{APPTYPE})$$

Dans la prochaine section, nous verrons qu'on peut définir une notion de type principal *qui représente tous les types possibles pour un terme donné.*

Enfin on peut s'intéresser à la décidabilité de certains problèmes liés au typage.

Théorème 28 (Décidabilité [CLW03, LW05])
Dans les systèmes $\rho 1$ et $\rho_\forall$, les deux problèmes suivants sont décidables.

1. *Étant donnés un contexte Γ, un terme A et un type τ, peut-on dériver $\Gamma \vdash_\Sigma A : \tau$?*
2. *Étant donnés un contexte Γ et un terme A, existe-t-il un type τ tel que $\Gamma \vdash_\Sigma A : \tau$?*

Dans le système $\rho_{\forall C}$, ces deux problèmes sont indécidables.

Démonstration : Dans le cas de $\rho 1$ et $\rho_\forall$, commençons par montrer une réduction du premier problème au second : pour décider si $\Gamma \vdash_\Sigma A : \tau$, il suffit de chercher un type τ' tel que $\Gamma \vdash_\Sigma A : \tau'$. Si un tel type n'existe pas, alors τ ne peut pas être un type valide pour A. Réciproquement, si τ' existe, alors par unicité, τ est un type valide pour A si et seulement si $\tau \equiv \tau'$.

On peut donc se contenter de donner des algorithmes d'inférence de type, ce que nous ferons dans la prochaine section.

Dans le cas de $\rho_{\forall C}$, on peut utiliser le résultat de J. B. Wells [Wel99] : la vérification et l'inférence de type dans le système $\lambda 2$ à la Curry sont des problèmes indécidables. La réduction de ces problèmes à ceux que l'on considère est quasi-triviale, puisque nous considérons une extension conservative du λ-calcul : un λ-terme est typable dans $\lambda 2$ si et seulement s'il est typable dans $\rho_{\forall C}$ avec filtrage syntaxique. □

4.1.3 Inférence de type

Complétons tout d'abord la démonstration de décidabilité de $\rho 1$ et $\rho_\forall$ esquissée à la section précédente, après quoi nous verrons comment restreindre $\rho_{\forall C}$ pour obtenir un typage décidable. Cette section étend les résultats publiés dans [LW05].

L'algorithme d'inférence de type pour $\rho_\forall$ est donné dans la figure 4.4. Il est également correct pour le typage dans $\rho 1$.

Proposition 2 (Correction et complétude de l'inférence de type dans $\rho_\forall$)
Pour tout terme A et tout contexte Γ :

1. *Si $Type(A;\Gamma) = \tau$ alors $\Gamma \vdash_\Sigma A : \tau$ dans $\rho_\forall$, ainsi que dans $\rho 1$ si A ne comporte pas d'abstraction ni d'application de type.*
2. *Si $Type(A;\Gamma) = \mathsf{false}$ alors A n'est pas typable dans $\rho_\forall$ avec le contexte Γ.*
3. *Si $\Gamma \vdash_\Sigma A : \tau$ dans $\rho 1$ (resp. $\rho_\forall$) alors $Type(A;\Gamma) = \tau$.*
4. *Si A n'est pas typable dans $\rho_\forall$ avec le contexte Γ, alors $Type(A;\Gamma) = \mathsf{false}$.*

Démonstration : Par récurrence sur la structure de A. □

L'indécidabilité du typage dans $\rho_{\forall C}$ est due au même problème que dans le cas du λ-calcul : le langage de types est trop important. En effet, la liaison de variables peut apparaître n'importe où dans un type, ce qui empêche de contrôler l'utilisation des règles de typage (ABSTYPE) et (APPTYPE). Nous adopterons donc la même solution qu'en ML : on se restreindra désormais à des *schémas de type* (notés σ) où les lieurs se trouvent tous en tête :

$$\begin{aligned} \mathcal{T}^*_{\ll} &::= \mathcal{X}^* \mid \mathcal{K}^*(\overline{\mathcal{T}^*_{\ll}}) \mid \mathcal{T}^*_{\ll} \to \mathcal{T}^*_{\ll} \\ \mathfrak{S} &::= \mathcal{T}^*_{\ll} \mid \forall \mathcal{X}^*.\mathfrak{S} \end{aligned}$$

$$
\begin{array}{lll}
Type(A;\Gamma) \triangleq & \text{Discriminer selon la forme de } A : & \\
& A \equiv x & \Rightarrow \tau \\
& & \quad \text{ssi } (x{:}\tau) \in \Gamma \\
& A \equiv f & \Rightarrow \tau \\
& & \quad \text{ssi } (f{:}\tau) \in \Sigma \\
& A \equiv A_1 \wr A_2 & \Rightarrow Type(A_1;\Gamma) \\
& & \quad \text{ssi } Type(A_1;\Gamma) = Type(A_2;\Gamma) \\
& A \equiv \lambda P{:}\Delta.A_1 & \Rightarrow Type(P;\Gamma,\Delta) \to Type(A_1;\Gamma,\Delta) \\
& & \quad \text{ssi } Type(P;\Gamma,\Delta) \neq \mathsf{false} \neq Type(A_1;\Gamma,\Delta) \\
& A \equiv A_1\, A_2 & \Rightarrow \tau_2 \\
& & \quad \text{ssi } Type(A_1;\Gamma) = \tau_1 \to \tau_2 \text{ et } Type(A_2;\Gamma) = \tau_1 \\
& A \equiv \lambda\alpha.A_1 & \Rightarrow \forall\alpha.Type(A_1;\Gamma) \\
& & \quad \text{ssi } Type(A_1;\Gamma) \neq \mathsf{false} \text{ et } \alpha \notin \mathcal{FV}(\Gamma) \\
& A \equiv A_1\,\sigma & \Rightarrow \tau_1[\alpha := \sigma] \\
& & \quad \text{ssi } Type(A_1;\Gamma) = \forall\alpha.\tau_1 \\
& \text{sinon} & \Rightarrow \mathsf{false}
\end{array}
$$

FIG. 4.4 – Algorithme d'inférence de type dans $\rho_\forall$

Les contextes et les signatures associeront désormais un unique schéma de type σ à chaque variable (resp. constante), mais le système de types permettra de dériver uniquement des jugements de la forme $\Gamma \vdash_\Sigma A : \tau$, autrement dit on ne typera les termes qu'avec des types simples. Pour passer d'un schéma à un type dans les règles pour les constantes et les variables, on définit une relation $\leq$ exprimant qu'un type τ est une instance d'un schéma de type σ.

$$\sigma \leq \tau \text{ ssi } \sigma \triangleq \forall\overline{\alpha_i}.\tau' \text{ et } \exists\overline{\tau_i},\ \tau \triangleq \tau'[\overline{\alpha_i := \tau_i}].$$

De plus, pour ne pas perdre totalement l'aspect polymorphe du calcul, on ajoute une construction similaire au let de ML dans le langage de termes : la *contrainte de filtrage* $[P \ll B]A$ permet de lier des variables par un motif P tout en sachant déjà quel sera l'argument B à filtrer selon P. Cette connaissance supplémentaire autorise une forme restreinte de polymorphisme dans la mesure où on peut généraliser certaines variables de type apparaissant dans les types des variables libres de P.

$$\mathcal{T}_\rho^{\ll} \quad ::= \quad \mathcal{X} \mid \mathcal{K} \mid \lambda\mathcal{P}.\mathcal{T}_\rho^{\ll} \mid \mathcal{T}_\rho^{\ll}\,\mathcal{T}_\rho^{\ll} \mid \mathcal{T}_\rho^{\ll} \wr \mathcal{T}_\rho^{\ll} \mid [\mathcal{P} \ll \mathcal{T}_\rho^{\ll}]\mathcal{T}_\rho^{\ll}$$

À cette nouvelle construction on associe une règle de réduction très similaire à $\mapsto_\rho$, et les règles de réduction sont donc :

$$
\begin{array}{rcll}
(\lambda P.A)\,B & \to_\rho & A\theta_1 \wr \ldots \wr A\theta_n & \text{si } \mathcal{S}ol(P \ll_{\mathbb{T}} B) = \{\theta_1,\ldots,\theta_n\} \text{ avec } n > 0 \\
[P \ll B]A & \to_\sigma & A\theta_1 \wr \ldots \wr A\theta_n & \text{si } \mathcal{S}ol(P \ll_{\mathbb{T}} B) = \{\theta_1,\ldots,\theta_n\} \text{ avec } n > 0 \\
(A_1 \wr A_2)\,B & \to_\delta & A_1\,B \wr A_2\,B &
\end{array}
$$

Les règles de typage sont alors celles de la figure 4.5. La règle (MATCH) utilise une fonction auxiliaire $\mathsf{Gen}(\Delta;\Gamma)$ qui transforme les types des variables de Δ en schémas de types, en évitant les variables libres dans Γ pour ne pas créer d'incohérences.

$$(\textsc{var})\ \frac{}{\Gamma, x{:}\sigma \vdash_\Sigma x : \tau}\ \left(\begin{array}{c}\sigma \leq \tau \\ x \notin \mathcal{D}om(\Gamma)\end{array}\right) \qquad (\textsc{const})\ \frac{\emptyset \vdash_\Sigma \sigma}{\Gamma \vdash_\Sigma f : \tau}\ \left(\begin{array}{c}\sigma \leq \tau \\ (f{:}\sigma) \in \Sigma\end{array}\right)$$

$$(\textsc{match})\ \frac{\Gamma \vdash_\Sigma B : \tau_1 \qquad \Gamma, \Delta \vdash_\Sigma P : \tau_1 \qquad \Gamma, \mathsf{Gen}(\Delta; \Gamma) \vdash_\Sigma A : \tau_2}{\Gamma \vdash_\Sigma [P \ll B]A : \tau_2}\ \left(\begin{array}{c}\mathcal{BV}^*(\Delta) = \emptyset \\ \mathcal{D}om(\Delta) = \mathcal{FV}(P)\end{array}\right)$$

$$(\textsc{abs})\ \frac{\Gamma, \Delta \vdash_\Sigma P : \tau_1 \qquad \Gamma, \Delta \vdash_\Sigma A : \tau_2}{\Gamma \vdash_\Sigma \lambda P.A : \tau_1 \to \tau_2}\ \left(\begin{array}{c}\mathcal{BV}^*(\Delta) = \emptyset \\ \mathcal{D}om(\Delta) = \mathcal{FV}(P)\end{array}\right)$$

Les règles (ABSTYPE) et (APPTYPE) ne sont plus valables.

$$\text{où}\ \left\{\begin{array}{rcl}\mathsf{Gen}(\tau; \Gamma) & \triangleq & \forall \overline{\alpha}.\tau \qquad \text{avec } \overline{\alpha} = \mathcal{FV}(\tau) \setminus \mathcal{FV}(\Gamma) \\ \mathsf{Gen}(x{:}\tau, \Delta; \Gamma) & \triangleq & x{:}\mathsf{Gen}(\tau; \Gamma), \mathsf{Gen}(\Delta; \Gamma)\end{array}\right.$$

FIG. 4.5 – Règles de typage spécifiques au fragment décidable du système polymorphe $\rho^{\ll}_{\forall}$

Exemple 28 (Polymorphisme permis par la construction [$\ll$])
Prenons la signature $\Sigma = \{a{:}\iota, b{:}\kappa, f{:}\iota \to \kappa \to \iota\}$. *Le terme* $\big(\lambda x.f\,(x\,a)\,(x\,b)\big)(\lambda y.y)$ *n'est pas typable dans* $\rho^{\ll}_{\forall}$ *car lors du typage la première abstraction, il faut fixer un* type *pour* x, *qui ne peut être à la fois* $\iota \to \iota$ *et* $\kappa \to \kappa$.

En revanche, le terme $[x \ll \lambda y.y]\big(f\,(x\,a)\,(x\,b)\big)$, *qui admet les mêmes réductions, est typable :*

$$(\textsc{match})\ \frac{\vdash_\Sigma \lambda y.y : \alpha \to \alpha \qquad x{:}\alpha \to \alpha \vdash_\Sigma x : \alpha \to \alpha \qquad \dfrac{\vdots}{\mathsf{Gen}(x{:}\alpha \to \alpha; \emptyset) \vdash_\Sigma f\,(x\,a)\,(x\,b) : \iota}}{\Gamma \vdash_\Sigma [x \ll \lambda y.y]\big(f\,(x\,a)\,(x\,b)\big) : \iota}$$

On voit que $\mathsf{Gen}(x{:}\alpha \to \alpha; \emptyset)$ *affecte le* schéma de type $\forall\alpha.(\alpha \to \alpha)$ *à* x, *ce qui permet qu'il soit appliqué à la fois à* a *et à* b *dans* $f\,(x\,a)\,(x\,b)$.

Il n'est pas très difficile de vérifier que ce système vérifie les propriétés vues à la section précédente : génération, substitution, préservation du type. Le typage n'est visiblement toujours pas unique ; cependant, par un léger abus de langage, on dira qu'un schéma de type est *le* type d'un terme s'il représente tous les types qu'on peut lui attribuer :

Définition 34 (Type principal)
Un schéma de type σ *est un* type principal *pour le terme* A *dans le contexte* Γ *si et seulement si :*

1. *Si* $\sigma = \forall\overline{\alpha}.\tau'$ *alors* $\Gamma \vdash_\Sigma A : \tau'$.
2. *Pour tout jugement de typage* $\Gamma \vdash_\Sigma A : \tau$ *dérivable dans* $\rho^{\ll}_{\forall}$, *on a* $\sigma \leq \tau$.

Remarquons deux propriétés immédiates :
– si σ est un type principal pour A, alors $\Gamma \vdash_\Sigma A : \tau$ pour *tout* type τ tel que $\sigma \leq \tau$;
– le type principal d'un terme est unique modulo le renommage de ses variables liées.

Nous sommes maintenant en mesure de présenter un algorithme qui infère un type principal étant donnés un terme A et un contexte Γ. Il démontre donc à la fois la décidabilité du typage et l'existence d'un type principal pour tout terme. Cet algorithme est un raffinement de l'algorithme W de L. Damas et R. Milner [DM82] pour un système polymorphe à la Curry dans le λ-calcul ; cependant, nous suivrons ici la présentation plus moderne adoptée par F. Pottier pour l'inférence de types dans Caml [Pot02].

Notre algorithme est très similaire à celui du langage Caml [Cam04]. La principale différence réside dans le traitement du filtrage, qui chez nous fait partie intégrante de l'abstraction au lieu d'utiliser une construction `match` distincte.

L'algorithme W est donné dans la figure 4.6. Il prend pour arguments :
- le terme à typer A ;
- l'environnement de typage Γ ;
- un ensemble (infini dénombrable) de variables de type fraîches $\mathcal{V}$ dans lesquelles on pourra se servir pour créer de nouveaux types.

Si A est typable, l'algorithme renvoie :
- un type τ dont il suffit d'abstraire les variables libres pour obtenir un schéma de type principal ;
- une valuation θ des variables de types libres dans Γ (qui servira notamment à instancier les variables prises dans $\mathcal{V}$) ;
- le sous-ensemble $\mathcal{V}'$ des variables non encore utilisées dans $\mathcal{V}$.

La fonction Inst (définie en bas de la figure 4.6) transforme un schéma de type en un type dont on connaît les variables libres. La procédure auxiliaire mgu calcule un unificateur plus général ; on peut en trouver une description complète par exemple dans [Rob65]. Notons que l'échec (false) doit être propagé strictement, et doit notamment être renvoyé quand la procédure mgu échoue.

Théorème 29 (Correction de W)
Si $\mathsf{W}(\Gamma; A; \mathcal{V}) = (\tau; \theta; \mathcal{V}')$, *alors* $\Gamma\theta \vdash_\Sigma A : \tau$.

Démonstration : Par récurrence sur la structure de A. Voyons les cas des variables, de l'abstraction, de l'application et de la contrainte de filtrage (les constantes se traitent comme les variables, et les structures comme les applications).

- Si $A \equiv x$ alors $(\tau; \mathcal{V}) = \mathsf{Inst}(\Gamma(x); \mathcal{V})$ et $\theta = \mathcal{I}d$ (la substitution identité). Par définition de Inst on a $\Gamma(x) \leq \tau$ d'où $\Gamma \vdash_\Sigma x : \tau$.
- Si $A \equiv \lambda P.A_1$ alors $\tau = \tau_1\theta_2 \to \tau_2$ et $\theta = \theta_2 \circ \theta_1$ où τ_1, θ_1, τ_2 et θ_2 sont calculés par des appels récursifs sur les sous-termes P et A_1.
  Posons $\Delta = \overline{x : \alpha_x}$. Par hypothèse de récurrence on a $\Gamma\theta_1, \Delta\theta_1 \vdash_\Sigma P : \tau_1$ et $\Gamma\theta, \Delta\theta \vdash_\Sigma A_1 : \tau_2$. Par substitution sur les variables de type (lemme 24.2), on a aussi $\Gamma\theta, \Delta\theta \vdash_\Sigma P : \tau_1\theta_2$, d'où
  $$(\text{ABS})\ \frac{\Gamma\theta, \Delta\theta \vdash_\Sigma P : \tau_1\theta_2 \qquad \Gamma\theta, \Delta\theta \vdash_\Sigma A_1 : \tau_2}{\Gamma\theta \vdash_\Sigma \lambda P.A_1 : \tau_1\theta_2 \to \tau_2}$$
- Si $A \equiv A_1\, A_2$ alors $\tau = \alpha\mu$ et $\theta = \mu \circ \theta_2 \circ \theta_1$ où $\tau_1\theta_2\mu = \tau_2\mu \to \alpha\mu$ et $\tau_1, \theta_1, \tau_2, \theta_2$ sont calculés par des appels récursifs sur les sous-termes A_1 et A_2.
  Par hypothèse de récurrence on a $\Gamma\theta_1 \vdash_\Sigma A_1 : \tau_1$ et $\Gamma\theta_1\theta_2 \vdash_\Sigma A_2 : \tau_2$. Par substitution sur les variables de type, on a aussi $\Gamma\theta \vdash_\Sigma A_1 : \tau_1\theta_2\mu$ et $\Gamma\theta \vdash_\Sigma A_2 : \tau_2\mu$. Étant donné que $\tau_1\theta_2\mu = \tau_2\mu \to \alpha\mu$, on a
  $$(\text{APP})\ \frac{\Gamma\theta \vdash_\Sigma A_1 : \tau_2\mu \to \alpha\mu \qquad \Gamma\theta \vdash_\Sigma A_2 : \tau_2\mu}{\Gamma\theta \vdash_\Sigma A_1\, A_2 : \alpha\mu}$$

$$
\begin{array}{rcl}
\mathsf{W}(\Gamma; A; \mathcal{V}) & \triangleq & \text{Discriminer selon la forme de } A : \\
A \equiv x & \Rightarrow & (\tau\,; \mathcal{I}d\,; \mathcal{V}') \text{ où } (\tau\,; \mathcal{V}') = \mathsf{Inst}(\Gamma(x)\,; \mathcal{V}) \\
 & & \text{ssi } x \in \mathcal{D}om(\Gamma) \\
A \equiv f & \Rightarrow & (\tau\,; \mathcal{I}d\,; \mathcal{V}') \text{ où } (\tau\,; \mathcal{V}') = \mathsf{Inst}(\Gamma(f)\,; \mathcal{V}) \\
 & & \text{ssi } f \in \mathcal{D}om(\Gamma) \\
A \equiv A_1 \wr A_2 & \Rightarrow & (\tau; \theta; \mathcal{V}') = (\tau_2\mu\,; \mu \circ \theta_2 \circ \theta_1\,; \mathcal{V}_2) \\
 & & \text{avec } (\tau_1; \theta_1; \mathcal{V}_1) = \mathsf{W}(\Gamma; A_1; \mathcal{V}) \\
 & & \quad (\tau_2; \theta_2; \mathcal{V}_2) = \mathsf{W}(\Gamma\theta_1; A_2; \mathcal{V}_1) \\
 & & \text{et } \mu = \mathsf{mgu}(\tau_1\theta_2 = \tau_2) \\
A \equiv \lambda P.A_1 & \Rightarrow & (\tau; \theta; \mathcal{V}') = (\tau_1\theta_2 \to \tau_2; \theta_2 \circ \theta_1; \mathcal{V}_2) \\
 & & \text{avec } \overline{x} = \mathcal{FV}(P) \text{ et } \overline{\alpha_x} \in \mathcal{V}_1 \\
 & & \quad (\tau_1; \theta_1; \mathcal{V}_1) = \mathsf{W}(\Gamma, \overline{x{:}\alpha_x}\,; P\,; \mathcal{V} \setminus \{\overline{\alpha_x}\}) \\
 & & \text{et } (\tau_2; \theta_2; \mathcal{V}_2) = \mathsf{W}(\Gamma\theta_1, \overline{x{:}\alpha_x\theta_1}\,; A_1\,; \mathcal{V}_1) \\
A \equiv A_1\, A_2 & \Rightarrow & (\tau; \theta; \mathcal{V}') = (\alpha\mu\,; \mu \circ \theta_2 \circ \theta_1\,; \mathcal{V}_2 \setminus \{\alpha\}) \\
 & & \text{avec } (\tau_1; \theta_1; \mathcal{V}_1) = \mathsf{W}(\Gamma; A_1\,; \mathcal{V}) \\
 & & \quad (\tau_2; \theta_2; \mathcal{V}_2) = \mathsf{W}(\Gamma\theta_1\,; A_2\,; \mathcal{V}_1) \\
 & & \quad \alpha \in \mathcal{V}_2 \\
 & & \text{et } \mu = \mathsf{mgu}(\tau_1\theta_2 = \tau_2 \to \alpha) \\
A \equiv [P \ll A_2]A_1 & \Rightarrow & (\tau; \theta; \mathcal{V}') = (\tau_3\,; \theta_3 \circ \mu \circ \theta_2 \circ \theta_1\,; \mathcal{V}_3) \\
 & & \text{avec } (\tau_1; \theta_1; \mathcal{V}_1) = \mathsf{W}(\Gamma; A_2\,; \mathcal{V}) \\
 & & \quad \overline{x} = \mathcal{FV}(P) \text{ et } \overline{\alpha_x} \in \mathcal{V}_1 \\
 & & \quad (\tau_2; \theta_2; \mathcal{V}_2) = \mathsf{W}(\Gamma\theta_1, \overline{x{:}\alpha_x}\,; P\,; \mathcal{V}_1 \setminus \{\overline{\alpha_x}\}) \\
 & & \quad \mu = \mathsf{mgu}(\tau_1\theta_2 = \tau_2) \\
 & & \text{et } (\tau_3; \theta_3; \mathcal{V}_3) = \mathsf{W}(\Gamma\theta_1\theta_2\mu, \overline{x{:}\mathsf{Gen}(\alpha_x\theta_2\mu\,; \Gamma\theta_1\theta_2\mu)}\,; A_1\,; \mathcal{V}_2) \\
\text{sinon} & \Rightarrow & \mathsf{false} \\
\mathsf{Inst}(\forall\overline{\alpha}.\tau\,;\ \mathcal{V}) & \triangleq & (\tau[\overline{\alpha := \beta}]\,;\ \mathcal{V} \setminus \{\overline{\beta}\}) \quad \text{où les } \overline{\beta} \text{ sont des variables distinctes prises dans } \mathcal{V}
\end{array}
$$

FIG. 4.6 – L'algorithme W pour $\rho^{\ll}_{\forall}$

– Si $A = [P \ll A_2]A_1$ alors $\tau = \tau_3$ et $\theta = \theta_3 \circ \mu$ où $\tau_1\theta_2\mu = \tau_2\mu$ et $\tau_1, \theta_1, \tau_2, \theta_2, \tau_3, \theta_3$ sont calculés par des appels récursifs sur les sous-termes A_2, P et A_1.
Posons $\Delta = \overline{x : \alpha_x}$. Par hypothèse de récurrence on a $\Gamma\theta_1 \vdash_\Sigma A_2 : \tau_1$ et $\Gamma\theta_1\theta_2, \Delta\theta_2 \vdash_\Sigma P : \tau_2$ et $\Gamma\theta_1\theta_2\mu\theta_3, \mathsf{Gen}(\Delta\theta_2\mu; \Gamma\theta_1\theta_2\mu)\theta_3 \vdash_\Sigma A_1 : \tau_3$. On voit assez facilement que

$$\mathsf{Gen}(\Delta\theta_2\mu; \Gamma\theta_1\theta_2\mu)\theta_3 = \mathsf{Gen}(\Delta\theta_2\mu\theta_3; \Gamma\theta_1\theta_2\mu\theta_3) \quad \text{car} \quad \mathcal{D}om(\theta_3) \cap \mathcal{D}om(\Gamma\theta_1\theta_2\mu) = \emptyset$$

Par substitution sur les variables de type on a $\Gamma\theta \vdash_\Sigma A_2 : \tau_1\theta_2\mu\theta_3$ et $\Gamma\theta, \Delta\theta_2\mu\theta_3 \vdash_\Sigma P : \tau_2\mu\theta_3$. Étant donné que $\tau_1\theta_2\mu = \tau_2\mu$, on a

$$(\textsc{match})\ \frac{\Gamma\theta, \Delta\theta_2\mu\theta_3 \vdash_\Sigma P : \tau_2\mu\theta_3 \qquad \begin{array}{c}\Gamma\theta \vdash_\Sigma A_2 : \tau_1\theta_2\mu\theta_3 \\ \Gamma\theta_1\theta_2\mu\theta_3, \mathsf{Gen}(\Delta\theta_2\mu\theta_3; \Gamma\theta_1\theta_2\mu\theta_3) \vdash_\Sigma A_1 : \tau_3\end{array}}{\Gamma\theta \vdash_\Sigma [P \ll A_2]A_1 : \tau_3}$$

□

Pour démontrer la complétude, il faut faire des manipulations un peu plus subtiles sur les substitutions, notamment vérifier comment elles se comportent sur les nouvelles variables de type que l'on introduit. On définit donc une égalité affaiblie sur les substitutions, qui sera suffisante pour la démonstration.

Définition 35 (Coïncidence)
Deux substitutions θ_1 et θ_2 coïncident hors de $\mathcal{V}$, *noté* $\theta_1 \overset{\not\mathcal{V}}{=} \theta_2$, *si* $\alpha\theta_1 = \alpha\theta_2$ *pour tout* $\alpha \notin \mathcal{V}$.

En particulier, on remarquera que la grossièreté de cette égalité croît avec l'ensemble de variables considéré : si $\mathcal{V} \subseteq \mathcal{V}'$ et si $\theta_1 \overset{\not\mathcal{V}}{=} \theta_2$, alors $\theta_1 \overset{\not\mathcal{V}'}{=} \theta_2$.

On démontre alors trois propositions par récurrence simultanée. Dans ce théorème et dans sa démonstration, ϕ et ψ dénotent des substitutions.

Théorème 30 (Complétude et principalité de W)
Pour tous $\mathcal{V}$ et Γ tels que $\mathcal{V} \cap \mathcal{FV}^*(\Gamma) = \emptyset$, si $\Gamma\phi \vdash_\Sigma A : \tau'$, alors :

1. $\mathsf{W}(\Gamma; A; \mathcal{V}) \neq \mathsf{fail}$;
2. *il existe τ, θ et $\mathcal{V}'$ (uniques) tels que* $\mathsf{W}(\Gamma; A; \mathcal{V}) = (\tau; \theta; \mathcal{V}')$;
3. $\tau' = \tau\psi$ *et* $\phi \overset{\not\mathcal{V}}{=} \psi \circ \theta$ *pour un certain* ψ.

Démonstration : Par récurrence sur la structure de A. Comme pour la correction, nous ne traitons que les cas représentatifs.

– Si $A \equiv x$ alors $\Gamma(x)\phi \leq \tau'$, d'où $\Gamma(x) = \forall\overline{\alpha}.\tau_1$ et $\tau' = \tau_1\phi[\overline{\alpha := \tau}]$. Dans ce cas W n'échoue pas et $\tau = \mathsf{Inst}(\Gamma(x); \mathcal{V}) = \tau_1[\overline{\alpha := \beta}]$ avec $\theta = \mathcal{I}d$. Prenons

$$\psi \triangleq \begin{cases} [\overline{\beta := \tau}] & \text{sur } \overline{\beta} = \mathcal{V} \setminus \mathcal{V}' \\ \phi & \text{ailleurs} \end{cases}$$

On a alors $\tau' = \tau_1\phi[\overline{\alpha := \tau}] = \tau_1[\overline{\alpha := \beta}]\phi[\overline{\beta := \tau}] = \tau\psi$ et $\phi \overset{\not\mathcal{V}}{=} \psi$.

– Si $A \equiv \lambda P.A_1$ alors $\tau' = \tau'_1 \to \tau'_2$ où $\Gamma\phi, \Delta \vdash_\Sigma P : \tau'_1$ et $\Gamma\phi, \Delta \vdash_\Sigma A_1 : \tau'_2$ pour un certain Δ tel que $\mathcal{D}om(\Delta) = \mathcal{FV}(P)$. Par un renommage approprié des variables liées de A, on peut toujours considérer que $\Delta = \overline{x : \alpha_x}$ et étendre ϕ pour que le typage de P et A_1 aient eu lieu dans le contexte $\Gamma\phi, \Delta\phi$.

Par hypothèse de récurrence, $\mathsf{W}(\Gamma, \Delta; P; \mathcal{V} \setminus \{\overline{\alpha_x}\}) = (\tau_1; \theta_1; \mathcal{V}_1)$ tels que $\tau_1' = \tau_1\psi_1$ et $\phi \overset{\mathcal{V}\setminus\{\overline{\alpha_x}\}}{=} \psi_1 \circ \theta_1$ pour un certain ψ_1. De plus $\mathcal{V} \setminus \{\overline{\alpha_x}\} \subseteq \mathcal{V}$ donc $\phi \overset{\mathcal{V}}{=} \psi_1 \circ \theta_1$.
En particulier $(\Gamma, \Delta)\phi = (\Gamma, \Delta)\theta_1\psi_1$ donc par hypothèse de récurrence $\mathsf{W}(\Gamma\theta_1, \Delta\theta_1; A_1; \mathcal{V}_1) = (\tau_2; \theta_2; \mathcal{V}_2)$ tels que $\tau_2' = \tau_2\psi_2$ et $\psi_1 \overset{\mathcal{V}_1}{=} \psi_2 \circ \theta_2$ pour un certain ψ_2. De plus $\mathcal{V}_1 \subseteq \mathcal{V}$ donc $\psi_1 \overset{\mathcal{V}}{=} \psi_2 \circ \theta_2$.
Par conséquent, $\mathsf{W}(\Gamma; \lambda P.A_1; \mathcal{V})$ n'échoue pas et renvoie $(\tau_1\theta_2 \to \tau_2; \theta_2 \circ \theta_1; \mathcal{V}_2)$. D'où, pour $\psi = \psi_2$, on a :

$$\tau\psi = (\tau_1\theta_2 \to \tau_2)\psi_2 = \tau_1\psi_1 \to \tau_2\psi_2 = \tau_1' \to \tau_2' = \tau'$$
$$\phi \overset{\mathcal{V}}{=} \psi_1 \circ \theta_1 \overset{\mathcal{V}}{=} \psi_2 \circ \theta_2 \circ \theta_1 \overset{\mathcal{V}}{=} \psi \circ \theta_2 \circ \theta_1$$

– Si $A = A_1\, A_2$ alors il existe un type τ_1' tel que $\Gamma\phi \vdash_\Sigma A_1 : \tau_1' \to \tau'$ et $\Gamma\phi \vdash_\Sigma A_2 : \tau_1'$.
Par hypothèse de récurrence, $\mathsf{W}(\Gamma; A_1; \mathcal{V}) = (\tau_1; \theta_1; \mathcal{V}_1)$ tels que $\tau_1' \to \tau' = \tau_1\psi_1$ et $\phi \overset{\mathcal{V}}{=} \psi_1 \circ \theta_1$ pour un certain ψ_1.
En particulier, $\Gamma\phi = \Gamma\theta_1\psi_1$ donc par hypothèse de récurrence $\mathsf{W}(\Gamma\theta_1; A_2; \mathcal{V}_1) = (\tau_2; \theta_2; \mathcal{V}_2)$ tels que $\tau_1' = \tau_2\psi_2$ et $\psi_1 \overset{\mathcal{V}_1}{=} \psi_2 \circ \theta_2$ pour un certain ψ_2. De plus $\mathcal{V}_1 \subseteq \mathcal{V}$ donc $\psi_1 \overset{\mathcal{V}}{=} \psi_2 \circ \theta_2$.
Alors $\psi_3 = \psi_2 \circ [\alpha := \tau']$ est un unificateur pour $\tau_1\theta_2$ et $\tau_2 \to \alpha$:

$$\tau_1\theta_2\psi_3 = \tau_1\theta_2\psi_2 = \tau_1\psi_1 = \tau_1' \to \tau' = \tau_2\psi_2 \to \alpha[\alpha := \tau'] = (\tau_2 \to \alpha)\psi_3$$

Pour vérifier ces égalités on remarquera que τ_1 ne contient aucune variable de $\mathcal{V}_1$, ni α *a fortiori*. Par conséquent, l'unificateur le plus général de ces deux types est un certain μ tel que $\psi_3 = \psi \circ \mu$. Nous pouvons conclure : en effet $\tau' = \alpha\psi_3 = \alpha\mu\psi = \tau\psi$ et

$$\phi \overset{\mathcal{V}}{=} \psi_1 \circ \theta_1 \overset{\mathcal{V}}{=} \psi_2 \circ \theta_2 \circ \theta_1 \overset{\mathcal{V}}{=} \psi_3 \circ \theta_2 \circ \theta_1 \overset{\mathcal{V}}{=} \psi \circ \mu \circ \theta_2 \circ \theta_1 \overset{\mathcal{V}}{=} \psi \circ \theta$$

– Si $A \equiv [P \ll A_2]A_1$ alors il existe un type τ_1' et un contexte $\Delta = \overline{x : \alpha_x}$ tels que $\Gamma\phi \vdash_\Sigma A_2 : \tau_1'$ et $\Gamma\phi, \Delta\phi \vdash_\Sigma P : \tau_1'$ et $\Gamma\phi, \mathsf{Gen}(\Delta\phi; \Gamma) \vdash_\Sigma A_1 : \tau_2$ (où, comme pour l'abstraction, ϕ est étendue convenablement sur $\mathcal{D}om(\Delta)$).
Par hypothèse de récurrence, $\mathsf{W}(\Gamma; A_2; \mathcal{V}) = (\tau_1; \theta_1; \mathcal{V}_1)$ avec $\tau_1' = \tau_1\psi_1$ et $\phi \overset{\mathcal{V}}{=} \psi_1 \circ \theta_1$ pour un certain ψ_1.
En particulier $\mathcal{D}om(\theta_1) \cap \mathcal{D}om(\Delta) = \emptyset$ donc $\Gamma\phi, \Delta\phi = (\Gamma\theta_1, \Delta)\psi_1$. Par hypothèse de récurrence, $\mathsf{W}(\Gamma\theta_1, \Delta; P; \mathcal{V}_1 \setminus \{\overline{\alpha_x}\}) = (\tau_2; \theta_2; \mathcal{V}_2)$ avec $\tau_1' = \tau_2\psi_2$ et $\psi_1 \overset{\mathcal{V}_1\setminus\{\overline{\alpha_x}\}}{=} \psi_2 \circ \theta_2$ pour un certain ψ_2. De plus $\mathcal{V}_1 \setminus \{\overline{\alpha_x}\} \subseteq \mathcal{V}$ donc $\psi_1 \overset{\mathcal{V}}{=} \psi_2 \circ \theta_2$.
On a alors $\tau_1\theta_2\psi_2 = \tau_1\psi_1 = \tau_1' = \tau_2\psi_2$ (car τ_1 n'a pas de variables dans $\mathcal{V}_1$), donc les types $\tau_1\theta_2$ et τ_2 admettent un unificateur le plus général μ tel que $\psi_2 = \psi_3 \circ \mu$ pour un certain ψ_3.
Jusqu'ici, nous avons vu que

$$\phi \overset{\mathcal{V}}{=} \psi_1 \circ \theta_1 \overset{\mathcal{V}}{=} \psi_2 \circ \theta_2 \circ \theta_1 \overset{\mathcal{V}}{=} \psi_3 \circ \mu \circ \theta_2 \circ \theta_1$$
$$\psi_1 \overset{\mathcal{V}_1\setminus\{\overline{\alpha_x}\}}{=} \psi_2 \circ \theta_2 \overset{\mathcal{V}_1\setminus\{\overline{\alpha_x}\}}{=} \psi_3 \circ \mu \circ \theta_2$$

Or aucune variable de $\mathcal{V}$ n'est libre dans Γ et les variables libres de Δ sont les $\overline{\alpha_x}$ donc

$$\Gamma\phi, \Delta\phi = \Gamma\phi, \Delta\psi_1 = \Gamma\theta_1\theta_2\mu\psi_3, \Delta\theta_2\mu\psi_3 = (\Gamma\theta_1\theta_2\mu, \Delta\theta_2\mu)\psi_3$$

Par hypothèse de récurrence, $\mathsf{W}(\Gamma\theta_1\theta_2\mu, \Delta\theta_2\mu; A_1; \mathcal{V}_2) = (\tau_3; \theta_3; \mathcal{V}_3)$ tels que $\tau' = \tau_3\psi_4$ et $\psi_3 \overset{\mathcal{V}_2}{=} \psi_4 \circ \theta_3$ pour un certain ψ_4. De plus $\mathcal{V}_2 \subseteq \mathcal{V}$ donc $\psi_3 \overset{\mathcal{V}}{=} \psi_4 \circ \theta_3$.

Pour $\tau = \tau_3$ et $\theta = \theta_3 \circ \mu \circ \theta_2 \circ \theta_1$ on a donc bien $\tau' = \tau\psi_4$ et $\phi \overset{\mathcal{V}}{=} \psi_3 \circ \mu \circ \theta_2 \circ \theta_1 \overset{\mathcal{V}}{=} \psi_4 \circ \theta_3 \circ \mu \circ \theta_2 \circ \theta_1 \overset{\mathcal{V}}{=} \psi_4 \circ \theta$.

□

Corollaire 31 (Décidabilité du typage dans $\rho^{\ll}_{\forall}$)
Dans le système $\rho^{\ll}_{\forall}$, les deux problèmes suivants sont décidables.

1. *Étant donnés un contexte Γ, un terme A et un type τ, peut-on dériver $\Gamma \vdash_\Sigma A : \tau$?*
2. *Étant donnés un contexte Γ et un terme A, existe-t-il un type τ tel que $\Gamma \vdash_\Sigma A : \tau$?*

L'algorithme W répond directement au second problème. Pour le premier, par principalité, il faut et il suffit que le type τ soit une instance de celui trouvé par W, et on est donc ramené à un problème de filtrage modulo renommage des variables de variables de type liées.

4.2 Récursion typée

4.2.1 Typage d'un point fixe

Ces différents systèmes de types étant des extensions conservatives de leurs homologues dans le λ-calcul, on pourrait croire qu'ils vérifient exactement les mêmes propriétés. Or, il apparaît que la normalisation forte, l'une des propriétés essentielles des λ-calculs typés, n'est pas conservée : les termes habituellement rejetés par les systèmes de types du λ-calcul le sont aussi ici, mais il existe des termes non normalisants propres au ρ-calcul qui sont acceptés par les systèmes tels que $\rho 1$. Pour bien comprendre pourquoi, regardons à quoi devrait ressembler une hypothétique dérivation de type pour le λ-terme $\omega\omega$:

$$\dfrac{\dfrac{\vdots}{\vdash \lambda x.x\,x : (\tau \to \sigma) \to \upsilon} \quad \dfrac{\dfrac{(\text{VAR})\ \overline{x : \tau \vdash x : \nu \to \sigma} \qquad (\text{VAR})\ \overline{x : \tau \vdash x : \nu}}{(\text{APP})\ x : \tau \vdash x\,x : \sigma}}{(\text{ABS})\ \vdash \lambda x.x\,x : \tau \to \sigma}}{(\text{APP})\ \vdash (\lambda x.x\,x)\,(\lambda x.x\,x) : \upsilon}$$

En λ-calcul simplement typé par exemple, on voit que pour que les deux utilisations de la règle (VAR) soient valides, il faudrait que $\nu = \tau = \nu \to \sigma$, ce qui est impossible car aucun type ν ne peut être un de ses sous-termes propres. Ceci traduit le fait qu'on a essayé d'appliquer x à lui-même, et donc que x (en tant qu'argument) devrait être dans son propre domaine (en tant que fonction).

Pour parvenir à typer un tel terme, il faut donc trouver un moyen pour que x accepte x lui-même comme argument. La solution classique est celle des *types récursifs* dont M. Coppo a montré qu'ils permettent de typer tout λ-terme [Cop85]. Pour typer ω, on affecte à x le type $\mu t.(t \to t)$ et on utilise un mécanisme de $\mathsf{fold}/\mathsf{unfold}$ pour passer de $\mu t.(t \to t)$ à $\mu t.(t \to t) \to \mu t.(t \to t)$ et inversement. Le terme

$$\big(\lambda x.(\mathsf{unfold}(x)\,x)\big)\,\mathsf{fold}\big(\lambda x.(\mathsf{unfold}(x)\,x)\big)$$

a alors pour type $\mu t.(t \to t)$ et il admet des réductions infinies selon la règle β et la règle $\mathsf{unfold}(\mathsf{fold}(A)) \mapsto A$.

Dans $\rho 1$, grâce au filtrage, on peut obtenir un résultat similaire sans utiliser de types récursifs : prenons une variable x de type $\iota \to \kappa$; pour obtenir un terme de type ι, il nous suffit de

considérer une constante f de type $(\iota \to \kappa) \to \iota$. On a alors la dérivation de typage suivante, où $\Gamma \triangleq (x{:}\iota \to \kappa)$.

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash_\Sigma f : (\iota \to \kappa) \to \iota \qquad \Gamma \vdash_\Sigma x : \iota \to \kappa}{\Gamma \vdash_\Sigma f\, x : \iota}\,(\textsc{app}) \qquad \dfrac{\Gamma \vdash_\Sigma x : \iota \to \kappa \qquad \overline{\Gamma \vdash_\Sigma f\, x : \iota}}{\Gamma \vdash_\Sigma x\,(f\, x) : \kappa}\,(\textsc{app})}{\vdash_\Sigma \lambda(f\, x){:}\Gamma.x\,(f\, x) : \iota \to \kappa}\,(\textsc{abs})$$

On appellera désormais ω_f le terme $\lambda(f\, x){:}\Gamma.x\,(f\, x)$. On a donc :

$$\dfrac{\dfrac{\dfrac{\vdots}{\vdash_\Sigma \omega_f : \iota \to \kappa} \qquad \dfrac{\overline{\vdash_\Sigma f : (\iota \to \kappa) \to \iota} \qquad \dfrac{\vdots}{\vdash_\Sigma \omega_f : \iota \to \kappa}}{\vdash_\Sigma f\, \omega_f : \iota}\,(\textsc{app})}{\vdash_\Sigma \omega_f\,(f\, \omega_f) : \kappa}\,(\textsc{app})$$

Et ce terme admet bien une chaîne infinie de réductions (en fait c'est la seule possible) :

$$\begin{array}{rcl} \omega_f\,(f\, \omega_f) & \equiv & (\lambda(f\, x).x\,(f\, x))\,(f\, \omega_f) \\ & \mapsto_\rho & x\,(f\, x)[x := \omega_f] \\ & = & \omega_f\,(f\, \omega_f) \\ & \mapsto_\rho & \ldots \end{array}$$

Il est bien entendu possible de définir de même un combinateur de point fixe : pour $x : \tau \to \tau$, on se donne une variable $y : \iota \to (\tau \to \tau) \to \tau$ et une constante $f : (\iota \to (\tau \to \tau) \to \tau) \to \iota$ et alors le terme

$$\mathsf{Y}_f \triangleq (\lambda(f\, y).\lambda x.x\,(y\,(f\, y)\, x))\,\big(f\,(\lambda(f\, y).\lambda x.x\,(y\,(f\, y)\, x))\big)$$

admet pour type $(\tau \to \tau) \to \tau$ et, pour tout A de type $\tau \to \tau$,

$$\mathsf{Y}_f\, A \;\mapsto\!\!\!\!\to_\rho\; A\,(\mathsf{Y}_f\, A) \;\mapsto\!\!\!\!\to_\rho\; \ldots$$

4.2.2 Formalisation de programmes définis par réécriture

Cette section reprend des résultats exposés dans [CLW03]. Nous allons utiliser des points fixes en combinaison avec les capacités de filtrage du ρ-calcul afin d'encoder des systèmes de réécriture sous forme de ρ-termes. Pour cela, nous commençons par étendre légèrement la syntaxe pour pouvoir détecter explicitement les échecs de filtrage ; nous montrons ensuite comment représenter les TRS en général ainsi que certaines stratégies de réécriture.

Notons qu'un codage des systèmes de réécriture a déjà été donné pour une version antérieure du ρ-calcul [CK01]. Les apports des travaux présentés ici sont :

- l'adaptation de l'encodage dans la nouvelle version, ce qui n'est pas trivial car l'ancienne version disposait de règles de congruence et de distributivité qu'on ne peut pas utiliser ici ;
- un traitement plus atomique des échecs de filtrage (l'ancien encodage utilisait directement un opérateur primitif first au lieu de le définir comme nous allons le faire au moyen de stk) ;
- ce nouvel encodage est typable.

L'opérateur stk

On a vu qu'en cas d'échec de filtrage, la ρ-réduction n'avait pas lieu : par exemple, $(\lambda a.b)\,c$ est en forme normale. En revanche, nous n'avons pour l'instant aucun moyen dans le calcul pour identifier de tels échecs. De plus, il nous est impossible de le différencier d'un échec non définitif : par exemple, dans $(\lambda x.((\lambda a.b)\,x))\,a$, le sous-terme $(\lambda a.b)\,x$ est en forme normale, mais après instanciation de x par le radical externe, il devient le radical $(\lambda a.b)\,a$ qui se réduit à a.

Pour cette section, nous ajouterons donc au calcul une constante dédiée stk avec l'idée qu'elle représente tous les échecs de filtrage définitifs.

$$\begin{array}{rcl} \mathcal{P}^{\mathsf{stk}} & ::= & \mathcal{X} \mid \mathsf{stk} \mid \mathcal{K}\,\overline{\mathcal{P}^{\mathsf{stk}}} \\ \mathcal{T}_\rho^{\mathsf{stk}} & ::= & \mathcal{X} \mid \mathcal{K} \mid \mathsf{stk} \mid \lambda\mathcal{P}^{\mathsf{stk}}{:}\Delta.\mathcal{T}_\rho^{\mathsf{stk}} \mid \mathcal{T}_\rho^{\mathsf{stk}}\,\mathcal{T}_\rho^{\mathsf{stk}} \mid \mathcal{T}_\rho^{\mathsf{stk}} \wr \mathcal{T}_\rho^{\mathsf{stk}} \end{array}$$

La sémantique exacte de stk devrait être la suivante : si pour aucune instanciation et réduction de l'argument, il n'existe de filtre, alors le ρ-radical se réduit à stk :

$$\frac{\forall \theta_1, \theta_2,\ \forall B',\ B\theta_1 \mapsto\!\!\!\!\!\rightarrow_{\rho\delta} B' \ \Rightarrow\ P\theta_2 \not\equiv B'}{(\lambda P.A)\,B \to_{\mathsf{stk}} \mathsf{stk}}$$

Cependant, une telle règle de réduction pose problème dans la mesure où la condition de réduction semble indécidable. En effet, B peut contenir un λ-terme ayant un nombre arbitraire (éventuellement infini) de réductions possibles, qu'il faut *a priori* toutes explorer pour décider si le filtrage selon P est possible.

On ne peut donc pas définir la sémantique opérationnelle de notre calcul de cette façon. Cependant, en pratique et plus particulièrement dans le cadre des TRS, il n'est pas nécessaire d'être aussi général. On se contentera donc de la condition suffisante suivante.

Définition 36 (Échecs définitifs)

La relation $\not\sqsubseteq$ *sur* $\mathcal{P}^{\mathsf{stk}} \times \mathcal{T}_\rho^{\mathsf{stk}}$ *est définie comme suit :*

$$\begin{array}{lcll} \mathsf{stk} & \not\sqsubseteq & g\,\overline{B} & \text{si } g \not\equiv \mathsf{stk} \\ \mathsf{stk} & \not\sqsubseteq & \lambda Q.B & \\ f\,P_1 \ldots P_m & \not\sqsubseteq & g\,B_1 \ldots B_n & \text{si } f \not\equiv g \text{ ou } n \neq m \text{ ou } \exists i,\ P_i \not\sqsubseteq B_i \\ f\,\overline{P} & \not\sqsubseteq & \mathsf{stk} & \\ f\,\overline{P} & \not\sqsubseteq & \lambda Q.B & \\ f\,\overline{P} & \not\sqsubseteq & (\lambda Q.A)B & \text{si } Q \not\sqsubseteq B \text{ ou } f\,\overline{P} \not\sqsubseteq A \end{array}$$

À partir de cette relation, on peut alors définir la réduction qui gouverne la détection des échecs.

Définition 37 (Réduction $\to_{\mathsf{stk}}$**)**

La relation $\to_{\mathsf{stk}}$ *est définie par les règles :*

$$\begin{array}{rcll} (\lambda P{:}\Delta.A)\,B & \to_{\mathsf{stk}} & \mathsf{stk} & \text{si } P \not\sqsubseteq B \\ \mathsf{stk} \wr A & \to_{\mathsf{stk}} & A & \\ A \wr \mathsf{stk} & \to_{\mathsf{stk}} & A & \\ \mathsf{stk}\,A & \to_{\mathsf{stk}} & \mathsf{stk} & \end{array}$$

On écrira $\mapsto_{\mathsf{stk}}$, $\mapsto\!\!\!\!\!\rightarrow_{\mathsf{stk}}$ *et* $=_{\mathsf{stk}}$ *les relations dérivées de* $\to_{\mathsf{stk}}$.

Pour l'encodage des systèmes de réécriture, on considérera donc que la sémantique opérationnelle des ρ-termes est donnée par la réduction suivante.

Définition 38 (Réduction $\mapsto_{\rho\delta}^{\mathsf{stk}}$)
La relation $\mapsto_{\rho\delta}^{\mathsf{stk}}$ est définie comme $\mapsto_{\mathsf{stk}} \cup \mapsto_{\rho\delta}$.

Montrons que la définition de la relation $\not\sqsubseteq$ est correcte, c'est-à-dire que les paires qui la vérifient sont bien des échecs de filtrage définitifs.

Lemme 32 (Correction de $\not\sqsubseteq$)
Pour tous P et A, si $P \not\sqsubseteq A$ alors $\forall \theta_1, \theta_2,\ \forall A',\ A\theta_1 \mapsto\!\!\!\!\twoheadrightarrow_{\rho\delta}^{\mathsf{stk}} A' \ \Rightarrow\ P\theta_2 \not\equiv A'$.

Démonstration : Par récurrence sur A. Précisons que la quantification sur P fait partie de l'hypothèse de récurrence, ce qui nous permettra de changer de motif lors des utilisations de la récurrence.

Si $A \equiv x$ alors on n'a jamais $P \not\sqsubseteq A$;

Si $A \equiv g$ alors $g\theta_1 \equiv g$ est en forme normale. Par ailleurs, $P \not\sqsubseteq g$ si et seulement si $P \equiv f\,\overline{P}$ avec $f \neq g$ ou $P \equiv \mathsf{stk}$. Il s'ensuit que $P\theta_2 \equiv f\,\overline{P\theta_1}$ ou stk, qui ne peuvent en aucun cas être identiques à g.

Si $A \equiv \mathsf{stk}$ alors $P \not\sqsubseteq \mathsf{stk}$ n'est possible que pour $P \equiv f\,\overline{P}$. Immédiatement $\mathsf{stk}\theta_1 \equiv \mathsf{stk}$ est en forme normale et ne peut être identique à $P\theta_2$.

Si $A \equiv A_1 \wr A_2$ alors $P \not\sqsubseteq A$ est impossible.

Si $A \equiv \lambda Q.A_1$ alors $P \not\sqsubseteq A$ impose que $P \not\equiv x$. Mais alors $A\theta_1 \equiv \lambda Q.A_1\theta_1$ ne peut se réduire qu'à un terme de la forme $\lambda Q.A_1'$. Comme P est forcément stk ou un terme algébrique différent d'une variable, $P\theta_2 \not\equiv \lambda Q.A_1'$.

Si $A \equiv A_1\,A_2$ quatre sous-cas sont à envisager. Les deux derniers sont les seuls cas utilisant l'hypothèse de récurrence.

- **Si** $P \equiv \mathsf{stk}$ **et** $A \equiv g\,A_1 \ldots A_n$ avec $g \not\equiv \mathsf{stk}$ alors $A\theta_1 \equiv g\,A_1\theta_1 \ldots A_n\theta_1 \mapsto\!\!\!\!\twoheadrightarrow_{\rho\delta} g\,A_1' \ldots A_n'$ qui ne peut se réduire à $P\theta_2 \equiv \mathsf{stk}$ puisque $g \not\equiv \mathsf{stk}$.
- **Si** $A \equiv g\,A_1 \ldots A_n$ **et** $P \equiv f\,P_1 \ldots P_m$ **avec** $f \neq g$ **ou** $n \neq m$ alors de même $A\theta_1 \mapsto\!\!\!\!\twoheadrightarrow_{\rho\delta}$ $g\,A_1' \ldots A_n'$ qui ne peut être identique à $P\theta_2 \equiv g\,P_1\theta_2 \ldots P_m\theta_2$ puisque $f \not\equiv g$ ou $n \neq m$.
- **Si** $A \equiv g\,A_1 \ldots A_n$ **et** $P \equiv g\,P_1 \ldots P_n$ alors $P \not\sqsubseteq A$ impose que $\exists i, P_i \not\sqsubseteq A_i$. Mais encore une fois $A\theta_1 \mapsto\!\!\!\!\twoheadrightarrow_{\rho\delta} g\,A_1' \ldots A_n'$ avec $A_i\theta_1 \mapsto\!\!\!\!\twoheadrightarrow_{\rho\delta} A_i'$. Par hypothèse de récurrence sur A_i, on a $A_i' \not\equiv P_i\theta_2$, et donc $g\,A_1' \ldots A_n' \not\equiv P\theta_2$.
- **Si** $A \equiv (\lambda Q.A_1)\,B$ **avec** $P \not\equiv x$ alors $Q \not\sqsubseteq B$ ou $P \not\sqsubseteq A_1$. Dans le premier cas, par hypothèse de récurrence sur B, le terme $B\theta_1$ ne peut être réduit à aucune instance de Q. Par conséquent, le ρ-radical de tête de $A\theta_1$ ne sera jamais réduit, et donc A ne peut pas se réduire à une instance de P. Dans le second cas, supposons que le radical de tête de $A\theta_1$ puisse être réduit (sinon on procède comme dans le premier cas). Alors on a

$$A\theta_1 \ \equiv\ (\lambda Q.A_1\theta_1)\,B\theta_1 \ \mapsto\!\!\!\!\twoheadrightarrow_{\rho}\ (\lambda Q.A_1')\,B' \ \mapsto_{\rho}\ A_1'\theta_1' \wr \ldots \wr A_1'\theta_n' \ \mapsto_{\rho}\ A_{11}'' \wr \ldots \wr A_{1n}''$$

avec $A_1\theta_1\theta_i' \mapsto\!\!\!\!\twoheadrightarrow_{\rho\delta} A_{1i}''$ pour tout i. Par hypothèse de récurrence sur A_1 (et pour les substitutions $\theta_1\theta_i'$) aucun terme A_{1i}'' ne peut être identique à $P\theta_2$.

□

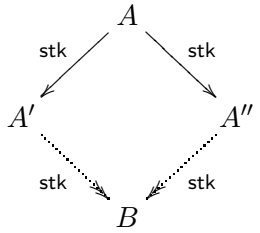
Corollaire 33 (Stabilité de $\not\sqsubseteq$)
La relation $P \not\sqsubseteq A$ est conservée par substitution et réduction de A.

Il est donc légitime d'utiliser cette relation pour éliminer les échecs de filtrage définitifs dans les structures.

Nous montrons maintenant la confluence de $\mapsto_{\rho\delta}^{\mathsf{stk}}$ grâce au lemme de Hindley-Rosen [Ros73], qui énonce que si deux relations confluentes $\mapsto_R$ et $\mapsto_S$ commutent, alors la relation $\mapsto_R \cup \mapsto_S$ est confluente. Nous savons déjà que $\mapsto_{\rho\delta}$ est confluente (théorème 16) ; démontrons deux lemmes préliminaires.

Lemme 34 (Confluence et terminaison de $\mapsto_{\mathsf{stk}}$)
La relation $\mapsto_{\mathsf{stk}}$ est confluente.

Démonstration : La terminaison est évidente puisque, dans les quatre règles données, la taille des termes décroît clairement par $\mapsto_{\mathsf{stk}}$. Pour montrer la confluence, on peut donc se contenter de montrer la confluence locale : si $A \mapsto_{\mathsf{stk}} A'$ et $A \mapsto_{\mathsf{stk}} A''$ alors il existe B tel que $A' \mapsto\!\!\!\!\rightarrow_{\mathsf{stk}} B$ et $A'' \mapsto\!\!\!\!\rightarrow_{\mathsf{stk}} B$.

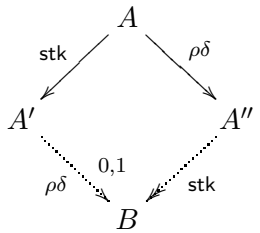


Cette dernière propriété se montre facilement : toutes les paires critiques (c.-à-d. les cas où les deux réductions ont lieu dans des sous-termes non disjoints de A) convergent vers stk, sauf le cas $\mathsf{stk} \wr A = A \wr \mathsf{stk}$ qui converge vers A. □

Avant de montrer que $\mapsto_{\rho\delta}$ et $\mapsto_{\mathsf{stk}}$ commutent, voyons un premier diagramme.

Lemme 35 (Commutation locale de $\mapsto_{\rho\delta}$ et $\mapsto_{\mathsf{stk}}$)
Si $A \mapsto_{\mathsf{stk}} A'$ et $A \mapsto_{\rho\delta} A''$ alors il existe un terme B tel que $A' \equiv B$ ou $A' \mapsto_{\rho\delta} B$ et $A'' \mapsto\!\!\!\!\rightarrow_{\mathsf{stk}} B$.

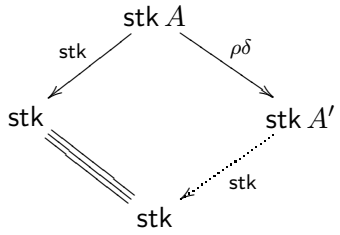
On notera $\mapsto^{0,1}$ une réduction en au plus 1 pas, et donc ce lemme se traduit par le diagramme



Démonstration : Lorsque les deux réductions ont lieu dans des sous-termes disjoints de A, le diagramme est trivialement vérifié (avec une réduction effective $A' \mapsto_{\rho\delta} B$).

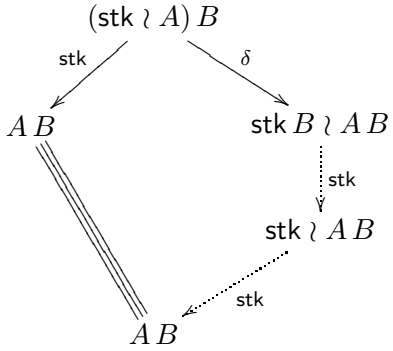
Les paires critiques avec la relation $\mapsto_\rho$ ne posent pas de problème particulier : par correction de $\not\sqsubseteq$, un terme $(\lambda P.A)\,B$ ne peut être à la fois un ρ-radical et un stk-radical. Les stk-réductions

ont donc forcément lieu dans A et B, et on ferme le diagramme grâce à la stabilité de $\not\sqsubseteq$ par substitution. On peut remarquer que le cas



est un de ceux à cause desquels on n'a pas forcément besoin d'une ρ-réduction pour fermer le diagramme.

Les paires critiques avec $\mapsto_\delta$ sont les plus importantes, puisque stk modifie le comportement des structures. On voit ici une justification pour la quatrième règle de définition de $\rightarrow_{\mathsf{stk}}$, qui est nécessaire pour fermer le diagramme suivant :

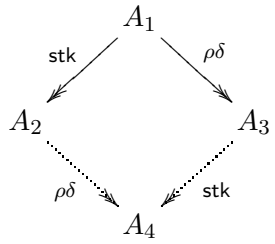


Les autres cas ne posent pas de problème particulier. □

Théorème 36 (Confluence de $\mapsto_{\rho\delta}^{\mathsf{stk}}$ [CLW03])

La relation $\mapsto_{\rho\delta}^{\mathsf{stk}}$ est confluente.

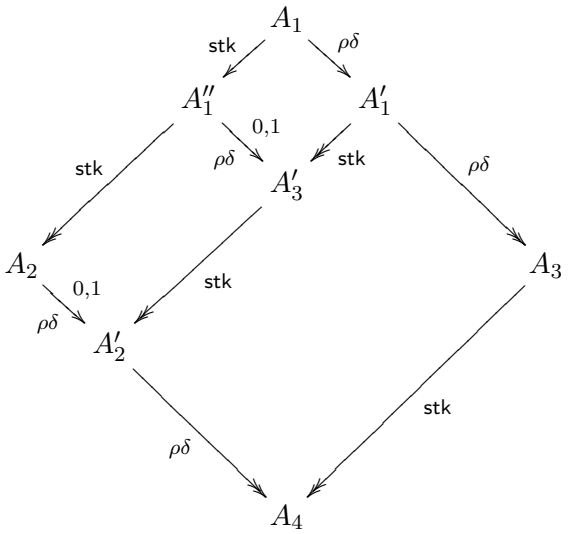
Démonstration : Comme on a vu que $\mapsto_{\rho\delta}$ et $\mapsto_{\mathsf{stk}}$ sont toutes deux confluentes, il ne nous reste qu'à montrer qu'elles commutent : si $A_1 \mapsto\!\!\!\!\rightarrow_{\mathsf{stk}} A_2$ et $A_1 \mapsto\!\!\!\!\rightarrow_{\rho\delta} A_3$ alors il existe un terme A_4 tel que $A_2 \mapsto\!\!\!\!\rightarrow_{\rho\delta} A_4$ et $A_3 \mapsto\!\!\!\!\rightarrow_{\mathsf{stk}} A_4$.



On procède par récurrence sur la longueur de la réduction $A_1 \mapsto\!\!\!\!\rightarrow_{\rho\delta} A_3$.

Si $A_1 \equiv A_3$ alors trivialement $A_3 \mapsto\!\!\!\!\rightarrow_{\mathsf{stk}} A_2$.

Si $A_1 \mapsto_{\not\beta\delta} A'_1 \twoheadmapsto_{\not\beta\delta} A_3$ alors nous allons établir le diagramme suivant, qui est similaire à la démonstration du lemme de Newman [New42].



Montrons qu'il existe A'_2 tel que $A'_1 \twoheadmapsto_{\mathsf{stk}} A'_2$ et $A_2 \mapsto_{\not\beta\delta}^{0,1} A'_2$, par récurrence sur la longueur de $A_1 \twoheadmapsto_{\mathsf{stk}} A_2$.

Si $A_1 \equiv A_2$ alors immédiatement $A'_2 = A'_1$ convient.

Si $A_1 \mapsto_{\mathsf{stk}} A''_1 \twoheadmapsto_{\mathsf{stk}} A_2$ alors on peut appliquer la commutation locale (lemme 35) : il existe A'_3 tel que $A''_1 \mapsto_{\not\beta\delta}^{0,1} A'_3$ et $A'_1 \twoheadmapsto_{\mathsf{stk}} A'_3$. Si $A''_1 \equiv A'_3$ alors immédiatement $A'_1 \twoheadmapsto_{\mathsf{stk}} A'_3 \twoheadmapsto_{\mathsf{stk}} A_2$ donc $A'_2 = A_2$ convient. Sinon, on peut appliquer l'hypothèse de récurrence sur la réduction plus courte $A''_1 \twoheadmapsto_{\mathsf{stk}} A_2$, et donc il existe A'_2 tel que $A'_3 \twoheadmapsto_{\mathsf{stk}} A'_2$ et $A_2 \mapsto_{\not\beta\delta}^{0,1} A_2$. Par conséquent on a également $A'_1 \twoheadmapsto_{\mathsf{stk}} A'_3 \twoheadmapsto_{\mathsf{stk}} A'_2$.

Il ne nous reste plus qu'à utiliser l'hypothèse de récurrence sur la réduction $A'_1 \twoheadmapsto_{\not\beta\delta} A_3$ pour trouver un A_4 tel que $A'_2 \twoheadmapsto_{\not\beta\delta} A_4$ et $A_3 \twoheadmapsto_{\mathsf{stk}} A_4$. On a alors bien $A_2 \twoheadmapsto_{\not\beta\delta} A_4$ puisque $A_2 \mapsto_{\not\beta\delta}^{0,1} A'_2 \twoheadmapsto_{\not\beta\delta} A_4$.

Les deux réductions commutent, et on peut donc conclure grâce au lemme de Hindley-Rosen. □

Voyons enfin l'incidence de stk sur le typage. Étant donné que stk est susceptible de remplacer $(\lambda P.A)\,B$ pour n'importe quelle valeur de A, la préservation du type n'est possible que si le terme stk peut avoir tous les types possibles : on rajoutera donc la règle

$$(\textsc{stuck})\ \frac{}{\Gamma \vdash_\Sigma \mathsf{stk} : \tau}$$

Il serait possible de plutôt considérer une *famille* de constantes stk_τ qui garde l'information du type de A, mais cela demanderait de faire des vérifications de type pour décider de l'application de la réduction $\mapsto_{\mathsf{stk}}$.

Les propriétés de génération, substitution et préservation du type restent valides ; évidemment, l'unicité et la décidabilité sont remises en cause. On peut les retrouver en imposant des conditions raisonnables sur les termes : par exemple, si stk n'apparaît que dans des structures dont au moins un membre n'est pas stk, alors le type de stk dans cette structure est imposé par les autres termes et donc le typage reste unique et décidable. Ce sera le cas de tous les termes que nous définirons ; notons cependant que cette condition n'est pas préservée par la réduction $\mapsto_{\not\beta\delta}^{\mathsf{stk}}$.

Encodage de TRS

Pour alléger les notations, dans tout le reste de cette section, nous omettrons généralement les types, mais tous les termes construits restent typables dans $\rho 1$, à condition de bien choisir les types des constantes rec utilisées. Commençons par montrer un exemple de point fixe incluant un système de réécriture qui définit l'addition, après quoi nous donnerons une méthode systématique de transformation d'un TRS en ρ-terme. On considère la signature

$$\Sigma = \{0{:}\iota, S{:}\iota \to \iota, add{:}\iota \to \iota \to \iota, rec{:}(\kappa \to \iota \to \iota) \to \kappa\}$$

et on définit le ρ-terme

$$plus \;\triangleq\; \lambda(rec\, z).\left(\begin{array}{l} \lambda(add\, 0\, y).y \\ \wr\lambda(add\,(S\,x)\,y).S\;\;(z\,(rec\, z)\,(add\, x\, y)) \end{array}\right)$$

Comme dans le combinateur de point fixe usuel, la variable z va transporter une copie de $plus$ au fur et à mesure des réductions, et permettre des « appels récursifs ». La figure 4.7 montre comment ce terme calcule l'addition de deux entiers de Peano ; les expressions $\overline{m}$, $\overline{m+n}$ et $\overline{m-n}$ sont des abréviations pour les termes $S(\ldots(S\,0)\ldots)$ avec le nombre de S correspondant.

$$\begin{array}{ll}
& plus\,(rec\; plus)\;\;(add\,\overline{n}\,\overline{m}) \\
\mapsto_{\rho\delta} & (\lambda(add\,0\,y).y)\;\;(add\,\overline{n}\,\overline{m}) \\
& \wr\,(\lambda(add\,(S\,x)\,y).S\;\;(plus\,(rec\; plus)\;\;(add\, x\, y)))\;\;(add\,\overline{n}\,\overline{m}) \\
\mapsto_{\rho} & (\lambda(add\,0\,y).y)\;\;(add\,\overline{n}\,\overline{m}) \\
& \wr\, S\,(plus\,(rec\; plus)\;\;(add\,\overline{n-1}\,\overline{m})) \\
\mapsto_{\mathsf{stk}} & S\,(plus\,(rec\; plus)\;\;(add\,\overline{n-1}\,\overline{m})) \\
\mapsto_{\rho\delta} & S\left(\begin{array}{l} (\lambda(add\,0\,y).y)\;\;(add\,\overline{n-1}\,\overline{m}) \\ \wr\,(\lambda(add\,(S\,x)\,y).S\,(plus\,(rec\; plus)\;\;(add\, x\, y)))\;\;(add\,\overline{n-1}\,\overline{m}) \end{array}\right) \\
& \vdots \\
\mapsto^{\mathsf{stk}}_{\rho\delta} & S\,(\ldots(S\left(\begin{array}{l} (\lambda(add\,0\,y).y)\;\;(add\,0\,\overline{m}) \\ \wr\,(\lambda(add\,(S\,x)\,y).S\,(plus\,(rec\; plus)\;\;(add\, x\, y)))\;\;(add\,0\,\overline{m}) \end{array}\right)\ldots) \\
\mapsto_{\rho\delta} & S\,(\ldots(S\left(\begin{array}{l} \overline{m} \\ \wr\,(\lambda(add\,(S\,x)\,y).S\,(plus\,(rec\; plus)\;\;(add\, x\, y)))\;\;(add\,0\,\overline{m}) \end{array}\right)\ldots) \\
\mapsto_{\mathsf{stk}} & S\,(\ldots(S\,\overline{m})) \\
\equiv & \overline{m+n}
\end{array}$$

FIG. 4.7 – Réductions d'un ρ-terme encodant l'addition

Le terme $plus(rec\, plus)\;\;(add\,\overline{n}\,\overline{m})$ admet évidemment des réductions infinies, mais la réduction de la figure 4.7 montre qu'il est faiblement normalisant. La réduction $\mapsto_{\mathsf{stk}}$ élimine les $n-1$ premières tentatives d'application de la règle $\lambda(add\,0\,y).y$ car $0 \not\sqsubseteq \overline{n}$ tant que $n > 0$; en fin de réduction $\mapsto_{\mathsf{stk}}$ détecte l'échec $S\,x \not\sqsubseteq 0$ ce qui permet d'éliminer le sous-terme non terminant. La forme normale $\overline{m+n}$ (unique par confluence) est bien le résultat attendu.

Pour encoder le système de réécriture, il est ici suffisant de réappliquer $z\,(rec\; z)$ à un seul sous-terme dans les membres droits des règles ; par contre, en général, on ne sait pas quels sont

les sous-termes des membres droits susceptibles d'être encore réécrits, et il faut donc prévoir un mécanisme qui « essaye » des règles sur un terme et le laisse tel quel si aucune règle ne convient.

On peut définir un tel mécanisme grâce à stk. En effet, si $A_1, A_2, \ldots, A_n$ sont des termes, posons

$$\mathsf{first}(A_1, A_2, \ldots, A_n) \triangleq \lambda x.\,((\lambda \mathsf{stk}.A_n\, x \wr \lambda y.y)\,(\ldots\,(\lambda \mathsf{stk}.A_2\, x \wr \lambda y.y)\,(A_1 x)))$$

Alors on vérifie que, pour tout terme B, le terme $\mathsf{first}(A_1, \ldots, A_n)\, B$ s'évalue comme $A_i\, B$ si

$$\begin{cases} \forall j < i, & A_j\, B & \mapsto_{\rho\delta}^{\mathsf{stk}} & \mathsf{stk} \\ & A_i\, B & \mapsto_{\rho\delta}^{\mathsf{stk}} & f\,\overline{B} \end{cases}$$

En effet, pour tout i on a les réductions suivantes : si $A_i\, B$ s'évalue à stk

$$(\lambda \mathsf{stk}.A_{i+1}B \wr \lambda y.y)\,(A_i\, B) \mapsto_{\rho\delta}^{\mathsf{stk}} (\lambda \mathsf{stk}.A_{i+1}B)\,\mathsf{stk} \wr (\lambda y.y)\,\mathsf{stk} \mapsto_{\rho\delta}^{\mathsf{stk}} A_{i+1}B \wr \mathsf{stk} \mapsto_{\mathsf{stk}} A_{i+1}B$$

et si $A_i\, B \mapsto_{\rho\delta}^{\mathsf{stk}} f\,\overline{B}$ alors $\mathsf{stk} \not\sqsubseteq f\,\overline{B}$ et donc

$$(\lambda \mathsf{stk}.A_{i+1}B \wr \lambda y.y)\,(A_i\, B) \mapsto_{\rho\delta}^{\mathsf{stk}} (\lambda \mathsf{stk}.A_{i+1}B)\,(f\overline{B}) \wr (\lambda y.y)\,(f\,\overline{B}) \mapsto_{\rho\delta}^{\mathsf{stk}} \mathsf{stk} \wr f\,\overline{B} \mapsto_{\mathsf{stk}} f\,\overline{B}$$

En particulier, on utilisera souvent $\mathsf{first}(A_1, \ldots, A_n, \lambda y.y)\, B$, qui essaye de même les différents $A_i\, B$ et renvoie son argument B à l'identique dans le cas où tous les $A_i\, B$ échouent.

Nous sommes maintenant en mesure de donner un encodage d'un TRS $\mathcal{R}$ quelconque.

Définition 39 (Encodage d'un TRS en ρ-calcul)
Soit $\mathcal{R} = \{l_i \to r_i\}^{i=1..n}$ un TRS sur la signature $\Sigma = \{a_1, \ldots, a_m\}$; soit z une variable non utilisée dans $\mathcal{R}$. Le terme $[\![\mathcal{R}]\!]$ est défini comme suit (avec la représentation usuelle des termes algébriques) :

$$[\![\mathcal{R}]\!] \triangleq \lambda(rec\,z).\mathsf{first}\begin{pmatrix} \lambda l_1.z\,(rec\,z)\,r_1, \\ \ldots \\ \lambda l_n.z\,(rec\,z)\,r_n, \\ \lambda(a_1\,\overline{x}).z\,(Rec\,z)\,(a_1\,\overline{z\,(rec\,z)\,x}), \\ \ldots \\ \lambda(a_m\,\overline{x}).z\,(Rec\,z)\,(a_m\,\overline{z\,(rec\,z)\,x}), \end{pmatrix} \wr \lambda(Rec\,z).\mathsf{first}\begin{pmatrix} \lambda l_1.z\,(rec\,z)\,r_1, \\ \ldots \\ \lambda l_n.z\,(rec\,z)\,r_n, \\ \lambda y.y \end{pmatrix}$$

Pour évaluer un terme t selon le système $\mathcal{R}$, on évalue le ρ-terme $[\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,t$. Notons que cette représentation applique en fait une stratégie de réécriture bien précise. Dans les langages comme Maude [CDE+03] ou Obj* [FN96], elle correspond à la stratégie locale `f (0 1 2 ...n)` pour toute constante f d'arité n, doublée d'un ordre sur les règles.

En effet, les règles sont d'abord essayées en tête des termes puisque $rec\,z$ est le seul motif qui filtre $rec\,[\![\mathcal{R}]\!]$. Si aucune règle ne convient, les termes $\lambda(a_i\,\overline{x}).\ldots$ déclenchent l'évaluation dans les sous-termes immédiats. Ensuite, quand ces sous-termes ne sont plus réductibles en tête, l'identité leur est appliquée, et le terme $[\![\mathcal{R}]\!]\,(Rec\,[\![\mathcal{R}]\!])$ relance une réduction de tête. Quand le terme est en forme normale, l'identité s'applique et tous les $[\![\mathcal{R}]\!]$ s'éliminent. Montrons que cette encodage est correct et complet pour tout TRS convergent (autrement dit confluent et terminant).

Théorème 37 (Correction et complétude de l'encodage des TRS [CLW03])

1. *Pour tout TRS $\mathcal{R}$, pour tous termes algébriques t et t',*

$$[\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,[\![t]\!] \mapsto_{\rho\delta}^{\mathsf{stk}} [\![t']\!] \quad \Rightarrow \quad t \twoheadrightarrow_{\mathcal{R}} t' \quad \textit{et } t' \textit{ est en forme } \mathcal{R}\textit{-normale.}$$

2. *Pour tout TRS convergent $\mathcal{R}$, pour tous termes algébriques t et t' tels que t' est en forme $\mathcal{R}$-normale,*

$$t \twoheadrightarrow_{\mathcal{R}} t' \quad \Rightarrow \quad [\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,[\![t]\!] \mapsto_{\rho\delta}^{\mathsf{stk}} [\![t']\!]$$

En particulier, la seconde propriété assure que $[\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,[\![t]\!]$ est faiblement normalisant pour tout terme algébrique t.

Démonstration :

1. $[\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,[\![t]\!] \mapsto_{\rho\delta}^{\mathsf{stk}} [\![t']\!] \;\Rightarrow\; t \twoheadrightarrow_{\mathcal{R}} t'$

   Définissons une fonction d'effacement $|\ |$ qui va supprimer tous les sous-termes issus de l'encodage de $\mathcal{R}$ et permet ainsi de récupérer le terme algébrique courant. Il suffit pour cela de ne garder que des constantes en tête des applications :

$$\begin{array}{rcll}
|A\,A_1\,\ldots\,A_n| & \triangleq & A\,|A_1|\,\ldots\,|A_n| & \text{si } A \text{ est une constante} \\
|A\,A_1\,\ldots\,A_n| & \triangleq & |A|\,|A_1|\,\ldots\,|A_n| & \text{si } A \text{ est une structure} \\
|A\,A_1\,\ldots\,A_n| & \triangleq & |A_n| & \text{sinon} \\
|A \wr B| & \triangleq & |A| & \text{si } A \not\mapsto_{\rho\delta}^{\mathsf{stk}} \mathsf{stk} \\
|A \wr B| & \triangleq & |B| & \text{sinon}
\end{array}$$

   Dans un terme A apparaissant le long de la réduction $[\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,[\![t]\!] \mapsto_{\rho\delta}^{\mathsf{stk}} [\![t']\!]$, on peut distinguer :
   - les ρ-réductions internes à $[\![\mathcal{R}]\!]$ (avec les motifs $rec\,z$), qui ne modifient pas $|A|$;
   - les δ-réductions, les $\overline{\mathsf{stk}}$-réductions ainsi que les ρ-réductions pour les abstractions du type $\lambda(a_i\,\overline{x}).z\,(Rec\,z)\,(a_i\,z\,(rec\,z)\,x)$, qui sont chargées de distribuer $[\![\mathcal{R}]\!]$ dans les différents sous-termes et d'élaguer les échecs de filtrage. Encore une fois, ces réductions ne modifient pas $|A|$;
   - les ρ-réductions pour les abstractions $\lambda l_i.z\,(rec\,z)\,r_i$ correspondant aux règles de $\mathcal{R}$, qui sont celles qui nous intéressent. Supposons donc que dans A on a un radical de la forme $(\lambda l_i.z\,(rec\,z)\,r_i)\,u$ pour un sous-terme u de $|A|$. Si la ρ-réduction peut effectivement avoir lieu, $A \mapsto_p A'$ et on a $u \equiv l_i\theta$ pour une certaine substitution θ, d'où $(\lambda l_i.z\,(rec\,z)\,r_i)\,u \mapsto_p z\,(rec\,z)\,r_i\theta$. Par conséquent $|A'|$ est exactement $|A|$ dans lequel u a été remplacé par $r_i\theta$, ce qui correspond bien à utiliser la règle $l_i \to r_i$ sur u.
     On en conclut donc que les $|A|$ distincts apparaissant le long du chemin de réduction $[\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,[\![t]\!] \mapsto_{\rho\delta}^{\mathsf{stk}} [\![t']\!]$ constituent un chemin de réécriture de $|[\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,[\![t]\!]| = t$ à $|[\![t']\!]| = t'$.

   Enfin, $[\![t']\!]$ étant un terme algébrique obtenu par réduction de $[\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,[\![t]\!]$, toutes les instances de $[\![\mathcal{R}]\!]$ ont été supprimées. Ceci n'est possible que si l'identité $\lambda y.y$ a été atteinte dans tous les termes $\mathsf{first}(\lambda l_1.z\,(rec\,z)\,r_1, \ldots, \lambda y.y)$, ce qui assure qu'aucune règle de $\mathcal{R}$ ne pouvait être appliquée nulle part, c.-à-d. que t' est en forme $\mathcal{R}$-normale.

2. $t \twoheadrightarrow_{\mathcal{R}} t' \;\Rightarrow\; [\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,[\![t]\!] \mapsto_{\rho\delta}^{\mathsf{stk}} [\![t']\!]$ si $\mathcal{R}$ est convergent.

   On considère l'ordre suivant sur les termes : $u \preceq t$ si et seulement si il existe un contexte $C\Box$ tel que $t \twoheadrightarrow_{\mathcal{R}} C[u]$ (autrement dit u est un sous-terme d'un réduit de t). Il n'est pas difficile de se convaincre que $\preceq$ est bien réflexif et transitif ; son antisymétrie est due à la

terminaison de $\mathcal{R}$: si $u \preceq t$ et $t \preceq u$, alors il existe deux contextes $C\Box$ et $C'\Box$ tels que $t \twoheadrightarrow_{\mathcal{R}} C[u] \twoheadrightarrow_{\mathcal{R}} C[C'[t]]$, ce qui n'est possible que pour $C\Box = \Box = C'\Box$ et pour un chemin de réécriture de longueur nulle, d'où $t \equiv u$.

De plus, pour terme t, il existe un nombre fini de termes u tels que $t \twoheadrightarrow_{\mathcal{R}} u$ (la longueur des dérivations est bornée et le nombre de réductions possibles à partir d'un terme donné est fini). Comme chacun de ces termes n'admet qu'un nombre fini de sous-termes, le nombre de termes u tels que $u \preceq t$ est fini également : il est alors immédiat que $\preceq$ est un ordre bien fondé.

Montrons donc par récurrence Noetherienne sur $\preceq$ que pour tout terme t, le ρ-terme $[\![\mathcal{R}]\!]\,(rec[\![\mathcal{R}]\!])\,[\![t]\!]$ se réduit à la forme $\mathcal{R}$-normale de t (qui est bien définie puisque $\mathcal{R}$ est convergent).

(Cas de base) Les termes minimaux sont les constantes a d'arité nulle qui ne sont membres gauches d'aucune règle de $\mathcal{R}$. Par conséquent, la seule abstraction de $[\![\mathcal{R}]\!]$ qui s'applique à une telle constante est $\lambda a.z\,(Rec\,z)\,a$, et donc

$$[\![\mathcal{R}]\!]\,(rec[\![\mathcal{R}]\!])\,a \mapsto^{\mathsf{stk}}_{\rho\delta} [\![\mathcal{R}]\!]\,(Rec[\![\mathcal{R}]\!])\,a$$

Dans ce second terme, il n'y a plus que les règles de $\mathcal{R}$ (qui ne s'appliquent pas à a) et l'identité, donc on obtient bien a qui est sa propre forme normale.

(Cas inductif) Supposons donc que pour tout u tel que $u \prec t$, le terme $[\![\mathcal{R}]\!]\,(rec[\![\mathcal{R}]\!])\,[\![u]\!]$ se réduit à la forme $\mathcal{R}$-normale de u. Deux cas sont à distinguer :

Si une règle de $\mathcal{R}$ s'applique en tête de t alors c'est la première de ces règles qui sera sélectionnée dans $[\![\mathcal{R}]\!]$, et donc $[\![\mathcal{R}]\!]\,(rec[\![\mathcal{R}]\!])\,[\![t]\!] \mapsto^{\mathsf{stk}}_{\rho\delta} [\![\mathcal{R}]\!]\,(rec[\![\mathcal{R}]\!])\,[\![u]\!]$ tel que $t \to_{\mathcal{R}} u$ (en exactement un pas). En particulier $u \prec t$ et donc par hypothèse de récurrence $[\![\mathcal{R}]\!]\,(rec[\![\mathcal{R}]\!])\,[\![u]\!] \mapsto^{\mathsf{stk}}_{\rho\delta} [\![u']\!]$ qui est la forme normale de u et donc également la forme normale de t.

Sinon aucune des abstractions représentant les règles de $\mathcal{R}$ ne s'applique. Le terme t s'écrit $f(\overline{t_i})$ (où le vecteur de sous-termes $\overline{t_i}$ est éventuellement vide), et donc la règle correspondant à la constante f s'applique :

$$[\![\mathcal{R}]\!]\,(rec[\![\mathcal{R}]\!])\,[\![t]\!] \mapsto^{\mathsf{stk}}_{\rho\delta} [\![\mathcal{R}]\!]\,(Rec[\![\mathcal{R}]\!])\,(f\,\overline{[\![\mathcal{R}]\!]\,(rec[\![\mathcal{R}]\!])\,t_i})$$

Chacun des t_i est un sous-terme strict de t donc $t_i \prec t$, d'où par hypothèse de récurrence $[\![\mathcal{R}]\!]\,(rec[\![\mathcal{R}]\!])\,[\![t_i]\!] \mapsto^{\mathsf{stk}}_{\rho\delta} [\![t'_i]\!]$ où t'_i est la forme $\mathcal{R}$-normale de t_i.

Si $f(\overline{t'_i})$ est en forme $\mathcal{R}$-normale, alors aucune des abstractions représentant les règles ne s'applique, et c'est l'identité qui est appliquée : $[\![\mathcal{R}]\!]\,(Rec[\![\mathcal{R}]\!])\,(f\,\overline{[\![t'_i]\!]}) \mapsto^{\mathsf{stk}}_{\rho\delta} f\,\overline{[\![t'_i]\!]}$ qui est la forme normale de t.

Si une règle de $\mathcal{R}$ s'applique à $f(\overline{t'_i})$ pour donner un terme u, l'abstraction correspondante s'applique et on a donc $[\![\mathcal{R}]\!]\,(Rec[\![\mathcal{R}]\!])\,(f\,\overline{[\![t'_i]\!]}) \mapsto^{\mathsf{stk}}_{\rho\delta} [\![\mathcal{R}]\!]\,(rec[\![\mathcal{R}]\!])\,[\![u]\!]$ avec $u \prec t$. Par hypothèse de récurrence, $[\![\mathcal{R}]\!]\,(rec[\![\mathcal{R}]\!])\,[\![u]\!] \mapsto^{\mathsf{stk}}_{\rho\delta} [\![u']\!]$ qui est la forme normale de u et donc également la forme normale de t.

□

Pour accompagner cette démonstration, on peut montrer des contre-exemples à la complétude dans les cas où le TRS $\mathcal{R}$ n'est pas convergent.

1. Si $\mathcal{R}$ n'est pas confluent : pour $\mathcal{R} = \{\mathsf{random} \to 0, \mathsf{random} \to 1\}$, on a la réduction $[\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,\mathsf{random} \mapsto^{\mathsf{stk}}_{\rho\delta} 0$ mais par contre $\mathsf{random} \to_{\mathcal{R}} 1$ n'est pas représenté dans le ρ-calcul.
2. Si $\mathcal{R}$ n'est pas terminant : pour

$$\mathcal{R} = \begin{cases} g(x) & \to & g(Sx) \\ g(x) & \to & 0 \\ f(0) & \to & 0 \end{cases}$$

on a $\quad f(g(0)) \to_{\mathcal{R}} \ldots \to_{\mathcal{R}} f(g(S^n 0)) \to_{\mathcal{R}} f(0) \to_{\mathcal{R}} 0$
mais $\quad [\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,(f(g\,0)) \mapsto^{\mathsf{stk}}_{\rho\delta} \ldots \mapsto^{\mathsf{stk}}_{\rho\delta} [\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,f([\![\mathcal{R}]\!]\,(rec\,[\![\mathcal{R}]\!])\,g(S^n 0)) \mapsto^{\mathsf{stk}}_{\rho\delta} \ldots$

Nous avons ici utilisé l'opérateur first de façon assez restrictive, mais des combinaisons plus subtiles des diverses règles du TRS permettent d'encoder d'autres stratégies.

Exemple 29 (Encodage de systèmes de calcul)
On peut modifier $[\![\mathcal{R}]\!]$ pour simuler les stratégies de réduction suivantes :

1. *En plaçant les règles de propagation $a_i\,\overline{x} \to z\,(Rec\,z)\,(a_i\,\overline{z\,(rec\,z)\,x})$, en début de* first*, on obtient une stratégie* innermost *: les règles sont d'abord appliquées sur les sous-termes les plus profonds.*
2. *Si au lieu de mettre toutes les règles dans un unique* first*, on hiérarchise un peu plus leur application, on peut obtenir un contrôle très fin sur l'application de $\mathcal{R}$.*

Conjecture 1 (Encodage des stratégies d'ELAN)
Ce type d'encodage permet de représenter toutes les stratégies à la ELAN :
- *les stratégies élémentaires que sont* `id`, `fail` *et les règles ont une représentation immédiate ;*
- *la stratégie* $\texttt{first}(S_1, \ldots, S_n)$ *se représente avec notre terme* first *;*
- *la stratégie* $\texttt{try}(S)$ *s'écrit simplement* $\mathsf{first}(S, \lambda y.y)$ *;*
- *la stratégie* $\texttt{repeat}*(S)$ *se représente à l'aide d'un point fixe supplémentaire pour itérer la stratégie, couplé avec un* try *pour arrêter l'itération quand la stratégie S ne s'applique plus.*
- ...

Enfin, signalons que l'encodage $[\![\mathcal{R}]\!]$ est typable dans $\rho 1$ si $\mathcal{R}$ est bien typé, autrement dit si les membres gauches et droits de toutes les règles de $\mathcal{R}$ sont typables et ont le même type. Le type des constantes rec et Rec est alors $(\kappa \to \tau \to \tau) \to \kappa$ si $\mathcal{R}$ manipule des données de type τ.

4.3 À propos de la sémantique du système de types

Pour bien comprendre ce qui fait que les systèmes de types étudiés dans ce chapitre n'assurent pas la normalisation forte, et puisque celle-ci est fortement liée à la consistance des systèmes de déduction, étudions le système $\rho 1$ à la lumière de l'isomorphisme de Curry-Howard-de Bruijn.

Tout d'abord, on peut penser que les types des constantes de Σ jouent un rôle important. En effet, à cause de la règle (CONST), tous les types apparaissant dans Σ devraient être vus comme des axiomes dans la logique correspondante. Or la constante f utilisée dans le terme ω_f a pour type $(\iota \to \kappa) \to \iota$, qui ne correspond pas à une tautologie de la logique propositionnelle. On peut donc espérer recouvrer la normalisation forte en restreignant les types autorisés dans la signature. Étudions trois candidats pour cette restriction.

1. Une première idée est de n'autoriser que des tautologies : cependant, en considérant une variante de ω_f dans lequel toutes les occurrences de ι sont remplacées par $\iota \to \iota$ et toutes les occurrences de κ par $\kappa \to \kappa$, la constante f a alors pour type $((\iota \to \iota) \to (\kappa \to \kappa)) \to (\iota \to \iota)$ qui bien est une tautologie, mais la dérivation de typage et les réductions sont les mêmes. Cette restriction est donc insuffisante.
2. On pourrait également considérer une restriction comme la condition de *positivité* due à N. P. Mendler [Men87] et qui est utilisée dans le calcul des constructions inductives [Wer94] : pour toute constante f, si $f : \tau_1 \to \ldots \to \tau_n \to \iota$ alors chacun des τ_i doit s'écrire $\tau_{i1} \to \ldots \tau_{in} \to \kappa$ où seul κ peut éventuellement être ι mais ι n'apparaît dans aucun τ_{ij}.
   Avec une telle restriction, l'exemple de point fixe que nous avons donné ne peut être typé. Cependant, on ne peut transposer directement au ρ-calcul les résultats du calcul des constructions inductives : en effet, dans CIC toutes les constantes sont vues comme des constructeurs de types qui doivent être donnés au moment de la déclaration de ce type, alors qu'en ρ-calcul tout est donné par la signature sur laquelle on a en quelque sorte moins de contrôle. Considérons la variante suivante de ω_f :
   $$\begin{aligned} \omega_{fg} &\triangleq \lambda(f\,x).\Big((\lambda(g\,y).(y\,(f\,(g\,y))))\,x\Big) \\ \omega_{fg}\,(f\,(g\,\omega_{fg})) &\equiv \lambda(f\,x).\Big((\lambda(g\,y).(y\,(f\,(g\,y))))\,x\Big)\,(f\,(g\,\omega_{fg})) \\ &\mapsto_\rho (\lambda(g\,y).(y\,(f\,(g\,y))))\,(g\,\omega_{fg}) \\ &\mapsto_\rho \omega_{fg}\,(f\,(g\,\omega_{fg})) \\ &\mapsto_\rho \ldots \end{aligned}$$
   Ce terme est à nouveau typable avec la signature $\{f{:}\kappa \to \iota, g{:}(\iota \to \kappa) \to \kappa\}$ dans laquelle chacune des constantes respecte la condition de positivité, mais en calcul des constructions inductives cette situation correspond à des déclarations mutuellement dépendantes pour les types ι et κ.
3. Mentionnons enfin qu'un système de types simples très similaire à $\rho 1$ et assurant la normalisation forte a été étudié pour une version antérieure du ρ-calcul [CK01]. Le cadre est cependant assez différent de celui que nous étudions ici, puisque les règles de réduction du calcul ne sont pas les mêmes. En particulier, cette version comportait la réduction
   $$(Congruence) \qquad (f\,A_1\,\ldots\,A_n)\,(f\,B_1\,\ldots\,B_n) \;\mapsto\; f\,(A_1\,B_1)\,\ldots\,(A_n\,B_n)$$
   qui a une incidence très forte sur le système de types, puisqu'elle impose une surcharge de toutes les constantes : si $f : (\tau_1 \to \sigma_1) \to \ldots (\tau_n \to \sigma_n) \to (\tau \to \sigma)$, alors $f : \tau_1 \to \ldots \tau_n \to \tau$ et $f : \sigma_1 \to \ldots \sigma_n \to \sigma$, et réciproquement. La signature doit alors affecter à chaque constante une famille de types.
   Par ailleurs, cette surcharge entraîne immédiatement la stabilité des candidats de réductibilité par application des constantes, ce qui facilite largement la démonstration de leur correction.

Une autre cause d'inconsistance, probablement plus significative, est la règle de typage de l'abstraction (les autres règles étant identiques à celles du λ-calcul). Dans le λ-calcul simplement typé, en effaçant les termes de la règle (ABS), on obtenait une règle d'introduction pour l'implication. Ici on trouve :

$$\frac{\Gamma, \Delta \vdash_\Sigma P : \sigma \qquad \Gamma, \Delta \vdash_\Sigma A : \tau}{\Gamma \vdash_\Sigma \lambda P{:}\Delta.A : \sigma \to \tau} \qquad\qquad \frac{\Gamma, \Delta \vdash \phi \qquad \Gamma, \Delta \vdash \psi}{\Gamma \vdash \phi \Rightarrow \psi}$$

Il y a donc un double problème : d'une part, on construit une implication avec les deux propositions ϕ et ψ qui devraient plutôt être en conjonction, puisqu'elles sont démontrables dans un même contexte ; d'autre part, on décharge une partie Δ du contexte sans en garder de trace dans la conclusion $\Gamma \vdash \phi \Rightarrow \psi$.

Partant de ce constat, on peut essayer de construire une règle d'abstraction qui ait un sens au niveau logique. La proposition ϕ peut être omise, dans la mesure où elle correspond au typage de P, qui n'est présent que pour assurer la bonne formation des termes. Par ailleurs, pour décharger le contexte Δ, il faut le faire apparaître dans la conclusion, à gauche d'une implication. On aurait donc une règle du genre

$$\frac{\Gamma, x_1{:}\sigma_1, \ldots x_n{:}\sigma_n \vdash_\Sigma P : \sigma \qquad \Gamma, x_1{:}\sigma_1, \ldots x_n{:}\sigma_n \vdash_\Sigma A : \tau}{\Gamma \vdash_\Sigma \lambda P{:}(x_1{:}\sigma_1, \ldots x_n{:}\sigma_n).A : \bigwedge_{i=1}^{n} \sigma_i \to \tau} \qquad \frac{\Gamma, \phi_1, \ldots, \phi_n \vdash \psi}{\Gamma \vdash \bigwedge_{i=1}^{n} \phi_i \Rightarrow \psi}$$

Cette règle est admissible du point de vue logique mais n'est pas tout-à-fait adaptée pour le typage : il faudrait modifier également la règle d'application pour tenir compte du nouveau typage des motifs. De plus, la conjonction $\bigwedge_{i=1}^{n} \sigma_i$ ne donne aucune information sur la façon dont les différents σ_i sont organisés dans le motif P. Pour peu que l'argument comporte des sous-termes ayant les mêmes types mais soit construit différemment, il y a donc un risque d'instancier des variables avec des termes n'ayant pas le bon type. Il faut donc enrichir le langage de types de façon plus importante : le chapitre suivant est consacré à l'étude d'une solution utilisant des types dépendants.

5

P^2TS et normalisation dans $\rho_{\rightarrow}$ et ρP

De la même façon que les systèmes de types du chapitre précédent étaient inspirés des systèmes de types simple et polymorphe du λ-calcul, nous allons maintenant étudier les systèmes de types purs à motifs (P^2TS pour Pure Pattern Type Systems), introduits par G. Barthe, H. Cirstea, C. Kirchner et L. Liquori [BCKL03], qui sont une classe de systèmes de types pour le ρ-calcul inspirés des PTS.

Nous commencerons par donner une description légèrement restreinte des P^2TS, que nous comparerons avec la formulation originale pour préciser et justifier les restrictions choisies. Nous démontrerons ensuite la normalisation forte du système simplement typé, grâce à une traduction vers un système du λ-cube. Enfin nous étendrons ce résultat au système avec types dépendants par un mécanisme classique d'effacement des types dépendants. La démonstration de normalisation forte dans $\rho_{\rightarrow}$ a été publiée dans [Wac04].

5.1 Systèmes de types purs à motifs (P^2TS)

5.1.1 Définitions

Tout comme dans le cadre des PTS, nous ne ferons ici plus de différence syntaxique entre les termes et les types : tous les pseudo-termes sont donnés au même niveau, et c'est le système de types qui fournira une classification si elle a lieu d'être. On ajoute à la syntaxe une Π-abstraction qui servira à représenter les types dépendants, ainsi qu'une construction $[\ll]$ appelée *contrainte de filtrage* sur laquelle nous reviendrons. Comme pour les systèmes de types $\rho 1$ et $\rho_\forall$, le typage est *à la Church* : les abstractions sont décorées par un contexte Δ qui donne les types des variables libres du motif. On remarquera que la contrainte de filtrage est un lieur et est donc aussi indexée par un Δ.

$$\begin{array}{rcl} \mathcal{P} & ::= & \mathcal{X} \mid \mathcal{K}\,\overline{\mathcal{P}} \\ \mathcal{T}_\rho & ::= & \mathcal{S} \mid \mathcal{X} \mid \mathcal{K} \mid \mathcal{T}_\rho\,\mathcal{T}_\rho \mid \lambda\mathcal{P}{:}\Delta.\mathcal{T}_\rho \mid \Pi\mathcal{P}{:}\Delta.\mathcal{T}_\rho \mid [\mathcal{P} \ll_\Delta \mathcal{T}_\rho]\mathcal{T}_\rho \mid \mathcal{T}_\rho \wr \mathcal{T}_\rho \end{array}$$

Les motifs seront supposés linéaires dans toute cette section ; dans la section 5.2, lors de la traduction en λ-calcul, cette restriction pourra être levée. Lorsque le motif est réduit à une variable, par similarité avec les PTS et pour alléger les notations, nous écrirons parfois $\lambda x{:}C.A$ au lieu de $\lambda x{:}(x{:}C).A$. Lorsque les types sont évidents ou lorsque seules les réductions d'un terme nous intéressent, nous omettrons les contextes Δ pour alléger les notations. Les notions

de variable libre ou liée sont adaptées en conséquence :

$$\begin{aligned}
\mathcal{FV}(x) &= \{x\} \\
\mathcal{FV}(a) &= \emptyset \\
\mathcal{FV}(A\,B) &= \mathcal{FV}(A) \cup \mathcal{FV}(B) \\
\mathcal{FV}(\lambda P{:}\Delta.A) &= (\mathcal{FV}(A) \cup \mathcal{FV}(P) \cup \mathcal{FV}(\Delta)) \setminus \mathcal{D}om(\Delta) \\
\mathcal{FV}(\Pi P{:}\Delta.A) &= (\mathcal{FV}(A) \cup \mathcal{FV}(P) \cup \mathcal{FV}(\Delta)) \setminus \mathcal{D}om(\Delta) \\
\mathcal{FV}([P \ll_\Delta A]B) &= ((\mathcal{FV}(B) \cup \mathcal{FV}(P) \cup \mathcal{FV}(\Delta)) \setminus \mathcal{D}om(\Delta)) \cup \mathcal{FV}(A) \\
\mathcal{FV}(A \wr B) &= \mathcal{FV}(A) \cup \mathcal{FV}(B) \\
\mathcal{FV}(\Delta, x{:}A) &= \mathcal{FV}(\Delta) \cup \mathcal{FV}(A)
\end{aligned}$$

$$\begin{aligned}
\mathcal{BV}(x) &= \emptyset \\
\mathcal{BV}(a) &= \emptyset \\
\mathcal{BV}(A\,B) &= \mathcal{BV}(A) \cup \mathcal{BV}(B) \\
\mathcal{BV}(\lambda P{:}\Delta.A) &= \mathcal{BV}(A) \cup \mathcal{BV}(P) \cup \mathcal{D}om(\Delta) \cup \mathcal{BV}(\Delta) \\
\mathcal{BV}(\Pi P{:}\Delta.A) &= \mathcal{BV}(A) \cup \mathcal{BV}(P) \cup \mathcal{D}om(\Delta) \cup \mathcal{BV}(\Delta) \\
\mathcal{BV}([P \ll_\Delta A]B) &= \mathcal{BV}(A) \cup \mathcal{BV}(P) \cup \mathcal{BV}(B) \cup \mathcal{D}om(\Delta) \cup \mathcal{BV}(\Delta) \\
\mathcal{BV}(A \wr B) &= \mathcal{BV}(A) \cup \mathcal{BV}(B) \\
\mathcal{BV}(\Delta, x{:}A) &= \mathcal{BV}(\Delta) \cup \mathcal{BV}(A)
\end{aligned}$$

Encore une fois, on raisonnera généralement modulo α-conversion.

Les règles de typage sont données dans la figure 5.1, ainsi que les règles permettant de vérifier le jugement $\Sigma\ sig$ de bonne formation de la signature. Comme pour les PTS, elles sont paramétrées par un triplet $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ qui donne respectivement l'ensemble des sortes, l'ensemble des axiomes $s_1 : s_2$ et l'ensemble des règles de produit admissibles.

Les règles elles-mêmes sont très similaires à celles des PTS, à ceci près que le motif P est utilisé dans les types dépendants. Ceci rend le typage d'une application plus difficile : dans les PTS, on pouvait se contenter de substituer l'argument B à la variable x liée dans le type dépendant. Ici, il n'existe pas toujours un filtre entre le motif P et l'argument B : il faut donc pouvoir définir des préradicaux dans les types. Ceci justifie *a posteriori* la construction $[\ll]$ qui est utilisée à cette fin. Elle est sensiblement équivalente à l'une des propositions de R. Bloo, F. Kamareddine, T. Laan et R. Nederpelt pour inclure une notion de Π-réduction dans le λ-cube. Leurs autres solutions consistent à ne considérer qu'une préservation faible du type [KN96] ou à étudier la Π-réduction en conjonction avec des définitions et des substitutions explicites [KLN03]. La règle (Appl) a pour prémisse $\Gamma \vdash_\Sigma [P \ll_\Delta B]C : s$ qui force toutes les vérifications de correction des types à être faites plus haut dans la dérivation de typage. Dans les PTS, ces vérifications étaient faites directement dans la règle de typage de l'application.

Les règles (Sig) et (WeakΣ) assurent la correction de la signature ; elles sont directement empruntées à LF. Enfin, la règle (Match) peut être vue comme une règle dérivée d'une abstraction immédiatement suivie d'une application, ce qui explique que c'est la sorte s_2 et non pas s_3 qui est utilisée dans sa conclusion.

Pour éliminer les contraintes de filtrage dans le cas où un filtre existe, on conserve les mêmes

$$(\textsc{Sig})\ \frac{}{\emptyset\ sig} \qquad (\textsc{WeakΣ})\ \frac{\Sigma\ sig \quad \vdash_\Sigma A : s \quad f \notin \mathcal{D}om(\Sigma)}{\Sigma, f{:}A\ sig}\ (s \in \mathcal{S})$$

$$(\textsc{Axiom})\ \frac{\Sigma\ sig}{\vdash_\Sigma s_1 : s_2}\ ((s_1,s_2) \in \mathcal{A}) \qquad (\textsc{Const})\ \frac{\Sigma\ sig \quad f{:}A \in \Sigma}{\vdash_\Sigma f : A}$$

$$(\textsc{Var})\ \frac{\Gamma \vdash_\Sigma A : s}{\Gamma, x{:}A \vdash_\Sigma x : A}\ (x \notin \Gamma, s \in \mathcal{S}) \qquad (\textsc{Struct})\ \frac{\Gamma \vdash_\Sigma A : C \qquad \Gamma \vdash_\Sigma B : C}{\Gamma \vdash_\Sigma A \wr B : C}$$

$$(\textsc{WeakΓ})\ \frac{\Gamma \vdash_\Sigma A : B \quad \Gamma \vdash_\Sigma C : s \quad x \notin \mathcal{D}om(\Gamma)}{\Gamma, x{:}C \vdash_\Sigma A : B}\ (x \notin \Gamma, s \in \mathcal{S})$$

$$(\textsc{Conv})\ \frac{\Gamma \vdash_\Sigma A : B \qquad \Gamma \vdash_\Sigma C : s}{\Gamma \vdash_\Sigma A : C}\ \left(\begin{array}{c} s \in \mathcal{S} \\ B =_{\beta\delta} C \end{array}\right)$$

$$(\textsc{Abs})\ \frac{\Gamma, \Delta \vdash_\Sigma B : C \qquad \Gamma \vdash_\Sigma \Pi P{:}\Delta.C : s}{\Gamma \vdash_\Sigma \lambda P{:}\Delta.B : \Pi P{:}\Delta.C}\ (s \in \mathcal{S})$$

$$(\textsc{Appl})\ \frac{\Gamma \vdash_\Sigma A : \Pi P{:}\Delta.C \qquad \Gamma \vdash_\Sigma [P \ll_\Delta B]C : s}{\Gamma \vdash_\Sigma A\,B : [P \ll_\Delta B]C}\ (s \in \mathcal{S})$$

$$(\textsc{Prod})\ \frac{\Gamma, \Delta \vdash_\Sigma P : A \qquad \Gamma, \Delta \vdash_\Sigma A : s_1 \qquad \Gamma, \Delta \vdash_\Sigma C : s_2}{\Gamma \vdash_\Sigma \Pi P{:}\Delta.C : s_3}\ \left(\begin{array}{c} \mathcal{D}om(\Delta) = \mathcal{FV}(P) \\ (s_1,s_2,s_3) \in \mathcal{R} \end{array}\right)$$

$$(\textsc{Match})\ \frac{\Gamma, \Delta \vdash_\Sigma P : A \quad \Gamma \vdash_\Sigma B : A \quad \Gamma, \Delta \vdash_\Sigma A : s_1 \quad \Gamma, \Delta \vdash_\Sigma C : s_2}{\Gamma \vdash_\Sigma [P \ll_\Delta B]C : s_2}\ \left(\begin{array}{c} \mathcal{D}om(\Delta) = \mathcal{FV}(P) \\ \exists s_3,\ (s_1,s_2,s_3) \in \mathcal{R} \end{array}\right)$$

FIG. 5.1 – Règles de typage des P^2TS

réductions que pour le ρ-calcul non typé auxquelles on ajoute la σ-réduction :

$$\begin{array}{rcll}(\lambda P{:}\Delta.A)\,B & \to_\sigma & A\theta_1 \wr \ldots \wr A\theta_n & \text{si } \mathcal{S}ol(P \ll B) = \{\theta_1, \ldots, \theta_n\} \text{ avec } n > 0\\ [P \ll_\Delta B]A & \to_\sigma & A\theta_1 \wr \ldots \wr A\theta_n & \text{si } \mathcal{S}ol(P \ll B) = \{\theta_1, \ldots, \theta_n\} \text{ avec } n > 0\\ (A \wr B)C & \to_\delta & A\,C \wr B\,C & \end{array}$$

Comme les contraintes de filtrage n'apparaissent que dans les types, la σ-réduction n'est en fait utilisée que dans la règle (CONV). On appellera contrainte *irrésoluble* un terme $[P \ll B]C$ tel que P ne subsume pas B (autrement dit un σ-préradical qui n'est pas un σ-radical). De même que pour les ρ-préradicaux, une contrainte irrésoluble $[P \ll B]C$ peut devenir résoluble après instanciation ou réduction de B.

On appellera $\to_{\rho\sigma\delta}$ la relation $\to_\sigma \cup \to_\rho \cup \to_\delta$, et comme d'habitude $\mapsto_{\rho\sigma\delta}$, $\twoheadrightarrow_{\rho\sigma\delta}$ et $=_{\rho\sigma\delta}$ seront ses relations dérivées.

Enfin, signalons que, par imitation du λ-calcul, on s'intéressera plus particulièrement aux systèmes du ρ-cube.

Exemple 30 (Les systèmes du ρ-cube)
Dans le cas où $\mathcal{S} = \{*, \Box\}$ et où le seul axiome est $\mathcal{A} = \{(*, \Box)\}$, en imposant que tout triplet de $\mathcal{R}$ soit de la forme (s_1, s_2, s_2) et que $\mathcal{R}$ contienne toujours au moins $(*, *, *)$, on définit une collection de huit systèmes de types. Nous verrons que chacun est une extension conservative du système correspondant dans le λ-cube.

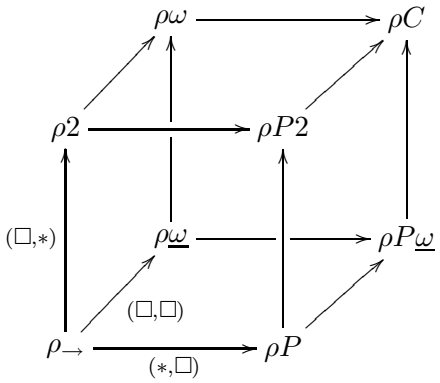


FIG. 5.2 – Le ρ-cube

Il nous faut cependant remarquer ici que, contrairement au λ-calcul, les systèmes sans types dépendants ne sont pas équivalents à leur formulation avec des types flèches.

Remarque 7 (Non-équivalence des présentations)
Les systèmes de types $\rho 1$ et $\rho_{\to}$ ne sont pas équivalents.

Nous avons vu au chapitre précédent que le terme $f\,\omega_f = f\,(\lambda(f\,x){:}(x{:}\iota \to \kappa).(x\,(f\,x)))$ est typable dans $\rho 1$, ce qui n'est pas le cas dans $\rho_{\to}$. En effet, le type $\iota \to \kappa$ s'écrit ici $\Pi y{:}(y{:}\iota).\kappa$; posons alors $\Gamma = (z{:}\Pi y{:}(y{:}\iota).\kappa)$ et $\Delta = (x{:}\Pi y{:}(y{:}\iota).\kappa)$. Une dérivation de typage pour $f\,\omega_f$

devrait être de la forme suivante :

$$(\textsc{Appl})\ \frac{\vdash_\Sigma f : \Pi z{:}\Gamma.\iota \qquad (\textsc{Match})\ \dfrac{\Gamma \vdash_\Sigma z : \Pi y{:}(y{:}\iota).\kappa \qquad (\textsc{Abs})\ \dfrac{\Delta \vdash_\Sigma x(fx) : \kappa \quad \vdash_\Sigma \Pi(fx){:}\Delta.\kappa : s}{\vdash_\Sigma \omega_f : \Pi(fx){:}\Delta.\kappa}}{\vdash_\Sigma [z \ll_\Gamma \omega_f]\iota : *}}{\vdash_\Sigma f\,\omega_f : [z \ll_\Gamma \omega_f]\iota}$$

or la règle (Match) ne peut être appliquée correctement ici, car le motif z et l'argument ω_f n'ont pas les mêmes types : ils diffèrent par le motif de leur abstraction de tête, qui est y pour z et qui est $f\,x$ pour ω_f. Pour pouvoir appliquer cette règle il faudrait avoir le motif $f\,x$ dans le type de z, qui lui-même est issu du type de f. On obtiendrait donc une circularité, à savoir que f apparaîtrait dans son propre type, ce qui est impossible en vertu de la règle (WeakΣ).

La même remarque est valable pour les systèmes polymorphes : $\rho_\forall$ et $\rho 2$ ne sont pas équivalents.

5.1.2 Comparaison avec les P^2TS originaux

Passons en revue les différences entre cette présentation et celle de [BCKL03]. Les deux premières résident essentiellement dans la formulation du système de types et ne constituent pas un véritable changement dans la sémantique du calcul. La dernière est plus importante puisqu'elle concerne le filtrage, dont nous avons considéré une version assez restreinte.

Signature

Nous avons ici distingué le contexte, qui donne les types des variables libres, de la signature, qui donne les types des constantes. En plus d'une séparation claire des concepts, ceci assure notamment que tous les types des constantes sont clos, grâce à la règle (WeakΣ).

Dans la présentation initiale des P^2TS, constantes et variables sont déclarées dans un unique contexte, ce qui semble assurer un traitement uniforme de ces deux espèces de termes atomiques. Pourtant, les seules règles d'inférence permettant de charger ou de décharger des déclarations de types pour des constantes dans le contexte sont la règle (Start) de typage des atomes et la règle (Weak). Par conséquent, dans une dérivation de typage, toutes les déclarations de constantes peuvent être repoussées au niveau du typage des atomes, ce qui revient plus ou moins à avoir une signature séparée.

Contrainte de filtrage

L'utilisation que nous faisons ici de la construction $[\ll]$ est un peu plus limitée que dans la plupart des présentations récentes du ρ-calcul (typé ou non). En effet on trouve souvent les règles

$$\begin{aligned}(\lambda P{:}\Delta.A)\,B &\to_{\rho'} [P \ll_\Delta B]A\\ [P \ll_\Delta B]A &\to_\sigma A\theta_1 \wr \ldots \wr A\theta_n\end{aligned}$$

On remarquera cependant que, dans un tel calcul, il est toujours possible de placer les ρ'-réductions immédiatement avant les σ-réductions qui leur correspondent. Une séquence $\to_{\rho'} \to_\sigma$ correspond alors exactement à notre règle $\to_p$.

De plus, les contraintes de filtrage sont souvent considérées au niveau des termes, comme nous l'avons fait dans le système $\rho^{\ll}_\forall$. Il serait possible ici de faire de même, comme dans la version initiale des P^2TS, mais cela conduit à la non-unicité du typage : en effet, un terme comme $[P \ll B]C$ doit alors être typé par $[P \ll B]D$ où $\vdash_\Sigma C : D$, mais il y a alors ambiguïté avec la règle (Match) dans le cas où D est (convertible à) une sorte.

Classe de motifs et filtrage utilisés

Pour les raisons évoquées au chapitre 3 (filtrage d'ordre supérieur, confluence), nous ne considérons que des motifs algébriques linéaires (pas d'abstractions, pas de variables actives et une même variable n'apparaît qu'une fois). Notons que la confluence des P^2TS a été démontrée pour une classe de motifs plus large (une extension de RPC, voir définition 28). Cependant, le filtrage utilisé dans ce cas n'est pas un filtrage d'ordre supérieur mais un filtrage syntaxique étendu pour certaines formes d'abstractions, dont la sémantique exacte reste à préciser.

Par ailleurs, on remarquera que, dans la définition des variables libres pour les P^2TS, toutes les variables libres de P ne sont pas forcément liées par une abstraction : c'est Δ qui détermine quelles variables sont liées. Nous avons ensuite imposé $\mathcal{D}om(\Delta) = \mathcal{FV}(P)$ dans le système de types, alors que dans [BCKL03] cette condition est remplacée par $\mathcal{D}om(\Delta) \subseteq \mathcal{FV}(P)$. Dans ce cas il est possible d'instancier des variables dans les motifs, comme dans l'exemple suivant :

$$(\lambda x{:}(x{:}\iota).\lambda(f\,x\,y){:}(y{:}\iota).y)\,3 \;\mapsto_{\rho}\; \lambda(f\,3\,y){:}(y{:}\iota).y$$

Le terme $\lambda(f\,x\,y){:}(y{:}\iota).y$ représente donc un schéma de règle de réécriture $f(x,y) \rightarrow y$ dans laquelle x est un *paramètre* que l'on peut choisir arbitrairement.

Cette généralisation du paradigme est séduisante, mais elle pose plusieurs problèmes :

- l'instanciation d'une variable dans un motif peut créer un motif qui n'est pas conforme à la classe choisie initialement ;
- si une variable est instanciée par un terme comportant un radical, il faut envisager des réductions dans les motifs ;
- il est possible d'encoder un filtrage non linéaire.

Les deux premiers points peuvent se résoudre soit par le choix d'une classe de motifs très large (ce qui pose les problèmes de filtrage d'ordre supérieur évoqués au chapitre 3), soit par une stratégie de réduction très stricte.

Le dernier est plus problématique : comme nous l'avons vu sur l'exemple 22, un unique motif non linéaire peut remettre en cause la confluence du calcul, or celle-ci nous sera nécessaire pour étudier le système de types et démontrer la normalisation forte. À titre d'information, montrons comment encoder la règle $(d\,x\,x) \rightarrow e$:

$$\lambda(d\,x\,y){:}(x{:}\tau, y{:}\tau).((\lambda x{:}\emptyset.e)\,y)$$

Alors pour tous termes A et B,

$$\Big(\lambda(d\,x\,y){:}(x{:}\tau, y{:}\tau).((\lambda x{:}\emptyset.e)\,y)\Big)\,(d\,A\,B) \;\mapsto_{\rho}\; (\lambda A{:}\emptyset.e)\,B$$

qui se réduit à e si et seulement si $A =_{\rho\delta} B$. Ceci ne signifie pas qu'un tel calcul n'est pas confluent, puisqu'il faudrait également pouvoir définir des points fixes pour encoder totalement l'exemple de Klop (exemple 17) ; le problème se situe plutôt dans la circularité des dépendances entre les diverses propriétés que nous voulons démontrer sur le calcul.

5.1.3 Propriétés

Stipulons immédiatement que la confluence se démontre exactement comme dans le ρ-calcul non typé, à ceci près qu'il faut tenir compte des réductions dans les contextes, ainsi que de la σ-réduction, qui se comporte très similairement à la ρ-réduction.

Théorème 38 (Confluence dans les P^2TS [BCKL03])

Avec un filtrage syntaxique, la réduction $\mapsto_{\rho\sigma\delta}$ est confluente sur l'ensemble des pseudo-termes des P^2TS dont les motifs vérifient RPC.

Le lemme de génération a toujours la même forme, mais il faut raisonner modulo $=_{\beta\delta}$ pour tenir compte d'éventuelles applications de la règle (Conv).

Lemme 39 (Génération)
Dans tout P^2TS, pour tout jugement de typage dérivable, il est possible de reconstruire une dérivation en fonction de la forme de A :

si $\Gamma \vdash_\Sigma s : E$ *alors* $E =_{\beta\delta} s'$ *avec* $(s, s') \in \mathcal{A}$;

si $\Gamma \vdash_\Sigma x : E$ *alors* $E =_{\beta\delta} A$ *avec* $(x{:}A) \in \Gamma$ *et* $\exists s \in \mathcal{S},\ \Gamma \vdash_\Sigma A : s$;

si $\Gamma \vdash_\Sigma f : E$ *alors* $E =_{\beta\delta} A$ *avec* $(f{:}A) \in \Sigma$ *et* $\exists s \in \mathcal{S},\ \vdash_\Sigma A : s$;

si $\Gamma \vdash_\Sigma A \wr B : E$ *alors* $E =_{\beta\delta} C$ *avec* $\Gamma \vdash_\Sigma A : C$ *et* $\Gamma \vdash_\Sigma B : C$;

si $\Gamma \vdash_\Sigma \lambda P{:}\Delta.B : E$ *alors* $E =_{\beta\delta} \Pi P{:}\Delta.C$ *avec* $\Gamma, \Delta \vdash_\Sigma B : C$ *et* $\exists s \in \mathcal{S},\ \Gamma \vdash_\Sigma \Pi P{:}\Delta.C : s$;

si $\Gamma \vdash_\Sigma A\,B : E$ *alors* $E =_{\beta\delta} [P \ll_\Delta B]C$ *avec* $\Gamma \vdash_\Sigma A : \Pi P{:}\Delta.C$ *et* $\exists s \in \mathcal{S},\ \Gamma \vdash_\Sigma [P \ll_\Delta B]C : s$;

si $\Gamma \vdash_\Sigma \Pi P{:}\Delta.C : E$ *alors* $\exists (s_1, s_2, s_3) \in \mathcal{R},\ \exists A$ *tels que* $E =_{\beta\delta} s_3$ *avec* $\Gamma, \Delta \vdash_\Sigma P : A$ *et* $\Gamma, \Delta \vdash_\Sigma A : s_1$ *et* $\Gamma, \Delta \vdash_\Sigma C : s_2$;

si $\Gamma \vdash_\Sigma [P \ll_\Delta B]C : E$ *alors* $\exists (s_1, s_2, s_3) \in \mathcal{R},\ \exists A$ *tels que* $E =_{\beta\delta} s_2$ *avec* $\Gamma, \Delta \vdash_\Sigma P : A$ *et* $\Gamma \vdash_\Sigma B : A$ *et* $\Gamma, \Delta \vdash_\Sigma A : s_1$ *et* $\Gamma, \Delta \vdash_\Sigma C : s_2$.

Démonstration : Par inspection des règles de typage, en remarquant que les règles (Conv) et (WeakΓ) ne modifient pas le sujet du jugement de typage, et que les règles d'affaiblissement peuvent toujours être repoussées jusqu'au niveau du typage des atomes. □

Le lemme de correction assure que tous les types utilisés sont bien formés.

Lemme 40 (Correction)
Pour tous pseudo-termes A, B, pour tout contexte Γ, si $\Gamma \vdash_\Sigma A : B$ alors il existe une sorte $s \in \mathcal{S}$ telle que $\Gamma \vdash_\Sigma B : s$ ou $B \equiv s$.

Démonstration : Par récurrence structurelle sur une dérivation de $\Gamma \vdash_\Sigma A : B$. Les occurrences de (WeakΓ) peuvent être omises car elles ne modifient ni A ni B.

Si la dérivation se termine par une des règles (Conv), (Var), (Abs) ou (Appl), alors $\Gamma \vdash_\Sigma B : s$ est une prémisse immédiate.

Si la dérivation se termine par une des règles (Axiom), (Prod) ou (Match), alors $B \in \mathcal{S}$.

Si la dérivation se termine par une règle (Struct), on conclut par hypothèse de récurrence sur l'une quelconque des prémisses.

Si la dérivation se termine par une règle (Const), alors elle est précédée d'autant d'occurrences de (WeakΣ) qu'il n'y a de constantes dans Σ, et l'une d'entre elles possède la prémisse $\vdash_\Sigma B : s$. □

Dans le cadre des P^2TS, la notion de substitution bien typée doit tenir compte des types dépendants et donc de la présence éventuelle des variables d'un motif dans son type.

Définition 40 (Substitution bien typée dans les P^2TS)
Dans un contexte Γ, une substitution θ est dite bien typée relativement à un contexte Δ *si*

1. $\mathcal{D}om(\theta) = \mathcal{D}om(\Delta)$
2. *Pour tout* $(x{:}A) \in \Delta$, *on a* $\Gamma \vdash_\Sigma x\theta : A\theta$.

On notera en particulier que si Δ est réduit à $(x{:}A)$, cela signifie que $\Gamma \vdash_\Sigma x\theta : A$ car la variable x ne peut pas apparaître dans son propre type.

La définition d'un filtrage bien typé doit également tenir compte de cette modification.

Définition 41 (Théorie bien typée)
Une théorie équationnelle $\mathbb{T}$ est une théorie de filtrage bien typée dans les P^2TS si, pour tous termes P, A, B, pour tous contextes Γ, Δ, si $\Gamma, \Delta \vdash_\Sigma P : A$ et $\Gamma \vdash_\Sigma B : A$, alors toute solution de l'équation de filtrage $P \ll^?_{\mathbb{T}} B$ est également bien typée relativement à Δ.

Le bon typage du filtrage syntaxique dépend *a priori* de la signature choisie, et en particulier du type de chaque constante. Nous nous contenterons donc d'en faire l'hypothèse systématiquement ; pour une signature donnée, il est en général assez facile de décider si le filtrage est bien typé, car on peut examiner systématiquement tous les motifs selon leur constante de tête.

Nous sommes maintenant en mesure de démontrer que les P^2TS sont une extension conservative des PTS.

Théorème 41 (Conservativité par rapport aux PTS)
Pour tous pseudo-λ-termes A et B, pour tout contexte Γ, le jugement $\Gamma \vdash A : B$ est dérivable dans un PTS $(\mathcal{S}, \mathcal{A}, \mathcal{R})$ si et seulement s'il est dérivable dans le P^2TS $(\mathcal{S}, \mathcal{A}, \mathcal{R})$, avec une signature vide.

Démonstration : Tout d'abord, on peut voir qu'avec une signature vide, le filtrage syntaxique est bien typé. En effet un motif ne peut être qu'une variable : supposons donc $\Gamma, x{:}C \vdash_\Sigma x : A$ et $\Gamma \vdash_\Sigma B : A$ Par génération $A =_{\beta\delta} C$ et $\Gamma \vdash_\Sigma C : s$. La solution du problème de filtrage $x \ll^?_\emptyset B$ est $[x := B]$ et on a donc bien $\Gamma \vdash_\Sigma x[x := B] : C$.

La signature étant vide, (WeakΣ) et (Const) ne sont jamais utilisées, et la règle (Sig) est toujours valide.

Les termes considérés étant des λ-termes, la règle (Struct) n'est jamais utilisée.

Les règles (Axiom), (WeakΓ), (Conv) et (Var) sont exactement identiques dans les PTS et dans les P^2TS.

Comme les motifs sont réduits à des variables, les règles (Abs) et (Prod) sont également identiques à celles des PTS, à condition de remplacer le contexte $\Delta = (x{:}A)$ par le type A. La prémisse $\Gamma, \Delta \vdash_\Sigma x : A$ de (Prod) ne pose pas de problème puisque $\Delta = (x{:}A)$ et la correction du type A est vérifiée par la prémisse $\Gamma, \Delta \vdash_\Sigma A : s_1$.

Les seules règles pouvant poser problème sont donc (Appl) et (Match), la seconde n'ayant d'ailleurs aucune homologue dans les PTS. Cependant, par génération, (Match) suit toujours (Appl) pour démontrer sa seconde prémisse. Au pire elles sont séparées par des conversions et des affaiblissements, qui ne modifient ni la contrainte de filtrage ni la sorte s. On a donc toujours une sous-dérivation de la forme

$$(\text{Appl})\ \frac{\Gamma \vdash_\Sigma A : \Pi x{:}(x{:}D).C \qquad (\text{Match})\ \dfrac{\Gamma \vdash_\Sigma B : D \quad \Gamma, x{:}D \vdash_\Sigma D : s_1 \quad \Gamma, x{:}D \vdash_\Sigma C : s_2}{\Gamma \vdash_\Sigma [x \ll_{x:D} B]C : s_2}}{\Gamma \vdash_\Sigma A\, B : [x \ll_{x:D} B]C}$$

La prémisse $\Gamma, x{:}D \vdash_\Sigma x : D$ de la règle (Match) a été omise pour les mêmes raisons que dans le cas de (Prod) : elle est triviale.

Comparons avec une sous-dérivation pour le même terme dans un PTS :

$$(\text{Appl})\ \frac{\Gamma \vdash_\Sigma A : \Pi x{:}D.C \qquad \Gamma \vdash_\Sigma B : D}{\Gamma \vdash_\Sigma A\, B : C[x := B]}$$

Pour A, les prémisses de typage sont donc bien identiques en PTS et en P^2TS. Pour le typage de B, dans les P^2TS, on demande $\Gamma \vdash_\Sigma B : D$ donc les prémisses de typage de B sont identiques. Enfin, pour l'attribution des sortes à D et C, il faut utiliser la correction du typage dans les P^2TS : le type de A est correct donc on a (pour un triplet $(s_1, s_2, s_3) \in \mathcal{R}$)

$$(\text{PROD})\ \frac{\Gamma \vdash D : s_1 \qquad \Gamma, x{:}D \vdash C : s_2}{\Gamma \vdash \Pi x{:}D.C : s_3}$$

La présence ou l'absence de la déclaration $x{:}D$ dans le contexte de typage de D ne modifie pas la validité de la dérivation de typage. □

Le lemme de substitution constitue une étape essentielle pour la préservation du type.

Lemme 42 (Substitution)
Pour tous pseudo-termes A et B, pour tous contextes Γ, Δ, pour toute substitution θ bien typée relativement à Δ, si $\Gamma, \Delta \vdash_\Sigma A : B$ alors $\Gamma \vdash_\Sigma A\theta : B\theta$.

Démonstration : Par récurrence sur la structure de A, en utilisant le bon typage de θ au niveau des variables de $\mathcal{D}om(\theta)$. □

Lorsque la théorie de filtrage est bien typée, la préservation du type ne pose pas de problème particulier. Elle se démontre encore en commençant par le cas d'un unique pas de réduction.

Théorème 43 (Préservation du type [BCKL03])
Pour tous pseudo-termes A et B, pour tout contexte Γ, si $\Gamma \vdash_\Sigma A : B$ et si $A \mapsto_{\rho\sigma\delta} A'$ alors $\Gamma \vdash_\Sigma A' : B$.

Démonstration : Par récurrence sur la structure de A.
Les cas où la réduction n'a pas lieu en tête de A ne posent pas de problème.
Les réductions (ρ) et (σ) se traitent grâce au lemme de substitution et au bon typage du filtrage.
La δ-réduction se traite en exploitant le fait que tous les membres de la structure ont le même type. □

Corollaire 44
Si $\Gamma \vdash_\Sigma A : B$ et si $A \mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta} A'$ alors $\Gamma \vdash_\Sigma A' : B$ (par récurrence sur la longueur de $A \mapsto\!\!\!\!\rightarrow_{\rho\sigma\delta} A'$).

La propriété d'unicité du type n'est pas vraie dans tous les P^2TS : il faut s'assurer que la formation des produits n'est pas ambiguë. On notera que cette condition supplémentaire n'est pas particulière à notre calcul : c'est également le cas dans les PTS. La condition donnée ici est notamment vérifiée par tous les systèmes du cube.

Théorème 45 (Unicité du type) *Soit un P^2TS tel que :*
- *si $(s_1 : s_2) \in \mathcal{A}$ et $(s_1 : s_2') \in \mathcal{A}$, alors $s_2 \equiv s_2'$;*
- *si $(s_1, s_2, s_3) \in \mathcal{R}$ et $(s_1, s_2, s_3') \in \mathcal{R}$, alors $s_3 \equiv s_3'$.*

Alors pour tous pseudo-termes A, B, B', pour tout contexte Γ, si $\Gamma \vdash_\Sigma A : B$ et $\Gamma \vdash_\Sigma A : B'$, alors $B =_{\rho\sigma\delta} B'$.

Démonstration : Par récurrence sur la structure de A. On utilise systématiquement le lemme de génération, et on applique l'hypothèse de récurrence sur les prémisses qu'il fournit. □

La propriété de normalisation forte n'est pas valable non plus dans tous les PTS, mais elle l'est notamment dans tous les systèmes du λ-cube. En ce qui concerne les P^2TS, nous la démontrerons dans la prochaine section pour les deux systèmes $\rho_\rightarrow$ et ρP. La technique de démonstration employée sera la réduction vers un autre calcul fortement normalisant : nous traduirons les termes de $\rho_\rightarrow$ dans le système $\lambda\omega$ du λ-cube, de sorte que tout terme typé se traduise en un terme typé, et de sorte qu'une $\rho\sigma\delta$-réduction se traduise en au moins une β-réduction. La normalisation forte de $\rho_\rightarrow$ sera alors une conséquence de celle de $\lambda\omega$. Nous réduirons ensuite ρP à $\rho_\rightarrow$.

Pour les autres systèmes du ρ-cube rien ne semble indiquer que la normalisation forte soit invalide, mais à ce jour aucune démonstration n'a abouti.

Conjecture 2 (Normalisation forte dans le ρ-cube)
Dans tous les systèmes du ρ-cube, pour tous pseudo-termes A, B, pour tout contexte Γ, si $\Gamma \vdash_\Sigma A : B$ alors A est fortement normalisant selon $\mapsto_{\rho\sigma\delta}$.

Nous pouvons immédiatement énoncer deux conséquences de la normalisation forte.

Théorème 46 (Décidabilité du typage dans les P^2TS)
Dans tout P^2TS fortement normalisant vérifiant l'unicité du typage, les deux problèmes suivants sont décidables :

1. *Étant donnés un contexte Γ et deux pseudo-termes A, B, peut-on dériver $\Gamma \vdash_\Sigma A : B$?*
2. *Étant donnés un contexte Γ et un pseudo-terme A, existe-t-il un pseudo-terme B tel que $\Gamma \vdash_\Sigma A : B$?*

Démonstration : Par unicité du typage, le premier problème se réduit à résoudre le second et à décider de l'égalité modulo $=_{\rho\sigma\delta}$ de deux pseudo-termes. Cette dernière égalité est décidable par comparaison des formes normales.

Enfin, l'inférence d'un type canonique se fait grâce au lemme de génération et en utilisant les informations de typage données dans les contextes Δ des abstractions et des contraintes de filtrage. □

La consistance n'est pas à proprement parler une conséquence de la normalisation forte, mais avec cette dernière elle assure que le type $\Pi\alpha{:}*.\alpha$ n'est pas habité.

Théorème 47 (Consistance dans les P^2TS)
Dans tout P^2TS étendant $\rho 2$, il n'existe aucun terme A en forme normale tel que $\emptyset \vdash_\Sigma A : \Pi\alpha{:}*.*$.

Démonstration : Par l'absurde : on suppose $\emptyset \vdash_\Sigma A : \Pi\alpha{:}*.*$ et A en forme normale. Par inspection d'une dérivation de type pour ce jugement on trouve un radical ou une variable libre dans A. □

5.2 Normalisation forte dans $\rho_\rightarrow$ et dans ρP

Comme nous l'avons dit à propos des PTS (théorème 4), les techniques de démonstration de normalisation forte utilisent généralement une interprétation qui associe aux types des ensembles de termes. Typiquement, le type $\Pi x{:}A.B$ est interprété comme l'espace des fonctions qui, appliquées à un argument de type A, renvoient une valeur de type B. Pour les P^2TS cette technique semble échouer pour la raison suivante : l'interprétation d'un type $\Pi P{:}\Delta.C$ devrait être également un espace de fonctions dont le domaine est formé de termes qui sont dans l'interprétation du type de P, mais ces termes devraient également être des instances de P dont les sous-termes appartenant aux interprétations respectives des types des différentes variables de P. Cette imbrication d'interprétations mène rapidement à des circularités dans les définitions des interprétations.

Nous utiliserons donc ici une autre technique de démonstration courante : la réduction vers un autre calcul fortement normalisant. Nous allons donc définir une traduction du système $\rho_\rightarrow$ vers le système $\lambda\omega$ du λ-cube, telle que tout terme typé se traduit en un terme typé, et telle qu'une $\rho\sigma\delta$-réduction se traduise en au moins une β-réduction.

5.2.1 Traduction non typée en lambda-calcul

Pour des raisons de clarté de l'exposé, nous commencerons par définir la fonction de traduction $[\![\ ]\!]$ sur les pseudo-termes ; nous utilisons le typage uniquement pour définir une notion d'*arité maximale* pour les constantes et les structures. Nous montrerons la correction de cette traduction du point de vue des réductions.

Nous enrichirons ensuite la traduction pour tenir compte des types, et nous verrons que le λ-calcul simplement typé ne suffit pas : nous avons besoin de certains mécanismes de typage de $\lambda\omega$ pour reproduire ceux de $\rho_\rightarrow$. La correction du point de vue des types sera démontrée dans la section suivante.

Posons quelques conventions de notation pour la suite de ce chapitre : nous utiliserons $\vdash_\Sigma$ pour les jugements de typage dans $\rho_\rightarrow$ et $\vdash_\omega$ pour les jugements dans $\lambda\omega$.

Dans le système $\lambda\omega$, les types sont formés par des Π-abstractions $\Pi x{:}\sigma.\tau$. Cependant, si $x \notin \mathcal{FV}(\tau)$, on peut utiliser l'abréviation $\sigma \rightarrow \tau$ du λ-calcul simplement typé. Nous l'utiliserons dans deux cas en particulier :

- si $\vdash_\omega \sigma : *$, alors $\vdash_\omega \tau : *$ (on ne considère pas de types dépendants). La variable x n'apparaît alors pas dans τ.
- si $\vdash_\omega \sigma : \Box$ et $\vdash_\omega \tau : \Box$, c.-à-d. quand $\Pi x{:}\sigma.\tau$ est un genre, x ne peut pas non plus apparaître dans τ.

Préliminaires

Commençons par quelques lemmes techniques cruciaux pour la construction de la traduction.

Lemme 48

Dans $\rho_\rightarrow$, si $\Gamma \vdash_\Sigma A : \Box$ alors $A \equiv *$.

Démonstration : Par récurrence sur une dérivation pour le jugement $\Gamma \vdash_\Sigma A : \Box$: on distingue selon la dernière règle utilisée.

(Const) alors la règle (WeakΣ) impose que $\vdash_\Sigma \Box : s$, ce qui est impossible (le seul axiome de $\mathcal{A}$ est $(* : \Box)$).

(Axiom) alors on a bien $A \equiv *$ puisque le seul axiome de $\mathcal{A}$ est $(* : \Box)$.

(WeakΓ) ou (Struct) on conclut par hypothèse de récurrence sur la prémisse $\Gamma \vdash_\Sigma A : \Box$.

(Conv) ou (Var) alors une prémisse impose $\Gamma \vdash_\Sigma \Box : s$ ce qui est impossible (le seul axiome de $\mathcal{A}$ est $(* : \Box)$).

(Abs) ou (Appl) ne conviennent pas car le type ne peut pas être $\Box$.

(Prod) ou (Match) ne conviennent pas car on aurait $(s_1, s_2, \Box) \in \mathcal{R}$ or la seule règle de $\mathcal{R}$ est $(*, *, *)$.

□

Lemme 49 (Forme des types)

Dans le système $\rho_\to$ et pour un contexte Γ, on appellera type *tout terme C tel que $\Gamma \vdash_\Sigma C : *$ et tel que C ne comporte aucune structure.*

Les types appartiennent alors tous au langage suivant :

$$C \ ::= \ \mathcal{X} \mid \mathcal{K} \mid \Pi P{:}\Delta.C \mid [P \ll_\Delta A]C$$

où

- *les variables de type x prises dans $\mathcal{X}$ sont telles que $\Gamma, \overline{\Delta} \vdash_\Sigma x : *$, les $\overline{\Delta}$ étant des contextes qui apparaissent dans les Π-abstractions et les contraintes de filtrage de C ;*
- *les types atomiques ι pris dans $\mathcal{K}$ sont des constantes de Σ telles que $\vdash_\Sigma \iota : *$.*

Démonstration : Par récurrence sur une dérivation pour le jugement $\Gamma \vdash_\Sigma C : *$: on distingue selon la dernière règle utilisée.

(Conv) ne peut pas modifier le jugement $\Gamma \vdash_\Sigma C : *$. En effet, on aurait alors $\Gamma \vdash_\Sigma C : B$ pour un certain B tel que $B =_{\beta\delta} *$. Par correction, on aurait également $\Gamma \vdash_\Sigma B : s$ pour une certaine sorte s. Par confluence $B \mapsto\!\!\!\!\!\rightarrow_{\beta\delta} *$ et donc par préservation du type $\Gamma \vdash_\Sigma * : s$. Par unicité, on trouve $s \equiv \Box$: on a donc $\Gamma \vdash_\Sigma B : \Box$, ce qui d'après le lemme 48 assure que $B \equiv *$. La règle de conversion est donc ici superflue.

(Axiom) ne convient pas car il n'existe aucun axiome $(s_1 : *)$ dans $\mathcal{A}$.

(WeakΓ) on conclut par hypothèse de récurrence sur la prémisse $\Gamma \vdash_\Sigma C : *$.

(Const) et (Var) correspondent aux deux premiers cas de notre grammaire de types. La variable x était présente dans le contexte initial ou a été chargée par une règle (Prod) ou (Match).

(Struct) on a supposé que C ne comportait aucune structure.

(Abs) ou (Appl) ne conviennent pas car le type ne peut pas être $*$.

(Prod) ou (Match) correspondent aux deux derniers cas de notre grammaire de types. Elles imposent que $s_1 = s_2 = s_3$ car la seule règle de $\mathcal{R}$ est $(*, *, *)$, et donc on peut appliquer l'hypothèse de récurrence sur la prémisse $\Gamma, \Delta \vdash_\Sigma C : *$.

□

Remarquons que le fait de ne pas considérer de structures dans les types n'est pas vraiment une restriction : aucune règle de typage ne permet de dériver des jugements $\Gamma \vdash_\Sigma A : C$ où C est une structure.

Par ailleurs, notons qu'une variable x de type $*$ ne peut être abstraite dans $\rho_\to$, ce qui lui donne un statut quasi similaire à celui d'une constante. Sans perte de généralité, on pourra donc toujours supposer que les types ne comportent pas de variables de type.

Rappelons que les constantes sont données sans arité ; cependant, les types nous permettent de retrouver cette notion dans une certaine mesure. Le lemme suivant énonce une propriété relativement intuitive de $\rho_\to$: un terme peut recevoir autant d'arguments qu'il y a de Π-abstractions dans son type.

Lemme 50 (Arité maximale)
Fixons un contexte Γ. *On définit l'*arité maximale *d'un type comme suit :*

$$\begin{array}{rcl} \alpha(x) & \triangleq & 0 \\ \alpha(\iota) & \triangleq & 0 \\ \alpha(\Pi P{:}\Delta.C) & \triangleq & 1+\alpha(C) \\ \alpha([P \ll_\Delta A]C) & \triangleq & \alpha(C) \end{array}$$

Pour tout terme A *tel que* $\Gamma \vdash_\Sigma A : C$, *si* $\Gamma \vdash_\Sigma A\,\overline{B}_{(1..k)} : D$, *alors* $k \leq \alpha(C)$ *et* $\alpha(D) = \alpha(C) - k$.

Démonstration : On vérifie facilement que α est stable par conversion du type, car la variable x qui se trouve tout à droite du type ne peut être instanciée (il faudrait au moins se placer dans le système $\rho 2$).

On procède par récurrence sur k.

si $k = 0$ **:** trivial

si $0 < k$ **:** le terme $A\,\overline{B}_{(1..k)}$ étant typable, ses sous-termes le sont aussi donc $\Gamma \vdash_\Sigma A\,\overline{B}_{(1..k-1)} : E$.
Par hypothèse de récurrence on a donc $k-1 \leq \alpha(C)$ et $\alpha(E) = \alpha(C) - (k-1)$.
Par génération, une dérivation de type pour $A\,\overline{B}_{(1..k)}$ doit utiliser la règle

$$(\textsc{Appl})\ \frac{\Gamma \vdash_\Sigma A\,\overline{B}_{(1..k-1)} : \Pi Q{:}\Delta.E_1 \qquad \Gamma \vdash_\Sigma [Q \ll_\Delta B_k]E_1 : s}{\Gamma \vdash_\Sigma A\,\overline{B}_{(1..k)} : [Q \ll_\Delta B_k]E_1}$$

où $E =_{\beta\delta} \Pi Q{:}\Delta.E_1$ et $D =_{\beta\delta} [Q \ll_\Delta B_k]E_1$. On en conclut d'une part que $\alpha(E) = \alpha(E_1) + 1 \geq 1$ donc $\alpha(C) = \alpha(E) + k - 1 \geq k$ et d'autre part que $\alpha(D) = \alpha(E_1) = \alpha(E) + 1 = \alpha(C) - (k-1) + 1 = \alpha(C) - k$.

□

En particulier, on utilisera cette notion pour les constantes et les structures :
- si $\vdash_\Sigma f : C$ on notera α_f la quantité $\alpha(C)$ que par abus de langage on appellera aussi arité maximale de f ;
- si $\Gamma \vdash_\Sigma A \wr B : C$ on notera $\alpha_{A \wr B}$ la quantité $\alpha(C)$.

Corollaire 51 (Compatibilité des arités dans une équation de filtrage)
Si un préradical $(\lambda(f\,\overline{P}_{(1..p)}).A)\,(g\,\overline{B}_{(1..q)})$ *ou* $[f\,\overline{P}_{(1..p)} \ll g\,\overline{B}_{(1..q)}]A$ *apparaît dans un terme typable, alors* $\alpha_f - p = \alpha_g - q$.

Démonstration : En effet on invoque alors forcément la règle (Match), dont les prémisses imposent que $\Gamma \vdash_\Sigma f\,\overline{P}_{(1..p)} : A$ et $\Gamma \vdash_\Sigma g\,\overline{B}_{(1..q)} : A\theta$. Vue la forme des types dans $\rho_\to$, $\alpha(A) = \alpha(A\theta)$ donc immédiatement $\alpha_f - p = \alpha(A) = \alpha_g - q$.

En particulier, quand $f \equiv g$, la condition $\alpha_f - p = \alpha_g - q$ se réduit à $p = q$, ce qui est essentiel pour la résolution d'une telle équation de filtrage. □

Traduction non typée

Rappelons que nous considérons des motifs algébriques et un filtrage syntaxique. Pour démontrer les premières propriétés des P^2TS, nous avons supposé la linéarité des motifs ; pour l'encodage en λ-calcul elle n'est pas nécessaire et nous pouvons donc nous placer dans ce cadre un peu plus général.

Dans le cas d'un filtrage plus élaboré, il faudrait encoder une stratégie d'évaluation du ρ-calcul dans le λ-calcul. De plus il faudrait enrichir la traduction pour qu'elle engendre toutes les solutions des problèmes de filtrage, qui peuvent être arbitrairement nombreuses. Nous conjecturons que la plupart des théories équationnelles dites *syntaxiques*, caractérisées par C. Kirchner et F. Klay [KK90], peuvent être encodées. En effet, leur propriété fondamentale est qu'on peut les décrire par l'application d'une seule règle en tête de terme, suivie par des manipulations dans les sous-termes, ce qui semble possible à représenter avec des λ-termes typés. Il est également vraisemblable que le filtrage syntaxique sur les abstractions de [BCKL03] peut être encodé.

L'algorithme de filtrage syntaxique que nous représentons en λ-calcul se décrit par un algorithme assez simple : vérifier si l'argument a la même constante de tête que le motif (ce dont on se dispense en fait dans la traduction), et relancer le filtrage sur chaque sous-terme avec le sous-motif qui lui correspond. Si l'on considère un motif non linéaire, les différentes occurrences d'une même variable seront renommées, et les tests d'égalité inhérents au filtrage non linéaire sont donc omis. Cette simplification est l'une des raisons pour lesquelles notre traduction est complète mais non correcte, au sens où il peut y avoir, dans le λ-terme obtenu, des β-réductions qui ne correspondent à aucune réduction dans le ρ-terme initial. Pour démontrer la normalisation forte, il nous suffit que le λ-terme soit sujet à au moins autant de réductions que le ρ-terme, et donc la complétude suffira.

La traduction est donnée en figure 5.3, par une fonction $[\![\ ]\!]$ qui à tout ρ-terme associe un λ-terme. Toutes les variables que nous introduisons doivent être fraîches ; elles sont toutes liées dans le λ-terme, sauf la variable $x_\perp$ qui restera libre. Nous l'utiliserons notamment pour représenter certains échecs de filtrage, ainsi que comme « joker » pour des arguments inutiles de certains λ-termes. Rappelons que α_f (resp. $\alpha_{A \wr B}$) dénote l'arité maximale de f (resp. de $A \wr B$).

$$
\begin{aligned}
[\![x]\!] &\triangleq x \\
[\![f]\!] &\triangleq \lambda \overline{x}_{(1..\alpha_f)}.\ \lambda z.(z\, \overline{x}_{(1..\alpha_f)}) \\
[\![A \wr B]\!] &\triangleq \lambda \overline{x}_{(1..\alpha_{A \wr B})}.\ \big((\lambda z.[\![A]\!]\overline{x}_{(1..\alpha_{A \wr B})})([\![B]\!]\overline{x}_{(1..\alpha_{A \wr B})})\big) \\
[\![\lambda x.A]\!] &\triangleq \lambda x.[\![A]\!] \\
[\![\lambda(f\, \overline{P}_{(1..p)}).A]\!] &\triangleq \lambda u.(u\overline{x_\perp}_{(p+1..\alpha_f)}\, [\![\lambda \overline{P}_{(1..p)}.\lambda \overline{x'}_{(p+1..\alpha_f)}.A]\!]) \\
&\quad \text{(en renommant les éventuelles occurrences de variables non linéaires dans } \overline{P}_{(1..p)}) \\
[\![A\, B]\!] &\triangleq [\![A]\!]\, [\![B]\!]
\end{aligned}
$$

FIG. 5.3 – Traduction non typée

Remarque 8 *Pour les constantes f telles que $\alpha_f = 0$ on a :*

$$
\begin{aligned}
[\![f]\!] &\triangleq \lambda z.z \\
[\![\lambda f.A]\!] &\triangleq \lambda u.(u[\![A]\!])
\end{aligned}
$$

Détaillons quelques éléments clefs de cette traduction :

- Dans $[\![f]\!]$, les variables $x_1 \dots x_{\alpha_f}$ sont vouées à être instanciées par les arguments de f, ce qui justifie *a posteriori* l'étude des arités maximales.
  La variable z peut être instanciée par toute fonction qui veut récupérer séparément les différents arguments de f, et c'est ce qui permettra de faire du filtrage.
- Une structure $[\![A \wr B]\!]$ n'est pas encodée exactement comme une paire du λ-calcul : au lieu d'écrire $\lambda z.(z\,A\,B)$ on prend $(\lambda z.A)\,B$. Ce terme est susceptible de se réduire en A, mais encore une fois seule la complétude de la traduction nous intéresse, pas sa correction.
  De plus, cette traduction facilitera la traduction des types : si A et B sont de type τ, alors $(\lambda z.A)\,B$ est également de type τ, ce qui correspond bien au type attendu pour une structure. Par contre, $\lambda z.(z\,A\,B)$ est de type $(\tau \rightarrow \tau \rightarrow \sigma) \rightarrow \sigma$, ce qui rendrait la traduction moins directe.
  Comme pour les constantes, on utilise $\alpha_{A \wr B}$ abstractions pour distribuer les arguments dans la structure, ce qui correspondra aux δ-réductions.
- Dans $[\![\lambda(f\,\overline{P}_{(1..p)}).A]\!]$, la variable u sera instanciée par l'argument de cette abstraction. Si c'est un terme algébrique (par exemple $g\,\overline{B}_{(1..q)}$), il sera donc appliqué à $\alpha_g - q$ occurrences de $x_\perp$, qui ne joueront pas un rôle déterminant dans les réductions mais qui sont là uniquement pour compléter le motif dans les cas où $q < \alpha_g$, c.-à-d. que la constante g est appliquée à moins d'arguments que ne le permet son arité maximale.
  Le dernier argument est la traduction du filtrage sur les sous-motifs $\overline{P}_{(1..p)}$. Il instanciera z dans $[\![g]\!]$ et chaque argument de g sera donc filtré contre le sous-motif qui lui correspond. L'identité des constantes de tête f et g n'est pas vérifiée, ce qui est une autre raison pour lesquelles la traduction est complète mais non correcte.
  Lorsqu'on considère des motifs non linéaires, il faut renommer les différentes occurrences d'une même variable dans les sous-motifs $\overline{P}_{(1..p)}$; une simple α-conversion à chaque formation d'un terme $\lambda\overline{P}_{(1..p)}.\lambda\overline{x'}_{(p+1..\alpha_f)}.A$ suffit à effectuer ce renommage.

Exemple 31 (Traduction d'un terme)

Prenons $\Sigma = \{a{:}\iota, f{:}\Pi x{:}(x{:}\iota).\iota\}$, d'où $\alpha_a = 0$ et $\alpha_f = 1$. Le terme $\big(\lambda y.(\lambda(f\,x).x)\,y\big)\,(f\,a)$ (les types des variables sont omis) se traduit ainsi :

$$\underbrace{(\lambda y.\Big(\overbrace{(\lambda u.(u\,(\lambda x.x)))}^{[\![\lambda(f\,x).x]\!]}\,y\Big))}_{[\![\lambda y.(\lambda(f\,x).x)\,y]\!]}\,\Big(\overbrace{(\lambda x_1.\lambda z.(z x_1))}^{[\![f]\!]}\,\overbrace{(\lambda v.v)}^{[\![a]\!]}\Big)$$

Montrons également comment les réductions sont reproduites par la traduction.

Exemple 32 (Traduction d'une ρ-réduction)

Le seul chemin de réduction du terme de l'exemple 31 est :

$$(\lambda y.(\lambda(f\,x).x)\,y)\,(f\,a) \;\mapsto_\rho\; (\lambda(f\,x).x)\,(f\,a) \;\mapsto_\rho\; a$$

En particulier le préradical interne $(\lambda(f\,x).x)\,y$ ne peut être réduit avant instanciation de y. Voyons comment la traduction reproduit ce comportement : on choisit de réduire d'abord les β-radicaux du sous-terme $[\![(\lambda(f\,x).x)\,y]\!]$, et on constate qu'ici aussi l'instanciation de y est nécessaire pour terminer la réduction.

À chaque pas de réduction, la λ-abstraction sélectionnée et son argument sont soulignés.

$$\begin{array}{ll} & (\lambda y.\Big((\underline{\lambda u}.(u(\lambda x.x)))\underline{y}\Big))\Big((\lambda x_1.\lambda z.(z x_1))(\lambda v.v)\Big) \\ \mapsto_\beta & \big(\lambda y.(y(\lambda x.x))\big)\Big((\underline{\lambda x_1}.\lambda z.(z x_1))\underline{(\lambda v.v)}\Big) \\ \mapsto_\beta & \big(\underline{\lambda y}.(y(\lambda x.x))\big)\underline{\big(\lambda z.(z(\lambda v.v))\big)} \\ \mapsto_\beta & \big(\underline{\lambda z}.(z(\lambda v.v))\big)\underline{(\lambda x.x)} \\ \mapsto_\beta & (\underline{\lambda x}.x)\underline{(\lambda v.v)} \\ \mapsto_\beta & (\lambda v.v) \\ = & [\![a]\!] \end{array}$$

Voyons également ce que donne un échec de filtrage.

Exemple 33 (Traduction d'un échec de filtrage)

Prenons cette fois $\Sigma = \{g{:}\Pi x{:}(x{:}\iota).\iota, f{:}\Pi x{:}(x{:}\iota).\iota\}$, d'où les arités maximales $\alpha_f = \alpha_g = 1$. Le terme $(\lambda(f\,x).x)\,(g\,y)$ est en forme normale puisque les constantes de tête f et g diffèrent. Cependant, comme notre traduction ne tient pas compte des constantes de tête, cet échec de filtrage ne se voit pas.

$$\begin{array}{ll} & \overbrace{(\overbrace{\lambda u.(u(\lambda x.x))}^{[\![\lambda(f\,x).x]\!]})}(\overbrace{(\underline{\lambda x_1}.\lambda z.(z x_1))}^{[\![g]\!]}\underline{y}) \\ \mapsto_\beta & \big(\underline{\lambda u}.(u(\lambda x.x))\big)\underline{\big(\lambda z.(z y)\big)} \\ \mapsto_\beta & \big(\underline{\lambda z}.(z y)\big)\underline{(\lambda x.x)} \\ \mapsto_\beta & (\underline{\lambda x}.x)\underline{y} \\ \mapsto_\beta & y \end{array}$$

Montrons alors la correction de cette traduction vis-à-vis des réductions. On commence par vérifier que la traduction est stable par substitution, ce qui nous permettra par la suite de traiter les ρ-réductions.

Lemme 52 (Clôture par substitution)

Pour tous termes $A, B_1, \ldots, B_n$, pour toutes variables $x_1, \ldots x_n$ (différentes de $x_\perp$),

$$[\![A[x_1 := B_1 \ldots x_n := B_n]]\!] \quad = \quad [\![A]\!][x_1 := [\![B_1]\!] \ldots x_n := [\![B_n]\!]] \tag{5.1}$$

Démonstration : On remarque aisément que les variables libres (resp. liées) le restent dans la traduction, et que $x_\perp$ est la seule nouvelle variable libre.

On procède par récurrence sur la structure de A. Les variables sont traduites par elles-mêmes et les constantes ne sont pas affectées par les substitutions, donc les cas de base ne posent pas de problème.

Dans tous les cas inductifs sauf celui de l'abstraction, la traduction de A utilise directement la traduction de tous les sous-termes de A, donc la substitution se propage directement à ceux-ci.

L'abstraction se traite par une autre récurrence sur le motif P : on démontre que si A vérifie l'équation (5.1), alors pour tous motifs $P_1 \ldots P_m$, le terme $\lambda\overline{P}_{(1..m)}.A$ vérifie également cette équation. Il s'agit ici essentiellement de vérifier que la fonction de traduction $[\![\ ]\!]$ s'évalue en un nombre fini d'appels récursifs : les motifs ne sont pas affectés par la substitution donc la propagation de la propriété (5.1) proprement dite au cours de l'induction est triviale.

L'ordre sur les ensembles de motifs $\overline{P}_{(1..m)}$ est donné par la mesure suivante :

$$\begin{aligned} \#(\overline{P}_{(1..m)}) &= \sum_{j=1}^{m} \#(P_j) \\ \#(x) &= 1 \\ \#(f\overline{P}_{(1..p)}) &= \sum_{j=1}^{p} \#(P_j) + \alpha_f + 1 \end{aligned}$$

Vérifions que cette mesure décroît au cours de la traduction. Si le motif P_1 est réduit à une variable alors c'est la règle de traduction $[\![\lambda x.A]\!] = \lambda x.[\![A]\!]$ qui s'applique et donc la mesure $\#$ décroît de 1.

Sinon, on a un terme de la forme $[\![\lambda f\overline{Q}_{(1..p)}.\lambda P_2 \ldots \lambda P_m.A]\!]$. La traduction décompose alors le premier motif $f\overline{Q}_{(1..p)}$ en ses sous-motifs ainsi qu'au plus α_f abstractions $\lambda x'$, et la constante f disparaît. La mesure décroît donc au moins de 1 :

$$\#(\overline{Q}_{(1..p)}\,\overline{x'}_{(p+1..\alpha_f)}) \;\leq\; \#(\overline{Q}_{(1..p)}) + \alpha_f \;=\; \#(f\overline{Q}_{(1..p)}) - 1$$

Nous utiliserons le même ordre pour traiter le cas des abstractions dans la démonstration du théorème 53. □

Théorème 53 (Simulation des réductions [Wac04])
Pour tous termes A et B, si $A \mapsto_{\rho\delta} B$, alors $[\![A]\!] \mapsto\!\!\!\!\rightarrow_{\beta} [\![B]\!]$ en un pas au moins.

Démonstration : Par récurrence sur la structure de A.

Comme pour le lemme de clôture par substitution, si la réduction a lieu dans un sous-terme de A, l'hypothèse de récurrence s'applique immédiatement puisque la traduction des sous-termes apparaît dans la traduction de A.

Si la réduction a lieu en tête de A, on distingue trois cas :

δ-réduction : la réduction $[\![(A \wr B)\,C]\!] \mapsto_{\beta} [\![A\,C \wr B\,C]\!]$ s'effectue en une β-réduction exactement : elle consiste à instancier la première des variables $\overline{x}_{(1..\alpha)}$ de $[\![A \wr B]\!]$ par C.

ρ-réduction : comme pour le lemme précédent, on vérifie en fait la propriété $[\![(\lambda\overline{P}.A)\,\overline{B}]\!] \mapsto_{p} [\![A\theta_{\overline{(P \ll B)}}]\!]$ pour un ensemble quelconque de motifs $\overline{P}$. On procède par récurrence sur $\overline{P}$ selon l'ordre induit par $\#$.

si $P \equiv x$ on a $[\![(\lambda x.A)B]\!] = (\lambda x.[\![A]\!])[\![B]\!] \mapsto_{\beta} [\![A]\!][x := [\![B]\!]]$ qui d'après le lemme 52 est égal à $[\![A[x := B]]\!]$.

si $P \equiv f\,P_1\,\ldots\,P_p$ alors

$$\begin{aligned} (\lambda(f\,P_1\,\ldots\,P_p).A)\,(f\,B_1\,\ldots\,B_p) &\mapsto_{p} A\theta_{(f\,P_1\,\ldots\,P_p \ll f\,B_1\,\ldots\,B_p)} \\ &= A\theta_{(P_1 \ll B_1)} \ldots \theta_{(P_p \ll B_p)} \end{aligned}$$

Alors la simulation de cette réduction commence par les β-réductions suivantes, où α_f et p peuvent être nuls mais au minimum la dernière β-réduction, qui instancie la

variable z, a lieu.

$$\begin{array}{ll} & [\![(\lambda(f\,\overline{P}_{(1..p)}).A)\,(f\,\overline{B}_{(1..p)})]\!] \\ = & \Big(\big(\lambda\overline{x}_{(1..\alpha_f)}.\ \lambda z.(z\overline{x}_{(1..x_{\alpha_f})})\big)\overline{[\![B]\!]}_{(1..p)}\Big)\ \overline{x_\perp}_{(p+1..\alpha_f)}\ [\![\lambda\overline{P}_{(1..p)}.\lambda\overline{x'}_{(p+1..\alpha_f)}.A]\!] \\ \mapsto\!\!\!\!\rightarrow_\beta & \Big(\lambda\overline{x}_{(p+1..\alpha_f)}.\ \lambda z.(z\overline{[\![B]\!]}_{(1..p)}\overline{x}_{(p+1..\alpha_f)})\Big)\ \overline{x_\perp}_{(p+1..\alpha_f)}\ [\![\lambda\overline{P}_{(1..p)}.\lambda\overline{x'}_{(p+1..\alpha_f)}.A]\!] \\ \mapsto\!\!\!\!\rightarrow_\beta & (\lambda z.(z\overline{[\![B]\!]}_{(1..p)}\overline{x_\perp}_{(p+1..\alpha_f)}))[\![\lambda\overline{P}_{(1..p)}.\lambda\overline{x'}_{(p+1..\alpha_f)}.A]\!] \\ \mapsto\!\!\!\!\rightarrow_\beta & [\![\lambda\overline{P}_{(1..p)}.\lambda\overline{x'}_{(p+1..\alpha_f)}.A]\!]\overline{[\![B]\!]}_{(1..p)}\overline{x_\perp}_{(p+1..\alpha_f)} \end{array}$$

Remarquons que dans le cas de motifs non linéaires, si le filtrage a lieu, alors tous les sous-termes de l'argument qui correspondent à une même variable x sont égaux. Ceci justifie que, dans la traduction, nous pouvons choisir l'un quelconque de ces sous-termes indifféremment, et que les autres sous-termes peuvent instancier des variables fraîches qui n'apparaissent pas dans le corps A de l'abstraction.

Par hypothèse de récurrence sur $P_1 \dots P_p$, les p réductions correspondant à ces nouveaux motifs sont correctement simulées. Les variables $x'_{p+1} \dots x'_{\alpha_f}$ sont instanciées par $x_\perp$, mais comme elles n'apparaissent pas dans $[\![A]\!]$ ceci n'affecte pas les futures réductions de $[\![A]\!]$. On a donc

$$\begin{array}{lll} & [\![(\lambda(f\,\overline{P}_{(1..p)}).A)\,(f\,\overline{B}_{(1..p)})]\!] & \\ \mapsto\!\!\!\!\rightarrow_\beta & [\![\lambda\overline{P}_{(1..p)}.\lambda\overline{x'}_{(p+1..\alpha_f)}.A]\!]\ \overline{[\![B]\!]}_{(1..p)}\overline{x_\perp}_{(p+1..\alpha_f)} & \\ = & [\![(\lambda\overline{P}_{(1..p)}.\lambda\overline{x'}_{(p+1..\alpha_f)}.A)\ \overline{B}_{(1..p)}]\!]\overline{x_\perp}_{(p+1..\alpha_f)} & \\ \mapsto\!\!\!\!\rightarrow_\beta & [\![A\theta_{(P_1 \ll B_1)} \dots \theta_{(P_p \ll B_p)}]\!] & \text{(par hypothèse de récurrence)} \\ = & [\![A\theta_{(f\,P_1\,\dots\,P_p \ll f\,B_1\,\dots\,B_p)}]\!] & \end{array}$$

□

Remarque 9 (Vers une traduction complète et correcte)

Comme nous l'avons déjà signalé, nous avons donné ici une traduction non typée complète mais non correcte, dans la mesure où cela est suffisant pour démontrer la normalisation forte. Néanmoins, si on ne considère que des motifs linéaires, il est possible d'assurer également la correction au prix d'une traduction plus complexe.

Par exemple, pour que l'algorithme de filtrage syntaxique encodé en λ-calcul vérifie aussi l'égalité des constantes de tête, il suffit de remplacer dans $[\![f]\!]$ le sous-terme $\lambda z.(z\,\overline{x}_{(1..\alpha_f)})$ par une projection $\lambda\overline{z}_{(1..S)}.(z_i\,\overline{x}_{(1..\alpha_f)})$ où S est la taille de la signature Σ et où i est un indice associé à la constante f. Réciproquement, on transforme $[\![\lambda(f\overline{P}).A]\!]$ en un S-uplet en appliquant la variable u à un vecteur de S arguments dont seul le i^e est $[\![\lambda\overline{P}.A]\!]$, les autres représentant un échec. L'application d'une projection à un S-uplet correspond alors à vérifier l'égalité des constantes de tête.

5.2.2 Traduction typée de $\rho_{\rightarrow}$ vers $\lambda\omega$

Les réductions étant donc bien préservées par la traduction, voyons comment on peut traiter les types. Nous commencerons par une étude relativement informelle des problèmes posés par le typage de la traduction, ce qui nous permettra par la même occasion de voir pourquoi il est pratique, voire indispensable, d'utiliser le système de types $\lambda\omega$.

Nous allons regarder de plus près trois constructions fondamentales de la traduction :

- le typage de la traduction d'une constante (qui justifie l'usage de termes dépendant de types) ;
- le typage d'une variable (qui justifie l'usage de types dépendant de types) ;
- la traduction des contraintes de filtrage qui figurent dans les types de $\rho_{\rightarrow}$.

Par la suite, nous utiliserons la notation suivante, pour tous types $\sigma_1, \ldots, \sigma_\alpha$ de $\lambda\omega$.

$$\bigwedge \overline{\sigma}_{(1..\alpha)} \quad \triangleq \quad \Pi\beta{:}*\,.\big((\overline{\sigma}_{(1..\alpha)} \rightarrow \beta) \rightarrow \beta\big)$$

Cette notation n'est pas innocente : si on regarde le type ainsi défini comme une proposition dans PROP2, il est exactement équivalent à la conjonction des propositions $\sigma_1, \ldots, \sigma_\alpha$. En particulier, quand $\alpha = 0$, on a

$$\bigwedge \emptyset \quad \triangleq \quad \Pi\beta{:}*\,.(\beta \rightarrow \beta)$$

Traduction d'une constante

Étudions donc le typage de la traduction d'une constante, en commençant par un exemple. Supposons $f : \Pi x{:}\iota.\iota$: alors $\alpha_f = 1$, d'où la traduction ci-dessous, donnée avec un type possible.

$$\vdash_\omega [\![f]\!] = \lambda x_1.\lambda z.(z\ x_1) : \sigma \rightarrow (\sigma \rightarrow \beta) \rightarrow \beta$$

Ici σ est un type supposé correct pour l'argument de f et β peut être un type quelconque, que l'on va déterminer à partir du type de z. Si on considère maintenant un terme B tel que $[\![B]\!] : \sigma$, on a donc $\vdash_\omega [\![f\,B]\!] : (\sigma \rightarrow \beta) \rightarrow \beta$.

Pour déterminer la valeur de β, observons un type possible pour la traduction d'une abstraction :

$$\vdash_\omega [\![\lambda(f\,x).A]\!] = \lambda u.\,\big(u\big(\lambda x.[\![A]\!]\big)\big) : ((\sigma \rightarrow \tau) \rightarrow \gamma) \rightarrow \gamma$$

où τ est le type de $[\![A]\!]$. On constate donc que $[\![\lambda(f\,x).A]\!]$ s'applique à $[\![f\,B]\!]$ si et seulement si

$$(\sigma \rightarrow \tau) \rightarrow \gamma \;=\; (\sigma \rightarrow \beta) \rightarrow \beta$$

ce qui se réduit à la simple condition $\tau = \beta = \gamma$. Cependant, β apparaît dans le type de $[\![f]\!]$ alors que τ est le type de $[\![A]\!]$. Il faudrait donc que l'argument $f\,B$ «devine» le type de retour de la fonction qui va lui être appliquée. Ceci n'est possible que si l'argument dépend d'un type, et que la fonction elle-même lui communique ce type. Nous utiliserons donc des types polymorphes, qui permettent ce style de manipulations.

Dans notre exemple, il suffit de prendre β comme une variable de type et de modifier la traduction comme suit (avec l'abréviation de type vue plus haut) :

$$\begin{aligned} [\![f]\!] &\triangleq \lambda x_1.\lambda\beta{:}*\,.\lambda z.(z x_1) : \sigma \rightarrow \bigwedge \sigma \\ [\![\lambda(f\,x).A]\!] &\triangleq \lambda u.(u\ \tau\ \lambda x.[\![A]\!]) : (\bigwedge \sigma) \rightarrow \tau \end{aligned}$$

et alors on a bien $\vdash_\omega [\![\lambda(f\,x).A]\!][\![f\,B]\!] : \tau$ pour tout B tel que $[\![B]\!] : \sigma$.

Le cas général se traite de façon tout-à-fait semblable : le terme $[\![f]\!]$ a donc un type de la forme $\sigma_1 \rightarrow \ldots \sigma_{\alpha_f} \rightarrow \bigwedge \overline{\sigma}_{(1..\alpha_f)}$.

Un autre intérêt du polymorphisme est que la variable $x_\perp$ peut être utilisée partout où il nous faut un terme quelconque avec un type arbitraire : nous prendrons $x_\perp$ de type $\Pi(\iota : *).\iota$ (aussi écrit $\perp$), de sorte que si $[\![\Gamma]\!] \vdash_\omega \sigma : *$, alors $[\![\Gamma]\!] \vdash_\omega x_\perp \sigma : \sigma$. Ceci suffit à ce que l'usage que nous avons fait de $x_\perp$ dans la version non typée soit compatible avec le typage. Ainsi, dans la traduction de $\lambda(f\,P_1\,\ldots\,P_p).A$, les $\alpha_f - p$ occurrences de $x_\perp$ reçoivent le même type que les x'_n auxquels ils correspondent.

Traduction d'une variable

Un autre point clef est le typage des variables. En effet, pour respecter le lemme de substitution des PTS, notre traduction devrait affecter le même type à une variable et à tous les termes susceptibles de l'instancier. Or, pour l'instant, des ρ-termes ayant le même type dans $\rho_\to$ peuvent être traduits par des λ-termes ayant des types assez différents. Considérons les exemples suivants, où $\Sigma = \{\iota{:}*, a{:}\iota, f : \Pi y{:}\iota.\iota\}$:

$$\begin{aligned} x : \Pi y{:}\iota.\iota \;&\vdash_\Sigma x &&: \Pi y{:}\iota.\iota \\ &\vdash_\Sigma \lambda y{:}\iota.y &&: \Pi y{:}\iota.\iota \\ &\vdash_\Sigma \lambda y{:}\iota.a &&: \Pi y{:}\iota.\iota \\ &\vdash_\Sigma f &&: \Pi y{:}\iota.\iota \end{aligned}$$

Les trois termes $\lambda y.y$, $\lambda y.a$ et f sont susceptibles d'instancier x puisqu'ils ont le même type. En revanche, si on suppose le type ι traduit par un type β_y, la traduction donne

$$\begin{aligned} &\vdash_\omega \lambda y{:}\beta_y.y &&: \beta_y \to \beta_y \\ &\vdash_\omega \lambda y{:}\beta_y.[\![a]\!] &&: \beta_y \to \bigwedge \emptyset \\ &\vdash_\omega [\![f]\!] &&: \beta_y \to \bigwedge \beta_y \end{aligned}$$

On voit que la forme du type est toujours la même, à l'exception du type de retour, qui peut éventuellement utiliser β_y. Par conséquent, plutôt que de traduire $\Pi y{:}\iota.\iota$ par $\beta_y \to \beta_y$, on se donne une variable de type supplémentaire β_x qui permettra de construire le type de retour. La traduction de $\Pi y{:}\iota.\iota$ devient alors $\beta_y \to \beta_x\beta_y$, où β_x est susceptible d'être instanciée par un type dépendent d'un type, ce qui justifie l'usage du système $\lambda\omega$.

Dorénavant, lorsqu'on traduira une abstraction $\lambda x.A$, il faudra également ajouter la variable β_x dans le contexte ; lorsqu'on traduira une application $A\,B$, il faudra abstraire sur β_x et passer un argument supplémentaire qui permet de construire le type de retour attendu.

Dans nos exemples, en supposant que $\beta_y : *$, la variable β_x doit être instanciée comme suit :

$\lambda y.y$	$\beta_x := \lambda\beta{:}*.\beta$	$(\lambda\beta{:}*.\beta)\,\beta_y \mapsto_\beta \beta_y$
$\lambda y.a$	$\beta_x := \lambda\beta{:}*.\bigwedge\emptyset$	$(\lambda\beta{:}*.\bigwedge\emptyset)\,\beta_y \mapsto_\beta \bigwedge\emptyset$
f	$\beta_x := \lambda\beta{:}*.\bigwedge\beta$	$(\lambda\beta{:}*.\bigwedge\beta)\,\beta_y \mapsto_\beta \bigwedge\beta_y$

Nous définirons par la suite une fonction $\mathbb{K}(\cdot\ ;\ \cdot)$ chargée de calculer un genre convenable pour les variables β_x. Par exemple ici il faut que $\beta_x : * \to *$.

Traduction d'une contrainte de filtrage Un troisième problème qui peut se poser dans la traduction typée est la présence de contraintes de filtrage dans les types de $\rho_\to$. Lorsqu'une telle contrainte peut être résolue, nous la simplifierons systématiquement pour se ramener le plus possible au cas des types sous forme de Π-abstractions.

Si toutefois un terme $(\lambda P.A)\,B$ a pour type $[P \ll B]C$ et que cette contrainte ne peut être résolue, nous la représenterons de la façon suivante : le terme sera traduit par $[\![\lambda P.A]\!](w[\![B]\!])$, où w est une variable fraîche que nous appellerons *variable d'ajournement*. En prenant $\vdash_\omega w : \sigma \to \tau_1$ si $\vdash_\omega [\![B]\!] : \sigma$, on aura donc bien $\vdash_\omega [\![(\lambda P.A)\,B]\!] : \tau_2$ mais la réduction sera bloquée tant que w n'est pas instanciée.

Si, à la suite d'une application ultérieure qui transforme B en B', la contrainte peut être résolue, cela assurera que P et B' ont le même type ; on instanciera alors w par l'identité, ce qui supprimera l'encodage de la contrainte dans la traduction également. Ce mécanisme est détaillé dans la traduction typée.

La traduction typée

Comme nous allons produire des λ-termes typés (à la Church), nous devrons indexer chaque λ-abstraction par le type de la variable qu'elle lie. Par conséquent, la première étape de la traduction typée est de définir des traductions pour certains types.

Comme annoncé, pour toute variable x du ρ-terme à traduire, nous allons utiliser une variable de (constructeur de) type β_x. Commençons donc par définir la fonction $\mathbb{K}(C\ ;\ \overline{k})$ qui calcule un genre approprié pour ces nouvelles variables : l'idée est simplement de collecter dans $\overline{k}$ les genres de toutes les variables qui sont liées dans C. Si $x : C$, alors on aura $\beta_x : \mathbb{K}(C\ ;\ )$.

$$\begin{aligned}
\mathbb{K}(\iota\ ;\ \overline{k}) &\triangleq \overline{k} \rightarrow * \\
\mathbb{K}(\Pi P{:}\Delta.C\ ;\ \overline{k}) &\triangleq \mathbb{K}(C\ ;\ \overline{k}, \overline{\mathbb{K}(C_y\ ;\ )}_{((y:C_y)\in\Delta)}) \\
\mathbb{K}([P \ll_\Delta B]C\ ;\ \overline{k}) &\triangleq \mathbb{K}(C\ ;\ \overline{k})
\end{aligned}$$

Ensuite, nous pouvons définir les traductions de types proprement dites. Il nous faut pour cela trois traductions mutuellement dépendantes.

$(\!|P|\!)$ consiste, à partir du motif P, à trouver le type que doit avoir la variable u dans la traduction d'une abstraction $\lambda P.A$, ce qui est essentiel pour le typage des abstractions.

Les types que l'on va obtenir correspondent à l'idée qu'on a pu s'en faire au paragraphe précédent, lorsqu'on a étudié le typage des constantes. Par exemple, on aura

$$(\!|f\ (g\ x_1)\ x_2|\!) = \bigwedge \left((\wedge[\![\sigma_1]\!]^{x_1}), [\![\sigma_2]\!]^{x_2}\right)$$

ce qui correspond bien à la façon dont nous avons traduit le filtrage : la forme des motifs est conservée, mais les constantes de tête sont oubliées. La définition générale de $(\!|\ |\!)$ est la suivante :

$$\begin{aligned}
(\!|f\ \overline{P}\ ;\ \Delta|\!) &\triangleq [\![C]\!]^f_{\overline{(\!|P\ ;\ \Delta|\!)}} && \text{si } \Delta \vdash_\Sigma f\ \overline{P} : C \\
(\!|x\ ;\ \Delta|\!) &\triangleq [\![C]\!]^x && \text{si } \Delta \vdash x : C
\end{aligned}$$

Pour alléger les notations, on omettra généralement le contexte Δ dans lequel sont déclarés les types des variables de P, et on écrira simplement $(\!|P|\!)$.

$[\![C]\!]^x_{\overline{\gamma}}$ traduit le type C lorsqu'il est le type de la variable x. Intuitivement, il s'agit de traduire les différentes Π-abstractions rencontrées dans C à l'aide de la fonction $(\!|\ |\!)$, de collecter dans γ les β_y correspondant à toutes les variables liées dans C, et enfin d'avoir le type de retour $\beta_x\overline{\gamma}$, comme on l'a vu au paragraphe précédent sur le typage des variables. Les diverses variables de type et les variables d'ajournement $w_{(P \ll B)}$ seront ajoutées au contexte.

$$\begin{aligned}
[\![\iota]\!]^x_{\overline{\gamma}} &\triangleq \beta_x\overline{\gamma} && \text{si } \iota \text{ est atomique} \\
[\![\Pi P{:}\Delta.C]\!]^x_{\overline{\gamma}} &\triangleq (\!|P|\!) \rightarrow [\![C]\!]^x_{\overline{\gamma},\overline{\beta_y}} && \text{où les } \overline{\beta_y} \text{ correspondant aux } y \in \mathcal{D}om(\Delta) \\
[\![[P \ll_\Delta B]C]\!]^x_{\overline{\gamma}} &\triangleq [\![C]\!]^x_{\overline{\gamma}}
\end{aligned}$$

$[\![C]\!]^f_{\overline{\tau}}$ traduit le type C lorsqu'il est le type d'une constante, et il diffère de $[\![C]\!]^x$ uniquement par le type de retour, qui ne sera pas de la forme $\beta_x\overline{\gamma}$ mais plutôt $\bigwedge\overline{\tau}$.

Cette définition n'est donc pas indispensable puisqu'on pourrait la retrouver par instanciation judicieuse d'un β_x, mais nous la conservons pour ne pas trop obscurcir la définition

de $(\!|\ |\!)$.

$$[\![\iota]\!]^f_{\overline{\tau}} \triangleq \bigwedge \overline{\tau} \qquad \text{si } \iota \text{ est atomique}$$

$$[\![\Pi P{:}\Delta.C]\!]^f_{\overline{\tau}} \triangleq (\!|P|\!) \rightarrow [\![C]\!]^f_{\overline{\tau},(\!|P|\!)}$$

$$[\![[P \ll_\Delta B]C]\!]^f_{\overline{\tau}} \triangleq [\![C]\!]^f_{\overline{\tau}}$$

Munis de ces traductions de types, nous pouvons maintenant donner la traduction des contextes, en figure 5.4.

Comme, au cours de la traduction, on ajoute et on retire des variables de types et des variables d'ajournement dans le contexte, on définit $[\![\Gamma/A;C]\!]$ qui est la traduction du contexte Γ sachant qu'on veut y typer $[\![A]\!]$. Le troisième argument C est initialisé au type de A, et dans certains cas il est parcouru pour obtenir la traduction du contexte. Le cas de base est donné par $[\![\Gamma/\ ;\ ]\!]$ qui sera commun à toutes les traductions. Comme il a été dit plus haut, pour pouvoir utiliser la variable $x_\perp$ partout où un terme quelconque est nécessaire, nous lui donnons le type $\perp$, de sorte que pour tout type σ on a $\vdash_\omega x_\perp \sigma : \sigma$.

La traduction des termes est donnée en figure 5.5. Elle diffère de la traduction non typée essentiellement par les types manipulés dans la traduction des constantes et des abstractions, et au niveau de l'application.

Lors de la traduction de l'abstraction sur une variable x, on ajoute également un certain nombre de radicaux qui retranscrivent les termes apparaissant dans le type de x, et ce pour que les réductions qui ont lieu dans les contextes Δ ne soient pas oubliées dans la traduction (dans un type donné le nombre de σ-réductions est borné mais dans $[P \ll B]C$ le terme B peut se réduire *a priori* indéfiniment). Cette technique est classique dans les traductions de systèmes dont les types peuvent comporter des termes.

La traduction de l'application dans le cas d'une contrainte résoluble utilise une notion de *mise à jour des variables* engendrée par une contrainte, définie comme suit.

Définition 42 (Mise à jour des variables engendrée par une contrainte)
Soit une contrainte de filtrage $P \ll_\Delta B$.
La mise à jour des variables *engendrée par cette contrainte est une substitution θ telle que :*
- *Si $[\![\Gamma/B;]\!] \vdash_\omega [\![B]\!] : (\!|P|\!)[\overline{\beta_x := \tau_{x(x\in\mathcal{D}om(\Delta))}}]$, alors*
  1. *Pour tout $x \in \mathcal{D}om(\Delta)$, on a $\theta(\beta_x) = \tau_x$.*
  2. *On pose $\theta(w_{(P\ll B)}) = \lambda x{:}(\!|P|\!)[\overline{\beta_x := \tau_{x(x\in\mathcal{D}om(\Delta))}}].x$*
  3. *Pour toute $w_{Q\ll D}$ dont le type comporte une variable $\beta_x \in \mathcal{D}om(\theta)$, ajouter à θ la mise à jour des variables engendrée par $Q \ll D\theta$.*
- *Sinon, $\theta(w_{(P\ll B)}) = w'_{(P\ll B)}$ de type $\sigma \rightarrow (\!|P|\!)\theta$, où σ est le type de $[\![B]\!]$.*

On notera que θ est construite récursivement, mais que cette récursion est bien fondée : la contrainte $P \ll B$ ne peut mettre à jour une variable $w_{Q\ll D}$ que si $\mathcal{FV}(D) \cap \mathcal{FV}(P) \neq \emptyset$, ce qui engendre un ordre bien fondé sur les contraintes. On notera également que :
- à une variable de type, θ associe un type ;
- à une variable d'ajournement, θ associe soit une variable d'ajournement, soit l'identité.

Il peut sembler superflu de remplacer w par w' dans le cas où la contrainte n'est pas résoluble, mais pour pouvoir lier une variable β_x il faut qu'elle n'apparaisse dans le type d'aucune variable libre, et il faudra donc lier w avant de pouvoir lier les β_x.

$$
\begin{array}{rcl}
[\![\emptyset/\ ;\ ]\!] & \triangleq & x_\perp : \perp \\
[\![\Gamma, x{:}C/\ ;\ ]\!] & \triangleq & [\![\Gamma/\ ;\ ]\!],\ \beta_x : \mathbb{K}(C\ ;\ ),\ x : [\![C]\!]^x \qquad (\text{ si } \Gamma \vdash_\Sigma C : *) \\
[\![\Gamma, x{:}*/\ ;\ ]\!] & \triangleq & [\![\Gamma/\ ;\ ]\!],\ x : *
\end{array}
$$

$$
\begin{array}{rcl}
[\![\Gamma/x; \Pi P_n{:}\Delta_n.C]\!] & \triangleq & [\![\Gamma/x; C]\!], \overline{\beta_y{:}\mathbb{K}(C_y\ ;\ )}_{((y:C_y)\in\Delta_n)} \\
[\![\Gamma/x; [P \ll_\Delta B]C]\!] & \triangleq & [\![\Gamma/x; C]\!] \\
[\![\Gamma/x; \iota]\!] & \triangleq & [\![\Gamma/\ ;\ ]\!], \beta_x{:}\mathbb{K}(C_x\ ;\ ) \\
[\![\Gamma/f; \Pi P_n{:}\Delta_n.C]\!] & \triangleq & [\![\Gamma/f; C]\!], \overline{\beta_y{:}\mathbb{K}(C_y\ ;\ )}_{((y:C_y)\in\Delta_n)} \\
[\![\Gamma/f; [P \ll_\Delta B]C]\!] & \triangleq & [\![\Gamma/f; C]\!] \\
[\![\Gamma/f; \iota]\!] & \triangleq & [\![\Gamma/\ ;\ ]\!] \\
[\![\Gamma/A \wr B; C]\!] & \triangleq & [\![\Gamma/A; C]\!], [\![\Gamma/B; C]\!] \\
[\![\Gamma/\lambda x{:}C.A; \Pi x{:}C.D]\!] & \triangleq & [\![\Gamma, x{:}C/A; D]\!] \ \setminus\ x \\
[\![\Gamma/\lambda P{:}\Delta.A; \Pi P{:}\Delta.C]\!] & \triangleq & [\![\Gamma, \Delta/A; C]\!] \ \setminus\ \mathcal{D}om(P) \\
[\![\Gamma/A\,B; [P \ll_\Delta B]C]\!] & \triangleq & \big([\![\Gamma/A; \Pi P{:}\Delta.C]\!] \ \setminus\ \overline{\beta_x}, \overline{w}\big)\ ,\ [\![\Gamma/B; D]\!]\ ,\ \overline{w'} \\
 & & \qquad \text{si } \exists \overline{\tau_x},\ [\![\Gamma/B; D]\!] \vdash_\omega [\![B]\!] : (\!|P|\!)[\overline{\beta_x := \tau_x}]_{x\in\mathcal{D}om(\Delta)} \\
 & & \text{où } \overline{\beta_x}, \overline{w} \text{ sont les variables mises à jour par } P \ll_\Delta B \\
 & & \text{et } \overline{w'} \text{ sont les variables d'ajournement créées par la mise à jour} \\
[\![\Gamma/A\,B; [P \ll_\Delta B]C]\!] & \triangleq & [\![\Gamma/A; \Pi P{:}\Delta.C]\!]\ ,\ [\![\Gamma/B; D]\!]\ ,\ w_{(P\ll B)} : \sigma \to (\!|P|\!) \\
 & & \qquad \text{si } [\![\Gamma/B; D]\!] \vdash_\omega [\![B]\!] : \sigma \text{ et } \nexists \overline{\tau_x},\ \sigma =_\beta (\!|P|\!)[\overline{\beta_x := \tau_x}]_{x\in\mathcal{D}om(\Delta)}
\end{array}
$$

FIG. 5.4 – Traduction des contextes

$$
\begin{array}{rcl}
[\![x]\!] & \triangleq & x \\
[\![f]\!] & \triangleq & \lambda\overline{x{:}(\!|P|\!)}_{(1..\alpha_f)}.\lambda\beta{:}*\,.\lambda z{:}(\overline{(\!|P|\!)}_{(1..\alpha_f)} \rightarrow \beta).(z\overline{x}_{(1..\alpha_f)}) \\
 & & \text{où } \vdash_\Sigma f : \Pi\overline{P{:}\Delta}_{(1..\alpha_f)}.\iota \\
[\![A \wr B]\!] & \triangleq & \lambda\overline{x{:}(\!|P|\!)}_{(1..\alpha_{A\wr B})}.\left(\lambda z{:}\sigma.[\![A]\!]\,\overline{x}_{(1..\alpha_{A\wr B})}\right)\,([\![B]\!]\,\overline{x}_{(1..\alpha_{A\wr B})}) \\
 & & \text{où } \Gamma \vdash_\Sigma A \wr B : \Pi\overline{P{:}\Delta}_{(1..\alpha_{A\wr B})}.\iota \text{ et } [\![\Gamma/B;C]\!] \vdash_\omega [\![B]\!]\overline{x} : \sigma \\
[\![\lambda x{:}C.A]\!] & \triangleq & \lambda x{:}[\![C]\!]^x\,.\,(\lambda\overline{y{:}\tau}.[\![A]\!])\,\overline{[\![D]\!]} \\
 & & \text{où } \overline{D} \text{ sont les termes apparaissant dans } C \text{ et } \overline{[\![\Gamma]\!] \vdash_\omega [\![D]\!] : \tau} \\
[\![\lambda(f\,\overline{P}_{(1..p)}){:}\Delta.A]\!] & \triangleq & \lambda u{:}(\!|f\,\overline{P}|\!).(u\overline{(x_\perp\,(\!|P'|\!))}_{(p+1..\alpha_f)}\ \ \tau\ [\![\lambda\overline{P{:}\Delta}_{(1..p)}.\lambda\overline{x'{:}(\!|P'|\!)}_{(p+1..\alpha_f)}.A]\!]) \\
 & & \text{où } \Delta \vdash_\Sigma f\,\overline{P}_{(1..p)} : \Pi\overline{P'{:}\Delta}_{(p+1..\alpha_f)}.\iota \quad \text{et} \quad [\![\Gamma/A;C]\!] \vdash_\omega [\![A]\!] : \tau \\
[\![A\,B]\!] & \triangleq & \left(\lambda\overline{\beta_x}.\lambda\overline{w}.[\![A]\!]\right)\,\overline{\theta(\beta_x)}\,\overline{\theta(w)}\,[\![B]\!] \\
 & & \text{où } \overline{\beta_x}, \overline{w} \text{ sont les variables mises à jour par } P \ll B \\
 & & \qquad \text{si } \exists\overline{\tau_x},\ [\![\Gamma/B;D]\!] \vdash_\omega [\![B]\!] : (\!|P|\!)[\overline{\beta_x := \tau_x}]_{x\in\mathcal{D}om(\Delta)} \\
[\![A\,B]\!] & \triangleq & [\![A]\!]\,(w\,[\![B]\!]) \\
 & & \qquad \text{si } \nexists\overline{\tau_x},\ [\![\Gamma/B;D]\!] \vdash_\omega [\![B]\!] : (\!|P|\!)[\overline{\beta_x := \tau_x}]_{x\in\mathcal{D}om(\Delta)}
\end{array}
$$

FIG. 5.5 – Traduction typée

5.2.3 Correction de la traduction typée

Nous démontrons que cette seconde traduction préserve le typage et les réductions, ce qui nous permettra de démontrer la normalisation forte (théorème 57).

Lemme 54 (Correction du genre)
Si $\Gamma \vdash_\Sigma C : *$ *alors pour un* x *frais quelconque* $[\![\Gamma/x;C]\!] \vdash_\omega [\![C]\!]^x : *$.

Démonstration : En particulier, dans $[\![\Gamma/x;C]\!]$, on trouve $\beta_x{:}\mathbb{K}(C\ ;\ )$. La démonstration est immédiate : par récurrence, dans la définition de $\mathbb{K}(C\ ;\ \overline{k})$, les genres calculés pour les β_y sont corrects, et donc celui calculé pour β_x l'est aussi. □

Théorème 55 (Préservation du typage [Wac04])
Si $\Gamma \vdash_\Sigma A : C : *$ *alors, pour une variable fraîche* z, $\exists\tau_A,\ [\![\Gamma/A;C]\!] \vdash_\omega [\![A]\!] : [\![C]\!]^z[\beta_z := \tau_A]$

Démonstration : Nous allons démontrer que τ_A peut être défini comme suit :

$$
\begin{array}{rcll}
\tau_x & \triangleq & \beta_x & \\
\tau_f & \triangleq & \lambda\overline{\beta_y}.\bigwedge\overline{(\!|P_n|\!)}_{(1..\alpha_f)} & \text{si } f : \Pi P_1 \ldots \Pi P_{\alpha_f}.\iota \\
\tau_{A\wr B} & \triangleq & \tau_A & \\
\tau_{\lambda P.A} & \triangleq & \lambda\overline{\beta_x}.\tau_A & \text{si } \mathcal{FV}(P) = \overline{x} \\
\tau_{A\,B} & \triangleq & \tau_A\,\overline{\tau_x} & \text{si } \exists\overline{\tau_x},\ [\![\Gamma/B;D]\!] \vdash_\omega [\![B]\!] : (\!|P|\!)[\overline{\beta_x := \tau_x}] \\
\tau_{A\,B} & \triangleq & \tau_A\,\overline{\beta_x} & \text{sinon}
\end{array}
$$

Il est assez facile de voir que si deux ρ-types C et C' sont convertibles modulo $=_{\rho\delta}$, alors $[\![C]\!]^z =_\beta [\![C']\!]^z$, et donc on peut choisir un représentant quelconque du type de A. La démonstration se fait par récurrence sur la structure de A. Rappelons que A est un terme et ne comporte donc pas de contrainte de filtrage.

Pour une variable x on a

$$(\textsc{Var})\ \frac{\Gamma \vdash_\Sigma C : s}{\Gamma, x{:}C \vdash_\Sigma x : C}$$

La sorte s est forcément $*$. Par le lemme 54, on a $[\![\Gamma/x; C]\!] \vdash_\omega [\![C]\!]^x : *$, d'où

$$(\textsc{Var})\ \frac{[\![\Gamma/\ ;\ ]\!], \beta_x : \mathbb{K}(C\ ;\ ) \vdash_\omega [\![C]\!]^x : *}{[\![\Gamma, x{:}C/\ ;\ ]\!] \vdash_\omega x : [\![C]\!]^x}$$

Pour une constante f on a

$$(\textsc{Const})\ \frac{\Sigma\ sig \qquad f : C \in \Sigma}{\vdash_\Sigma f : C}$$

Alors, dans la dérivation de $\Sigma\ sig$, on trouve $\vdash_{\Sigma'} C : s$ pour un certain préfixe Σ' de Σ, et la sorte s est forcément $*$. Il nous faut donc montrer que $[\![\emptyset/f; C]\!] \vdash_\omega [\![f]\!] : [\![C]\!]^f$. Les contraintes de filtrage qui apparaissent dans C n'étant pas traduites dans $[\![C]\!]^f$, il suffit de constater que les variables $\overline{x}_{(1..\alpha_f)}$ liées dans $[\![f]\!]$ ont bien les types $\overline{(\!|P|\!)}_{(1..\alpha_f)}$ correspondant.

Quant au type de retour, par définition de $[\![C]\!]^f$ il s'agit de $\bigwedge \overline{(\!|P|\!)}_{(1..\alpha_f)}$, qui est bien un type valide pour $\lambda\beta{:}*.\lambda z.(z\overline{x}_{(1..\alpha_f)})$.

Il ne nous reste plus qu'à exprimer ce type comme $[\![C]\!]^z[\beta_z := \tau_f]$ pour un certain τ_f. Il suffit pour cela que τ_f reconstruise les $(\!|P|\!)$ à partir des $\overline{\beta_y}$ présents dans $[\![\Gamma/f; C]\!]$, ce qui est immédiat (à α-conversion près des β_y) :

$$\tau_f \ \triangleq\ \lambda\overline{\beta_y}.\bigwedge \overline{(\!|P_n|\!)}_{(1..\alpha_f)}$$

Remarquons que, d'après le lemme 54 et par définition de $\bigwedge$, on a $[\![\Delta]\!] \vdash_\omega \bigwedge \overline{(\!|P_n|\!)}_{(1..\alpha_f)} : *$ et donc τ_f a bien le même genre que β_z.

Pour une structure $A \wr B$ on a

$$(\textsc{Struct})\ \frac{\Gamma \vdash_\Sigma A : C \qquad \Gamma \vdash_\Sigma B : C}{\Gamma \vdash_\Sigma A \wr B : C}$$

Par hypothèse de récurrence $[\![\Gamma/A; C]\!] \vdash [\![A]\!] : [\![C]\!]^z[\beta_z := \tau_A]$ et $[\![\Gamma/B; C]\!] \vdash [\![B]\!] : [\![C]\!]^z[\beta_z := \tau_B]$. Rappelons que la structure n'est pas traduite comme la paire usuelle du λ-calcul mais comme $(\lambda z.[\![A]\!])[\![B]\!]$, et donc c'est bien le type de $[\![A]\!]$ qui sera celui de $[\![A \wr B]\!]$, d'où $\tau_{A\wr B} = \tau_A$.

Il ne reste qu'à constater que les variables $x_1 \ldots x_{\alpha_{A\wr B}}$ liées dans $[\![A \wr B]\!]$ ont bien les types $(\!|P_n|\!)$ correspondants, ce qui est immédiat. Le contexte $[\![\Gamma/A \wr B; C]\!]$ est défini comme l'union des deux contextes, ce qui permet bien de typer l'ensemble du terme $[\![A \wr B]\!]$, à condition d'utiliser les mêmes β_y dans $[\![A]\!]$, dans $[\![B]\!]$ et dans les types des $\overline{x}_{(1..\alpha_{A\wr B})}$.

Pour une abstraction $\lambda P{:}\Delta.A$ on a

$$(\textsc{Abs})\ \frac{\Gamma, \Delta \vdash_\Sigma A : C \qquad \Gamma \vdash_\Sigma \Pi P{:}\Delta.C : s}{\Gamma \vdash_\Sigma \lambda P{:}\Delta.A : \Pi P{:}\Delta.C}$$

Alors nécessairement $s \equiv *$.

Si P est réduit à une variable x alors dans la traduction x a pour type $[\![C_x]\!]^x$.Par hypothèse de récurrence $[\![\Gamma, x{:}C_x/A; C]\!] \vdash_\omega [\![A]\!] : [\![C]\!]^z[\beta_z := \tau_A]$, et les termes $[\![D]\!]$ sont typables (c'est la seule information nécessaire pour typer les radicaux $(\lambda y.\ldots)\,[\![D]\!]$, qui n'influent pas sur le reste du typage). On en déduit que $[\![\Gamma/\lambda x.A; \Pi x.C]\!] \vdash_\omega [\![\lambda x.A]\!] : [\![\Pi x.C]\!]^z[\beta_z := \lambda\beta_x.\tau_A]$.
Sinon, dans $[\![\lambda(f\,\overline{P}_{(1..p)}).A]\!]$, la variable u a bien pour type $(\!|f\,\overline{P}_{(1..p)}|\!)$. Il suffit donc de vérifier que $u\overline{(x_\perp\,(\!|P'|\!))}_{(p+1..\alpha_f)}\ \tau\ [\![\lambda\overline{P{:}\Delta}_{(1..p)}.\lambda\overline{x'{:}\,(\!|P'|\!)}_{(p+1..\alpha_f)}.A]\!]$ a pour type $[\![C]\!]^z[\beta_z := \tau]$ pour un certain τ.
Par définition $(\!|f\,\overline{P}_{(1..p)}|\!) \equiv \overline{(\!|P'|\!)}_{(p+1..\alpha_f)} \to \bigwedge \overline{(\!|P|\!)}_{(1..p)}, \overline{(\!|P'|\!)}_{(p+1..\alpha_f)}$.
Les arguments $x_\perp\,(\!|P'|\!)$ absorbent bien les premiers arguments attendus par u, puis τ instancie la variable de type liée dans $\bigwedge \overline{(\!|P|\!)}_{(1..p)}, \overline{(\!|P'|\!)}_{(p+1..\alpha_f)}$. Enfin, par hypothèse de récurrence, on a

$$[\![\Gamma/\lambda\overline{P}.\lambda\overline{x'}.A; \Pi\overline{P}.\Pi\overline{P'}.C]\!] \vdash_\omega [\![\lambda\overline{P{:}\Delta}_{(1..p)}.\lambda\overline{x'{:}\,(\!|P'|\!)}_{(p+1..\alpha_f)}.A]\!] : [\![\Pi\overline{P}_{(1..p)}.\Pi\overline{P'}_{(p+1..\alpha_f)}.C]\!]^z[\beta_z := \tau]$$

avec $\tau \equiv \lambda\overline{\beta_x}.\tau_A$ où $\overline{\beta_x} = \mathcal{FV}(\overline{P}, \overline{P'})$. Les variables libres des P' ont été introduites uniquement pour la traduction, et donc on peut se dispenser des β_x correspondants.
On en conclut donc que $[\![\Gamma/\lambda f\,\overline{P}_{(1..p)}.A; \Pi f\,\overline{P}_{(1..p)}.C]\!] \vdash_\Sigma [\![\lambda f\,\overline{P}_{(1..p)}.A]\!] : (\!|f\,\overline{P}_{(1..p)}|\!) \to [\![C]\!]^z[\beta_z := \lambda\overline{\beta_x}_{(x\in\mathcal{FV}(\overline{P}))}.\tau_A]$ et on a

$$(\!|f\,\overline{P}_{(1..p)}|\!) \to [\![C]\!]^z[\beta_z := \lambda\overline{\beta_x}_{(x\in\mathcal{FV}(\overline{P}))}.\tau_A] \;=\; [\![\Pi f\,\overline{P}_{(1..p)}.C]\!]^z[\beta_z := \lambda\overline{\beta_x}_{(x\in\mathcal{FV}(\overline{P}))}.\tau_A]$$

On trouve donc bien

$$\tau_{\lambda P.A} \triangleq \lambda\overline{\beta_x}.\tau_A$$

Pour une application $A\,B$ on a

$$(\textsc{Appl})\ \frac{\Gamma \vdash_\Sigma A : \Pi P{:}\Delta.C \qquad \Gamma \vdash_\Sigma [P \ll_\Delta B]C : s}{\Gamma \vdash_\Sigma A\,B : [P \ll_\Delta B]C}$$

Alors nécessairement $s \equiv *$.

On distingue les deux cas envisagés en figure 5.5.

Si $\exists\overline{\tau_x},\ [\![\Gamma/B; D]\!] \vdash_\omega [\![B]\!] : (\!|P|\!)[\overline{\beta_x := \tau_x}]$ alors on a $[\![A\,B]\!] = (\lambda\overline{\beta_x}.\lambda\overline{w}.[\![A]\!])\,\overline{\theta(\beta_x)}\,\overline{\theta(w)}\,[\![B]\!]$.
Par hypothèse de récurrence sur A on a $[\![\Gamma/A; \Pi P{:}\Delta.C]\!] \vdash_\omega [\![A]\!] : [\![\Pi P{:}\Delta.C]\!]^z[\beta_z := \tau_A]$.
Vérifions qu'il est légitime d'instancier certains $w_{Q\ll D}$ par l'identité : en effet, si la contrainte $Q \ll D$ est résoluble, alors $\vdash_\omega [\![D]\!] : (\!|Q|\!)[\overline{\beta_y := \tau_{y_{(y\in\mathcal{FV}(Q))}}}]$ et on a ajouté les abstractions $\lambda\overline{\beta_y}$ et les arguments τ_y correspondants. Par conséquent $\vdash_\omega w_{Q\ll D} : (\!|Q|\!)[\overline{\beta_y := \tau_{y_{(y\in\mathcal{FV}(Q))}}}] \to (\!|Q|\!)[\overline{\beta_y := \tau_{y_{(y\in\mathcal{FV}(Q))}}}]$ et on peut donc l'instancier par $\lambda x{:}\,(\!|Q|\!)[\overline{\beta_y := \tau_{y_{(y\in\mathcal{FV}(Q))}}}].x$. On a alors $\left([\![\Gamma/A; \Pi P{:}\Delta.C]\!] \setminus \overline{\beta_x}, \overline{w}\right), \overline{w'} \vdash_\omega (\lambda\overline{\beta_x}.\lambda\overline{w}.[\![A]\!])\,\overline{\tau_x}\,\overline{t} : [\![\Pi P{:}\Delta.C]\!]^z[\beta_z := \tau_A][\overline{\beta_x := \tau_x}]$. Or

$$\begin{aligned}
[\![\Pi P{:}\Delta.C]\!]^z[\beta_z := \tau_A][\overline{\beta_x := \tau_x}] &= (\!|P|\!)[\overline{\beta_x := \tau_x}] \to [\![C]\!]^z_{\overline{\beta_x}}[\beta_z := \tau_A][\overline{\beta_x := \tau_x}] \\
&= (\!|P|\!)[\overline{\beta_x := \tau_x}] \to [\![C]\!]^z_{\overline{\tau_x}}[\beta_z := \tau_A] \\
&= (\!|P|\!)[\overline{\beta_x := \tau_x}] \to [\![C]\!]^z[\beta_z := \tau_A\,\overline{\tau_x}]
\end{aligned}$$

Le type attendu pour l'argument correspond donc bien au type de $[\![B]\!]$, et le contexte de typage de l'application est l'union des deux contextes, d'où $[\![\Gamma/A\,B; [P \ll_\Delta B]C]\!] \vdash_\omega [\![A\,B]\!] : [\![C]\!]^z[\beta_z := \tau_A\,\overline{\tau_x}]$, et donc

$$\tau_{A\,B} \triangleq \tau_A\overline{\tau_x}$$

Si $\sharp\overline{\tau_x}$, $[\![\Gamma/B; D]\!] \vdash_\omega [\![B]\!] : (\!|P|\!)[\overline{\beta_x := \tau_x}]$, alors on a $[\![A\,B]\!] = [\![A]\!](w[\![B]\!])$ où w a pour type $\sigma \rightarrow (\!|P|\!)$ (où σ est le type de $[\![B]\!]$).
Par hypothèse de récurrence, $[\![\Gamma/A; \Pi P{:}\Delta.C]\!] \vdash_\omega [\![A]\!] : [\![\Pi P{:}\Delta.C]\!]^z[\beta_z := \tau_A]$, or

$$\begin{aligned}[\![\Pi P{:}\Delta.C]\!]^z[\beta_z := \tau_A] &= (\!|P|\!) \rightarrow [\![C]\!]^z_{\overline{\beta_x}}[\beta_z := \tau_A] \\ &= (\!|P|\!) \rightarrow [\![C]\!]^z[\beta_z := \tau_A\,\overline{\beta_x}]\end{aligned}$$

Le contexte de typage de l'application est l'union des deux contextes auxquels on a ajouté la variable w, d'où $[\![\Gamma/A\,B; [P \ll_\Delta B]C]\!] \vdash_\omega [\![A\,B]\!] : [\![C]\!]^z[\beta_z := \tau_A\,\overline{\beta_x}]$, et donc

$$\tau_{A\,B} \triangleq \tau_A\overline{\beta_x}$$

□

Lemme 56 (Préservation des réductions)
Le lemme 52 (stabilité de la traduction par substitution) et le théorème 53 (à toute $\rho\sigma\delta$-réduction correspond au moins une β-réduction) sont toujours valides dans la traduction typée.

Démonstration : La démonstration se fait de façon très similaire au cas non typé : nous n'avons fait qu'ajouter des λ-abstractions et des applications sur des types, sur les termes apparaissant dans les ρ-types, et sur des variables d'ajournement. Il suffit donc de vérifier que celles-ci se comportent correctement.

1. Les seules abstractions et applications sur les types apparaissent dans les traductions des constantes, des abstractions et des applications. Il est immédiat qu'à une abstraction de type correspond exactement une application de type.
   Il suffit donc de vérifier que les variables de types ont le bon genre (lemme 54) et que les λ-termes produits par $[\![\ ]\!]$ sont bien typés (théorème 55).
2. Les radicaux ajoutés dans $\lambda x{:}C.A$ permettent de traduire les réductions de la forme $\lambda P{:}\Delta.A \mapsto_{\rho\sigma\delta} \lambda P{:}\Delta'.A$, où Δ' est Δ dans lequel un type C a subi une réduction $\mapsto_{\rho\sigma\delta}$. Ce type C est nécessairement le type d'une variable x de P, et donc lors de la traduction $[\![\lambda P{:}\Delta.A]\!]$, on traduit un certain $\lambda x{:}C.A'$. Par conséquent, pour le sous-terme D de C dans lequel a lieu la réduction, le λ-terme $[\![D]\!]$ apparaît dans $[\![\lambda P{:}\Delta.A]\!]$ et donc par hypothèse de récurrence la réduction se traduit dans $[\![D]\!]$ également.
   Lorsqu'on veut traduire la réduction d'un radical $(\lambda P{:}\Delta.A)\,B$, il suffit de réduire d'abord tous les radicaux $(\lambda y.[\![A']\!])\,[\![D]\!]$ qui apparaissent dans $[\![(\lambda P{:}\Delta.A)\,B]\!]$. La réduction de ces radicaux ne pose pas de problème, car les variables $\overline{y}$ sont fraîches, et la ρ-réduction fait disparaître le contexte Δ du terme original, et il n'est donc plus nécessaire de retranscrire les termes $[\![D]\!]$ dans le terme traduit. On procède ensuite comme dans le cas non typé.
3. Ce qui est plus important est de vérifier que notre utilisation des variables d'ajournement est correcte. Nous démontrerons donc les deux propriétés suivantes.

(a) Les contraintes résolubles ne sont pas ajournées.
Soit un terme $(\lambda P.A)\,B$ tel que la contrainte $P \ll B$ soit résoluble. Alors

$$\exists \overline{\tau_x},\ [\![\Gamma/B; D]\!] \vdash_\omega [\![B]\!] : (\!|P|\!)[\overline{\beta_x := \tau_x}]_{x \in \mathcal{D}om(\Delta)}$$

où D est le type commun à P et B. Ceci assure en particulier que $(\lambda P.A)\,B$ n'est pas traduit par $[\![\lambda P.A]\!]\,(w\,[\![B]\!])$.
En effet, si cette contrainte est résoluble, on a $B = P\theta_{P \ll B}$. D'après le lemme 52, on a donc $[\![B]\!] = [\![P]\!][\![\theta_{P \ll B}]\!][\beta_x := \tau_{x\theta}]$. On verra dans le théorème 55 (qui n'utilise pas celui-ci) que $[\![\Gamma/P; D]\!] \vdash_\omega [\![P]\!] : [\![D]\!]^z[\beta_z := \tau_P] = (\!|P|\!)$.
Par ailleurs on a $[\![\Gamma/P; D]\!] \setminus \overline{\beta_x} \subseteq [\![\Gamma/B; D]\!]$ et

$$[\![D]\!]^z[\beta_z := \tau_B] = [\![D]\!]^z[\beta_z := \tau_P][\beta_x := \tau_{x\theta}] = (\!|P|\!)[\beta_x := \tau_{x\theta}]$$

Le typage étant stable par substitution dans $\lambda\omega$, on en déduit $[\![\Gamma/B; D]\!] \vdash_\omega [\![B]\!] : (\!|P|\!)[\beta_x := \tau_{x\theta}]$.

(b) Les contraintes devenues résolubles apparaissent dans le type.
Soit un ρ-terme $(\lambda P.A)\,B$ tel que la contrainte de filtrage $P \ll B$ soit irrésoluble. Si $(\lambda P.A)\,B$ est un sous-terme d'un terme A' dans lequel la contrainte devient résoluble, alors dans le typage de A' la contrainte devient résoluble.
Il suffit pour cela de considérer un genre de forme σ-longue pour les types. Procédons par récurrence sur la structure de A' :

Si A' est le terme A lui-même alors de façon triviale la contrainte apparaît dans le type $[P \ll B]C$ de A.

Si $A' \equiv A_1 \wr A_2$ alors nécessairement A' et A_1 et A_2 ont le même type. Par hypothèse de récurrence, la contrainte $[P \ll B]$ apparaît dans le type de celui des deux sous-termes A_1 et A_2 dont A est un sous-terme, et donc elle apparaît dans le type de A'.

Si $A' \equiv \lambda P.A_1$ alors par hypothèse de récurrence, la contrainte apparaît dans le type D de A_1, et donc elle apparaît dans le type $\Pi P.D$ de A'.

Si $A' \equiv A_1\,A_2$ où A est un sous-terme de A_1 par génération on a $\vdash A_1 : \Pi Q.C'$. La contrainte $[P \ll B]$ ayant été supposée irrésoluble, A est donc un sous-terme strict de A_1 ; par hypothèse de récurrence, on peut prendre C' tel que la contrainte y apparaît et donc elle apparaît dans le type $[Q \ll A_2]C'$ de A'. De plus si c'est la σ-réduction de la contrainte $Q \ll A_2$ qui rend la contrainte $P \ll B$, alors on a $[Q \ll A_2]C' =_\sigma C'\theta_{Q \ll A_2}$ où la contrainte $P \ll B\theta$ devient résoluble.

Si $A' \equiv A_1\,A_2$ où A est un sous-terme de A_2 alors par génération $\vdash A_1 : \Pi Q.C'$ et donc, comme A est un sous-terme de A_2, il apparaît (et donc la contrainte également) dans le type $[Q \ll A_2]C'$ de A'.

Cette propriété justifie que les contraintes de filtrage qui deviennent résolubles peuvent être détectées lors du typage, et donc également lors de la traduction. Les instanciations des variables d'ajournement effectuées lors de la traduction des applications sont donc suffisantes, et dans le λ-terme $[\![A']\!]$ il ne restera de variables d'ajournement que pour les échecs de filtrage définitifs.

□

Théorème 57 (Normalisation forte dans $\rho_\to$ [Wac04])
Si $\Gamma \vdash_\Sigma A : C$ dans le système $\rho_\to$ alors A est fortement normalisant.

Démonstration :

Si $\Gamma \vdash_\Sigma C : *$ alors, d'après le théorème 55, on a $\exists \tau,\ [\![\Gamma/A; C]\!] \vdash_\omega [\![A]\!] : [\![C]\!]^z[\beta_z := \tau]$. Supposons qu'il existe un chemin de réduction infini à partir de A. Par le lemme 56, il existe également un tel chemin à partir de $[\![A]\!]$. Mais par normalisation forte du système $\lambda\omega$ (théorème 4), le terme $[\![A]\!]$ ne peut avoir de réductions infinies, et donc par contradiction, A est fortement normalisant.

Si $\Gamma \vdash_\Sigma C : \Box$ alors A est un ρ-type. Vue la forme des types, les seules réductions qui peuvent avoir lieu dans A sont :
- des σ-réductions, qui sont en nombre borné par le nombre de contraintes de filtrage apparaissant dans A.
- des réductions dans les termes qui apparaissent dans A, qui sont finies d'après le cas précédent.

□

Pour étendre cette démonstration aux autres systèmes du ρ-cube, deux pistes sont possibles. D'une part, il est peut-être possible de trouver des traductions des ρ-termes dépendant de types et des ρ-types dépendant de types dans un système du λ-calcul ; une telle traduction ne serait cependant pas une extension immédiate de la précédente, dans la mesure où nous avons largement exploité la forme des types pouvant être formés dans $\rho_\to$. D'autre part, comme montré par exemple dans [GN91], en λ-calcul il est possible de traduire un système avec types dépendant de termes dans le système correspondant sur la face gauche du cube. Il est vraisemblable que le même genre de traduction fonctionne aussi systématiquement en ρ-calcul ; nous verrons par exemple dans la prochaine section comment encoder ρP dans $\rho_\to$, ce qui nous permettra de démontrer la normalisation forte pour ρP.

Revenons brièvement sur l'idée d'une démonstration par modèles. L'encodage vu ici fournit la base d'une interprétation, puisqu'on pourrait *a priori* interpréter un ρ-type comme l'interprétation de sa traduction. En revanche, plusieurs problèmes se posent : notamment, un type ne se traduit pas toujours de la même manière selon le terme qu'il type, et la variable $x_\bot : \bot$ est dans tous les contextes de typage. Il reste intéressant de constater qu'on a pu interpréter le type d'un motif comme une conjonction des types des différentes variables de ce motif, ce qui est assez fidèle à l'intention annoncée à la fin du chapitre 4.

5.2.4 Extension au système dépendant

Théorème 58 (Normalisation forte dans ρP)
Tout ρ-terme typable uniquement avec les règles de produit $(*,*)$ et $(*,\Box)$ est fortement normalisant.

Démonstration : Suivons les grandes lignes de la démonstration de normalisation forte de LF [HHP93]. Nous allons définir une traduction ε des sortes et des types, et une traduction $|\cdot|$ des ρP-termes et types en $\rho_\to$-termes et types, telle que $|\cdot|$ efface les types dépendants et préserve les réductions. Nous utiliserons une constante particulière 0 telle que $\vdash_\Sigma 0 : *$ et une famille de constantes $\pi_{P:\Delta}$ pour tout motif P et tout contexte Δ. La constante $\pi_{P:\Delta}$ a pour type

$\Pi x_1{:}0 \ldots \Pi x_n{:}0.\Pi y{:}(\Pi P{:}\varepsilon(\Delta).0).0$ où n est le nombre de variables libres dans P.

$$
\begin{array}{rcll}
\varepsilon(\Box) & \triangleq & 0 & \\
\varepsilon(*) & \triangleq & 0 & \\
\varepsilon(x) & \triangleq & x & \text{si } \vdash_\Sigma x : C : \Box \\
\varepsilon(f) & \triangleq & f & \text{si } \vdash_\Sigma f : C : \Box \\
\varepsilon(\Pi P{:}\Delta.C) & \triangleq & \Pi P{:}\varepsilon(\Delta).\varepsilon(C) & \\
\varepsilon(\lambda P{:}\Delta.A) & \triangleq & \varepsilon(A) & \\
\varepsilon([P \ll_\Delta B]C) & \triangleq & [P \ll_{\varepsilon(\Delta)} |B|]\varepsilon(C) & \\
\varepsilon(A\,B) & \triangleq & \varepsilon(A) &
\end{array}
$$

$$
\begin{array}{rcll}
|x| & \triangleq & x & \\
|f| & \triangleq & f & \\
|\Pi P{:}\Delta.C| & \triangleq & \pi_{P:\Delta}\ |C_1| \ldots |C_n|\ (\lambda P{:}\varepsilon(\Delta).|C|) & \\
 & & \multicolumn{2}{l}{\text{si } \Delta \equiv x_1{:}C_1 \ldots x_n{:}C_n} \\
|\lambda P{:}\Delta.A| & \triangleq & \lambda P{:}\varepsilon(\Delta).\big((\lambda y_1{:}0 \ldots \lambda y_n{:}0.|A|)\,|C_1| \ldots |C_n|\big) & \\
 & & \multicolumn{2}{l}{\text{si } \Delta \equiv x_1{:}C_1 \ldots x_n{:}C_n} \\
|A \wr B| & \triangleq & |A| \wr |B| & \\
|A\,B| & \triangleq & |A|\,|B| & \\
|[P \ll_\Delta B]C| & \triangleq & [P \ll_{\varepsilon(\Delta)} |B|]\ \big((\lambda y_1{:}0 \ldots \lambda y_n{:}0.|C|)\,|C_1| \ldots |C_n|\big) & \\
 & & \multicolumn{2}{l}{\text{si } \Delta \equiv x_1{:}C_1 \ldots x_n{:}C_n}
\end{array}
$$

La fonction ε est étendue aux contextes et aux signatures, où elle opère sur chaque type. La correction de ces fonctions est assurée par trois lemmes :

1. Si $\Gamma \vdash_\Sigma B : C : \Box$ ou $\Gamma \vdash_\Sigma B : \Box$ dans ρP, alors $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} \varepsilon(B) : *$ dans $\rho_\to$.
2. Si $\Gamma \vdash_\Sigma A : C$ dans ρP, alors $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} |A| : \varepsilon(C)$ dans $\rho_\to$.
3. Si $A \mapsto_{\beta\delta} A'$, alors $|A| \mapsto\!\!\!\!\to_{\beta\delta} |A'|$ en au moins un pas.

□

Lemme 59

Si $\Gamma \vdash_\Sigma B : C : \Box$ ou $\Gamma \vdash_\Sigma B : \Box$ dans ρP, alors $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} \varepsilon(B) : *$ dans $\rho_\to$.

Démonstration : Par récurrence sur la structure de B. Immédiat si $\vdash_\Sigma B : \Box$. Les seuls cas intéressants pour $\vdash_\Sigma B : C : \Box$ sont l'abstraction et l'application :

Si $B \equiv \lambda P{:}\Delta.B_1$ alors par génération $C =_{\beta\delta} \Pi P{:}\Delta.C_1$ avec $\Gamma, \Delta \vdash_\Sigma B_1 : C_1$ et $\Gamma \vdash_\Sigma \Pi P{:}\Delta.C_1 : \Box$, et donc par une seconde étape de génération $\Gamma, \Delta \vdash_\Sigma C_1 : \Box$.
Par hypothèse de récurrence $\varepsilon(\Gamma, \Delta) \vdash_{\varepsilon(\Sigma)} \varepsilon(B_1) : *$, et donc il en va de même pour $\lambda P{:}\Delta.B_1$.

Si $B \equiv B_1\,B_2$ alors par génération $C =_{\beta\delta} [P \ll B_2]C_1$ avec $\Gamma \vdash_\Sigma B_1 : \Pi P.C_1$ et $\Gamma \vdash_\Sigma [P \ll B_2]C_1 : \Box$. Par une seconde étape de génération on trouve $\Gamma \vdash_\Sigma C_1 : \Box$, et donc on a $\Gamma \vdash_\Sigma B_1 : \Pi P.C_1 : \Box$, d'où par hypothèse de récurrence $\varepsilon(\Gamma) \vdash_\Sigma \varepsilon(B_1\,B_2) = \varepsilon(B_1) : *$

□

Lemme 60

Si $\Gamma \vdash_\Sigma A : C$ dans ρP, alors $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} |A| : \varepsilon(C)$ dans $\rho_\to$.

Démonstration : Par récurrence sur une dérivation de $\Gamma \vdash_\Sigma A : C$, en distinguant selon la dernière règle de typage employée.

Si la dernière règle est (Sig), (WeakΣ) **ou (Axiom)** c'est immédiat. Pour toute prémisse $\vdash_\Sigma C : s$, d'après le lemme 59 on a $\vdash_{\varepsilon(\Sigma)} \varepsilon(C) : *$.

Si la dernière règle est

$$(\textsc{Var})\ \frac{\Gamma \vdash_\Sigma C : s}{\Gamma, x{:}C \vdash_\Sigma x : C}$$

D'après le lemme 59 on a $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} \varepsilon(C) : *$. Comme $\varepsilon(\Gamma, x{:}C)(x) = \varepsilon(C)$ on a bien $\varepsilon(\Gamma, x{:}C) \vdash_{\varepsilon(\Sigma)} x : \varepsilon(C)$.

Si la dernière règle est

$$(\textsc{Const})\ \frac{\Sigma\ sig \qquad f : C \in \Sigma}{\vdash_\Sigma f : C}$$

Immédiatement $\varepsilon(\Sigma)(f) = \varepsilon(C)$ et d'après le lemme 59 on a $\vdash_{\varepsilon(\Sigma)} \varepsilon(C) : *$.

Si la dernière règle est (WeakΓ) c'est immédiat : $\varepsilon(\Gamma, x{:}C) = \varepsilon(\Gamma), x{:}\varepsilon(C)$ et d'après le lemme 59 on a $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} \varepsilon(C) : *$.

Si la dernière règle est (Conv) , il est facile de voir que si $C =_{\beta\delta} B$, alors $\varepsilon(C) =_{\beta\delta} \varepsilon(B)$, et d'après le lemme 59 on a $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} \varepsilon(C) : *$. On applique donc immédiatement l'hypothèse de récurrence et une étape de conversion dans $\rho_{\rightarrow}$.

Si la dernière règle est

$$(\textsc{Struct})\ \frac{\Gamma \vdash_\Sigma A : C \qquad \Gamma \vdash_\Sigma B : C}{\Gamma \vdash_\Sigma A \wr B : C}$$

Par hypothèse de récurrence $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} |A| : \varepsilon(C)$ et $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} |B| : \varepsilon(C)$, d'où $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} |A \wr B| : \varepsilon(C)$

Si la dernière règle est

$$(\textsc{Abs})\ \frac{\Gamma, \Delta \vdash_\Sigma A : C \qquad \Gamma \vdash_\Sigma \Pi P{:}\Delta.C : s}{\Gamma \vdash_\Sigma \lambda P{:}\Delta.A : \Pi P{:}\Delta.C}$$

Par hypothèse de récurrence $\varepsilon(\Gamma, \Delta) \vdash_{\varepsilon(\Sigma)} |A| : \varepsilon(C)$ et d'après le lemme 59 on a $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} \varepsilon(\Pi P{:}\Delta.C) : *$.

On a par ailleurs $|\lambda P{:}\Delta.A| = \lambda P{:}\varepsilon(\Delta).\big((\lambda y_1{:}0 \ldots \lambda y_n{:}0.|B|)\,|C_1| \ldots |C_n|\big)$, où les n arguments $|C_i|$ sont absorbés par les n abstractions sur les y_i, et n'ont donc aucune incidence sur le typage (par hypothèse de récurrence on a $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} |C_i| : \varepsilon(s) = 0$ qui est bien le type attendu pour y_i).

On a donc effectivement $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} |\lambda P{:}\Delta.A| : \Pi P{:}\varepsilon(\Delta).\varepsilon(C) = \varepsilon(\Pi P{:}\Delta.C)$.

Si la dernière règle est

$$(\textsc{Appl})\ \frac{\Gamma \vdash_\Sigma A : \Pi P{:}\Delta.C \qquad \Gamma \vdash_\Sigma [P \ll_\Delta B]C : s}{\Gamma \vdash_\Sigma A\,B : [P \ll_\Delta B]C}$$

Par hypothèse de récurrence $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} |A| : \varepsilon(\Pi P{:}\Delta.C) = \Pi P{:}\varepsilon(\Delta).\varepsilon(C)$ et d'après le lemme 59 on a $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} \varepsilon([P \ll_\Delta B]C) : *$, d'où $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} |A\,B| : [P \ll_{\varepsilon(\Delta)} |B|]\varepsilon(C) = \varepsilon([P \ll_\Delta B]C)$.

Si la dernière règle est

$$(\textsc{Prod})\ \frac{\Gamma,\Delta \vdash_\Sigma P : A \qquad \Gamma,\Delta \vdash_\Sigma A : s_1 \qquad \Gamma,\Delta \vdash_\Sigma C : s_2}{\Gamma \vdash_\Sigma \Pi P{:}\Delta.C : s_2}$$

Par hypothèse de récurrence $\varepsilon(\Gamma,\Delta) \vdash_{\varepsilon(\Sigma)} |C| : \varepsilon(s_2) = 0$ et de même pour tout $(x_i{:}C_i) \in \Delta$ on a $\varepsilon(\Gamma,\Delta) \vdash_{\varepsilon(\Sigma)} |C_i| : \varepsilon(s) = 0$. Vu le type de $\pi_{P:\Delta}$, on a donc $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} |\Pi P{:}\Delta.C| : 0 = \varepsilon(s_3)$.

Si la dernière règle est

$$(\textsc{Match})\ \frac{\Gamma,\Delta \vdash_\Sigma P : A \quad \Gamma \vdash_\Sigma B : A \quad \Gamma,\Delta \vdash_\Sigma A : s_1 \quad \Gamma,\Delta \vdash_\Sigma C : s_2}{\Gamma \vdash_\Sigma [P \ll_\Delta B]C : s_2}$$

Notons que $|P| = P$. Par hypothèse de récurrence on a donc $\varepsilon(\Gamma,\Delta) \vdash_{\varepsilon(\Sigma)} P : \varepsilon(A)$ et $\varepsilon(\Gamma,\Delta) \vdash_{\varepsilon(\Sigma)} |B| : \varepsilon(A)$ et $\varepsilon(\Gamma,\Delta) \vdash_{\varepsilon(\Sigma)} |C| : \varepsilon(s_2) = 0$.
Toujours par hypothèse de récurrence, pour tout $(x_i{:}C_i) \in \Delta$ on a $\varepsilon(\Gamma,\Delta) \vdash_{\varepsilon(\Sigma)} |C_i| : \varepsilon(s) = 0$ qui correspondent bien au type attendu par les y_i.
On a donc bien $\varepsilon(\Gamma) \vdash_{\varepsilon(\Sigma)} |[P \ll_\Delta B]C| : 0 = \varepsilon(s_3)$.

□

Lemme 61
Si $A \mapsto_{\rho\delta} A'$, alors $|A| \mapsto\!\!\!\!\to_{\rho\delta} |A'|$ en au moins un pas.

Démonstration : Immédiat : pour tout terme A, quel que soit le sous-terme B de A susceptible de se réduire (y compris si B apparaît dans un type), le terme $|B|$ apparaît dans $|A|$. Par ailleurs, les ρ (resp. δ)-radicaux sont traduits par des ρ (resp. δ)-radicaux. □

Une première formalisation dans ρP

En guise de transition vers le chapitre suivant, montrons une première utilisation de ρP en tant que «Logical Framework avec motifs», légitimée par la normalisation forte de ce système. L'exemple suivant est dû à L. Liquori [Liq05].

Dans [AHMP92], on trouve une formalisation en LF de λ_v, le λ-calcul avec appel par valeurs de G. Plotkin [Plo75]. L'idée centrale de cette formalisation est d'utiliser deux types atomiques : o (qui désigne tous les λ-termes) et v (qui ne désigne que les valeurs, c.-à-d. les variables et les λ-abstractions). Pour exprimer que toute valeur est un terme, on utilise un opérateur de coercion «!» et on a (entre autres) les constantes suivantes dans la signature :

$$\begin{array}{rcl} ! & : & v \to o \\ \lambda_v & : & (v \to o) \to v \\ @ & : & o \to o \to o \\ = & : & o \to o \to * \\ \beta_v & : & \Pi x{:}(v \to o).\ \Pi y{:}v.\ \big(@(!(\lambda_v\, x), !y) = x\, y\big) \end{array}$$

Pour représenter les λ-termes, on prend les variables dans v et on a donc

$$|x| \ =\ !x \qquad |\lambda x.M| \ =\ !\lambda_v(\lambda x{:}v.|M|) \qquad |M\,N| \ =\ @(|M|,|N|)$$

Dans ρP , on peut utiliser le filtrage pour remplacer cette opération de coercion. On n'a donc plus qu'un type atomique o, et la constante ! devient un simple marqueur syntaxique des valeurs. L'appel par valeurs devient donc en fait une ρ-réduction qui filtre sur un motif de la forme $!x$. La signature est modifiée comme suit :

$$\begin{array}{rcl} ! & : & o \rightarrow o \\ \lambda_v & : & (\Pi !z{:}(z{:}o).o) \rightarrow o \\ @ & : & o \rightarrow o \rightarrow o \\ = & : & o \rightarrow o \rightarrow * \\ \beta_v & : & \Pi x{:}(\Pi !z{:}(z{:}o).o).\ \Pi y{:}o.\ \big(@(!(\lambda_v\, x), y) = x\, y\big) \end{array}$$

La représentation des termes devient :

$$|x| \ = \ !x \qquad\qquad |\lambda x.M| \ = \ !\lambda_v(\lambda !x{:}(x{:}o).|M|) \qquad\qquad |M\,N| \ = \ @(|M|, |N|)$$

Ainsi, un radical $(\lambda x.M)\,N$ s'écrit $@(!(\lambda_v\, \lambda !z.|M|), |N|)$. D'après l'axiome β_v, ce terme est égal à $(\lambda !z.|M|)\,|N|$. Enfin, si N est une valeur alors $|N|$ s'écrit $!N'$ où N' est une variable ou une λ-abstraction, et donc $(\lambda !z.|M|)\,(!N') =_{\rho\sigma\delta} |M|[z := N']$.

On observe donc un glissement de la formalisation : au lieu de se faire totalement au niveau des types (donc sémantiquement), une partie passe dans la syntaxe. Il semble donc possible en général, lorsqu'on représente un système formel dans ρP, de mieux séparer les concepts qui donnent vraiment du sens au système de ceux qui sont purement formels et peuvent être relégués dans la syntaxe.

6

Calcul de réécriture et correspondance de Curry-Howard-de Bruijn

Dans ce chapitre, nous traitons certains aspects de l'utilisation d'un calcul avec motifs tel que le ρ-calcul en tant que langage de termes de preuve. En particulier, nous aborderons cette question avec l'objectif de construire un assistant à la démonstration. Nous verrons que nous pouvons établir des liens avec la déduction modulo, ce à quoi on pouvait s'attendre puisqu'elle consiste à ajouter une forme de réécriture dans les systèmes de déduction.

Dans la première section, nous décrirons comment utiliser les P^2TS pour représenter les démonstrations en déduction naturelle modulo. Dans la seconde section nous étudierons une variante de la déduction modulo pour laquelle nous définirons des termes de preuve avec motifs, ce qui nous donne un nouveau système de types qui reste à étudier.

6.1 Termes de preuve riches pour la déduction modulo

Nous avons vu que la déduction modulo (voir section 1.3) est basée sur le principe de Poincaré, consistant à occulter la partie calculatoire des démonstrations. Cela se traduit d'ailleurs dans les λ-termes de preuve pour la déduction modulo, qui sont très concis. Cependant, dans le cadre des assistants à la démonstration, il est souhaitable de vérifier d'autres propriétés, notamment le *critère de de Bruijn*, selon lequel le noyau qui vérifie les démonstrations doit être le plus réduit possible (afin de pouvoir en montrer la correction à la main). Or, il est à craindre qu'un assistant basé sur la déduction modulo doive intégrer une routine de réécriture, qui peut être assez fastidieuse à vérifier.

Nous allons donc assouplir le principe de Poincaré au sens où, même si la démonstration peut être effectuée et présentée sans tenir compte des étapes de calcul, nos termes de preuve conserveront une trace des pas de réécriture effectués, ce qui permettra de vérifier les démonstrations en ajoutant uniquement un algorithme de filtrage à un algorithme de vérification de type usuel.

Nous commencerons par montrer comment sont construits les termes de preuve, puis nous étudierons différents aspects techniques. Nous préciserons notamment la puissance de filtrage nécessaire, et nous expliciterons la correspondance entre les termes et les démonstrations. Enfin, nous discuterons de l'utilisation concrète de ce langage de termes de preuve.

6.1.1 Représentation des conversions par réécriture dans les termes P^2TS

Pour cette section, nous utiliserons une formulation légèrement différente (mais logiquement équivalente) de la déduction modulo. Au lieu de considérer les propositions modulo la congruence $\cong$ et donc d'avoir des conditions d'application du genre $A \cong B$ sur toutes les règles, on considère les règles standard de la logique du premier ordre (voir figures 1.4, 1.5 et 1.6), auxquelles on ajoute une règle de conversion explicite (Cong).

$$(\text{Cong})\ \frac{\Gamma \vdash_{\cong} \phi}{\Gamma \vdash_{\cong} \psi} \quad \text{si } \phi \cong \psi$$

Ainsi on peut garder une trace précise des étapes de calcul. La plupart du temps, nous considérerons même une application de la règle (Cong) pour chaque pas de réécriture effectué dans les propositions. Pour simplifier la présentation, dans toute cette section nous nous placerons dans la logique minimale (les seuls connecteurs sont $\Rightarrow$ et $\forall$), mais ces travaux s'étendent facilement aux autres connecteurs.

Voyons deux exemples, dans lesquels nous utilisons un prédicat d'égalité $=$ binaire.

Exemple 34 (Une démonstration avec conversion sur les termes)
Considérons la théorie des groupes, dans laquelle les propriétés de l'élément neutre sont données par la congruence, avec notamment $e * x \cong x$. On peut alors par exemple démontrer que e est le seul élément neutre, sans utiliser aucun axiome de l'égalité :

$$\begin{array}{rl} (Ax) & \dfrac{}{\forall y.(y * e' = y) \vdash_{\cong} \forall y.(y * e' = y)} \\ (\forall E) & \dfrac{\forall y.(y * e' = y) \vdash_{\cong} \forall y.(y * e' = y)}{\forall y.(y * e' = y) \vdash_{\cong} e * e' = e} \\ (\text{Cong}) & \dfrac{\forall y.(y * e' = y) \vdash_{\cong} e * e' = e}{\forall y.(y * e' = y) \vdash_{\cong} e' = e} \quad \text{avec } e * e' \cong e' \\ (\Rightarrow I) & \dfrac{\forall y.(y * e' = y) \vdash_{\cong} e' = e}{\vdash_{\cong} \forall y.(y * e' = y) \Rightarrow e' = e} \end{array}$$

Exemple 35 (Une démonstration avec conversion sur les propositions)
On peut représenter l'ensemble des entiers relatifs $\mathbb{Z}$ par trois constructeurs 0, p et s (resp. pour le prédécesseur et le successeur), en quotientant l'algèbre des termes par la congruence

$$s(p(z)) \cong z \cong p(s(z))$$

Il est alors naturel d'étendre la congruence sur les propositions de la façon suivante :

$$s(x) = y \quad \cong \quad x = p(y) \qquad et \qquad p(x) = y \quad \cong \quad x = s(y)$$

Dans ce cadre on peut démontrer l'injectivité du successeur :

$$\begin{array}{rl} (Ax) & \dfrac{}{s(a) = s(b) \vdash_{\cong} s(a) = s(b)} \\ (\text{Cong}) & \dfrac{s(a) = s(b) \vdash_{\cong} s(a) = s(b)}{s(a) = s(b) \vdash_{\cong} a = p(s(b))} \quad \text{avec } s(a){=}s(b) \ \cong \ a{=}p(s(b)) \\ (\text{Cong}) & \dfrac{s(a) = s(b) \vdash_{\cong} a = p(s(b))}{s(a) = s(b) \vdash_{\cong} a = b} \quad \text{avec } p(s(b)) \cong b \\ (\Rightarrow I) & \dfrac{s(a) = s(b) \vdash_{\cong} a = b}{\vdash_{\cong} s(a) = s(b) \Rightarrow a = b} \end{array}$$

Notons que chacune des deux congruences utilisées ici peuvent être définies par un système de réécriture, respectivement

$$e * x \to x$$

et

$$\begin{cases} s(p(z)) \to z \\ p(s(z)) \to z \\ s(x) = y \to x = p(y) \\ p(x) = y \to x = s(y) \end{cases}$$

Les deux congruences sont décidables. Pour la première, c'est immédiat dans la mesure où le TRS associé est terminant et confluent ; pour la seconde il faut passer par une technique de complétion du TRS [KB70] afin d'avoir la confluence, ce qui est déjà plus laborieux.

Intéressons-nous maintenant à la représentation linéaire des démonstrations. Dans le langage de termes de preuve de Dowek et Werner [DW03] (voir section 1.3.2), les deux démonstrations précédentes s'écrivent respectivement $\lambda\alpha.(\alpha e)$ et $\lambda\alpha.\alpha$. Ces termes de preuve constituent des *témoins*, au sens où ils contiennent l'information nécessaire à la reconstruction de la démonstration. En revanche, ils sont trop abstraits pour réellement *représenter* les démonstrations : il est difficile de se convaincre que $\lambda\alpha.\alpha$ est une démonstration de $s(x) = s(y) \Rightarrow x = y$.

Nous allons donc utiliser les P^2TS pour concilier les deux aspects des démonstrations en déduction modulo. Les règles de déduction seront représentées par des termes issus du λ-calcul, et les règles de conversion seront représentés par des ρ-termes dans lesquels les règles de réécriture utilisées apparaissent explicitement. Ces derniers termes rappellent largement les démonstrations typiques du Logical Framework, dans la mesure où nous utiliserons une famille de constantes qui permettent de rendre compte des règles de conversion.

Le P^2TS dans lequel nous nous plaçons est une variante de ρC, et les travaux présentés ici reposent donc sur la conjecture de normalisation forte pour ce système. Néanmoins, notre utilisation des règles de produit $(\Box, *)$ et $(\Box, \Box)$ sera assez restreinte, et la réduction présentée au chapitre précédent s'adapterait vraisemblablement plus facilement au langage de termes considéré ici qu'au système ρC dans son intégralité. Par ailleurs, comme expliqué dans le prochain paragraphe, nous distinguerons deux nuances de la sorte $*$, ce qui est assez commun lorsqu'on représente la logique du premier ordre en théorie des types.

Une variante du système de types

Le problème qui se pose lors de l'utilisation de ρP (ou λP d'ailleurs) pour représenter la logique des prédicats est qu'un terme B de type $*$ peut représenter soit une proposition, soit un ensemble. Pour lever cette ambiguïté, l'approche proposée par S. Berardi [Ber88], et implantée dans Coq, consiste à utiliser trois sortes $*^s, *^p, \Box$, où $*^s$ et $*^p$ correspondent respectivement aux ensembles et aux propositions (dans Coq elles sont nommées respectivement Set, Prop et Type). Ainsi, si $\vdash_\Sigma A : B : *^s$ alors A est un élément de l'ensemble B ; si $\vdash_\Sigma A : B : *^p$ alors A est une démonstration de la proposition B. La hiérarchie des sortes est donnée par les axiomes $\vdash *^s : \Box$ et $\vdash *^p : \Box$, et les règles de produit du système simplement typé sont

$$\left\{(*^s, *^s)\ ,\ (*^s, *^p)\ ,\ (*^p, *^p)\right\}$$

Enfin, les types dépendant de termes utilisent la règle $(*^s, \Box)$ et les termes dépendant de types utilisent la règle $(\Box, *^p)$.

Il est facile de voir que l'habitabilité et la normalisation sont équivalentes dans ces systèmes de types et dans les systèmes correspondants du ρ-cube, grâce à la fonction d'effacement $*^s \mapsto *$ et $*^p \mapsto *$. Diverses variantes sur ce principe ont été étudiées par H. Geuvers [Geu93].

Pour faciliter la lecture des termes de preuve, nous utiliserons les conventions de notation suivantes :

- ϕ, ψ, $\chi \in \Phi$ représentent des propositions, c.-à-d. que $\vdash \phi : *^p$;
- μ, $\nu \in \mathcal{S}et$ représentent des ensembles, c.-à-d. que $\vdash \mu : *^s$;
- les variables $\alpha, \beta \in \mathcal{X}^p$ et les termes $\pi, \pi' \in \mathcal{T}_\rho$ représentent des démonstrations, c.-à-d. que $\vdash \alpha : \phi : *^p$;
- les variables $x, y \in \mathcal{X}^s$ et les termes $t, s \in \mathcal{T}$ représentent des termes et des fonctions algébriques, c.-à-d. que $\vdash x : \mu : *^s$.

Voyons maintenant comment représenter les termes et les propositions de la logique du premier ordre dans ces systèmes.

Définition 43 (Représentation de la logique du premier ordre en ρ-calcul)
On considère une signature multisortée, dont les sortes ne doivent pas être confondues avec $*^s$, $*^p$ et $\Box$. La signature que nous utiliserons dans ρC est définie comme suit.

- *Pour toute sorte de la signature on prend un type atomique μ tel que $\vdash \mu : *^s$.*
- *Pour toute fonction n-aire $f : \mu_1, \ldots, \mu_n \mapsto \mu$ on prend une constante f telle que $\vdash f : \Pi x_1{:}\mu_1 \ldots \Pi x_n{:}\mu_n.\mu : *^s$.*
- *Pour tout prédicat n-aire p sur $\mu_1 \times \ldots \times \mu_n$ on prend une constante p telle que $\vdash p : \Pi x_1{:}\mu_1 \ldots \Pi x_n{:}\mu_n.*^p : \Box$.*

Par ailleurs, on utilisera un contexte contenant les déclarations $\vdash x : \mu$ pour toute variable x de sorte μ considérée en logique du premier ordre.

Les termes et les propositions sont représentés ainsi :

$$
\begin{array}{rcll}
[\![x]\!] & \triangleq & x & \\
[\![f(t_1, \ldots, t_n)]\!] & \triangleq & f\,[\![t_1]\!] \ldots [\![t_n]\!] & \\
[\![p(t_1, \ldots, t_n)]\!] & \triangleq & p\,[\![t_1]\!] \ldots [\![t_n]\!] & \\
[\![\phi \Rightarrow \psi]\!] & \triangleq & \Pi\alpha{:}[\![\phi]\!].[\![\psi]\!] & \text{où } \alpha \text{ est une variable fraîche} \\
[\![\forall x.\phi]\!] & \triangleq & \Pi x{:}\mu.[\![\phi]\!] & \text{où } \mu \text{ est la sorte de } x
\end{array}
$$

On voit aisément que pour tout terme t de sorte μ on a $\vdash [\![t]\!] : \mu : *^s$ et que pour toute proposition ϕ on a $\vdash [\![\phi]\!] : *^p$.

Congruence sur les propositions

Pour rendre compte des conversions sur les propositions, nous utiliserons une constante dédiée Rew^p qui prend en argument une proposition ϕ, un terme de preuve de cette proposition, une règle de réécriture R et qui applique R à ϕ. Comme le motif (c.-à-d. le membre gauche) de la règle R apparaît dans son type, il nous faut en fait définir une famille de constantes indexée par les membres gauches des règles de réécriture :

$$\mathsf{Rew}^p_l : \Pi\phi{:}\,*^p \;.\; \Pi R{:}(\Pi(l{:}*^p).*^p) \;.\; \Pi\alpha{:}\phi \;.\; R\phi$$

Alors, pour une proposition ϕ dont une démonstration est π, la proposition ϕ' obtenue en réécrivant ϕ selon la règle de réécriture $l \to r$ a pour terme de preuve $\mathsf{Rew}^p_l\ \phi\ (\lambda l.r)\ \pi$. Et le type de ce terme est $[l \ll \phi]r$, qui est bien convertible à ϕ'.

Les termes de preuve obtenus ici seront donc « hybrides », au sens où :

- comme dans [DW03], les règles de déduction de la logique du premier ordre seront représentées par des constructions du langage de termes de preuve, ce qui est typique de la correspondance de Curry-Howard-de Bruijn ;

– les règles de conversion sont représentées par l'utilisation de certaines constantes spécifiques dans la signature, ce qui est typique d'un Logical Framework.

La famille de constantes Rew_l^p dépend directement du système de réécriture $\mathcal{R}$ qui définit la congruence $\cong$. Si ce TRS est fini, alors la famille est finie également.

On voit d'où vient l'utilisation des différentes règles de produit : la règle $(*^s, \Box)$ est indispensable pour typer les symboles de prédicats ; $(\Box, \Box)$ est nécessaire à la correction du type d'une règle R qui réécrit des propositions ; enfin $(\Box, *^p)$ permet d'abstraire sur ϕ et R.

Congruence sur les termes

Nous venons de voir comment représenter l'application d'une règle de réécriture en tête d'une proposition, et nous verrons dans la prochaine section que cela suffit en fait pour effectuer une réécriture à une position arbitraire d'une formule, grâce à une expansion appropriée de l'inférence logique.

En revanche, quand il s'agit de réécrire un terme algébrique qui apparaît dans une formule (comme argument d'un symbole de prédicat p), nous devons déterminer exactement où ce terme se situe dans la formule. La réécriture sur les termes se fera donc au moyen de constantes dont le type rappelle l'axiome de Leibniz, mais où la démonstration de l'égalité de deux termes est remplacée par une règle de réécriture :

$$\begin{aligned}\mathsf{Rew}_l^s : \quad & \Pi\phi{:}(\Pi(y{:}\mu).*^p)\ .\ \Pi x{:}\mu\ .\ \Pi R{:}(\Pi(l{:}\mu).\mu)\ . \\ & \Pi\alpha{:}\phi\, x\ .\ (\phi(Rx))\end{aligned}$$

Ici ϕ est une proposition dépendant d'un terme, et c'est ce terme que nous allons réécrire. Pour tout terme algébrique t tel que $l \to$ réécrit t en u en tête, pour toute proposition $\phi\, t$ dont une démonstration est π, la proposition $\phi\, u$ a pour terme de preuve $\mathsf{Rew}_l^s\ \phi\ t\ (\lambda l.r)\ \pi$. En effet, le type de ce terme est $\phi\,([l \ll t]r)$, qui est convertible à $\phi\, u$.

Exemple 36

Les démonstrations des exemples 34 et 35 ont pour termes de preuve respectifs

$$\lambda\alpha\,.\,\mathsf{Rew}^s_{e*x}\ \big(\lambda y.(y{=}e)\big)\ (e * e')\ \big(\lambda(e{*}x).x\big)\ (\alpha e)$$

et

$$\lambda\alpha\,.\,\mathsf{Rew}^s_{p(s\,z)}\ \big(\lambda w.(a{=}w)\big)\ p(s\,b)\ \big(\lambda p(s\,z).z\big)\ \Big(\mathsf{Rew}^p_{s\,x=y}\ \big(s\,a{=}s\,b\big)\ \big(\lambda(s\,x{=}y).(x{=}p\,y)\big)\ \alpha\Big)$$

Pour résumer cette section, la définition 44 donne la grammaire des ρ-termes que nous utiliserons. Notons que la distinction entre termes algébriques et termes de preuve est implicitement basée sur le système de types, mais que tous les termes produits par cette grammaire ne sont pas typables.

Définition 44 (ρ-termes de preuve pour la déduction modulo)

Les ρ-termes de preuve pour la déduction modulo sont les termes bien typés de la grammaire suivante

$$\begin{array}{lrcl}
\textit{Termes algébriques} & \mathcal{T} & ::= & \mathcal{K} \mid \mathcal{X}^s \mid \mathcal{T}\,\mathcal{T}' \\
\textit{Propositions} & \Phi & ::= & \mathcal{X}^p \mid \Pi\mathcal{X}^p{:}\Phi.\Phi \mid \Pi\mathcal{X}^s{:}\mathcal{S}et.\Phi \\
\textit{Schémas de propositions} & \Phi_x & ::= & \lambda\mathcal{X}^s{:}\mathcal{S}et.\Phi \\
\textit{Règles de réécriture} & \mathcal{R} & ::= & \lambda\mathcal{T}{:}\mathcal{S}et.\mathcal{T} \mid \lambda\Phi{:}*^p.\Phi \\
\textit{Termes de preuve} & \mathcal{T}_\rho & ::= & \mathcal{X}^p \mid \lambda\mathcal{X}^p{:}\Phi.\mathcal{T}_\rho \mid \mathcal{T}_\rho\,\mathcal{T}_\rho \mid \lambda\mathcal{X}^s{:}\mathcal{S}et.\mathcal{T}_\rho \mid \mathcal{T}_\rho\,\mathcal{T} \\
 & & & \mid \mathsf{Rew}_l^p\ \Phi\ \mathcal{R}\ \mathcal{T}_\rho \mid \mathsf{Rew}_l^s\ \Phi_x\ \mathcal{T}\ \mathcal{R}\ \mathcal{T}_\rho
\end{array}$$

6.1.2 Aspects techniques

Nous allons voir que, pour une représentation complète des démonstrations en déduction modulo, il nous faudra parfois utiliser les règles de réécriture de droite à gauche. Ceci nous mènera à considérer un filtrage légèrement étendu, puisque les membres droits des règles qui réécrivent les propositions ne sont plus nécessairement des propositions atomiques. Nous démontrerons ensuite la correspondance entre termes de preuve et démonstrations, et nous conclurons cette section par une discussion sur l'utilisation de ces termes de preuve.

Inversion des règles

Il est assez facile de voir qu'il est nécessaire d'appliquer les règles de réécriture de gauche à droite et de droite à gauche, car la congruence $\cong$ est définie comme la clôture réflexive, transitive et *symétrique* de $\rightarrow_{\mathcal{R}}$. Si par exemple une proposition ψ est démontrable et la seule règle dont nous disposons est $\phi \rightarrow \psi$, alors ϕ est démontrable puisque $\phi \cong \psi$.

En supposant maintenant que π est un terme de preuve pour ψ, la seule façon vraisemblable de construire un terme de preuve pour ϕ est $\mathsf{Rew}^p_\psi\ \psi\ (\lambda\psi.\phi)\ \pi$, où on trouve la règle inversée $\psi \rightarrow \phi$.

On pourrait croire que le fait de considérer des règles de réécriture dans les deux sens de lecture est une source immédiate de non-terminaison, mais il ne faut pas oublier que, dans les termes de preuve que nous construisons, chaque règle de réécriture apparaît une fois pour *chacune* de ses applications. La décidabilité de la congruence $\cong$ n'est d'ailleurs plus nécessaire, pour la même raison : nous gardons une trace de tous les pas de réécriture qui sont effectués. Le système de réécriture $\mathcal{R}$ peut donc être quasi arbitraire, et ni sa confluence ni sa terminaison ne sont requises.

Voyons maintenant sur un exemple comment les règles inversées permettent d'appliquer une règle de réécriture à l'intérieur d'une formule, bien que le terme de preuve $\mathsf{Rew}^p_l\ \phi\ R\ \pi$ ne permette d'appliquer R qu'en tête de ϕ. Supposons par exemple une inférence comportant la règle

$$(\textsc{Cong})\ \frac{\vdots}{\Gamma \vdash_{\cong} \phi_1 \Rightarrow \phi_2}{\Gamma \vdash_{\cong} \psi_1 \Rightarrow \phi_2}$$

où la proposition ϕ_1 se réécrit en ψ_1 en tête. Il est alors possible de développer cette démonstration en une démonstration équivalente mais telle que la règle de conversion a lieu sur ϕ_1 uniquement :

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, \psi_1 \vdash_{\cong} \phi_1 \Rightarrow \phi_2}\quad \dfrac{(Ax)\ \overline{\Gamma, \psi_1 \vdash_{\cong} \psi_1}}{(\textsc{Cong})\ \Gamma, \psi_1 \vdash_{\cong} \phi_1}}{(\Rightarrow E)\ \Gamma, \psi_1 \vdash_{\cong} \phi_2}}{(\Rightarrow I)\ \Gamma \vdash_{\cong} \psi_1 \Rightarrow \phi_2}$$

Dans cette démonstration développée, la règle (Cong) a pour prémisse ψ_1 et pour conclusion ϕ_1 : la règle de réécriture $\phi_1 \rightarrow \psi_1$ doit être appliquée dans le sens inverse, alors qu'elle était appliquée dans le sens direct lorsque la règle de conversion portait sur $\phi_1 \Rightarrow \phi_2$.

Un point délicat dans l'inversion des règles de réécriture est l'existence de règles *irrégulières*, c.-à-d. dont le membre gauche comporte des variables qui n'apparaissent pas dans le membre droit. L'application d'une telle règle de droite à gauche crée donc des variables libres, mais ce

mécanisme s'avère assez adapté pour représenter des démonstrations génériques dont on peut lier ou instancier les variables libres par la suite.

Par exemple, considérons un prédicat binaire $last$ dont on veut qu'il soit vérifié si son second argument est le dernier élément de la liste qui est son premier argument. Une façon de le définir est d'utiliser la congruence définie par la règle irrégulière

$$last(y{::}(z{::}zz), x) \;\rightarrow\; last(z{::}zz, x)$$

et de se donner un axiome $LastAx : \forall x.last(x{::}nil, x)$.

Pour faciliter la lecture, nous écrirons l pour le membre gauche $last(y{::}(z{::}zz), x)$ et r pour le membre droit $last(z{::}zz, x)$. La représentation de la règle $\lambda l.r$ est bien un terme clos, alors que dans la règle inversée $\lambda r.l$ la variable y est libre. Voyons quelques termes de preuve constructibles à partir de cette règle de réécriture et de cet axiome.

- $LastAx\ b$ (qui correspond à une règle $(\forall E)$) est un terme de preuve clos pour $last(b{::}nil, b)$.
- $\mathsf{Rew}^p_r\ last(b{::}nil, b)\ (\lambda r.l)\ (LastAx\ b)$, obtenu par un pas de réécriture, est un terme de preuve pour $last(y{::}(b{::}nil), b)$ où y est une variable libre (rappelons que y apparaît dans l). On peut voir ce terme comme une preuve générique sur tous les y possibles.
- $\lambda y.\mathsf{Rew}^p_r\ last(b{::}nil, b)\ (\lambda r.l)\ (LastAx\ b)$ est un terme de preuve clos pour la proposition $\forall y.last(y{::}(b{::}nil), b)$, obtenu par une simple abstraction sur y (qui correspond à une règle $(\forall I)$).
- $\big(\lambda y.\mathsf{Rew}^p_r\ last(b{::}nil, b)\ (\lambda r.l)\ (LastAx\ b)\big)\ a$ est un terme de preuve clos pour la proposition $last(a{::}(b{::}nil), b)$. Par réduction du radical externe, la règle de réécriture $\lambda r.l$ devient $\lambda r.(l[y := a])$, qui en est une instance close.

De même il serait possible de construire des termes de preuve pour un nombre arbitraire d'éléments dans la liste, soit de façon générique comme dans $\forall x_1 \ldots \forall x_n.last(x_1{::}(\ldots(x_n{::}(d{::}nil))), d)$, soit en prenant des instances closes : $last(a_1{::}(\ldots(a_n{::}(d{::}nil))), d)$.

Filtrage modulo α-conversion

Voyons maintenant le genre de problèmes de filtrage qui se posent dans les termes de preuve construits. Nous pouvons déjà remarquer, vue la grammaire donnée dans la définition 44, que la partie déductive de nos termes de preuve n'utilise que des variables comme motifs. Les problèmes de filtrage non triviaux sont donc tous dûs aux règles de réécriture que les constantes Rew^p et Rew^s prennent en argument.

Le filtrage (et donc la réécriture) sur les termes algébriques ne pose pas de problème : il s'agit d'un filtrage syntaxique ordinaire.

Le filtrage sur les propositions atomiques est également syntaxique, puisqu'on considère un symbole de prédicat comme une simple constante. Par contre, comme nous appliquons également des règles de réécriture de droite à gauche, il faudra également pouvoir filtrer sur des propositions non atomiques.

Nous avons représenté l'implication comme un type produit $\Pi\alpha{:}\phi.\psi$ où α n'apparaît pas dans ψ, et la quantification universelle comme $\Pi x{:}\mu.\phi$, où x peut apparaître dans ϕ. Il est donc visible que la variable α n'est pas significative dans l'implication, alors que dans la quantification on ne peut renommer la variable x qu'à condition de renommer toutes ses occurrences libres dans ϕ. On constate du même coup que les seules variables libres d'un motif de proposition sont des variables libres dans des prédicats, qui ne peuvent donc être que des variables algébriques.

Exemple 37 (Motifs de propositions non atomiques)

- $\Pi\beta{:}(x{\neq}0).(y{=}0)$ *représente* $x{\neq}0 \Rightarrow y{=}0$. *Il subsume* $\Pi\alpha{:}(t \neq 0).(t' = 0)$ *pour toute variable* α *et tous termes* t *et* t'.
- $\Pi y{:}int.(x{*}y = 0)$ *représente* $\forall y.(x{*}y = 0)$. *Il subsume* $\Pi z{:}int.(t{*}z = 0)$ *pour toute variable* z *et tout terme* t.

Le filtrage devra donc se faire modulo α-conversion des variables liées.

Définition 45 (Variables libres, liées d'un motif de proposition)
Les variables libres et liées d'un motif de proposition sont définies par

$$\begin{array}{rclcrcl} \mathcal{FV}(p\,t_1\,\ldots\,t_n) & \triangleq & \bigcup \mathcal{V}ar(t_i) & \qquad & \mathcal{BV}(p\,t_1\,\ldots\,t_n)) & \triangleq & \emptyset \\ \mathcal{FV}(\Pi\alpha{:}\phi.\psi) & \triangleq & \mathcal{FV}(\phi) \cup \mathcal{FV}(\psi) & & \mathcal{BV}(\Pi\alpha{:}\phi.\psi) & \triangleq & \mathcal{BV}(\phi) \cup \mathcal{BV}(\psi) \cup \{\alpha\} \\ \mathcal{FV}(\Pi x{:}\mu.\phi) & \triangleq & \mathcal{FV}(\phi) \setminus \{x\} & & \mathcal{BV}(\Pi x{:}\mu.\phi) & \triangleq & \mathcal{BV}(\phi) \cup \{x\} \end{array}$$

Rappelons qu'une solution du problème de filtrage $P \ll^?_\alpha A$ est une substitution θ telle que :
- $\mathcal{D}om(\theta) = \mathcal{FV}(P)$;
- $P\theta =_\alpha A$.

Un algorithme de filtrage modulo α-conversion est obtenu à partir de l'algorithme syntaxique en renommant systématiquement les variables liées. Nous pouvons même nous dispenser de certains de ces renommages en exploitant le fait que les variables α ne sont pas significatives.

Proposition 3 (Correction et complétude de l'algorithme de filtrage)
Les règles de transformation des problèmes de filtrage de la figure 6.1 vérifient les propriétés suivantes.

1. *Tout problème de filtrage* $P \ll^? M$ *a une forme normale unique selon ces règles.*
2. *Si cette forme normale est vide, alors* P *et* M *sont identiques à* α*-conversion près.*
3. *Si cette forme normale est de la forme* $\{x_i \ll^? t_i\}_{i\in I}$ *avec* $I \neq \emptyset$, *alors* $\theta = [x_i := t_i]$ *est l'unique (modulo* α*) substitution telle que* $P\theta =_\alpha M$.
4. *Sinon,* P *ne subsume pas* M.

$$\begin{array}{rcll} \{f(t_1,\ldots,t_n) \ll^? g(s_1,\ldots,s_n)\} \cup S & \Longrightarrow & \{t_1 \ll^? s_1,\ldots,t_n \ll^? s_n\} \cup S & \text{si } f = g \\ \{p(t_1,\ldots,t_n) \ll^? q(s_1,\ldots,s_n)\} \cup S & \Longrightarrow & \{t_1 \ll^? s_1,\ldots,t_n \ll^? s_n\} \cup S & \text{si } p = q \\ \{\Pi\alpha{:}P_1.P_2 \ll^? \Pi\beta{:}Q_1.Q_2\} \cup S & \Longrightarrow & \{P_1 \ll^? Q_1\ ,\ P_2 \ll^? Q_2\} \cup S & \\ \{\Pi x{:}\tau.P \ll^? \Pi y{:}\sigma.Q\} \cup S & \Longrightarrow & \{P[x := z] \ll^? Q[y := z]\} \cup S & \text{pour un } z \text{ frais} \\ \{x \ll^? t\ ,\ x \ll^? t'\} \cup S & \Longrightarrow & \bot & \text{si } t \not\equiv t' \end{array}$$

FIG. 6.1 – Filtrage sur les propositions

La correspondance

Enfin, précisons à quel point il existe une correspondance de Curry-Howard-de Bruijn entre nos termes et la déduction modulo. Nous établirons une correspondance entre propositions et types, ainsi qu'entre démonstrations et termes ; nous verrons ensuite qu'une partie seulement de l'élimination des coupures se traduit par des réductions dans nos termes.

Commençons par définir deux fonctions pour traduire les termes (resp. les formules) de la logique du premier ordre en termes algébriques (resp. types) de ρC, et *vice versa*.

Définition 46 (Correspondance entre propositions et types)

1. *La traduction $[\![\ ]\!] : PRED \to \rho C$ est détaillée dans la définition 43. On peut l'étendre aux contextes afin d'assigner une variable à chaque hypothèse :*

$$[\![\Gamma, \phi]\!] \triangleq [\![\Gamma]\!], x_\phi : [\![\phi]\!]$$

2. *La traduction réciproque $|\ | : \rho C \to PRED$ consiste essentiellement à effacer de l'information :*

$$\begin{array}{rcl} |x| & \triangleq & x \\ |f\, t_1 \ldots t_n| & \triangleq & f(|t_1|, \ldots, |t_n|) \\ |p\, t_1 \ldots t_n| & \triangleq & p(|t_1|, \ldots, |t_n|) \\ |\Pi\alpha{:}\phi.\psi| & \triangleq & |\phi| \Rightarrow |\psi| \\ |\Pi x{:}\mu.\phi| & \triangleq & \forall x.|\phi| \\ |\Gamma, \alpha{:}\phi| & \triangleq & |\Gamma|, |\phi| \\ |\Gamma, x{:}\mu| & \triangleq & |\Gamma| \end{array}$$

Théorème 62 (Correspondance entre démonstrations et termes)

1. *Tout ρ-type (sans contrainte de filtrage irrésoluble) habité par un ρ-terme de preuve (déf. 44) représente une proposition démontrable :*

$$\Gamma \vdash \pi : \phi \ \Rightarrow\ |\Gamma| \vdash_{\cong} |\phi|$$

La congruence $\cong$ est définie par l'ensemble des règles de réécriture apparaissant dans π.
La démonstration de $|\Gamma| \vdash_{\cong} |\phi|$ est obtenue par une traduction directe de π.

2. *Toute proposition démontrable est représentée par un type habité par un terme de preuve :*

$$\Gamma \vdash_{\cong} \phi \ \Rightarrow\ \exists \pi,\ \ [\![\Gamma]\!] \vdash \pi : [\![\phi]\!]$$

Les seules règles de réécriture apparaissant dans π sont des règles définissant $\cong$.
La démonstration que représente π est une démonstration équivalente à la démonstration originale de $\Gamma \vdash_{\cong} \phi$, obtenue par transformation des règles de conversion.

Comme dans un isomorphisme de Curry-Howard-de Bruijn usuel, un terme de preuve représente donc exactement une démonstration. Par contre, la symétrie n'est pas totale, puisqu'une démonstration doit parfois être transformée en une autre démonstration (équivalente) pour trouver un terme de preuve qui la représente.

Démonstration :

1. Ce sens de la correspondance est assez immédiat : par récurrence sur la structure de π, on efface les termes dans la dérivation de typage de π.

   $\pi \equiv \alpha$: alors $(\alpha{:}\phi) \in \Gamma$, donc $|\phi| \in |\Gamma|$ et immédiatement

$$(Ax)\ \frac{}{|\Gamma| \vdash_{\cong} |\phi|}$$

$\pi \equiv \lambda\alpha{:}\phi_1.\pi'$: alors $\phi =_{\beta\delta} \Pi\alpha{:}\phi_1.\phi_2$ tel que $\Gamma, \alpha{:}\phi_1 \vdash \pi' : \phi_2$. Par hypothèse de récurrence $|\Gamma|, |\phi_1| \vdash_{\cong} |\phi_2|$ d'où

$$(\Rightarrow I)\ \frac{|\Gamma|, |\phi_1| \vdash_{\cong} |\phi_2|}{|\Gamma| \vdash_{\cong} |\phi_1| \Rightarrow |\phi_2|}$$

et en effet $|\Pi\alpha{:}\phi_1.\phi_2| = |\phi_1| \Rightarrow |\phi_2|$.

$\pi \equiv \lambda x.\pi'$: alors $\phi =_{\beta\delta} \Pi x{:}\mu.\phi_1$ tel que $\Gamma, x{:}\mu \vdash \pi' : \phi_1$. Par hypothèse de récurrence $|\Gamma| \vdash_{\cong} |\phi_1|$ où $x \notin \mathcal{FV}(\Gamma)$ d'où

$$(\forall I)\ \frac{|\Gamma| \vdash_{\cong} |\phi_1|}{|\Gamma| \vdash_{\cong} \forall x.|\phi_1|}\ (x \notin \mathcal{FV}(\Gamma))$$

et en effet $|\Pi x{:}\mu.\phi_1| = \forall x.|\phi_1|$.

$\pi \equiv \pi'\,\pi''$: alors il existe ψ tel que $\Gamma \vdash \pi' : \Pi\alpha{:}\psi.\phi$ et $\Gamma \vdash \pi'' : \psi$. Par hypothèse de récurrence $|\Gamma| \vdash_{\cong} |\Pi\alpha{:}\psi.\phi|$ et $|\Gamma| \vdash_{\cong} |\psi|$. Or $|\Pi\alpha{:}\psi.\phi| = |\psi| \Rightarrow |\phi|$, d'où

$$(\Rightarrow E)\ \frac{|\Gamma| \vdash_{\cong} |\psi| \Rightarrow |\phi| \qquad |\Gamma| \vdash_{\cong} |\psi|}{|\Gamma| \vdash_{\cong} |\phi|}$$

$\pi \equiv \pi'\,t$: alors il existe deux types μ et ψ tels que $\Gamma \vdash \pi' : \Pi x{:}\mu.\psi$ et $\Gamma \vdash t : \mu$, avec $\phi =_{\beta\delta} \psi[x := t]$. Par hypothèse de récurrence $|\Gamma| \vdash_{\cong} |\Pi x{:}\mu.\psi|$; or $|\Pi x{:}\mu.\psi| = \forall x.|\psi|$ et $|\psi[x := t]| = |\psi|[x := t]$ d'où

$$(\forall E)\ \frac{|\Gamma| \vdash_{\cong} \forall x.|\psi|}{|\Gamma| \vdash_{\cong} |\psi|[x := t]}$$

$\pi \equiv \mathsf{Rew}^p_l \phi'(\lambda l.r)\pi'$: alors $\phi =_{\beta\delta} [l \ll \phi']r$ tel que $\Gamma \vdash \pi' : \phi'$. Si le problème de filtrage $l \ll^? \phi'$ n'a pas de solution, alors ϕ contient une contrainte de filtrage irrésoluble et ne représente donc pas une proposition. Sinon, $\phi =_{\beta\delta} r\theta_{(l \ll \phi')}$ et par hypothèse de récurrence $|\Gamma| \vdash_{\cong} |\phi'|$; par ailleurs la règle $\lambda l.r$ réécrit ϕ' en ϕ et apparaît dans π donc $\phi \cong \phi'$, d'où

$$(\textsc{Cong})\ \frac{|\Gamma| \vdash_{\cong} |\phi'|}{|\Gamma| \vdash_{\cong} |\phi|} \quad \text{AVEC } \phi \cong \phi'$$

$\pi \equiv \mathsf{Rew}^s_l \phi' t(\lambda l.r)\pi'$: alors $\phi =_{\beta\delta} \phi'([l \ll t]r)$ tel que $\Gamma \vdash \pi' : \phi'(t)$. Si le problème de filtrage $l \ll^? t$ n'a pas de solution, alors ϕ contient une contrainte de filtrage irrésoluble et ne représente donc pas une proposition. Sinon, $\phi =_{\beta\delta} \phi'(r\theta_{(l \ll t)})$ et par hypothèse de récurrence $|\Gamma| \vdash_{\cong} |\phi'(t)|$; par ailleurs la règle $\lambda l.r$ réécrit $\phi'(t)$ en ϕ et apparaît dans π donc $\phi \cong \phi'(t)$, d'où

$$(\textsc{Cong})\ \frac{|\Gamma| \vdash_{\cong} |\phi'(t)|}{|\Gamma| \vdash_{\cong} |\phi|} \quad \text{AVEC } \phi \cong \phi'(t)$$

2. L'autre sens de la correspondance demande plus de travail. Pour construire un terme de preuve, nous devons transformer la démonstration de sorte que les conversions soient appliquées règle par règle et en tête des formules. Procédons en trois étapes.

(a) Pour chaque règle de conversion portant sur une congruence $\phi \cong \psi$, par définition de $\cong$ on a un chemin de la forme

et de longueur finie n. On peut donc remplacer cette règle de conversion par n règles de conversion, chacune correspondant exactement à un pas de réécriture (dans le sens direct ou inverse).

(b) La seconde étape consiste à développer les règles de conversion qui utilisent une règle de réécriture sur les propositions de sorte que la réécriture ait lieu en tête de la proposition. Soit donc une règle

$$(\textsc{Cong})\ \frac{\Gamma \vdash_{\cong} \phi}{\Gamma \vdash_{\cong} \psi} \quad \text{avec } \phi \leftrightarrow_{\mathcal{R}} \psi$$

où $\leftrightarrow_{\mathcal{R}}$ désigne un unique pas de réécriture, dans un sens ou dans l'autre. Distinguons différents cas sur ce pas de réécriture.

- si $\phi \leftrightarrow_{\mathcal{R}} \psi$ en tête de la proposition, la règle de conversion est développée. On passe alors à l'étape (c).
- si $\phi \equiv \phi_1 \Rightarrow \phi_2$ avec $\phi_1 \leftrightarrow_{\mathcal{R}} \psi_1$ et $\psi \equiv \psi_1 \Rightarrow \phi_2$, alors on remplace la règle de conversion par

$$\cfrac{\cfrac{\Gamma, \psi_1 \vdash_{\cong} \phi_1 \Rightarrow \phi_2 \qquad \cfrac{(Ax)\ \overline{\Gamma, \psi_1 \vdash_{\cong} \psi_1}}{(\textsc{Cong})\ \Gamma, \psi_1 \vdash_{\cong} \phi_1}}{(\Rightarrow E)\ \Gamma, \psi_1 \vdash_{\cong} \phi_2}}{(\Rightarrow I)\ \Gamma \vdash_{\cong} \psi_1 \Rightarrow \phi_2}$$

et on développe la conversion sur $\psi_1 \cong \phi_1$.

- si $\phi \equiv \phi_1 \Rightarrow \phi_2$ avec $\phi_2 \leftrightarrow_{\mathcal{R}} \psi_2$ et $\psi \equiv \phi_1 \Rightarrow \psi_2$, alors on remplace la règle de conversion par

$$\cfrac{\cfrac{\cfrac{\Gamma, \phi_1 \vdash_{\cong} \phi_1 \Rightarrow \phi_2 \qquad (Ax)\ \overline{\Gamma, \phi_1 \vdash_{\cong} \phi_1}}{(\Rightarrow E)\ \Gamma, \phi_1 \vdash_{\cong} \phi_2}}{(\textsc{Cong})\ \Gamma, \phi_1 \vdash_{\cong} \psi_2}}{(\Rightarrow I)\ \Gamma \vdash_{\cong} \phi_1 \Rightarrow \psi_2}$$

et on développe la conversion sur $\phi_2 \cong \psi_2$.

- si $\phi \equiv \forall x.\phi_1$ avec $\phi_1 \leftrightarrow_{\mathcal{R}} \psi_1$ et $\psi \equiv \forall x.\psi_1$, alors on remplace la règle de conversion par

$$\cfrac{\cfrac{\cfrac{\Gamma \vdash_{\cong} \forall x.\phi_1}{(\forall E)\ \Gamma \vdash_{\cong} \phi_1}}{(\textsc{Cong})\ \Gamma \vdash_{\cong} \psi_1}}{(\forall I)\ \Gamma \vdash_{\cong} \forall x.\psi_1}$$

et on développe la conversion sur $\phi_1 \cong \psi_1$.

On remarque que, dans les prémisses de la démonstration développée, le contexte est augmenté par une formule (ψ_1 ou ϕ_1) , mais par un simple affaiblissement, les démonstrations des prémisses restent valides.

(c) Enfin, dans la troisième étape, on traduit les règles d'inférence en règles de typage. Cette traduction est l'inverse de celle utilisée pour montrer l'autre sens de la correspondance :
- les axiomes sont traduits par des variables ;
- l'introduction de $\Rightarrow$ est traduite par une λ-abstraction sur une variable α ;
- l'introduction de $\forall$ est traduite par une λ-abstraction sur une variable x ;
- l'élimination de $\Rightarrow$ et $\forall$ sont traduites par une application ;
- la conversion sur les propositions (resp. sur les termes) est traduite en une construction Rew_l^p (resp. Rew_l^s). Pour les termes, c'est toujours possible ; pour les propositions c'est possible puisque les deux étapes précédentes assurent que les réécritures ont lieu pas par pas et en tête des formules.

Remarquons que le développement des conversions peut aussi se faire pour les autres connecteurs de la logique du premier ordre.

□

Malheureusement, la correspondance ne se prolonge pas parfaitement à l'élimination des coupures. En effet, comme expliqué dans [DW03], le principal problème de la règle de conversion explicite est qu'une coupure n'est plus locale : elle est toujours formée d'une règle d'introduction suivie d'une règle d'élimination mais qui peuvent être séparées par un nombre arbitraire de conversions. Par exemple, si $p \cong q$ alors la démonstration

$$\dfrac{\dfrac{\dfrac{\dfrac{}{p \vdash_{\cong} p}(Ax)}{\vdash_{\cong} p \Rightarrow p}(\Rightarrow I)}{\vdash_{\cong} q \Rightarrow p}(\textsc{Cong}) \qquad \dfrac{\vdots}{\vdash_{\cong} q}}{\vdash_{\cong} p}(\Rightarrow E)$$

se réduit à

$$\dfrac{\dfrac{\vdots}{\vdash_{\cong} q}}{\vdash_{\cong} p}(\textsc{Cong})$$

Dans nos termes de preuve, les conversions sont également représentées explicitement. Dans certains cas, ceci n'empêchera pas que les coupures apparaissent comme des radicaux, par exemple la démonstration ci-dessus a pour terme de preuve

$$\Big(\lambda\alpha{:}q.\big((\lambda\beta{:}p.\beta)(\mathsf{Rew}_q^p\ q\ (\lambda q.p)\ \alpha)\big)\Big)\pi$$

où π est un terme de preuve pour q. Ce terme se réduit à $\mathsf{Rew}_q^p\ q\ (\lambda q.p)\ \pi$, qui représente bien la démonstration réduite.

En revanche, si l'introduction et l'élimination d'une coupure sont séparées par trop de conversions, il se peut que le radical n'apparaisse pas dans le terme de preuve. Par exemple une coupure

sur l'implication dont la prémisse principale subit des conversions aura un terme de preuve de la forme $(\mathsf{Rew}^p \ldots \lambda\alpha.\pi \ldots)\,\pi'$, où le radical $(\lambda\alpha.\pi)\,\pi'$ est bloqué par les opérateurs Rew^p. Ces «coupures irréductibles» peuvent être vues comme un symptôme du manque de règles de réduction dans le langage de termes de preuve.

Conjecture 3 (Correspondance entre coupures et radicaux)
En ajoutant des règles de réduction du genre

$$\mathsf{Rew}^p_l \phi\ (\lambda l.r)\ \left(\mathsf{Rew}^p_{l'}\ \phi'\ (\lambda l'.r')\ \pi\right)\ \mapsto\ \pi \quad si\ \ l = r'\ \wedge\ l' = r$$

à toute coupure on peut faire correspondre un (ou plusieurs) radicaux dans le terme de preuve. Cette règle de réduction correspond à l'élimination de ce que D. Prawitz appelle une coupure folding-unfolding *[Pra65]. D'autres règles de réduction peuvent être obtenues en considérant une condition de réduction plus générale que $l = r' \wedge l' = r$.*

Liens avec d'autres formalismes

Un intérêt important des termes de preuve construits ici est qu'ils contiennent assez d'informations pour produire d'autres types de témoins des démonstrations qu'ils représentent. Nous allons notamment voir comment extraire le contenu déductif et le contenu calculatoire d'une démonstration ; enfin nous proposons un fonctionnement pour un assistant à la démonstration basé sur ces termes de preuve.

Il est possible de retrouver la structure logique d'une démonstration à partir de son ρ-terme de preuve : la fonction $\|\ \|$ définie ci-dessous transforme un ρ-terme de preuve en un λ-terme de preuve qui représente (au sens vu à la section 1.3.2) la même démonstration mais avec des conversions implicites. Par conséquent, un ρ-terme de preuve π tel que $\|\pi\|$ est en forme normale représente une démonstration en forme normale.

$$\begin{array}{rcl}
\|\ \| & : & \rho C \longrightarrow \Lambda \\
\|x\| & \triangleq & x \\
\|\lambda x{:}\mu.\pi\| & \triangleq & \lambda x{:}\mu.\|\pi\| \\
\|\pi\, t\| & \triangleq & \|\pi\|\,\|t\| \\
\|\lambda\alpha{:}\phi.\pi\| & \triangleq & \lambda\alpha{:}\phi.\|\pi\| \\
\|\pi\,\pi'\| & \triangleq & \|\pi\|\,\|\pi'\| \\
\|\mathsf{Rew}^p_l\ \phi\ (\lambda l.r)\ \pi\| & \triangleq & \pi \\
\|\mathsf{Rew}^s_l\ \phi\ t\ (\lambda l.r)\ \pi\| & \triangleq & \pi
\end{array}$$

D'un autre côté, on peut également extraire d'une démonstration l'ensemble des règles de réécriture utilisées, par la fonction $\mathcal{R}(\ )$:

$$\begin{array}{rcl}
\mathcal{R}(\cdot) & : & \rho C \longrightarrow \mathrm{TRS} \\
\mathcal{R}(x) & \triangleq & \emptyset \\
\mathcal{R}(\lambda x{:}\mu.\pi) & \triangleq & \mathcal{R}(\pi) \\
\mathcal{R}(\pi t) & \triangleq & \mathcal{R}(\pi) \\
\mathcal{R}(\lambda\alpha{:}\phi.\pi) & \triangleq & \mathcal{R}(\pi) \\
\mathcal{R}(\pi\,\pi') & \triangleq & \mathcal{R}(\pi) \cup \mathcal{R}(\pi') \\
\mathcal{R}(\mathsf{Rew}^p_l\ \phi\ (\lambda l.r)\ \pi) & \triangleq & \mathcal{R}(\pi) \cup \{l \to r\} \\
\mathcal{R}(\mathsf{Rew}^s_l\ \phi\ t\ (\lambda l.r)\ \pi) & \triangleq & \mathcal{R}(\pi) \cup \{l \to r\}
\end{array}$$

Ainsi, si la congruence est connue *a priori*, on peut :

- vérifier qu'aucune règle de réécriture non autorisée n'a été utilisée ;
- éventuellement caractériser le sous-ensemble utilisé du système de réécriture, et voir que la démonstration peut se faire avec une puissance calculatoire moindre.

Un assistant à la démonstration basé sur ce formalisme pourrait être simplement une extension de Coq avec des tactiques permettant de produire les constructions Rew^s et Rew^p. Il serait cependant très fastidieux de faire des démonstrations dans un tel environnement, puisque l'utilisation d'une règle de réécriture demanderait sensiblement le même travail que l'utilisation de l'égalité de Leibniz.

Une méthode plus prometteuse serait d'utiliser un outil externe qui effectue les calculs et produit les termes de preuves correspondants. Une implantation d'un paradigme de démonstration très proche a déjà été réalisée par Q.-H. Nguyen [KKN02], puisque son outil utilise le langage à base de règles ELAN pour vérifier l'égalité de deux termes relativement à un système de réécriture, et produit un terme de preuve correspondant pour Coq. Du point de vue de l'utilisateur, le processus de démonstration interactive reste le même, avec la possibilité supplémentaire d'appeler la tactique ElanRewrite pour mettre en forme normale tous les termes algébrique d'un but. Il serait donc probablement possible d'utiliser l'interface Coq/ELAN pour construire un assistant à la démonstration qui produise des ρ-termes de preuve.

6.2 Vers une déduction naturelle généralisée

Dans la section précédente, nous avons décrit comment utiliser le ρ-calcul comme un langage de termes de preuve pour la déduction modulo. Si l'on imagine un assistant à la démonstration basé sur la déduction modulo, ce langage de preuves facilite la vérification automatique des démonstrations, mais du point de vue de l'utilisateur il ne facilite en rien la démonstration interactive.

Nous allons voir ici les problèmes qui peuvent se poser si on utilise la déduction modulo dans un cadre de démonstration interactive, et nous proposerons un formalisme voisin, mais peut-être plus naturel d'utilisation. Nous en démontrerons la correction et la complétude, et nous proposerons un langage de termes de preuve utilisant une forme de filtrage assez restreint.

6.2.1 Au-delà de la déduction modulo

Rappelons qu'en déduction modulo, les propositions sont considérées modulo une congruence, elle-même le plus souvent définie par un système de réécriture $\mathcal{R}$ qui réécrit :

- des termes en termes ;
- des propositions atomiques en propositions quelconques.

Dans le cadre de la démonstration interactive, le premier type de règles semble souhaitable, afin d'identifier des termes qui ne diffèrent que par des étapes de calcul. Cependant, le second type de règles peut être une source de confusion, car il n'est alors pas toujours évident d'avoir une idée précise de ce que signifie une proposition donnée. Lorsque $\mathcal{R}$ est confluent et terminant, il est bien entendu toujours possible de considérer la forme $\mathcal{R}$-normale des propositions, mais dans ce cas on n'utilise toujours le même représentant dans la classe de congruence de chaque proposition, et on perd donc beaucoup de l'intérêt d'avoir considéré une congruence sur les formules.

Un exemple

Pour illustrer ce propos, considérons un théorème de la théorie des ensembles : $\emptyset \subseteq A$ et étudions sa démonstration en déduction naturelle avec ou sans modulo.

En déduction naturelle, il faut se donner une théorie $\mathcal{T}h$ sous forme d'un ensemble d'axiomes qui contienne au moins :

– la définition du symbole d'inclusion : $\forall X.\forall Y.(X \subseteq Y \Leftrightarrow \forall x(x \in X \Rightarrow x \in Y))$;

– l'axiome de définition de l'ensemble vide : $\forall x.(x \in \emptyset \Rightarrow \bot)$.

Le symbole $\Leftrightarrow$ utilisé ici n'est pas un connecteur primitif de la logique, il n'est qu'une abréviation pour la conjonction des deux implications.

Une démonstration dans PRED de la proposition considérée s'écrit

$$\dfrac{\dfrac{(Ax)\ \dfrac{}{\mathcal{T}h \vdash \forall X.\forall Y.(X \subseteq Y \Leftrightarrow \forall x.(x \in X \Rightarrow x \in Y))}}{(\forall I)(\forall I)\ \mathcal{T}h \vdash \emptyset \subseteq A \Leftrightarrow \forall x.(x \in \emptyset \Rightarrow x \in A)}}{(\wedge E)\ \mathcal{T}h \vdash \forall x.(x \in \emptyset \Rightarrow x \in A) \Rightarrow \emptyset \subseteq A}$$

$$\dfrac{\dfrac{(Ax)\ \dfrac{}{\mathcal{T}h, x \in \emptyset \vdash \forall x.(x \in \emptyset \Rightarrow \bot)}}{(\forall E)\ \mathcal{T}h, x \in \emptyset \vdash x \in \emptyset \Rightarrow \bot} \qquad (Ax)\ \dfrac{}{\mathcal{T}h, x \in \emptyset \vdash x \in \emptyset}}{(\Rightarrow E)\ \dfrac{\dfrac{\dfrac{\dfrac{\mathcal{T}h, x \in \emptyset \vdash \bot}{\mathcal{T}h, x \in \emptyset \vdash x \in A}(\bot E)}{\mathcal{T}h \vdash x \in \emptyset \Rightarrow x \in A}(\Rightarrow I)}{\mathcal{T}h \vdash \forall x.(x \in \emptyset \Rightarrow x \in A)}(\forall I)}{\mathcal{T}h \vdash \emptyset \subseteq A}}$$

Analysons rapidement cette démonstration : on peut repérer dans la branche de droite les trois règles $(\forall I)$, $(\Rightarrow I)$ et $(\bot E)$ qui constituent le cœur de la démonstration. Tout le reste est constitué d'axiomes, qui servent ici à utiliser les propositions de la théorie $\mathcal{T}h$, et de règles d'élimination, qui servent à sélectionner dans ces axiomes l'information qui est utile pour poursuivre la démonstration.

En passant à la déduction modulo on s'abstrait effectivement des règles qui n'ont pas un sens déductif essentiel dans la démonstration. Considérons par exemple une théorie vide et une congruence définie par le système de réécriture

$$\mathcal{R} = \left\{ \begin{array}{rcl} X \subseteq Y & \rightarrow & \forall x(x \in X \Rightarrow x \in Y) \\ x \in \emptyset & \rightarrow & \bot \end{array} \right.$$

Une démonstration dans PRED modulo $=_{\mathcal{R}}$ de $\emptyset \subseteq A$ devient (avec la formulation où les conversions sont implicites) :

$$\dfrac{\dfrac{(Ax)\ \dfrac{}{x \in \emptyset \vdash_{\cong} \bot}\qquad (x \in \emptyset \ \cong\ \bot)}{(\bot E)\ x \in \emptyset \vdash_{\cong} x \in A}}{(\Rightarrow I)\ \dfrac{\vdash_{\cong} x \in \emptyset \Rightarrow x \in A}{(\forall I)\ \vdash_{\cong} \emptyset \subseteq A}\qquad (\emptyset \subseteq A \ \cong\ \forall x.(x \in \emptyset \Rightarrow x \in A))}$$

On retrouve bien nos trois règles essentielles de la démonstration sans modulo. Par contre, il peut s'avérer plus difficile de construire une telle démonstration. En effet, en déduction naturelle, la règle de déduction utilisée concerne toujours le connecteur de tête de la conclusion (si c'est une introduction) ou d'une prémisse (si c'est une élimination). Ici, pour comprendre que la proposition

$\emptyset \subseteq A$ se démontre en utilisant une règle $(\forall I)$, il faut tout d'abord l'évaluer par le système de réécriture pour connaître son connecteur de tête.

Dans toute cette section, nous allons étudier une variation sur le thème de la déduction modulo, où les règles de réécriture sur les propositions atomiques ne seront plus utilisées dans la congruence, mais seront transformées en règles d'introduction et d'élimination pour des symboles de prédicats. Dans notre exemple, cela donne

$$(\subseteq I)\ \frac{\Gamma, x \in X \vdash x \in Y}{\Gamma \vdash X \subseteq Y}\ (x \notin \mathcal{FV}(\Gamma)) \qquad (\subseteq E)\ \frac{\Gamma \vdash X \subseteq Y \qquad \Gamma \vdash t \in X}{\Gamma \vdash t \in Y} \qquad (\emptyset E)\ \frac{\Gamma \vdash t \in \emptyset}{\Gamma \vdash \phi}$$

Et on peut donc former une troisième version de la démonstration de $\emptyset \subseteq A$:

$$\begin{array}{r} (Ax)\ \dfrac{}{x \in \emptyset \vdash x \in \emptyset} \\ (\emptyset E)\ \dfrac{x \in \emptyset \vdash x \in \emptyset}{x \in \emptyset \vdash x \in A} \\ (\subseteq I)\ \dfrac{x \in \emptyset \vdash x \in A}{\vdash \emptyset \subseteq A} \end{array}$$

qui est plus courte que celle en déduction modulo (les deux règles d'introduction ont été contractées en une seule) et où les symboles introduits et éliminés apparaissent effectivement dans la conclusion ou dans une prémisse. On peut également arguer que les règles d'introduction et d'élimination ainsi définies pour $\subseteq$ correspondent bien à la sémantique qu'on lui donne en mathématiques.

Des règles de réécriture aux règles de déduction

Voyons maintenant comment, dans le cas général d'une proposition atomique P réécrite par une règle $P \to \phi$, on peut construire de nouvelles règles de déduction.

L'idée de base est d'effectuer les introductions (resp. éliminations) de tous les connecteurs $\wedge$, $\forall$ et $\Rightarrow$ apparaissant dans ϕ, puis de collecter toutes les prémisses et conditions d'application de la dérivation ainsi formée pour définir la nouvelle règle. Parfois plusieurs décompositions de ϕ peuvent être suivies, ce qui donne lieu à des règles d'élimination multiples (les introductions sont uniques pour les trois connecteurs considérés).

Exemple 38 *Si* $P \to Q \wedge (R \wedge S)$ *on a :*

$$\frac{\Gamma \vdash Q \qquad \dfrac{\Gamma \vdash R \qquad \Gamma \vdash S}{\Gamma \vdash R \wedge S}}{\Gamma \vdash Q \wedge (R \wedge S)} \qquad \frac{\Gamma \vdash Q \wedge (R \wedge S)}{\Gamma \vdash Q} \qquad \frac{\Gamma \vdash Q \wedge (R \wedge S)}{\Gamma \vdash R \wedge S} \qquad \frac{\Gamma \vdash Q \wedge (R \wedge S)}{\Gamma \vdash R \wedge S}$$

$$\frac{\Gamma \vdash R \wedge S}{\Gamma \vdash R} \qquad \frac{\Gamma \vdash R \wedge S}{\Gamma \vdash S}$$

D'où les règles

$$(PI)\ \frac{\Gamma \vdash Q \qquad \Gamma \vdash R \qquad \Gamma \vdash S}{\Gamma \vdash P} \qquad (PE_1)\ \frac{\Gamma \vdash P}{\Gamma \vdash Q} \qquad (PE_2)\ \frac{\Gamma \vdash P}{\Gamma \vdash R} \qquad (PE_3)\ \frac{\Gamma \vdash P}{\Gamma \vdash S}$$

Il peut également ne pas y avoir de règle d'introduction si $\phi \equiv \bot$, comme c'était le cas pour la définition de l'ensemble vide $\emptyset$.

L'algorithme de décomposition de ϕ pour le calcul des nouvelles règles procède comme suit.

Définition 47 (Calcul des règles de déduction associées à règle de réécriture)

Soit une règle de réécriture $P \to \phi$. *On initialise la procédure par le jugement* $\Gamma \vdash \phi^*$ *où la formule* ϕ *est marquée par* *, *ce qui signifie qu'elle doit être décomposée.*

1. *Pour trouver la règle d'introduction de P, on applique (de bas en haut) les règles de la figure 6.2 tant qu'il reste des formules marquées auxquelles ces règles s'appliquent.*
2. *Pour trouver les règles d'élimination de P, on applique (de haut en bas) les règles de la figure 6.3 tant qu'il reste des formules marquées auxquelles ces règles s'appliquent.*

Enfin on collecte toutes les prémisses ainsi formées, les conditions d'application et la conclusion, et on remplace ϕ par P pour bien obtenir des règles qui concernent P.

$$\frac{\Gamma \vdash \phi^* \qquad \Gamma \vdash \psi^*}{\Gamma \vdash (\phi \wedge \psi)^*} \qquad \frac{\Gamma \vdash \phi^*}{\Gamma \vdash (\forall x.\phi)^*}\ (x \notin \mathcal{FV}(\Gamma))$$

$$\frac{\Gamma, \psi \vdash \phi^*}{\Gamma \vdash (\psi \Rightarrow \phi)^*}$$

FIG. 6.2 – Calcul de la règle d'introduction

$$\frac{\Gamma \vdash (\phi \wedge \psi)^*}{\Gamma \vdash \phi^*} \qquad \frac{\Gamma \vdash (\phi \wedge \psi)^*}{\Gamma \vdash \psi^*} \qquad \frac{\Gamma \vdash (\forall x.\phi)^*}{\Gamma \vdash (\phi[x := t])^*}$$

$$\frac{\Gamma \vdash (\psi \Rightarrow \phi)^* \qquad \Gamma \vdash \psi}{\Gamma \vdash \phi^*} \qquad \frac{\Gamma \vdash \bot^*}{\Gamma \vdash \vartheta}$$

FIG. 6.3 – Calcul des règles d'élimination

Remarquons que, si plusieurs règles de réécriture ont la même proposition atomique P comme membre gauche, on obtient plusieurs règles d'introduction et d'élimination pour P.

Expliquons brièvement pourquoi certaines sous-formules de ϕ ne sont pas marquées pour la décomposition :

1. Dans le calcul des règles d'élimination, une seule prémisse est marquée parce que nous appliquons ces règles du haut vers le bas. Si on marquait une autre prémisse, il faudrait faire intervenir des introductions pour la décomposer. Par ailleurs, on ne marque jamais la formule ϑ car elle est arbitraire et n'est pas une sous-formule de ϕ.
2. Dans le calcul de la règle d'introduction, on ne marque pas les sous-formules qui, dans la règle d'élimination correspondante, se retrouvent dans le contexte car on ne peut les décomposer une fois qu'elles y sont.
3. On ne décompose pas les $\vee$ ni les $\exists$ pour préserver la complétude du système (voir lemme 63).

Un exemple typique est celui de l'implication, pour laquelle l'antécédent n'est pas marqué dans la règle d'introduction, et la prémisse mineure n'est pas marquée dans la règle d'élimination. En effet, supposons par exemple que $P \rightarrow (Q \Rightarrow R) \Rightarrow S$, des introductions et éliminations systématiques donneraient :

$$(P\,I)\ \frac{\Gamma, Q \Rightarrow R \vdash_+ S}{\Gamma \vdash_+ P} \qquad (P\,E)\ \frac{\Gamma \vdash_+ P \qquad \Gamma, Q \vdash_+ R}{\Gamma \vdash_+ S}$$

Pour permettre l'élimination d'une coupure associée à ces règles, nous avons choisi d'interdire la décomposition de $Q \Rightarrow R$ dans $(P\,E)$, ce que décrivent les formules marquées.

Un autre choix pourrait être de décomposer également $Q \Rightarrow R$ dans $(P\,I)$, ce qu'aucune règle de déduction naturelle ne permet. Il faudrait donc forcer une décomposition dans le style des règles gauche du calcul des séquents, et même ainsi on n'aurait pas une façon immédiate de réorganiser les prémisses pour éliminer la coupure :

$$\begin{array}{ll} (P\,I) & \dfrac{\Gamma, R \vdash_+ S \qquad \Gamma \vdash_+ Q}{\Gamma \vdash_+ P} \qquad\qquad \Gamma, Q \vdash_+ R \\ (P\,E) & \dfrac{}{\Gamma \vdash_+ S} \end{array}$$

Définition 48 (Système de déduction naturelle généralisé)
Étant donné un système de réécriture $\mathcal{R} = \mathcal{R}_t \cup \mathcal{R}_P$ tel que $\mathcal{R}_t$ réécrit des termes et $\mathcal{R}_P$ réécrit des propositions atomiques, le système de déduction naturelle généralisé *associé à $\mathcal{R}$ est donné par :*

- *les règles de la logique du premier ordre ;*
- *les règles construites à partir de $\mathcal{R}_P$;*

où toutes les propositions sont considérées modulo $=_{\mathcal{R}_t}$. On notera $\Gamma \vdash_+ \phi$ les jugements dérivés dans un tel système.

Notons enfin qu'un travail similaire peut être conduit à partir du calcul des séquents modulo [DHK03]. On obtient alors un calcul des séquents généralisé par de nouvelles règles gauches et droites pour chaque symbole de prédicat défini par une règle de réécriture.

Comme il est possible en calcul des séquents de décomposer des formules à gauche comme à droite du symbole $\vdash$, la décomposition des formules est plus systématique qu'ici : en considérant des connecteurs multiplicatifs, on peut décomposer sous les connecteurs $\vee$ et $\exists$. En revanche, l'intérêt du point de vu de la démonstration interactive est moins évident : les démonstrations en calcul des séquents sont moins "naturelles', au sens où la structure des formules n'est plus lisible dans les démonstrations qu'on effectue.

Travaux similaires

Jusque dans les travaux originels de D. Prawitz [Pra65], on trouve des règles de déduction permettant d'introduire ou d'éliminer une proposition arbitraire. En effet, il propose d'ajouter aux règles de la logique du premier ordre des règles de *folding* et d'*unfolding* grâce auxquelles on peut directement remplacer une proposition atomique P par une autre proposition Q si l'axiome $P \Leftrightarrow Q$ est présent dans la théorie. Comme remarqué par G. Dowek [Dow01], la déduction modulo généralise la déduction naturelle avec *folding-unfolding* dans la mesure où ces deux règles peuvent être présentées comme des étapes de conversion, qui sont implicites en déduction modulo. Notre généralisation de la déduction naturelle va encore plus loin, puisqu'une unique règle effectue non seulement le remplacement d'une proposition atomique P par sa définition Q, mais aussi la décomposition de Q selon ses différents connecteurs.

L'idée de créer des nouvelles règles d'inférence en contractant plusieurs règles n'est pas nouvelle non plus : X. Huang a défini les *démonstrations au niveau des assertions* [Hua94], où comme ici on essaye de porter la démonstration au niveau des concepts mathématiques manipulés. Par contre, ses travaux et ceux qui en sont issus sont assez centrés sur la *reconstruction* de démonstrations : on confie généralement le problème à un logiciel de démonstration automatique, et on cherche ensuite à présenter la démonstration obtenue sous une forme lisible. Même si une démonstration est faite au niveau des assertions, sa vérification passe toujours par une phase d'expansion pour revenir au niveau des règles de la déduction naturelle, alors que notre formalisme permet de *construire, présenter et vérifier* les démonstrations au niveau des assertions.

Cela dit, il pourrait être intéressant de réutiliser la technique de *chunking* de Huang, qui consiste à détecter dans une démonstration en déduction naturelle usuelle des sous-preuves correspondant à une assertion. Nous aurions alors un moyen de récupérer des démonstrations antérieures (faites en Coq par exemple) et de les rendre plus lisibles et donc plus faciles à maintenir et à réutiliser.

On peut également citer les techniques de *focalisation*, proposées par J.-M. Andreoli [And92], qui consistent à regrouper en une seule étape des inférences sur des symboles similaires : par exemple

$$\dfrac{\dfrac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q} \quad \dfrac{\Gamma \vdash R \quad \Gamma \vdash S}{\Gamma \vdash R \wedge S}}{\Gamma \vdash (P \wedge Q) \wedge (R \wedge S)} \quad \text{devient} \quad \dfrac{\Gamma \vdash P \quad \Gamma \vdash Q \quad \Gamma \vdash R \quad \Gamma \vdash S}{\Gamma \vdash (P \wedge Q) \wedge (R \wedge S)}$$

Le but de cette manipulation est plus de réduire le non-déterminisme dans la recherche de démonstration que d'avoir un mécanisme de démonstration de haut niveau : il permet d'identifier des démonstrations équivalentes, mais ses aspects sémantiques sont assez restreints.

Enfin, le calcul CoRe (pour *contextual reasoning*) de S. Autexier [Aut05] permet un raisonnement au niveau des assertions, grâce à une notion de *pas de démonstration* très générale, dont la justification peut être une définition, un lemme ou une hypothèse. Il serait donc intéressant d'étudier si notre déduction naturelle généralisée peut être vue comme une instance de CoRe.

6.2.2 Premiers résultats

Comme pour la déduction modulo, le lemme d'équivalence qui démontre la correction et la complétude par rapport à la logique du premier ordre est un passage obligé. On notera d'ailleurs qu'il démontre du même coup l'équivalence avec la déduction modulo.

Nous nous intéresserons ensuite à la notion de coupure dans ce cadre, et enfin nous étudierons des instances particulières de systèmes de déduction généralisés, qui permettent de représenter la logique d'ordre supérieur ou les prédicats définis par induction.

Lemme 63 (Équivalence)
Pour tout système déduction naturelle généralisé associé à un TRS $\mathcal{R}$, il existe une théorie $\mathcal{T}h$ telle que $\Gamma \vdash_+ \phi$ si et seulement si $\Gamma, \mathcal{T}h \vdash \phi$.

Démonstration : Pour construire la théorie $\mathcal{T}h$ on procède exactement comme en déduction modulo. S'il n'existe pas déjà un prédicat d'égalité $=$, on en ajoute un ainsi que les axiomes correspondants. Chaque règle de réécriture $l \to r$ se traduit alors en un axiome :

- $\forall \overline{x}.(l = r)$ si $l \to r \in \mathcal{R}_t$
- $\forall \overline{x}.(l \Leftrightarrow r)$ si $l \to r \in \mathcal{R}_P$

où $\overline{x}$ est l'ensemble des variables libres dans l et r.

Voyons alors la correction et la complétude de $\vdash_+$ par rapport à cette théorie. Nous ne nous préoccuperons pas de $=_{\mathcal{R}_t}$ ni des axiomes correspondants, qui se traitent tout-à-fait comme en déduction modulo.

Si $\Gamma \vdash_+ \phi$ alors $\Gamma, \mathcal{T}h \vdash \phi$: pour obtenir une démonstration de $\Gamma, \mathcal{T}h \vdash \phi$, on peut reproduire tels quels les règles de la logique du premier ordre ainsi que les axiomes de Γ utilisés dans la démonstration de $\Gamma \vdash_+ \phi$. Il suffit donc de traduire les règles d'inférence généralisées.

Supposons par exemple une introduction généralisée induite par une règle $P \to \phi$:

$$(P\,I)\ \frac{H_1 \quad \dots \quad H_n}{\Gamma \vdash_+ P}\ (\mathcal{C})$$

Par construction de cette règle d'inférence, il existe une démonstration en logique du premier ordre dont l'ensemble des prémisses est $H_1, \dots, H_n$, dont les conditions d'application sont celles de $\mathcal{C}$ et dont la conclusion est ϕ.
Pour obtenir une démonstration de $\Gamma, \mathcal{T}h \vdash P$ utilisant les mêmes prémisses, il suffit d'ajouter $\mathcal{T}h$ à Γ (ce qui est toujours possible par affaiblissement) et de compléter la démonstration comme suit :

$$\dfrac{\dfrac{(Ax)\ \dfrac{}{\Gamma, \mathcal{T}h \vdash \phi \Leftrightarrow P}}{(\wedge E)\ \dfrac{}{\Gamma, \mathcal{T}h \vdash \phi \Rightarrow P}} \qquad \overset{H_1 \quad \dots \quad H_n}{\Gamma, \mathcal{T}h \vdash \phi}}{(\Rightarrow E)\ \Gamma, \mathcal{T}h \vdash P}$$

Le cas de l'élimination se traite de façon tout-à-fait similaire : pour une règle

$$(P\,E)\ \frac{\Gamma \vdash_+ P \qquad H_1 \quad \dots \quad H_n}{\Gamma \vdash_+ C}\ (\mathcal{C})$$

il existe une démonstration en logique du premier ordre ayant la même conclusion, les mêmes conditions d'application, et les mêmes prémisses excepté $\Gamma \vdash P$ qui devient $\Gamma \vdash \phi$. On peut donc également ajouter $\mathcal{T}h$ par affaiblissement et compléter la démonstration ainsi :

$$\dfrac{\dfrac{(Ax)\ \dfrac{}{\Gamma, \mathcal{T}h \vdash P \Leftrightarrow \phi}}{(\wedge E)\ \dfrac{}{\Gamma, \mathcal{T}h \vdash P \Rightarrow \phi}} \qquad \Gamma, \mathcal{T}h \vdash P}{(\Rightarrow E)\ \Gamma, \mathcal{T}h \vdash \phi} \qquad \overset{H_1 \quad \dots \quad H_n}{\Gamma, \mathcal{T}h \vdash C}$$

Si $\Gamma, \mathcal{T}h \vdash \phi$ **alors** $\Gamma \vdash_+ \phi$: Comme $\vdash_+$ englobe la logique du premier ordre, toutes les règles d'inférence utilisées dans la démonstration de $\Gamma, \mathcal{T}h \vdash \phi$ peuvent être utilisées également dans la démonstration que nous allons construire, à l'exception des règles (Ax) qui utilisent un axiome de $\mathcal{T}h$. Par conséquent, pour démontrer la complétude de $\vdash_+$, il suffit de voir que tous les axiomes de $\mathcal{T}h$ sont démontrables, autrement dit que tout proposition P définie par une règle de réécriture $P \to \phi$ peut être démontrée équivalente à sa définition ϕ.
Construisons une démonstration de $\phi \vdash_+ P$ la forme suivante :

$$\dfrac{(Ax)\ \dfrac{}{\vdots} \qquad\qquad (Ax)\ \dfrac{}{\vdots}}{(K\,E)\ \dfrac{}{\phi, \Gamma_1 \vdash_+ \phi_1} \quad \dots \quad \dfrac{}{\phi, \Gamma_n \vdash_+ \phi_n}}{(P\,I)\ \phi \vdash_+ P}$$

où les K sont des connecteurs choisis parmi $\wedge$, $\forall$ et $\Rightarrow$; les ϕ_i sont des sous-formules de ϕ qui ont été marquées lors du calcul des règles $(P\,I)$ et $(P\,E)$; les Γ_i sont constitués de sous-formules de ϕ.

On démontrera donc la proposition suivante : pour tout jugement $\Gamma_i \vdash \phi_i^*$ rencontré dans la construction de $(P\,I)$, on a $\Gamma_i, \phi \vdash_+ \phi_i$. En particulier, les prémisses de $(P\,I)$ sont dans ce cas, et donc on aura bien la démonstration recherchée. On procède par récurrence sur la construction de $(P\,I)$ (remarquons que plus cette construction est petite, plus la formule ϕ_i^* est grande puisqu'on descend dans les sous-termes de ϕ^*) :

Si $\Gamma_i \vdash \phi_i^*$ **est** $\emptyset \vdash \phi^*$ **lui-même** alors trivialement $\Gamma_i, \phi \vdash_+ \phi$ (et $\Gamma_i = \emptyset$). Ce sont les axiomes qui apparaissent tout en haut de notre démonstration de $\phi \vdash_+ P$.

Si $\Gamma_i \vdash \phi_i^*$ **est obtenu à partir de** $\Gamma_i \vdash (\phi_i \wedge \phi_j)^*$ alors par hypothèse de récurrence on a $\Gamma_i, \phi \vdash_+ \phi_i \wedge \phi_j$. Par application d'une règle $(\wedge E)$, on démontre $\Gamma_i, \phi \vdash_+ \phi_i$.

Si $\Gamma_i \vdash \phi_i^*$ **est obtenu à partir de** $\Gamma_i \vdash (\forall x.\phi_i)^*$ alors par hypothèse de récurrence on a $\Gamma_i, \phi \vdash_+ \forall x.\phi_i$. Par application d'une règle $(\forall E)$, on démontre $\Gamma_i, \phi \vdash_+ \phi_i$.

Si $\Gamma_i, \phi_j \vdash \phi_i^*$ **est obtenu à partir de** $\Gamma_i \vdash (\phi_j \Rightarrow \phi_i)^*$ alors par hypothèse de récurrence on a $\Gamma_i, \phi \vdash_+ \phi_j \Rightarrow \phi_i$. Par affaiblissement pour ajouter ϕ_j dans le contexte puis par application d'une règle $(\Rightarrow E)$, on démontre $\Gamma_i, \phi_j, \phi \vdash_+ \phi_i$.

Pour l'autre sens de l'équivalence, la démarche est très similaire : on construit une démonstration de $P \vdash_+ \phi$ de la forme

$$(K\,I)\ \frac{\begin{array}{cc}(P\,E)\ \dfrac{P, \Gamma_1 \vdash_+ P \quad \ldots \quad P, \Gamma_1 \vdash_+ \phi_j}{P, \Gamma_1 \vdash_+ \phi_1} & \\ \vdots & \ldots \\ \end{array} \qquad \begin{array}{c}(P\,E)\ \dfrac{P, \Gamma_n \vdash_+ P \quad \ldots \quad P, \Gamma_n \vdash_+ \phi_k}{P, \Gamma_n \vdash_+ \phi_n} \\ \vdots \end{array}}{P \vdash_+ \phi}$$

Pour cela, on procède par récurrence sur une décomposition de ϕ selon les connecteurs $\wedge$, $\forall$ et $\Rightarrow$. On démontre que pour toutes les prémisses $P, \Gamma_i \vdash_+ \phi_i$ ainsi obtenues, il existe une règle d'élimination de P dont la conclusion est ϕ_i et dont les prémisses sont uniquement P et des formules de Γ_i. La démonstration se fait sur des sous-buts et sur des étapes de construction de $(P\,E)$, dont les prémisses et les $(P\,E)$ sont des cas particuliers.

Si $\phi_i \equiv \phi_j \wedge \phi_k$ alors la décomposition de ϕ_i donne les deux sous-buts $P, \Gamma_i \vdash_+ \phi_j$ et $P, \Gamma_i \vdash_+ \phi_k$. Dans la construction des $(P\,E)$, lorsqu'on considère $(\phi_j \wedge \phi_k)^*$, on construit deux règles dont les conclusions sont obtenues à partir de ϕ_j^* et ϕ_k^*. Aucune nouvelle prémisse n'est créée, donc par hypothèse de récurrence on a bien les règles nécessaires.

Si $\phi_i \equiv \forall x.\phi_j$ alors la décomposition de ϕ_i donne le sous-but $P, \Gamma_i \vdash_+ \phi_j$ avec $x \notin \Gamma_i$. Dans la construction des $(P\,E)$, lorsqu'on considère $(\forall x.\phi_j)^*$, on construit une règle dont la conclusion est obtenue à partir de $\phi_j^*[x := t]$. Aucune nouvelle prémisse n'est créée, donc par hypothèse de récurrence, en prenant $t \equiv x$, on a bien les règles nécessaires.

Si $\phi_i \equiv \phi_j \Rightarrow \phi_k$ alors la décomposition de ϕ_i donne le sous-but $P, \Gamma_i, \phi_i \vdash_+ \phi_j$ et $P, \Gamma_i \vdash_+ \phi_k$. Dans la construction des $(P\,E)$, lorsqu'on considère $(\phi_j \Rightarrow \phi_k)^*$, on construit une règle dont la conclusion est obtenue à partir de ϕ_k^*, et on ajoute la prémisse $\vdash \phi_j$. Or ϕ_j a été ajoutée au contexte dans la décomposition de ϕ_i et donc on pourra la démontrer.

Si ϕ_i est atomique ou a un autre symbole de tête alors la décomposition de ϕ_i s'arrête, et on obtient une règle d'élimination pour P. Or P est dans le contexte P, Γ_i, et par hypothèse de récurrence, toutes les autres prémisses ont été chargées dans Γ_i, donc toutes les prémisses de $(P\,E)$ sont démontrables.

La théorie $\mathcal{T}h$ utilisée ici étant la même que pour le lemme d'équivalence de la déduction modulo, on a également le résultat suivant : le système de déduction généralisée associé à un TRS $\mathcal{R}$ est consistant si et seulement si la déduction modulo $=_{\mathcal{R}}$ est consistante. □

Remarque 10 (Décomposition des disjonctions et quantifications existentielles)
Dans la construction des nouvelles règles d'inférence, nous n'avons décomposé la formule de définition ϕ que selon $\wedge$, $\forall$ et $\Rightarrow$, et ce pour pouvoir démontrer le lemme d'équivalence. En effet, on pourrait ajouter un niveau de décomposition pour $\vee$ et $\exists$, au prix de la complétude de $\vdash_+$ (la correction reste valable). Un moyen de retrouver la complétude est de raisonner dans $\vdash_+$ avec le sous-ensemble de la théorie $\mathcal{T}h$ composé des axiomes $\phi \Rightarrow P$.

Définition 49 (Coupure généralisée)
Dans $\vdash_+$, les coupures sont celles de la logique du premier ordre auxquelles s'ajoutent celles formées d'une règle $(P\,I)$ immédiatement suivie d'une règle $(P\,E)$.

Exemple 39 (Une coupure sur le symbole $\subseteq$)
La démonstration

$$(\subseteq E)\ \frac{\begin{array}{c}\vdots\ \mathcal{D}_2\\ \Gamma \vdash_+ t \in X\end{array} \qquad (\subseteq I)\ \dfrac{\begin{array}{c}\vdots\ \mathcal{D}_1\\ \Gamma, x \in X \vdash_+ x \in Y\end{array}}{\Gamma \vdash_+ X \subseteq Y}\ (x \notin \mathcal{FV}(\Gamma))}{\Gamma \vdash_+ t \in Y}$$

se réduit à

$$\dfrac{\dfrac{\begin{array}{c}\vdots\ \mathcal{D}_2\\ \cdots\end{array} \quad \begin{array}{c}\vdots\ \mathcal{D}_2\\ \cdots\end{array}}{\begin{array}{c}\vdots\ \mathcal{D}_1\\ \cdots\end{array}}}{\Gamma \vdash_+ t \in Y}$$

Théorème 64 (Élimination des coupures généralisées)
Dans $\vdash_+$, toute démonstration admet un représentant sans coupure si et seulement si sa traduction en logique du premier ordre admet un représentant sans coupure.

Démonstration : Par application du lemme d'équivalence. Une coupure $(P\,I)(P\,E)$ se traduit en logique du premier ordre par n coupures correspondant aux n connecteurs rencontrés dans la définition ϕ de P. □

Remarquons que si $P \longrightarrow \phi_1\,K\,\phi_2$, les successions de règles $(P\,I)$ $(K\,E)$ et $(K\,I)$ $(P\,E)$ ne peuvent pas être des coupures, tout simplement parce que ces règles ne peuvent pas se succéder dans une démonstration valide : la prémisse principale de $(K\,E)$ est une formule dont le connecteur de tête est K alors que la conclusion de $(P\,I)$ est une formule atomique, et *vice versa*.

On voit aussi ici une justification *a posteriori* pour ne pas avoir considéré des règles de réécriture dont le membre gauche est une proposition non atomique. En effet, si par exemple on avait eu une règle $\psi_1 \wedge \psi_2 \to \phi$, elle se serait traduit par une règle d'introduction

$$(\psi I)\ \frac{H_1 \qquad \dots \qquad H_n}{\Gamma \vdash_+ \psi_1 \wedge \psi_2}$$

et il aurait donc été possible de former la démonstration

$$(\wedge E)\ \frac{(\psi I)\ \dfrac{H_1 \qquad \dots \qquad H_n}{\Gamma \vdash_+ \psi_1 \wedge \psi_2}}{\Gamma \vdash_+ \psi_1}$$

dans laquelle le connecteur $\wedge$ est introduit puis éliminé mais qui ne peut pas être simplifiée. La notion de coupure serait donc obscurcie par des règles d'introduction et d'élimination supplémentaires sur des propositions non atomiques.

Représentation d'autres logiques

Pour terminer cette section, observons ce qui se passe lorsqu'on formalise d'autres logiques dans la logique du premier ordre. Par exemple, la logique d'ordre supérieur peut être formulée comme une théorie du premier ordre [Dow97]. En déduction modulo, cela se représente par des règles de réécriture comme

$$\epsilon(\alpha(\forall, x)) \quad \to \quad \forall y \epsilon(\alpha(x, y))$$

Il est intéressant de constater que si on construit les règles d'introduction et d'élimination associées, on retrouve des règles de déduction de la logique d'ordre supérieure :

$$\frac{\Gamma \vdash \epsilon(\alpha(x, y))}{\Gamma \vdash \epsilon(\alpha(\forall, x))}\ (y \notin \mathcal{FV}(\Gamma)) \qquad\qquad \frac{\Gamma \vdash \epsilon(\alpha(\forall, x))}{\Gamma \vdash \epsilon(\alpha(x, \vartheta))}$$

On peut donc supposer que notre formalisme fournit une vision unifiée de plusieurs logiques ; pour conforter cette idée, il faudrait étudier des représentations d'autres logiques comme la logique modale, ce qui fera l'objet d'études futures.

Une autre piste de recherche prometteuse est la représentation des prédicats inductifs. La représentation de l'arithmétique comme une théorie modulo proposée dans [DW05] utilise le prédicat suivant pour définir les entiers naturels :

$$n \in N \quad \to \quad \forall P.(0 \in P \Rightarrow \forall m.(m \in N \Rightarrow m \in P \Rightarrow S(m) \in P))$$

La règle d'élimination obtenue à partir de cette définition constitue alors une règle de démonstration pour les prédicats inductifs sur les entiers naturels :

$$(N\,E)\ \frac{\Gamma \vdash n \in N \qquad \Gamma \vdash 0 \in P \qquad \Gamma \vdash \forall m.(m \in P \Rightarrow S(m) \in P)}{\Gamma \vdash n \in P}\ (P \notin \mathcal{FV}(\Gamma))$$

En poussant la décomposition un peu plus loin, on obtient une règle encore plus concise :

$$(N\,E)\ \frac{\Gamma \vdash n \in N \qquad \Gamma \vdash 0 \in P \qquad \Gamma, m \in P \vdash S(m) \in P}{\Gamma \vdash n \in P}\ (P,m \notin \mathcal{FV}(\Gamma))$$

Il est vraisemblable qu'on puisse obtenir une telle règle pour tous les ensembles inductifs à partir du moment où on en trouve une représentation comme une théorie modulo.

6.2.3 Une théorie des types généralisée

Dans cette dernière section, nous définissons un langage de termes de preuve avec motifs, qui fait en partie le lien avec le ρ-calcul. Nous avons vu que, lorsqu'on utilise le λ-calcul comme langage de termes de preuve pour la logique du premier ordre, il faut ajouter des constructions *ad hoc* pour chaque connecteur. En toute logique, il faudrait ici aussi considérer un nouveau constructeur pour chaque règle $(P\,I)$ ou $(P\,E)$, puis définir de nouvelles règles de réduction pour représenter l'élimination des coupures. Cependant, nos nouvelles règles sont en quelque sorte dérivées des règles pour l'implication, la quantification universelle et la conjonction. Pour traduire celles-ci en termes de preuve, il suffit donc de pouvoir transmettre dans une abstraction :

- des variables de démonstrations (pour les implications) ;
- des variables de termes (pour les quantifications) ;
- des projections (pour les conjonctions).

Nous rassemblerons donc toutes ces informations dans un motif ; les projections elles-mêmes seront données par des ρ-termes qui sélectionnent une feuille dans un arbre de conjonctions, ce qui généralise les projections du λ-calcul.

Le langage de termes de preuve est donc celui des termes de preuve de la logique du premier ordre, auquel s'ajoutent une construction d'arbres binaires (que l'on notera T, U), un langage de motifs et une abstraction correspondante :

$$\begin{array}{rcl} \mathcal{A} & ::= & \mathcal{T}_\lambda \mid <\mathcal{A},\mathcal{A}> \\ \mathcal{T}_\lambda & ::= & \lambda P(\overline{x};\overline{\alpha};z).(z\mathcal{A}) \mid \mathcal{T}_\lambda\, P(\overline{t};\overline{\mathcal{T}_\lambda};\lambda\mathcal{A}.\alpha) \end{array}$$

Dans un motif $P(\overline{x};\overline{\alpha};z)$, les variables $\overline{x}$ seront instanciées par des termes algébriques ; les variables $\overline{\alpha}$ seront instanciées par des termes de preuve ; enfin la variable z sera instanciée par une projection d'un arbre binaire vers une de ses feuilles.

Les règles d'introduction ont été construites de bas en haut. Pour trouver le terme de preuve associé à une telle règle, on procède comme suit.

Définition 50 (Calcul des règles de typage des λ-abstractions)
Soit une règle d'introduction $(P\,I)$ associée à une règle de réécriture $P \to \phi$.

1. *Pour chaque prémisse $\Gamma_i \vdash \phi_i$, on suppose un terme de preuve A_i.*
2. *Reprendre la construction de la règle $(P\,I)$.*
3. *Initialiser le calcul de la règle de typage au niveau des prémisses, en assimilant A_i à $\lambda P(;;z).(zA_i)$.*
4. *Parcourir la construction de $(P\,I)$ de haut en bas en suivant les règles de la figure 6.4 pour produire un terme de preuve $\lambda P(\overline{x};\overline{\alpha};z).(zA)$.*

Une fois ce terme de preuve défini, on peut également construire les termes de preuve des éliminations correspondantes.

Définition 51 (Calcul des règles de typage des applications)
Soit une règle d'élimination $(P\,E)$ associée à une règle de réécriture $P \to \phi$.

1. *Pour chaque prémisse $\Gamma \vdash \psi$ (sauf la prémisse principale), on suppose donné un terme de preuve C_ψ. Pour la prémisse principale $\Gamma \vdash P$ on suppose un terme de preuve A.*
2. *Calculer le terme de preuve $\lambda P(\overline{x};\overline{\alpha};z).(zA')$ pour la règle d'introduction $(P\,I)$ construite à partir de la même règle de réécriture $P \to \phi$.*

$$\frac{\Gamma \vdash \lambda P(\overline{x};\overline{\alpha};z).(zA) : \phi^* \qquad \Gamma \vdash \lambda P(\overline{y};\overline{\beta};z).(zB) : \psi^*}{\Gamma \vdash \lambda P(\overline{x},\overline{y};\overline{\alpha},\overline{\beta};z).(z{<}A,B{>}) : (\phi \wedge \psi)^*}$$

$$\frac{\Gamma \vdash \lambda P(\overline{x};\overline{\alpha};z).(zA) : \phi^*}{\Gamma \vdash \lambda P(y,\overline{x};\overline{\alpha};z).(zA) : (\forall y.\phi)^*}\ (x \notin \mathcal{FV}(\Gamma))$$

$$\frac{\Gamma, \alpha_\psi{:}\psi \vdash \lambda P(\overline{x};\overline{\alpha};z).(zA) : \phi^*}{\Gamma \vdash \lambda P(\overline{x};\alpha_\psi,\overline{\alpha};z).(zA) : (\psi \Rightarrow \phi)^*}$$

FIG. 6.4 – Calcul des règles de typage des λ-abstractions

3. *Reprendre la construction de la règle* $(P\,E)$.
4. *Initialiser le calcul de la règle de typage en affectant à la prémisse principale* $\Gamma \vdash \phi$ *le terme* $A\,P(\overline{x};\overline{\alpha};\lambda\beta.\beta)$, *où* A *est le terme de preuve de* $\Gamma \vdash P$ *et où les* $\overline{x}$ *et* $\overline{\alpha}$ *sont les mêmes que pour* $(P\,I)$.
5. *Parcourir la construction de la règle d'élimination (de haut en bas) selon les règles de la figure 6.5 pour instancier les variables* $\overline{\alpha}, \overline{x}$ *et modifier la projection* $\lambda\beta.\beta$.
6. *Les* $\overline{x}$ *et les* $\overline{\alpha}$ *non instanciés peuvent être remplacés par des termes quelconques ; ils ne joueront de toute façon aucun rôle dans les éventuelles réductions du terme de preuve.*

$$\frac{\Gamma \vdash A\,P(\overline{t};\overline{B};\lambda T.\alpha) : (\phi \wedge \psi)^*}{\Gamma \vdash A\,P(\overline{t};\overline{B};\lambda T[\alpha := {<}\beta,\gamma{>}].\beta) : \phi^*} \qquad \frac{\Gamma \vdash A\,P(\overline{t};\overline{B};\lambda T.\alpha) : (\phi \wedge \psi)^*}{\Gamma \vdash A\,P(\overline{t};\overline{B};\lambda T[\alpha := {<}\beta,\gamma{>}].\gamma) : \psi^*}$$

$$\frac{\Gamma \vdash A\,P(\overline{t};\overline{B};\lambda T.\alpha) : (\psi \Rightarrow \phi)^* \qquad \Gamma \vdash C_\psi : \psi}{\Gamma \vdash A\,P(\overline{t};\overline{B}[\alpha_\psi := C_\psi];\lambda T.\alpha) : \phi^*}$$

$$\frac{\Gamma \vdash A\,P(\overline{t};\overline{B};\lambda T.\alpha) : (\forall y.\phi)^*}{\Gamma \vdash A\,P(\overline{t}[y := u];\overline{B};\lambda T.\alpha) : (\phi[y := u])^*} \qquad \frac{\Gamma \vdash A\,P(\overline{t};\overline{B};\lambda T.\alpha) : \bot^*}{\Gamma \vdash A\,P(\overline{t};\overline{B};\lambda T.botelim(\alpha)) : \vartheta}$$

FIG. 6.5 – Calcul des règles de typage des applications

Remarquons que lorsque la formule ϕ ne comporte pas de conjonction, la projection utilisée est toujours $\lambda\alpha.\alpha$, et on peut donc s'en dispenser.

Exemple 40 (Termes de preuve pour l'inclusion)
Notre définition de $\subseteq$ *utilise un témoin, charge une hypothèse dans le contexte, et ne comporte pas de conjonction. Les termes de preuve associés sont donc donnés par les règles de typage :*

$$(\subseteq I)\ \frac{\Gamma, \alpha{:}x \in X \vdash A : x \in Y}{\Gamma \vdash \lambda{\subseteq}(x;\alpha;z).(zA) : X \subseteq Y} \qquad (\subseteq E)\ \frac{\Gamma \vdash A : X \subseteq Y \qquad \Gamma \vdash B : t \in X}{\Gamma \vdash A{\subseteq}(t;B;\lambda\alpha.\alpha) : t \in Y}$$

Ici la projection n'était pas nécessaire.

Remarquons d'ailleurs que, si on dispose de ce genre de termes de preuve, les constructions spécifiques à la conjonction ne sont plus nécessaires : on peut utiliser le filtrage.

$$(\wedge I)\ \frac{\Gamma \vdash A : \phi \qquad \Gamma \vdash B : \psi}{\Gamma \vdash \lambda\wedge(;;z).(z{<}A, B{>}) : \phi \wedge \psi}$$

$$(\wedge E_l)\ \frac{\Gamma \vdash A : \phi \wedge \psi}{\Gamma \vdash A\wedge(;;\lambda{<}\alpha,\beta{>}.\alpha) : \phi} \qquad\qquad (\wedge E_r)\ \frac{\Gamma \vdash A : \phi \wedge \psi}{\Gamma \vdash A\wedge(;;\lambda{<}\alpha,\beta{>}.\beta) : \psi}$$

L'élimination d'une coupure généralisée est représentée par deux ρ-réductions successives : la première $\mapsto_p$ transmet les témoins $\overline{t}$ et les lemmes $\overline{B}$ à la démonstration T d'un arbre de conjonctions.

$$(\lambda P(\overline{x};\overline{\alpha};z).(zU))\, P(\overline{t};\overline{B};\lambda T.\beta) \quad \mapsto_p \quad (\lambda T.\beta)\,(U[\overline{x := t}, \overline{\alpha := B}])$$

La seconde sélectionne une feuille de cet arbre :

$$(\lambda T.\beta)\, U \quad \mapsto_p \quad A$$

où A est le terme situé dans U à la même position que β dans T.

Il reste à étudier les propriétés de ce calcul typé. En particulier, il serait intéressant d'étudier sa normalisation forte afin de pouvoir démontrer directement l'élimination des coupures sans passer par l'équivalence avec la déduction modulo. La difficulté d'une telle étude est similaire à celle de la déduction modulo : le système de types est paramétré par le TRS $\mathcal{R}$ puisque c'est à partir de celui-ci qu'on construit les règles de typage.

Il est notamment intéressant de voir les règles d'inférence et les termes de preuve que l'on obtient lorsqu'on considère des règles de réécriture incorrectes. Par exemple, la déduction modulo devient inconsistante si on prend $\mathcal{R} = \{R \ \rightarrow \ R \Rightarrow \bot\}$, et d'ailleurs les démonstrations de $\vdash_{\cong} \bot$ n'admettent pas de représentant sans coupure. Les règles de $\vdash_+$ associées à cette règle de réécriture sont

$$(R\,I)\ \frac{\Gamma, R \vdash_+ \bot}{\Gamma \vdash_+ R} \qquad\qquad (R\,E)\ \frac{\Gamma \vdash_+ R \qquad \Gamma \vdash_+ R}{\Gamma \vdash_+ \bot}$$

La règle $(R\,E)$ montre bien que R pose problème, puisqu'elle permet d'introduire $\bot$ à la seule condition de savoir démontrer R. On peut construire la démonstration suivante de $\vdash_+ \bot$:

$$\dfrac{\dfrac{\dfrac{(Ax)\ \overline{R \vdash_+ R} \qquad (Ax)\ \overline{R \vdash_+ R}}{R \vdash_+ \bot}(R\,E)}{\vdash_+ R}(R\,I) \qquad \dfrac{\dfrac{(Ax)\ \overline{R \vdash_+ R} \qquad (Ax)\ \overline{R \vdash_+ R}}{R \vdash_+ \bot}(R\,E)}{\vdash_+ R}(R\,I)}{\vdash_+ \bot}(R\,E)$$

Lorsqu'on la décore avec des termes de preuve, on obtient la dérivation de typage suivante :

$$\dfrac{\dfrac{\dfrac{(Ax)\ \overline{\alpha{:}R \vdash_+ \alpha : R} \qquad (Ax)\ \overline{\alpha{:}R \vdash_+ \alpha : R}}{\alpha{:}R \vdash_+ \alpha\, R(\alpha) : \bot}(R\,E)}{\vdash_+ \lambda R(\alpha).\alpha\, R(\alpha) : R}(R\,I) \qquad \dfrac{\dfrac{(Ax)\ \overline{\alpha{:}R \vdash_+ \alpha : R} \qquad (Ax)\ \overline{\alpha{:}R \vdash_+ \alpha : R}}{\alpha{:}R \vdash_+ \alpha\, R(\alpha) : \bot}(R\,E)}{\vdash_+ \lambda R(\alpha).\alpha\, R(\alpha) : R}(R\,I)}{\vdash_+ \big(\lambda R(\alpha).\alpha\, R(\alpha)\big)\, R\big(\lambda R(\alpha).\alpha\, R(\alpha)\big) : \bot}(R\,E)$$

Et il est facile de voir que le terme de preuve $\big(\lambda R(\alpha).\alpha\, R(\alpha)\big)\ R\big(\lambda R(\alpha).\alpha\, R(\alpha)\big)$ n'est autre que le ρ-terme $\omega_f\,(f\,\omega_f)$ vu à la section 4.2.1.

Retour sur les systèmes de types pour le ρ-calcul et les logiques sous-jacentes

Les deux sections de ce chapitre abordent des problèmes similaires mais relèvent d'approches quasiment opposées, qui chacune nous ont permis de mieux comprendre certains aspects du calcul de réécriture.

Dans la première approche, nous nous sommes basés sur un système des P^2TS pour définir des termes de preuve pour la déduction naturelle modulo. Les P^2TS étant eux-mêmes une extension des PTS, il a été immédiatement possible de réutiliser le fait que les λ-termes constituaient une représentation linéaire des démonstrations en logique du premier ordre. Par contre, nous avons utilisé les motifs et les capacités de filtrage présentes dans les P^2TS plutôt dans le style d'un Logical Framework, puisque l'utilisation d'une constante Rew^p ou Rew^s correspond exactement à l'utilisation d'une règle (Cong).

Cette utilisation finalement assez restreinte du filtrage suggère que les P^2TS n'ont pas significativement plus de pouvoir expressif du point de vue déductif que les PTS. Un véritable isomorphisme de Curry-Howard-de Bruijn pour les P^2TS, au sens où un terme représente une démonstration et se réduit de la même façon, donnerait vraisemblablement une logique très comparable à celles obtenues pour les PTS correspondants. En revanche, la possibilité d'exprimer des motifs et des règles de réécriture dans les termes augmente sensiblement la capacité de représentation du calcul, et devrait donc permettre de spécifier des systèmes formels complexes de façon plus lisible et efficace.

Pour la seconde approche nous avons suivi la démarche inverse : nous sommes partis d'un système de déduction pour lequel nous avons cherché des termes de preuve appropriés, et pour lesquels une forme limitée de filtrage s'avère utile. Il en découle un système de types pour un fragment du ρ-calcul, que nous pourrons étudier en tant que tel pour voir s'il correspond à un paradigme de programmation intéressant.

Dans ce nouveau système de types le filtrage correspond à une vision unifiée des règles d'introduction, alors que l'application d'une fonction à un terme correspond à l'élimination du symbole de prédicat qui se trouve en tête de l'argument. Nous sommes donc probablement plus proches d'un isomorphisme de Curry-Howard-de Bruijn étendu, au sens où les mécanismes supplémentaires (par rapport au λ-calcul) correspondent précisément à un concept propre à la logique dont nous représentons les démonstrations.

Conclusion et perspectives

Nous avons étudié différents systèmes de types pour le calcul de réécriture, tous conçus comme des extensions conservatives de systèmes de types pour le λ-calcul. Ceci nous a permis de mieux comprendre l'incidence du filtrage sur un calcul typé. On notera en particulier que la formulation simplement typée et la formulation avec Π-abstractions, qui sont équivalentes dans le λ-calcul, ne le sont pas ici. Nous avons ensuite établi les bases pour l'utilisation du calcul de réécriture dans le domaine de la logique, en proposant deux formalismes voisins de la déduction modulo pour lesquels un langage de termes de preuve avec motifs est utile.

Typage

Nous avons montré que les versions avec types simples et polymorphisme vérifient la plupart des propriétés usuelles : préservation du type, unicité et décidabilité du typage. Par contre, nous avons vu également qu'elles autorisent la définition de points fixes, ce qui nous a conduit à les étudier dans l'optique de la représentation de programmes.

Nous avons donc exploré l'inférence de types pour la version polymorphe du calcul, qui s'avère demander des restrictions très proches de celles faites dans ML, et confirme donc que ce langage atteint les limites de la décidabilité du typage.

Ensuite, nous nous sommes intéressés plus particulièrement à l'encodage de systèmes de réécriture et des stratégies guidant leur application. Ceci nous a conduit à définir une réduction étendue qui détecte certains échecs de filtrage. Nous avons démontré la correction et la confluence de cette réduction, et nous avons détaillé comment elle permettait de représenter l'action d'un système de réécriture par un ρ-terme typé.

En vue d'obtenir un calcul typé normalisant, nous nous sommes alors tournés vers les P^2TS, une famille de ρ-calculs avec types dépendants. Tout comme pour les types simples, la plupart des propriétés usuelles ont pu être démontrées par des techniques similaires à celles utilisées pour le λ-calcul. Nous avons également obtenu la normalisation forte pour deux systèmes de cette famille, par le biais d'une traduction vers le système $\lambda\omega$. La terminaison d'un calcul et la consistance d'un système de déduction étant des propriétés très liées, ce résultat légitime l'usage des P^2TS comme langage de termes de preuve pour un système de déduction.

Logique

Partant du résultat établi pour les P^2TS, nous avons cherché à les utiliser dans un contexte déductif, de la même manière que le λ-calcul est utilisé pour représenter les démonstrations en

déduction naturelle. Le filtrage et la possibilité de représenter des règles de réécriture suggérait naturellement de s'intéresser à la déduction naturelle modulo, une extension de la logique du premier ordre dans laquelle la réécriture joue un rôle important.

Nous avons établi comment utiliser un sous-langage des P^2TS comme langage de termes de preuve pour la déduction modulo, où chaque application d'une règle de réécriture est retranscrite dans le terme de preuve. La représentation linéaire des démonstrations ainsi obtenue est plus riche, dans la mesure où elle favorise la vérification des démonstrations. Par contre, elle ne constitue pas un isomorphisme de Curry-Howard-de Bruijn complet, au sens où l'élimination des coupures n'est pas totalement retranscrite par les réductions des termes.

Nous avons ensuite proposé un second système de déduction, qui est une variante de la déduction modulo dans laquelle les règles de réécriture sur les propositions sont transformées en règles d'introduction et d'élimination qui généralisent celles de la logique du premier ordre. Nous avons montré l'équivalence de ce formalisme avec la déduction modulo, et nous avons caractérisé la notion de coupure liée aux nouvelles règles de déduction.

Pour obtenir une représentation linéaire des démonstrations, nous avons introduit un langage de termes de preuve avec motifs qui permet de représenter uniformément toutes les nouvelles règles de déduction, et dans lequel l'élimination d'une coupure se traduit par une forme de filtrage. Ce système de déduction naturelle généralisé peut donc être vu comme un autre système de types pour un fragment du ρ-calcul.

Perspectives

Pour poursuivre les travaux exposés dans ce manuscrit, plusieurs directions de recherche peuvent être explorées. Du côté des systèmes de types, il reste à démontrer la normalisation forte dans les autres systèmes du ρ-cube. Indépendamment de cela, il serait intéressant de considérer des types conjonctions pour les structures ; enfin le système de types induit par la déduction naturelle généralisée mérite une étude à part entière.

Normalisation forte des autres systèmes du ρ-cube

Nous avons démontré la normalisation dans $\rho_{\rightarrow}$ et ρP, et il reste donc six systèmes à traiter dans le ρ-cube. Il n'est pas évident que la technique de démonstration utilisée s'étende aux autres systèmes, dans la mesure où la forme des types y est beaucoup moins contrainte. Rappelons d'ailleurs que, dans le λ-calcul, la normalisation forte de $\lambda 2$ et des systèmes qui l'étendent a également nécessité la conception d'une nouvelle technique de démonstration, à savoir les candidats de réductibilité.

Une piste de travail serait de bénéficier de la traduction de $\rho_{\rightarrow}$ en λ-calcul typé pour trouver une interprétation des ρ-types simples, et de s'appuyer sur celle-ci pour construire des candidats de réductibilité appropriés à $\rho 2$.

Types conjonctions pour les structures

Que ce soit dans le système simplement typé ou dans les P^2TS, nous avons utilisé une règle de typage relativement restrictive pour les structures, dans la mesure où elle impose que tous les membres d'une même structure aient le même type. Il est vraisemblable que l'utilisation de types conjonctions et d'une notion de sous-typage permettrait de lever cette restriction.

L'utilisation de types conjonctions pour le ρ-calcul est un problème difficile, dans la mesure où, même dans les systèmes de types à la P^2TS, une relation de sous-typage trop souple invalide très rapidement la normalisation forte des termes typés.

Système de types induit par la déduction naturelle généralisée

Les P^2TS ont d'abord été étudiés comme un système de types puis comme un langage de termes de preuve ; pour la déduction naturelle généralisée, nous pourrions suivre la démarche inverse, et étudier les propriétés du système de types que nous avons défini. Il semble qu'on ne puisse y démontrer l'absurde par une démonstration en forme normale, et par conséquent, établir des conditions de normalisation forte permettrait de démontrer la cohérence des logiques correspondantes.

Concernant les systèmes logiques présentés au chapitre 6, nous pourrions modifier la sémantique opérationnelle des ρ-termes de preuve pour la déduction modulo, afin qu'elle rende compte exactement de l'élimination des coupures. Notre proposition de déduction naturelle généralisée peut également être améliorée, ne serait-ce qu'en décomposant totalement les formules lors du calcul des nouvelles règles ; par ailleurs, nous pourrions y traiter des exemples plus conséquents que ceux vus jusqu'ici, pour guider les futures améliorations à apporter.

Réductions supplémentaires dans les termes de preuve pour la déduction modulo

Nous avons remarqué que l'élimination des coupures en déduction modulo n'était pas totalement retranscrite dans les ρ-termes de preuve que nous avons défini. Il serait donc intéressant de définir de nouvelles réductions sur les ρ-termes afin d'obtenir un isomorphisme de Curry-Howard-de Bruijn complet. Le calcul serait alors pourvu d'une nouvelle sémantique opérationnelle qui pourrait faire l'objet d'une étude à part entière.

Extension de la déduction naturelle généralisée aux connecteurs disjonctifs

Lors de la construction des nouvelles règles d'introduction et d'élimination, nous n'avons pas décomposé les formules sous les connecteurs $\vee$ et $\exists$, sans quoi l'équivalence avec la logique du premier ordre n'est pas valide. Il n'est pas exclu qu'on puisse traiter ces connecteurs également (et donc totalement décomposer une formule en propositions atomiques), à condition de trouver la décomposition adéquate.

Dans le même ordre d'idée, on pourrait étudier un calcul des séquents généralisé, où on considère de nouvelles règles gauches et droites sur les prédicats définis par réécriture. La décomposition des formules y poserait vraisemblablement moins de problèmes.

Représentation de logiques dans la déduction naturelle généralisée

Un moyen d'affirmer l'intérêt et la généralité de notre système de déduction serait d'y formaliser d'autres logiques, comme nous l'avons ébauché pour la logique d'ordre supérieur ou pour les prédicats inductifs. Les logiques modales, par exemple, fournissent une famille de systèmes déductifs dont la représentation pourrait être très informative.

Pour finir, l'implantation d'un ou plusieurs prototypes constituerait un complément appréciable à cette thèse. Le développement d'un logiciel basé sur l'un ou l'autre des systèmes de déduction présentés au chapitre 6 permettrait en effet générer plus facilement et plus sûrement des exemples à grande échelle.

Implantation d'un assistant utilisant des ρ-termes de preuve

Afin de justifier par la pratique l'utilité des ρ-termes de preuve que nous avons proposés pour la déduction modulo, il serait intéressant d'implanter un assistant à la démonstration

qui manipule effectivement cette représentation linéaire des démonstrations. Nous pourrions alors comparer le noyau de vérification des démonstrations de ce logiciel avec différentes expérimentations autour de l'introduction de la réécriture dans Coq par exemple.

Implantation de la déduction naturelle généralisée

Pour l'instant, nous n'avons vu que des exemples jouets dans notre système de déduction naturelle généralisée. Une implantation permettrait d'y développer des démonstrations à plus grande échelle, et donc de mieux cerner en pratique les intérêts et les limites de ce formalisme. Il serait également intéressant de le faire tester par des utilisateurs peu familiers avec la démonstration interactive, afin de mesurer si le formalisme correspond mieux à l'intuition mathématique, comme nous l'avons allégué.

Table des figures

Index

Bibliographie

[AHK77] K. Appel, W. Haken et J. Koch, « Every planar map is four colorable », *Illinois Journal of Mathematics*, vol. 21, 1977, p. 429 – 567.

[AHMP92] A. Avron, F. Honsell, I. Mason et R. Pollack, « Using typed lambda calculus to implement formal systems on a machine », *Journal of Automated Reasoning*, vol. 9, n° 3, 1992, p. 309–354.

[And92] J.-M. Andreoli, « Logic programming with focusing proofs in linear logic », *Journal of Logic and Computation*, vol. 2, n° 3, 1992, p. 297–347.

[Aut05] S. Autexier, « The CoRe calculus », dans R. Nieuwenhuis, éditeur, *Proceedings of the 20th International Conference on Automated Deduction*, coll. « To appear in LNCS », 2005.

[Bar84] H. Barendregt, *The Lambda-Calculus, its syntax and semantics*, coll. « Studies in Logic and the Foundation of Mathematics ». Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1984. Second edition.

[Bar90] F. Barbanera, « Adding algebraic rewriting to the calculus of constructions : strong normalization preserved », dans S. Kaplan et M. Okada, éditeurs, *Proc. of the 2nd Int. Workshop on Conditional and Typed Rewriting Systems*, vol. 516 (coll. *Lecture Notes in Computer Science*), p. 260–271. Springer, 1990.

[Bar92] H. P. Barendregt, « Lambda calculi with types », dans S. Abramsky, D. Gabbay et T. Maibaum, éditeurs, *Handbook of Logic in Computer Science*. Clarendon Press, 1992.

[BB02] H. Barendregt et E. Barendsen, « Autarkic computations in formal proofs », *Journal of Automated Reasoning*, vol. 28, n° 3, avril 2002, p. 321–336.

[BBCK05] C. Bertolissi, P. Baldan, H. Cirstea et C. Kirchner, « A rewriting calculus for cyclic higher-order term graphs », dans M. Fernandez, éditeur, *Proceedings of the 2nd International Workshop on Term Graph Rewriting*, vol. 127(5) (coll. *Electronic Notes in Theoretical Computer Science*), p. 21–41, Roma (Italy), septembre 2005.

[BCD+00] P. Borovanský, H. Cirstea, H. Dubois, C. Kirchner, H. Kirchner, P.-E. Moreau, C. Ringeissen et M. Vittek, ELAN *V 3.4 User Manual.* LORIA, Nancy (France), fourth édition, janvier 2000.

[BCDC83] H. Barendregt, M. Coppo et M. Dezani-Ciancaglini, « A filter lambda model and the completeness of typa assignment », *Journal of Symbolic Logic*, vol. 48, n° 4, 1983, p. 931–940.

[BCK05] C. Bertolissi, H. Cirstea et C. Kirchner, « A rewriting calculus semantics of combinatory reduction systems », *Higher-order and Symbolic Computation*, 2005. To appear.

[BCKL03] G. Barthe, H. Cirstea, C. Kirchner et L. Liquori, « Pure Patterns Type Systems », dans *Principles of Programming Languages - POPL2003, New Orleans, USA*. ACM, janvier 2003.

[Ben93] L. S. van Benthem Jutting, « Typing in pure type systems », *Information and Computation*, vol. 105, n° 1, 1993, p. 30–41.

[Ber88] S. Berardi, « Towards a mathematical analysis of the coquand-huet calculus of constructions and the other systems in barendregt's cube ». Rapport technique, Dept. Computer Science, Carnegie Mellon University and Dipartimento Matematica, Universita di Torino, Italy, 1988.

[Ber05] C. Bertolissi, *A rewriting calculus semantics for higher-order rewriting and graph rewriting*. Thèse de doctorat, Institut National Polytechnique de Lorraine, 2005.

[Bla04] F. Blanqui, « A type-based termination criterion for dependently-typed higher-order rewrite systems », dans V. van Oostrom, éditeur, *Proceedings of the 15th Conference on Rewriting Techniques and Applications*, vol. 3091 (coll. *Lecture Notes in Computer Science*), p. 24–39. Springer, 2004.

[Bla05a] F. Blanqui, « Decidability of type-checking in the calculus of algebraic constructions with size annotations », dans L. Ong, éditeur, *Proceedings of CSL'05*, coll. « Lecture Notes in Computer Science ». Springer, 2005. To appear.

[Bla05b] F. Blanqui, « Definitions by rewriting in the calculus of constructions », *Mathematical Structures in Computer Science*, vol. 15, n° 1, 2005, p. 37–92.

[Bla05c] F. Blanqui, « Inductive types in the calculus of algebraic constructions », *Fundamenta Informaticae*, vol. 65, n° 1–2, 2005, p. 61–86.

[BN98] F. Baader et T. Nipkow, *Term Rewriting and all That*. Cambridge University Press, 1998.

[Bor98] P. Borovanský, *Le contrôle de la réécriture : étude et implantation d'un formalisme de stratégies*. Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, France, octobre 1998. also TR LORIA 98-T-326.

[Bru70] N. de Bruijn, « The mathematical language AUTOMATH, its usage, and some of its extensions », dans S. Verlag, éditeur, *Symposium on Automatic Demonstration*, vol. 125 (coll. *Lecture Notes in Mathematics*), p. 29 – 61, Versailles, 1970.

[Bru80] N. G. de Bruijn, « A survey of the project AUTOMATH », dans J. P. Seldin et J. R. Hindley, éditeurs, *Essays on Combinatory Logic, Lambda Calculus, and Formalism*, p. 589–606. Academic Press inc., New York (NY, USA), 1980.

[BT88] V. Breazu-Tannen, « Combining algebra and higher-order types », dans *Proceedings 3rd IEEE Symposium on Logic in Computer Science, Edinburgh (UK)*, p. 82–90, 1988.

[Cam04] Projet Cristal, INRIA, Rocquencourt, France, *The* Caml *language*, 2004. `http://caml.inria.fr`.

[CDE+03] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer et C. Talcott, « The maude 2.0 system », dans R. Nieuwenhuis, éditeur, *Rewriting Techniques and Applications (RTA 2003)*, n° 2706, coll. « Lecture Notes in Computer Science », p. 76–87. Springer, June 2003.

[CF58] H. B. Curry et Feys, *Combinatory Logic*, vol. 1. Elsevier Science Publishers B. V. (North-Holland), Amsterdam, 1958.

[CFK05] H. Cirstea, G. Faure et C. Kirchner, « A rho-calculus of explicit constraint application », dans N. Martí-Oliet, M. Clavel et A. Verdejo, éditeurs, *Proceedings of the 5th workshop on rewriting logic and applications*, vol. 117 (coll. *Electronic Notes in Theoretical Computer Science*), p. 51–67, 2005.

[CH88] Th. Coquand et G. Huet, « The calculus of constructions », *Information and Computation*, vol. 76, 1988, p. 95 – 120.

[CHL96] P.-L. Curien, T. Hardin et J.-J. Lévy, « Confluence properties of weak and strong calculi of explicit substitutions », *Journal of the ACM*, vol. 43, n° 2, mars 1996, p. 362–397.

[Chu40] A. Church, « A formulation of the simple theory of types », *Journal of Symbolic Logic*, vol. 5, 1940, p. 56–68.

[Chu33] A. Church, « A set of postulates for the foundation of logics », *Annals of Mathematics*, vol. 33 and 34, 1932/33, p. 346–366 and 839–864.

[Cir00] H. Cirstea, *Calcul de réécriture : fondements et applications.* Thèse de Doctorat d'Université, Université Henri Poincaré - Nancy I, 2000.

[CK01] H. Cirstea et C. Kirchner, « The rewriting calculus — Part I *and* II », *Logic Journal of the Interest Group in Pure and Applied Logics*, vol. 9, n° 3, mai 2001, p. 427–498.

[CK04] S. Cerrito et D. Kesner, « Pattern matching as cut elimination », *Theoretical Computer Science*, vol. 323, n° 1–3, 2004, p. 71–127.

[CKL01] H. Cirstea, C. Kirchner et L. Liquori, « The Rho Cube », dans F. Honsell, éditeur, *Foundations of Software Science and Computation Structures*, coll. « Lecture Notes in Computer Science », p. 166–180, Genova, Italy, avril 2001.

[CKLW03] H. Cirstea, C. Kirchner, L. Liquori et B. Wack, « Rewrite strategies in the rewriting calculus », dans B. Gramlich et S. Lucas, éditeurs, *Third International Workshop on Reduction Strategies in Rewriting and Programming, WRS'03*, vol. 86(4) (coll. *Electronic Notes in Theoretical Computer Science*), p. 593–624, Valencia, Spain, juin 2003.

[CLW03] H. Cirstea, L. Liquori et B. Wack, « Rewriting calculus with fixpoints : Untyped and first-order systems », dans S. Berardi, M. Coppo et F. Damiani, éditeurs, *International Workshop on Types for Proofs and Programs, TYPES 2003*, vol. 3085 (coll. *Lecture Notes in Computer Science*), p. 147–161, Torino, Italy, 2003. Springer.

[Cop85] M. Coppo, « A completeness theorem for recursively defined types », dans W. Brauer, éditeur, *Proceedings of the 12th Colloquium on Automata, Languages and Programming*, vol. 194 (coll. *Lecture Notes in Computer Science*), p. 120–129. Springer, 1985.

[Coq85] T. Coquand, *Une théorie des constructions.* Thèse de doctorat, Université Paris VII, janvier 1985.

[Coq04] Projet LogiCal, INRIA, Rocquencourt, France, *The* Coq *proof assistant*, 2004. `http://coq.inria.fr`.

[CP90] T. Coquand et C. Paulin, « Inductively defined types », dans P. Martin-Löf et G. Mints, éditeurs, *COLOG-88. Proceedings of International Conference on Computer Logic, Tallinn, Estonia*, vol. 417 (coll. *Lecture Notes in Computer Science*), p. 50–66. Springer-Verlag, 1990.

[Cur34] H. B. CURRY, « Functionality in combinatory logic », *Proceedings of the National Academy of Sciences of the USA*, vol. 20, n° 11, novembre 1934, p. 584–590.

[DHK03] G. DOWEK, T. HARDIN et C. KIRCHNER, « Theorem proving modulo », *Journal of Automated Reasoning*, vol. 31, n° 1, Nov 2003, p. 33–72.

[DM82] L. DAMAS et R. MILNER, « Principal Type-Schemes for Functional Programs », dans *Proc. of POPL*, p. 207–212. The ACM Press, 1982.

[Dow97] G. DOWEK, « Proof normalization for a first-order formulation of higher-order logic », dans E. GUNTER et A. FELTY, éditeurs, *Proceedinds of* Theorem proving in higher order logics 1997, vol. 1275 (coll. *Lecture Notes in Computer Science*), p. 105–119. Springer-Verlag, 1997.

[Dow01] G. DOWEK, « About folding-unfolding cuts and cuts modulo », *Journal of Logic and Computation*, vol. 11, n° 3, 2001, p. 419–429.

[DW03] G. DOWEK et B. WERNER, « Proof normalization modulo », *Journal of Symbolic Logic*, vol. 68, n° 4, 2003, p. 1289–1316.

[DW05] G. DOWEK et B. WERNER, « Arithmetic as a theory modulo », dans J. GIESL, éditeur, *Proceedings of RTA'05*, vol. 3467 (coll. *Lecture Notes in Computer Science*), p. 423–437. Springer, 2005.

[Fau05] G. FAURE. « Rhomcal : an implementation in tom of the explicit rho-calculus ». `http://www.loria.fr/~faure/RhomCal/`, 2005.

[Fer93] M. FERNÁNDEZ, *Modèles de calcul multiparadigmes fondés sur la réécriture.* Thèse de doctorat, Université Paris Sud, 1993.

[FK02] G. FAURE et C. KIRCHNER, « Exceptions in the rewriting calculus », dans S. TISON, éditeur, *Proceedings of RTA'2002*, vol. 2378 (coll. *Lecture Notes in Computer Science*), p. 66–82, Utrecht (The Netherlands), juillet 2002. Springer-Verlag.

[FM05] G. FAURE et A. MIQUEL. « Towards a denotational semantics for the ρ-calculus ». To be published, 2005.

[FN96] K. FUTATSUGI et A. NAKAGAWA, « An Overview of Cafe Project », dans *Proceedings of First CafeOBJ Workshop*, Yokohama (Japan), août 1996.

[For02] J. FOREST, « A weak calculus with explicit operators for pattern matching and substitution », dans S. TISON, éditeur, *Proceedings of RTA'02*, vol. 2378 (coll. *Lecture Notes in Computer Science*), p. 174–191, Copenhagen, Danemark, 2002. Springer.

[For03] J. FOREST, *Réécriture d'ordre supérieur avec motifs.* Thèse de doctorat, Université Paris XI, 2003.

[GBT89] J. GALLIER et V. BREAZU-TANNEN, « Polymorphic rewriting conserves algebraic strong normalization and confluence », dans *16th Colloquium Automata, Languages and Programming*, vol. 372 (coll. *Lecture Notes in Computer Science*), p. 137–150. Springer-Verlag, 1989.

[Geu93] H. GEUVERS, *Logics and type systems.* PhD thesis, Nijmegen University, 1993.

[Gir72] J.-Y. GIRARD, *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur.* Thèse de doctorat, Université Paris VII, juin 1972.

[GLT89] J.-Y. GIRARD, Y. LAFONT et P. TAYLOR, *Proofs and Types*, vol. 7 (coll. *Cambridge Tracts in Theoretical Computer Science*). Cambridge University Press, 1989.

[GN91] H. Geuvers et M. Nederhof, « A Modular Proof of Strong Normalization for the Calculus of Constructions », *Journal of Functional Programming*, vol. 1, n° 2, 1991, p. 155–189.

[Gon05] G. Gonthier, « A computer-checked proof of the four colour theorem ». Research Report, Microsoft Research Cambridge, 2005. Available at `http://research.microsoft.com/~gonthier/4colproof.pdf`.

[HHP93] R. Harper, F. Honsell et G. Plotkin, « A framework for defining logics », *Journal of the ACM*, 1993.

[Hou05] C. Houtmann, « Theory-plugged ρ-calculus ». Rapport de stage, ENS Cachan, 2005.

[How80] W. Howard, « The formulas–as–types notion of construction », dans J. P. Seldin et J. R. Hindley, éditeurs, *Essays on Combinatory Logic, Lambda Calculus, and Formalism*, p. 479–490. Academic Press inc., New York (NY, USA), 1980.

[Hua94] X. Huang, « Reconstructing proofs at the assertion level », dans A. Bundy, éditeur, *12th International Conference on Automated Deduction*, vol. 814 (coll. *Lecture Notes in Computer Science*), p. 738–752. Springer, 1994.

[Hue76] G. Huet, *Résolution d'equations dans les langages d'ordre 1,2, ...,ω*. Thèse de Doctorat d'Etat, Université de Paris 7 (France), 1976.

[Hue80] G. Huet, « Confluent reductions : Abstract properties and applications to term rewriting systems », *Journal of the ACM*, vol. 27, n° 4, octobre 1980, p. 797–821. Preliminary version in 18th Symposium on Foundations of Computer Science, IEEE, 1977.

[Jay04] B. Jay, « The pattern calculus », *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 26, n° 6, 2004, p. 911–937.

[JO95] J.-P. Jouannaud et M. Okada, « Abstract data type systems ». Rapport technique, LRI and Keio University, avril 1995.

[JR99] J.-P. Jouannaud et A. Rubio, « The higher-order recursive path ordering », dans G. Longo, éditeur, *Proceedings of Logic in Computer Science*, p. 402–411. IEEE Computer Society, 1999.

[KB70] D. E. Knuth et P. B. Bendix, « Simple word problems in universal algebras », dans J. Leech, éditeur, *Computational Problems in Abstract Algebra*, p. 263–297. Pergamon Press, Oxford, 1970.

[Kha90] Z. Khasidashvili, « Expression reduction systems », dans *Proceedings of I. Vekua Institute of Applied Mathematics*, vol. 36, p. 200–220, 1990.

[KK90] C. Kirchner et F. Klay, « Syntactic theories and unification », dans *Proceedings 5th IEEE Symposium on Logic in Computer Science, Philadelphia (Pa., USA)*, p. 270–277, juin 1990.

[KK99] C. Kirchner et H. Kirchner. « Rewriting, solving, proving ». A preliminary version of a book available at `www.loria.fr/~ckirchne/rsp.ps.gz`, 1999.

[KKN02] C. Kirchner, H. Kirchner et Q.-H. Nguyen, « External rewriting for skeptical proof assistants », *Journal of Automated Reasoning*, vol. 29, n° 3-4, 2002, p. 309–336.

[KKV95] C. Kirchner, H. Kirchner et M. Vittek, « Designing CLP using Computational Systems », dans P. Van Hentenryck et S. Saraswat, éditeurs, *Principles and Practice of Constraint Programming.* The MIT press, 1995.

[KLN03] F. KAMAREDDINE, T. LAAN et R. NEDERPELT, « Revisiting the notion of function », *Journal of Logic and Algebraic Programming*, vol. 54, n° 1–2, 2003, p. 65–107.

[Klo80] J. W. KLOP, *Combinatory Reduction Systems*. Thèse de doctorat, CWI, 1980.

[KN96] F. KAMAREDDINE et R. NEDERPELT, « Canonical typing and π-conversion in the Barendregt cube », *Journal of Functional Programming*, vol. 6, n° 2, 1996, p. 245 – 267.

[KOR93] J. KLOP, V. VAN OOSTROM et F. VAN RAAMSDONK, « Combinatory reduction systems : introduction and survey », *Theoretical Computer Science*, vol. 121, 1993, p. 279–308.

[KPT96] D. KESNER, L. PUEL et V. TANNEN, « A typed pattern calculus », *Information and Computation*, vol. 124, n° 1, 10 janvier 1996, p. 32–61.

[Liq05] L. LIQUORI. « Formalisation du λcalcul par valeurs dans ρp ». Communication personnelle, 2005.

[Loa03] R. LOADER, « Higher order β matching is undecidable », *Logic Journal of the IGPL*, vol. 11, n° 1, 2003, p. 51–68.

[LS04] L. LIQUORI et B. SERPETTE, « irho : an imperative rewriting calculus », dans E. MOGGI et D. S. WARREN, éditeurs, *Proceedings of the 6th International ACM SIGPLAN Conference on Principle and Practice of Declarative Programming*, p. 167–178. ACM, 2004.

[LW05] L. LIQUORI et B. WACK, « The polymorphic rewriting calculus : Type checking *vs.* type inference », dans N. MARTÍ-OLIET, M. CLAVEL et A. VERDEJO, éditeurs, *Fifth International Workshop on Rewriting Logic and its Applications, WRLA'04*, vol. 117 (coll. *Electronic Notes in Theoretical Computer Science*), p. 89–111, Barcelona, 2005.

[Men87] N. P. MENDLER, *Inductive Definition in Type Theory*. PhD thesis, Cornell University, Ithaca, USA, 1987.

[Mil78] R. MILNER, « A theory of type polymorphism in programming », *Journal of Computer and System Sciences*, vol. 17, n° 3, 1978, p. 348–375.

[Mil91] D. MILLER, « A logic programming language with lambda-abstraction, function variables, and simple unification », *Journal of Logic and Computation*, vol. 1, n° 4, 1991, p. 497–536.

[New42] M. H. A. NEWMAN, « On theories with a combinatorial definition of equivalence », dans *Annals of Math*, vol. 43, p. 223–243, 1942.

[Nip91] T. NIPKOW, « Combining matching algorithms : The regular case », *Journal of Symbolic Computation*, 1991, p. 633–653.

[Oka89] M. OKADA, « Strong normalizability for the combined system of the typed Lambda-calculus and an arbitrary convergent term rewrite system », dans *Proceedings of the ACM-SIGSAM 1989 International Symposium on Symbolic and Algebraic Computation, Portland (Oregon)*, p. 357–363. ACM Press, juillet 1989. Report CRIN 89-R-220.

[Oos90] V. VAN OOSTROM, « Lambda calculus with patterns ». Technical report, Vrije Universiteit, Amsterdam, novembre 1990.

[Pad00] V. PADOVANI, « Decidability of fourth-order matching », *Mathematical Structures in Computer Science*, vol. 3, n° 10, juin 2000, p. 361–372.

[Plo75] G. D. PLOTKIN, « Call-by-name, call-by-value, and the λ-calculus », *Theoretical Computer Science*, vol. 1, 1975, p. 125–159.

[Pot02] F. POTTIER. « Cours de DEA : Typage et programmation », 2002. `http://pauillac.inria.fr/~fpottier/mpri/dea-typage.ps.gz`.

[Pra65] D. PRAWITZ, *Natural Deduction, a Proof-theoretical Study*. 1965.

[Rey74] J. C. REYNOLDS, « Towards a theory of type structure », dans B. ROBINET, éditeur, *Symposium on Programming*, vol. 19 (coll. *Lecture Notes in Computer Science*), p. 408–423. Springer, 1974.

[Rob65] J. A. ROBINSON, « A machine-oriented logic based on the resolution principle », *Journal of the ACM*, vol. 12, 1965, p. 23–41.

[Ros73] B. K. ROSEN, « Tree-manipulating systems and Church-Rosser theorems », *Journal of the ACM*, vol. 20, n° 1, 1973, p. 160–187.

[SS04] C. SCHÜRMANN et A. STUMP, « Logical semantics for the rewriting calculus », dans M. BONACINA et T. BOY DE LA TOUR, éditeurs, *Fifth Workshop on Strategies in Automated Deduction*, vol. 125(2) (coll. *Electronic Notes in Theoretical Computer Science*), p. 149–164, 2004.

[SU98] M. H. SØRENSEN et P. URZYCZYN, « Lectures on the curry-howard isomorphism ». Lecture Notes n° 98/14, DIKU, Copenhagen, 1998.

[Ter89] J. TERLOUW, « Een nadere bewijstheoretische analyse van GSTT's ». Manuscrit, Faculty of Mathematics and Computer Science, University of Nijmegen, Netherlands, avril 1989.

[Ter02] TERESE, *Term Rewriting Systems*. Cambridge University Press, 2002. M. Bezem, J. W. Klop and R. de Vrijer, eds.

[Toy87] Y. TOYAMA, « On the Church-Rosser property for the direct sum of term rewritig systens », *Journal of the ACM*, vol. 34, n° 1, janvier 1987, p. 128–143.

[Vit94] M. VITTEK, ELAN *: Un cadre logique pour le prototypage de langages de programmation avec contraintes*. Thèse de Doctorat d'Université, Université Henri Poincaré – Nancy 1, octobre 1994.

[Wac04] B. WACK, « The simply-typed pure pattern type system ensures strong normalization », dans J.-J. LÉVY, E. MAYR et J. MITCHELL, éditeurs, *Third International Conference on Theoretical Computer Science, IFIP TCS'04*, p. 633 – 646, Toulouse, France, August 2004. IFIP, Kluwer Academic Publishers.

[WC03] D. WALUKIEWICZ-CHRZĄSZCZ, « Termination of rewriting in the calculus of constructions », *Journal of Functional Programming*, vol. 13, n° 2, 2003, p. 339–414.

[Wel99] J. B. WELLS, « Typability and type checking in system F are equivalent and undecidable », *Annals of Pure and Applied Logic*, vol. 98, n° 1–3, 1999, p. 111—156.

[Wer94] B. WERNER, *Une Théorie des Constructions Inductives*. Thèse de doctorat, Université Paris 7, 1994.

Résumé

Le calcul de réécriture est un formalisme qui intègre les mécanismes fonctionnels du lambda-calcul et les capacités de filtrage de la réécriture. Cette thèse est consacrée à l'étude de systèmes de types pour ce calcul, et à son utilisation dans le domaine de la déduction.

Nous étudions les propriétés et les applications de deux paradigmes de typage. Le premier est inspiré du lambda-calcul simplement typé, mais il en diffère par le fait qu'un terme peut être typé sans pour autant que ses réductions soient terminantes. Nous nous focalisons donc sur l'utilisation de cette version typée pour la modélisation de programmes, notamment la représentation de systèmes de réécriture avec une stratégie de réduction.

La seconde famille de systèmes de types que nous étudions est adaptée des Pure Type Systems, et comporte des types dépendants. Nous en démontrons la normalisation forte (qui légitime l'utilisation du langage en logique) par le biais d'une traduction vers un lambda-calcul avec types dépendant de types.

Enfin nous proposons deux approches pour l'utilisation du calcul de réécriture en logique, plus particulièrement en lien avec la déduction modulo. La première approche est une utilisation des systèmes avec types dépendants pour définir des termes de preuve pour la déduction modulo. Dans la seconde, nous définissons une généralisation de la déduction naturelle et nous montrons qu'une forme limitée de filtrage est utile pour représenter les règles de ce système de déduction.

Mots-clés: Lambda-calcul, réécriture, filtrage, typage, déduction, principe de Poincaré.

Abstract

The rewriting calculus is a formalism integrating functional mechanisms from the lambda-calculus and matching capabilities from rewriting. This thesis is devoted to the study of type systems for this calculus, and to its applications to the domain of deduction.

We study the properties and the applications of two typing paradigms. The first one is inspired by the simply typed lambda-calculus, but it differs from it in the sense that even a well-typed term may have a non-terminating reduction. Thus, we focus on the use of this typed version for modeling programs, and especially for representing rewriting systems.

The second family of type systems we study is adapted from the Pure Type Systems, and features dependent types. We show the strong normalization (which legitimates the use of the language in logic) in two of these systems, via a translation into a lambda-calculus with types depending on types.

Finally, we propose two ways of using the rewriting calculus in logic, more particularly in relation with deduction modulo. In the first approach, we use the systems with dependent types to define proof terms for deduction modulo. In the second case, we define a generalization of natural deduction and we show that a restricted form of matching is useful in order to represent the rules of this deduction system.

Keywords: Lambda-calculus, rewriting, pattern matching, type systems, deduction, Poincaré principle.