Aspects typés du calcul de réécriture

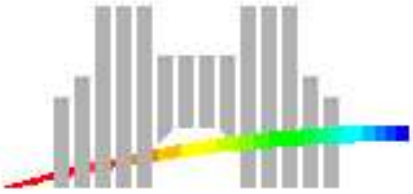
Benjamin Wack

Aspects typés du calcul de réécriture

Rapport de stage
DEA d'Informatique Fondamentale
École Normale Supérieure de Lyon

Benjamin Wack
Encadrants : Claude Kirchner, Horatiu Cirstea, Luigi Liquori

4 février - 3 juillet 2002



Résumé

Le calcul de réécriture, ou ρ-calcul, proposé dans la thèse d'H. Cirstea, intègre dans un même système le λ-calcul et la réécriture. Pour cela, les abstractions ne se font plus seulement sur des variables mais sur des motifs ; une règle de réécriture devient alors un terme à part entière, et peut être manipulée en tant que citoyen de première classe. On profite ainsi à la fois des mécanismes d'ordre supérieur du λ-calcul et de l'expressivité du filtrage de la réécriture.

La puissance du filtrage peut être adaptée au moyen d'une théorie arbitraire ; en réécriture classique, il en résulterait un certain non-déterminisme, mais le ρ-calcul est pourvu d'un mécanisme approprié qui permet de maintenir une liste de résultats possibles. La complexité des divers éléments du calcul font qu'il n'est pas confluent et donc qu'il faut définir des stratégies, ou des restrictions, pour assurer la confluence.

Nous nous intéressons ici au ρ-calcul en tant qu'extension du λ-calcul et nous en étudions les aspects typés, l'objectif à terme étant de mieux comprendre les relations entre ce calcul et les systèmes logiques, et ainsi d'étendre l'isomorphisme de Curry-Howard.

Notre étude se base sur une généralisation du cube de Barendregt, où les deux abstracteurs λ et Π sont unifiés en un seul. Il nous faut aussi tenir compte des nouveaux éléments du ρ-calcul : puissance du filtrage, non-déterminisme, problèmes de confluence.

Sous les bonnes conditions, nous parvenons ainsi à prouver la plupart des propriétés habituelles des calculs typés : lemme de substitution, correction, préservation du type, consistance. Cependant, l'unicité du type n'est généralement plus valable, notamment à cause de l'unification des abstracteurs. Nous décrivons le nombre de types distincts que peut avoir un terme donné, et nous montrons enfin que l'unicité du typage reste valide dans les systèmes $\rho\rightarrow$ et $\rho 2$.

Mots-clés : Calcul de réécriture, réécriture, lambda calcul, théorie des types, cube de Barendregt, unicité du type.

Table des matières

Table des définitions

Introduction

En informatique, la réécriture fournit à la fois une base théorique très générale, en permettant par exemple de spécifier le comportement d'un programme, et des applications pratiques très variées, notamment lorsqu'elle est utilisée comme un paradigme de programmation. Ainsi, la réécriture apparaît dans les règles d'inférence décrivant une logique, et peut être utilisée comme base d'un prouveur de théorèmes. On peut également donner une sémantique rigoureuse à un langage, puis utiliser la réécriture pour garantir certaines propriétés des programmes, ce qui rejoint la notion de logiciels sûrs. La réécriture est tout aussi répandue au niveau de l'implémentation proprement dite. Elle peut constituer le cœur d'un langage et donc être utilisée de façon implicite par le programmateur, comme dans Mathematica. À l'inverse, elle fait souvent partie intégrante du langage par le biais des motifs et du filtrage, comme dans Prolog ou ML.

Le calcul de réécriture (ou ρ-calcul) a été proposé par C. Kirchner et H. Cirstea, la thèse de ce dernier [Cir00] décrivant en détail les fondements et les propriétés de base. Ce calcul unifie le λ-calcul et la réécriture pour profiter de l'expressivité de ces deux formalismes : le filtrage est utilisable très largement, et de plus une règle de réécriture devient un objet de première classe, à son tour manipulable par le calcul lui-même. Ici, c'est plutôt les aspects hérités du λ-calcul que nous allons étudier. En effet, pour assurer certaines propriétés, il faut définir un système de types, qui attribue à chaque terme bien formé un type qui décrit son comportement de façon statique.

Il est apparu assez vite que le λ-calcul typé avait également un sens en termes de logique : c'est le fameux isomorphisme de Curry-Howard, selon lequel un type est aussi une proposition, et un terme est aussi une preuve de la proposition correspondant à son type (voir [SU98] par exemple). Les différentes études de la littérature montrent que, selon le système de types choisi, on capture un fragment plus ou moins important de la logique. Comme le ρ-calcul ajoute de l'expressivité en incluant la réécriture, il est naturel de s'interroger sur la puissance de la logique qui lui correspond. La première étape de cette exploration consiste à étudier les propriétés des systèmes de types du ρ-calcul, et c'est là l'objet de ce rapport.

On notera que d'autres études ont déjà été faites à propos des interactions entre réécriture, λ-calcul et types. Th. Coquand a exploré les possibilités d'un filtrage avec des types dépendants [Coq92], plutôt dans une optique de modélisation des langages fonctionnels. S. Cerrito et D. Kesner ont proposé un calcul avec filtrage et substitutions explicites pour modéliser le calcul des séquents [CK99] ; la syntaxe et les règles de réduction en sont très fortement conditionnées par les règles de déduction de la logique intuitionniste. F. Blanqui a étudié des systèmes de types basés sur les *Pure Type Systems*, avec une relation de conversion *a priori* arbitraire, et notamment le cas où cette relation est donnée par un système de réécriture [Bla01]. L'originalité de notre approche est de partir du calcul de réécriture, déjà pourvu d'une sémantique opérationnelle, et de chercher dans quelle mesure il apporte une signification supplémentaire aux systèmes de types basés sur sa syntaxe.

Dans le chapitre 1, je commencerai par rappeler brièvement les principes des deux paradigmes qui sont à l'origine du ρ-calcul. D'une part, le λ-calcul [Chu41] permet de modéliser la plupart des langages fonctionnels, où les fonctions sont elles-mêmes des objets du calcul, et à ce titre peuvent par exemple être données comme argument d'une fonction. D'autre part, la réécriture [Klo92] fournit un mécanisme puissant pour transformer divers objets selon des règles très générales.

Je décrirai ensuite le ρ-calcul, avec sa syntaxe et ses règles d'évaluation. J'expliquerai comment ajuster la puissance du filtrage au moyen d'une théorie arbitraire ; je montrerai le non-déterminisme qui en découle ainsi que les mécanismes proposés par le ρ-calcul pour traiter ce non-déterminisme sans perte d'information. Enfin je montrerai sur des exemples les divers aspects couverts par le ρ-calcul, notamment la possibilité d'encodage du λ-calcul, et les différences avec la réécriture.

Pour finir cette description opérationnelle du calcul, je donnerai quelques résultats autour d'une propriété fondamentale du ρ-calcul : il n'est pas confluent dans sa version la plus générale. Les raisons de cette non-confluence seront décrites au moyen d'exemples, et plusieurs moyens de retrouver la confluence seront envisagés, suivant les propositions de [Cir01, BCKL02].

Dans le second chapitre, je m'attacherai à la construction de systèmes de types pour le ρ-calcul. Je commencerai par rappeler les principes du λ-cube de Barendregt [Bar92], qui décrit de façon uniforme 8 systèmes de types pour le λ-calcul. Je donnerai les principales propriétés de ces systèmes, qui sont les bonnes propriétés qu'on peut raisonnablement attendre de tout calcul typé.

Suivant la présentation de [CKL01], je montrerai ensuite comment on peut construire une extension du λ-cube qui tient compte de tous les nouveaux éléments du ρ-calcul, et qu'on appellera naturellement le ρ-cube. Avec le même formalisme, on peut ajouter un neuvième système inspiré du calcul des constructions étendu [Luo90], qui tire sa puissance d'une hiérarchie infinie dénombrable de sortes (alors qu'il n'y en a que deux dans le cube classique). Nous verrons par la suite que ce système se révèle indispensable pour décrire le comportement du ρ-calcul typé.

Au cours de mes travaux, je me suis aperçu que le calcul typé n'était pas utilisable en l'état. J'ai donc proposé deux aménagements fondamentaux pour rendre les neuf systèmes de types utilisables, que je décrirai et motiverai. Le premier consiste à imposer une contrainte supplémentaire sur les substitutions, afin que le filtrage se comporte bien relativement aux types des termes manipulés. Le second est plus subtil : on verra que, pour que le typage ait un sens, il faut que le calcul soit confluent. Or les restrictions faites pour obtenir la confluence du ρ-calcul peuvent perturber gravement les jugements de typage ; j'ai donc mis au point une règle d'inférence modifiée pour avoir toutes les propriétés voulues, *indépendemment* de la stratégie confluente choisie.

Le troisième et dernier chapitre est dédié aux lemmes que j'ai pu prouver à propos des systèmes de types du ρ-cube, en suivant les grandes lignes de Barendregt. Je commence donc par montrer que les substitutions préservent certaines relations : l'équivalence modulo la réduction, le jugement de typage. Je suis alors en mesure de montrer les propriétés les plus importantes : correction, préservation du type, consistance. Par rapport aux preuves classiques du λ-calcul, il faut souvent rechercher plus d'informations dans les dérivations de typage, afin de tenir compte des nouveaux éléments du calcul. On verra que l'on ne peut démontrer que des versions faibles de ces théorèmes, à savoir que les conclusions se font dans un système de types plus puissant que les hypothèses.

Je finis cette étude en abordant le problème de l'unicité du typage. En effet, à cause de l'identification des abstracteurs λ et Π, cette propriété du λ-calcul est ici mise en défaut. Je montre donc qu'on peut construire des termes ayant un nombre de types quelconque (non borné), mais que pour un terme donné le nombre de types possibles est fini. Je termine en montrant l'unicité du type dans le système $\rho 2$ (le calcul polymorphe), qui peut être vu comme un modèle étendu de ML ; ce résultat reste valable dans $\rho\!\rightarrow$ (le calcul simplement typé) en tant que sous-système du précédent.

Chapitre 1

Le calcul de réécriture

Ce chapitre pose les bases du ρ-calcul, en donnant sa définition comme continuation de deux autres formalismes : le λ-calcul et la réécriture. On verra qu'il faut imposer des contraintes supplémentaires sur le calcul pour qu'il soit confluent.

1.1 Les fondements

1.1.0 Préliminaires : quelques définitions

Je rappelle ici des notions de base qui apparaîtront fréquemment au cours de ce rapport : quelques propriétés des relations binaires, les notions de système d'inférence et de substitution simultanée.

Définition 1 (Relations dérivées)
Étant donnée une relation binaire $\mapsto$ *, on définit :*

- *la relation inverse de* $\mapsto$ *, notée* $\leftarrow\!\!\shortmid$ *, avec* $x \leftarrow\!\!\shortmid y$ *si et seulement si* $y \mapsto x$ *;*
- *la* fermeture transitive *de* $\mapsto$ *, notée* $\mapsto^+$ *, est la plus petite relation transitive contenant* $\mapsto$ *;*
- *la* fermeture réflexive et transitive *de* $\mapsto$ *est notée* $\mapsto^*$ *;*
- *la fermeture réflexive, symétrique et transitive de* $\mapsto$ *, notée* $\leftrightarrow^*$ *ou encore* $=$*, est la plus petite relation transitive contenant* $\mapsto^*$ *et* ${}^*\!\leftarrow\!\!\shortmid$ *.*

Définition 2 (Terminaison)
Pour une relation $\mapsto$ *, un terme* t *est dit* réductible *par* $\mapsto$ *s'il existe un terme* t' *tel que* $t \mapsto t'$*. Dans le cas contraire, il est* irréductible.

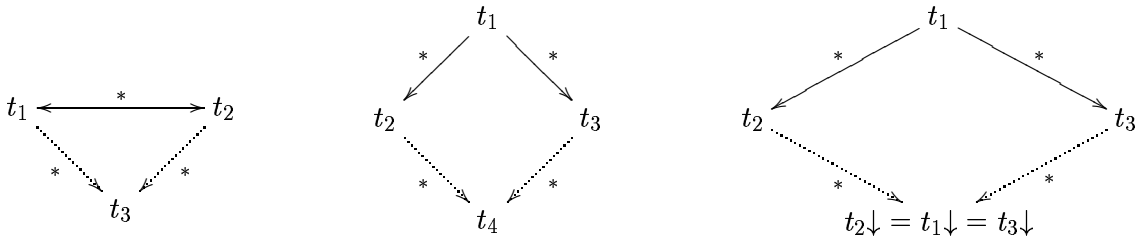
On appelle forme normale *de* t *tout terme* t' *irréductible tel que* $t \mapsto^* t'$*. Lorsque un terme* t *a une unique forme normale, celle-ci est notée* $t\!\downarrow$*.*

La relation $\mapsto$ *est dite* faiblement normalisante *si tout terme* t *admet (au moins) une forme normale ; elle est* fortement normalisante *s'il n'y a pas de suite infinie de réductions (toute réduction partant d'un terme donné aboutit à une forme normale).*

Définition 3 (Propriétés de confluence)

1. $\mapsto$ *a la propriété de* Church-Rosser *si :* $t_1 \leftrightarrow^* t_2 \Rightarrow \exists t_3,\ t_1 \mapsto^* t_3$ *et* $t_2 \mapsto^* t_3$
2. $\mapsto$ *est* confluente *si :* $t_1 \mapsto^* t_2$ *et* $t_1 \mapsto^* t_3 \Rightarrow \exists t_4,\ t_2 \mapsto^* t_4$ *et* $t_3 \mapsto^* t_4$
3. $\mapsto$ *est* convergente *si* $\mapsto$ *est fortement normalisante et a la propriété de Church-Rosser.*

Ces différentes définitions se représentent chacune par un diagramme :



Church-Rosser *confluence* *convergence*

On peut prouver facilement que la confluence et la propriété de Church-Rosser sont équivalentes pour toute relation binaire $\mapsto$: un sens de l'implication est évident ; pour l'autre il suffit d'effectuer une récurrence sur le nombre d'occurrences de $\mapsto^*$ et $^* \leftarrow$ dans la relation $\leftrightarrow^*$.

Ces propriétés sont importantes dans la mesure où l'on peut vouloir décider de l'égalité $t \leftrightarrow^* t'$. La propriété de Church-Rosser permet alors de se contenter de chercher un terme t'' auquel on peut aboutir par réductions à partir de t et t'. Si en plus la normalisation forte est vérifiée, il suffit de réduire t et t' à leurs formes normales, qui seront en fait identiques si $t \leftrightarrow^* t'$, et fourniront donc le témoin t'' recherché.

Pour décrire les mécanismes de typage, on utilisera la notion de *système d'inférence*. Informellement, un système d'inférence est constitué d'un ensemble de *règles* du genre :

$$\frac{\text{Prémisse 1} \quad \text{Prémisse 2} \quad \ldots}{\text{Conclusion}} \quad (\textit{Étiquette})$$

ainsi que d'*axiomes*, souvent écrits comme des règles sans prémisses.

Une règle se lit de la façon suivante : si toutes les prémisses sont vérifiées, alors la conclusion est vérifiée. Il faut remarquer que les règles comportent généralement des métavariables, et sont donc en fait des *schémas de preuve*. L'étiquette ne joue aucun autre rôle que de permettre une description plus succincte de la règle.

Un jugement est alors dit *dérivable* dans le système d'inférence que l'on considère, si on peut construire un arbre dont la racine (située tout en bas) est le jugement à dériver, les feuilles sont des axiomes, et les nœuds correspondent à des règles dont les métavariables sont correctement instanciées.

Définition 4 (Substitution)
Une substitution (généralement notée σ) est une application d'un ensemble (fini) de variables vers l'ensemble des termes. On note $\mathcal{D}om\ \sigma$ son domaine, et $\mathcal{R}an\ \sigma$ son codomaine, i.e. l'ensemble des termes images $\{\sigma x \mid x \in \mathcal{D}om\ \sigma\}$.

Elle est en fait définie sur l'ensemble des termes, en posant $\sigma t = t'$ où t' est le terme t dans lequel toutes les variables du domaine de σ ont été remplacées par leur image. C'est donc une notion de substitution simultanée*, autrement dit une occurrence d'une variable z apparaissant dans un certain σx ne sera pas affectée par σ.*

Si σ est donnée explicitement, on pourra la noter $[x_1/t_1 \ldots x_n/t_n]$, avec $\mathcal{D}om\ \sigma = \{x_1, \ldots, x_n\}$ et $\sigma x_i = t_i$. Dans tous les cas, on notera son application de manière préfixe.

Définition 5 (Contexte)
Pour une notion donnée de terme, un contexte $\mathsf{Ctx}[\ ]$ *est un "terme avec un trou", c'est-à-dire que, pour tout terme t, $\mathsf{Ctx}[t]$ est lui-même un terme. On peut évidemment généraliser cette notion avec plusieurs "trous".*

1.1.1 Le λ-calcul

Introduit par A. Church [Chu41] et très répandu depuis, le λ-calcul est un formalisme extrêmement simple qui permet, entre autres, de modéliser les langages fonctionnels. Il se caractérise par sa capacité à effectuer des calculs d'ordre supérieur, c'est-à-dire à manipuler une fonction comme n'importe quel objet, et donc à définir des fonctions sur les fonctions (comme par exemple la composition).

Définition 6 (λ-calcul)
On définit le λ-calcul par sa syntaxe et par la règle de β-réduction :

$$\begin{array}{lllll} \textit{Variables} & \mathcal{V}ar & ::= & x, y, z \ldots \\ \textit{Termes} & \Lambda & ::= & \mathcal{V}ar \mid \lambda \mathcal{V}ar.\Lambda \mid \Lambda\Lambda \\ (\beta - red) & (\lambda x.M)N & \mapsto_\beta & [x/N]M \end{array}$$

Ici, l'ensemble de variables $\mathcal{V}ar$ est a priori infini dénombrable. On remarque comme caractéristique que l'abstraction (représentée par le symbole λ) se fait sur une variable, mais que tout terme M peut être passé en argument à un autre terme N par l'application MN.

Rappelons que le λ-calcul est confluent et que, si on lui ajoute un système de types convenable, il est également fortement normalisant (voir définitions 2 et 3). Ces propriétés en font un paradigme de calcul réellement effectif, puisqu'elles garantissent le déterminisme du calcul et donnent un moyen de s'assurer de la terminaison d'un "programme".

Je ne rappellerai pas ici toutes les subtilités du λ-calcul, mais il convient de s'attarder un peu pour voir le problème que peut représenter la substitution qui apparaît lors de la β-réduction. Elle est généralement considérée comme un mécanisme externe au calcul, qui ne va donc pas perturber la structure du terme auquel elle s'applique. On distingue pour cela, dans un même terme, deux types de variables :

Définition 7 (Variables libres et liées)

$$\begin{array}{lrcl} \textit{Variables libres} & FV(x) & \triangleq & \{x\} \\ & FV(\lambda x.M) & \triangleq & FV(M)\backslash\{x\} \\ & FV(MN) & \triangleq & FV(M) \cup FV(N) \\ \textit{Variables liées} & BV(x) & \triangleq & \emptyset \\ & BV(\lambda x.M) & \triangleq & BV(M) \cup \{x\} \\ & BV(MN) & \triangleq & BV(M) \cup BV(N) \end{array}$$

On remarque immédiatement que toute variable d'un terme est, soit libre, soit liée : $FV(M) \cup BV(M) = \mathcal{V}ar(M)$. Un terme est dit *clos* s'il ne comporte aucune variable libre. Le nom exact des variables liées n'est pas vraiment important puisqu'il ne sert qu'à dénoter quelle variable est liée par quel abstracteur. Lorsqu'on effectue une substitution, on veut donc qu'elle ne concerne que les variables libres. Il existe trois moyens standards de s'en assurer :

L'α-conversion : afin de rendre réellement opérationnel cette "anonymat" des variables liées, on ajoute au calcul une relation $\mapsto_\alpha$ qui consiste à remplacer, dans un terme, toutes les occurrences d'une variable donnée x par une variable y dite "fraîche". L'application d'une substitution se fait alors modulo $=_\alpha$, ce qui est réellement intéressant dans le cas de l'abstraction :

$$[x/N](\lambda x.M) \mapsto_\alpha [x/N](\lambda y.[x/y]M) = (\lambda y.[x/y]M) \; {}_\alpha\!\leftarrow\!\!\!\shortmid \; \lambda x.M$$

En raisonnant modulo la fermeture symétrique réflexive transitive $=_\alpha$, une variable liée pourra donc toujours se voir attribuer un nom distinct de ceux concernés par la substitution.

La convention d'hygiène de Barendregt : on suppose tout simplement que, dans tout terme, une même variable ne peut pas être liée à un endroit et utilisée hors du champ de son premier lieur. Avec cette approche, on fait plus de suppositions d'ordre supérieur, mais on reste plus près de la définition syntaxique du calcul. En effet, on ne parle plus ici de substitution, mais on s'appuie sur la forme du terme avant réduction. On évite ainsi les phénomènes de *capture* :

$$(\lambda x.(\lambda y.x))y \ \mapsto_\beta \ (\lambda y.y)$$

Dans cet exemple, on trouve l'identité après réduction, alors qu'initialement les deux occurrences de la variable y n'avaient aucun rapport. Avec l'hygiène, le problème aurait été évité car le terme initial aurait dû s'écrire $(\lambda x.(\lambda z.x))y$.

Il faut toutefois faire attention aux duplications qui peuvent apparaître lors des réductions :

$$(\lambda x.xx)(\lambda y.y) \ \mapsto_\beta \ (\lambda y.y)(\lambda y.y)$$

Ici, bien que le terme de départ ait respecté la convention, le terme réduit devrait s'écrire $(\lambda y.y)(\lambda z.z)$ pour être correct. Il faut donc parfois procéder à un renommage en cours de route.

Les indices de de Bruijn : proposé dans [Bru72], ce formalisme n'utilise plus de noms de variables, mais les remplace simplement par des entiers qui permettent de repérer à quel abstracteur elles correspondent, l'entier en question donnant le niveau de profondeur qui le sépare de son abstracteur. Donnons deux exemples de traduction :

$$\begin{array}{lcr} \lambda x.\lambda y.(xyz) & \rightsquigarrow & \lambda\lambda(213) \\ \lambda x.(x(\lambda y.(xy))) & \rightsquigarrow & \lambda(1\lambda(21)) \end{array}$$

Ainsi, une substitution consiste simplement à remplacer un entier donné par le terme voulu, en incrémentant cet entier à chaque fois qu'on entre dans une abstraction supplémentaire.

Nous verrons par la suite que le ρ-calcul pose le même genre de problèmes. Nous supposerons donc de manière générale que la convention de Barendregt est respectée, car l'emploi d'indices à la de Bruijn risquerait d'être très complexe. Remarquons que, dans sa thèse [Cir00], H. Cirstea a proposé une version du ρ-calcul avec substitutions explicites, ce qui nous donne une certaine légitimité pour l'emploi "implicite" de l'α-conversion.

1.1.2 Les systèmes de réécriture

Une idée centrale de la réécriture [Klo92] est d'imposer une direction dans l'utilisation d'équations en définissant des règles de réécriture ; on peut également la voir comme une description formelle de l'ensemble des transformations qu'on désire appliquer à un objet pour le mettre sous une forme voulue.

Définition 8 (Système de réécriture)
Une règle de réécriture *est une paire de termes orientée, noté $l \to r$, où l est le membre gauche de la règle et r son membre droit.*

Un système de réécriture *sur les termes est un ensemble de règles de réécriture.*

Deux conditions sont imposées habituellement sur la construction des règles de réécriture :

1. le membre gauche d'une règle de réécriture n'est pas une variable ($\forall x \in \mathcal{V}ar,\ l \neq x$) ;
2. l'ensemble des variables du membre droit est inclus dans l'ensemble des variables du membre gauche ($\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$).

L'ensemble des variables d'une règle $l \to r$, noté $\mathcal{V}ar(l \to r)$, est défini par $\mathcal{V}ar(l) \cup \mathcal{V}ar(r)$ et si la condition précédente est satisfaite alors $\mathcal{V}ar(l \to r) = \mathcal{V}ar(l)$.

Une règle de réécriture est *linéaire à gauche* si son membre gauche est linéaire. Un système de réécriture est linéaire à gauche si toutes ses règles le sont.

Définition 9 (Relation de réécriture)
La relation de réécriture $\mapsto_R$ associée à un système de réécriture R est définie par : $t \mapsto_R t'$ s'il existe un contexte $\mathsf{Ctx}[\,]$ (du langage des termes), une règle $l \to r$ dans R et une substitution σ telles que $t = \mathsf{Ctx}[\sigma l]$ et $t' = \mathsf{Ctx}[\sigma r]$.

Un système de réécriture R est dit convergent (resp. est confluent, termine) si la relation de réécriture $\mapsto_R$ est convergente (resp. est confluente, termine).

En ajoutant des conditions sur l'application des règles de réécriture, les systèmes de réécriture sont naturellement étendus à des systèmes de réécriture *conditionnels*. Plusieurs définitions de ces systèmes ont été proposées, et la différence essentielle est l'interprétation des conditions.

Un système de réécriture *conditionnel naturel* (natural conditional rewriting system) a des règles de réécriture de la forme :

$$l \to r \ \ \mathsf{si} \ \ s_1 \leftrightarrow^* t_1 \wedge \ldots \wedge s_n \leftrightarrow^* t_n$$

où les prédicats d'égalité $s_i \leftrightarrow^* t_i$ sont appelées les conditions de la règle.

Le problème de décision de $\leftrightarrow^*$ peut alors être arbitrairement complexe. Pour obtenir un système plus raisonnable, on peut choisir des conditions plus restrictives portant sur les formes normales, qui seront équivalentes dans le cas d'un système de réécriture convergent :

Un système de réécriture *conditionnel standard* (standard (join) conditional rewriting system) a des règles de réécriture de la forme :

$$l \to r \ \ \mathsf{si} \ \ s_1 \downarrow = t_1 \downarrow \ \wedge \ldots \wedge \ s_n \downarrow = t_n \downarrow$$

Dans le cas du ρ-calcul, comme nous voulons décrire un maximum de contraintes au niveau du calcul, et non pas au niveau méta, nous n'utiliserons pas de conditions. Cependant, il restera possible d'injecter une notion d'égalité externe au calcul, au moyen d'une théorie *a priori* arbitraire.

Pour finir cette présentation de la réécriture, comme notre objectif est de dégager le sens logique du ρ-calcul, nous devons parler de la *logique de réécriture*, introduite par J. Meseguer [Mes92, MOM93]. Son principe est de donner une sémantique aux systèmes de réécriture conditionnels.

Sans rentrer dans les détails, qui sont assez techniques, à un terme t est associée une classe d'équivalence $[t]_{\mathcal{E}}$, modulo un certain ensemble $\mathcal{E}$ d'axiomes équationnels sur les termes. Les formules de la logique sont définies comme des *séquents* de la forme suivante :

$$\pi : [t]_{\mathcal{E}} \to [t']_{\mathcal{E}}$$

Le terme de preuve π représente alors une "méthode" pour dériver les termes de la classe d'équivalence $[t']_{\mathcal{E}}$ à partir des termes de la classe d'équivalence $[t]_{\mathcal{E}}$.

Les *règles de déduction* permettant de construire les séquents et les termes de preuve associés sont alors les suivantes :

- des règles de *réflexivité* et de *transitivité*, qui sont indispensables à ce genre de logique ;
- une règle de *congruence* afin de tenir compte des symboles de fonctions n-aires présents dans les termes ;
- une règle de *remplacement* pour chaque règle de réécriture $l \to r$ (étiquetée ℓ). Ce sont celles-ci qui donnent véritablement son sens à la logique, en l'associant à un système de réécriture R. En effet, elles imposent que
  $\ell(\pi_1, \dots, \pi_n) : [l(t_1, \dots, t_n)]\varepsilon \to [r(t'_1, \dots, t'_n)]\varepsilon \quad \textbf{si} \quad \pi_1 : [t_1]\varepsilon \to [t'_1]\varepsilon, \dots, \pi_n : [t_n]\varepsilon \to [t'_n]\varepsilon$

Il apparaît donc que la logique de réécriture de Meseguer est un formalisme assez *ad hoc*, dont le sens hors de la réécriture n'est pas évident au premier abord. Notamment, on constate que le système de déduction ainsi créé dépend beaucoup du système de réécriture R, et qu'il est facile de le rendre inconsistant en choisissant des règles $l \to r$ appropriées.

Notre approche, qui s'inspire plus des λ-calculs typés, devrait nous permettre, par la suite, de dégager des conclusions logiques plus proches des systèmes de déduction existants, voire d'établir un isomorphisme de Curry-Howard pour une logique plus étendue.

1.2 Les principes du ρ-calcul

1.2.1 Composantes de base

Depuis la thèse de H. Cirstea, différentes présentations du calcul ont été proposées ; dans tout ce document, nous adopterons le formalisme présenté dans [CKL01]. En effet, il se prête mieux à l'étude des types, l'aspect du ρ-calcul qui nous intéresse ici.

Nous utiliserons généralement les majuscules A, B, C, D... pour parler de termes du ρ-calcul. Cependant, quand on considérera un ρ-terme obtenu à partir d'un λ-terme, on pourra utiliser M, N, etc. De même, un ρ-terme issu de la réécriture sera parfois écrit $l \to r$ s'il n'y a pas d'ambiguïté. L'égalité syntaxique de deux termes sera notée $\equiv$.

Commençons par donner une définition formelle de la syntaxe du ρ-calcul, que nous expliciterons par la suite :

Sortes	$\mathcal{S}$	::=	$* \mid \Box \mid \Box_1 \mid \cdots \mid \Box_n \mid \dots$
Variables	$\mathcal{V}ar$	::=	$X, Y, Z, \dots, X_1, X_2, \dots, X_n \dots$
Constantes	$\mathcal{C}st$	::=	$a, b, c, \dots, f, g, h \dots$
Termes	$\mathcal{T}$	::=	$\mathcal{S} \mid \mathcal{V}ar^{\mathcal{T}} \mid \mathcal{C}st^{\mathcal{T}} \mid \mathcal{T} \to \mathcal{T} \mid \mathcal{T}\bullet\mathcal{T} \mid null \mid \mathcal{T}, \mathcal{T}$

La signification de ces divers éléments est la suivante :

- Sortes : de même que pour le λ-calcul typé, ou plus généralement les *Pure Type Systems*, les sortes peuvent être vues comme des "types de types" ; on verra donc leur utilité dans le chapitre 2. Une sorte quelconque sera notée s, s', s_1, s_2 ....
- Variables : au sens habituel, on dispose d'un ensemble infini (dénombrable) de variables. On les notera généralement avec des lettres majuscules. Le grand changement par rapport aux calculs typés classiques est la présence dans X^A d'une décoration A qui représente le type de la variable X. Cette notation est assez proche d'un typage *à la Church*, mais on impose ici les décorations sur *toutes* les variables, et pas seulement celles qui sont liées. Les contextes de typage ne seront alors plus nécessaires, mais on doit ajouter une condition qui correspond à la cohérence des contextes (voir définition 10 plus loin).

- Constantes : elles se comportent à peu près comme les variables, si ce n'est qu'elles ne peuvent jamais être instanciées. De même, les types des constantes sont indiqués dans la décoration et doivent respecter la convention de décoration 10. On dénotera généralement par a, b, c... les symboles 0-aires, et par f, g, h ceux correspondant à des fonctions. Cependant, comme nous travaillons avec une syntaxe très proche de la currification, les symboles utilisés n'ont pas d'arité à proprement parler. Enfin, quand on voudra parler d'un terme qui peut être soit une variable, soit une constante, on le notera par une lettre grecque α, β ....
- Abstraction $\rightarrow$: on a choisi ici la même notation que pour la flèche de réécriture, car un terme de la forme $l \rightarrow r$ correspond effectivement à la fonction qui prend un argument de la forme l et renvoie r après instanciation correcte des variables. Nous verrons également comment cet abstracteur permet d'encoder le λ-calcul. Dans un cadre typé, cet abstracteur $\rightarrow$ réalisera l'unification des abstracteurs λ et Π.
- Application $\bullet$: ce symbole est utilisé pour noter l'application d'un terme à un autre (ce qui, en λ-calcul, se fait simplement par juxtaposition des deux termes). Il est utilisé également pour les symboles de fonctions, de sorte que $f \bullet X \bullet Y$ représente effectivement l'application du symbole constant f aux variables X et Y, au lieu de l'habituel $f(X, Y)$.
- Structures : les deux derniers constituants syntaxiques du ρ-calcul sont assez originaux. En effet, on s'autorise une structure vide (notée *null*) ou bien composée de plusieurs éléments et construite avec l'opérateur ",". Selon les propriétés de cette virgule, la structure peut être une liste, un arbre binaire, un ensemble ou un multi-ensemble. Nous verrons que cette structure nous donne des moyens appropriés pour traiter l'échec et le non-déterminisme qui peuvent apparaître en calcul de réécriture.

Voyons la convention nécessaire sur les types des variables et des constantes. Elle correspond exactement à vérifier qu'une variable donnée n'apparaît qu'une seule fois dans le contexte, ce qu'on fait en λ-calcul typé.

Définition 10 (Convention de décoration)
Pour toutes occurrences α^A et α^B d'une même variable ou constante α apparaissant dans un terme (ou dans une même inférence de type), les types attribués à α sont cohérents : $A \equiv B$.

Pour alléger les notations, lorsqu'il n'y a pas d'ambiguïté, l'ordre de priorité des opérateurs sera "$\bullet$", "$\rightarrow$" et "," (le plus prioritaire étant $\bullet$). Similairement au λ-calcul, l'opérateur d'application $\bullet$ est associatif à gauche, et l'opérateur d'abstraction $\rightarrow$ est associatif à droite.

Pour compléter ce tour d'horizon, il nous faut préciser comment certaines notions s'adaptent à notre calcul :

Variables libres et liées : ici, c'est l'abstracteur $\rightarrow$ qui lie les variables. Il faut tenir compte du fait qu'on peut trouver un *motif*, et plus seulement une variable, à gauche de ce lieur, mais le principe reste le même. Par contre, les décorations des variables et des constantes ne doivent pas être liées par $\rightarrow$: en effet, dans un λ-calcul à la Church, le type des variables liées est considéré comme faisant partie du contexte, et pas du terme lui-même. On peut résumer de la manière suivante :

Définition 11 (Adaptation de FV et BV pour des variables décorées)

1. $FV^{\downarrow}(a^A) \triangleq \emptyset$ $\quad$ $FV^{\downarrow}(X^A) \triangleq \{X\}$
   $FV^{\downarrow}(A \bullet B) \triangleq FV^{\downarrow}(A, B) \triangleq FV^{\downarrow}(A) \cup FV^{\downarrow}(B)$
   $FV^{\downarrow}(A \rightarrow B) \triangleq FV^{\downarrow}(B) \setminus FV^{\downarrow}(A)$

2. $FV^{\uparrow}(a^A) \triangleq FV^{\uparrow}(X^A) \triangleq FV(A)$
   $FV^{\uparrow}(A \bullet B) \triangleq FV^{\uparrow}(A, B) \triangleq FV^{\uparrow}(A) \cup FV^{\uparrow}(B)$
   $FV^{\uparrow}(A \to B) \triangleq (FV^{\uparrow}(B) \setminus FV^{\downarrow}(A)) \cup (FV^{\uparrow}(A) \setminus FV^{\downarrow}(A))$

3. $FV(null) \triangleq FV(*) \triangleq FV(\Box_i) \triangleq \emptyset$
   $FV(A) \triangleq FV^{\downarrow}(A) \cup FV^{\uparrow}(A)$

4. $BV(null) \triangleq BV(*) \triangleq BV(\Box_i) \triangleq \emptyset$
   $BV(a^A) \triangleq BV(X^A) \triangleq BV(A)$
   $BV(A \bullet B) \triangleq BV(A, B) \triangleq BV(A) \cup BV(B)$
   $BV(A \to B) \triangleq BV(A) \cup BV(B) \cup FV^{\downarrow}(A)$

Une comparaison détaillée avec les notions de variable libre ou liée dans le λ-calcul permet de s'assurer de la correction de ces définitions. On peut d'ailleurs vérifier rigoureusement les conditions suivantes : pour tout terme A, on a $FV(A) \cup BV(A) = \mathcal{V}ar(A)$ et aussi $FV(A) \cap BV(A) = \emptyset$.

Conventions sur les variables : comme déjà mentionné dans la section 1.1.1, nous supposerons systématiquement que la convention d'hygiène de Barendregt est respectée.

Application d'une substitution : ayant supposé toutes les conventions nécessaires, nous considérerons qu'une substitution s'applique sans problème : il n'y a jamais de capture et les variables liées ne sont pas concernées par une substitution. Cependant, on n'oubliera pas de propager la substitution jusque dans les décorations de type si c'est nécessaire :

$$\sigma\ Y^A \triangleq \begin{cases} Y^{\sigma A} & \text{si} \quad Y \notin \mathcal{D}om\ \sigma \\ B & \text{si} \quad \sigma = [\ldots Y/B \ldots] \end{cases} \quad \begin{array}{l} \\ \text{(si une variable } Z^C \text{ apparaît dans } B, \\ \text{elle n'est pas affectée par } \sigma, \text{ voir déf. 4)} \end{array}$$

1.2.2 Évaluation d'un ρ-terme

La syntaxe des termes du ρ-calcul étant fixée, définissons précisément ses règles d'évaluation. Comme nous introduisons une notion de filtrage afin de disposer de l'expressivité de la réécriture, il nous faut déjà définir l'égalité selon laquelle nous allons comparer deux termes. De manière générale, nous pouvons considérer une égalité définie dans une théorie arbitraire $\mathbb{T}$, avec les contraintes suivantes.

Définition 12 (Théorie de filtrage)
A priori toute théorie consistante est utilisable dans ce cadre, et on notera les jugements d'égalité :

$$\Vdash A \overset{\mathbb{T}}{=} B$$

Pour pouvoir l'utiliser en pratique, on supposera cependant que l'égalité ainsi définie est une congruence, ce qu'on vérifie grâce aux règles d'inférence suivantes :

$$\frac{}{\Vdash A \overset{\mathbb{T}}{=} A}\ (Réflexivité) \qquad \frac{\Vdash A \overset{\mathbb{T}}{=} B \quad \Vdash B \overset{\mathbb{T}}{=} C}{\Vdash A \overset{\mathbb{T}}{=} C}\ (Transitivité)$$

$$\frac{\Vdash A \overset{\mathbb{T}}{=} B}{\Vdash B \overset{\mathbb{T}}{=} A}\ (Symétrie) \qquad \frac{\Vdash A \overset{\mathbb{T}}{=} B}{\Vdash \mathsf{Ctx}[A] \overset{\mathbb{T}}{=} \mathsf{Ctx}[B]}\ (Contexte)$$

Les classes d'équivalence correspondantes peuvent donc être des singletons ou des ensembles quelconques, éventuellement infinis. En pratique, les théories utilisées seront inspirées de l'algèbre :

- $\mathbb{T}_c^f$: La théorie de la commutativité ajoute la règle $\Vdash f(A\ B) \overset{\mathbb{T}_c^f}{=} f(B\ A)$
- $\mathbb{T}_a^f$: La théorie de l'associativité ajoute la règle $\Vdash f(f(A\ B)\ C) \overset{\mathbb{T}_a^f}{=} f(A\ f(B\ C))$
- $\mathbb{T}_i^f$: La théorie de l'idempotence ajoute la règle $\Vdash f(A\ A) \overset{\mathbb{T}_i^f}{=} A$

Dans ces trois exemples, la théorie porte sur un symbole f choisi parmi les constantes ; il est également possible de modifier le comportement du symbole de structure "," pour adapter le comportement du ρ-calcul. Ainsi, une virgule associative donne une structure de liste ordonnée ; si elle est aussi commutative on a un multi-ensemble ; si on lui ajoute l'idempotence, c'est un ensemble.

Munis de cette égalité, nous pouvons définir le mécanisme de filtrage que nous allons utiliser.

Définition 13 (Équation de filtrage et solution)
Étant donnée une théorie équationnelle $\mathbb{T}$ sur les termes :

1. *une* équation de filtrage *est de la forme $A \ll_{\mathbb{T}} B$;*
2. *une substitution σ est une* solution *de $A \ll_{\mathbb{T}} B$ si $\sigma A \overset{\mathbb{T}}{=} B$.*

On peut alors définir la fonction $\mathcal{S}ol$, des équations de filtrage vers les ensemble de substitutions :

1. Si toute substitution σ est solution du système S, il est dit *trivial*. Par convention, on considère que la substitution identité $\mathbb{ID}$ est l'unique solution du système : $\mathcal{S}ol(\mathsf{S}) \triangleq \{\mathbb{ID}\}$.
2. Si S n'admet aucune solution, $\mathcal{S}ol(\mathsf{S}) \triangleq \emptyset$.
3. Sinon, $\mathcal{S}ol(\mathsf{S}) \triangleq \{\sigma \mid \sigma \text{ solution de } \mathsf{S}\}$.

Notons immédiatement que, dans le cas de la théorie vide, on peut résoudre effectivement une équation de filtrage, et que la solution, si elle existe, est unique. Pour cela, on peut utiliser un algorithme dû à G. Huet [Hue76] qui se décrit facilement par un système de réécriture. On définit la *pseudo-substitution* $\mathbb{F}$, qui représente un échec de filtrage ; on écrit alors le système d'équations de filtrage sous la forme $\bigwedge_{i=1}^{n} A_i \ll_\emptyset B_i$, et on le soumet aux règles suivantes :

$$
\begin{array}{lllll}
(Decomposition) & A_1 \diamond_1 A_2 \ll_\emptyset B_1 \diamond_2 B_2 & \rightsquigarrow & \begin{cases} A_1 \ll_\emptyset B_1 \wedge A_2 \ll_\emptyset B_2 & \text{si } \diamond_1 \equiv \diamond_2 \\ \mathbb{F} & \text{sinon} \end{cases} \\
(Merging) & (X^C \ll_\emptyset A) \wedge (X^C \ll_\emptyset B) & \rightsquigarrow & \begin{cases} X^C \ll_\emptyset A & \text{si } \Vdash A \overset{\emptyset}{=} B \\ \mathbb{F} & \text{sinon} \end{cases} \\
(Clash) & A \ll_\emptyset X^B & \rightsquigarrow & \mathbb{F} \quad \text{si } A \notin \mathcal{V}ar \\
(Symbols) & f^A \ll_\emptyset g^B & \rightsquigarrow & \begin{cases} \mathbb{ID} \text{ si } f^A \equiv g^B \\ \mathbb{F} \quad \text{sinon} \end{cases} \\
(Propagation) & \mathbb{F} \wedge \mathsf{S} & \rightsquigarrow & \mathbb{F} \\
(Propagation') & \mathbb{ID} \wedge \mathsf{S} & \rightsquigarrow & \mathsf{S}
\end{array}
$$

FIG. 1.1 – Évaluation d'un filtrage syntaxique

Ici le symbole logique $\wedge$ est associatif et commutatif. Les lettres f et g représentent des constantes (d'arité nulle ou pas). Les jokers $\diamond_1$, $\diamond_2$ peuvent être au choix "," ou "•". En fait, en examinant cet algorithme, on se rend compte qu'il effectue une sorte de récurrence sur le terme de gauche, qui est le membre gauche d'une abstraction. C'est un filtrage purement syntaxique, et à ce titre on ne l'emploie pas sur des motifs contenant $\rightarrow$.

En effet, comparer deux abstractions de cette façon reviendrait à admettre que deux fonctions sont égales si et seulement si elles ont exactement la même formulation, le même texte. Cela constituerait évidemment une perte d'information sémantique importante. De plus, on pourrait obtenir des résultats trompeurs. En effet, le problème de filtrage $Y^* \rightarrow Z^* \ll_{\mathbb{T}} X^* \rightarrow X^*$ a bel et bien une solution : $\sigma = [Y^*/X^*, Z^*/X^*]$, qui est correcte syntaxiquement mais n'a aucun sens du point de vue des ρ-termes. Pour le traiter correctement, il faudrait adapter un algorithme de filtrage d'ordre supérieur [DHK00] afin qu'il prenne en compte les symboles de constantes.

On remarquera que la fin de l'exécution de l'algorithme correspond à l'obtention d'une forme normale, qui est soit $\mathbb{F}$, soit $\bigwedge_{X_j \in \mathcal{V}ar\{A_i\}} X_j \ll_\emptyset C_j$. Si c'est $\mathbb{F}$, alors le problème de filtrage n'a pas de solution ; sinon, après avoir éliminé toutes les équations de la forme $X^A \ll_\emptyset X^A$, on a directement l'unique substitution qui est solution du problème de filtrage. L'élément $\mathbb{ID}$ est défini comme neutre pour $\wedge$ par la règle (*Propagation'*) ; s'il ne reste que lui à la fin, toute substitution convient et le problème est donc trivial ; comme convenu, on pose $\{\mathbb{ID}\}$ comme solution.

Nous possédons à présent tous les outils nécessaires pour décrire les règles de réduction du ρ-calcul :

$$(A \rightarrow B)\bullet C \quad \mapsto_\rho \quad \begin{cases} \sigma_1 B, \ldots, \sigma_n B, \ldots & \text{où} \quad \mathcal{S}ol(A \ll_{\mathbb{T}} C) = \{\sigma_1, \ldots, \sigma_n \ldots\} \\ null & \text{si} \quad \mathcal{S}ol(A \ll_{\mathbb{T}} C) = \emptyset \end{cases}$$

$$(A, B)\bullet C \quad \mapsto_\delta \quad A\bullet C, B\bullet C$$

$$null\bullet C \quad \mapsto_\nu \quad null$$

FIG. 1.2 – Les règles de réduction du ρ-calcul

Ces règles sont celles qui s'appliquent en position de tête (à "top-level"), mais il faut bien voir qu'elles restent valables dans tout contexte : $\mathsf{Ctx}[A] \mapsto \mathsf{Ctx}[A']$ si $A \mapsto A'$. Commentons rapidement leur signification :

ρ-réductions : C'est ici qu'on introduit réellement la réécriture dans le calcul, puisque le filtrage intervient. Le nombre de solutions σ pouvant apparaître dépend largement de la théorie, et peut à priori être non borné, voire infini. Cependant, les structures nous permettent de conserver tous ces résultats à la fois, et donc de manipuler un comportement proche du non-déterminisme. Le cas où aucune substitution ne convient sera appelé *échec de filtrage*, et c'est le seul cas où le sous-terme *null* peut apparaître dans un terme qui ne le contenait pas auparavant.

δ-réduction : Cette règle correspond à la distribution d'une structure à gauche d'une application. Nous verrons dans les exemples suivants le sens qu'on peut donner à cette règle, notamment en ce qui concerne l'application de plusieurs règles de réécriture.

ν-réduction : Elle correspond exactement à la réduction δ, mais dans le cas où la structure considérée est vide. Elle ne fait que propager un *null* déjà existant. On remarquera qu'un *null* à droite d'une application ne se propage pas forcément.

Afin de familiariser le lecteur avec ce formalisme, donnons quelques exemples montrant les divers aspects du ρ-calcul. Les décorations de type ne jouant aucun rôle dans les réductions, nous les omettrons pour une meilleure lisibilité.

- Premièrement, voyons comment encoder le λ-calcul : il suffit de n'autoriser qu'une variable à gauche de l'abstracteur, et alors $X \to M$ correspond exactement à $\lambda x.M$. Comme les membres gauches sont des variables, les problèmes de filtrage qui se posent sont toujours de la forme $X \ll_\emptyset C$, dont la solution est évidente. Ainsi, pour les termes classiques IK et Ω, on a :

  $$[\![(\lambda x.x)(\lambda yz.y)]\!] \simeq (X \to X)\bullet(Y \to Z \to Y) \mapsto_\rho (Y \to Z \to Y) \simeq [\![\lambda yz.y]\!]$$
  $$[\![(\lambda x.xx)(\lambda x.xx)]\!] \simeq (X \to X\bullet X)\bullet(X \to X\bullet X) \mapsto_\rho (X \to X\bullet X)\bullet(X \to X\bullet X) \mapsto_\rho \ldots$$

- Voyons maintenant comment les constantes et les motifs ajoutent de la puissance de calcul par rapport au λ-calcul :

  $(f\bullet X \to g\bullet X)\bullet(f\bullet a) \quad \mapsto_\rho \quad g\bullet a$ *(On peut extraire un sous-terme de l'argument)*

  $(h\bullet X\bullet X \to X)\bullet(h\bullet b\bullet b) \quad \mapsto_\rho \quad b$ *(Le filtrage permet de comparer deux sous-termes)*

- La différence essentielle entre ρ-calcul et systèmes de réécriture est la suivante : un ρ-terme ne peut agir que localement, alors que les règles d'un système s'appliquent tant que c'est possible. Ainsi, dans un système de réécriture comportant la règle $a \to f(a)$, le terme a ne sera jamais en forme normale puisqu'il va se réduire indéfiniment en des termes de la forme $f^n(a)$. Inversement, en ρ-calcul, le terme qui représente une règle est consommé par la réduction, et donc on pourra avoir des formes normales :

  $(a \to f(a))\bullet a \mapsto_\rho f(a)\downarrow$ ou encore $(a \to a)\bullet((a \to a)\bullet a) \mapsto_\rho (a \to a)\bullet a \mapsto_\rho a\downarrow$

- Tout en restant dans la théorie vide, le symbole de structure permet d'introduire un certain non-déterminisme lié à la réécriture. Ainsi, on peut appliquer simultanément deux règles de réécriture distinctes :

  $(X \oplus Y \to X, Z \oplus b \to c)\bullet a \oplus b \quad \mapsto_\delta \quad (X \oplus Y \to X)\bullet a \oplus b, (Z \oplus b \to c)\bullet a \oplus b \quad \mapsto^*_\rho \quad a, c$

  Ou encore utiliser la solution d'une même équation de filtrage pour deux résultats distincts :

  $(X \oplus Y \to X, Y)\bullet a \oplus b \quad \mapsto_\rho \quad a, b$

- Diverses causes peuvent être à l'origine d'un échec de filtrage. Le premier cas échoue à cause d'un symbole de tête différent ; le second à cause de sous-termes non filtrables ; le troisième à cause d'une même variable qui devrait recevoir deux instanciations distinctes.

  $(a \to b)\bullet c \quad \mapsto_\nu \quad \mathit{null}$

  $(f\bullet a \to b)\bullet(f\bullet b) \quad \mapsto_\nu \quad \mathit{null}$

  $(f\bullet X\bullet X \to X)\bullet(f\bullet a\bullet b) \quad \mapsto_\nu \quad \mathit{null}$

- Enfin, une théorie non vide peut modifier très largement le comportement du calcul. On part d'un même terme mais on modifie les propriétés de l'opérateur $\oplus$ (ici en notation infixe : $X \oplus Y \simeq \oplus\bullet X\bullet Y$). Cet opérateur est successivement syntaxique, associatif, commutatif et enfin associatif-commutatif :

  $(X \oplus Y \to X)\bullet(a \oplus (b \oplus c)) \quad \mapsto_\emptyset \quad a$

  $(X \oplus Y \to X)\bullet(a \oplus (b \oplus c)) \quad \mapsto_A \quad a, a \oplus b$

  $(X \oplus Y \to X)\bullet(a \oplus (b \oplus c)) \quad \mapsto_C \quad a, b \oplus c$

  $(X \oplus Y \to X)\bullet(a \oplus (b \oplus c)) \quad \mapsto_{AC} \quad a, b, c, a \oplus b, b \oplus c, a \oplus c$

1.3 Problèmes de confluence

Il apparaît rapidement que le ρ-calcul n'est pas confluent, ce qui pose des problèmes de déterminisme si on considère un terme comme un programme à exécuter. Nous allons voir sur quelques exemples les causes principales de cette non-confluence, puis nous décrirons les mécanismes proposés par H. Cirstea et L. Liquori pour assurer la confluence du calcul.

1.3.1 Exemples de réductions non confluentes

Une première catégorie de réductions non confluentes est due à l'apparition du terme *null* à cause d'un échec.

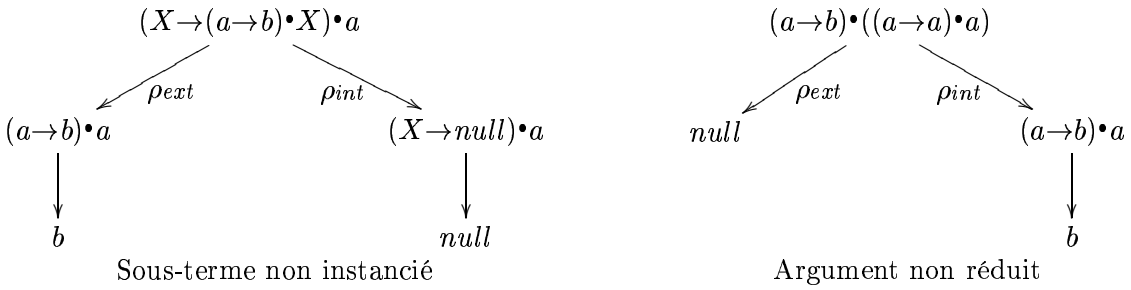


FIG. 1.3 – Réductions non confluentes par échec

Les deux exemples 1.3 suffisent à montrer que les stratégies consistant à réduire le rédex le plus externe, ou le plus interne, ne seraient pas appropriées. En effet, si une réduction ne débouche pas sur *null*, on voudrait pouvoir la privilégier afin de garder un maximum d'informations sur le terme considéré. Or, pour le premier terme c'est le rédex interne qui pose problème, alors que pour le second c'est le rédex externe.

Une autre idée naturelle serait de simplement supprimer les réductions à *null*, puisqu'elles semblent être la cause essentielle de la non-confluence. Les exemples 1.4 montrent que cela ne suffirait pas :

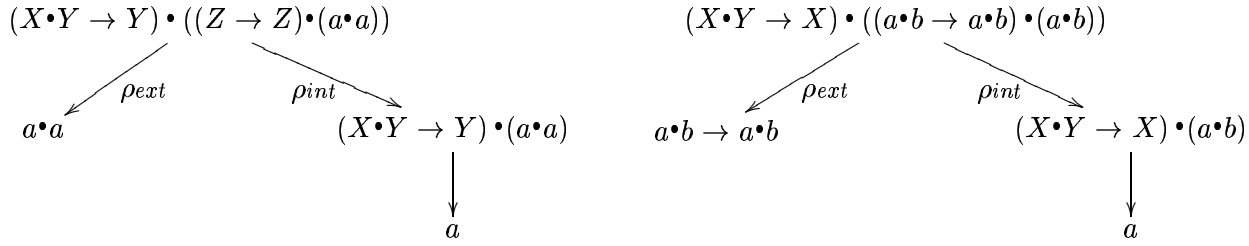


FIG. 1.4 – Réductions non confluentes par variable active

Dans ces termes, le problème semble se situer plutôt au niveau du motif $X{\bullet}Y$. En effet, si on passe alors en argument l'application $A{\bullet}B$ de deux ρ-termes, on peut avoir le choix de considérer directement la substitution $[X/A, Y/B]$ ou bien de réduire $A{\bullet}B$ puis de retenter le filtrage. Il devient

alors très difficile de discerner quel chemin a été suivi rien qu'en examinant les résultats finaux. On dit dans ce cas que la variable problématique X est *active*, parce qu'elle est en position fonctionnelle.

Il apparaît donc nécessaire d'effectuer les bonnes restrictions afin de recouvrer la confluence du calcul. Dans sa thèse, H. Cirstea avait proposé une stratégie relativement complexe, qui faisait entrer en jeu la linéarité des règles, la présence de structures dans l'argument auquel s'appliquait une règle, et des considérations de premier ordre sur ce même argument. Ici, nous reprendrons deux stratégies proches de l'appel par valeurs, et nous étudierons la possibilité d'obtenir la confluence par une restriction syntaxique.

1.3.2 Utilisation de stratégies d'évaluation

H. Cirstea a proposé deux stratégies [Cir01] qui s'inspirent assez largement de l'appel par valeurs de G. Plotkin pour le λ-calcul [Plo75]. Elles sont très similaires dans leur principe : on définit une sous-classe des ρ-termes, puis on n'autorise les réductions que sous certaines conditions exprimées en termes d'appartenance à cette classe :

- Le ρ-calcul avec appel par valeurs, ρ_v :

  Valeurs $\quad \mathcal{V} ::= \mathcal{S} \mid X^{\mathcal{T}} \mid a^{\mathcal{T}} \mid \mathcal{V} \to \mathcal{V} \mid \underline{a^{\mathcal{T}} \bullet \mathcal{V} \bullet \ldots \bullet \mathcal{V}} \mid \mathit{null} \mid \mathcal{V}, \mathcal{V}$

  $(A \to B) \bullet C$ est réduit seulement si $\begin{cases} A \in \mathcal{V},\ C \in \mathcal{V} \\ FV(C) = \emptyset \end{cases}$

  Les réductions $\mapsto_\delta$ et $\mapsto_\nu$ sont inchangées.

- Le ρ-calcul avec appel par valeurs rigides, ρ_{rv}^{ok} :

  Valeurs rigides $\quad \mathcal{RV} ::= \mathcal{S} \mid X^{\mathcal{T}} \mid a^{\mathcal{T}} \mid \mathcal{RV} \to \mathcal{RV} \mid \underline{a^{\mathcal{T}} \bullet \mathcal{RV} \bullet \ldots \bullet \mathcal{RV}} \mid \mathcal{RV}, \mathcal{RV}$

  $(A \to B) \bullet C$ est réduit seulement si $\begin{cases} A \in \mathcal{RV} \\ \exists \sigma,\ \sigma A \overset{\mathbb{T}}{=} C \end{cases}$

  La réduction $\mapsto_\delta$ est inchangée, et $\mapsto_\nu$ n'a plus lieu d'être

Tout en restant très descriptif, voyons en quoi consistent exactement ces stratégies et ce qui fait qu'elles vont atteindre leur but :

- La stratégie d'appel par valeurs fait une restriction raisonnable sur les termes : on ne peut plus mettre un terme quelconque à gauche d'un $\bullet$, ce qui revient plus ou moins à ne considérer que des termes du premier ordre. Cependant, on autorise toujours les variables en position active. C'est la contrainte posée sur la réduction qui est véritablement restrictive : en imposant que l'argument C soit une valeur et un terme clos, on évite à la fois les problèmes de sous-terme non réduit et de variable non instanciée. Ces restrictions entraînent également que les réductions sont faites d'abord dans l'argument, et donc il n'y aura plus d'ambiguïté sur le terme à affecter aux variables actives.
- Le ρ_{rv}^{ok} est un peu plus difficile à appréhender puisqu'il mélange divers concepts. En effet, comme envisagé dans la section précédente, toutes les réductions menant à *null* sont bloquées (d'où le "^{ok}") : il ne reste donc *a priori* plus que les problèmes de variable active. On restreint donc la forme des motifs pour ne plus avoir que des constantes à gauche d'une application. Ceci nous permet de ne poser aucune condition sur l'argument C, et on vérifie que le calcul alors considéré est bien confluent.

Nous n'entrerons pas plus en détails dans l'étude de ces stratégies. La démonstration de la confluence est assez technique ; elle utilise une version parallèle de la relation de réduction, de la même manière que la démonstration classique de confluence pour le λ-calcul donnée par Tait et Martin-Löf.

Remarquons que ces deux stratégies se contentent d'imposer des contraintes sur le membre gauche d'une abstraction et sur l'argument auquel cette abstraction est appliquée. Par la suite, nous démontrerons des propriétés indépendamment de la méthode employée pour obtenir la confluence, mais toujours en supposant que le membre droit des abstractions n'est pas concerné. Cela aurait vraisemblablement peu de sens puisque le problème de filtrage associé à une ρ-réduction est entièrement déterminé par le membre gauche et l'argument.

1.3.3 Utilisation de restrictions syntaxiques

Les stratégies sont bien adaptées lorsqu'on considère un calcul comme un modèle de langage de programmation. Cependant, si c'est plutôt l'aspect de terme de preuve qui nous intéresse, il peut sembler déplacé de parler de stratégie. Une autre solution pour obtenir la confluence est alors de restreindre le calcul en contraignant directement la forme des termes, puis en prouvant que ces contraintes restent vérifiées au cours des réductions.

Dans [BCKL02], L. Liquori étudie un calcul assez proche du nôtre, mais vu comme une extension conservative des *Pure Type Systems* de S. Berardi (voir [Bar92] par exemple). La technique employée pour assurer la confluence est largement inspirée des travaux de V. van Oostrom [Oos90]. Comme pour le ρ_{ok}, on supprime les réductions menant à *null* ; puis la seule restriction qu'on pose est sur la forme des motifs, autrement dit des membres gauches des règles.

Définition 14 (Condition de motif rigide)
Un terme $A \equiv \mathsf{Ctx}[X_1, \dots X_n]$ *est un* motif rigide *si, pour toute substitution* σ *portant sur* $X_1, \dots X_n$*, toute réduction parallèle de* σA *reste un terme de la forme* $\mathsf{Ctx}[\sigma' X_1, \dots \sigma' X_n]$*, où les* $\sigma' X_i$ *sont obtenus par réduction parallèle des* σX_i*.*

En particulier, on constate que l'ensembles des termes linéaires, en forme normale et sans variable libre en position active sont des motifs rigides.

La classe des termes admis est alors la même qu'auparavant, sauf pour la forme $\mathcal{T} \to \mathcal{T}$ qui devient $\mathcal{P} \to \mathcal{T}$ où $\mathcal{P}$ est un motif rigide. Comme généralement avec la confluence, il suffit alors de prouver que la relation de réduction parallèle vérifie la propriété du losange (ou confluence forte) pour les termes ainsi formés. Comme la propriété de motif rigide est préservée par réduction, on obtient la confluence pour la relation de réduction simple.

Cette restriction n'est ici donnée qu'à titre informatif, puisque sa validité a en fait été prouvée par L. Liquori pour un autre calcul, où notamment les deux abstracteurs λ et Π sont encore différenciés. Toutefois, la cause essentielle de non-confluence semble être la présence de motifs, et donc la condition de motif rigide devrait être adaptable au ρ-calcul tel que nous le présentons ici. Cela pourrait faire l'objet d'une étude ultérieure.

Chapitre 2

Un système de types à la Barendregt

Les principes opérationnels du calcul étant fixés, nous pouvons maintenant nous intéresser à ses aspects plus statiques, et notamment les systèmes de types qu'on peut lui attribuer. Je commencerai par décrire les 8 systèmes rassemblés par Barendregt sous le nom de *lambda-cube*, qui donnent une vision plus modulaire de l'isomorphisme de Curry-Howard, ainsi que les propriétés classiques sur ces systèmes. Je ne donnerai pas d'exemple d'inférence de type, puisque les exemples classiques seront repris dans le contexte du ρ-cube, qui nous intéresse plus directement.

Je présenterai ensuite l'adaptation de ces systèmes au ρ-calcul, proposée dans [CKL01]. Enfin je décrirai deux insuffisances de ces adaptations qui sont apparues au cours de mes travaux, ainsi que les modifications que nous avons adoptées pour y remédier.

2.1 Le λ-cube de Barendregt

2.1.1 Définition uniforme de 8 systèmes de types

Pour le λ-calcul, le typage se fait généralement de la façon suivante : on écrit un jugement de la forme $\Gamma \vdash M : N$, qui signifie qu'on peut attribuer le type N au terme M, dans le contexte Γ. Ce Γ est en fait un ensemble de préconditions $x : A$ qui attribuent un type (A) à certaines des variables (x), et constitue l'ensemble des hypothèses de typage nécessaires pour dériver le jugement de typage $M : N$. On remarquera que les variables liées par un abstracteur λ ou Π sont également données avec leur type : c'est la présentation dite *à la Church*. Dans le λ-cube de Barendregt, les règles d'inférence qui déterminent le typage sont les suivantes :

$$\frac{(s_1, s_2) \in \mathcal{A}}{\vdash s_1 : s_2}\ (axiom) \qquad \frac{\Gamma \vdash A : s}{\Gamma, x : A \vdash x : A}\ (start)\ ,\ x \notin \Gamma$$

$$\frac{\Gamma \vdash A : B \quad \Gamma \vdash C : s}{\Gamma, x : C \vdash A : B}\ (weak)\ ,\ x \notin \Gamma$$

$$\frac{\Gamma \vdash F : \Pi(x : A).B \quad \Gamma \vdash a : A}{\Gamma \vdash Fa : [x/a]B}\ (appl) \qquad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi(x : A).B : s_3}\ (prod)\ ,\ (s_1, s_2, s_3) \in \mathcal{R}$$

$$\frac{\Gamma, x : A \vdash b : B \quad \Gamma \vdash \Pi(x : A).B : s}{\Gamma \vdash \lambda(x : A).b : \Pi(x : A).B}\ (abs) \qquad \frac{\Gamma \vdash A : C \quad \Gamma \vdash B : s \quad B =_\beta C}{\Gamma \vdash A : B}\ (conv)$$

Donnons une idée rapide de la signification de ces règles ; nous laissons (*axiom*) et (*prod*) de côté pour l'instant. (*start*) et (*weak*) effectuent la gestion du contexte, en vérifiant qu'il est cohérent et que les types attribués aux variables sont corrects. (*appl*) décrit le typage de l'application d'un terme à un autre : cette règle nécessite une substitution d'ordre supérieur dans les types, car ceux-ci ne sont pas pourvus de la même notion de réduction que les termes. (*abs*) attribue aux abstractions un type *produit dépendant*, formé par l'abstracteur Π ; ainsi, le produit $\Pi(x : A).B$ décrit un type B prenant un argument x de type A, x pouvant éventuellement apparaître dans B. (*conv*) est une règle de conversion qui postule que deux types équivalents modulo la clôture réflexive, symétrique et transitive de $\mapsto_\beta$ le sont également en termes de jugements de typage.

On s'aperçoit que la règle (*axiom*) est paramétrée par un ensemble $\mathcal{A}$ qui caractérise les relations entre les différentes sortes. Dans notre cas, les seules sortes seront $*$ et $\Box$, et le seul axiome sera $\vdash * : \Box$.

De même, la règle (*prod*), qui régit la formation des produits dépendants avec l'abstracteur Π, est paramétrée par l'ensemble $\mathcal{R}$. Nous ne nous intéresserons qu'à des systèmes dans lesquels $s_2 = s_3$ pour toute règle de $\mathcal{R}$, et nous les noterons donc simplement (s_1, s_2) au lieu de (s_1, s_2, s_2) ; par ailleurs, la règle $(*, *)$ est indispensable pour que le λ-calcul simplement typé soit reconnu par le système. Comme nous ne disposons que de deux sortes distinctes, il ne reste plus qu'à choisir si on considère ou non les règles $(*, \Box)$, $(\Box, *)$ et $(\Box, \Box)$, ce qui nous donne 8 systèmes distincts. On les représente traditionnellement par un cube dont ils occupent les sommets :

Système	Règles associées			
$\lambda\!\to$	$(*,*)$			
$\lambda 2$	$(*,*)$	$(\Box,*)$		
$\lambda P = \mathsf{LF}$	$(*,*)$		$(*,\Box)$	
$\lambda P2$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	
$\lambda\underline{\omega}$	$(*,*)$			$(\Box,\Box)$
$\lambda\omega$	$(*,*)$	$(\Box,*)$		$(\Box,\Box)$
$\lambda P\underline{\omega}$	$(*,*)$		$(*,\Box)$	$(\Box,\Box)$
$\lambda P\omega = \lambda C$	$(*,*)$	$(\Box,*)$	$(*,\Box)$	$(\Box,\Box)$

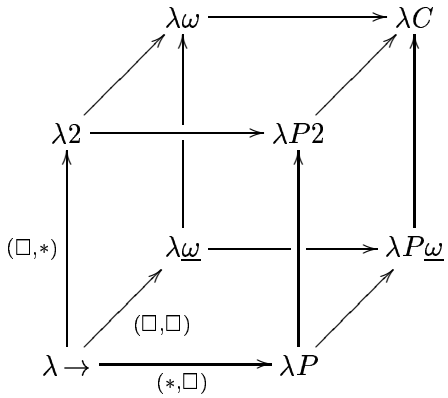


FIG. 2.1 – Le λ-cube de Barendregt

Le système $\lambda\!\to$ représente le λ-calcul simplement typé, avec $\mathcal{R} = \{(*, *)\}$. Ensuite, selon la direction dans laquelle on part, on ajoute des règles et donc de l'expressivité aux types :

$(\Box, *)$ permet de construire des termes dépendant de types. On obtient ainsi la notion de polymorphisme : le terme $\lambda(\alpha : *).\lambda(x : \alpha).x$ représente l'identité, mais paramétrée par un type α quelconque qui sera donné comme premier argument. Le système associé est appelé $\lambda 2$ car on obtient un langage du second ordre, dans lequel on peut par exemple exprimer l'absurde ou la négation ; il correspond au système F de Girard [GLT89].

$(\Box, \Box)$ permet de construire des types dépendant de types. Cette "circularité" n'est cependant pas suffisante pour obtenir l'ordre supérieur, car aucune règle (*prod*) ne permet de passer des termes aux types. Ce système est donc peu utilisé, et il est noté $\lambda\underline{\omega}$ par imitation de $\lambda\omega$ qui représente le véritable ordre supérieur.

$(*, \Box)$ permet de construire des types dépendant de termes. Ainsi, on retrouve des paradigmes déjà envisagés en programmation : le type $\lambda(n : int).\mathsf{List}\ n$ peut représenter un type de liste dont on connaît la taille n, et on pourra par exemple caractériser les fonctions qui laissent invariante la taille d'une liste. Ce système a été proposé pour la première fois par N. de Bruijn au sein du projet AUTOMATH [Bru70]. Il permet d'exprimer la logique des prédicats (en considérant un type dépendant d'un terme comme une proposition dépendant d'une variable) et se note donc λP (pour *P*rédicat). Il correspond également au "Logical Framework" (LF) de [HHP87].

Si on suit plus d'une flèche, on combine plusieurs règles, ce qui donne parfois des résultats intéressants :

$(\Box, *) + (\Box, \Box)$: la première règle permet de passer au second ordre, puis la suivante donne accès à tout l'ordre supérieur. On peut donc quantifier sur toutes les propositions : c'est le système Fω de Girard, que l'on note aussi parfois $\lambda\omega$.

$(\Box, *) + (\Box, \Box) + (*, \Box)$: en considérant à la fois les 3 règles possibles, on obtient le système λC, qui correspond en fait au Calcul des Constructions de Coquand [CH88], puisqu'il englobe les systèmes Fω et LF.

Notons que la forme la plus générale des systèmes vus ci-dessus, avec des axiomes $\mathcal{A}$ et des règles $\mathcal{R}$ arbitraires, sont ce qu'on appelle les *Pure Type Systems*, ou PTS. La plupart d'entre eux vérifient les lemmes qui seront énoncés dans la partie suivante. Cependant, ils ne sont pas tous fortement normalisants, ce qui permet l'existence d'un terme inconsistant (voir lemme 2.5). C'est par exemple le cas dans le système $\lambda*$ où le seul axiome de $\mathcal{A}$ est $\vdash * : *$ (et non plus $\vdash * : \Box$), qui constitue le premier paradoxe de Girard.

2.1.2 Résultats classiques

Ces propriétés classiques des systèmes du λ-cube sont résumées par Barendregt dans [Bar92]. Comme nous le verrons, elles expriment souvent des propriétés de correction et sont largement souhaitables pour un calcul typé. La plupart des résultats que nous démontrerons au cours du chapitre 3 sont directement inspirés de ceux-ci.

Le premier lemme est purement technique : il assure qu'un jugement de typage est invariant par substitution, à condition que celle-ci se fasse avec des termes de type approprié. Il sert uniquement en un point précis pour la démonstration de la préservation du type (lemme 2.4).

Lemme 2.1 (Substitution dans le λ-cube)

$$\begin{gathered}\forall\ x, A, B, C, D \text{ tels que :} \\ \Gamma,\ x : A,\ \Delta \vdash B : C \\ \Gamma \vdash D : A \\ \text{On a} \quad \Gamma, [x/D]\Delta \vdash [x/D]B : [x/D]C\end{gathered}$$

Le prochain lemme est également utilisé principalement dans les preuves des suivants. Il permet, à partir d'un jugement de typage et selon la forme du terme, d'obtenir des informations sur la forme de son type. C'est en fait la formalisation d'une simple observation des règles de typage.

Lemme 2.2 (Génération dans le λ-cube)

1.	$\Gamma \vdash s : C$	$\Rightarrow \exists s',$	$(s : s') \in \mathcal{A}$	et $C =_\beta s'$	
2.	$\Gamma \vdash x : C$	$\Rightarrow \exists s, B,$	$\Gamma \vdash B : s$	et $C =_\beta B$	et $x : B \in \Gamma$
3.	$\Gamma \vdash \Pi(x : A).B : C$	$\Rightarrow \exists s_1, s_2, C,$	$\Gamma \vdash A : s_1$	et $C =_\beta s_2$	et $\Gamma, x : A \vdash B : s_2$
4.	$\Gamma \vdash \lambda(x : A).b : C$	$\Rightarrow \exists s, B,$	$\Gamma \vdash \Pi(x : A).B : s$	et $C =_\beta \Pi(x : A).B$	et $\Gamma, x : A \vdash b : B$
5.	$\Gamma \vdash Fa : C$	$\Rightarrow \exists x, A, B,$	$\Gamma \vdash F : \Pi(x : A).B$	et $C =_\beta [x/a]B$	et $\Gamma \vdash a : A$

Le lemme suivant stipule que le type d'un terme est *correct*, autrement dit qu'il a lui-même un type qui est une sorte. Dans la littérature sur le λ-calcul, il est rarement mis en évidence car il sert essentiellement à répartir les termes en trois classes : sortes, types et éléments. Nous l'utiliserons de façon plus intensive car la règle de conversion est très importante dans le ρ-cube.

Lemme 2.3 (Correction dans le λ-cube)
Pour tous termes A et B, $\Gamma \vdash A : B \;\Rightarrow\; \exists s, \quad B \equiv s$ ou $\Gamma \vdash B : s$.

Les trois lemmes précédemment énoncés permettent de démontrer le théorème suivant. Il affirme qu'un terme typable, une fois réduit, admet les mêmes types qu'avant réduction. Cette propriété est fondamentale, puisque le type est censé donner une information statique sur le terme, et on ne voudrait pas qu'un terme initialement correct puisse se réduire en un terme incorrect.

Théorème 2.4 (Préservation du type dans le λ-cube)
Le type est préservé par réduction :

$$\forall A, B, \quad \Gamma \vdash A : B \;\wedge\; A \mapsto_\beta^* A' \quad \Rightarrow \quad \Gamma \vdash A' : B$$

Dans tous les systèmes étendant $\lambda 2$, on peut définir le type $\Pi(\alpha : *).\alpha$. Il est noté $\bot$ et il est dit *absurde*, car l'existence d'un terme A ayant pour type $\bot$ rendrait le calcul inconsistant. En effet, pour tout type B, on aurait alors (par la règle de typage (*appl*)) le jugement $\vdash AB : B$, et donc tout type serait *habité* (autrement dit il existerait un terme ayant ce type).

Le système de types n'ayant d'intérêt que s'il permet de discriminer des types habités ou non, on dit qu'il est inconsistant si $\bot$ est habité. Pour décider de cette propriété, on commence par s'assurer que le lemme suivant est vrai :

Lemme 2.5 (Consistance dans le λ-cube)
Pour tout terme A en forme normale, $\not\vdash A : \bot$

Comme les termes en forme normale ne violent pas la consistance, on peut alors se ramener à des conditions de normalisation :

Remarque 2.1
Si le calcul considéré est (faiblement) normalisant, alors tout terme typable admet (au moins) une forme normale. Par préservation du type, si un terme habite $\bot$, alors sa forme normale l'habite aussi et le lemme précédent est contradictoire.

On démontre notamment que tous les systèmes du λ-cube sont normalisants, et donc consistants.

Le lemme d'unicité assure qu'un même terme ne peut admettre deux types distincts, à conversion près. Cette propriété offre plusieurs avantages pour démontrer d'autres propositions, mais à notre connaissance elle n'est pas indispensable.

Lemme 2.6 (Unicité du typage dans le λ-cube)

$$\forall A, B_1, B_2, \quad \Gamma \vdash A : B_1 \;\wedge\; \Gamma \vdash A : B_2 \quad \Rightarrow \quad B_1 =_\beta B_2$$

Le dernier lemme montre que les systèmes de types considérés sont *a priori* utilisables en pratique : un jugement de typage est calculable et vérifiable. On notera que la présentation de Church, qui donne les types des variables liées, est indispensable pour avoir ce théorème. Il reste vrai pour tous les PTS tant que l'ensemble des sortes est fini.

Théorème 2.7 (Décidabilité du typage)
La vérification d'un jugement de typage et l'inférence d'un type pour un terme donné sont des problèmes décidables, quel que soit le contexte.

2.2 Le ρ-cube

Prenant modèle sur Barendregt, nous construisons de manière uniforme 8 systèmes de types pour le ρ-calcul, ainsi qu'un neuvième système inspiré du Calcul des Constructions Étendu (ECC) de Z. Luo [Luo90] qui sera traité à part. Suivant les mêmes motivations que [BKN99], à savoir la simplification du typage des applications, nous introduisons une notion de réduction au niveau des types ; cependant, en assimilant *totalement* les abstracteurs λ et Π dans $\rightarrow$, nous évitons les problèmes habituels de correction.

En plus des éléments propres au calcul tels que les structures, une différence importante est l'absence de contextes. En effet, comme on l'a vu dans la présentation de la section 1.2.1, les variables et les constantes sont décorées avec leur type, et donc il n'est plus nécessaire de le stocker hors du terme, sous réserve que les différentes décorations soient cohérentes. Ainsi, un jugement de typage dans un contexte vide correspond simplement à un terme clos sans constantes.

2.2.1 Adaptation des règles de typage

Système	Règles associées ($i, j \in \mathbb{N}$)			
$\rho\rightarrow$	$(*,*)$			
$\rho 2$	$(*,*)$	$(\Box_0,*)$		
ρP	$(*,*)$		$(*,\Box_0)$	
$\rho P2$	$(*,*)$	$(\Box_0,*)$	$(*,\Box_0)$	
$\rho\underline{\omega}$	$(*,*)$			$(\Box_0,\Box_0)$
$\rho\omega$	$(*,*)$	$(\Box_0,*)$		$(\Box_0,\Box_0)$
$\rho P\underline{\omega}$	$(*,*)$		$(*,\Box_0)$	$(\Box_0,\Box_0)$
ρCC	$(*,*)$	$(\Box_0,*)$	$(*,\Box_0)$	$(\Box_0,\Box_0)$
ρECC	$(*,*)$	$(\Box_i,*)$	$(*,\Box_i)$	$(\Box_i,\Box_j)$

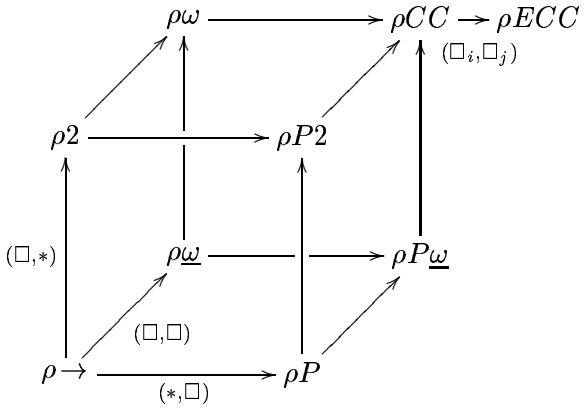


FIG. 2.2 – Les 9 (8 + 1) systèmes du ρ-cube

Règles inspirées du λ-cube :

$$\frac{}{\vdash * : \Box_0}\ (Axiom_*) \qquad \frac{i \in \mathbb{N}}{\vdash \Box_i : \Box_{i+1}}\ (Axiom_{\Box_i}) \qquad \frac{\vdash A : s}{\vdash \alpha^A : A}\ (Start)$$

$$\frac{\vdash A : C \rightarrow D \quad \vdash C : E \quad \vdash B : E}{\vdash A\bullet B : (C \rightarrow D)\bullet B}\ (Appl) \qquad \frac{\vdash A : A' \quad \vdash A' : s_1 \quad \vdash B : s_2}{\vdash A{\rightarrow}B : s_2}\ (Prod)$$

$$\frac{\vdash B : C \quad \vdash A \rightarrow C : s}{\vdash A{\rightarrow}B : A \rightarrow C}\ (Abs) \qquad \frac{\vdash A : C \quad \vdash B : s \quad B =_\rho C}{\vdash A : B}\ (Conv)$$

Règles spécifiques aux structures :

$$\frac{\vdash A : s}{\vdash null : A}\ (Null) \qquad \frac{\vdash A : C \quad \vdash B : C}{\vdash A, B : C}\ (Struct)$$

FIG. 2.3 – Les règles d'inférence de type dans le ρ-cube

Certaines de ces règles sont une traduction directe de celles du λ-cube, notamment $(Start)$ qui se contente d'utiliser les décorations au lieu des contextes, et $(Conv)$ qui remplace $=_\beta$ par $=_\rho$. Explicitons les autres règles, qui diffèrent véritablement :

$(Axiom_{\Box_i})$ Cette collection d'axiomes ne sert en fait que pour le système ρECC, qui sera détaillé dans la section suivante. Elle engendre une hiérarchie infinie dénombrable ordonnée de sortes.

$(Appl)$ Une traduction directe de cette règle aurait donné pour $A \bullet B$ un type de la forme $[C/B]D$. Cependant, nous avons choisi d'*identifier* les deux abstracteurs λ et Π dans la flèche $\to$. Ceci nous permet donc d'écrire la prémisse $\vdash A : C \to D$ au lieu d'un Π-terme, et de laisser la règle de conversion effectuer la substitution dans $(C \to D) \bullet B$ si nécessaire. Nous avons donc supprimé le seul élément du typage qui était extérieur au calcul.

Par ailleurs, la présence d'un motif C au lieu d'une variable dans le type de A nous oblige à relancer une inférence de type $\vdash C : E$, qui n'était pas nécessaire dans le λ-cube grâce aux notations de Church.

$(Prod)$ Cette règle fonctionne exactement comme pour le produit dépendant du λ-cube, avec les mêmes conséquences lorsqu'on considère les couples $(*, \Box)$, $(\Box, *)$ ou $(\Box, \Box)$ comme instances de (s_1, s_2). Nous verrons que le système ρECC fait une utilisation beaucoup plus générale de cette règle. Lorsqu'il n'y a pas d'ambiguïté, on condensera parfois les deux premières prémisses $\vdash A : A'$ et $\vdash A' : s_1$ en $\vdash A : A' : s_1$.

(Abs) Les seules particularités notables de cette règle sont la présence d'un motif, qui a peu d'influence à cet endroit, et la contraction des deux abstracteurs λ et Π en $\to$, qui a déjà été remarquée à propos de $(Prod)$. Le typage n'est plus directement guidé par la syntaxe, puisque (Abs) comme $(Prod)$ sont *a priori* candidates pour typer une abstraction. Enfin, le jugement $\vdash B : C$ sera dérivé correctement sans contexte grâce aux hypothèses faites sur les décorations des variables.

$(Null)$ La constante *null* doit pouvoir admettre tout type bien formé. C'est en effet nécessaire pour que la règle $(Struct)$ s'applique au cas "$A, null$" (qui est en fait $\overset{\mathbb{T}}{=} A$), où A peut être quelconque et donc avoir un type *a priori* arbitraire.

$(Struct)$ Le choix fait pour le typage d'une structure est que tous ses éléments doivent être du même type. Il serait envisageable de prendre $\vdash A, B : C, D$ si $\vdash A : C$ et $\vdash B : D$, mais cela demanderait de réaménager la règle $(Appl)$ dans le cas des applications de la forme $(A, B) \bullet C$.

Nous pouvons désormais donner quelques exemples d'inférences de type. La plupart sont tirés de [CKL01] et sont en fait des traductions d'inférences déjà envisageables dans le λ-cube. Comme on ne s'intéresse pour l'instant pas à ρECC, on écrira $\Box$ pour $\Box_0$.

- Le système $\rho\to$ correspond à peu près au ρ-calcul simplement typé proposé dans [CK00]. Considérons les constantes int^*, 3^{int^*} et 4^{int^*}. Le premier exemple montre comment fonctionnent les structures ainsi que les règles $(Start)$ et $(Axiom_*)$. Par la suite, nous omettrons ces deux dernières règles lorsqu'elles sont évidentes grâce aux décorations.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash * : \Box}\,(Axiom_*)}{\vdash int^* : *}\,(Start) \qquad \dfrac{\dfrac{\dfrac{}{\vdash * : \Box}\,(Axiom_*)}{\vdash int^* : *}\,(Start)}{\vdash 3^{int^*} : int^*}\,(Start) \qquad \vdash 4^{int^*} : int^*}{\vdash 3^{int^*}, 4^{int^*} : int^*}\,(Struct)$$

$$\dfrac{\dfrac{\dfrac{\vdash X^{int^*} : int^* \quad \vdash int^* : *}{\vdash X^{int^*} : int^* \qquad \vdash X^{int^*} \to int^* : *}\ (Prod)}{\vdash X^{int^*} \to X^{int^*} : X^{int^*} \to int^*}\ (Abs) \qquad \vdash X^{int^*} : int^* \quad \vdash 3^{int^*} : int^*}{\dfrac{\vdash (X^{int^*} \to X^{int^*})\bullet 3^{int^*} : (X^{int^*} \to int^*)\bullet 3^{int^*} \qquad \vdash int^* : * \qquad (X^{int^*} \to int^*)\bullet 3^{int^*} =_\rho int^*}{\vdash (X^{int^*} \to X^{int^*})\bullet 3^{int^*} : int^*}\ (Conv)}\ (Appl)$$

– Dans le système $\rho 2$, voyons comment se typent le terme absurde $\bot \triangleq X^* \to X^*$, et l'identité polymorphe. Dans les deux cas, au moins une règle produit $(\Box, *)$ est utilisée :

$$\dfrac{\dfrac{\dfrac{\vdash X^* : * \quad \vdash * : \Box \quad \vdash X^* : *}{\vdash X^* \to X^* : *}\ (Prod\ \Box, *)}{\vdash Y^{X^*} : X^* \to X^* : *}\ (Abs) \qquad \dfrac{\vdash X^* : * : \Box \qquad \dfrac{\vdash Y^{X^*} : X^* : * \quad \vdash X^* : *}{\vdash Y^{X^*} \to X^* : *}\ (Prod)}{\vdash X^* \to (Y^{X^*} \to X^*) : *}\ (Prod\ \Box, *)}{\vdash X^* \to (Y^{X^*} \to Y^{X^*}) : X^* \to (Y^{X^*} \to X^*)}\ (Abs)$$

Cette dernière inférence montre bien l'intérêt du système $\rho 2$: il permet d'écrire des fonctions dépendant d'un type. Comme le ρ-calcul inclut également un mécanisme de filtrage, ce système nous fournit donc un modèle proche de ML ; le terme $Y^* \to X^{Y^*} \to X^{Y^*}$ correspond au programme `fun id x = x`, de type `'a -> 'a`. On notera que ML ne demande pas qu'on lui passe explicitement le type `'a` en argument, mais qu'il va lui-même essayer de l'inférer à partir de l'argument auquel est appliquée `id`. Les résultats de typage qu'on pourra prouver à propos de $\rho 2$ ont donc un intérêt tout particulier dans cette optique.

– De même, on peut trouver des termes nécessitant l'une ou l'autre des différentes règles de produit. Ainsi on a :
  dans $\rho\underline{\omega}$, $\quad \vdash X^* \to * : \Box$
  dans ρP, $\quad \vdash X^{int^*} \to * : \Box$
  dans $\rho P\underline{\omega}$, $\quad \vdash X^* \to Y^{X^*} \to * : \Box$
  dans ρCC, $\quad \vdash X^* \to f^{Y^* \to Y^*} \to * : \Box$

– Notons enfin que le système de types s'adapte bien à la présence de symboles de constantes :
$\vdash f^{X^{int^*} \to int^*}\bullet 3^{int^*} : (X^{int^*} \to int^*)\bullet 3^{int^*} =_\rho int^*$
$\vdash (f^{Y^* \to *}\bullet Z^* \to g^{Y^* \to *}\bullet Z^*)\bullet f^{Y^* \to *}\bullet a^* : (f^{Y^* \to *}\bullet Z^* \to (Y^* \to *)\bullet Z^*)\bullet f^{Y^* \to *}\bullet a^*$
$=_\rho (Y^* \to *)\bullet a^* =_\rho *$

2.2.2 Un système supplémentaire : ρECC

Le neuvième système de types proposé est une extension de ρCC, inspirée du Calcul des Constructions Étendu [Luo90]. Le fragment qui nous intéresse ici est la présence d'une hiérarchie infinie de sortes commençant par $\Box_0$ ($= \Box$), et gouvernée par les axiomes :

$$\dfrac{i \in \mathbb{N}}{\vdash \Box_i : \Box_{i+1}}\ (Axiom_{\Box_i})$$

Un premier intérêt est qu'on peut désormais manipuler tous les éléments du calcul sans distinction : par exemple, la sorte $\Box$ est désormais typable alors que, dans le cube proprement

dit, il fallait la traiter à part. Pour la formation des produits dépendants, tous les couples de sortes sont admis dans ρECC : $(*,*)$, $(*,\Box_j)$, $(\Box_i,*)$ ou $(\Box_i,\Box_j)$ pour tous $i,j \in \mathbb{N}$.

Il est facile de voir l'utilité de telles règles si on introduit des variables décorées telles que $X^{\Box}$, ou encore si on veut utiliser la règle d'inférence (Abs) sur des abstractions qui seraient typées comme un produit dans le ρ-cube :

$$\dfrac{\dfrac{\vdash \Box : \Box_1}{\vdash X^{\Box} : \Box}\ (Start) \qquad \vdash \Box : \Box_1 \quad \vdash X^{\Box} : \Box}{\vdash X^{\Box} \to X^{\Box} : \Box}\ (Prod\ \Box_1, \Box)$$

$$\dfrac{\vdash Y^* : * \quad \vdash * : \Box \quad \vdash * : \Box}{\vdash Y^* \to * : \Box}\ (Prod\ \Box, \Box) \qquad \dfrac{\vdash * : \Box \qquad \dfrac{\vdash Y^* : * \quad \vdash * : \Box \quad \vdash \Box : \Box_1}{\vdash Y^* \to \Box : \Box_1}\ (Prod\ \Box, \Box_1)}{\vdash Y^* \to * : Y^* \to \Box}\ (Abs)$$

D'autre part, nous pourrons dériver très généralement des jugements de la forme $\vdash A : s$, et utiliser la règle de conversion plus facilement pour typer des applications :

$$\dfrac{\dfrac{\vdash X^{\Box} : \Box : \Box_1 \quad \vdash X^{\Box} : \Box}{\vdash X^{\Box} \to X^{\Box} : \Box} \qquad \vdash X^{\Box} : \Box \quad \vdash Z^{\Box} : \Box}{\vdash (X^{\Box} \to X^{\Box}) \bullet Z^{\Box} : (X^{\Box} \to \Box) \bullet Z^{\Box}}\ (Appl) \qquad \vdash \Box : \Box_1 \quad (X^{\Box} \to \Box) \bullet Z^{\Box} =_{\rho} \Box}{\vdash (X^{\Box} \to X^{\Box}) \bullet Z^{\Box} : \Box}\ (Conv)$$

2.3 Quelques aménagements nécessaires

Cette section présente deux problèmes du calcul typé, rencontrés en mettant au point les démonstrations du chapitre 3. Les solutions que nous avons apportées sont décrites en détail et motivées.

2.3.1 Introduction des types dans la substitution

Il nous faut adapter les notions de substitution et de filtrage dans le contexte du ρ-cube, autrement dit avec des variables et des constantes *typées*. En effet, dans le lemme 2.1 de substitution du λ-cube, il suffit de prendre en compte le type de la seule variable liée x ; la coïncidence du type de x avec son image σx est assurée par la règle de typage $(appl)$.

Dans notre cas, c'est un motif et non plus une simple variable que nous traitons, et donc nous devons imposer de manière plus explicite les correspondances de types entre variables et arguments. Notre choix est de garder des théories non typées, afin de bien séparer les concepts. La vérification des types interviendra au niveau de la substitution ; il faut bien voir que le filtrage se fait alors modulo une théorie, comme précédemment, mais aussi modulo des règles de typage, par le biais de la définition de substitution typée.

Définition 15 (Substitution typée)
Pour un jugement de typage $\vdash$ *sur les termes, une substitution* $\sigma = [X_1^{A_1}/B_1 \ldots X_n^{A_n}/B_n]$ *est dite* substitution typée *si* $\quad \forall i,\ \vdash B_i : \sigma A_i$

Cette définition de substitution remplacera désormais la définition usuelle (déf. 4) ; on demandera notamment que la solution d'une équation de filtrage soit une substitution typée.

Dans cette définition, la substitution elle-même apparaît dans les contraintes : on veut que le type de $\sigma(X_i^{A_i})$ soit un certain σA_i, car des variables de $\mathcal{D}om\ \sigma$ peuvent éventuellement apparaître dans A_i lui-même. Cependant, à cause de la convention de décoration 10, on est sûr qu'une variable X^A n'apparaît pas dans son propre type A ; on peut en fait prouver qu'il n'y a aucune "circularité" du genre X^Y et Y^X, car cela mènerait à des décorations infinies. Partant de là, il est possible de définir un ordre (partiel) sur les variables de $\mathcal{D}om\ \sigma$ de la façon suivante : $Y \prec X^A$ si $Y \in \mathcal{V}ar(A)$, et donc on peut ranger les contraintes $\vdash B_i : \sigma A_i$ dans un ordre qui inclut $\prec$, ce qui assure que, dans les différents σA_i, n'apparaîtront que des variables pour lesquelles la contrainte a déjà été vérifiée.

Remarque 2.2 *La contrainte de typage posée sur la substitution donne* au plus *autant de solutions que la simple condition syntaxique* $\sigma A \overset{\mathbb{T}}{=} B$*, puisqu'on ne fait qu'*ajouter *des contraintes. Un filtrage unitaire reste donc unitaire avec cette nouvelle définition. Par contre, la puissance de calcul du filtrage est augmentée, puisqu'on lui ajoute la capacité de décider des jugements de typage* $\vdash B_i : \sigma A_i$.

La contrainte de type est posée directement sur les variables et leur image par σ ; on verra que c'est la contrainte minimale, puisqu'elle est utilisée littéralement dans la preuve du lemme 3.2 de substitution pour le ρ-calcul. On pourrait imaginer qu'une contrainte sur B ou sur σA, au niveau des termes, serait suffisante. Cependant, contrairement au λ-calcul, les contraintes de type posées dans la règle (*Appl*) sont posées sur le motif et pas sur une variable, et donc une variable peut être suffisamment "cachée" pour que son type n'affecte pas le motif lui-même. On pourrait donc obtenir une substitution qui respecte les types de B et σA mais pas celui de la variable en question.

Dans la suite, on manipulera souvent les substitutions directement ; il ne faut pas pour autant oublier qu'elles apparaissent en tant que solutions d'une équation $A \ll_{\mathbb{T}} B$ lorsqu'on traite un rédex de la forme $(A \to C)\bullet B$. En particulier, on supposera très généralement que la stratégie choisie est assez cohérente pour que, si $\sigma \in \mathcal{S}ol(A \ll_{\mathbb{T}} B)$, alors un rédex de la forme $(X \to C)\bullet\sigma X$ soit réductible (avec $X \in \mathcal{D}om\ \sigma$). Remarquons que c'est le cas pour les deux stratégies que l'on connaît : dans le ρ_v-calcul avec des motifs du premier ordre, comme B est une valeur close, σX est aussi une valeur close, en tant que sous-terme (du premier ordre) de B ; c'est trivialement vrai pour le ρ_{rv}^{ok}-calcul puisque X est toujours une valeur rigide.

2.3.2 Gestion des restrictions confluentes

Pour raisonner sur le système de types, nous avons besoin de nous placer dans une restriction confluente du ρ-calcul. En effet, dans le ρ-calcul de base, en adaptant légèrement le premier exemple de non-confluence (figure 1.3), on peut montrer que tous les termes sont convertibles entre eux :

$$\forall B, C, \qquad \begin{array}{ccccccc} & (X\to(a\to B)\bullet X)\bullet a & & & (X\to(a\to C)\bullet X)\bullet a & & \\ \swarrow_{\rho} & & \searrow_{\rho} & & \swarrow_{\rho} & & \searrow_{\rho} \\ B & & & null & & & C \end{array} \qquad \text{donc } B =_\rho C$$

Par conséquent, à cause de la règle de conversion, les jugements de typage n'auraient plus aucune valeur. Il suffirait de savoir qu'un terme A est typable pour que, quel que soit le terme C tel que $\vdash C : s$, on puisse affirmer :

$$\frac{\vdash A : B \quad \vdash C : s \quad B =_\rho null =_\rho C}{\vdash A : C}\ (Conv)$$

Il faut donc choisir une restriction ou une stratégie confluente, telles que celles présentées dans la section 1.3. Ceci pose déjà un problème, puisque selon le choix fait pour assurer la confluence, le calcul et le système de types ne sont plus les mêmes. Nous allons voir sur quelques exemples que les difficultés rencontrées se révèlent bien plus critiques, puis nous proposerons une solution basée sur l'adaptation du système de types.

Pour se persuader que les changements opérés pour retrouver la confluence ne sont pas anodins, il suffit de constater que, avec la stratégie ρ_v, la contrainte de clôture posée sur l'argument diminue réellement l'expressivité. Ainsi, certains λ-termes et leurs réductions ne sont plus encodables :

$\lambda(\alpha:*).(\lambda(\beta:*).\beta)\alpha$ est un terme clos et typable (son type est $\Pi(\alpha:*).*$) et :

$$\lambda(\alpha:*).(\lambda(\beta:*).\beta)\alpha \quad \mapsto_\beta \quad \lambda(\alpha:*).\alpha$$

Mais dans ρ_v : $\quad X^* \to (Y^* \to Y^*)\bullet X^* \quad \not\mapsto_\rho \quad X^* \to X^*$

Car X^* est libre dans $(Y^* \to Y^*)\bullet X^*$; pour les mêmes raisons ce terme n'est pas typable.

Difficultés de typage

Du point de vue du typage, ce problème va se traduire par une restriction considérable de la règle $(Conv)$, ce qui se répercute directement sur la règle $(Appl)$. Considérons en effet la dérivation de type suivante, dans le système ρECC :

$$\dfrac{\dfrac{\dfrac{}{\vdash * : \square}\quad \dfrac{\dfrac{}{\vdash a^* : * : \square}\quad \dfrac{}{\vdash \square : \square_1}\,(Prod)}{\vdash a^* \to \square : \square_1}\,(Abs)}{\vdash a^* \to * : a^* \to \square}\quad \dfrac{}{\vdash a^* : *}\quad \dfrac{}{\vdash X^* : *}}{\vdash (a^* \to *)\bullet X^* : (a^* \to \square)\bullet X^*}\,(Appl)$$

Dans ρ_v, le type $(a^* \to \square)\bullet X^*$ est en forme normale à cause de $FV(X^*) \neq \emptyset$; dans ρ_{rv}^{ok}, il est en forme normale parce que l'équation de filtrage $a^* \ll_{\mathbb{T}} X^*$ n'a pas de solution. On a donc un type "minimal" au sens des réductions ; en particulier, un examen plus poussé montre qu'on ne pourra jamais dériver $\vdash (a^* \to *)\bullet X^* : \square$. Au premier abord, on peut penser que c'est la présence d'une variable libre qui perturbe le typage. Cependant, dans le terme *clos* suivant, on "oublie" au cours de l'inférence que X^* était lié et on retrouve le même problème :

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash a^* \to b^* : a^* \to *}\quad \dfrac{}{\vdash a^* : *}\quad \dfrac{}{\vdash X^* : *}}{\vdash (a^* \to b^*)\bullet X^* : (a^* \to *)\bullet X^*}\,(Appl)\quad \dfrac{\dfrac{}{\vdash X^* : * : \square}\quad \dfrac{\textbf{???}}{\vdash \boldsymbol{(a^* \to *)\bullet X^* : \square}}}{\vdash X^* \to (a^* \to *)\bullet X^* : \square}\,(Prod)}{\vdash X^* \to (a^* \to b^*)\bullet X^* : X^* \to (a^* \to *)\bullet X^*}\,(Abs)\quad \dfrac{}{\vdash X^* : *}\quad \dfrac{}{\vdash a^* : *}}{\vdash (X^* \to (a^* \to b^*)\bullet X^*)\bullet a^* : (X^* \to (a^* \to *)\bullet X^*)\bullet a^*}\,(Appl)$$

L'inférence est donc bloquée parce qu'on ne peut pas prouver $\vdash (a^* \to *)\bullet X^* : \square$ dans un sous-arbre. Or, lorsqu'on considère le terme entier, il paraîtrait raisonnable de lui attribuer le type $(X^* \to (a^* \to *)\bullet X^*)\bullet a^*$; comme X^* est à nouveau lié, ce terme est d'ailleurs réductible (dans ρ_v comme dans ρ_{rv}^{ok}) de la façon suivante : $(X^* \to (a^* \to *)\bullet X^*)\bullet a^* \;\mapsto_\rho\; (a^* \to *)\bullet a^* \;\mapsto_\rho\; *$

Dans un premier temps, ayant observé que la présence de variables liées dans les sous-inférences posait problème, nous avons envisagé de marquer ces variables et de relâcher un peu les conditions de confluence en leur présence. Mais d'une part, cela ne résout pas le problème d'expressivité de ρ_v, et d'autre part, un traitement trop laxiste des variables liées donnerait lieu à d'autres cas de non-confluence.

Règle d'application modifiée

Finalement, la solution retenue est d'*ajouter* une règle au système d'inférence. C'est une variante de la règle (*Appl*) initiale, qui vient la compléter mais pas la remplacer :

$$\frac{\vdash A : C \to s \quad \vdash C : E \quad \vdash B : E}{\vdash A \bullet B : s} \quad (ApplSort)$$

FIG. 2.4 – Une règle supplémentaire pour contourner les stratégies

Remarque 2.3 *Il faudra désormais faire attention car certaines applications peuvent être typées par l'une ou l'autre de ces règles. On pourrait rendre* (*Appl*) *et* (*ApplSort*) *mutuellement exclusives en précisant* $D \not\equiv s$ *dans* (*Appl*) *afin de mieux guider le typage par la syntaxe. Cependant, on peut toujours remplacer* $C \to s$ *par* $C \to (a \to s)\bullet a$ *au moyen de deux conversions encadrant la règle* (*Appl*) *; il n'y a donc rien à gagner à ajouter une condition sur* (*Appl*).

Explicitons la signification de (*ApplSort*) : un peu comme dans le λ-cube, on effectue en fait une substitution $[C/B]s$ au niveau "méta", mais comme cette substitution a lieu sur la sorte s qui n'est pas affectée, on peut se passer de $[C/B]$. La construction des types reste donc strictement dans les limites du calcul. Avec la seule règle (*Appl*), on aurait obtenu le type $(C \to s)\bullet D$, qui dans une version sans restriction du ρ-calcul se serait réduit soit à s soit à *null*. Distinguons donc les divers cas possibles afin de vérifier que cette nouvelle règle n'introduit pas d'incohérence.

1. *Le rédex* $(C \to s)\bullet B$ *est conforme à la stratégie et peut donc être réduit.*
   (a) $\mathcal{S}ol(C \ll_{\mathbb{T}} B) \neq \emptyset$: alors l'applications des diverses substitutions correspondantes laissera s inchangé, et donc la règle (*Appl*) aurait permis de déduire $\vdash A\bullet B : s$ de toute façon.
   (b) $\mathcal{S}ol(C \ll_{\mathbb{T}} B) = \emptyset$: dans ce cas, l'application de (*Appl*) donnerait $\vdash A\bullet B$: *null*. Cependant, en examinant les règles d'inférence, on se rend compte que seuls deux cas sont possibles avec $\vdash A : C \to s$:
      i. $A \equiv C' \to D$ avec $C' =_\rho C$ et $\vdash D : s$. Dans ce cas, le résultat $\mathcal{S}ol(C \ll_{\mathbb{T}} B) = \emptyset$ s'applique également au niveau des termes, et donc $A\bullet B \equiv (C' \to D)\bullet B =_\rho (C \to D)\bullet B =_\rho null$. Il s'ensuit que la préservation du type ne nous donnera que $\vdash null : s$, ce qui est de toute façon vrai.
      ii. $A \equiv f^D$ avec $D =_\rho C \to s$, et alors avec la règle (*ApplSort*) on ne fait qu'ajouter le jugement $\vdash f\bullet B : s$, qui est tout-à-fait naturel puisque ce terme est bien formé.
2. *Le rédex* $(C \to s)\bullet B$ *n'est pas réductible parce qu'il entre en conflit avec la stratégie.*
   (a) *On ne peut pas convertir* C *et* B *pour être dans un cas admis par la stratégie* : c'est justement le cas pour lequel on a établi (*ApplSort*). Le même raisonnement que pour le cas 1.(b) s'applique : le rédex $A\bullet B$ est de la forme $(C \to D)\bullet B$ ou $f^{C\to s}\bullet B$, et ne sera donc jamais réduit non plus. Lui donner le type s ne présente alors pas vraiment de risque pour la préservation du type.
   (b) C *et* B *sont convertibles pour se conformer à la stratégie* : le terme considéré devient alors un certain $(C_1 \to s)\bullet B_1$, et alors on peut se ramener au cas 1.
   Ce cas est en fait plus complexe : il se peut C et B ne coïncident jamais avec la stratégie, par exemple pour des raisons de variable libre dans B. Cependant, dans un sur-terme $(A \to (C \to s)\bullet B)\bullet D$ (qui a été "oublié" à ce niveau du typage), il peut arriver que les variables libres de B soient instanciées lorsqu'on résout le filtrage $A \ll_{\mathbb{T}} D$. Mais dans ce cas aussi, on récupérera un terme de la forme $(C_1 \to s)\bullet B_1$, qui se traite comme le cas 1.

Chapitre 3

Propriétés et problèmes du typage dans le ρ-cube

Ce chapitre résume les différentes propriétés du typage dans le ρ-cube qui ont été prouvés au cours du stage, sous les aménagements proposés dans la section 2.3. La grande majorité de ces lemmes sont des traductions de ceux existants pour le λ-cube et présentés dans la section 2.1.2. Par manque de place, les démonstrations sont reportées en annexe (page 37).

Comme déjà mentionné précédemment, on se place dans une restriction confluente, sinon tous les termes bien formés deviennent convertibles et les lemmes n'ont plus aucun sens. Si rien n'est précisé, le jugement de typage $\vdash$ dans lequel est exprimée la conclusion d'un lemme est déductible dans le même système que les jugements exprimés dans les hypothèses. Chacun des lemmes de ce chapitre est utilisable dans n'importe lequel des neuf systèmes.

3.1 Adaptation des propriétés habituelles des calculs typés

Commençons par étudier les propriétés de base des substitutions. Il nous faut tout d'abord vérifier que les substitutions préservent la conversion, ce qui nous permettra de manipuler directement les substitutions et non plus les rédex. On prouve ensuite le résultat classique de préservation du type par substitution, qui est le lemme clef pour prouver la préservation par réduction.

Ce premier lemme a pour but de montrer que l'égalité de conversion $=_\rho$ est stable par substitution. Il sera essentiellement utilisé dans la démonstration des lemmes suivants, lorsqu'il faudra tenir compte à la fois d'une substitution et d'une conversion au niveau des types. On remarquera qu'il est nécessaire puisque, si on obtient une substitution σ comme solution d'un problème de filtrage $C \ll_{\mathbb{T}} D$, on ne peut *a priori* pas différencier facilement σ des autres éléments de $\mathcal{S}ol(C \ll_{\mathbb{T}} D)$.

Lemme 3.1 (Compatibilité de la substitution avec la conversion)
Pour tous termes A, B et pour toute substitution σ telle que $\mathcal{D}om\ \sigma \cap \mathcal{R}an\ \sigma = \emptyset$, on a

$$A =_\rho B \ \Rightarrow\ \sigma A =_\rho \sigma B$$

Preuve : page 37.

Remarque 3.1 *Pour ce lemme et pour le suivant, on introduit une condition sur le domaine et le codomaine de σ car σ est* a priori *quelconque. En réalité, lorsqu'on appliquera ces deux lemmes, la substitution σ sera la solution d'une équation de filtrage $C \ll_{\mathbb{T}} D$ obtenue dans un rédex $(C \to A)\bullet D$, et donc on aura $\mathcal{D}om\ \sigma \subseteq BV(C)$; d'après la convention d'hygiène de Barendregt, il est alors impossible que les variables de $\mathcal{D}om\ \sigma$ apparaissent dans D, et donc* a fortiori *elles n'apparaissent pas dans $\mathcal{R}an\ \sigma$.*

Dans la section 2.3, nous avons adopté, pour les substitutions, une définition plus puissante : elle vérifie que le type de chaque variable apparaissant dans la substitution est cohérent avec le type du terme qui lui est substitué (déf. 15). Le lemme de substitution est alors plus simple à énoncer, mais il faut bien se souvenir que la substitution σ est typée.

Lemme 3.2 (Compatibilité de la substitution avec le jugement de typage)

$$\begin{array}{c} \forall\ \sigma, B, C \text{ tels que :} \\ \vdash B : C \\ \sigma \text{ est une substitution typée telle que } \mathcal{D}om\ \sigma \cap \mathcal{R}an\ \sigma = \emptyset \\ \text{On a} \vdash \sigma B : \sigma C \end{array}$$

Preuve : page 37.

Pour poursuivre, il faudra souvent utiliser des informations sur l'inférence du type d'un terme. Pour cela, on utilise un lemme de génération adapté au ρ-calcul, valable dans les 9 systèmes. On remarque que l'identification des abstracteurs oblige à considérer deux cas pour une abstraction $A_1 \to A_2$; de même, l'introduction de la règle ($ApplSort$) demande d'envisager deux cas pour une application $A_1 \bullet A_2$. C'est cependant rarement gênant par la suite, puisqu'à chaque fois le cas où le type est une sorte s est assez simple à traiter.

Lemme 3.3 (Génération dans le ρ-cube)
Soit un ρ-terme A typable, c'est-à-dire qu'il existe un terme B tel que $\vdash A : B$. Alors :

1. *si $A \equiv s$, alors $\exists s',\ B =_\rho s'$ et $\vdash s : s'$*
2. *si $A \equiv \alpha^C$, alors $B =_\rho C$*
3. *si $A \equiv A_1, A_2$, alors $\vdash A_1 : B$ et $\vdash A_2 : B$*
4. *si $A \equiv A_1 \to A_2$, alors :*
   - *soit $\exists s, C$ avec $\vdash A_2 : C$, $\vdash A_1 \to C : s$ et $B =_\rho A_1 \to C$;*
   - *soit $\exists s_1, s_2, C$ avec $\vdash A_2 : s_2$, $\vdash A_1 : C : s_1$ et $B =_\rho s_2$*
5. *si $A \equiv A_1 \bullet A_2$, alors :*
   - *soit $\exists C, D, E$ tels que $\vdash A_1 : C \to D$, $\vdash C : E$, $\vdash A_2 : E$ et $B =_\rho (C \to D)\bullet A_2$;*
   - *soit $\exists C, s, E$ tels que $\vdash A_1 : C \to s$, $\vdash C : E$, $\vdash A_2 : E$ et $B =_\rho s$*

Preuve : page 38.

Dans le ρ-calcul, le lemme de correction est d'une importance capitale, puisqu'on utilise quasi systématiquement la conversion après une règle ($Appl$), et on a donc régulièrement besoin de prouver que $\vdash B : s$ pour un certain type B. Dans le λ-calcul, cette nécessité n'apparaît qu'à un seul endroit de la preuve pour la préservation du type, alors que nous allons l'utiliser très régulièrement.

L'utilité du système ρECC apparaît ici pleinement : il est nécessaire au cours de la démonstration d'utiliser un jugement de la forme $\vdash s : s'$, et il faut donc pouvoir exhiber une sorte s' convenable pour toute sorte s. Ceci n'est possible que si l'ensemble des sortes est au moins infini dénombrable, ce qui nous amène à nous placer dans ρECC. En pratique, quel que soit le système (du cube) dans lequel on a l'hypothèse $\vdash A : B$, la conclusion de ce lemme n'est déductible que dans le système de types ρECC, d'où le nom de correction *faible*.

Lemme 3.4 (Correction faible)
Pour tous termes A et B, $\vdash A : B \;\Rightarrow\; \exists s,\ \vdash_{\rho ECC} B : s$.

Preuve : page 38.

Remarque 3.2 *La règle ($ApplSort$) trouve tout son intérêt dans la démonstration de ce lemme, et dans celle du théorème de préservation du type. Lorsqu'une réduction $(A \to B)\bullet C$ est utilisée au cours de ces démonstrations, c'est uniquement parce qu'on sait (grâce à un autre rédex $(A \to D)\bullet C$) que le membre gauche A et l'argument C sont conformes à la stratégie ou à la restriction retenue.*

Ce sont donc des preuves pourvues d'une certaine modularité *: elles restent valables pour toute autre stratégie que ρ_v ou ρ_{rv}^{ok}, à la seule condition que celle-ci ne pose de condition que sur le membre gauche et l'argument d'un rédex.*

On dispose maintenant de tous les outils pour démontrer que le typage est préservé par réduction. Il nous faut de plus l'hypothèse d'un filtrage unitaire, sans quoi on risque d'introduire des structures indésirables dans les types. Cette condition pourrait, semble-t-il, être relâchée, mais sous une forme peu évidente à exprimer : si on regarde la règle ($Struct$), on s'aperçoit qu'il faudrait qu'un même type soit valable pour tous les $\sigma C,\ \sigma \in \mathcal{S}ol(A \ll_{\mathbb{T}} B)$, ce qui peut être faux si jamais des variables de $\mathcal{D}om\ \sigma$ apparaissent dans le type de σC, qui dépend alors fortement de σ.

Comme pour le lemme 3.4 de correction, la conclusion ne peut être dérivée que dans ρECC. D'une part, c'est parce que le lemme de correction est utilisé à plusieurs reprises dans la démonstration ; mais d'autre part, il nous faut parfois introduire un jugement de typage $\vdash s : s'$ sur une sorte s *a priori* quelconque. Ainsi, même si on parvenait à prouver un lemme de correction forte, on n'obtiendrait pas immédiatement un lemme de préservation forte.

Théorème 3.5 (Préservation faible du type)
Si la théorie $\mathbb{T}$ a un filtrage unitaire, le type est préservé par réduction :

$$\forall A, B, \quad \vdash A : B \ \wedge\ A \mapsto_\rho A' \quad \Rightarrow \quad \vdash_{\rho ECC} A' : B$$

Preuve : page 39.

Corollaire 3.6
Ce résultat reste valable pour la relation $\mapsto_\rho^*$: il suffit de faire une récurrence sur le nombre d'occurrences de $\mapsto_\rho$ dans $A \mapsto_\rho^* A'$. Si $A \equiv A'$, le résultat est trivial ; sinon, on applique l'hypothèse de récurrence et le lemme de préservation du type sur la dernière réduction.

Il reste enfin le lemme de consistance, qui prouve que notre calcul typé a un sens logique.

Il est évident que les constantes peuvent poser problème pour ce lemme, ne serait-ce que parce qu'on peut arbitrairement considérer $a^\perp$; d'où l'interdiction des constantes en plus de la clôture. Cette restriction est tout-à-fait cohérente, puisqu'on peut considérer qu'une constante avec sa décoration de type constitue un élément du contexte, et donc contredit la clôture du terme considéré. On interdira par ailleurs la présence de *null*, puisque ce dernier peut se voir attribuer n'importe quel type, en particulier $\perp$.

Lemme 3.7 (Consistance dans le ρ-cube)
Pour tout terme A clos, sans constantes, ne contenant pas null et en forme normale,

$$\not\vdash A : \perp \quad (\perp \triangleq X^* \to X^*)$$

Preuve : page 41.

Remarque 3.3 *Le lemme précédent s'étend au cas où A n'est pas forcément en forme normale mais est faiblement normalisant. En effet, il existe alors A' tel que $A \mapsto_\rho^* A'$ et A' est en forme normale. Par préservation du type, on aura $\vdash_{\rho ECC} A' : \bot$, or le lemme précédent s'applique à A'. Notons que, comme on utilise la préservation du type, l'hypothèse de filtrage unitaire est également nécessaire.*

Pour clore cette section sur les propriétés habituelles, notons que nous n'avons pas abordé les questions de décidabilité pour la vérification et l'inférence de type. En λ-calcul, on a vu que ces deux problèmes étaient décidables à la seule condition d'adopter la présentation de Church, où les variables liées sont pourvues explicitement d'un type dans la syntaxe du calcul.

Pour le ρ-calcul, on pourrait s'imaginer que les décorations des variables et des constantes remplissent le même rôle. Cependant, l'indécidabilité trouve ici une autre cause : la relation de réduction $\mapsto_\rho$ du calcul intervient dans le typage, par la règle de conversion. Or le filtrage qui intervient dans $\mapsto_\rho$ est paramétré par une théorie $\mathbb{T}$ arbitraire, et il est assez fréquent de rencontrer des théories indécidables. Il pourrait être intéressant de déterminer si la décidabilité du typage est directement liée à celle du filtrage, ou bien si, pour certaines théories décidables, le typage reste indécidable.

3.2 Questions d'unicité

Dans un premier temps, on donne un aperçu des cas où l'unicité est en échec, au moyen de contre-exemples. Ensuite, on montre qu'un terme peut admettre un nombre non borné mais forcément fini de types distincts modulo $=_\rho$. Enfin, on prouve l'unicité du type dans $\rho 2$, et l'unicité dans $\rho\!\rightarrow$ en est une conséquence immédiate.

3.2.1 Cas d'échec de l'unicité

Le premier terme ayant plusieurs types est tiré de [CKL01] : $X^{Y^*} \to Y^*$ est typable dans ρP des deux façons suivantes.

$$\dfrac{\vdash Y^* : * \qquad \dfrac{\vdash X^{Y^*} : Y^* : * \quad \vdash * : \Box}{\vdash X^{Y^*} \to * : \Box}\ (Prod\ *, \Box)}{\vdash X^{Y^*} \to Y^* : X^{Y^*} \to *}\ (Abs) \qquad\qquad \dfrac{\vdash X^{Y^*} : Y^* : * \quad \vdash Y^* : *}{\vdash X^{Y^*} \to Y^* : *}\ (Prod\ *, *)$$

Ces inférences de type sont évidemment valides aussi dans $\rho P\underline{\omega}$, $\rho P2$ et ρCC, et montrent donc également la non-unicité du type dans ces systèmes.

On peut par ailleurs contredire l'unicité du type dans $\rho\omega = \rho 2 + \rho\underline{\omega}$, grâce au terme $X^* \to X^*$, avec les inférences suivantes :

$$\dfrac{\vdash X^* : * \qquad \dfrac{\vdash X^* : * : \Box \quad \vdash * : \Box}{\vdash X^* \to * : \Box}\ (Prod\ \Box, \Box)}{\vdash X^* \to X^* : X^* \to *}\ (Abs) \qquad\qquad \dfrac{\vdash X^* : * : \Box \quad \vdash X^* : *}{\vdash X^* \to X^* : *}\ (Prod\ \Box, *)$$

On voit bien ici que c'est le choix entre les deux règles de produit $(\Box, *)$ et $(\Box, \Box)$ qui donne deux types différents, alors que dans le cas de ρP, l'unicité était brisée parce que la règle $(*, \Box)$ permet d'appliquer la règle (Abs) une fois de plus qu'avec la seule règle $(*, *)$.

Pour être tout-à-fait cohérent, il faut vérifier que les différents types proposés sont bien distincts modulo $=_\rho$. Pour cela, il faut, comme toujours, se placer dans un ρ-calcul confluent, et alors il suffit d'observer que ces types sont tous en forme normale. Considérons par exemple $X^* \to *$ et $*$: par la propriété de Church-Rosser, s'ils étaient convertibles, alors on pourrait tous deux les réduire en un même terme ; mais comme ces deux termes ne comportent pas $\bullet$, le seul terme auquel chacun peut se réduire est lui-même. Comme $X^* \to * \not\equiv *$, on peut conclure que $X^* \to * \neq_\rho *$. On peut procéder de même pour $X^{Y^*} \to *$ et $*$, ainsi que pour tous les types dont on voudra par la suite prouver qu'ils sont distincts.

Résumons les divers échecs de l'unicité. On ne s'intéressera pas au système $\rho\underline{\omega}$, car il ne représente rien de particulier en termes de logique (il lui manque la quantification universelle sur les propositions pour pouvoir exprimer l'ordre supérieur) :

Système(s)	Cause de l'échec	Exemple
ρP	choix entre deux règles	$\vdash X^{Y^*} \to Y^* : *$ $(Prod)$ $\vdash X^{Y^*} \to Y^* : X^{Y^*} \to *$ (Abs)
$\rho\omega$	choix entre un typage dans $\rho 2$ ou dans $\rho\underline{\omega}$	$\vdash X^* \to X^* : *$ $\rho 2$ $\vdash X^* \to X^* : X^* \to *$ $\rho\underline{\omega}$
$\rho P2$, $\rho P\underline{\omega}$ ρCC, ρECC	en tant que systèmes étendant ρP (et/ou $\rho\omega$)	

FIG. 3.1 – Récapitulatif des cas d'échec de l'unicité

Enfin, remarquons que dans ρP comme dans $\rho\omega$, on peut construire arbitrairement des termes ayant un nombre n de types, $n \in \mathbb{N}$. Il suffit pour cela de compliquer légèrement les termes précédents ; les inférences de type se font de façon très similaire, en choisissant d'appliquer la règle (Abs) entre 0 et n fois puis en finissant par une règle $(Prod)$

$$\vdash_{\rho P} X_1^{Y_1^*} \to X_2^{Y_2^*} \to \ldots \to X_n^{Y_n^*} \to Y_n^* : \begin{cases} * \\ X_1^{Y_1^*} \to * \\ \ldots \\ X_1^{Y_1^*} \to X_2^{Y_2^*} \to \ldots \to X_n^{Y_n^*} \to * \end{cases}$$

$$\vdash_{\rho\omega} X_1^* \to X_2^* \to \ldots \to X_n^* \to X_n^* : \begin{cases} * \\ X_1^* \to * \\ \ldots \\ X_1^* \to X_2^* \to \ldots \to X_n^* \to * \end{cases}$$

FIG. 3.2 – Termes ayant $n+1$ types distincts, dans ρP ou dans $\rho\omega$

Cependant, si ces derniers exemples montrent que le nombre de types distincts pour un terme n'est pas borné, le lemme suivant prouve qu'il est fini, et ce dans chacun des 9 systèmes de types envisagés. Comme déjà évoqué à propos de la consistance, il faut mettre *null* à l'écart puisqu'il est susceptible d'avoir un type quelconque, et donc de générer un terme ayant un nombre infini de types.

Lemme 3.8 (Finitude du typage)
Tout ρ-terme ne contenant pas null *a un nombre fini de types distincts (modulo $=_\rho$).*

Preuve : page 43.

3.2.2 Unicité dans $\rho 2$

Après avoir vu quelques contre-exemples dans les autres systèmes, nous pouvons nous intéresser à l'unicité du type dans $\rho 2$. Ce résultat est particulièrement intéressant si on considère ce système comme un modèle étendu de ML. En effet, il inclut le polymorphisme, et le filtrage du ρ-calcul permet des motifs très expressifs (en ML, ils doivent être linéaires). Or le mécanisme de typage est fondamental en ML, puisqu'un programme n'est accepté que s'il est possible de le typer.

Nous utilisons largement les lemmes précédents, notamment la préservation du type et donc il nous faut un filtrage unitaire. Pour les raisons habituelles, nous excluons *null* car il peut donner lieu à un nombre de types arbitrairement grand.

Théorème 3.9 (Unicité du typage dans $\rho 2$)
Avec une théorie $\mathbb{T}$ ayant un filtrage unitaire, pour tout terme A ne contenant pas null :

$$\vdash_{\rho 2} A : B \quad \wedge \quad \vdash_{\rho 2} A : B' \quad \Rightarrow \quad B =_\rho B'$$

Preuve : page 43.

Corollaire 3.10 *Le typage est également unique dans $\rho\!\rightarrow$, en tant que sous-système de $\rho 2$. En effet, tout jugement de typage dérivable dans $\rho\!\rightarrow$ est immédiatement dérivable dans $\rho 2$ aussi, et donc tout terme a au moins autant de types distincts dans $\rho 2$ que dans $\rho\!\rightarrow$.*

Le type d'un programme ML est généralement assez simple : il permet principalement de s'assurer que le type des éléments passés en arguments à une fonction est bien conforme à ce que cette fonction attend. En particulier, il ne tient pratiquement pas compte des mécanismes de filtrage ; dans notre cas, nous avons défini des notions de filtrage et de conversion au niveau des types, ce qui pourrait permettre de définir un système de types plus "précis" pour ML. On peut également voir notre calcul comme une extension de ML, puisqu'il permet de définir des motifs plus généraux et des types avec une syntaxe plus proche de celle des termes ; nos résultats sont alors applicables à ML par une simple restriction du langage.

La propriété d'unicité, quant à elle, prouve la fiabilité de la méthode : en effet, s'il était possible d'attribuer deux types distincts à un même programme, il serait difficile de déterminer lequel de ces types a été réellement inféré. Ainsi, d'une machine à une autre, ou même d'un utilisateur à un autre, le même programme pourrait être traité différemment par ML, ce qui introduirait un non-déterminisme indésirable.

Conclusion et perspectives

Nous avons étudié quelques aspects typés du calcul de réécriture, qui intègre le λ-calcul et les systèmes de réécriture en un unique formalisme. Notre choix a été de considérer assez systématiquement le point du vue du λ-calcul afin de profiter de tous les travaux déjà effectués en théorie des types. Ainsi, nous avons vu qu'il fallait parfois imposer des contraintes supplémentaires (stratégies, restrictions) au calcul pour prouver des propriétés indispensables, telles que la confluence.

Toujours dans l'optique du ρ-calcul en tant qu'extension du λ-calcul, nous avons étudié un ensemble de systèmes de types calqué sur le λ-cube de Barendregt. Les principales nouveautés de ce ρ-cube sont la présence de motifs à gauche de l'abstracteur ; la possibilité d'avoir des structures dans les termes ; la contraction des abstracteurs λ et Π en un seul abstracteur $\rightarrow$, qui introduit une notion de conversion au niveau des types. Pour cette dernière raison entre autres, l'introduction d'un neuvième système ρECC, pourvu d'une hiérarchie infinie de sortes, apparaît nécessaire.

Un examen approfondi des propriétés des types ainsi formés nous a révélé deux défauts de ces systèmes : un manque de contraintes sur les types des variables instanciées lors des réductions d'une part, et une certaine incompatibilité avec les restrictions nécessaires pour assurer la confluence. Nous avons donc proposé des modifications qui résolvent ces problèmes, tout en restant cohérentes avec le reste du calcul. Nous avons motivés ces choix, et le fait que le calcul vérifie alors les propriétés usuelles sur les types confirme *a posteriori* la valeur de nos adaptations.

Ces bases étant fixées, nous avons pu démontrer que la plupart des propriétés connues du λ-calcul typé s'adaptaient au ρ-calcul. Ainsi, les lemmes décrivant la stabilité du calcul et du typage par substitution sont vérifiés sous les mêmes hypothèses. Les propriétés de correction et de préservation du type sont vérifiées dans une version faible, à savoir que leur conclusion n'est déductible que dans le système ρECC qui est le plus puissant. Enfin, la consistance du calcul est vérifiée en l'absence de constantes, ce qui correspond à une clôture stricte des termes.

Nous avons étudié à part la propriété d'unicité du type, car elle est relativement immédiate pour le λ-calcul mais pose des problèmes pour le calcul de réécriture. En effet, l'unique abstracteur $\rightarrow$ fait qu'on peut confondre, par exemple, un type dépendant et un terme. Nous avons ainsi montré que, en général, le nombre de types distincts pour un même terme était fini mais non borné. Nous avons cependant prouvé que l'unicité du type restait vraie dans le système $\rho 2$, qu'on peut voir comme une modélisation étendue de ML puisqu'il inclut les motifs du premier ordre quelconques (ML n'accepte que les motifs linéaires) et le polymorphisme.

Pour poursuivre dans cette direction, nous pourrions dans un premier temps essayer de préciser les résultats obtenus sur l'unicité du type. Par exemple, nous savons caractériser des termes ayant un nombre arbitraire de types distincts dans certains systèmes, mais étant donné un terme, nous ne connaissons aucune borne inférieure ou supérieure sur le nombre de types qu'on peut lui attribuer. Une étude plus poussée des propriétés de l'abstracteur $\rightarrow$ pourrait nous permettre

d'énumérer au moins approximativement ces types. Inversement, nous aimerions trouver une contrainte supplémentaire qui garantit l'unicité : notre conjecture est qu'un terme "vrai" n'a qu'un seul type, autrement dit qu'avec $\vdash A : B : *$, un unique B est possible.

Il nous manque également une propriété fondamentale des types, que nous n'avons pas encore étudiée : la normalisation. Celle-ci, même faible, est indispensable pour donner son sens au lemme de consistance ; elle apporte également une certaine légitimité au calcul en tant qu'outil effectif, puisqu'elle garantit la terminaison des programmes correspondants aux termes. La technique envisagée pour l'instant serait de définir une application qui associerait à chaque terme du ρ-calcul un terme d'un langage normalisant (comme λC) en conservant les réductions.

Au delà des résultats portant purement sur les types, nous espérons trouver un système de déduction en logique intuitionniste qui correspondrait à notre calcul typé, afin d'étendre l'isomorphisme de Curry-Howard. Comme le montrent les travaux de S. Cerrito et D. Kesner [CK99], les motifs se prêtent particulièrement bien à la modélisation de formules logiques ; nous voudrions donc comprendre quelle expressivité peuvent apporter le filtrage muni d'une théorie arbitraire, le non-déterminisme qui en découle et les structures qui permettent de le manipuler.

Dans le même ordre d'idée, l'échec de l'unicité pourrait se révéler très informatif. Dans sa version originale, l'isomorphisme de Curry-Howard montre un terme comme une preuve, et son type comme la proposition prouvée par cette preuve. Si un terme a plusieurs types, il faudrait donc envisager que la même preuve est valable pour plusieurs propositions distinctes, ce qui est surprenant au premier abord et demande à être approfondi.

Remerciements

Merci à Claude Kirchner et Horatiu Cirstea pour avoir proposé ce sujet et fourni un encadrement aussi agréable que motivant, que je serai ravi de retrouver en thèse.

Grazie mille à Luigi Liquori pour l'intérêt qu'il a porté à mon travail, l'aide et les discussions (parfois longues...) sur le ρ-calcul et sur les types en général.

Merci à Daniel Hirschkoff et Germain Faure de m'avoir conforté dans mon choix quand j'ai envisagé d'effectuer mon stage avec Claude et Horatiu.

Merci aux gens de Protheo, et plus généralement du Loria, pour leur accueil au sein du laboratoire.

Merci à Assia pour l'inspiration épistolaire rhotascique.

Merci maman, merci papa.

Merci aux résidents de Pere Felip Monlau pour les travaux dirigés de catalan et de castillan.

Merci à Richard, Boubou, Nico, Delphine, Amélie, Baden, Vico, Marie, pour les soirées et les week-ends nancéens.

Merci à Eugénie qui sait bien pourquoi.

Bibliographie

[Bar92] Barendregt (H.), *Handbook of Logic in Computer Science*, chap. Lambda Calculi with Types, p. 117–309. Clarendon Press, 1992.

[BCKL02] Barthe (G.), Cirstea (H.), Kirchner (C.) et Liquori (L.). « Pure patterns type systems ». To be published, 2002.

[BKN99] Bloo (R.), Kamareddine (F.) et Nederpelt (R.), « On Pi-conversion in the lambda-cube and the combination with abbreviations », *Annals of Pure and Applied Logic*, vol. 97, n° 1 – 3, 1999, p. 27 – 45.

[Bla01] Blanqui (F.), *Théorie des Types et Réécriture (Type Theory and Rewriting)*. Thèse de doctorat, Université Paris-Sud (France), 2001. Available at `http ://www.lri.fr/~blanqui`. An english version will be available soon.

[Bru70] de Bruijn (N.), « The mathematical language AUTOMATH, its usage, and some of its extensions », dans Verlag (S.), éditeur, *Symposium on Automatic Demonstration*, vol. 125 (coll. *Lecture Notes in Mathematics*), p. 29 – 61, Versailles, 1970.

[Bru72] de Bruijn (N.), « Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem », *Indagationes Mathematicae*, vol. 34, n° 5, 1972, p. 381–392.

[CH88] Coquand (T.) et Huet (G.), « The calculus of constructions », *Information and Computation*, vol. 76, 1988, p. 95 – 120.

[Chu41] Church (A.), *The Calculi of Lambda Conversion*. Princeton University Press, Princeton, 1941.

[Cir00] Cirstea (H.), *Calcul de réécriture : fondements et applications*. Thèse de Doctorat d'Université, Université Henri Poincaré - Nancy I, 2000.

[Cir01] Cirstea (H.). « Stratégies confluentes pour le ρ-calcul ». Communication personnelle, 2001.

[CK99] Cerrito (S.) et Kesner (D.), « Pattern matching as cut elimination », dans Longo (G.), éditeur, *Fourteenth Annual IEEE Symposium on Logic in Computer Science*, Trento, Italy, juillet 1999. IEEE Comp. Soc. Press.

[CK00] Cirstea (H.) et Kirchner (C.), « The typed rewriting calculus », dans *Third International Workshop on Rewriting Logic and Application*, Kanazawa (Japan), septembre 2000.

[CKL01] Cirstea (H.), Kirchner (C.) et Liquori (L.), « The Rho Cube », dans Honsell (F.), éditeur, *Foundations of Software Science and Computation Structures*, vol. 2030 (coll. *Lecture Notes in Computer Science*), p. 166–180, Genova, Italy, avril 2001.

[Coq92] Coquand (T.), « Pattern matching with dependent types », dans Nordström (B.), Pettersson (K.) et Plotkin (G.), éditeurs, *Informal proceedings workshop on types for proofs and programs*, p. 71 – 84. Dept. of Computing Science, Chalmers Univ. of Technology and Göteborg Univ., Båstad, Suède, juin 1992.

[DHK00] Dowek (G.), Hardin (T.) et Kirchner (C.), « Higher-order unification via explicit substitutions », *Information and Computation*, vol. 157, n° 1/2, 2000, p. 183–235.

[GLT89] Girard (J.-Y.), Lafont (Y.) et Taylor (P.), *Proofs and Types*, vol. 7 (coll. *Cambridge Tracts in Theoretical Computer Science*). Cambridge University Press, 1989.

[HHP87] Harper (R.), Honsell (F.) et Plotkin (G.), « A framework for defining logics », dans *Second Symposium of Logic in Computer Science*, p. 194 – 204, Ithaca, N. Y., 1987. IEEE, Washington DC.

[Hue76] Huet (G.), *Résolution d'equations dans les langages d'ordre 1,2, ...,ω*. Thèse de Doctorat d'Etat, Université de Paris 7 (France), 1976.

[Klo92] Klop (J. W.), « Term rewriting systems », dans Abramsky, Gabbay et Maibaum, éditeurs, *Handbook of Logic in Computer Science*, vol. 2. Clarendon, 1992.

[Luo90] Luo (Z.), « ECC : An Extended Calculus of Constructions », dans *Proceedings of LICS*, p. 385–395, 1990.

[Mes92] Meseguer (J.), « Conditional rewriting logic as a unified model of concurrency », *Theoretical Computer Science*, vol. 96, n° 1, 1992, p. 73–155.

[MOM93] Martí-Oliet (N.) et Meseguer (J.), « Rewriting logic as a logical and semantic framework ». Rapport technique n° SRI-CSL-93-05, SRI International, Menlo Park, California, août 1993.

[Oos90] van Oostrom (V.), « Lambda calculus with patterns ». Rapport technique, Vrije Universiteit, Amsterdam, novembre 1990.

[Plo75] Plotkin (G.), « Call by name, call by value and the λ-calculus », *Theoretical Computer Science*, vol. 1, 1975, p. 125–159.

[SU98] Sørensen (M. H.) et Urzyczyn (P.), « Lectures on the curry-howard isomorphism ». Lecture Notes n° 98/14, DIKU, Copenhagen, 1998.

Annexe : démonstrations du chapitre 3

Voici les différentes preuves. On suppose connue la technique de preuve par récurrence sur la structure du terme ou sur la structure de l'inférence de type.

Lemme 3.1 (Compatibilité de la substitution avec la conversion)
Pour tous termes A, B et pour toute substitution σ telle que $\mathcal{D}om\ \sigma \cap \mathcal{R}an\ \sigma = \emptyset$, on a

$$A =_\rho B \ \Rightarrow\ \sigma A =_\rho \sigma B$$

Preuve : On va se baser sur les égalités suivantes :

$$\sigma A =_\rho (X_1 \to \ldots \to X_n \to A)\bullet\sigma X_1\bullet\ldots\bullet\sigma X_n =_\rho (X_1 \to \ldots \to X_n \to B)\bullet\sigma X_1\bullet\ldots\bullet\sigma X_n =_\rho \sigma B$$

où les X_i forment le domaine de σ. On effectue en fait un encodage de la substitution sous forme de réductions, afin de pouvoir inclure ces dernières dans la conversion.

La seconde égalité se prouve grâce à l'hypothèse $A =_\rho B$ et en utilisant le fait que la relation de réduction est stable par contexte : $\mathsf{Ctx}[A] \mapsto \mathsf{Ctx}[A']$ si $A \mapsto A'$ (voir la figure 1.2 et le paragraphe suivant). Le seul travail à faire est donc de vérifier la première égalité, puisque la troisième est tout-à-fait similaire.

Dans le terme $(X_1 \to \ldots \to X_n \to A)\bullet\sigma X_1\bullet\ldots\bullet\sigma X_n$, comme tous les membres gauches sont simplement des variables X_i, les n problèmes de filtrage considérés sont de la forme $X_i \ll_{\mathbb{T}} \sigma X_i$, et ont donc chacun une solution unique (modulo $\mathbb{T}$) : $[X_i/\sigma X_i]$. Ce terme se réduit donc à $[X_n/\sigma X_n]...[X_1/\sigma X_1]A$; or on a supposé que $\mathcal{D}om\ \sigma \cap \mathcal{R}an\ \sigma = \emptyset$, et donc l'ordre dans lequel on effectue les substitutions n'a pas d'importance :

$(X_1 \to \ldots \to X_n \to A)\bullet\sigma X_1\bullet\ldots\bullet\sigma X_n \quad =_\rho \quad [X_n/\sigma X_n]...[X_1/\sigma X_1]A \quad \equiv \quad \sigma A$

□

Lemme 3.2 (Compatibilité de la substitution avec le jugement de typage)

$$\begin{array}{c} \forall\ \sigma, B, C \text{ tels que :} \\ \vdash B : C \\ \sigma \text{ est une substitution typée telle que } \mathcal{D}om\ \sigma \cap \mathcal{R}an\ \sigma = \emptyset \\ \text{On a} \vdash \sigma B : \sigma C \end{array}$$

Preuve : On procède par récurrence sur la dérivation de $\vdash B : C$.

(*Axiom*) $B \equiv s$ et $C \equiv s'$, donc $\sigma B \equiv B$ et $\sigma C \equiv C$, et donc immédiatement $\vdash \sigma B : \sigma C$.

(*Start*) $B \equiv \alpha^C$ et $\vdash C : s$. Si $B \in \mathcal{D}om\ \sigma$, notre définition de substitution assure que B et σB ont des types cohérents, et on a donc $\vdash \sigma(\alpha^C) : \sigma C$.
Dans le cas où $\alpha \notin \mathcal{D}om\ \sigma$ (éventuellement parce que $\alpha \in \mathcal{F}$), on peut encore appliquer l'hypothèse de récurrence pour avoir $\vdash \sigma C : \sigma s = s$, et donc $\vdash \sigma B = \sigma\alpha^C = \alpha^{\sigma C} : \sigma C$.

(*null*) $B \equiv null$ et $\vdash C : s$; par hypothèse de récurrence, $\vdash \sigma C : \sigma s = s$, d'où $\vdash \sigma B = null : \sigma C$

(*Struct*) $B \equiv B_1, B_2$, avec $\vdash B_1 : C$ et $\vdash B_2 : C$. Par hypothèse de récurrence, on a $\vdash \sigma B_1 : \sigma C$ et $\vdash \sigma B_2 : \sigma C$, et on en déduit $\vdash \sigma B_1, \sigma B_2 = \sigma B : \sigma C$

(*Abs*) $B \equiv B_1 \to B_2$ et $C \equiv B_1 \to C_2$ avec $\vdash B_2 : C_2$ et $\vdash B_1 \to C_2 : s$. Toujours par hypothèse de récurrence, $\vdash \sigma B_2 : \sigma C_2$ et $\vdash \sigma(B_1 \to C_2) : \sigma s = s$. Même si l'$\alpha$-conversion intervient, la substitution σ agira de façon identique sur B_1 dans B et dans C puisque les mêmes variables sont liées. Par abus de notation, on considérera donc que les variables liées sont renommées pour ne pas interférer avec σ, ce qui nous permet d'écrire $\vdash \sigma B = \sigma B_1 \to \sigma B_2 : \sigma B_1 \to \sigma C_2 = \sigma(B_1 \to C_2) = \sigma C$

(*Appl*) Idem avec $B \equiv B_1 \bullet B_2$ et $C \equiv (C_1 \to C_2) \bullet B_2$, en appliquant l'hypothèse de récurrence sur $\vdash B_1 : C_1 \to C_2$, $\vdash B_2 : E$ et $\vdash C_1 : E$.

(*ApplSort*) Idem avec $B \equiv B_1 \bullet B_2$ et $C \equiv s$ (on remarquera que $\sigma C = \sigma s = s$), en appliquant l'hypothèse de récurrence sur $\vdash B_1 : C_1 \to s$, $\vdash B_2 : E$ et $\vdash C_1 : E$.

(*Prod*) Idem, $B \equiv B_1 \to B_2$ et $C \equiv s_2$, récurrence sur $\vdash B_2 : s_2$, $\vdash B_1 : B_1'$ et $\vdash B_1' : s_1$ (avec $\sigma s_1 = s_1$ et $\sigma s_2 = s_2$).

(*Conv*) Ici il faut être un peu plus subtil : on a $\vdash B : C'$, $\vdash C : s$ et $C =_\rho C'$ et on peut utiliser l'hypothèse de récurrence pour affirmer $\vdash \sigma B : \sigma C'$ et $\vdash \sigma C : \sigma s = s$. Il nous reste à prouver l'égalité $\sigma C =_\rho \sigma C'$, qu'on obtient par application du lemme 3.1 de substitution modulo $=_\rho$. On peut donc appliquer (*Conv*) pour déduire $\vdash \sigma B : \sigma C$.

□

Lemme 3.3 (Génération dans le ρ-cube)
Soit un ρ-terme A typable, c'est-à-dire qu'il existe un terme B tel que $\vdash A : B$. Alors :

1. *si $A \equiv s$, alors $\exists s'$, $B =_\rho s'$ et $\vdash s : s'$*
2. *si $A \equiv \alpha^C$, alors $B =_\rho C$*
3. *si $A \equiv A_1, A_2$, alors $\vdash A_1 : B$ et $\vdash A_2 : B$*
4. *si $A \equiv A_1 \to A_2$, alors :*
   - *soit $\exists s, C$ avec $\vdash A_2 : C$, $\vdash A_1 \to C : s$ et $B =_\rho A_1 \to C$;*
   - *soit $\exists s_1, s_2, C$ avec $\vdash A_2 : s_2$, $\vdash A_1 : C : s_1$ et $B =_\rho s_2$*
5. *si $A \equiv A_1 \bullet A_2$, alors :*
   - *soit $\exists C, D, E$ tels que $\vdash A_1 : C \to D$, $\vdash C : E$, $\vdash A_2 : E$ et $B =_\rho (C \to D) \bullet A_2$;*
   - *soit $\exists C, s, E$ tels que $\vdash A_1 : C \to s$, $\vdash C : E$, $\vdash A_2 : E$ et $B =_\rho s$*

Preuve : par simple inspection des règles de typage. □

Lemme 3.4 (Correction faible)
Pour tous termes A et B, $\vdash A : B \Rightarrow \exists s, \vdash_{\rho ECC} B : s$.

Preuve : Par récurrence sur la dérivation de $\vdash A : B$.

(*Axiom*) Alors forcément $B \equiv s$, donc $\exists s'$, $\vdash_{\rho ECC} B \equiv s : s'$.

(*Start*) $A \equiv \alpha^B$ et immédiatement $\vdash B : s$.

(*null*) $A \equiv null$ et immédiatement $\vdash B : s$

(*Struct*) $A \equiv A_1, A_2$, avec $\vdash A_1 : B$ et $\vdash A_2 : B$. Par hypothèse de récurrence sur $\vdash A_1 : B$, on conclut que $\vdash_{\rho ECC} B : s$.

(*Abs*) $A \equiv A_1 \to A_2$ et $B \equiv A_1 \to B_2$ avec $\vdash A_1 \to B_2 : s$, la conclusion est donc immédiate.

(*Prod*) $A \equiv A_1 \to A_2$ et $B \equiv s_2$, donc $\exists s,\ \vdash_{\rho ECC} B \equiv s_2 : s$.

(*Appl*) $A \equiv A_1 \bullet A_2$ et $B \equiv (B_1 \to B_2) \bullet A_2$, avec $\vdash A_1 : B_1 \to B_2$, $\vdash B_1 : E$ et $\vdash A_2 : E$. L'hypothèse de récurrence sur $\vdash A_1 : B_1 \to B_2$ donne alors $\vdash_{\rho ECC} B_1 \to B_2 : s$, et donc par génération $\vdash_{\rho ECC} B_2 : s$ et $\vdash_{\rho ECC} B_1 : B_1' : s'$. On peut alors dériver le typage suivant :

$$\dfrac{\dfrac{\dfrac{\vdash_{\rho ECC} B_1 : B_1' : s' \quad \vdash_{\rho ECC} s : s''}{\vdash_{\rho ECC} B_1 \to s : s''}\ (Prod) \qquad \vdash_{\rho ECC} B_2 : s}{\vdash_{\rho ECC} B_1 \to B_2 : B_1 \to s}\ (Abs) \qquad \vdash B_1 : E \quad \vdash A_2 : E}{\vdash_{\rho ECC} (B_1 \to B_2) \bullet A_2 : s}\ (ApplSort)$$

(*ApplSort*) De même que pour (*Prod*), on sait directement que $B \equiv s$, et donc $\exists s',\ \vdash_{\rho ECC} B \equiv s : s'$.

(*Conv*) On sait dans ce cas que $\vdash B : s$...

□

Théorème 3.5 (Préservation faible du type)
Si la théorie $\mathbb{T}$ a un filtrage unitaire, le type est préservé par réduction :

$$\forall A, B, \quad \vdash A : B \ \wedge\ A \mapsto_\rho A' \quad \Rightarrow \quad \vdash_{\rho ECC} A' : B$$

Preuve : Par récurrence sur la dérivation de $\vdash A : B$.

(*Axiom*) $A \equiv s$ donc A est en forme normale et $\nexists A',\ A \mapsto_\rho A'$.

(*Start*) $A \equiv \alpha^B$, idem.

(*null*) $A \equiv null$, idem.

(*Struct*) $A \equiv A_1, A_2$, et donc $A' \equiv A_1', A_2$ avec $A_1 \mapsto_\rho A_1'$ (par exemple ; le cas $A' \equiv A_1, A_2'$ avec $A_2 \mapsto_\rho A_2'$ se traite similairement). Par génération, $\vdash A_1 : B$ et $\vdash A_2 : B$, d'où par hypothèse de récurrence, $\vdash_{\rho ECC} A_1' : B$, et donc $\vdash_{\rho ECC} A' : B$.

(*Conv*) On applique l'hypothèse de récurrence sur $\vdash A : B'$ pour obtenir $\vdash_{\rho ECC} A' : B'$, et donc on peut relancer la conversion pour avoir $\vdash_{\rho ECC} A' : B$.

(*Abs*) $A \equiv A_1 \to A_2$ et $B \equiv A_1 \to B_2$, avec $\vdash A_2 : B_2$ et $\vdash A_1 \to B_2 : s$. On distingue deux cas :

- $\mathbf{A_1 \mapsto_\rho A_1'}$: alors $A_1 \to B_2 \mapsto_\rho A_1' \to B_2$, et donc par hypothèse de récurrence $\vdash_{\rho ECC} A_1' \to B_2 : s$. On dérive alors :

$$\dfrac{\dfrac{\vdash_{\rho ECC} A_1' \to B_2 : s \quad \vdash A_2 : B_2}{\vdash_{\rho ECC} A_1' \to A_2 : A_1' \to B_2}\ (Abs) \qquad \vdash A_1 \to B_2 : s \quad A_1 \to B_2 =_\rho A_1' \to B_2}{\vdash_{\rho ECC} A_1' \to A_2 : A_1 \to B_2}\ (Conv)$$

- $\mathbf{A_2 \mapsto_\rho A_2'}$: par hypothèse de récurrence, $\vdash A_2' : B_2$, d'où :

$$\dfrac{\vdash A_1 \to B_2 : s \quad \vdash_{\rho ECC} A_2' : B_2}{\vdash_{\rho ECC} A_1 \to A_2' : A_1 \to B_2}\ (Abs)$$

(*Prod*) $A \equiv A_1 \to A_2$ et $B \equiv s_2$, avec $\vdash A_1 : B_1 : s_1$. On distingue encore les deux cas :

– $\mathbf{A_1} \mapsto_\rho \mathbf{A'_1}$: par hypothèse de récurrence, $\vdash_{\rho ECC} A'_1 : B_1 : s_1$ et donc :

$$\frac{\vdash_{\rho ECC} A'_1 : B_1 : s_1 \quad \vdash A_2 : s_2}{\vdash_{\rho ECC} A'_1 \to A_2 : s_2} \ (Prod)$$

– $\mathbf{A_2} \mapsto_\rho \mathbf{A'_2}$: par hypothèse de récurrence, $\vdash_{\rho ECC} A'_2 : s_2$ et on conclut idem.

(*Appl*) $A \equiv A_1 \bullet A_2$ et $B \equiv (B_1 \to B_2) \bullet A_2$, avec $\vdash A_1 : B_1 \to B_2$, $\vdash B_1 : E$ et $\vdash A_2 : E$. On distingue cinq cas :

– $\mathbf{A_1} \mapsto_\rho \mathbf{A'_1}$: par hypothèse de récurrence, $\vdash_{\rho ECC} A'_1 : B_1 \to B_2$ donc :

$$\frac{\vdash_{\rho ECC} A'_1 : B_1 \to B_2 \quad \vdash A_2 : E \quad \vdash B_1 : E}{\vdash A'_1 \bullet A_2 : (B_1 \to B_2) \bullet A_2} \ (Appl)$$

– $\mathbf{A_2} \mapsto_\rho \mathbf{A'_2}$: par hypothèse de récurrence, $\vdash A'_2 : E$. Ici, il faut utiliser le lemme de correction pour affirmer que $\vdash_{\rho ECC} (B_1 \to B_2) \bullet A_2 : s$, et donc :

$$\frac{(B_1 \to B_2) \bullet A_2 =_\rho (B_1 \to B_2) \bullet A'_2 \quad \vdash_{\rho ECC} (B_1 \to B_2) \bullet A_2 : s \qquad \dfrac{\vdash A_1 : B_1 \to B_2 \quad \vdash B_1 : E \quad \vdash_{\rho ECC} A'_2 : E}{\vdash_{\rho ECC} A_1 \bullet A'_2 : (B_1 \to B_2) \bullet A'_2} \ (Appl)}{\vdash_{\rho ECC} A_1 \bullet A'_2 : (B_1 \to B_2) \bullet A_2} \ (Conv)$$

– $\mathbf{A_1} \equiv \mathbf{null}$: on a un $(\nu - ko)$-rédex. Alors $A' \equiv null$, et donc il suffit d'appliquer le lemme de correction pour trouver $\vdash_{\rho ECC} B : s$ et donc $\vdash_{\rho ECC} null : B$.

– $\mathbf{A_1} \equiv \mathbf{C_1}, \mathbf{C_2}$: on a un $(\nu - ok)$-rédex, donc $A' \equiv C_1 \bullet A_2, C_2 \bullet A_2$. Par génération, $\vdash C_1 : B_1 \to B_2$ et $\vdash C_2 : B_1 \to B_2$, d'où :

$$\frac{\dfrac{\vdash C_1 : B_1 \to B_2 \quad \vdash B_1 : E \quad \vdash A_2 : E}{\vdash C_1 \bullet A_2 : (B_1 \to B_2) \bullet A_2} \ (Appl) \quad \dfrac{\vdash C_2 : B_1 \to B_2 \quad \vdash B_1 : E \quad \vdash A_2 : E}{\vdash C_2 \bullet A_2 : (B_1 \to B_2) \bullet A_2} \ (Appl)}{\vdash C_1 \bullet A_2, C_2 \bullet A_2 : (B_1 \to B_2) \bullet A_2} \ (Struct)$$

– $\mathbf{A_1} \equiv \mathbf{C_1} \to \mathbf{C_2}$: on a un ρ-rédex, et $A' \equiv \{\sigma C_2 \mid \sigma \in \mathcal{S}ol(C_1 \ll_{\mathbb{T}} A_2)\}$.
Par génération sur $\vdash C_1 \to C_2 : B_1 \to B_2$, on sait qu'il existe s, D tels que $\vdash C_2 : D$, $\vdash C_1 \to D : s$ et $B_1 \to B_2 =_\rho C_1 \to D$. Si l'équation $C_1 \ll_{\mathbb{T}} A_2$ n'a pas de solution, on peut immédiatement conclure comme pour $A_1 \equiv null$. Sinon, comme on a supposé un filtrage unitaire, $\mathcal{S}ol(C_1 \ll_{\mathbb{T}} A_2) = \{\sigma\}$, et on peut utiliser le lemme de substitution 3.2 pour affirmer que $\vdash \sigma C_2 : \sigma D$ (au passage, on remarquera qu'ici σ est bien obtenue comme solution de l'équation de filtrage $C_1 \ll_{\mathbb{T}} A_2$, et, grâce à l'hygiène, vérifie donc la condition sur son domaine et son codomaine). On déduit alors la dérivation suivante (en utilisant encore le lemme de correction pour $(B_1 \to B_2) \bullet A_2$) :

$$\frac{\vdash_{\rho ECC} \sigma C_2 : \sigma D \quad \vdash_{\rho ECC} (B_1 \to B_2) \bullet A_2 : s \quad (B_1 \to B_2) \bullet A_2 =_\rho (C_1 \to D) \bullet A_2 =_\rho \sigma D}{\vdash_{\rho ECC} \sigma C_2 : (B_1 \to B_2) \bullet A_2} \ (Conv)$$

Remarque 3.4 *L'égalité $(C_1 \to D) \bullet A_2 =_\rho \sigma D$ devrait normalement dépendre de la stratégie choisie pour assurer la confluence. Cependant, par hypothèse, le rédex $(C_1 \to C_2) \bullet A_2$ est réductible, et on a supposé dans la section 1.3 que les stratégies ne prenaient en compte que le membre gauche et l'argument ; comme on conserve C_1 et A_2, on est en droit d'effectuer la réduction vers σD.*

Remarque 3.5 *C'est ici que l'hypothèse de filtrage unitaire intervient, pour assurer qu'il n'y a qu'un seul type σD. On pourrait en fait définir une condition sur la théorie imposant que, s'il y a plusieurs σ distinctes, les types de tous les σC sont identiques (par exemple parce que D n'est pas modifié par les substitutions).*

(*ApplSort*) $A \equiv A_1 \bullet A_2$ et $B \equiv s$, avec $\vdash A_1 : B_1 \to s$, $\vdash B_1 : E$ et $\vdash A_2 : E$. Les preuves sont très similaires au cas de (*Appl*) ; on distingue cinq cas :

- $\mathbf{A_1} \mapsto_\rho \mathbf{A'_1}$: par hypothèse de récurrence, $\vdash_{\rho ECC} A'_1 : B_1 \to s$ donc :

$$\frac{\vdash_{\rho ECC} A'_1 : B_1 \to s \quad \vdash A_2 : E \quad \vdash B_1 : E}{\vdash A'_1 \bullet A_2 : s} \ (Appl)$$

- $\mathbf{A_2} \mapsto_\rho \mathbf{A'_2}$: par hypothèse de récurrence, $\vdash A'_2 : E$, donc :

$$\frac{\vdash A_1 : B_1 \to s \quad \vdash B_1 : E \quad \vdash_{\rho ECC} A'_2 : E}{\vdash_{\rho ECC} A_1 \bullet A'_2 : s} \ (Appl)$$

- $\mathbf{A_1} \equiv \mathbf{null}$: on a un $(\nu - ko)$-rédex. Alors $A' \equiv null$, et donc $\vdash_{\rho ECC} null : s$ car $\vdash_{\rho ECC} s : s'$.
- $\mathbf{A_1} \equiv \mathbf{C_1}, \mathbf{C_2}$: on a un $(\nu - ok)$-rédex, donc $A' \equiv C_1 \bullet A_2, C_2 \bullet A_2$. Par génération, $\vdash C_1 : s$ et $\vdash C_2 : s$, d'où :

$$\frac{\dfrac{\vdash C_1 : B_1 \to s \quad \vdash B_1 : E \quad \vdash A_2 : E}{\vdash C_1 \bullet A_2 : s} \ (Appl) \quad \dfrac{\vdash C_2 : B_1 \to s \quad \vdash B_1 : E \quad \vdash A_2 : E}{\vdash C_2 \bullet A_2 : s} \ (Appl)}{\vdash C_1 \bullet A_2, C_2 \bullet A_2 : s} \ (Struct)$$

- $\mathbf{A_1} \equiv \mathbf{C_1} \to \mathbf{C_2}$: on a un ρ-rédex, et $A' \equiv \{\sigma C_2 \mid \sigma \in \mathcal{S}ol(C_1 \ll_{\mathbb{T}} A_2)\}$.
  Par génération sur $\vdash C_1 \to C_2 : B_1 \to s$, on sait qu'il existe s', D tels que $\vdash C_2 : D$, $\vdash C_1 \to D : s'$ et $B_1 \to s =_\rho C_1 \to D$. Par confluence, cette dernière égalité donne $s =_\rho D$. Si l'équation $C_1 \ll_{\mathbb{T}} A_2$ n'a pas de solution, on peut immédiatement conclure comme pour $A_1 \equiv null$. Sinon, le filtrage étant unitaire, $\mathcal{S}ol(C_1 \ll_{\mathbb{T}} A_2) = \{\sigma\}$, et on utilise le lemme de substitution 3.2 pour affirmer que $\vdash \sigma C_2 : \sigma D$ (ici aussi, σ vérifie la contrainte sur son domaine et son codomaine, en tant que solution d'une équation de filtrage). On déduit alors la dérivation suivante :

$$\frac{\vdash_{\rho ECC} \sigma C_2 : \sigma D \quad \vdash_{\rho ECC} s : s'' \quad \dfrac{s =_\rho D}{s =_\rho \sigma s =_\rho \sigma D} \ (Lemme\ 3.1)}{\vdash_{\rho ECC} \sigma C_2 : s} \ (Conv)$$

□

Lemme 3.7 (Consistance dans le ρ-cube)
Pour tout terme A clos, sans constantes, ne contenant pas null et en forme normale,

$$\not\vdash A : \bot \quad (\bot \triangleq X^* \to X^*)$$

Preuve : On suppose que $\vdash A : \bot$ et on distingue des cas sur la structure de A.

$A \equiv s$: par génération, $\bot =_\rho s'$, ce qui est impossible puisque ces deux termes sont distincts et en forme normale.

$A \equiv \alpha^B$: ce cas est impossible puisqu'on a supposé A clos et sans constantes.

$A \equiv A_1, A_2$: alors par génération $\vdash A_1 : \bot$, et donc on peut se ramener récursivement à un cas où A n'est pas une structure. Remarquons que les propriétés de clôture, de normalisation, etc. de A se transmettent évidemment à A_1, et que ce dernier est strictement moins long, donc ce procédé finira forcément.

$A \equiv A_1 \bullet A_2$: en fait, ce qui nous intéresse est l'application de tête, et A_1 pourrait être une application aussi. On écrit donc plutôt $A \equiv A_1 \bullet A_2 \bullet \ldots \bullet A_n$, où A_1 n'est plus une application.

Alors A_1 n'est ni une variable ni une constante (par clôture) ; $A_1 \not\equiv s$ (car pour que A soit typable on a $\vdash A_1 : C \to D$ or $\nvdash s : C \to D$) ; enfin si A_1 est une structure, A n'est pas en forme normale.

La seule possibilité est donc que $A_1 \equiv B_1 \to B_2$, et que le rédex $(B_1 \to B_2) \bullet A_2$ soit irréductible pour des raisons de stratégie. En tant que sous-terme de A, $(B_1 \to B_2) \bullet A_2$ est typable, et il nous faut donc distinguer deux cas :

$n = 2$ et $A \equiv (B_1 \to B_2) \bullet A_2$, alors $\vdash (B_1 \to B_2) \bullet A_2 : X^* \to X^*$. Par génération, on trouve soit $X^* \to X^* =_\rho s$ (ce qui est impossible), soit $X^* \to X^* =_\rho (B_1' \to B_2') \bullet A_2$ avec $\vdash B_1 \to B_2 : B_1' \to B_2'$. Une nouvelle application de la génération à ce dernier jugement donne $B_1' \to B_2' =_\rho B_1 \to C_1$.

Récapitulons alors les égalités obtenues : $X^* \to X^* =_\rho (B_1' \to B_2') \bullet A_2 =_\rho (B_1 \to C_1) \bullet A_2$. On peut alors appliquer la propriété de Church-Rosser, mais comme $X^* \to X^*$ est en forme normale, on a $(B_1 \to C_1) \bullet A_2 \mapsto_\rho^* X^* \to X^*$. Ceci implique que le rédex $(B_1 \to C_1) \bullet A_2$ est réductible, et donc c'est aussi le cas pour le rédex $(B_1 \to B_2) \bullet A_2$ (comme toujours, si la stratégie considérée ne tient compte que du membre gauche et de l'argument). Ceci contredit le fait que A est en forme normale.

$n > 2$ et alors le sous-terme $(B_1 \to B_2) \bullet A_2 \bullet A_3$ est typable, et donc $\exists C, D$ tels que $\vdash (B_1 \to B_2) \bullet A_2 : C \to D$. Mais alors un raisonnement similaire au précédent s'applique : on va trouver $(B_1 \to C_1) \bullet A_2 \mapsto_\rho^* C \to D$, ce qui prouve que le rédex $(B_1 \to B_2) \bullet A_2$ est réductible et contredit encore une fois l'hypothèse de forme normale.

Dans les deux cas, nous avons donc abouti à une contradiction, ce qui prouve que A ne peut être une application $A_1 \bullet A_2$.

$A \equiv A_1 \to A_2$: alors par génération on trouve B_2 et s tels que $\vdash A_2 : B_2$, $\vdash A_1 \to B_2 : s$ et $A_1 \to B_2 =_\rho X^* \to X^*$.

Par Church-Rosser, on en déduit que $A_1 \mapsto_\rho^* X^*$, et donc $A_1 \equiv X^*$ puisque A_1 est en forme normale en tant que sous-terme de A. Par ailleurs $B_2 =_\rho X^*$, et donc par conversion on trouve $\vdash A_2 : X^*$. Il nous faut alors distinguer des cas sur A_2 :

$A \equiv s$: impossible, sinon on a $\vdash s : X^*$.

$A \equiv \alpha$: par hypothèse, α ne peut être une constante. Si c'est une variable, on fait rentrer ce cas dans la catégorie $A \equiv A_1 \bullet \ldots \bullet A_n$, avec $A_1 \equiv \alpha$ et $n = 1$ (on verra d'ailleurs que c'est forcément le cas).

$A_2 \equiv C_1, C_2$: il suffit de reprendre la distinction de cas avec C_1 à la place de A_2, car il a les mêmes propriétés et il est strictement moins long.

$A_2 \equiv C_1 \to C_2$: on a vu que $\vdash A_2 : X^*$. Par génération, on a donc $X^* =_\rho s$ (ce qui est impossible), ou bien $X^* =_\rho C_1 \to C_2'$ (ce qui est encore impossible). Vu son type, A_2 ne peut donc pas être une abstraction.

$A_2 \equiv C_1 \bullet C_2$: un raisonnement similaire à celui mené pour $A \equiv A_1 \bullet A_2$ s'applique, et on montre par des considérations de typage que l'application de tête est forcément réductible. Un cas cependant reste ici possible : celui où le terme de tête est une variable, car A_2 est le membre droit d'une abstraction et peut donc comporter des variables libres (si celles-ci apparaissent dans A_1). Le seul cas envisageable est donc $A_2 \equiv Y \bullet C_1 \bullet \ldots \bullet C_n$.

Mais comme A est clos, $FV(A_2) \subseteq FV(A_1)$, et comme $A_1 \equiv X^*$, on a $Y \equiv X^*$. Encore une fois, la typabilité de A_2 voudrait qu'on ait un type flèche en tête de l'application, mais $\vdash X^* : *$. Il reste donc $n = 0$ et donc $A_2 \equiv X^*$. Le jugement de typage vu plus haut donne $\vdash X^* : X^*$ ce qui est absurde.

□

Lemme 3.8 (Finitude du typage)
Tout ρ-terme ne contenant pas null *a un nombre fini de types distincts (modulo $=_\rho$).*

Preuve : Par récurrence sur la structure du terme A. En éliminant les conversions en fin de typage, on peut toujours se ramener à l'inférence "canonique" en gardant le même type modulo $=_\rho$.

$A \equiv s$ Alors (par génération) tous ses types sont convertibles au même s' (avec $\vdash s : s'$).

$A \equiv \alpha^B$ Alors (par génération) tous ses types sont convertibles à B.

$A \equiv A_1, A_2$ Par hypothèse de récurrence, A_1 et A_2 n'ont qu'un nombre fini de types ; c'est donc le cas *a fortiori* pour A puisque tous ses types sont aussi des types pour A_1 et A_2.

$A \equiv A_1 \to A_2$ On montre la finitude des deux cas possibles, et donc par réunion la finitude pour tous les types :
- Si $\vdash A_1 \to A_2 : s$, alors par génération $\vdash A_2 : s$ et donc par hypothèse de récurrence, seul un nombre fini de s est susceptible de vérifier ce jugement.
- Si $\vdash A_1 \to A_2 : A_1 \to B_2$, les différents types construits sur ce modèle ne diffèrent que par B_2. Par génération, $\vdash A_2 : B_2$ et donc par hypothèse de récurrence, seul un nombre fini de B_2 conviennent, et donc A n'a qu'un nombre fini de "types flèche".

$A \equiv A_1 \bullet A_2$ Il faut encore traiter séparément la finitude de deux cas selon la règle employée :
- Si $\vdash A_1 \bullet A_2 : (C \to D) \bullet A_2$, alors $\vdash A_1 : C \to D$. Par hypothèse de récurrence, seul un nombre fini de $C \to D$ convient, et comme les types de A ne diffèrent que par $C \to D$, ils sont également en nombre fini.
- Si $\vdash A_1 \bullet A_2 : s$, alors $\vdash A_1 : C \to s$. De même, par hypothèse de récurrence, seul un nombre fini de $C \to s$ conviennent, et donc seul un nombre fini de sortes s sont possibles.

□

Théorème 3.9 (Unicité du typage dans $\rho 2$)
Avec une théorie $\mathbb{T}$ ayant un filtrage unitaire, pour tout terme A ne contenant pas null :

$$\vdash_{\rho 2} A : B \quad \wedge \quad \vdash_{\rho 2} A : B' \quad \Rightarrow \quad B =_\rho B'$$

Preuve : On procède par récurrence sur la structure de A. Dans le cas où plusieurs types distincts sont possibles, on prouvera qu'on ne peut pas être dans $\rho 2$.

$A \equiv s$: la condition $\vdash s : s'$ du lemme de génération permet de déterminer s' de manière unique, ce qui entraîne immédiatement $B =_\rho B'$.

$A \equiv \alpha^C$: on a directement $B =_\rho C =_\rho B'$.

$A \equiv A_1, A_2$: par génération, $\vdash A_1 : B$ et $\vdash A_1 : B'$, et donc par hypothèse de récurrence $B =_\rho B'$.

$A \equiv A_1 \bullet A_2$: commençons par éliminer un cas du lemme de génération. En effet, si on suppose qu'il existe C, s tels que $\vdash A_1 : C \to s$, alors par examen des règles on s'aperçoit que nécessairement $\vdash C \to s : s'$ (dans le *même* système !), car la dernière règle d'inférence employée est forcément $(Start)$, (Abs) ou $(Conv)$. Le lemme de génération appliqué à ce dernier jugement nous donne alors $\vdash s : s'$, et comme on est dans $\rho 2$, $s \equiv *$ et $s' \equiv \Box$:

$$\dfrac{\dfrac{\dfrac{\vdash C : C' : s_1 \quad \vdash * : \Box}{\vdash C \to * : s' = \Box}\ (s_1, \Box) \qquad \cdots}{\vdash A_1 : C \to *}\ (...) \qquad \vdash C : E \quad \vdash A_2 : E}{\vdash A_1 \bullet A_2 : s = *}\ (ApplSort)$$

Mais alors la règle de produit employée pour obtenir ce jugement est de la forme $(s_1, \Box)$, donc forcément $(*, \Box)$ ou $(\Box, \Box)$; or dans $\rho 2$ on ne dispose que de $(\Box, *)$ et $(*, *)$, et donc ce jugement y est impossible.

Par génération, on trouve donc C, D, C', D' tels que $\vdash A_1 : C \to D$ et $\vdash A_1 : C' \to D'$. L'hypothèse de récurrence appliquée à A_1 donne $C \to D =_\rho C' \to D'$, et permet de conclure grâce aux égalités : $B =_\rho (C \to D) \bullet A_2 =_\rho (C' \to D') \bullet A_2 =_\rho B'$.

$A \equiv A_1 \to A_2$: il faut distinguer selon les cas rencontrés dans le lemme de génération.

- Si $\exists C, C'$ tels que $\vdash A_2 : C$ et $\vdash A_2 : C'$, alors par hypothèse de récurrence $C =_\rho C'$ et donc $B =_\rho A_1 \to C =_\rho A_1 \to C' =_\rho B'$.
- Si $\exists s_2, s_2'$ tels que $\vdash A_2 : s_2$ et $\vdash A_2 : s_2'$, alors par hypothèse de récurrence $s_2 =_\rho s_2'$ et donc $B =_\rho s_2 =_\rho s_2' =_\rho B'$.
- Dans le dernier cas, il faut remonter plus loin dans les inférences de type, qui sont représentées ci-dessous. On a d'une part $\exists s, C$ tels que $\vdash A_2 : C$, $\vdash A_1 \to C : s$ et $B =_\rho A_1 \to C$; et d'autre part $\exists s_1, s_2, D$ avec $\vdash A_1 : D : s_1$, $\vdash A_2 : s_2$ et $B =_\rho s_2$.

$$\dfrac{\vdash A_2 : C (=_\rho s_2) \qquad \dfrac{\vdash A_1 : A_1' : s' \quad \vdash C : s}{\vdash A_1 \to C : s}\ (s', s)}{\vdash A_1 \to A_2 : A_1 \to C =_\rho B}\ (Abs) \qquad\qquad \dfrac{\vdash A_1 : D : s_1 \quad \vdash A_2 : s_2}{\vdash A_1 \to A_2 : s_2 =_\rho B'}\ (s_1, s_2)$$

Par hypothèse de récurrence sur A_2, on a $C =_\rho s_2$. Par ailleurs, le lemme de génération appliqué à $\vdash A_1 \to C : s$ permet d'affirmer que $\vdash C : s$, et donc par préservation du type on obtient $\vdash_{\rho ECC} s_2 : s$ (l'hypothèse de filtrage unitaire intervient ici pour pouvoir faire appel à la préservation du type).

D'où $s_2 \equiv *$ et $s \equiv \Box$ puisque les sortes s et s_2 sont apparues dans des inférences de types hors de ρECC. Examinons alors la première inférence de type : elle utilise la règle (s', s), avec $s \equiv \Box$. De même qu'avec $A \equiv A_1 \bullet A_2$, ceci suffit à prouver qu'une telle dérivation est impossible dans $\rho 2$ puisqu'on ne dispose que de $(\Box, *)$ et $(*, *)$.

□