

Stratégies Efficaces et Modèles d'Implantation pour les Langages Fonctionnels.

François-Régis Sinot

▶ **To cite this version:**

François-Régis Sinot. Stratégies Efficaces et Modèles d'Implantation pour les Langages Fonctionnels.. Informatique et langage [cs.CL]. Ecole Polytechnique X, 2006. Français. NNT : . pastel-00001952

HAL Id: pastel-00001952
https://pastel.hal.science/pastel-00001952

Submitted on 28 Jul 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Stratégies Efficaces et Modèles d'Implantation pour les Langages Fonctionnels

THÈSE

présentée par

François-Régis Sinot

en vue d'obtenir le grade de

Docteur de l'École Polytechnique
spécialité : INFORMATIQUE

Soutenue le 19 septembre 2006 devant la commission composée de :

Directeurs :	Maribel Fernández
	Jean-Pierre Jouannaud
Rapporteurs :	René David
	Vincent van Oostrom
Examinateurs :	Martin Hyland
	Jean-Jacques Lévy

Résumé

Dans les langages fonctionnels, l'efficacité dépend crucialement du choix de la stratégie d'évaluation et d'un modèle d'implantation adapté. Nous développons d'abord un λ-calcul avec substitutions explicites qui évite les problèmes habituels liés à la substitution et à l'α-conversion, dans lequel on peut définir les stratégies usuelles, mais aussi des stratégies avec un meilleur partage de calcul. Ensuite, nous développons un modèle d'implantation efficace pour ce calcul. Pour cela, nous proposons une représentation innnovante des variables libres, d'abord dans le cadre très général de la récriture d'ordre supérieur, puis avec plus de détails dans notre cas particulier. Nous obtenons ainsi un λ-calcul avec substitutions explicites sans noms ni indices, dans lequel les termes sont annotés avec de l'information qui indique comment les substitutions doivent être propagées, et qui constitue un modèle d'implantation efficace pour nos stratégies. Des machines abstraites sont alors définies, implantées et comparées expérimentalement aux meilleurs évaluateurs connus. Finalement, nous étudions les relations entre machines abstraites traditionnelles et réseaux d'interaction, deux modèles d'implantation courants mais très différents. Plus précisément, nous montrons comment certaines stratégies peuvent être implantées dans les réseaux d'interaction d'une façon très naturelle, rapprochant ainsi deux modèles utilisés pour l'implantation de stratégies efficaces.

Abstract

In functional languages, efficiency heavily relies on the choice of an evaluation strategy and an implementation model. We first develop a λ-calculus with explicit substitutions which avoids the usual problems of substitution and α-conversion, where we can define the usual strategies, as well as some strategies with more sharing of computations. We then develop an efficient implementation model for this calculus. To this end, we propose an innovative representation of free variables, first in the very general setting of higher-order rewriting, then with more details in our particular case. We thus obtain a λ-calculus with explicit substitutions without names nor indices, in which terms are annotated with information about how substitutions should be propagated, which is a suitable implementation model for our strategies. Abstract machines are then defined, implemented, and experimentally compared to the best known evaluators. Finally, we study the relationship between traditional abstract machines and interaction nets, two common but very different implementation models. More precisely, we show how some strategies can be implemented in interaction nets in a very natural way, thus bridging the gap between two models used to implement efficient strategies.

Table des matières

Introduction **1**

I Systèmes de réduction **7**
I.1 Récriture abstraite . . . 8
I.2 Récriture de termes . . . 9
I.2.1 Termes du premier ordre . . . 9
I.2.2 Substitutions du premier ordre . . . 10
I.2.3 Récriture du premier ordre . . . 10
I.2.4 Paires critiques . . . 11
I.3 λ-calcul . . . 11
I.3.1 λ-calcul nommé . . . 11
I.3.2 Notation de de Bruijn . . . 14
I.3.3 Logique combinatoire . . . 15
I.3.4 λ-calculs avec substitutions explicites . . . 16
I.3.5 λ-calcul simplement typé . . . 17
I.3.6 PCF . . . 18
I.4 Systèmes de récriture combinatoire . . . 19
I.4.1 Termes d'ordre supérieur . . . 19
I.4.2 Récriture combinatoire . . . 19
I.4.3 Avec substitutions explicites . . . 20
I.5 Stratégies . . . 21
I.5.1 Définitions . . . 21
I.5.2 Stratégies usuelles du λ-calcul . . . 21
I.5.3 Optimalité dans le λ-calcul . . . 22
I.6 Réseaux d'interaction . . . 23

II Réduction close **25**
II.1 Introduction . . . 26
II.2 Travaux antérieurs et motivations . . . 29
II.2.1 α-conversion . . . 30
II.2.2 Propagation des substitutions . . . 31
II.3 Calculs nommés . . . 34
II.3.1 Syntaxe : λ_{c} . . . 34

II.3.2 Arguments fermés : λ_{ca} . . . 37
II.3.3 Fonctions fermées : λ_{cf} . . . 43
II.3.4 Relation avec la β-réduction . . . 52
II.3.5 Arguments et/ou fonctions fermés : λ_{cl}, λ_{caf} . . . 54
II.3.6 Canonicité . . . 55
II.4 Extensions de λ_{c} . . . 58
II.4.1 Système de types pour λ_{c} . . . 58
II.4.2 λ_{c} comme langage de programmation . . . 61
II.5 Implantation : machines abstraites . . . 63
II.5.1 Stratégies closes . . . 63
II.5.2 Efficacité . . . 65
II.6 Conclusion . . . 65

III Directeurs pour la récriture d'ordre supérieur 67
III.1 Introduction . . . 68
III.2 Position du problème . . . 68
III.3 Des variables aux directeurs . . . 70
III.3.1 Variables d'un terme . . . 70
III.3.2 Renversons la vapeur . . . 71
III.3.3 Traductions . . . 71
III.3.4 Remarques . . . 72
III.4 Maintenir les directeurs . . . 72
III.4.1 Syntaxe . . . 73
III.4.2 Réduction . . . 73
III.4.3 Localité . . . 75
III.5 Systèmes de réduction combinatoire avec directeurs . . . 76
III.5.1 Syntaxe . . . 76
III.5.2 Réduction . . . 77
III.5.3 Propriétés . . . 78
III.5.4 Localité . . . 79
III.5.5 Substitutions explicites . . . 80
III.6 λ-calcul avec substitutions explicites . . . 81
III.7 Conclusion . . . 83

IV Chaînes directrices pour le λ-calcul 85
IV.1 Introduction . . . 86
IV.2 Chaînes directrices . . . 88
IV.2.1 Rappels . . . 88
IV.2.2 Syntaxe . . . 89
IV.2.3 Compilation et décompilation . . . 91
IV.3 Calcul ouvert . . . 95
IV.3.1 Règle *Beta* . . . 95
IV.3.2 Règles de propagation . . . 96
IV.3.3 Propriétés . . . 99

IV.4 Calculs simplifiés . . . 107
IV.4.1 Syntaxe . . . 108
IV.4.2 Calcul local ouvert . . . 109
IV.4.3 Calcul clos . . . 110
IV.5 Implantation réaliste . . . 116
IV.5.1 Réduction ouverte locale . . . 116
IV.5.2 Réduction close . . . 117
IV.6 Système de types pour les calculs avec chaînes directrices . . . 117
IV.7 Machine abstraite pour l'évaluation . . . 120
IV.8 Réduction en forme normale complète . . . 123
IV.8.1 Avec des noms . . . 123
IV.8.2 Avec des directeurs . . . 124
IV.9 Résultats expérimentaux . . . 125
IV.10Conclusion . . . 127

V Stratégies du λ-calcul et réseaux d'interaction 129
V.1 Introduction . . . 130
V.2 Appel par nom . . . 131
V.2.1 Sémantique à grands pas . . . 131
V.2.2 Sémantique à petits pas . . . 132
V.2.3 Codage des termes . . . 134
V.2.4 Evaluation par interaction . . . 135
V.2.5 Propriétés . . . 138
V.3 Appel par valeur . . . 139
V.4 Termes ouverts . . . 141
V.5 Appel par nécessité . . . 142
V.5.1 Sémantique à grands pas . . . 143
V.5.2 Equivalence des sémantiques à grands pas . . . 145
V.5.3 Sémantique à petits pas . . . 147
V.5.4 Codage des termes . . . 149
V.5.5 Evaluation par interaction . . . 150
V.5.6 Propriétés . . . 152
V.6 Réduction pleinement paresseuse . . . 158
V.7 Conclusion . . . 162

Conclusions et perspectives 163

Introduction

L'informatique est une discipline récente, issue de l'invention et de la diffusion des ordinateurs, et du besoin de raisonner formellement sur ces capacités nouvelles de calcul, de communication et de gestion de l'information. Contrairement à d'autres langues, le Français a formé les mots *informatique* et *ordinateur* sur des racines bien différentes, ce qui donne plus de facilité à parler d'informatique sans parler d'ordinateur. On peut distinguer l'*informatique appliquée*, qui se préoccupe directement des ordinateurs, et l'*informatique théorique*, qui a des applications ayant trait aux ordinateurs, mais qui peut aussi se concevoir indépendamment. En s'abstrayant de son objet d'étude, l'informatique a gagné le statut de science. On peut en effet parler d'informatique sans nécessairement parler d'ordinateurs, comme on peut parler de géométrie sans parler de champ de blé.[1] Le sujet de cette thèse se situe au croisement entre théorie et pratique : elle pose un problème concret, celui de l'efficacité, dans un cadre relativement abstrait, celui des langages fonctionnels.

Programmation et abstraction

Depuis la préhistoire, l'Homme invente des outils et les perfectionne pour les rendre plus résistants, plus efficaces, mais surtout plus adaptés à son usage, à sa morphologie. L'ordinateur introduit un nouveau rapport entre l'Homme et son outil : la communication, au sens intellectuel, et pas seulement physique. L'Homme ne communique plus seulement une force par l'intermédiaire de son bras et de sa main à un silex ou un couteau, mais il communique des instructions à un ordinateur par le biais d'un *langage de programmation*. C'est alors le rôle d'un *évaluateur* (*compilateur* ou *interprète*) de traduire le programme exprimé dans ce langage spécifique en instructions directement exécutables par un ordinateur.

Comme les premiers silex taillés, les premiers langages de programmation n'étaient pas très commodes. Ils étaient très proches du concret, de l'électronique : il s'agissait de donner des instructions directement compréhensibles par le microprocesseur. Au cours des années, l'Homme a amélioré la facilité d'utilisation (on parle même d'*ergonomie*) des langages : ils ont gagné en abstraction vis-à-vis de la machine. Il y a d'abord eu la *programmation structurée*, qui consiste à remplacer les redirections (*goto statements*) par des itérations et des boucles conditionnelles. Le langage est donc devenu plus proche du langage naturel, avec des constructions correspondant à des ordres qu'un humain pourrait donner. Il est aussi devenu plus abstrait,

[1] Les premières traces de géométrie remonteraient en effet à l'époque de l'Égypte antique, où il était nécessaire de mesurer les champs pour éviter tout différend après la décrue du Nil.

au sens où le programmeur n'a plus à se préoccuper de la localisation de son programme (par exemple, des numéros de lignes ou d'étiquettes). On peut aussi y voir un gain en modularité : un morceau de code peut être réutilisé sans devoir changer certaines étiquettes ou certaines références à des numéros de lignes. On peut trouver étonnant que ces évolutions aient eu leurs lots de réfractaires, alors qu'il est normal qu'une science jeune évolue rapidement.

Une étape supplémentaire a été franchie avec les langages *déclaratifs*, où l'utilisateur n'est plus invité à donner des ordres comme précédemment (*programmation impérative*), mais à donner des spécifications du résultat qu'il désire et à laisser une certaine liberté à l'ordinateur quant aux calculs nécessaires pour parvenir à ce résultat. Il est intéressant de noter que les langages déclaratifs rencontrent beaucoup d'opposition aujourd'hui, comme la programmation structurée à ses débuts, alors que plus personne ne conteste désormais son intérêt. Une branche importante des langages déclaratifs est constituée par les langages *fonctionnels purs* dans lesquels les calculs sont entièrement effectués par des fonctions (au sens mathématique). Ceci assure en particulier la *transparence référentielle*, c'est-à-dire que $f(x)$ ne dépend que de x, ce qui n'est pas vrai dans les langages impératifs (par exemple, $f(x)$ peut dépendre de l'état de la mémoire). Une conséquence importante de cela est que l'évaluation est libre de suivre un ordre quelconque, et que le résultat ne dépendra pas de cet ordre. Ce n'est évidemment pas le cas dans les langages impératifs, lesquels imposent donc un ordre d'évaluation. L'ordre choisi s'appelle la *stratégie d'évaluation*.

La théorie au cœur des langages fonctionnels est le *λ-calcul*, inventé par Church [Chu41]. L'unique opération du λ-calcul pur est la *β-réduction*, qui consiste à remplacer une fonction appliquée à un argument (on parle alors de *rédex* ou de *radical*) par le corps de la fonction dans lequel le paramètre formel a été remplacé par l'argument (le *réduit*). Les langages fonctionnels réels diffèrent sensiblement du λ-calcul pur, mais l'essentiel de notre propos peut être étudié dans ce cadre limité.

Programmation et efficacité

La notion d'efficacité a un sens concret évident : combien de temps faut-il pour exécuter tel programme, écrit dans tel langage, exécuté par tel évaluateur, sur telle machine ? Il y a beaucoup de paramètres. Le problème de concevoir des ordinateurs rapides relève du domaine de l'*architecture* des ordinateurs. Celui d'écrire un programme (un algorithme) efficace pour résoudre un problème donné dans un langage donné s'appelle l'*algorithmique*. Ce que nous appelons dorénavant *efficacité* dans cette thèse relève de la conception de l'évaluateur, et donc, dans un cadre fonctionnel, du choix de la stratégie d'évaluation et de son implantation efficace.

Remarquons d'abord que ce terrain est glissant : comment comparer deux évaluateurs ? En général, l'un peut être plus efficace sur certains programmes, et l'autre sur certains autres. Il faut donc être prudent, mais la peur ne doit pas pour autant nous empêcher d'avancer. Nous resterons donc à un niveau qualitatif pour le moment.

Le problème de l'efficacité dépend crucialement de ce que le langage autorise. Par exemple, ce problème n'existe pas dans un langage de très bas niveau où les instructions disponibles dans le langages correspondent assez directement à des instructions de l'ordinateur. En un sens, le

problème est mis sur les épaules du programmeur. Dans les langages impératifs de haut niveau, le problème est déjà sensiblement plus complexe et de nombreux travaux y ont été consacrés. Cependant, le problème prend une autre dimension dans le cas des langages fonctionnels, car, en dehors du problème classique des optimisations, le langage laisse la liberté de l'ordre d'évaluation, c'est-à-dire de la stratégie d'évaluation.

Ce qui est surprenant, c'est que la question de la stratégie d'évaluation ne suscite pas beaucoup d'intérêt, contrairement à certaines questions d'optimisation, par exemple. En fait, le problème semble clos, mais avec deux réponses, l'une pratique et l'autre théorique, radicalement opposées.

La réponse pratique se décline en deux variantes : l'appel par valeur et l'appel par nécessité. Ces deux stratégies ont comme point commun de ne jamais réduire à l'intérieur d'une fonction, mais elles diffèrent sur l'ordre dans lequel les réductions sont effectuées. En appel par valeur, un argument est toujours réduit avant d'être passé à une fonction (donc parfois inutilement), alors qu'en appel par nécessité, un terme n'est réduit que si c'est vraiment nécessaire. En théorie, l'appel par nécessité est donc meilleur, mais il est aussi plus coûteux à implanter, donc en pratique, les avis sont partagés.

La réponse théorique s'appelle la théorie de l'*optimalité*. Pour tout programme, la stratégie optimale effectue le nombre minimal de β-réductions, l'opération élémentaire dans les langages fonctionnels, pour évaluer ce programme. Le problème est que cette notion d'opération élémentaire est trop abstraite pour correspondre à une notion concrète d'efficacité. Une autre façon d'appréhender le problème est de dire que l'optimalité a un coût.

Efficacité et optimalité

Pour le λ-calcul, l'optimalité a été définie par Lévy en 1980 [Lév80] dans un sens bien particulier (et on parle donc parfois de Lévy-optimalité pour insister sur ce sens). Elle suppose qu'il est possible de réduire simultanément, pour un coût constant, plusieurs radicaux ayant une origine commune. C'est-à-dire qu'elle suppose qu'il y a un *partage de réductions*, qui n'est pas présent dans le λ-calcul considéré comme système de récriture. Une façon de réaliser concrètement ce partage de réductions consiste en un *partage de données*. Plus précisément, il suffit d'avoir une représentation des termes du λ-calcul telle que la représentation de tous les radicaux réduits simultanément dans la stratégie optimale soit unique. Il faut pour cela se placer dans un cadre plus riche que le λ-calcul, muni en particulier d'une notion fine de partage de données.

Il a fallu dix ans de recherche active pour trouver ce cadre. Field a montré que la notion d'environnement ou de substitution explicite n'était pas assez fine pour exprimer le partage de données nécessaire à la Lévy-optimalité [Fie90]. Simultanément, Lamping a exhibé un système de récriture de graphes (en fait des réseaux d'interaction au sens de Lafont [Laf90]) réalisant effectivement ce partage [Lam90]. Le système de Lamping semble fonctionner comme par magie, plutôt grâce à des astuces que grâce à de grands principes. De plus, il génère beaucoup de réductions qui ne correspondent pas à des β-réductions, donc qui n'invalident pas l'optimalité au sens de Lévy. De nombreux travaux ont suivi d'une part pour mieux comprendre le système, en particulier pour lui donner une explication en rapport avec la

logique linéaire, d'autre part pour en améliorer les performances [GAL92, AGN96, AG98]. Ces travaux n'ont pourtant pas entièrement rempli leur mission : la stratégie optimale reste vue comme difficile à comprendre et inutilisable en pratique.

En effet, il ne faut pas confondre efficacité et Lévy-optimalité : l'optimalité correspond au nombre minimum d'étapes de β-réduction, mais ne précise rien quant à d'éventuelles autres opérations, que l'on peut qualifier d'*administratives*. On sait en particulier qu'une étape de β-réduction (arbitraire) n'a pas un coût borné en général, quelle qu'en soit l'implantation [Sta79]. Cela n'empêche pas pour autant de comparer des implantations du λ-calcul. En l'occurrence, un certain nombre de travaux [LM96, ACM00] ont montré que l'implantation de Lamping et ses variantes sont particulièrement coûteuses en nombre d'étapes administratives : il semble qu'assurer le partage très fin nécessaire pour la réduction optimale ait un coût intrinsèque incompressible.

Optimalité et optimalités

C'est pourquoi des notions d'optimalité plus faibles ont été introduites. Par exemple, Yoshida [Yos94] et Maranget [Mar91, Mar92] prouvent l'optimalité d'une stratégie proche de l'appel par nécessité pour des variantes du λ-calcul faible. Récemment, Blanc, Lévy et Maranget [BLM05] ont formalisé une notion d'étiquetage d'un λ-calcul faible, ce qui peut aussi donner lieu à une nouvelle notion d'optimalité.

On peut interpréter cela en disant simplement que la stratégie optimale dépend des règles du jeu que l'on se donne. Ici, la règle du jeu principale concerne le partage qui est autorisé ou non. La définition de Lévy ne limite pas les moyens de partager des données. C'est pour cela que c'est la vraie notion d'optimalité (la notion maximale). Mais c'est aussi pour cela qu'elle ne donne pas nécessairement lieu à une implantation efficace. Si on limite le partage autorisé par exemple à une notion d'environnement ou de substitution explicite, on ne peut pas obtenir la stratégie optimale au sens de Lévy [Fie90]. En revanche, on peut effectivement définir une notion de stratégie optimale étant données ces restrictions, et éventuellement exhiber une implantation concrète de cette stratégie.

Il y a plusieurs modèles d'implantation du λ-calcul qui donnent lieu à une notion de coût plus réaliste que la réduction optimale : c'est le cas par exemple des calculs avec substitutions explicites et de certains systèmes de réseaux d'interaction. Nous utiliserons ces deux modèles dans le cadre de cette thèse. La question de trouver un modèle de coût universel, caractérisant le coût intrinsèque de la β-réduction, reste ouverte [LM96].

Contributions de la thèse

Notre contribution principale est de poser un regard neuf sur le problème théorique et pratique de l'implantation efficace des langages fonctionnels, à un niveau assez fondamental : il ne s'agit pas ici de discuter d'optimisations mineures, mais bien de bousculer les idées reçues sur les stratégies d'évaluation. Nous avançons deux arguments majeurs en ce sens.

D'une part, nous proposons une stratégie, la *stratégie close*, qui est une stratégie ni usuelle, ni optimale. Elle se présente comme une stratégie où certaines réductions sont autorisées sous

les abstractions, elle est donc moins faible que les stratégies usuelles. Pourtant, elle évite un certain nombre des défauts habituels des stratégies fortes, en particulier celui du renommage de variables (α-conversion). Nous appuyons la définition de cette stratégie de toutes les propriétés théoriques désirables. Nous montrons aussi comment elle peut être implantée de façon efficace grâce à un *calcul avec chaînes directrices*, lui aussi appuyé de toutes les propriétés désirables. Enfin, nous justifions par des résultats expérimentaux l'intérêt de cette stratégie pour l'implantation efficace des langages fonctionnels.

D'autre part, nous proposons un cadre simple permettant de faire le lien entre deux modèles de calculs traditionnellement utilisés pour l'implantation des langages fonctionnels (les *machines abstraites* et les *réseaux d'interaction*), mais radicalement opposés. En particulier, nous illustrons ce cadre sur les stratégies usuelles et montrons en quoi cette approche pourrait permettre d'améliorer certains aspects de la stratégie close.

Implicite contre explicite

L'histoire, bien que récente, de l'informatique, et sans doute d'autres domaines, montre que des phénomènes intéressants peuvent survenir quand certains aspects (en un sens très général) sont rendus plus explicites (par exemple, la logique linéaire), ou moins implicites (par exemple, la déduction modulo). Pourtant l'idée originale est souvent très simple, mais les conséquences peuvent être très surprenantes et sortir du cadre des motivations originales.

La variation sur le degré d'explicitation est un axe transversal de cette thèse. Nous commençons d'abord par définir un λ-calcul dans lequel les substitutions sont explicites. Mais les règles de réduction de ce calcul sont conditionnelles, ce qui ne nous paraît pas satisfaisant d'un point de vue d'implantation. Nous explicitons alors la façon d'implanter ces règles en internalisant les conditions, c'est-à-dire en les faisant passer dans le calcul lui-même.

Enfin, pour implanter des stratégies sous forme de réseaux d'interaction, nous introduisons un agent spécial chargé de déclencher les réductions. En d'autres termes, nous introduisons un objet syntaxique qui correspond de façon explicite au flot d'évaluation de la stratégie.

Survol de la thèse

Le Chapitre I introduit les notions essentielles nécessaires au développement de la thèse : récriture (premier ordre et ordre supérieur), λ-calcul, stratégies et réseaux d'interaction. Il s'agit principalement de fixer les notations.

Le Chapitre II construit pas à pas une famille de λ-calculs nommés avec substitutions explicites. En partant d'un calcul standard et complet, mais qui utilise l'α-conversion, nous ajoutons des indications sur les aspects calculatoires (effacement et copie). La caractéristique importante des calculs obtenus est la possibilité d'effectuer certaines réductions sous les abstractions, couplée à l'absence d'α-conversion. Ces calculs ne sont donc pas complets, au sens où une β-réduction arbitraire ne peut pas forcément être simulée, mais nous donnons des arguments pour justifier leur intérêt : ils bénéficient de propriétés souhaitables (confluence, préservation de la normalisation forte) et permettent d'exprimer des stratégies plus efficaces que les stratégies usuelles. Ce chapitre reprend pour l'essentiel l'article [FMS05a].

Néanmoins, les règles des calculs du Chapitre II ont des conditions sur les variables libres de certains sous-termes. Ces conditions n'apparaissent pas clairement comme étant faciles à implanter avec un coût constant. Le Chapitre III propose donc une réflexion sur la représentation explicite des ensembles de variables libres propice à implanter ces calculs. Le problème est étudié dans un cadre très général : celui de la récriture d'ordre supérieur. Ce chapitre correspond à l'article [Sin05b].

Le Chapitre IV propose alors d'appliquer les résultats du Chapitre III au calcul le plus intéressant du Chapitre II. La construction est réalisée étape par étape : le procédé est d'abord appliqué à un calcul complet (où toute β-réduction peut être simulée), puis le système est simplifié jusqu'à retrouver le calcul du Chapitre II. A chaque étape, toutes les propriétés essentielles pour de tels systèmes sont prouvées. Ainsi, nous obtenons finalement une implantation efficace pour une stratégie efficace. Quelques résultats expérimentaux sont décrits. Ce chapitre s'appuie sur les articles [SFM03, FMS05c].

En résumé, les Chapitres II à IV construisent une stratégie efficace et un modèle d'implantation réaliste pour cette stratégie. Cependant, pour justifier de l'efficacité de cette stratégie, nous n'avons que deux arguments : l'intuition et l'expérimentation. Ce n'est pas très satisfaisant, mais nous manquons hélas d'outils adéquats pour parler d'efficacité, par exemple d'un modèle de coût indépendant de l'implantation. Nous avons donc voulu faire un pas important dans cette direction, en reliant le monde des machines abstraites (auquel appartient la stratégie du Chapitre IV) et le monde des réseaux d'interaction (auquel appartient la stratégie optimale), dans le Chapitre V. Plus précisément, nous codons d'une façon tout à la fois naturelle et novatrice les stratégies usuelles dans les réseaux d'interaction. Cette tâche n'est évidemment pas terminée, mais nous espérons que ces travaux donneront un éclairage nouveau au problème. Ce chapitre rassemble les résultats de [Sin05a, Sin06b, Sin06a].

En dehors du Chapitre I, les chapitres sont relativement indépendants et sont précédés d'un petit résumé. Le Chapitre II fournit la motivation principale aux Chapitres III et IV et le Chapitre IV spécialise la construction du Chapitre III au cas du calcul du Chapitre II, mais la structure de chaque chapitre en rend aisée la lecture indépendamment des autres. Le Chapitre V est motivé par les chapitres qui le précèdent, mais il est lui aussi assez indépendant.

Pendant la période consacrée à cette thèse, l'auteur a contribué à d'autres travaux. Le Chapitre V montre comment certaines restrictions des réseaux d'interaction peuvent être assouplies sans perdre de propriétés importantes. Une autre avancée dans ce sens a été présentée dans [SM05]. Dans cet article, les agents peuvent avoir des ports principaux multiples, mais des conditions naturelles assurent que l'extension est conservative, c'est-à-dire qu'un tel système peut être traduit dans les réseaux d'interaction standard.

Le Chapitre V présente de nouveaux codages du λ-calcul dans les réseaux d'interaction. Nous nous sommes également intéressés dans l'article [FMS05b] au codage dans les réseaux d'interaction d'un formalisme plus riche, le ρ-calcul [CK01]. L'enjeu est d'ajouter une notion de filtrage de motif, ce qui perturbe la localité du codage : un échec de filtrage local peut avoir des répercussions sur la sémantique globale du terme. Ce type de phénomènes n'arrive pas dans le λ-calcul. Plusieurs solutions sont proposées ; en particulier, nous proposons un cadre plus général que les réseaux d'interaction, inspiré par les bigraphes [JM03].

Chapitre I

Systèmes de réduction

Ce chapitre rappelle les notions essentielles utilisées dans le reste de cette thèse : récriture de termes, λ-calcul, récriture d'ordre supérieur, stratégies, réseaux d'interaction.

I.1 Récriture abstraite

Dans le contexte de la récriture, on manipule essentiellement des relations binaires. Quelques éléments de vocabulaires et résultats peuvent être présentés de façon abstraite : c'est ce qu'on appelle la *récriture abstraite*. On se référera à [Ter03] pour davantage de détails.

Définition I.1.1 (*Système de réduction abstrait*)

Un *système de réduction abstrait* (ARS) $\mathcal{R}$ est un couple $(\mathcal{A}, \underset{\mathcal{R}}{\longrightarrow})$ où $\mathcal{A}$ est un ensemble et $\underset{\mathcal{R}}{\longrightarrow}$ une relation binaire sur $\mathcal{A}$: $\underset{\mathcal{R}}{\longrightarrow} \subseteq \mathcal{A} \times \mathcal{A}$. On note $\rightarrow$ au lieu de $\underset{\mathcal{R}}{\longrightarrow}$ lorsqu'il n'y a pas d'ambiguïté. On note $t \rightarrow u$ si $(t,u) \in \rightarrow$ et $t \not\rightarrow u$ si $(t,u) \notin \rightarrow$. On note $\rightarrow^*$ la clôture réflexive transitive de $\rightarrow$ et $\leftarrow$ la relation inverse de $\rightarrow$, c'est-à-dire la relation telle que $u \leftarrow t$ si et seulement si $t \rightarrow u$. Enfin, la relation d'interconvertibilité est l'équivalence engendrée par $\underset{\mathcal{R}}{\longrightarrow}$, elle est généralement notée $=_{\mathcal{R}}$ (ou simplement $=$).

Définition I.1.2

On dit que $t \in \mathcal{A}$ est *en $\mathcal{R}$-forme normale* s'il n'existe pas de $v \in \mathcal{A}$ tel que $t \underset{\mathcal{R}}{\longrightarrow} v$. On dit que v est *une $\mathcal{R}$-forme normale de t* si $t \underset{\mathcal{R}}{\longrightarrow}^* v$ et v est en $\mathcal{R}$-forme normale. On dit que t est *fortement normalisable* s'il n'y a pas de réduction infinie partant de t.

Définition I.1.3

Une relation $\rightarrow$ est dite :
- *localement confluente* si et seulement si $u_1 \leftarrow t \rightarrow u_2 \Rightarrow \exists v, u_1 \rightarrow^* v \ {}^*\!\leftarrow u_2$;
- *confluente* si et seulement si $u_1 \ {}^*\!\leftarrow t \rightarrow^* u_2 \Rightarrow \exists v, u_1 \rightarrow^* v \ {}^*\!\leftarrow u_2$;
- *fortement confluente* si et seulement si $u_1 \leftarrow t \rightarrow u_2 \Rightarrow \exists v, u_1 \rightarrow v \leftarrow u_2$;
- *faiblement normalisante* si tout élément a une forme normale ;
- *fortement normalisante* s'il n'y a aucune réduction infinie.

Lemme I.1.4 (*Lemme de Newman* [New42])

Si $\rightarrow$ est localement confluente et fortement normalisante, alors $\rightarrow$ est confluente.

Remarque I.1.5

En réalité, la présentation originale de Newman n'utilise pas des relations pour représenter les systèmes de réduction abstraits ; notre formulation est cependant la version moderne habituelle du lemme de Newman.

Définition I.1.6

On dit que $\underset{1}{\longrightarrow}$ commute avec $\underset{2}{\longrightarrow}$ si et seulement si $u_1 \underset{1}{\longleftarrow} t \underset{2}{\longrightarrow} u_2 \Rightarrow \exists v, u_1 \underset{2}{\longrightarrow} v \underset{1}{\longleftarrow} u_2$.

Lemme I.1.7 (*Lemme de Hindley-Rosen* [Hin64, Ros73])

Si $\underset{1}{\longrightarrow}$ et $\underset{2}{\longrightarrow}$ sont confluentes et $\underset{1}{\longrightarrow}^*$ commute avec $\underset{2}{\longrightarrow}^*$, alors $(\underset{1}{\longrightarrow} \cup \underset{2}{\longrightarrow})^*$ est confluente.

I.2 Récriture de termes

Les systèmes de récriture de termes ou systèmes de récriture du premier ordre sont une instance importante de systèmes de récriture abstraits. Cette section est tirée de [CJ95], mais on pourra aussi se référer à [DJ89, Ter03].

I.2.1 Termes du premier ordre

Définition I.2.1 (*Signature*)

Une signature est un couple $(\mathcal{S}, \mathcal{F})$ où
- $\mathcal{S}$ est un ensemble non vide de (noms de) *sortes* ;
- $\mathcal{F}$ est un ensemble non vide de (noms de) fonctions, disjoint de $\mathcal{S}$ et muni d'une *fonction de typage* τ qui associe à chaque symbole de $\mathcal{F}$ une suite non vide d'éléments de $\mathcal{S}$.

Si $\tau(f) = (s_1, \ldots, s_n, s)$, on note $f : s_1 \times \ldots \times s_n \to s$, où n est appelé *arité* de f, $s_1 \times \ldots \times s_n$ est appelé *domaine* de f et s est appelé *codomaine* de f. Les fonctions d'arité 0 sont appelées *constantes*. On désignera par $\mathcal{F}_n$ l'ensemble des symboles de $\mathcal{F}$ d'arité n et par $\mathcal{F}_{s_1 \times \ldots \times s_n \to s}$ l'ensemble des symboles de $\mathcal{F}$ de domaine $s_1 \times \ldots \times s_n$ et de codomaine s.

Soit $\mathbb{N}$ l'ensemble des entiers naturels, $\mathbb{N}_+$ l'ensemble des entiers naturels non nuls, $\mathbb{N}_+^*$ l'ensemble des suites (ou séquences) finies d'entiers naturels non nuls, et $\epsilon \in \mathbb{N}_+^*$ la suite vide. Chaque $i \in \mathbb{N}_+$ est identifié à la suite contenant l'unique élément i. Si $p, q \in \mathbb{N}_+^*$, $p \cdot q$ dénote leur concaténation, opération interne dont la séquence vide est élément neutre à droite et à gauche. $\mathbb{N}_+^*$ est muni de la relation d'ordre $\leq_{pref}$ définie par :

$$\forall p, q \in \mathbb{N}_+^*, p \leq_{pref} q \iff \exists r \in \mathbb{N}_+^*, p \cdot r = q$$

et on dira que p est un *préfixe* de q lorsque $p \leq_{pref} q$.

Un ensemble $\mathcal{E}$ de *positions* est un sous-ensemble non vide de $\mathbb{N}_+^*$ fermé pour l'ordre $\leq_{pref}$, contenant ϵ, appelée *racine* dans ce contexte, et tel que si $p \cdot (i+1) \in \mathcal{E}$, alors $p \cdot i \in \mathcal{E}$. Les positions maximales de $\mathcal{E}$ pour l'ordre $\leq_{pref}$ sont appelées *feuilles*.

Définition I.2.2 (*Terme*)

Un *terme* sur une signature $(\mathcal{S}, \mathcal{F})$ est une application t d'un ensemble de positions noté $Pos(t)$ dans $\mathcal{F}$, qui respecte l'arité des symboles de fonction :
- $t(p) \in \mathcal{F}_0$ si et seulement si p est une feuille de $Pos(t)$;
- si $t(p) = f : s_1 \times \ldots \times s_n \to s$, alors $\forall i \in \mathbb{N}, p \cdot i \in Pos(t) \iff 1 \leq i \leq n$ et $t(p \cdot i)$ a s_i pour codomaine.

Un terme est *fini* si son ensemble de positions est fini. On notera par $\mathcal{T}(\mathcal{F})$ l'ensemble des termes finis sur le vocabulaire $\mathcal{F}$. Étant donné un ensemble $\mathcal{V} = \bigcup_{s \in \mathcal{S}} \mathcal{V}_s$ disjoint de $\mathcal{F}$ et $\mathcal{S}$, de (noms de) constantes appelées *variables*, on notera par $\mathcal{T}(\mathcal{F}, \mathcal{V})$ l'ensemble $T(\mathcal{F} \cup \mathcal{V})$. Les termes de $\mathcal{T}(\mathcal{F}) \subseteq \mathcal{T}(\mathcal{F}, \mathcal{V})$ seront dits *clos* ou *fermés*. $Var(t)$ désigne l'ensemble des variables apparaissant comme étiquette à une position de t.

On peut évidemment aussi noter un terme (fini) sous forme d'une expression parenthésée. Plus précisément, on peut définir les termes par la grammaire suivante : $t_s ::= f(t_{s_1}, \ldots, t_{s_n})$ si $f \in \mathcal{F}_{s_1 \times \ldots \times s_n \to s}$.

Définition I.2.3 (*Sous-terme*)

Étant donnés un terme t et une position $p \in Pos(t)$, le sous-terme de t à la position p, noté $t_{|p}$, est défini par :
- $Pos(t_{|p}) = \{q \in \mathbb{N}_+^*, p \cdot q \in Pos(t)\}$;
- pour tout $q \in Pos(t_{|p}), (t_{|p})(q) = t(p \cdot q)$.

On note $t \lhd u$ si t est un sous-terme de u.

Définition I.2.4 (*Remplacement de sous-termes*)

Soient u et v deux termes et p une position de u telle que v et $u_{|p}$ sont de même sorte. Le terme $u[v]_p$ obtenu par remplacement dans u du sous-terme à la position p par v est défini par :
- $Pos(u[v]_p) = \{q \in Pos(u), p \not\leq_{pref} q\} \cup \{p \cdot q, q \in Pos(v)\}$;
- $u[v]_p(q) = u(q)$ si $q \in Pos(u)$ et $p \not\leq_{pref} q$;
- $u[v]_p(p \cdot q) = v(q)$ si $q \in Pos(v)$.

I.2.2 Substitutions du premier ordre

Définition I.2.5 (*Endomorphisme*)

Un endomorphisme de $\mathcal{T}(\mathcal{F}, \mathcal{V})$ est une famille d'applications $\{h_s\}_{s \in \mathcal{S}}$ indexée par $\mathcal{S}$ telles que, pour toute sorte $s \in \mathcal{S}$, h_s est une application de $\mathcal{T}(\mathcal{F}, \mathcal{V})$ dans $\mathcal{T}(\mathcal{F}, \mathcal{V})$ telle que, pour tout $t \in \mathcal{T}(\mathcal{F}, \mathcal{V})$, si $t(\epsilon)$ est de codomaine s, alors $(h_s(t))(\epsilon)$ est aussi de codomaine s et, si $f : s_1 \times \ldots \times s_n \to s$, alors, $\forall t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, \mathcal{V}), h_s(f(t_1, \ldots, t_n)) = f(h_{s_1}(t_1), \ldots, h_{s_n}(t_n))$.

Théorème I.2.6 (*Propriété universelle*)

Pour toute application f de $\mathcal{V}$ dans $\mathcal{T}(\mathcal{F}, \mathcal{V})$, telle que, si x est de sorte s, alors $f(x)$ est de sorte s, il existe un unique endomorphisme $\hat{f}$ de $\mathcal{T}(\mathcal{F}, \mathcal{V})$ tel que $\forall x \in \mathcal{V}, \hat{f}(x) = f(x)$. On confondra donc f et $\hat{f}$.

Les endomorphismes de $\mathcal{T}(\mathcal{F}, \mathcal{V})$ sont en fait appelés *substitutions*, et on note $t\sigma$ plutôt que $\sigma(t)$ l'application d'une substitution σ à un terme t. Si σ est une substitution, on appelle *support* ou *domaine* de σ l'ensemble $Dom(\sigma) = \{x \in \mathcal{V}, x\sigma \neq x\}$. Une substitution de domaine fini sera souvent noté en extension : $\{x_1 := t_1, \ldots, x_n := t_n\}$. Lorsque σ est de domaine fini, on note aussi $VIm(\sigma)$ le sous-ensemble de $\mathcal{V}$: $VIm(\sigma) = \bigcup_{x \in Dom(\sigma)} Var(x\sigma)$.

I.2.3 Récriture du premier ordre

Définition I.2.7 (*Système de récriture*)

Une *règle de récriture* est une paire de termes de même sorte, notée $l \to r$. Un ensemble de règles de récriture $\mathcal{R} = \{l_i \to r_i\}_i$ est appelé *système de récriture* (TRS).

Définition I.2.8 (*Relation de récriture*)

Étant donné un système de récriture $\mathcal{R}$, on dit que le terme t se récrit en le terme u à la position $p \in Pos(t)$, noté $t \to_{\mathcal{R}}^{p} u$, s'il existe une règle $l \to r \in \mathcal{R}$ et une substitution σ telles que $t_{|p} = l\sigma$ et $u = t[r\sigma]_p$. On pourra omettre p et $\mathcal{R}$, mais aussi préciser la règle $l \to r$ utilisée.

I.2.4 Paires critiques

La confluence locale est décidable, donc la confluence l'est aussi, si l'on suppose la terminaison grâce au Lemme I.1.4.

Définition I.2.9 (*Paire critique*)

Deux règles $l_1 \to r_1$ et $l_2 \to r_2$ (renommées de sorte que $Var(l_1) \cap Var(l_2) = \varnothing$) forment une *paire critique* s'il existe une position p et une substitution σ telles que $l_{1|p}$ n'est pas une variable et $l_2\sigma = l_{1|p}\sigma$ (dans le cas $l_1 \to r_1 = l_2 \to r_2$, on demande aussi $p \neq \epsilon$), ou *vice versa*.

Théorème I.2.10 (*Lemme des paires critiques*)

Un système de récriture $\mathcal{R}$ est localement confluent si et seulement si ses paires critiques sont confluentes.

I.3 λ-calcul

Le λ-calcul pur est un calcul de fonctions (tous les objets sont des fonctions) inventé par Church [Chu41]. Nous en donnons ici quelques éléments, pour une présentation plus complète, voir par exemple [Bar84].

I.3.1 λ-calcul nommé

Définition I.3.1 (*λ-termes*)

L'ensemble Λ des *λ-termes* est défini par la grammaire suivante :

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

où l'on suppose que x appartient à un ensemble infini $\mathcal{V}$ de *variables*.

Un terme de la forme $t\ u$ est appelé *application*, t est alors appelé *fonction*, et u *argument* de l'application. Un terme de la forme $\lambda x.t$ est appelé *abstraction*, x est alors appelé *variable liée*, et t *corps* de l'abstraction.

Exemple I.3.2

Les λ-termes $\lambda x.x$ et $\lambda y.y$ représentent tous deux la fonction identité, souvent notée $x \mapsto x$ ou $y \mapsto y$ en mathématiques.

Définition I.3.3 (*Variables libres*)

Si t est un λ-terme, on définit inductivement l'ensemble $\mathsf{fv}(t)$ des *variables libres* de t par :
- $\mathsf{fv}(x) = \{x\}$;
- $\mathsf{fv}(t\ u) = \mathsf{fv}(t) \cup \mathsf{fv}(u)$;
- $\mathsf{fv}(\lambda x.t) = \mathsf{fv}(t) \setminus \{x\}$.

On dit que x est *libre* dans t si $x \in \mathsf{fv}(t)$. On dit que t est *clos* si $\mathsf{fv}(t) = \varnothing$, *ouvert* sinon. On dit que x est *liée* dans t si un sous-terme de t est de la forme $\lambda x.u$.

Définition I.3.4 ($\lambda\mathsf{I}$-*calcul*)

On identifie une classe de termes, appelés $\lambda\mathsf{I}$-*termes*, définis par :

$$\begin{aligned} t, u ::= {} & x \mid t\ u \\ & \mid \lambda x.t \quad \text{si } x \in \mathsf{fv}(t). \end{aligned}$$

La notion de contexte est très générale en récriture. Elle a été utilisée implicitement dans la Section I.2, où nous avons préféré utiliser des positions. Nous formalisons ici cette notion dans le cas du λ-calcul.

Définition I.3.5 (*Contexte sur les λ-termes*)

Les *contextes* sont des termes avec des trous. Plus précisément, les contextes à un trou (noté $[\,]$) sur les λ-termes sont définis par :

$$C ::= [\,] \mid \lambda x.C \mid t\ C \mid C\ t.$$

On note parfois les contextes $C[\,]$ au lieu de C pour être plus explicite.

Définition I.3.6 (*Remplissage*)

L'opération de *remplissage* d'un contexte C par un terme v est notée $C[v]$ et définie inductivement par :
- $[\,][v] = v$;
- $(\lambda x.C)[v] = \lambda x.C[v]$;
- $(t\ C)[v] = t\ (C[v])$;
- $(C\ t)[v] = (C[v])\ t$.

Définition I.3.7 (*Clôture contextuelle*)

On dit qu'une relation $\mathcal{R}'$ est définie comme la clôture contextuelle d'une relation $\mathcal{R}$ si $\mathcal{R}'$ est la plus petite relation telle que :
- $t\mathcal{R}u \Rightarrow t\mathcal{R}'u$, et
- $t\mathcal{R}'u \Rightarrow C[t]\mathcal{R}'C[u]$.

On confondra souvent $\mathcal{R}$ et $\mathcal{R}'$.

Exemple I.3.8

- $(\lambda x.(x\ [\,]))[y] = \lambda x.(x\ y)$;
- $(\lambda x.(x\ [\,]))[x] = \lambda x.(x\ x)$.

On observe donc que des variables libres dans t peuvent être liées dans $C[t]$. C'est ce qu'on appelle une *capture* de variable. Dans le λ-calcul nommé, c'est quelque chose que l'on veut souvent éviter. On contourne le problème de la façon suivante :

Définition I.3.9 (*Substitution sans capture*)

Si x est libre dans t, le résultat de la substitution de x par v dans t est noté $t\{x := v\}$ et défini inductivement par :

- $x\{x := v\} = v$;
- $y\{x := v\} = y$ si $x \neq y$;
- $(t\ u)\{x := v\} = (t\{x := v\})\ (u\{x := v\})$;
- $(\lambda y.t)\{x := v\} = \lambda y.(t\{x := v\})$ si $x \neq y$ et $y \notin \mathsf{fv}(v)$.

Exemple I.3.10

- $(\lambda x.(x\ z))\{z := y\} = \lambda x.(x\ y)$;
- $(\lambda x.(x\ z))\{z := x\}$ n'est pas défini ;
- $(\lambda x'.(x'\ z))\{z := x\} = \lambda x'.(x'\ x)$.

Dans l'exemple ci-dessus, on ne veut pas avoir $(\lambda x.(x\ z))\{z := x\} = \lambda x.(x\ x)$ (capture), ce qui amène la substitution à ne pas être bien définie pour tous les termes. Mais les termes $\lambda x.(x\ z)$ et $\lambda x'.(x'\ z)$ sont moralement les mêmes (ils représentent la même fonction). Il nous faut donc considérer les λ-termes modulo une relation d'équivalence plus souple que l'égalité syntaxique.

Définition I.3.11 (*α-conversion*)

L'*α-conversion* ou *α-équivalence* est la clôture symétrique, réflexive, transitive et contextuelle de la relation :

$$\lambda x.t =_\alpha \lambda y.t\{x := y\} \quad \text{si } y \notin \mathsf{fv}(t).$$

Exemple I.3.12

$(\lambda x.(x\ z))\{z := x\} =_\alpha (\lambda x'.(x'\ z))\{z := x\} = \lambda x'.(x'\ x)$.

Désormais, on considérera donc que l'égalité entre termes s'entend modulo α-équivalence et non syntaxiquement. Si t est un λ-terme, $\hat{t}$ est un terme α-équivalent à t dans lequel toutes les variables liées sont renommées en variables *fraîches* (c'est-à-dire n'apparaissant pas dans le champ du discours). Dans un contexte donné, on peut toujours choisir d'appliquer $\hat{\cdot}$ à tous les termes, de sortes que toutes les variables liées ont des noms différents : c'est la *convention de Barendregt*.

Définition I.3.13 (*β-réduction*)

La *β-réduction* est la clôture contextuelle de la relation :

$$(\lambda x.t)\ u \to_\beta t\{x := u\}.$$

On appelle *formes normales de tête faibles* les termes de la forme $\lambda x.t$ ou $x\ t_1 \dots t_n$. Les formes normales de tête faibles closes sont donc les termes de la forme $\lambda x.t$ et sont parfois appelées *valeurs*. On dit que v est une forme normale de tête faible de t si v est une forme normale de tête faible et $t \to_\beta^* v$.

Théorème I.3.14

La relation de réduction $\to_\beta$ est confluente.

I.3.2 Notation de de Bruijn

L'utilisation de noms pour les variables oblige à considérer les termes modulo α-conversion. Cela complique les choses d'un point de vue pratique : effectuer une substitution de la forme $(\lambda y.t)\{x := v\}$ peut avoir un coût qui dépend du nommage de v. Cela n'est clairement pas un bon modèle pour les implantations, par exemple. Une façon standard de résoudre le problème est de représenter les variables par des indices de de Bruijn :

Définition I.3.15 (*λ-termes avec indices de de Bruijn*)

L'ensemble des *λ-termes avec indices de de Bruijn* est défini par la grammaire suivante, où $n \geq 1$ est un entier naturel :

$$t, u\ ::=\ n \mid \lambda t \mid t\ u.$$

Exemple I.3.16

- $\lambda 1$ correspond à la fonction identité $\lambda x.x$;
- $\lambda\lambda(1\ 2)$ correspond à $\lambda x.\lambda y.(y\ x)$.

L'ensemble des λ-termes avec indices de de Bruijn est clairement en bijection avec l'ensemble des classes d'α-équivalence des λ-termes.

Définition I.3.17 (*β-réduction*)

La *β-réduction* est la clôture contextuelle de la relation :

$$(\lambda t)\ u \to_\beta t\{1 := u\}.$$

Définition I.3.18 (*Substitution*)

La substitution est définie inductivement par :

- $n\{n := v\} = \mathcal{U}_0^n(v)$;
- $m\{n := v\} = m$ si $m < n$;

- $m\{n := v\} = m - 1$ si $m > n$;
- $(t\ u)\{n := v\} = (t\{n := v\})\ (u\{n := v\})$;
- $(\lambda t)\{n := v\} = \lambda(t\{n + 1 := v\})$.

Où $\mathcal{U}_i^n(t)$ effectue la mise à jour des indices pour éviter les captures :

- $\mathcal{U}_i^n(t\ u) = \mathcal{U}_i^n(t)\ \mathcal{U}_i^n(u)$;
- $\mathcal{U}_i^n(\lambda t) = \lambda \mathcal{U}_{i+1}^n(t)$;
- $\mathcal{U}_i^n(m) = m$ si $m \leq i$;
- $\mathcal{U}_i^n(m) = m + n - 1$ si $m > i$.

La notation de de Bruijn permet donc de définir précisément les opérations du λ-calcul, de façon sensiblement plus constructive qu'en faisant appel au choix d'un représentant adéquat dans une classe d'α-équivalence. Cependant, elle a l'inconvénient d'être peu lisible et d'introduire un certain coût algorithmique dans la substitution. Nous reviendrons sur ce point.

I.3.3 Logique combinatoire

Pour éliminer les problèmes liés au nommage des variables, il existe une alternative plus radicale aux indices de de Bruijn : supprimer les notions-mêmes de variable et d'abstraction. Plus précisément, il s'agit de se donner un certain nombre de termes clos de base (les *combinateurs*) et de ne considérer que des termes construits à l'aide de l'application et de ces combinateurs. On parle aussi de *Logique Combinatoire*. Voir [Bar84] pour plus de détails.

Définition I.3.19

La logique combinatoire est un système de récriture dont les termes sont construits à l'aide de l'application (binaire) et d'un ensemble de constantes (appelées combinateurs) contenant au moins S et K, et avec une règle de récriture par constante, dont :

$$\mathsf{K}\,x\,y \to x$$
$$\mathsf{S}\,x\,y\,z \to x\,z\,(y\,z)$$

L'intérêt de la logique combinatoire est qu'elle correspond au λ-calcul dans un sens que nous allons préciser un peu, tout en évitant d'entrer dans les détails techniques. On peut tout d'abord traduire les termes de la logique combinatoire en λ-termes (voir [ÇH98]) en utilisant en particulier les termes suivants (en surchargeant un peu les notations).

Définition I.3.20

Les λ-termes suivants sont des représentations adéquates des combinateurs K et S de la logique combinatoire, et sont aussi appelés combinateurs :

- $\mathsf{K} = \lambda x.\lambda y.x$
- $\mathsf{S} = \lambda x.\lambda y.\lambda z.x\ z\ (y\ z)$

On peut aussi définir les combinateurs du λ-calcul suivants :

- $\mathsf{I} = \lambda x.x$
- $\mathsf{B} = \lambda x.\lambda y.\lambda z.x\ (y\ z)$
- $\mathsf{C} = \lambda x.\lambda y.\lambda z.x\ z\ y$

Il est alors clair que les réductions de la logique combinatoire sont bien traduites en β-réductions. Mais dans l'autre sens, on peut également traduire un λ-terme en terme de la logique combinatoire, mais toutes les réductions du λ-calcul n'ont pas leur pendant en logique combinatoire. Cela peut s'interpréter simplement en disant que la logique combinatoire n'autorise pas les réductions sous ce qui correspond aux abstractions. Il y a alors deux approches pour donner une correspondance plus précise.

- L'approche décrite dans [Bar84] et attribuée à Curry consiste à étendre la logique combinatoire par de nouveaux axiomes permettant de simuler la réduction sous des abstractions.
- L'approche décrite dans [ÇH98], qui reprend [How70, Hin77], consiste à restreindre la règle β du λ-calcul pour correspondre plus fidèlement à la réduction de la logique combinatoire. Cette restriction consiste à n'autoriser une β-réduction que si le rédex est clos. Elle correspond donc exactement au calcul λ_{cl} défini dans la Section II.3.5.

Définition I.3.21

Un ensemble de combinateurs $\mathcal{C}$ est *complet* pour $\mathcal{A} \subseteq \Lambda$ si pour tout $t \in \mathcal{A}$ clos, il existe un terme u construit à l'aide de l'application et des combinateurs de $\mathcal{C}$ tel que $t =_\beta u$.

Si $\mathcal{C}$ est complet pour $\mathcal{A}$, on peut donc voir $\mathcal{A}$ comme un système de récriture du premier ordre dont les termes sont construits avec les symboles de $\mathcal{C}$ et l'application, à la condition, bien sûr, de se restreindre aux termes clos de $\mathcal{A}$.

Théorème I.3.22 (*Complétude*)

- $\{\mathsf{S}, \mathsf{B}, \mathsf{C}, \mathsf{K}\}$ est complet pour le λ-calcul ;
- $\{\mathsf{S}, \mathsf{K}\}$ est complet pour le λ-calcul ;
- $\{\mathsf{S}, \mathsf{B}, \mathsf{C}\}$ est complet pour le $\lambda\mathsf{I}$-calcul ;
- il existe un ensemble à un combinateur complet pour le $\lambda\mathsf{I}$-calcul ;
- il existe un ensemble à un combinateur complet pour le λ-calcul.

I.3.4 λ-calculs avec substitutions explicites

Dans le λ-calcul, aussi bien nommé qu'en notation de de Bruijn, la substitution est définie en dehors du système de calcul : on parle alors de méta-opération. Qui plus est, elle est définie de façon extensionnelle (par des égalités) et traitée de façon implicite (elle n'est pas comptée dans le coût d'une réduction). Dire comment la substitution est réalisée et quel coût doit lui être attribué est essentiellement une question d'implantation, et dépend crucialement de la représentation des termes. Par exemple, certaines machines abstraites implantent la substitution à l'aide d'environnements [CCM87, Cur91] (pour certaines stratégies particulières).

Une autre façon de donner un sens calculatoire à la substitution est de transformer sa définition extensionnelle en un système de récriture de termes, d'une part en enrichissant la structure des termes avec une construction correspondant à la substitution (alors qualifiée d'*explicite*), d'autre part en remplaçant les égalités définissant la substitution par un ensemble de règles de récriture. On parle alors de *λ-calcul avec substitutions explicites*.

La littérature abonde en λ-calculs avec substitutions explicites divers et variés. Il est d'usage de faire remonter cette tradition au calcul $\lambda C\xi\phi$ de de Bruijn [Bru78] (voir [BBLRD96]

pour une présentation moderne), même si c'est le $\lambda\sigma$-calcul [ACCL91] qui a véritablement popularisé l'idée. La motivation première de ces calculs et de certains autres (par exemple [Yos94, AFM+95, HMP98, Lan98, Nad02]) est de contrôler le processus de substitution dans les implantations. A ce titre, les indices de de Bruijn ont donc souvent la préférence (bien qu'il y ait quelques exceptions, voir par exemple [Ros96]).

L'approche que nous développerons au Chapitre II tranche sensiblement avec ces travaux, au sens où nous adopterons un calcul nommé tout en évitant tout besoin d'α-conversion. Nous ne décrivons donc pas davantage les calculs existants et incitons le lecteur à se reporter à la construction progressive du calcul du Chapitre II.

I.3.5 λ-calcul simplement typé

Les langages de programmation fonctionnels diffèrent du λ-calcul pur sur deux points. D'une part, un certain nombre de constructions syntaxiques sont ajoutées pour faciliter la vie du programmeur, par exemple les entiers et des fonctions arithmétiques. Nous donnerons plus de détails sur ce point dans la section suivante (I.3.6). Pour le moment, nous considérerons seulement que nous ajoutons aux termes une constante générique $\star$. D'autre part, un *système de type* restreint les termes à une certaine classe de termes dits *bien typés*. Cette restriction permet de détecter statiquement un certain nombre d'erreurs, par exemple un entier appliqué à quelque chose ne formera pas un terme bien typé. Une autre conséquence est que tous les termes normalisent fortement (voir le Théorème I.3.25). Ce dernier point est en général trop restrictif, c'est pourquoi on ajoute souvent une constante (bien typée par définition) qui autorise la récursion (donc la non-terminaison) de façon structurée (voir Section I.3.6). Nous présentons ici sommairement le λ-calcul simplement typé, qui est le système de type le plus simple pour le λ-calcul.

Définition I.3.23 (*Types simples*)

Les types simples sont définis par la grammaire suivante :

$$A, B \ ::= \ \top \mid A \to B.$$

On définit les environnements (notés Γ, Δ, *etc.*) comme des fonctions des variables dans les types simples, mais on adopte plutôt la notation de listes d'associations $x_1 : A_1, \ldots, x_n : A_n$.

Définition I.3.24 (*Termes bien typés*)

Un terme t est bien typé s'il existe un environnement Γ et un type simple A tel que $\Gamma \vdash t : A$ soit déductible dans le système suivant :

$$\frac{}{\vdash \star : \top}\ (\text{Const}) \qquad \frac{}{x : A \vdash x : A}\ (\text{Var})$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ (\text{Abs}) \qquad \frac{\Gamma \vdash t : A \to B \quad \Delta \vdash u : A}{\Gamma, \Delta \vdash t\ u : B}\ (\text{App})$$

Dans les règles de la Définition I.3.24 un certain nombre de règles dites *structurelles* ont été omises, conformément à l'usage. Nous reviendrons sur ce point dans la Section II.4.1.

THÉORÈME I.3.25

Tout terme simplement typable est fortement normalisable.

I.3.6 PCF

Le λ-calcul simplement typé défini à la Section I.3.5 est étendu en un langage de programmation fonctionnel minimaliste, PCF, par l'ajout de nouvelles constantes, règles de typage et règles de réduction.

DÉFINITION I.3.26 (*Termes de PCF*)

L'ensemble des termes de PCF est défini par la grammaire suivante :

$$\begin{aligned} t, u, b ::= {} & x \mid \lambda x.t \mid t\ u \\ & \mid n \mid \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{suc}(t) \mid \mathsf{pred}(t) \\ & \mid \mathsf{zer}(t) \mid \mathsf{if}(b, t, u) \mid \mathsf{fix}(t). \end{aligned}$$

DÉFINITION I.3.27 (*Typage de PCF*)

Le système de type de la Définition I.3.24 est étendu par les règles suivantes :

$$\frac{}{\Gamma \vdash n : \mathsf{nat}} \quad \frac{}{\Gamma \vdash \mathsf{tt} : \mathsf{bool}} \quad \frac{}{\Gamma \vdash \mathsf{ff} : \mathsf{bool}}$$

$$\frac{\Gamma \vdash t : \mathsf{nat}}{\Gamma \vdash \mathsf{suc}(t) : \mathsf{nat}} \quad \frac{\Gamma \vdash t : \mathsf{nat}}{\Gamma \vdash \mathsf{pred}(t) : \mathsf{nat}} \quad \frac{\Gamma \vdash t : \mathsf{nat}}{\Gamma \vdash \mathsf{zer}(t) : \mathsf{bool}}$$

$$\frac{\Gamma \vdash b : \mathsf{bool} \quad \Gamma \vdash t : A \quad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{if}(b, t, u) : A} \quad \frac{\Gamma \vdash t : A \to A}{\Gamma \vdash \mathsf{fix}(t) : A}$$

DÉFINITION I.3.28 (*Évaluation de PCF*)

Les règles suivantes sont ajoutées à la règle β :

$$\begin{aligned} \mathsf{fix}(t) &\to t(\mathsf{fix}(t)) & \mathsf{suc}(n) &\to n+1 \\ \mathsf{pred}(0) &\to 0 & \mathsf{pred}(n+1) &\to n \\ \mathsf{zer}(0) &\to \mathsf{tt} & \mathsf{zer}(n+1) &\to \mathsf{ff} \\ \mathsf{if}(\mathsf{tt}, t, u) &\to t & \mathsf{if}(\mathsf{ff}, t, u) &\to u \end{aligned}$$

I.4 Systèmes de récriture combinatoire

Les systèmes de récriture du premier ordre et le λ-calcul sont deux formalismes complémentaires : les premiers offrent une grande souplesse dans les symboles et règles utilisés, alors que le second permet de manipuler des fonctions. Il est donc naturel de vouloir combiner ces systèmes. C'est ce qu'on appelle la *récriture d'ordre supérieur*. En fait, la Section I.3.6 est un exemple intuitif d'un tel système. Le rôle de cette section est donc de donner une définition formelle d'une catégorie particulière de systèmes de récriture d'ordre supérieur : les systèmes de récriture combinatoire (CRS).

Nous renvoyons le lecteur à [Klo80, KOR93] pour une présentation détaillée des CRS. Les idées importantes sont l'utilisation de métavariables avec arité et d'une abstraction générique. Nous donnons une courte présentation des systèmes de réduction combinatoire. La présentation est inspirée principalement de Klop *et al.* [KOR93].

I.4.1 Termes d'ordre supérieur

Étant donné un ensemble de *variables* $\mathcal{V} = \{x, y, \ldots\}$, un ensemble de *symboles de fonctions* $\mathcal{F} = \{f^n, g^m, \ldots\}$, un ensemble de *métavariables* (avec arité) $\mathcal{MV} = \{Z^n, Y^m, \ldots\}$, alors l'ensemble des *métatermes* $\mathcal{MT}$ est défini par :

$$t \ ::= \ x \mid [x]t \mid f^n(t_1, \ldots, t_n) \mid Z^n(t_1, \ldots, t_n).$$

L'indice n sur les symboles fonctionnels et les métavariables s'appelle l'*arité* de ce symbole ; il est souvent omis quand il n'y a pas d'ambiguïté. Nous disons que la construction $[x]t$ *abstrait* x dans t, et les notions de variables *libres* et *liées* sont définies comme d'habitude selon cette notion d'abstraction (une occurrence d'une variable x est liée si elle est dans la portée d'un *abstracteur* $[x]$ et libre sinon). Un métaterme sans variable libre est dit *clos* ou *fermé* (et *ouvert* sinon). Les métatermes sont considérés égaux modulo le renommage de variables liées (*α-conversion*) et nous travaillerons sous la convention de Barendregt [Bar84]. Un *terme* est un métaterme sans occurrence de métavariable.

I.4.2 Récriture combinatoire

Définition I.4.1 (*Règle de récriture combinatoire*)

> Une *règle de récriture* est une paire $l \to r$ de métatermes fermés où l est un terme *construit* (c'est-à-dire qui commence par un symbole fonctionnel) et tels que les métavariables qui apparaissent dans r apparaissent également dans l et les métavariables de l apparaissent seulement sous la forme $Z(x_1, \ldots, x_n)$ où les x_i sont deux à deux distincts.

Définition I.4.2 (*Relation de réduction*)

> Nous disons que t se récrit en u, noté $t \to u$, par la règle $l \to r$, s'il y a une position p dans l'arbre de syntaxe de t et une substitution ς tels que $t|_p = l\varsigma$ et $u = t[r\varsigma]_p$.

Evidemment, la notion de substitution doit être redéfinie dans ce cadre. Un certain soin doit être pris pour éviter la capture de nom dans la substitution, comme dans le λ-calcul et contrairement aux TRS.

Définition I.4.3 (*Substitution d'ordre supérieur*)

Les substitutions assignent à chaque métavariable n-aire un *substitut* n-aire :

$$\varsigma(Z^n) = \underline{\lambda}(x_1, \ldots, x_n).t.$$

Les substitutions sont prolongées homomorphiquement aux métatermes comme suit (rappelons que les métavariables n-aires ne sont pas des métatermes si $n > 0$, elles ont besoin d'arguments, et que les variables de base se comportent comme des constantes pour les substitutions d'ordre supérieur) :

$$\begin{cases} x\varsigma = x; \\ ([x]t)\varsigma = [x](t\varsigma); \\ f(t_1, \ldots, t_n)\varsigma = f(t_1\varsigma, \ldots, t_n\varsigma); \\ Z(t_1, \ldots, t_n)\varsigma = \varsigma(Z)(t_1\varsigma, \ldots, t_n\varsigma). \end{cases}$$

Les substituts génèrent immédiatement une substitution simultanée de la façon suivante : si $\varsigma(Z) = \underline{\lambda}(x_1, \ldots, x_n).t$, alors $\varsigma(Z)(t_1, \ldots, t_n) = t\{x_1 := t_1, \ldots, x_n := t_n\}$ (avec la notion implicite habituelle de substitution modulo α-conversion).

Avec les substituts, nous sommes proches du sol familier du λ-calcul, évitant ainsi les captures de variables. Il suffit de dire que nous renommons suffisamment à la fois les substituts et les métatermes apparaissant dans la règle de récriture de la façon habituelle. Voir [KOR93] pour plus de détails.

I.4.3 Avec substitutions explicites

Comme dans le λ-calcul, une étape de réduction dans un CRS n'est pas une bonne unité de mesure de complexité, et il est naturel de vouloir définir une notion de CRS avec substitutions explicites. La qualité plaisante des CRS à cet égard est que, suivant Bloo et Rose [BR96], le formalisme ainsi nommé des CRS avec substitutions explicites (ou ESCRS) est en fait une sous-classe des CRS.

Plus précisément, un CRS est un ESCRS si toutes les applications de métavariables dans le membre droit de chaque règle de récriture est sous la forme $Z(x_1, \ldots, x_n)$ tel que $Z(x_1, \ldots, x_n)$ apparaît également dans le membre gauche de cette même règle. Les ESCRS évitent ainsi l'utilisation de l'outil puissant de la substitution d'ordre supérieur.

De plus, il y a une façon systématique d'*expliciter* (en nous référant à la formulation de [BR96]) un CRS, c'est-à-dire de donner un ESCRS avec des propriétés satisfaisantes de simulation, de préservation de la confluence et de préservation de la normalisation forte (avec quelques restrictions), voir [BR96] pour plus de détails.

I.5 Stratégies

I.5.1 Définitions

La notion de stratégie est intuitive. Par exemple, dans le cadre d'un jeu (au sens usuel, par exemple un jeu de société) à un ou plusieurs joueurs, une stratégie gagnante est une suite de coups, parmi ceux autorisés par les règles du jeu, qui amène inéluctablement à un état particulier, dit gagnant, du jeu.

Il y a plusieurs façons de formaliser cette notion, pas forcément toutes équivalentes. Par exemple, le choix d'un coup doit-il être déterministe ? calculable ? Peut-il dépendre d'un coup joué longtemps avant ? La stratégie est-elle abstraite ou est-elle décrite dans un langage particulier ? Ces discussions n'entrent pas dans le cadre de cette thèse. Nous adoptons ici les définitions de [Ter03, Chapitre 9].

Définition I.5.1 (*Stratégie*)

> Une *stratégie* pour un système de récriture abstrait $(\mathcal{A}, \to)$ est un sous-système de récriture abstrait $(\mathcal{A}, \rightsquigarrow)$ de $(\mathcal{A}, \to)$ (c'est-à-dire tel que $\rightsquigarrow \subseteq \to$) ayant les mêmes objets et formes normales.

Cette définition est celle que nous adopterons. Elle n'est cependant pas parfaite. Tout d'abord elle est parfois trop restrictive : nous verrons des cas où nous nous restreindrons à un ensemble d'objets plus petit et où la stratégie aura davantage de formes normales que le système complet (réduction en forme normale de tête faible, par exemple). Nous pourrons alors nous placer dans le cadre plus général des sous-systèmes de récriture. Elle est aussi trop permissive, et doit être restreinte aux stratégies déterministes (chaque objet a au plus un réduit) et fonctionnelles (le réduit ne dépend que de la source). Enfin, elle ne fournit pas de façon pratique de définir une stratégie. Pour cela, nous adopterons plutôt les formalismes issus de la sémantique opérationnelle, comme illustré à la section suivante.

I.5.2 Stratégies usuelles du λ-calcul

Le λ-calcul, en donnant un statut explicite à la notion de fonction, oppose deux visions du monde. La première, héritée des mathématiques, voit les fonctions comme des objets abstraits à l'intérieur desquels il n'est pas raisonnable de calculer. Cette vision a aussi un sens informatique, dans le cadre de la compilation : une fonction compilée est un morceau de programme qui attend des arguments pour s'exécuter, et qu'on ne sait pas réduire sans ces arguments. L'autre vision considère les fonctions comme des objets syntaxiques ordinaires sur lesquels les calculs sont autorisés. Cette opposition se cristallise dans la règle suivante :

$$\frac{t \to v}{\lambda x.t \to \lambda x.v}\,(\xi)$$

Le λ-calcul vérifie cette règle, qui est par exemple indispensable à la confluence. Cependant une stratégie du λ-calcul est libre de la réaliser ou non : dans le premier cas, on parle alors de stratégie *forte*, dans le second, de stratégie *faible*. En général, une stratégie faible ne peut pas réduire au-delà de la forme normale de tête faible.

Définition I.5.2 (*Appel par nom*)

Les règles d'évaluation du λ-calcul en *appel par nom* (pour les termes clos) sont les suivantes :

$$\frac{}{\lambda x.t \Downarrow \lambda x.t} \qquad \frac{t \Downarrow \lambda x.t' \quad t'\{x := u\} \Downarrow v}{t\ u \Downarrow v}$$

où $t \Downarrow v$ signifie qu'un terme clos t s'évalue en la valeur v.

Théorème I.5.3

Si t a une forme normale de tête faible, alors il existe v tel que $t \Downarrow v$ avec la stratégie d'appel par nom.

Définition I.5.4 (*Appel par valeur*)

Les règles d'évaluation du λ-calcul en *appel par valeur* (pour les termes clos) sont les suivantes :

$$\frac{}{\lambda x.t \Downarrow \lambda x.t} \qquad \frac{t \Downarrow \lambda x.t' \quad u \Downarrow v' \quad t'\{x := v'\} \Downarrow v}{t\ u \Downarrow v}$$

L'appel par nom et l'appel par valeur sont des stratégies très simples, qui sont en général faciles à implanter. Cependant, l'appel par valeur a un défaut : si l'argument d'une fonction n'est pas utilisé, il aura quand même été évalué avant d'effectuer la substitution. L'appel par nom a le défaut dual : si la valeur de l'argument d'une fonction est demandée plusieurs fois, alors il sera effectivement évalué plusieurs fois.

C'est pourquoi on peut préférer une stratégie qui remédie à ces deux défauts : il s'agit de l'*appel par nécessité*, aussi appelé *stratégie paresseuse*. Cette stratégie n'évalue un argument que s'il est nécessaire, et dans ce cas, elle ne l'évalue qu'une seule fois. La façon la plus naturelle de la définir est d'enrichir les termes avec une notion de mémoire ou d'environnement. Nous entrerons davantage dans les détails dans le Chapitre V. Nous nous intéresserons alors également à la stratégie *pleinement paresseuse*, qui est une variante de l'appel par nécessité qui permet un partage encore plus fin.

I.5.3 Optimalité dans le λ-calcul

La notion d'optimalité dans le λ-calcul a été définie par Lévy [Lév80] en 1980. Il y a trois façons à peu près équivalentes de formuler la notion d'optimalité. Une suite de réductions est optimale si :

- elle réalise le nombre minimum d'étapes de β-réduction (parmi toutes les réductions de mêmes termes initial et final) ;
- seuls les rédex nécessaires sont réduits et aucun rédex ni *rédex potentiel* n'est dupliqué ;
- chaque famille de rédex n'est réduite qu'une seule fois au cours de la réduction (la notion de famille est obtenue par clôture symétrique et transitive de celle de résidu).

La question de la réalisation d'une stratégie optimale est difficile. En effet, elle demande une structure de données permettant un partage très fin, car des rédex peuvent être dupliqués

avant même d'être sous la forme de rédex (il s'agit de *rédex potentiels*), comme le montre l'exemple suivant.

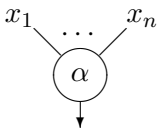
Exemple I.5.5

Dans le terme $(\lambda g.g\,(g\,(\lambda x.x)))\,(\lambda h.((\lambda f.f\,(f\,(\lambda z.z))\,(\lambda w.h\,(w\,(\lambda y.y))))))$, toute réduction duplique des rédex.

Il est donc nécessaire de mettre en œuvre un partage très fin des termes pour espérer définir une telle stratégie. Field [Fie90] a démontré que partager des environnements, des clôtures ou des termes ne suffit pas pour définir une stratégie optimale. Indépendamment, Lamping [Lam90] et Kathail [Kat90] ont proposé une solution au problème de l'optimalité, en définissant des algorithmes dans lesquels le partage ne se limite pas à des sous-termes, et en mettant en œuvre des structures de données complexes dans un système de récriture de graphe particulier, les réseaux d'interaction (voir la section suivante). Ces travaux ont ensuite été améliorés entre autres par Gonthier, Abadi, Lévy [GAL92] et Asperti *et al.* [AGN96, AG98].

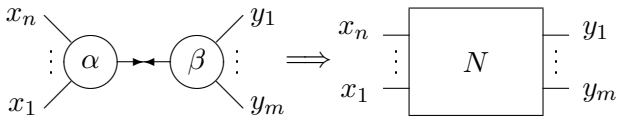
I.6 Réseaux d'interaction

Un système de *réseaux d'interaction* [Laf90] est spécifié par un ensemble de symboles Σ, et un ensemble de *règles d'interaction* $\mathcal{R}$. Chaque symbole $\alpha \in \Sigma$ a une *arité* (fixe) associée. Une occurrence d'un symbole $\alpha \in \Sigma$ est appelée un *agent*. Si l'arité du symbole α est n, alors l'agent a $n+1$ *ports* : un port distingué appelé *port principal*, représenté par une flèche, et n *ports auxiliaires* étiquetés $x_1, \ldots, x_n$ correspondant à l'arité du symbole. Un tel agent est représenté de la façon suivante :



Intuitivement, un réseau N construit sur Σ est un graphe (pas nécessairement connecté) avec des agents aux sommets. Les arêtes du graphe relient les agents aux ports de sorte qu'il y ait seulement une arête à chaque port. Les ports d'un agent qui ne sont pas reliés à un autre agent sont dits *libres*. Il y a deux cas particuliers de réseaux : un câblage (aucun agent) et le réseau vide ; les extrémités des câblages sont également dites libres.

Une règle d'interaction $((\alpha, \beta) \Longrightarrow N) \in \mathcal{R}$ remplace une paire d'agents $(\alpha, \beta) \in \Sigma \times \Sigma$ reliés ensemble par leurs ports principaux (on appelle ceci une *paire active* ou *rédex*, et on la note $\alpha \bowtie \beta$) par un réseau N. Les règles doivent satisfaire deux conditions : tous les ports libres sont préservés pendant la réduction (la réduction est locale, autrement dit, seule la partie du réseau impliquée dans la récriture est modifiée), et il y a au plus une règle pour chaque paire d'agents. En raison de cette dernière restriction, une règle est entièrement définie par son membre gauche ; une telle règle est ainsi parfois notée $\alpha \bowtie \beta$. Le diagramme suivant montre le format des règles d'interaction (N peut être n'importe quel réseau construit sur Σ).



Nous notons $\Longrightarrow$ pour la relation de réduction en une étape, ou $\underset{\alpha \bowtie \beta}{\Longrightarrow}$ si nous voulons être explicites quant à la règle utilisée, et $\Longrightarrow^*$ pour sa clôture réflexive transitive. Si un réseau ne contient aucune paire active, il est en forme normale. La propriété principale des réseaux d'interaction, outre la localité de la réduction, est que la réduction est fortement confluente. En effet, toutes les suites de réduction sont équivalentes par permutation. En particulier, les normalisations faible et forte coïncident (si une suite de réductions termine, alors toutes les suites de réductions terminent).

Si nous relâchons un peu les contraintes des réseaux d'interaction en permettant aux agents d'avoir un nombre quelconque de ports principaux, au lieu d'un seul, nous obtenons ce que nous appelons simplement des *réseaux*. Ce formalisme a été introduit par Alexiev [Ale99] sous le nom de *réseaux d'interaction avec ports principaux multiples* et a également été utilisé sous le nom de *multiports interaction nets* [Maz05], mais il remonte au moins à Bawden [Baw86].

Dans ce formalisme, chaque agent a ainsi un nombre fixe de ports principaux, tous représentés par une flèche ; les autres ports sont toujours qualifiés d'auxiliaires. Une paire active se compose toujours de deux agents reliés par des ports principaux des deux côtés. La réduction est encore locale, mais, en général, on ne peut espérer aucune propriété de confluence.

Si tous les agents dans un système de récriture de réseaux ont exactement un port principal et qu'au plus une règle peut être appliquée à n'importe quelle paire active, alors c'est un système de réseaux d'interaction [Laf90]. Dans ce cas, la réduction est fortement confluente.

Chapitre II

Réduction close

Partant du λ-calcul nommé, nous développons une famille de calculs avec substitutions explicites qui surmontent les problèmes syntaxiques habituels de la substitution. L'idée principale est que seules les substitutions closes peuvent traverser certaines constructions. Ceci donne une forme faible de réduction, appelée *réduction close*, qui est suffisamment riche pour capturer les stratégies d'évaluation d'appel par valeur et d'appel par nom du λ-calcul. De plus, puisque certaines substitutions peuvent traverser les abstractions et que certaines réductions sont autorisées sous les abstractions (sous certaines conditions), la réduction close fournit naturellement une notion de réduction efficace avec un degré de partage élevé et un faible coût d'implantation. Nous présentons une famille de machines abstraites pour la réduction close. Nos résultats expérimentaux montrent que la réduction close se comporte mieux que toutes les stratégies faibles standard, et que son faible coût d'implantation la rend plus efficace que la réduction optimale en pratique.

II.1 Introduction

Le λ-calcul est le modèle fondamental des langages de programmation fonctionnels. La règle de β-réduction décrit l'application d'une fonction (abstraction) à son argument, substituant le paramètre formel de la fonction par l'argument réel : $(\lambda x.t)\, u \to_\beta t\{x := u\}$, où la notation $t\{x := u\}$ représente la substitution sans capture de la Définition I.3.9. Un des problèmes principaux pour étudier la β-réduction est que cette opération de substitution est définie en dehors du système de récriture. Nous identifions deux points importants :

- la substitution doit préserver la signification du terme, ce qui implique par conséquent que des renommages de variables (α-conversions) sont nécessaires pour éviter la capture de variable, introduisant des substitutions *additionnelles* ;
- le travail calculatoire réel d'une étape de β-réduction est dans le processus de substitution : des termes peuvent être copiés ou effacés pendant la propagation de la substitution, d'ailleurs différentes stratégies sont possibles pour cette propagation.

Pour essayer de mieux comprendre ces problèmes cachés, de nombreux λ-calculs avec substitutions explicites ont été proposés (voir la Section I.3.4). Ceux-ci placent la méta-opération de substitution au même niveau que la β-réduction. Dans la plupart des travaux sur les substitutions explicites, les propriétés suivantes sont considérées comme cruciales :

- la simulation de la β-réduction dans sa forme forte (c'est-à-dire être capable de simuler la contraction de n'importe quel rédex) ;
- la préservation de la normalisation forte ;
- la confluence sur les termes avec ou sans métavariables.

De telles propriétés sont clairement souhaitables pour une étude complète de la β-réduction, et sont également essentielles pour certaines applications, voir par exemple [DHK01]. Cependant, si nous nous intéressons seulement au λ-calcul pour les *implantations* de langages fonctionnels, la β-réduction n'a pas besoin d'être simulée en toute généralité, et en particulier certaines stratégies inefficaces peuvent être éliminées. De plus, la confluence ouverte (avec métavariables) n'est plus une propriété essentielle. Cependant, la préservation de la normalisation forte demeure évidemment une propriété utile. Dans ce chapitre, nous étudions des λ-calculs avec substitutions explicites spécifiquement pour implanter le λ-calcul d'une manière efficace : les substitutions sont nécessaires pour exprimer les aspects calculatoires importants de la β-réduction tels que la copie ou l'effacement de termes.

Le λ-calcul *nommé* est souvent considéré comme un point de départ peu commode pour l'étude des substitutions explicites parce que certaines variables ont besoin d'être renommées pendant la réduction, et par conséquent de nouveaux noms doivent être générés dynamiquement (l'α-conversion est coûteuse). C'est en raison de ce défaut que la plupart des calculs avec substitutions explicites utilisent la notation de de Bruijn (voir la Section I.3.2). Ainsi, les calculs avec substitutions explicites sont devenus synonymes de notation de de Bruijn (bien qu'il y ait quelques exceptions, voir par exemple [Ros96]). L'avantage principal de cette notation est qu'elle évite de produire de nouveaux noms de variables, et elle est également considérée comme plus proche de l'implantation. Cependant, de nouveaux problèmes apparaissent :

- Des opérations de décalage d'indice (par exemple les opérations *shift* et *lift* du $\lambda\sigma$-calcul, correspondant essentiellement à la fonction $\mathcal{U}_i^n(t)$ de la Section I.3.2) sont nécessaires. Cela revient essentiellement à coder la génération de nouveaux noms.

- Les substitutions sont souvent représentées comme des *listes*, ce qui implique que de nouvelles opérations sont nécessaires pour manipuler cette structure.
- Certaines propriétés telles que la confluence et la préservation de la normalisation forte ne peuvent pas être facilement obtenues (voir par exemple [DG01]).

Il est bien connu que les problèmes ci-dessus peuvent être évités simplement en ne permettant pas la substitution à travers une abstraction. Presque tous les évaluateurs du λ-calcul sont basés sur la réduction faible qui est caractérisée précisément par l'interdiction de la réduction sous les abstractions. Par conséquent, la partie la plus coûteuse et la plus délicate du processus de substitution est enlevée du système. Cependant, ceci est réalisé à un prix parce que des termes avec des substitutions bloquées (des clôtures) peuvent être copiés, ce qui peut entraîner la duplication de rédex (et de rédex potentiels). Comme Çağman et Hindley [ÇH98] le font remarquer, l'α-conversion peut également être évitée si les β-rédex sont fermés (c'est-à-dire si $(\lambda x.t)\,u$ ne contient pas de variables libres). Cependant, cette restriction est également très forte, et elle a des inconvénients comparables.

Partant de ces remarques, nous proposons dans ce chapitre une famille de calculs *nommés* qui fournissent une base pour l'implantation des langages de programmation fonctionnels et surmontent les problèmes mentionnés ci-dessus. Un avantage majeur d'employer des noms est que nos calculs sont plus proches de l'intuition, et nous n'introduisons aucun des problèmes qui sont généralement associés aux notations alternatives (par exemple des manipulations d'indices). De plus, un calcul nommé mais sans α-conversion n'est pas moins proche de l'implantation qu'un calcul avec indices : les noms peuvent simplement correspondre à des pointeurs ou à des indices dans un tableau, alors que les indices de de Bruijn sont plutôt implantés par des listes ou des environnements. Les caractéristiques principales des calculs que nous définirons sont :

- l'absence d'α-conversion ;
- un contrôle fin du processus de substitution pour éviter des calculs inutiles ;
- une stratégie de réduction avec un bon degré de partage des calculs.

Notre point de départ est un système de récriture qui ajoute simplement à la règle β la définition de la substitution (habituellement au niveau méta) comme ensemble de règles de récriture conditionnelles. La définition habituelle de la substitution est cependant très naïve, au moins d'un point de vue calculatoire : des substitutions sont poussées de la racine du terme vers *toutes* les feuilles. Mais pourquoi devrait-on pousser des substitutions dans des branches du terme où il n'y a aucune occurrence de la variable libre à remplacer ? En effet, c'est l'essence-même du contre-exemple bien connu à la préservation de la normalisation forte dans $\lambda\sigma$ [Mel95]. Nous éviterons ce problème en explicitant également les phases de copie et d'effacement de la substitution, nous inspirant de divers calculs pour la logique linéaire (voir par exemple [Abr93]). Ceci nous permettra également de contrôler (et d'éviter) les copies et effacements de variables libres pendant le processus de substitution. Nous éliminerons également les problèmes de capture de variables en autorisant certaines réductions seulement quand un sous-terme est fermé. Ceci enlève la nécessité de générer des noms de variable frais pendant la réduction, ce qui répond aux objections principales contre les calculs avec substitutions explicites nommés.

Pour récapituler, l'aspect principal de la stratégie de réduction utilisée dans ce chapitre, que nous appelons *réduction close*, est qu'elle autorise facilement :

– la réduction sous, ainsi que la substitution à travers, les abstractions ;
– l'absence de copie de variables libres ;
– un glanage de cellules (*garbage collection*) propre.

Il y a plusieurs manières d'obtenir une stratégie de réduction close, qui mènent à différents calculs avec substitutions et gestion de ressources explicites. Le premier, que nous appelons λ_{ca} , est caractérisé par le fait que, dans la règle β, l'argument doit être fermé. Puisque c'est la seule règle qui crée une substitution, ceci implique que toutes les substitutions sont fermées. Bien que restrictif, ce calcul est suffisant pour l'exécution des programmes fonctionnels, et bénéficie de propriétés telles que la confluence, la terminaison des règles de substitution, et la préservation de la normalisation forte.

Du point de vue du partage des calculs, λ_{ca} est meilleur que les stratégies faibles standard mais n'est pas encore complètement satisfaisant. Nous définirons un deuxième calcul, appelé λ_{cf} , où l'idée principale est que nous pouvons créer des substitutions ouvertes mais qu'elles doivent être fermées quand elles traversent une abstraction. Comme λ_{ca}, ce calcul est présenté sous la forme d'un système de récriture travaillant sur des termes de base (nous ne récrivons pas de termes avec métavariables). Néanmoins, ce calcul répond en partie aux problèmes soulevés par [Muñ96], et est suffisant pour l'évaluation des programmes fonctionnels. Nous prouvons que λ_{cf} est confluent, préserve la normalisation forte, termine sur les termes typés, et peut simuler les stratégies d'évaluation habituelles, l'appel par nom et l'appel par valeur. Nous comparons également λ_{ca}, λ_{cf}, et des variantes telles que λ_{cl} , qui exige que les rédex soient fermés dans la règle β comme dans [ÇH98], et λ_{caf} qui généralise λ_{ca} et λ_{cf}.

Nous utilisons cette syntaxe avec substitutions explicites pour définir une famille de machines abstraites pour la réduction close en utilisant une sémantique opérationnelle structurée pour ce calcul. Ces machines ont été implantées, et les expérimentations indiquent que le niveau de partage obtenu est proche de celui de la réduction optimale, alors que la technologie que nous utilisons est nettement plus simple.

État de l'art. Ces travaux s'inscrivent dans le cadre de l'utilisation des substitutions explicites pour contrôler le processus de substitution dans le λ-calcul en insistant sur l'implantation, par exemple le λ-calcul en appel par nécessité de [AFM$^+$95] et les calculs avec environnements partagés [Yos94]. D'autres travaux orienté vers l'implantation sont [HMP98, Ros96, Nad99]. Dans ce dernier, des informations disant si un terme est clos ou non sont employées pour définir des réductions conditionnelles, bien qu'avec différentes motivations et dans un cadre tout à fait différent du nôtre (avec une notation de *suspension* qui emploie des indices de de Bruijn). Notre notion de réduction close est inspirée par une stratégie d'élimination des coupures dans la logique linéaire, utilisée dans une preuve de correction de la géométrie de l'interaction par Girard [Gir89].

Aperçu. Dans la section suivante nous motivons notre travail par l'étude des aspects problématiques des calculs avec substitutions explicites. Dans la Section II.3.1, nous introduisons les λ_{c}-termes, et dans les Sections II.3.2 et II.3.3, nous présentons deux calculs correspondant à deux stratégies différentes de réduction close : λ_{ca} (pour *argument fermé*, *closed argument* en anglais) et λ_{cf} (pour *fonction fermée* ou *closed function*). La relation avec la β-réduction et

les stratégies habituelles du λ-calcul est donnée dans la Section II.3.4. Dans la Section II.3.5, nous donnons deux autres variantes de réduction fermée, et discutons nos choix dans la Section II.3.6. Dans la Section II.4, nous donnons un système de type et montrons la réduction du sujet et la terminaison. Nous discuterons des implantations dans la Section II.5, où nous définissons et comparons des machines abstraites. Nous concluons le chapitre en Section II.6.

II.2 Travaux antérieurs et motivations

Notre point de départ est le λ-calcul avec β-réduction, où la méta-opération habituelle de la substitution est remplacée par un ensemble de règles de récriture conditionnelles inspirées par la définition originale de la substitution donnée par Church. Nous utilisons des calculs avec substitutions explicites nommés, mais la majeure partie de ce que nous disons peut être formulée en notation de de Bruijn. Comme dans la Section I.3, nous dénotons par $t\{x := u\}$ la substitution implicite habituelle et par Λ l'ensemble des λ-termes. Il est à noter que nous ne considérons pas ici les λ-termes modulo α-conversion, nous traiterons cet aspect explicitement.

Définition II.2.1 (*λ-calcul avec substitutions explicites*)

Soient x, y des variables, $\star$ une constante arbitraire (représentant par exemple les entiers ou les booléens ; cela sera utile dans la Section II.4), et t, u des termes définis par :

$$t, u ::= x \mid \star \mid \lambda x.t \mid t\ u \mid t[u/x].$$

Nous considérons que l'application associe à gauche : $t\,u\,v = (t\,u)\,v$, et adoptons une convention similaire pour les substitutions : $t[u/x][v/y] = (t[u/x])[v/y]$. Nous abrégeons $\lambda x.\lambda y.t$ en $\lambda xy.t$. La notion de variables libres d'un terme t (notation $\mathsf{fv}(t)$) est la même que d'habitude, avec $\mathsf{fv}(t[u/x]) = \mathsf{fv}((\lambda x.t)\,u)$. La Table II.1 donne les règles de récriture conditionnelles de ce calcul. La condition "z frais" signifie que z est une variable qui n'apparaît pas dans le membre gauche.

Tab. II.1 – λ-calcul avec substitutions explicites

Nom		Réduction		Condition
Beta	$(\lambda x.t)\,v$	$\to$	$t[v/x]$	
Cons	$\star[v/x]$	$\to$	$\star$	
Var$_1$	$x[v/x]$	$\to$	v	
Var$_2$	$y[v/x]$	$\to$	y	$x \neq y$
App	$(t\,u)[v/x]$	$\to$	$(t[v/x])\,(u[v/x])$	
Lam$_1$	$(\lambda x.t)[v/x]$	$\to$	$\lambda x.t$	
Lam$_2$	$(\lambda y.t)[v/x]$	$\to$	$\lambda y.t[v/x]$	$x \notin \mathsf{fv}(t) \vee y \notin \mathsf{fv}(v), x \neq y$
Lam$_3$	$(\lambda y.t)[v/x]$	$\to$	$\lambda z.t[z/y][v/x]$	$x \in \mathsf{fv}(t), y \in \mathsf{fv}(v), x \neq y, z$ frais
Comp	$t[u/x][v/y]$	$\to$	$t[v/y][u[v/y]/x]$	$x \notin \mathsf{fv}(v)$

La Table II.1 définit en fait un ensemble de schémas de réductions impliquant des métavariables de termes t, u, v. x, y, z peuvent être vus aussi bien comme des métavariables de variables du λ-calcul, ou comme des constantes dans un système de récriture avec un nombre infini de règles (par exemple, la règle Var_1 s'applique à n'importe quelle variable du λ-calcul avec une substitution). Par instantiation, ces schémas de règles définissent une relation de récriture sur les λ-termes, close par contexte. La réduction peut survenir dans *n'importe quel* contexte, en particulier sous les abstractions et dans les substitutions :

$$\frac{t \to t'}{t\ u \to t'\ u} \qquad \frac{u \to u'}{t\ u \to t\ u'} \qquad \frac{t \to t'}{\lambda x.t \to \lambda x.t'} \qquad \frac{t \to t'}{t[u/x] \to t'[u/x]} \qquad \frac{u \to u'}{t[u/x] \to t[u'/x]}$$

Il y a des variantes du calcul ci-dessus, par exemple si nous partons d'un terme dont toutes les variables liées sont initialement différentes (convention de Barendregt), alors nous pouvons aussi bien effectuer l'α-conversion dans les règles *App* et *Comp*, quand les substitutions sont copiées.[1] Cependant, nous préférons prendre la définition précédente, plus familière, comme point de départ.

En analysant ce calcul d'une perspective d'*implantation*, nous identifions certains défauts :

- Les substitutions sont propagées exhaustivement. Dans les règles *App* et *Comp*, la substitution v est copiée, même si la variable x n'apparaît pas libre dans l'un de ses sous-termes.
- Dans le même esprit, Var_2 et Lam_1 effacent le terme v quand il est évident que la substitution n'est pas nécessaire.
- L'α-conversion est nécessaire dans la règle Lam_3, qui, par conséquent, crée la substitution additionnelle $t[z/y]$ qui va aussi être propagée exhaustivement. De plus, une variable fraîche doit être générée par un certain processus externe. En particulier, ce calcul n'a aucun espoir d'être confluent si l'on ne considère pas les termes modulo α-conversion.
- La règle *Comp* peut être appliquée indéfiniment, et il n'y a donc aucun espoir de prouver des résultats de terminaison pour ce calcul en général.

Nous étudions maintenant des moyens de réduire ces surcoûts, d'abord en se concentrant sur l'élimination de l'α-conversion, puis sur la propagation de la substitution.

II.2.1 α-CONVERSION

Il y a plusieurs remèdes maladroits qui peuvent être prescrits pour éviter l'α-conversion. Pour des termes clos, il est bien connu que si nous interdisons la réduction sous les abstraction (ce qui reste suffisant pour réduire en forme normale de tête faible), alors l'α-conversion n'est pas nécessaire. Dans les calculs avec substitutions explicites, cette forme de *réduction faible* implique également d'interdire la propagation des substitutions à travers les abstractions (ainsi la règle *Lam* ci-dessus est enlevée du système). Cependant, nous observons que cette restriction est trop forte. Par exemple, il n'y aurait aucun problème à réduire le rédex de tête de $t = \lambda x.(\lambda y.y)\, x$, mais à la place ce rédex sera dupliqué par la réduction faible de l'application suivante : $(\lambda x.x\, x)\, t \to^* (\lambda x.(\lambda y.y)\, x)\, (\lambda x.(\lambda y.y)\, x)$. Pire, non seulement les rédex

[1] Par conséquent, dans le λ-calcul linéaire où l'effacement et surtout la copie ne sont pas permis, il n'y a aucunement besoin d'α-conversion si le terme initial a des noms différents pour toutes ses variables liées.

présents, mais également les rédex potentiels peuvent être copiés : par exemple, dans l'application précédente, prenons $t = (\lambda z.x\,y\,z)[\lambda x.x/x][\lambda y.y/y]$. Ici encore, aucune α-conversion n'est nécessaire.

Ceci motive notre première amélioration. Pour éviter de telles duplications, nous autoriserons la réduction et la propagation des substitutions sous les abstractions, mais seulement quand cela n'entraîne pas d'α-conversion. Il existe un certain nombre de solutions moins radicales que la réduction faible, par exemple :

- Les rédex clos [ÇH98]. Si la règle *Beta* : $(\lambda x.t)\,u \to t[u/x]$ est restreinte au cas où $\mathsf{fv}((\lambda x.t)\,u) = \varnothing$, alors il n'y a jamais besoin d'α-conversion : toutes les substitutions créées par cette règle seront fermées. Ce calcul autorise davantage de réductions que la réduction faible standard sur les termes clos, et a beaucoup de propriétés utiles.
- Éliminer la règle Lam_3. Comme c'est la seule règle qui demande une α-conversion, nous pouvons l'enlever du système. Bien que ce soit le système le moins restrictif, il demande toujours deux tests d'appartenance explicites (dans la règle Lam_2).

La question principale que nous nous posons est la suivante : pouvons-nous faire mieux que les solutions ci-dessus, en trouvant un compromis entre elles ? Les deux calculs que nous allons présenter peuvent être simplement compris comme un compromis qui offre autant de réductions que possible, sans le coût de l'α-conversion.

II.2.2 Propagation des substitutions

Pousser une substitution $[v/x]$ à travers une application copie naïvement le terme v (donc aussi tous les rédex présents dans v). Ceci peut être considéré comme la source principale d'inefficacité au niveau syntaxique. Considérons un terme tel que $(x\,t_1\,t_2\cdots t_n)[v/x]$, où la seule occurrence de x est celle visible. Pour que le processus de substitution s'accomplisse avec les règles données, il faut réaliser $n+1$ copies de la substitution, plus toutes les copies additionnelles dépendant de la structure des termes t_i. Toutes les copies seront finalement jetées, sauf une. Cependant, il y a une seule occurrence de x dans ce terme, ainsi aucune duplication (ni partage) n'est nécessaire. Ceci suggère notre prochaine amélioration du calcul, en remplaçant la règle pour la substitution dans les applications par l'ensemble de règles suivant :

$$\begin{array}{lcll} (t\,u)[v/x] & \to & (t[v/x])\,u & (x \in \mathsf{fv}(t), x \notin \mathsf{fv}(u)) \\ (t\,u)[v/x] & \to & t\,(u[v/x]) & (x \notin \mathsf{fv}(t), x \in \mathsf{fv}(u)). \end{array}$$

Ces règles correspondent aux cas où x est linéaire dans le terme (localement). Nous avons alors besoin de deux autres règles pour capturer les cas cruciaux où x n'apparaît pas du tout, et où il apparaît plusieurs fois[2] :

$$\begin{array}{lcll} (t\,u)[v/x] & \to & t\,u & (x \notin \mathsf{fv}(tu)) \\ (t\,u)[v/x] & \to & (t[v/x])\,(u[v/x]) & (x \in \mathsf{fv}(t), x \in \mathsf{fv}(u)). \end{array}$$

Factoriser les occurrences de variables linéaires et non-linéaires peut être fait au niveau syntaxique, ce qui permet d'éviter certaines conditions sur les règles. C'est ce qu'on appelle les opérateurs explicites ou la gestion de ressource explicite. Cette notion existe au

[2]Ces quatre alternatives pour pousser une substitution dans une application peuvent également être comprises en faisant appel aux combinateurs S, B, C, K de la Section I.3.3.

moins depuis [Abr93], et a été introduite dans la présente formulation par [FM99] (subsumé par [FMS05a]). Elle a également été utilisée par exemple dans [Oos01] et [KL05] avec des motivations différentes. C'est bien sûr une notion très naturelle, issue des réseaux de preuves et de la logique linéaire [Gir87]. Nous étendons donc le calcul avec les constructions et règles de réduction suivantes :

- Effacement : $(\epsilon_x.t)$. Si $x \notin \mathsf{fv}(t)$, alors on peut le faire apparaître explicitement grâce au constructeur d'effacement. La règle de réduction associée préserve cette notation :

  $$(\epsilon_x.t)[v/x] \to \epsilon_{x_1}.\epsilon_{x_2}.\cdots\epsilon_{x_n}.t$$

  où $\{x_1,\ldots,x_n\} = \mathsf{fv}(v)$. Remarquons cependant que si v est clos, alors la règle se simplifie en :

  $$(\epsilon_x.t)[v/x] \to t.$$

  En imposant $\mathsf{fv}(v) = \varnothing$, l'effacement (qui correspond au glanage de cellules) devient une opération simple et atomique.
- Copie : $(\delta_x^{y,z}.u)$. Si x apparaît deux fois dans un terme, nous renommons une occurrence en y et l'autre en z (y et z sont supposés frais) et nous utilisons le constructeur de copie, avec la règle suivante :

  $$(\delta_x^{y,z}.t)[v/x] \to \delta_{\vec{x}}^{\vec{y},\vec{z}}.t[v[\vec{y}/\vec{x}]/y][v[\vec{z}/\vec{x}]/z]$$

  où $\vec{x} = \mathsf{fv}(v)$, et $\vec{y}$ et $\vec{z}$ sont supposés frais Mais nous avons clairement introduit une multitude de nouvelles α-conversions, mais de nouveau, si v est clos, la règle se simplifie en :

  $$(\delta_x^{y,z}.t)[v/x] \to t[v/y][v/z].$$

Non seulement les règles de réduction ci-dessus sont beaucoup plus simples dans le cas où les substitutions sont fermées, mais c'est également une clef de l'efficacité. Si des termes qui contiennent des variables libres sont copiés, alors des rédex potentiels pourraient également être dupliqués. Ainsi notre choix de conception est de copier seulement les termes qui ne contiennent pas de variables libres, et qui n'ont donc besoin d'aucune α-conversion.

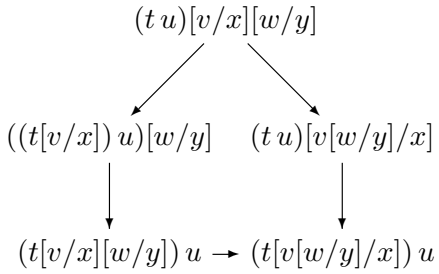
On peut dès maintenant remarquer que nous n'avons plus besoin des règles Var_2 et Lam_1 puisqu'aucune substitution n'atteindra jamais un terme à moins qu'il ne contienne la variable libre en train d'être substituée. Les substitutions sont maintenant guidées à travers le terme par le constructeur de copie et la règle *App* jusqu'à la destination correcte.

Avec l'addition des constructions explicites pour les effacements et copies de substitutions, toutes les variables apparaissent exactement une fois dans une terme. Ainsi la règle pour *Comp* se divise également en deux :

$$\begin{array}{lcll} t[u/x][v/y] & \to & t[v/y][u/x] & (y \in \mathsf{fv}(t)) \\ t[u/x][v/y] & \to & t[u[v/y]/x] & (y \in \mathsf{fv}(u)). \end{array}$$

La première est une cause directe de non-terminaison du système de récriture, et doit donc être exclue. La seconde, en revanche, est utile, car elle permet aux substitutions d'avancer vers leur destination. En fait, elle est plus qu'utile, car son inclusion autorise des chemins de

réduction plus courts. Un terme de la forme $(t\,u)[v/x][w/y]$, où $y \in \mathsf{fv}(v), x \in \mathsf{fv}(t)$ peut subir les réductions suivantes :

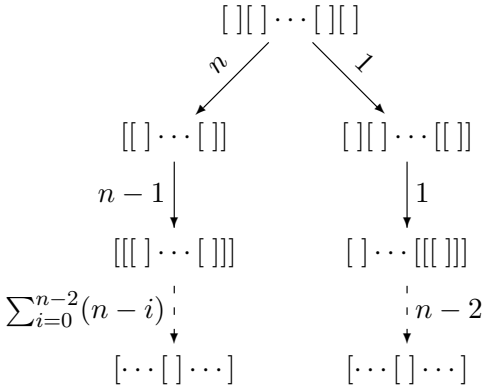


Clairement, autoriser la composition des substitutions avant de pousser les substitutions donne un chemin de réduction plus court. Le résultat suivant clarifie à quel point cette observation est importante.

PROPOSITION II.2.2 (*Ordre des substitutions*)

> Soit $t = t_0[t_1/x_0][t_2/x_1]\cdots[t_n/x_{n-1}]$ où $\mathsf{fv}(t_i) = \{x_i\}, i < n$, et $\mathsf{fv}(t_n) = \varnothing$. Une suite de réductions $t \to^* u$ pour propager les substitutions sera en $O(n)$ dans le meilleur cas et en $O(n^2)$ au pire.

Preuve. Il y a un choix entre composer les substitutions les plus internes ou les plus externes. Le diagramme suivant en illustre les conséquences. Pour simplifier le diagramme, nous n'écrivons pas les termes, mais seulement les crochets. Ainsi le terme de départ est $[\,][\,]\cdots[\,]$, et la règle de composition des substitutions s'écrit $[\,][\,] \to [[\,]]$. Nous indiquons sur les flèches le nombre de fois que la règle est appliquée :



Ainsi la séquence la plus interne requiert $\sum_{i=0}^{n}(n-i) = \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$, ce qui est en $O(n^2)$, alors que que la plus externe est en $O(n)$. □

L'ordre dans lequel nous effectuons la composition des substitutions a un effet spectaculaire sur l'efficacité du système de récriture. Une façon naturelle d'obtenir une stratégie externe efficace (*outermost*, le rédex le plus externe est réduit d'abord) est d'imposer que la substitution externe soit close.

Ceci conclut nos premières observations sur l'α-conversion et la propagation des substitutions. Partant de ce ces remarques, nous proposons dans les sections suivantes une famille de calculs, pour lesquels nous définissons d'abord une syntaxe commune.

II.3 Calculs nommés

II.3.1 Syntaxe : λ_{c}

Nous commençons par définir les λ_{c}-termes, qui sont des λ-termes avec des constructions explicites pour les substitutions, les copies et les effacements. Nous compilerons les λ-termes en λ_{c}-termes.

Définition II.3.1 (λ_{c}-*termes*)

Nous utilisons x, y, z pour dénoter des variables, t, u, v pour dénoter des termes, et $\star$ pour une constante arbitraire. La Table II.2 montre les constructions de termes, les contraintes sur les variables qui doivent être satisfaites pour chaque construction, et les variables libres associées. Λ_{c} dénote l'ensemble des λ_{c}-termes.

Tab. II.2 – λ_{c}-termes

Nom	Terme	Contrainte	Variables libres
Constante	$\star$	–	$\varnothing$
Variable	x	–	$\{x\}$
Abstraction	$\lambda x.t$	$x \in \mathsf{fv}(t)$	$\mathsf{fv}(t) \setminus \{x\}$
Application	$t\,u$	$\mathsf{fv}(t) \cap \mathsf{fv}(u) = \varnothing$	$\mathsf{fv}(t) \cup \mathsf{fv}(u)$
Effacement	$\epsilon_x.t$	$x \notin \mathsf{fv}(t)$	$\mathsf{fv}(t) \cup \{x\}$
Copie	$\delta_x^{y,z}.t$	$x \notin \mathsf{fv}(t), y \neq z, \{y, z\} \subseteq \mathsf{fv}(t)$	$(\mathsf{fv}(t) \setminus \{y, z\}) \cup \{x\}$
Substitution	$t[u/x]$	$x \in \mathsf{fv}(t), (\mathsf{fv}(t) \setminus \{x\}) \cap \mathsf{fv}(u) = \varnothing$	$(\mathsf{fv}(t) \setminus \{x\}) \cup \mathsf{fv}(u)$

Les contraintes sur les variables impliquent que chaque variable apparaît libre dans un terme exactement une fois (dans ce calcul nous pourrions avoir la règle $\eta : (\lambda x.t\,x) \to t$ sans la condition de garde $x \notin \mathsf{fv}(t)$).

Ces termes doivent être compris comme un langage intermédiaire vers lequel sont compilés les λ-termes. Dans le reste de ce chapitre, on supposera que tous les λ_{c}-termes sont produits par une fonction de compilation $[\![\cdot]\!] : \Lambda \to \Lambda_{\mathsf{c}}$, ou sont des réduits de tels termes.

DÉFINITION II.3.2 (*Compilation*)

Soit t un λ-terme. Sa compilation $[\![t]\!]$ dans λ_{c} est définie par : $[x_1]\dots[x_n]\langle t\rangle$ où $\mathsf{fv}(t) = \{x_1,\dots,x_n\}$, $n \geq 0$, nous supposons en toute généralité que les variables sont traitées dans l'ordre lexicographique, et $\langle\cdot\rangle$ est défini par : $\langle\star\rangle = \star$, $\langle x\rangle = x$, $\langle t\,u\rangle = \langle t\rangle\,\langle u\rangle$ et $\langle\lambda x.t\rangle = \lambda x.[x]\langle t\rangle$ si $x \in \mathsf{fv}(t)$, sinon $\langle\lambda x.t\rangle = \lambda x.\epsilon_x.\langle t\rangle$. Nous définissons $[\cdot]\cdot$ comme suit :

$$\begin{array}{lcll}
[x]x & = & x & \\
[x](\lambda y.t) & = & \lambda y.[x]t & \\
[x](t\,u) & = & \delta_x^{x',x''}.[x'](t\{x := x'\})\,[x''](u\{x := x''\}) & x \in \mathsf{fv}(t), x \in \mathsf{fv}(u) \\
 & = & ([x]t)\,u & x \in \mathsf{fv}(t), x \notin \mathsf{fv}(u) \\
 & = & t\,([x]u) & x \notin \mathsf{fv}(t), x \in \mathsf{fv}(u) \\
[x](\epsilon_y.t) & = & \epsilon_y.[x]t & \\
[x](\delta_y^{y',y''}.t) & = & \delta_y^{y',y''}.[x]t &
\end{array}$$

où la substitution $t\{x := u\}$ est la notion (implicite) habituelle, et les variables x' et x'' ci-dessus sont fraîches.

Il est à noter que les variables libres sont traitées dans l'ordre lexicographique. Par exemple :

$$[\![(x\,y)\,(x\,y)]\!] = [x][y]\langle(x\,y)\,(x\,y)\rangle = \delta_y^{y',y''}.\delta_x^{x',x''}.(x'\,y')\,(x''\,y'') \neq [y][x]\langle(x\,y)\,(x\,y)\rangle.$$

Nous aurions naturellement pu choisir n'importe quel ordre dans la définition.

Les termes produits par la compilation sont des termes *purs* : ils ne contiennent pas de substitution. L'essence de la compilation est de compter les variables, et de placer les constructeurs d'effacement et de copie là où ils sont nécessaires. La compilation particulière que nous avons choisie a la propriété que l'effacement est placé à la position la plus externe (*outermost*), et la copie à la plus interne (*innermost*), ce qui est intuitivement le plus efficace (effacer aussi rapidement que possible, mais attendre pour la duplication jusqu'à ce que ce soit nécessaire pendant la propagation de la substitution). En conséquence, les effacements n'apparaissent qu'immédiatement sous les abstractions, par exemple : $\lambda x.\epsilon_x.t$. Nous pourrions donc aussi bien introduire une nouvelle abstraction $\lambda_.t$ qui combine ces derniers. Cependant, nous préférons les maintenir séparés dans la majeure partie de ce chapitre. Nous reviendrons sur ce choix dans le Chapitre IV.

Nous étudions maintenant quelques propriétés générales de λ_{c} et de la fonction de compilation.

PROPOSITION II.3.3

Si t est un λ-terme alors :

1. $\mathsf{fv}([\![t]\!]) = \mathsf{fv}(t)$.
2. $[\![t]\!]$ est un λ_{c}-terme valide (satisfaisant les contraintes de la Table II.2).

Preuve.

1. La fonction de traduction n'efface pas de variable, et n'introduit que des variables liées.

2. Nous procédons par induction sur la structure de t. Si $t = \star$ ou $t = x$, alors le résultat est trivial (les deux se traduisent en termes sans contraintes). Supposons $\mathsf{fv}(t) = \{x_1, \ldots, x_n\}$. Si $t = \lambda x.u$, il y a alors deux cas à considérer :
   (a) Si $x \in \mathsf{fv}(u)$, alors :

   $$[x_1] \cdots [x_n]\langle \lambda x.u \rangle = [x_1] \cdots [x_n] \lambda x.[x]\langle u \rangle = \lambda x.[x_1] \cdots [x_n][x]\langle u \rangle.$$

   Par hypothèse $[x_1] \cdots [x_n][x]\langle u \rangle$ est un λ_{c}-terme valide, et par le point 1, nous savons que $x \in \mathsf{fv}([x_1] \cdots [x_n][x]\langle u \rangle)$, ainsi $\lambda x.[x_1] \cdots [x_n][x]\langle u \rangle$ est également valide.
   (b) Si $x \notin \mathsf{fv}(u)$, alors :

   $$[x_1] \cdots [x_n]\langle \lambda x.u \rangle = [x_1] \cdots [x_n] \lambda x.\epsilon_x.\langle u \rangle = \lambda x.\epsilon_x.[x_1] \cdots [x_n]\langle u \rangle.$$

   Par hypothèse $[x_1] \cdots [x_n]\langle u \rangle$ est un λ_{c}-terme valide, et $x \notin \mathsf{fv}([x_1] \cdots [x_n]\langle u \rangle)$ par le point 1, ainsi $\epsilon_x.[x_1] \cdots [x_n]\langle u \rangle$ est valide, et $\lambda x.\epsilon_x.[x_1] \cdots [x_n]\langle u \rangle$ est également valide.

   Enfin, si $t = u\, v$, alors $[x_1] \cdots [x_n]\langle u\, v \rangle = [x_1] \cdots [x_n](\langle u \rangle\langle v \rangle)$. Supposons également que $\mathsf{fv}(u) \cap \mathsf{fv}(v) = \{x_{i_1}, \ldots, x_{i_p}\}$ pour un certain $p \geq 0$. Maintenant, soient u' le terme $u\{x_{i_1} := x'_{i_1}, \ldots, x_{i_p} := x'_{i_p}\}$ et $v' = v\{x_{i_1} := x''_{i_1}, \ldots, x_{i_p} := x''_{i_p}\}$, et soient $y_{j_1}, \ldots, y_{j_q}$ et $z_{k_1}, \ldots, z_{k_r}$ les listes de variables libres de u' et v' respectivement, en ordre lexicographique, alors :

   $$[x_1] \cdots [x_n](\langle u \rangle\langle v \rangle) = \delta_{x_{i_p}}^{x'_{i_p}, x''_{i_p}} \ldots . \delta_{x_{i_1}}^{x'_{i_1}, x''_{i_1}} . [y_{j_1}] \ldots [y_{j_q}]\langle u' \rangle [z_{k_1}] \ldots [z_{k_r}]\langle v' \rangle.$$

   Par le point 1 et l'hypothèse d'induction, $[y_{j_1}] \ldots [y_{j_q}]\langle u' \rangle$ et $[z_{k_1}] \ldots [z_{k_r}]\langle v' \rangle$ sont deux λ_{c}-termes valides. Ils n'ont aucune variable en commun (puisque les variables communes ont été renommées), donc l'application est un λ_{c}-terme valide. Maintenant, $x_{i_1}, \ldots, x_{i_p}$ ne sont pas libres dans l'application, $x'_{i_j} \neq x''_{i_j}$ et x'_{i_j}, x''_{i_j} sont des variables libres de l'application (pour tout $1 \leq j \leq p$), ainsi le terme résultant est valide.

□

Nous pouvons facilement récupérer un λ-terme à partir d'un λ_{c}-terme en effaçant simplement les constructions additionnelles et en propageant les substitutions jusqu'au bout. Pour ceci nous donnons une fonction de décompilation (*readback*) :

DÉFINITION II.3.4 (*Décompilation*)

La fonction $(\cdot)^* : \Lambda_{\mathsf{c}} \to \Lambda$ est définie inductivement comme suit :

$$\begin{array}{lcllcl} \star^* & = & \star & x^* & = & x \\ (t\, u)^* & = & t^*\, u^* & (\lambda x.t)^* & = & \lambda x.t^* \\ (\epsilon_x.t)^* & = & t^* & (\delta_x^{y,z}.t)^* & = & t^*\{y := x\}\{z := x\} \\ (t[u/x])^* & = & t^*\{x := u^*\} & & & \end{array}$$

Une propriété utile de la décompilation est la suivante :

PROPOSITION II.3.5

Si t est un λ-terme, $[\![t]\!]^* = t$.

Preuve. Induction immédiate sur la structure du λ-terme t. □

EXEMPLE II.3.6 (*Termes issus de la compilation*)

Nous donnons quelques exemples de termes dans ce calcul, obtenus en utilisant la fonction de compilation :

$$\begin{array}{lcll} \mathsf{I} & = & [\![\lambda x.x]\!] & = \lambda x.x \\ \mathsf{K} & = & [\![\lambda x.\lambda y.x]\!] & = \lambda x.\lambda y.\epsilon_y.x \\ \mathsf{S} & = & [\![\lambda x.\lambda y.\lambda z.xz(yz)]\!] & = \lambda x.\lambda y.\lambda z.\delta_z^{z',z''}.(xz')(yz'') \\ 2 & = & [\![\lambda f.\lambda x.f(fx)]\!] & = \lambda f.\lambda x.\delta_f^{g,h}.g(hx) \\ \mathsf{Y} & = & [\![\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))]\!] & = \lambda f.\delta_f^{g,h}.(\lambda x.g(\delta_x^{x',x''}.x'x'')) \\ & & & \qquad (\lambda x.h(\delta_x^{x',x''}.x'x'')) \end{array}$$

A titre de contre-exemple, considérons le terme $\lambda x.\lambda y.\epsilon_x.y$. C'est un λ_{c}-terme valide (voir la Table II.2), mais qui ne résulte pas de la fonction de compilation. Notons cependant qu'il y a une manière standard de le transformer en un terme équivalent issu de la compilation : $[\![(\lambda x.\lambda y.\epsilon_x.y)^*]\!] = \lambda x.\epsilon_x.\lambda y.y$.

Ceci achève la définition et les propriétés de base de la syntaxe de notre calcul. Dans les sections suivantes nous définissons plusieurs relations de réduction sur les λ_{c}-termes, avec la propriété principale que l'α-conversion n'est pas nécessaire.

II.3.2 ARGUMENTS FERMÉS : λ_{ca}

A la suite de la Section II.2, nous observons que tous les problèmes de l'α-conversion peuvent être supprimés si toutes les substitutions sont fermées. Puisque la seule règle qui crée une substitution est *Beta* : $(\lambda x.t)\, v \to t[v/x]$, nous commençons par exiger que l'argument v soit fermé dans cette règle. Il y a un certain nombre de conséquences immédiates de cette contrainte, en particulier nous n'avons plus besoin de la règle *Comp*. Dans cette section nous formalisons ce calcul, appelé λ_{ca} et étudions ses propriétés. Cette stratégie d'implantation du λ-calcul est clairement faible, mais nous prouverons qu'elle est suffisante pour l'évaluation des programmes.

DÉFINITION II.3.7 (λ_{ca}-*calcul*)

L'ensemble Λ_{ca} des λ_{ca}-*termes valides* contient tous les λ_{c}-termes qui sont l'image d'un λ-terme par la fonction de compilation et leurs réduits par la relation de réduction $\to_{\mathsf{ca}}$ définie par le système de récriture conditionnel de la Table II.3.

Comme d'habitude, nous écrivons $\to_{\mathsf{ca}}^*$ pour la clôture réflexive et transitive de $\to_{\mathsf{ca}}$. La réduction peut être appliquée dans n'importe quel contexte, en particulier dans les substitutions et sous les abstractions. Les formes normales de ce calcul sont les termes irréductibles. Suivant la tradition commencée avec $\lambda\sigma$ [ACCL91], nous appelons σ le sous-ensemble de règles

TAB. II.3 – λ_{ca}-réduction

Nom	Réduction			Condition
Beta	$(\lambda x.t)\, v$	$\to_{\mathsf{ca}}$	$t[v/x]$	$\mathsf{fv}(v) = \varnothing$
Var	$x[v/x]$	$\to_{\mathsf{ca}}$	v	
App_1	$(t\, u)[v/x]$	$\to_{\mathsf{ca}}$	$(t[v/x])\, u$	$x \in \mathsf{fv}(t)$
App_2	$(t\, u)[v/x]$	$\to_{\mathsf{ca}}$	$t\, (u[v/x])$	$x \in \mathsf{fv}(u)$
Lam	$(\lambda y.t)[v/x]$	$\to_{\mathsf{ca}}$	$\lambda y.t[v/x]$	
$Copy_1$	$(\delta_x^{y,z}.t)[v/x]$	$\to_{\mathsf{ca}}$	$t[v/y][v/z]$	
$Copy_2$	$(\delta_{x'}^{y,z}.t)[v/x]$	$\to_{\mathsf{ca}}$	$\delta_{x'}^{y,z}.t[v/x]$	
$Erase_1$	$(\epsilon_x.t)[v/x]$	$\to_{\mathsf{ca}}$	t	
$Erase_2$	$(\epsilon_{x'}.t)[v/x]$	$\to_{\mathsf{ca}}$	$\epsilon_{x'}.t[v/x]$	

pour la substitution, c'est-à-dire toutes les règles exceptée *Beta*. Une σ-forme normale est un terme qui ne peut pas être réduit en utilisant les règles de σ.

REMARQUE II.3.8 (*Propriétés syntaxiques*)

Il est difficile (bien qu'*a priori* possible) de donner une caractérisation syntaxique exacte (par exemple une grammaire) des termes de Λ_{ca}. De plus certaines propriétés ne sont ni essentielles ni intéressantes (en particulier certaines propriétés sur l'ordre et le placement des opérateurs de copie). On peut cependant identifier les propriétés syntaxiques importantes suivantes, ce qui fournit une bonne surapproximation de Λ_{ca} (en utilisant également le Corollaire II.3.11).

- Du fait de la condition sur la règle *Beta*, toutes les substitutions produites sont *fermées*, et il n'y a ainsi aucun risque de capture de variable dans les règles *Lam*, $Copy_2$ et $Erase_2$. Ainsi aucune condition n'est requise si nous supposons que nous manipulons des λ_{ca}-termes valides.
- Dans un terme de Λ_{ca}, un sous-terme de la forme $\epsilon_x.t$ apparaît toujours sous la forme de $\lambda x.(\epsilon_x.t)[u_1/y_1]\dots[u_n/y_n]$ ou $(\epsilon_x.t)[u_1/y_1]\dots[u_n/y_n][u/x]$ (avec $y_1, \dots, y_n$ distincts de x). Autrement dit, il y a toujours un lieur pour x immédiatement au-dessus de ce sous-terme (modulo d'autres substitutions). Ce ne serait pas le cas, par exemple, si des substitutions ouvertes pouvaient être effacées.

Dans cette section nous travaillerons seulement avec des λ_{ca}-termes valides (c'est-à-dire des termes dérivés de la compilation de λ-termes, et de leurs réduits qui sont également valides par la Proposition II.3.10 ci-dessous).

EXEMPLE II.3.9

Nous donnons quelques exemples illustratifs. Le premier exemple indique que la réduction est plus forte que la réduction de la Logique Combinatoire de la Section I.3.3 (KI se réduit en une forme normale pure) :

$$\mathsf{KI} = (\lambda x.\lambda y.\epsilon_y.x)(\lambda x.x) \to_{\mathsf{ca}} (\lambda y.\epsilon_y.x)[\lambda x.x/x] \to_{\mathsf{ca}}^{*} \lambda y.\epsilon_y.\lambda x.x.$$

Pour voir que le calcul est faible, considérons :

$$\mathsf{22} = (\lambda f.\lambda x.\delta_f^{g,h}.g(hx))\mathsf{2} \to_{\mathsf{ca}} (\lambda x.\delta_f^{g,h}.g(hx))[\mathsf{2}/f] \to_{\mathsf{ca}}^* \lambda x.\mathsf{2}(\mathsf{2}x).$$

Maintenant il est impossible de dupliquer la variable x dans cette forme normale (faible) : il faut attendre une substitution pour clore le terme avant de continuer. Si nous l'appliquons à deux termes clos, alors la réduction peut continuer jusqu'en forme normale (forte) :

$$\begin{array}{rll}
\mathsf{22II} & \to_{\mathsf{ca}}^* & \mathsf{2(2I)I} \to_{\mathsf{ca}}^* (\lambda x.\delta_f^{g,h}.g(hx))[\mathsf{2I}/f]\mathsf{I} \\
& \to_{\mathsf{ca}} & (\lambda x.\delta_f^{g,h}.g(hx))[(\lambda x.\delta_f^{g,h}.g(hx))[\mathsf{I}/f]/f]\mathsf{I} \\
& \to_{\mathsf{ca}}^* & (\lambda x.\delta_f^{g,h}.g(hx))[(\lambda x.\mathsf{I}(\mathsf{I}x))/f]\mathsf{I} \\
& \to_{\mathsf{ca}}^* & (\lambda x.(\lambda x.\mathsf{I}(\mathsf{I}x))((\lambda x.\mathsf{I}(\mathsf{I}x))x))\mathsf{I} \\
& \to_{\mathsf{ca}}^* & (\lambda x.\mathsf{I}(\mathsf{I}x))((\lambda x.\mathsf{I}(\mathsf{I}x))\mathsf{I}) \\
& \to_{\mathsf{ca}}^* & (\lambda x.\mathsf{I}(\mathsf{I}x))(\mathsf{I}(\mathsf{II})) \\
& \to_{\mathsf{ca}}^* & (\lambda x.\mathsf{I}(\mathsf{I}x))\mathsf{I} \to_{\mathsf{ca}}^* \mathsf{I}(\mathsf{II}) \to_{\mathsf{ca}}^* \mathsf{I}.
\end{array}$$

Si on étudie en détail cet exemple, il y a plusieurs stratégies d'évaluation possibles. Le meilleur chemin (le plus court) à prendre doit toujours pousser les substitutions fermées à travers les applications, et réduire les termes en forme normale avant de les copier. Cependant, la restriction imposée dans la règle *Beta* ne nous permet pas de réduire $\lambda x.\mathsf{I}\,(\mathsf{I}x)$ avant de le copier, ainsi nous ne pouvons pas partager son calcul. Bien que λ_{ca} autorise davantage de partage que la réduction faible standard en forme normale de tête faible (Proposition II.3.44 ci-dessous), il n'est pas entièrement satisfaisant de ce point de vue. Pour pallier ce défaut, nous présentons une amélioration de la stratégie dans la Section II.3.3 .

Comme dernier exemple, considérons le terme Y, qui, dans le λ-calcul, a une forme normale de tête faible , mais aucune forme normale de tête. Cependant, bien que nous réduisions sous les abstractions, il n'y a aucune suite infinie de réductions partant de Y dans λ_{ca} puisque le seul rédex (souligné ci-dessous) a une variable libre h dans l'argument :

$$\lambda f.\delta_f^{g,h}.\underline{(\lambda x.g\,(\delta_x^{x',x''}.x'x''))(\lambda x.h\,(\delta_x^{x',x''}.x'x''))}.$$

Il est facile de voir cependant que le terme YI produit une réduction infinie, car l'argument est un terme fermé.

Notons que nous n'avons jamais besoin d'α-conversion dans les réductions, bien que le même nom de variable puisse apparaître plusieurs fois dans le terme en cours de réduction.

PROPOSITION II.3.10 (*Correction de* $\to_{\mathsf{ca}}$)

Soit t un λ_{ca}-terme valide, et $t \to_{\mathsf{ca}} u$, alors :

1. $\mathsf{fv}(t) = \mathsf{fv}(u)$;
2. u est aussi un λ_{c}-terme valide.

Preuve.

1. Seuls les termes clos sont effacés, et aucune nouvelle variable n'est créée pendant la réduction.

2. Il suffit de montrer que la relation $\to_{\mathsf{ca}}$ préserve les contraintes de variables données dans la Table II.2. Nous distinguons plusieurs cas :

   Beta : $(\lambda x.t)\,u \to_{\mathsf{ca}} t[u/x]$ si $\mathsf{fv}(u) = \varnothing$.

   Par hypothèse, $x \in \mathsf{fv}(t)$, et clairement $(\mathsf{fv}(t)\setminus\{x\})\cap\mathsf{fv}(u) = \varnothing$ parce que $\mathsf{fv}(u) = \varnothing$. Ainsi $t[u/x]$ est un λ_{c}-terme valide.

   Var : $x[v/x] \to_{\mathsf{ca}} v$. Trivial.

   App_1 : $(t\,u)[v/x] \to_{\mathsf{ca}} (t[v/x])\,u$ si $x \in \mathsf{fv}(t)$.

   Le terme $t[v/x]$ est valide, car $x \in \mathsf{fv}(t)$ et $\mathsf{fv}(v) = \varnothing$. Par hypothèse $\mathsf{fv}(t) \cap \mathsf{fv}(u) = \varnothing$, donc $(\mathsf{fv}(t) \setminus \{x\}) \cap \mathsf{fv}(u) = \varnothing$ d'où $(t[v/x])\,u$ est un λ_{c}-terme valide.

   App_2 : $(t\,u)[v/x] \to_{\mathsf{ca}} t\,(u[v/x])$ si $x \in \mathsf{fv}(u)$.

   Suit le même raisonnement que le cas App_1.

   Lam : $(\lambda y.t)[v/x] \to_{\mathsf{ca}} \lambda y.t[v/x]$.

   Le terme $t[v/x]$ est valide car $x \in \mathsf{fv}(t)$ et $\mathsf{fv}(v) = \varnothing$. Or, $y \in \mathsf{fv}(t)$ donc $\lambda y.t[v/x]$ est un λ_{c}-terme valide.

   $Copy_1$: $(\delta_x^{y;z}.t)[v/x] \to_{\mathsf{ca}} t[v/y][v/z]$.

   Le terme $t[v/y]$ est valide car $y \in \mathsf{fv}(t)$ et $\mathsf{fv}(v) = \varnothing$. De façon similaire, $t[v/y][v/z]$ est aussi un λ_{c}-terme valide parce que $z \in \mathsf{fv}(t[v/y])$.

   $Copy_2$: $(\delta_{x'}^{y;z}.t)[v/x] \to_{\mathsf{ca}} \delta_{x'}^{y;z}.t[v/x]$.

   Le terme $t[v/x]$ est valide pour les mêmes raisons que ci-dessus. Maintenant, $\delta_{x'}^{y;z}.t[v/x]$ est valide car y et z n'ont pas été liés par la substitution.

   $Erase_1$: $(\epsilon_x.t)[v/x] \to_{\mathsf{ca}} t$. Trivial.

   $Erase_2$: $(\epsilon_{x'}.t)[v/x] \to_{\mathsf{ca}} \epsilon_{x'}.t[v/x]$.

   En utilisant le même raisonnement que ci-dessus, le terme $t[v/x]$ est valide, et comme $x' \notin \mathsf{fv}(t[v/x])$, le terme $\epsilon_{x'}.t[v/x]$ est également valide.

□

Corollaire II.3.11
$\Lambda_{\mathsf{ca}} \subset \Lambda_{\mathsf{c}}$.

Preuve. Par la Proposition II.3.10, point 2, tous les λ_{ca}-termes sont des λ_{c}-termes valides. L'inclusion est stricte parce que $\lambda z.(\lambda x.xy)[z/y]$ est un λ_{c}-terme valide, mais pas un λ_{ca}-terme valide (la substitution est ouverte). □

Afin de prouver que les substitutions sont bien définies dans ce système, nous montrons que σ est fortement normalisant et confluent, et que les substitutions closes ne peuvent pas rester bloquées. Nous prouvons également que la commutation des substitutions peut être dérivée dans le système, ce qui justifie l'exclusion de la règle *Comp*. Pour prouver la terminaison des règles de substitution nous utilisons les résultats suivants :

DÉFINITION II.3.12 (*Distance d'une variable*)

Soit $|t|_x$ la *distance* de la racine du terme t à l'occurrence de la variable libre x, qui est définie par :

$$\begin{array}{llllll} |x|_x & = & |\epsilon_x.t|_x & = & 1 & \\ |\lambda y.t|_x & = & |\delta^{y,z}_{x'}.t|_x & = & |\epsilon_y.t|_x = 1+|t|_x & \\ |tu|_x & = & |t[u/y]|_x & = & 1+|t|_x & (\text{si } x \in \mathsf{fv}(t)) \\ |tu|_x & = & |t[u/y]|_x & = & 1+|u|_x & (\text{si } x \in \mathsf{fv}(u)) \\ |\delta^{y,z}_x.t|_x & = & 1+|t|_y+|t|_z & & & \end{array}$$

PROPOSITION II.3.13 (*Terminaison des substitutions*)

Il n'existe pas de suite infinie de réductions dans λ_{ca} en utilisant seulement les règles de σ.

Preuve. Nous définissons une interprétation qui associe à chaque terme t un multi-ensemble contenant un élément pour chaque sous-terme $w[s/x]$ de t, qui est $|w|_x$. Chaque application d'une règle de substitution diminue l'interprétation du terme, puisque nous appliquons toujours une substitution à un sous-terme de t ou nous l'effaçons, et la distance est strictement réduite. □

PROPOSITION II.3.14 (*Confluence des substitutions*)

Si $t =_\sigma u$ alors il existe s tel que $t \to^*_\sigma s$ et $u \to^*_\sigma s$.

Preuve. Puisqu'il n'y a aucune paire critique dans σ, le système est localement confluent, et puisqu'il termine, il est également confluent par le Lemme de Newman I.1.4. On peut aussi noter que les règles de substitution sont linéaires gauche, donc nous pouvons déduire la confluence directement puisque le système est orthogonal [Klo80] (on obtient cependant un résultat plus faible, puisqu'il faut alors considérer les termes modulo α-conversion). □

Nous nous intéressons maintenant au problème de la complétion des substitutions dans Λ_{ca}, c'est-à-dire au problème de savoir si les substitutions peuvent être propagées jusqu'à destination en considérant toujours que les termes sont dérivés de la compilation de λ-termes. En particulier, ceci implique que toutes les substitutions sont fermées.

LEMME II.3.15

Si v est un terme clos, alors $t[v/x]$ n'est pas une forme normale dans λ_{ca}.

Preuve. Par induction sur la structure de t. Puisque $t[v/x]$ est un λ_{ca}-terme, $x \in \mathsf{fv}(t)$, donc t ne peut pas être $\star$. Si t est une variable, application, abstraction, effacement ou copie, alors nous pouvons appliquer une des règles de substitution. Si $t = u[w/y]$ alors $x \in \mathsf{fv}(u)$ (x ne peut pas être libre dans w cars la règle *Beta* qui a créé la substitution ne peut pas avoir été appliquée avec un argument ouvert), donc nous pouvons appliquer l'hypothèse d'induction sur $u[w/y]$. □

Par conséquent, les substitutions ne restent pas bloquées :

PROPOSITION II.3.16 (*Complétion des substitutions closes*)

Tout terme de Λ_{ca} a une σ-forme normale qui est un terme pur (sans substitution).

Preuve. Toutes les substitutions de λ_{ca} sont fermées par définition. Ainsi par le Lemme II.3.15 n'importe quel sous-terme $t[v/x]$ peut être réduit, et puisque par la Proposition II.3.13 ce processus termine, nous atteindrons nécessairement un terme sans substitution. □

Nous montrons maintenant qu'il est possible de déduire la commutation des substitutions closes.

LEMME II.3.17 (*Commutation des substitutions*)

Dans Λ_{ca}, pour tous termes t, u, v, nous avons :

$$t[u/x][v/y] =_\sigma t[v/y][u/x].$$

Preuve. Moralement, ce lemme est trivial : il suffit de pousser les substitutions jusqu'au bout (ce qui est rendu possible par la Proposition II.3.16). Cependant, cela suppose que notre système implante correctement l'opération de substitution, c'est pourquoi nous préférons donner également une preuve plus syntaxique.

Soit $N(t, x, y)$ le nombre de positions des variables de t affectées par les substitutions $[u/x]$ et $[v/y]$ (comme u et v sont clos, les substitutions n'introduisent pas de nouvelles variables). Nous procédons par induction sur $(N(t, x, y), t)$ en utilisant l'ordre $(>_{\mathbb{N}}, \triangleright)_{lex}$ où $>_{\mathbb{N}}$ est l'ordre usuel sur les entiers naturels, $\triangleright$ dénote la relation super-terme, et *lex* indique la composition lexicographique. Sans perte de généralité nous supposons que t est une σ-forme normale (voir la Proposition II.3.16), et nous considérons tous les cas possibles pour t.

- t ne peut pas être une variable ou $\star$ puisque $x, y \in \mathsf{fv}(t)$ par définition de λ_{ca}.
- Si $t = \lambda z.s$, la propriété tient par induction.
- Si $t = s\,w$, alors nous distinguons deux cas : si $x, y \in \mathsf{fv}(s)$ ou $x, y \in \mathsf{fv}(w)$, le résultat suit par induction. Sinon, supposons $x \in \mathsf{fv}(s)$ et $y \in \mathsf{fv}(w)$ (le cas symétrique est semblable). Alors $t[u/x][v/y] \to^*_{\mathsf{ca}} (s[u/x])(w[v/y])$ et $t[v/y][u/x] \to^*_{\mathsf{ca}} (s[u/x])(w[v/y])$.
- Si $t = \epsilon_z.s$, alors le résultat suit par induction. Si $t = \epsilon_x.s$ (le cas $t = \epsilon_y.s$ est analogue), alors $t[u/x][v/y] \to_{\mathsf{ca}} s[v/y]$ et $t[v/y][u/x] \to_{\mathsf{ca}} (\epsilon_x.s[v/y])[u/x] \to_{\mathsf{ca}} s[v/y]$.
- Si $t = \delta_z^{z',z''}.s$, le résultat suit par induction. Si $t = \delta_x^{x',x''}.s$ alors $t[u/x][v/y] \to_{\mathsf{ca}} s[u/x'][u/x''][v/y]$, et $t[v/y][u/x] \to_{\mathsf{ca}} (\delta_x^{x',x''}.s[v/y])[u/x] \to_{\mathsf{ca}} s[v/y][u/x'][u/x'']$. Soit $s[u/x'] \to^*_\sigma t'$ en σ-forme normale. Puisque $N(t', x'', y) < N(t, x, y)$, par induction $t'[u/x''][v/y] =_\sigma t'[v/y][u/x'']$. D'où $s[u/x'][u/x''][v/y] =_\sigma s[u/x'][v/y][u/x'']$. Encore par induction, $s[u/x'][v/y] =_\sigma s[v/y][u/x']$. D'où $s[u/x'][u/x''][v/y] =_\sigma s[v/y][u/x'][u/x'']$. Nous en déduisons $t[u/x][v/y] =_\sigma t[v/y][u/x]$. Le cas $t = \delta_y^{y',y''}.s$ est analogue.
- Le cas $t = s[w/z]$ n'est pas possible puisque par hypothèse t est une σ-forme normale (voir la Proposition II.3.16).

□

La confluence locale est une conséquence facile du lemme précédent.

PROPOSITION II.3.18 (*Confluence locale*)
Soit $t \in \Lambda_{\mathsf{ca}}$. Si $t \to_{\mathsf{ca}} u$ et $t \to_{\mathsf{ca}} v$, il existe un terme $s \in \Lambda_{\mathsf{ca}}$ tel que $u \to_{\mathsf{ca}}^* s$ et $v \to_{\mathsf{ca}}^* s$.

Preuve. Il y a une seule paire critique, produite par la superposition de *Beta* et App_1 (les autres superpositions potentielles du système sont éliminées grâce aux conditions de garde des règles). Si $y \in \mathsf{fv}(\lambda x.t)$, et $\mathsf{fv}(u) = \mathsf{fv}(v) = \varnothing$ alors :

$$\begin{array}{ccccc} ((\lambda x.t)u)[v/y] & \longrightarrow & ((\lambda x.t)[v/y])u & \longrightarrow & (\lambda x.t[v/y])u \\ \downarrow & & & & \downarrow \\ t[u/x][v/y] & & =_\sigma & & t[v/y][u/x] \end{array}$$

Le Lemme II.3.17 et la Proposition II.3.14 montrent que cette paire critique peut être jointe en utilisant les règles de σ. On peut noter que si $y \in \mathsf{fv}(u)$ alors la paire critique disparaît, car l'étape de *Beta*-réduction est bloquée jusqu'à ce que la substitution soit effectuée. □

Bien que λ_{ca} bénéficie d'autres propriétés utiles, telles que la préservation de la normalisation forte et la confluence, nous ne poursuivrons pas son étude car nous allons présenter une version améliorée du calcul dans la section suivante, pour laquelle toutes ces propriétés seront prouvées.

II.3.3 FONCTIONS FERMÉES : λ_{cf}

La condition sur la règle *Beta* de λ_{ca} restreint le partage, comme le montre l'Exemple II.3.9. Pour éviter l'α-conversion il suffit d'imposer que les substitutions soient fermées dans les règles impliquant un lieur. C'est l'idée sous-jacente au système λ_{cf} , qui fournit un ensemble alternatif de règles de réduction pour les λ_{c}-termes. La règle *Comp* (qui autorise la composition des substitutions) ne sera plus dérivée, mais sera ajoutée d'une façon contrôlée afin d'éviter la non-terminaison. Rappelons que, d'après la Section II.2, la règle *Comp* avec la syntaxe linéaire se divise en deux règles :

$$\begin{array}{lcll} t[u/x][v/y] & \to & t[u[v/y]/x] & (y \in \mathsf{fv}(u)) \\ t[u/x][v/y] & \to & t[v/y][u/x] & (y \in \mathsf{fv}(t)). \end{array}$$

Pour λ_{ca}, aucune des deux règles n'était nécessaire, mais nous pouvions dériver la relation d'inter-convertibilité correspondant à la seconde. Nous ajouterons maintenant la première puisqu'elle permet au processus de substitution de progresser vers l'accomplissement. Cette règle laisse néanmoins une certaine liberté sur le choix de la stratégie d'application des substitutions, avec des conséquences étonnantes.

Cependant, en abandonnant λ_{ca}, c'est-à-dire en autorisant certaines substitutions ouvertes, nous pouvons avoir un problème de confluence dans le système dû à la paire critique *Beta*/*Lam*. Considérons le terme $((\lambda x.t)\, u)[v/y]$, où $y \in \mathsf{fv}(t)$, $\mathsf{fv}(u) \neq \varnothing$, et $\mathsf{fv}(v) = \varnothing$.

$$\begin{array}{ccccc} ((\lambda x.t)\, u)[v/y] & \longrightarrow & ((\lambda x.t)[v/y])\, u & \longrightarrow & (\lambda x.t[v/y])\, u \\ \downarrow & & & & \downarrow \\ t[u/x][v/y] & & \overset{?}{\longleftrightarrow} & & t[v/y][u/x] \end{array}$$

Puisque nous avons éliminé un des cas pour la composition des substitutions, nous ne pouvons pas fermer ce diagramme (le Lemme II.3.17 ne peut pas être appliqué avec une substitution ouverte). Cependant, nous pouvons éviter ce problème de confluence en autorisant la *Beta*-réduction seulement quand la fonction est fermée :

$$(\lambda x.t)\, u \quad \to \quad t[u/x] \qquad \text{si } \mathsf{fv}(\lambda x.t) = \varnothing$$

ce qui résout le problème ci-dessus en éliminant la branche gauche du diagramme. Dans d'autres travaux sur les substitutions explicites, il est intéressant de noter que c'est la branche droite qui est éliminée, voir par exemple [Muñ96].

Partant de ces observations, nous obtenons un nouveau calcul basé sur la réduction de fonctions closes, λ_{cf}.

Définition II.3.19 (λ_{cf}-*calcul*)

L'ensemble Λ_{cf} des λ_{cf}-*termes valides* contient tous les λ_{c}-termes qui sont l'image de la fonction de compilation et leurs réduits par la relation de réduction $\to_{\mathsf{cf}}$ produite par le système de récriture conditionnel de la Table II.4. De nouveau, nous appelons σ l'ensemble de règles pour la substitution, c'est-à-dire, toutes les règles exceptée *Beta*. Comme d'habitude, nous écrivons $\to^*_{\mathsf{cf}}$ pour la clôture réflexive transitive de $\to_{\mathsf{cf}}$. Il est à noter que les règles de réduction peuvent être appliquées dans tous les contextes. En particulier, des réductions peuvent avoir lieu dans les substitutions, et sous les abstractions.

Tab. II.4 – λ_{cf}-réduction

Nom	Réduction			Condition
Beta	$(\lambda x.t)u$	$\to_{\mathsf{cf}}$	$t[u/x]$	$\mathsf{fv}(\lambda x.t) = \varnothing$
Var	$x[v/x]$	$\to_{\mathsf{cf}}$	v	
App_1	$(t\,u)[v/x]$	$\to_{\mathsf{cf}}$	$(t[v/x])\,u$	$x \in \mathsf{fv}(t)$
App_2	$(t\,u)[v/x]$	$\to_{\mathsf{cf}}$	$t\,(u[v/x])$	$x \in \mathsf{fv}(u)$
Lam	$(\lambda y.t)[v/x]$	$\to_{\mathsf{cf}}$	$\lambda y.t[v/x]$	$\mathsf{fv}(v) = \varnothing$
$Copy_1$	$(\delta^{y,z}_x.t)[v/x]$	$\to_{\mathsf{cf}}$	$t[v/y][v/z]$	$\mathsf{fv}(v) = \varnothing$
$Copy_2$	$(\delta^{y,z}_{x'}.t)[v/x]$	$\to_{\mathsf{cf}}$	$\delta^{y,z}_{x'}.t[v/x]$	
$Erase_1$	$(\epsilon_x.t)[v/x]$	$\to_{\mathsf{cf}}$	t	$\mathsf{fv}(v) = \varnothing$
$Erase_2$	$(\epsilon_{x'}.t)[v/x]$	$\to_{\mathsf{cf}}$	$\epsilon_{x'}.t[v/x]$	
Comp	$t[w/y][v/x]$	$\to_{\mathsf{cf}}$	$t[w[v/x]/y]$	$x \in \mathsf{fv}(w)$

Remarque II.3.20

Il y a un certain nombre de variantes possibles des règles de réduction :

- La règle $Erase_2$ n'a aucune condition de garde. Il est clair, à partir des contraintes sur les variables de la Table II.2, que si $(\epsilon_{x'}.t)[v/x]$ est un terme bien formé, alors $\epsilon_{x'}.t[v/x]$ l'est aussi, car x' ne peut pas être libre dans v. De même, dans $Copy_2$, x' ne peut pas être libre dans v, et y, z sont des variables fraîches introduites par la compilation.

- Les règles App_1, App_2 et *Comp* peuvent également être fermées. Ceci donne un calcul plus faible, avec une stratégie plus dirigée, mais en compensation, cela privilégie le chemin de réduction le plus court, et évite certaines paires critiques du système. Cependant, nous préférons autoriser ces règles avec des termes ouverts puisque ce calcul donne les conditions minimales pour la réduction close avec des propriétés raisonnables. Nous reviendrons sur ce choix dans le Chapitre IV.
- Nous pourrions également ajouter un certain nombre de règles d'optimisation, par exemple $t[x/x] \to t$, mais nous préférons ne pas les inclure dans la théorie de base. Celle-ci correspond à $t[id] \to t$ dans d'autres calculs avec substitutions explicites.

Puisque la syntaxe des termes est la même que dans λ_{ca}, nous utilisons les mêmes fonctions de compilation et de décompilation pour traduire les λ-termes. L'ensemble Λ_{cf} des *termes valides* est cependant différent de Λ_{ca} (par exemple le terme $\lambda z.(\lambda x.x\, y)[z/y]$ est dans Λ_{cf} mais pas dans Λ_{ca}). Néanmoins, la propriété syntaxique de placement des effacements de Λ_{ca} subsiste dans Λ_{cf} (car on n'efface toujours pas de substitution ouverte) :

Lemme II.3.21

> Tout sous-terme $\epsilon_x.t$ d'un terme de Λ_{cf} apparaît sous la forme $\lambda x.(\epsilon_x.t)[u_1/y_1]\dots[u_n/y_n]$ ou $(\epsilon_x.t)[u_1/y_1]\dots[u_n/y_n][u/x]$, avec $y_1,\dots,y_n$ distincts de x et $u_1,\dots,u_n$ clos.

Preuve. Induction facile sur la dérivation partant d'un terme compilé (en utilisant l'hypothèse de validité). □

Exemple II.3.22

Le terme $\lambda x.\lambda y.(x\, z)[y/z]$ n'est pas un λ_{cf}-terme valide, puisque la règle *Beta* doit avoir été appliquée quand la fonction $\lambda z.(x\, z)$ n'était pas fermée (ce n'est pas une réduction valide). En outre, notons que $\lambda x.\lambda y.\epsilon_x.y$ n'est pas valide, puisqu'il ne provient pas de la fonction de compilation (ϵ_x n'est pas à la bonne place) et aucune réduction de λ_{cf} ne peut le produire.

Nous donnons maintenant quelques exemples de réductions de λ_{cf}.

Exemple II.3.23

Considérons à nouveau les termes de l'Exemple II.3.6. Le terme KI se réduit toujours en une forme normale pure :

$$\mathsf{KI} = (\lambda x.\lambda y.\epsilon_y.x)(\lambda x.x) \to_{\mathsf{cf}} (\lambda y.\epsilon_y.x)[\lambda x.x/x] \to^*_{\mathsf{cf}} \lambda y.\epsilon_y.\lambda x.x.$$

Bien que ce calcul ne soit pas aussi faible que λ_{ca}, la réduction de $\mathsf{22}$ n'atteint pas une forme normale complète.

$$\begin{array}{rcl}
\mathsf{22} & = & (\lambda f x.\delta_f^{g,h}.g(hx))\mathsf{2} \\
& \to_{\mathsf{cf}} & (\lambda x.\delta_f^{g,h}.g(hx))[\mathsf{2}/f] \\
& \to^*_{\mathsf{cf}} & \lambda x.\mathsf{2}(\mathsf{2}x) \\
& \to_{\mathsf{cf}} & \lambda x.(\lambda x.\delta_f^{g,h}.g(hx))[\mathsf{2}x/f] \\
& \to_{\mathsf{cf}} & \lambda x.(\lambda x.\delta_f^{g,h}.g(hx))[(\lambda x.\delta_f^{g,h}.g(hx))[x/f]/f].
\end{array}$$

Pour continuer la réduction, il suffit d'appliquer ceci à un terme clos, par exemple à I (nous

avions besoin de deux arguments dans λ_{ca}, voir l'Exemple II.3.9).

$$\begin{array}{rll} \mathsf{22I} & \to^*_{\mathsf{cf}} & ((\lambda x.\delta^{g,h}_f.g(hx))[(\lambda x.\delta^{g,h}_f.g(hx))[x/f]/f])[\mathsf{I}/x] \\ & \to^*_{\mathsf{cf}} & (\lambda x.\delta^{g,h}_f.g(hx))[(\lambda x.\delta^{g,h}_f.g(hx))[\mathsf{I}/f]/f] \\ & \to^*_{\mathsf{cf}} & (\lambda x.\delta^{g,h}_f.g(hx))[(\lambda x.\mathsf{I}(\mathsf{I}x))/f] \\ & \to^*_{\mathsf{cf}} & (\lambda x.\delta^{g,h}_f.g(hx))[\mathsf{I}/f] \\ & \to^*_{\mathsf{cf}} & (\lambda x.\mathsf{I}(\mathsf{I}x)) \\ & \to^*_{\mathsf{cf}} & \lambda x.x. \end{array}$$

Ici aussi, il n'y a aucune réduction infinie partant de Y, bien que nous réduisions sous les abstractions, puisque le seul rédex (souligné) a une variable libre g dans la fonction :

$$\lambda f.\delta^{g,h}_f.\underline{(\lambda x.g(\delta^{x',x''}_x.x'x''))(\lambda x.h(\delta^{x',x''}_x.x'x''))}.$$

Le terme YI produit une séquence infinie : $\mathsf{YI} \to^*_{\mathsf{cf}} \mathsf{I}(\mathsf{YI}) \to^*_{\mathsf{cf}} \mathsf{YI} \to^*_{\mathsf{cf}} \cdots$

Proposition II.3.24 (*Correction de* $\to_{\mathsf{cf}}$)

Soit t un λ_{cf}-terme, et $t \to_{\mathsf{cf}} u$, alors :

1. $\mathsf{fv}(t) = \mathsf{fv}(u)$.
2. u est un λ_{c}-terme valide, autrement dit $\to_{\mathsf{cf}}$ préserve les contraintes de variables de la Table II.2.

Preuve. Les deux points suivent le même raisonnement que pour λ_{ca}, Proposition II.3.10. □

Corollaire II.3.25

$\Lambda_{\mathsf{cf}} \subset \Lambda_{\mathsf{c}}$.

Preuve. Par la Proposition II.3.24, Point 2, tous les λ_{cf}-termes sont des λ_{c}-termes valides. L'inclusion est stricte parce que $\lambda x.\lambda y.(x\,z)[y/z]$ est un λ_{c}-terme valide, mais pas un λ_{cf}-terme valide. □

Dans le reste de cette section, nous supposons donc que tous les termes sont dans Λ_{cf}. Nous prouvons la terminaison des règles de substitution en utilisant la même technique d'interprétation que pour λ_{ca} :

Proposition II.3.26 (*Terminaison des substitutions*)

Il n'y a aucune séquence infinie de réductions dans λ_{cf} en utilisant seulement les règles de σ.

Une autre propriété importante du système est que les substitutions fermées ne restent pas bloquées.

Lemme II.3.27

Si v est un terme clos, alors $t[v/x]$ n'est pas une forme normale dans λ_{cf}.

Preuve. Par induction sur la structure de t. Puisque $t[v/x]$ est un λ_{cf}-terme, $x \in \mathsf{fv}(t)$, donc t ne peut pas être $\star$. Si t est une variable, application, abstraction, effacement ou copie, alors

nous pouvons appliquer une des règles pour la substitution. Si $t = u[w/y]$ il y a alors deux cas :

1. Si $x \in \mathsf{fv}(w)$ nous pouvons appliquer la règle *Comp*.
2. Si $x \in \mathsf{fv}(u)$ alors w doit être clos, sinon la règle *Beta* n'a pas pu être appliquée pour créer $[v/x]$. Par conséquent nous pouvons appliquer l'hypothèse d'induction sur $u[w/y]$.

□

Pour λ_{cf}, nous n'avons pas un résultat analogue à la Proposition II.3.16 pour λ_{ca} (tous les termes ne peuvent pas être réduits en termes purs dans λ_{cf}). Nous regardons maintenant le problème de la préservation de la normalisation forte (un calcul de substitutions explicites préserve la normalisation forte si la compilation d'un λ-terme fortement normalisable est fortement normalisable). A la différence de beaucoup de systèmes de substitutions explicites, λ_{cf} et λ_{ca} préservent la normalisation forte. Notre preuve est inspirée de celle de λ_υ [LRD95] ; nous la donnons pour λ_{cf} mais elle s'applique aussi à λ_{ca}. Nous définissons d'abord une notion de dérivation infinie minimale. Intuitivement, une dérivation est minimale si nous réduisons toujours un rédex le plus bas possible pour conserver la non-terminaison. Nous dénotons par $\rightarrow_{Beta,p}$ une *Beta*-réduction à la position p.

Définition II.3.28 (*Dérivation minimale*)

Une dérivation infinie de λ_{cf}

$$t_1 \rightarrow_{Beta,p_1} t'_1 \rightarrow^*_\sigma \cdots t_i \rightarrow_{Beta,p_i} t'_i \rightarrow^*_\sigma \cdots$$

est *minimale* si pour toute autre dérivation infinie

$$t_1 \rightarrow_{Beta,p_1} t'_1 \rightarrow^*_\sigma \cdots t_i \rightarrow_{Beta,q} u \rightarrow^*_\sigma \cdots$$

nous avons $p_i \not\leq_{pref} q$, c'est-à-dire $q \neq p_i p'$ pour tout p'.

En d'autres termes, dans n'importe quelle autre dérivation infinie, ou bien p_i et q sont disjoints ou bien q et au-dessus de p_i, ce qui signifie que le *Beta*-rédex que nous réduisons est l'un des plus bas. En particulier, si $(\lambda x.t)\,u$ est un *Beta*-rédex dans une dérivation minimale, alors t et u sont fortement normalisables.

Lemme II.3.29

1. Si $t \rightarrow^*_\sigma t'$, alors $t^* = t'^*$.
2. Si $t \rightarrow_{Beta} t'$ est un pas de réduction dans une dérivation minimale partant de la compilation d'un λ-terme, alors $t^* \rightarrow^+_\beta t'^*$.

Preuve.

1. Rapide inspection des règles de substitution.
2. Le terme t contient un *Beta*-rédex, et l'hypothèse de minimalité pour la dérivation assure que dans le décompilé t^* ce rédex n'est pas effacé (mais il peut naturellement être copié). En effet, supposons le contraire. D'après la Définition II.3.4 et le Lemme II.3.21,

t contient un sous-terme de la forme $(\epsilon_x.t)[u_1/y_1]\dots[u_n/y_n][r/x]$ où r contient le *Beta*-rédex de ce pas de réduction. Par minimalité, la règle $Erase_1$ ne sera jamais appliquée sur ce sous-terme (sinon, il n'aurait pas fallu réduire r). Or, cette substitution pour x provient d'une *Beta*-réduction antérieure. Il est alors facile d'extraire une autre dérivation infinie dans laquelle cette *Beta*-réduction n'a pas eu lieu : les étapes internes ou indépendantes sont les mêmes ; les seules étapes à la racine de ce sous-terme sont forcément des *Comp* et sont remplacées par des App_2. Ceci contredit l'hypothèse de minimalité. Par conséquent, on a bien $t^* \to_\beta^+ t'^*$.

□

Remarque II.3.30

La preuve ci-dessus dépend fortement du fait que l'on puisse identifier précisément l'abstraction correspondant à un effacement : c'est trivial pour un terme issu de la compilation, et les règles de réduction ne bouleversent pas la situation, en particulier parce que l'on n'efface pas de termes ouverts.

Nous prouverons un résultat légèrement plus général que la préservation de la normalisation forte qui nous servira également pour prouver la terminaison des λ_{cf}-termes typables dans la Section II.4.1.

Proposition II.3.31

Si t^* est un λ-terme fortement normalisable, alors t est fortement normalisable.

Preuve. Par l'absurde, supposons qu'il y ait une suite infinie de réductions à partir de t dans λ_{cf}. En particulier, il y a une suite infinie minimale de réductions

$$t \to_\sigma^* t_1 \to_{Beta} t_2 \to_\sigma^* t_3 \to_{Beta} t_4 \cdots$$

Puisque les règles de substitution terminent, chaque dérivation infinie contient un nombre infini d'applications de *Beta*. Par le Lemme II.3.29, nous obtenons une dérivation infinie pour t^* (contradiction) :

$$t^* = t_1^* \to_\beta^+ t_2^* = t_3^* \to_\beta^+ t_4^* \cdots$$

□

Proposition II.3.32 (*Préservation de la normalisation forte*)

Si t est la traduction d'un λ-terme fortement normalisable s alors t est fortement normalisable.

Preuve. Conséquence directe de la Proposition II.3.31 puisque $t^* = s$ par la Proposition II.3.5. □

Nous tournons maintenant notre attention vers le problème de la confluence[3] de λ_{cf}. Il est facile de voir que le système est localement confluent.

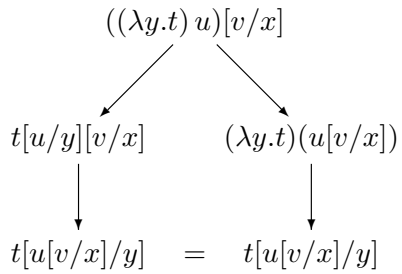
[3] Puisque la relation de récriture est définie seulement sur les termes de base, c'est un problème de confluence de base (*ground confluence*).

PROPOSITION II.3.33 (*Confluence locale*)

Si $t \to_{\mathsf{cf}} u$ et $t \to_{\mathsf{cf}} v$ alors il existe un terme s tel que $u \to_{\mathsf{cf}}^* s$ et $v \to_{\mathsf{cf}}^* s$.

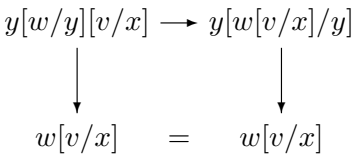
Preuve. Puisque le système n'est pas défini modulo α-conversion, on peut utiliser le lemme des paires critiques pour les systèmes de récriture du premier ordre. Il y a sept paires critiques à considérer, qui convergent comme montré ci-dessous. On élimine toutes les autres superpositions potentielles du système en raison des contraintes sur les variables libres.

1. $((\lambda y.t)\, u)[v/x]$ où $\mathsf{fv}(\lambda y.t) = \varnothing$ et $x \in \mathsf{fv}(u)$:

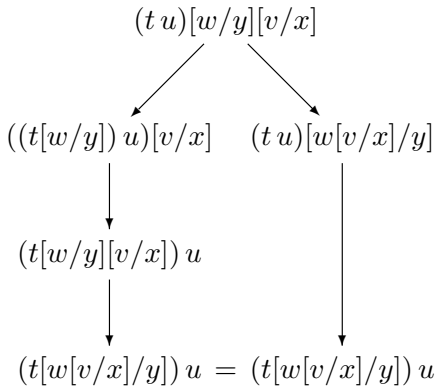


   A noter que si $x \in \mathsf{fv}(t)$, la paire critique est éliminée, puisque l'étape de *Beta*-réduction est bloquée jusqu'à ce que la substitution soit effectuée.

2. $y[w/y][v/x]$ où $x \in \mathsf{fv}(w)$:

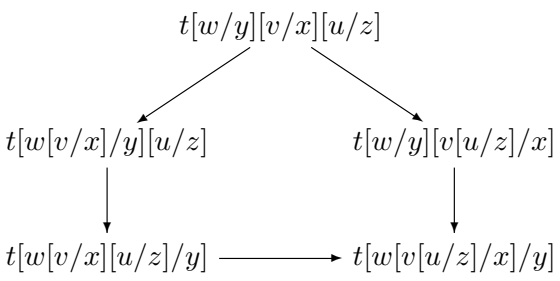


3. $(t\, u)[w/y][v/x]$ où $y \in \mathsf{fv}(t)$ et $x \in \mathsf{fv}(w)$:



4. Le cas où $y \in \mathsf{fv}(u)$ est presque identique au cas précédent.
5. $(\epsilon_{x'}.t)[w/y][v/x]$ où $x \in \mathsf{fv}(w)$: similaire au cas pour l'application ci-dessus.

6. $(\delta_z^{z',z''}.t)[w/y][v/x]$ où $x \in \mathsf{fv}(w)$: à nouveau similaire aux cas application et effacement ci-dessus.
7. $t[w/y][v/x][u/z]$ où $z \in \mathsf{fv}(v)$ et $x \in \mathsf{fv}(w)$:



□

Remarque II.3.34

Si nous limitons toutes les règles de λ_{cf} aux substitutions fermées, alors il y a seulement une paire critique qui a besoin d'être analysée pour la confluence locale (la première), et dans ce cas nous obtenons la confluence forte directement, plutôt que la confluence locale (mais nous pouvons faire moins de réductions naturellement).

Corollaire II.3.35 (*Confluence des substitutions*)

Si $t =_\sigma u$ alors il existe s tel que $t \to_\sigma^* s$ et $u \to_\sigma^* s$.

Preuve. Par le Lemme de Newman I.1.4, puisque les règles de substitution terminent par la Proposition II.3.26. □

Grâce au lemme de Newman, nous pouvons également déduire la confluence pour les λ_{cf}-termes fortement normalisable. Mais nous pouvons prouver la confluence sans supposer la terminaison. Pour ceci nous prouvons d'abord la commutation de *Beta* et des σ-réductions.

Lemme II.3.36 (*Commutation de Beta et σ dans λ_{cf}*)

Si $b \ {}_{Beta}{}^{*}\!\leftarrow s \to_\sigma^* a$ alors il existe c tel que $b \to_\sigma^* c \ {}_{Beta}{}^{*}\!\leftarrow a$.

Preuve. Nous utilisons une relation auxiliaire $\Rightarrow$, qui effectue les *Beta*-réductions en une étape en parallèle. Elle est définie par induction :

1. $t \Rightarrow t$,
2. $(\lambda x.t)\, u \Rightarrow t'[u'/x]$ si $\mathsf{fv}(\lambda x.t) = \varnothing$, $t \Rightarrow t'$ et $u \Rightarrow u'$,
3. $\lambda x.t \Rightarrow \lambda x.t'$ si $t \Rightarrow t'$,
4. $t\, u \Rightarrow t'\, u'$ si $t \Rightarrow t'$ et $u \Rightarrow u'$,
5. $t[v/x] \Rightarrow t'[v'/x]$ si $t \Rightarrow t'$ et $v \Rightarrow v'$,
6. $\delta_x^{y,z}.t \Rightarrow \delta_x^{y,z}.t'$ si $t \Rightarrow t'$,
7. $\epsilon_x.t \Rightarrow \epsilon_x.t'$ si $t \Rightarrow t'$.

Notons d'abord que $\Rightarrow^*$ coïncide avec $\rightarrow^*_{Beta}$: en effet, $\rightarrow_{Beta} \subseteq \Rightarrow$ par définition, et $\Rightarrow \subseteq \rightarrow^*_{Beta}$ par une induction facile.

Pour prouver le lemme, il suffit de montrer que si $b \Leftarrow s \rightarrow_\sigma a$ alors il existe un terme c tel que $b \rightarrow_\sigma c \Leftarrow a$. Alors nous pouvons fermer le diagramme $b \;{}^*_{Beta}\!\leftarrow s \rightarrow^*_\sigma a$ (ce qui est équivalent à $b \;{}^*\!\Leftarrow s \rightarrow^*_\sigma a$), par induction sur la longueur de la dérivation $s \Rightarrow^* b$.

Nous procédons par induction sur la définition de $s \Rightarrow b$. Nous distinguons les cas suivants :

1. $s = b$. Alors nous prenons $c = a$.
2. $s = (\lambda x.t)\, u \Rightarrow b = t'[u'/x]$, $\mathsf{fv}(\lambda x.t) = \varnothing$, $t \Rightarrow t'$ et $u \Rightarrow u'$. Puisqu'aucune règle de σ ne s'applique à la racine de s, alors soit $t \rightarrow_\sigma t''$ (et $\mathsf{fv}(\lambda x.t'') = \varnothing$ par la Proposition II.3.24), soit $u \rightarrow_\sigma u''$. Dans le premier cas, par induction, il existe un terme t''' tel que $t' \rightarrow_\sigma t'''$ et $t'' \Rightarrow t'''$, donc nous pouvons prendre $c = t'''[u'/x]$. Dans le second cas, par induction, il existe u''' tel que $u' \rightarrow_\sigma u'''$ et $u'' \Rightarrow u'''$, d'où nous pouvons prendre $c = t'[u'''/x]$.
3. Dans les cas $s = \lambda x.t \Rightarrow b = \lambda x.t'$ où $t \Rightarrow t'$, $s = tu \Rightarrow b = t'u'$ où $t \Rightarrow t'$ et $u \Rightarrow u'$, $s = \delta^{y,z}_x.t \Rightarrow b = \delta^{y,z}_x.t'$ où $t \Rightarrow t'$, et $s = \epsilon_x.t \Rightarrow b = \epsilon_x.t'$ où $t \Rightarrow t'$, la propriété suit directement par induction.
4. $s = t[v/x] \Rightarrow b = t'[v'/x]$, $t \Rightarrow t'$ et $v \Rightarrow v'$. Nous distinguons deux cas selon la position de l'étape de σ-réduction $s \rightarrow_\sigma a$.
   - Si elle ne s'applique pas à la racine de s alors la propriété tient par induction.
   - Si elle s'applique à la racine, alors il y a une substitution θ et une règle $l \rightarrow r$ dans σ telle que $s = l\theta$ et $a = r\theta$.
     - Si tous les *Beta*-rédex contractés lors de la réduction $s \Rightarrow b$ sont sous des variables de l (c'est-à-dire si elles sont dans θ) alors ces variables sont identifiées de façon unique (car l est linéaire gauche) et nous pouvons donc définir une substitution θ' telle que $\theta \Rightarrow \theta'$ et que le diagramme commute : $s = l\theta \rightarrow_\sigma r\theta = a \Rightarrow r\theta' = c$ et $s = l\theta \Rightarrow l\theta' = b \rightarrow_\sigma r\theta' = c$.
     - S'il y a un pas de *Beta*-réduction à la racine de t alors nous avons une paire critique entre la σ-règle appliquée à la racine de s et la règle *Beta* appliquée à la racine de t. Dans ce cas la σ-règle appliquée doit être App_2 (elle ne peut pas être App_1 à cause des restrictions sur les variables). Alors le diagramme commute comme suit : $s = ((\lambda y.w)u)[v/x] \rightarrow_{App_2} (\lambda y.w)(u[v/x]) = a \Rightarrow w'[u'[v'/x]/y] = c$ et $s = ((\lambda y.w)u)[v/x] \Rightarrow b = w'[u'/y][v'/x] \rightarrow_{Comp} w'[u'[v'/x]/y] = c$.

Ceci conclut la preuve. □

Proposition II.3.37 (*Confluence*)

> Soit t un terme de Λ_{cf}. Si $t \rightarrow^*_{\mathsf{cf}} u$ et $t \rightarrow^*_{\mathsf{cf}} v$, il y a alors un terme s tel que $u \rightarrow^*_{\mathsf{cf}} s$ et $v \rightarrow^*_{\mathsf{cf}} s$.

Preuve. Nous avons déjà montré la confluence des σ-règles (Corollaire II.3.35) et il est facile de voir que la règle *Beta* seule est confluente. Nous déduisons la confluence de Λ_{cf} à partir du Lemme II.3.36 en utilisant le Lemme de Hindley-Rosen I.1.7. □

II.3.4 Relation avec la β-réduction

La réduction close (λ_{ca} et λ_{cf}) est une théorie strictement plus petite que celle induite par la β-réduction habituelle : tous les β-rédex ne peuvent pas être réduits, et de plus les substitutions peuvent ne pas se propager complètement. Le but de cette section est de prouver qu'elle est tout de même assez puissante pour simuler les stratégies d'évaluation habituelles pour le λ-calcul, ce qui démontre que nous n'avons pas perdu en expressivité en ajoutant des contraintes sévères sur le système de réduction, du moins en ce qui concerne la réduction en forme normale de tête faible de termes clos.

Nous prouvons ci-dessous que nous pouvons simuler l'appel par nom et l'appel par valeur, dans λ_{ca} et λ_{cf} (nous donnons seulement les détails des preuves pour λ_{cf}).

Le lemme suivant sera utile pour établir la correspondance avec la β-réduction dans le λ-calcul.

Lemme II.3.38 (*Substitution*)

Soient t et w des λ-termes tels que $\mathsf{fv}(t) = \{x_1, \ldots, x_n\}$, $n \geq 1$, et $\mathsf{fv}(w) = \varnothing$, alors $([x_1] \ldots [x_n]\langle t \rangle)[\langle w \rangle / x_1] \to_{\mathsf{cf}}^* [x_2] \ldots [x_n]\langle t\{x_1 := w\}\rangle$.

Preuve. Par la Proposition II.3.3 (Partie 1), $\mathsf{fv}([\![w]\!]) = \varnothing$, et $x_1 \in \mathsf{fv}([x_1] \ldots [x_n]\langle t \rangle)$. Nous procédons par induction sur t.

$t = x$. $([x]\langle x \rangle)[\langle w \rangle / x] = x[\langle w \rangle / x] \to_{\mathsf{cf}} \langle w \rangle = \langle x\{x := w\}\rangle$.

$t = \lambda y.u$, $y \in \mathsf{fv}(u)$. Supposons pour simplifier que x est la seule variable libre de t.

$$\begin{array}{rcll} ([x]\langle \lambda y.u \rangle)[\langle w \rangle / x] & = & ([x](\lambda y.[y]\langle u \rangle))[\langle w \rangle / x] & \\ & = & (\lambda y.[x][y]\langle u \rangle)[\langle w \rangle / x] & \\ & \to_{\mathsf{cf}} & \lambda y.([x][y]\langle u \rangle)[\langle w \rangle / x] & \\ & = & \lambda y.[y]\langle u\{x := w\}\rangle & \text{(induction)} \\ & = & \langle \lambda y.u\{x := w\}\rangle & \\ & = & \langle (\lambda y.u)\{x := w\}\rangle. & \end{array}$$

$t = \lambda y.u$, $y \notin \mathsf{fv}(u)$. Suit de façon similaire au cas ci-dessus, avec $\langle \lambda y.u \rangle = \lambda y.\epsilon_y.\langle u \rangle$.

$t = u\,v$, $x \in \mathsf{fv}(u)$. Supposons pour simplifier que x est la seule variable libre de t.

$$\begin{array}{rcll} ([x]\langle u\,v \rangle)[\langle w \rangle / x] & = & ([x](\langle u \rangle\langle v \rangle))[\langle w \rangle / x] & \\ & = & (([x]\langle u \rangle)\langle v \rangle)[\langle w \rangle / x] & \\ & \to_{\mathsf{cf}} & (([x]\langle u \rangle)[\langle w \rangle / x])\langle v \rangle & \\ & = & \langle u\{x := w\}\rangle\langle v \rangle & \text{(induction)} \\ & = & \langle (u\{x := w\})\,v \rangle & \\ & = & \langle (u\,v)\{x := w\}\rangle. & \end{array}$$

$t = u\,v$, $x \in \mathsf{fv}(v)$. Similaire au cas ci-dessus.

$t = u\,v$, $x \in \mathsf{fv}(u) \wedge x \in \mathsf{fv}(v)$.

$$\begin{array}{rcll}
([x]\langle u\,v\rangle)[\langle w\rangle/x] & = & ([x](\langle u\rangle\langle v\rangle))[\langle w\rangle/x] & \\
& = & (\delta_x^{x',x''}.([x']\langle u\rangle\{x := x'\}))([x'']\langle v\rangle\{x := x''\}))[\langle w\rangle/x] & \\
& \to_{\mathsf{cf}} & (([x']\langle u\rangle\{x := x'\})[\langle w\rangle/x'])(([x'']\langle v\rangle\{x := x''\})[\langle w\rangle/x'']) & \\
& = & \langle u\{x := x'\}\{x' := w\}\rangle\langle v\{x := x''\}\{x'' := w\}\rangle & \text{(induction)} \\
& = & \langle u\{x := w\}\rangle\langle v\{x := w\}\rangle & \\
& = & \langle (u\{x := w\})\,(v\{x := w\})\rangle & \\
& = & \langle (u\,v)\{x := w\}\rangle. &
\end{array}$$

□

Nous remarquons que le lemme ci-dessus est vrai également pour λ_{ca} (la preuve est identique). Nous considérons maintenant les deux stratégies tour à tour.

Nous rappelons les règles d'évaluation pour la réduction faible de termes clos en appel par nom, données à la Section I.5.2 :

$$\frac{}{\lambda x.t \Downarrow \lambda x.t} \qquad \frac{t \Downarrow \lambda x.t' \quad t'\{x := u\} \Downarrow v}{t\ u \Downarrow v}$$

Proposition II.3.39 (*Simulation de l'appel par nom dans* λ_{cf})

Soit t un λ-terme clos. Si $t \Downarrow v$ par la stratégie appel par nom, il y a alors une suite de réductions $[\![t]\!] \to_{\mathsf{cf}}^* [\![v]\!]$.

Preuve. Par induction sur la taille de la dérivation de $t \Downarrow v$. Le cas où t est une forme normale de tête faible est évident. Sinon c'est une application. En utilisant la traduction et l'hypothèse d'induction, nous obtenons :

$$[\![t\,u]\!] = [\![t]\!][\![u]\!] \to_{\mathsf{cf}}^* [\![\lambda x.t']\!][\![u]\!].$$

Il y a maintenant deux cas à considérer :

- Si $x \in \mathsf{fv}(t')$ alors grâce au Lemme II.3.38 et à l'hypothèse, nous obtenons :

$$[\![\lambda x.t']\!][\![u]\!] = (\lambda x.[x]\langle t'\rangle)\langle u\rangle \to_{\mathsf{cf}} ([x]\langle t'\rangle)[\langle u\rangle/x] \to_{\mathsf{cf}}^* [\![t'\{x := u\}]\!] \to_{\mathsf{cf}}^* [\![v]\!].$$

- Sinon :

$$[\![\lambda x.t']\!][\![u]\!] = (\lambda x.\epsilon_x.\langle t'\rangle)\langle u\rangle \to_{\mathsf{cf}} (\epsilon_x.\langle t'\rangle)[\langle u\rangle/x] \to_{\mathsf{cf}} \langle t' \rangle = [\![t'\{x := u\}]\!] \to_{\mathsf{cf}}^* [\![v]\!]$$

ce qui complète la preuve. □

Nous observons également que le même raisonnement fonctionne pour le λ_{ca}-calcul.

Nous rappelons les règles d'évaluation pour la réduction faible de termes clos en appel par valeur, données à la Section I.5.2 :

$$\frac{}{\lambda x.t \Downarrow \lambda x.t} \qquad \frac{t \Downarrow \lambda x.t' \quad u \Downarrow v' \quad t'\{x := v'\} \Downarrow v}{t\ u \Downarrow v}$$

Proposition II.3.40 (*Simulation de l'appel par valeur dans* λ_{cf})

Soit t un λ-terme clos. Si $t \Downarrow v$ par la stratégie d'appel par valeur, il existe alors une suite de réductions $[\![t]\!] \to_{\mathsf{cf}}^* [\![v]\!]$.

Preuve. La preuve suit la même structure que ci-dessus pour la simulation de l'appel par nom. En utilisant la traduction et l'hypothèse deux fois, nous obtenons :

$$[\![t\,u]\!] = [\![t]\!][\![u]\!] \to^*_{\mathsf{cf}} [\![\lambda x.t']\!][\![u]\!] \to^*_{\mathsf{cf}} [\![\lambda x.t']\!][\![v']\!].$$

Il y a deux cas à considérer, qui sont identiques à ceux donnés pour la preuve de la simulation d'appel par nom ci-dessus. □

Nous observons de nouveau que le même raisonnement fonctionne également pour le λ_{ca}-calcul.

Remarque II.3.41

Notons que même pour la réduction de termes clos en forme normale de tête faible, nous avons besoin de la composition des substitutions dans λ_{cf}, à moins que nous imposions une stratégie consistant à réduire d'abord les rédex les plus extérieurs. Par exemple, supposons que c est un terme fermé et $t = \lambda z.\epsilon_z.x$, alors $(\lambda y.(\lambda x.t)\,y)\,c \to_{\mathsf{cf}} (\lambda y.t[y/x])\,c \to_{\mathsf{cf}} t[y/x][c/y]$ où la substitution pour x ne peut pas être propagée (parce que y n'est pas un terme fermé), ainsi la seule règle applicable est une composition. De plus cette règle permet davantage de partage de calculs.

Remarque II.3.42

Bien que nous puissions réduire certains termes ouverts (par exemple $(\lambda x.x)\,y \to^*_{\mathsf{cf}} y$), nous ne pouvons pas en général simuler la réduction en forme normale de tête faible de termes ouverts, par exemple : $(\lambda x.x\,y)(\lambda x.x)$ est irréductible. Nous ne pouvons pas en général simuler la réduction en forme normale de tête, même pour des termes fermés, par exemple $\lambda x.(\lambda y.\epsilon_y.x)\,t$ est en forme normale. Cependant, le calcul est plus fort que la réduction faible standard, puisque nous pouvons réduire sous des abstractions :

$$\lambda x.(\lambda y.y)\,x \to_{\mathsf{cf}} \lambda x.y[x/y] \to_{\mathsf{cf}} \lambda x.x.$$

II.3.5 Arguments et/ou fonctions fermés : λ_{cl}, λ_{caf}

Nous avons défini dans les sections précédentes deux calculs différents qui bénéficient de bonnes propriétés de confluence et de préservation de la normalisation forte, tous les deux basés sur une forme restreinte de la règle *Beta*. Comme ces restrictions sont indépendantes, la question de savoir si des combinaisons de ces calculs sont possibles et intéressantes se pose naturellement.

Si nous prenons l'intersection des deux calculs, c'est-à-dire si nous limitons la règle *Beta* au cas où la fonction et l'argument sont fermés nous obtenons une version avec substitutions explicites de la réduction de la Logique Combinatoire, comme décrit dans [How70, Hin77, ÇH98]. Nous appelons ce système λ_{cl} . Ainsi, une *Beta*-réduction est autorisée si et seulement si le rédex est clos. Ce type de réduction a de très bonnes propriétés : en particulier, si deux rédex ne se superposent pas, leurs résidus ne se superposent pas non plus. C'est pourquoi des λ-calculs similaires sont souvent utilisés, par exemple dans [HO93, BLM05].

Cependant, plutôt que de restreindre encore plus l'ensemble des réductions possibles, on voudrait plutôt prendre l'union des deux calculs, c'est-à-dire autoriser la règle *Beta* dès que, soit la fonction, soit l'argument, est fermé, tout en gardant toutes les σ-règles de λ_{cf}. Nous appellerons ce nouveau calcul λ_{caf} . Mais maintenant la confluence n'est plus gratuite. Rappelons-nous le diagramme du problème (possible) de confluence de Section II.3.3 :

$$\begin{array}{ccccc} ((\lambda x.t)\,u)[v/y] & \longrightarrow & ((\lambda x.t)[v/y])\,u & \longrightarrow & (\lambda x.t[v/y])\,u \\ \downarrow & & & & \downarrow \\ t[u/x][v/y] & & \overset{?}{\longleftrightarrow} & & t[v/y][u/x] \end{array}$$

où $x, y \in \mathsf{fv}(t)$ et $\mathsf{fv}(v) = \varnothing$.

La branche gauche est une application de la règle *Beta*. Cependant, $y \in \mathsf{fv}(t)$ (et si ce n'est pas le cas, alors il n'y a aucun problème de confluence), ainsi ce ne peut pas être la règle *Beta* de λ_{cf}. Par conséquent c'est la règle *Beta* avec argument fermé, et u est fermé. Dans la branche droite, nous appliquons une règle *Lam* puisque v est clos aussi. Par conséquent nous sommes exactement dans le même cas que dans le Lemme II.3.17 et nous avons $t[u/x][v/y] =_\sigma t[v/y][u/x]$ (la preuve est semblable au Lemme II.3.17).

Puisque λ_{caf} diffère de λ_{cf} seulement par la règle *Beta*, la discussion précédente justifie la propriété suivante.

Proposition II.3.43

Le calcul λ_{caf} est localement confluent.

Preuve. Semblable à la Proposition II.3.33, en utilisant l'argument précédent pour le cas 1, les autres cas ne changent pas. □

Les autres propriétés prouvées pour λ_{ca} et λ_{cf} sont encore valides pour λ_{caf}, les preuves peuvent être facilement adaptées. Nous comparons maintenant les quatre systèmes :

Proposition II.3.44

Pour des termes clos,
- λ_{ca}, λ_{cf} permettent davantage de partage que les réductions faibles standard (appel par nom et appel par valeur).
- λ_{ca}, λ_{cf} permettent davantage de partage que λ_{cl}.
- λ_{caf} permet davantage de partage que λ_{ca} et λ_{cf}.

Preuve. Le premier point est une conséquence triviale des propositions de la Section II.3.4, et les deux autres points proviennent immédiatement de l'inspection des systèmes. □

II.3.6 Canonicité

Jusqu'ici nous avons décrit un certain nombre de calculs sans α-conversion obtenus en autorisant certaines règles seulement quand certains termes sont clos. Ces choix peuvent paraître arbitraires, mais ils ne le sont pas : nous allons voir dans cette section, de façon un petit peu informelle, qu'il n'y a en effet pas beaucoup d'autres choix satisfaisants.

Adoptant la syntaxe de λ_{c}, nous spécifions nos objectifs *a posteriori* :

- Nous voulons une règle *Beta* $(\lambda x.t)\,u \to t[u/x]$ avec des restrictions éventuelles et des règles pour propager les substitutions vers les occurrences des variables libres correspondantes.
- Nous ne voulons pas avoir à nous préoccuper de l'α-conversion. Ainsi, nous devons restreindre la règle *Lam* mais également la règle $Copy_1$. La règle *Lam* la plus générale sans α-conversion est la suivante :

  $$(\lambda y.t)[v/x] \to \lambda y.t[v/x] \quad \text{si } x \neq y \text{ et } y \notin \mathsf{fv}(v)$$

  mais alors la réduction est autorisée ou non en fonction du nom d'une variable liée. En particulier, deux termes α-équivalents n'ont pas forcément les mêmes rédex, ce qui pourrait créer des problèmes de confluence. Une approche plus sûre est de généraliser la condition en $\mathsf{fv}(v) = \varnothing$. Pour l'autre règle, l'$\alpha$-conversion est nécessaire dès que la substitution est ouverte. Ainsi, nous imposons :

  $$(\delta_x^{y,z}.t)[v/x] \to t[v/y][v/z] \quad \text{si } \mathsf{fv}(v) = \varnothing.$$

- Nous voulons que notre calcul permette la réduction (au moins) jusqu'en forme normale de tête faible, qui est en effet ce qui est nécessaire dans les implantations de langages fonctionnels, et c'est la condition minimale pour qu'un λ-calcul ait une quelconque utilité en pratique.
- Nous voulons préserver la normalisation forte, ce qui implique également que nous voulons que les σ-règles soient fortement normalisantes. La raison en est que nous adoptons une perspective d'implantation et nous voulons être libres de choisir n'importe quelle stratégie pour réduire les λ-termes fortement normalisables. Ce n'est donc pas strictement indispensable, mais c'est tout de même souhaitable, puisque cela facilite le raisonnement.
- Nous demandons finalement à notre calcul d'être confluent. Ceci semble de nouveau être une condition minimale pour une implantation déterministe du λ-calcul et pour être libre de choisir n'importe quelle stratégie. Cette dernière condition amène quelques restrictions dans les règles. En effet, nous avons cette paire critique $Beta/App_1$, que nous écrivons pour la dernière fois :

  $$\begin{array}{ccccc} ((\lambda x.t)\,u)[v/y] & \longrightarrow & ((\lambda x.t)[v/y])\,u & \longrightarrow & (\lambda x.t[v/y])\,u \\ \downarrow & & & & \downarrow \\ t[u/x][v/y] & & \overset{?}{\longleftrightarrow} & & t[v/y][u/x] \end{array}$$

  où $x, y \in \mathsf{fv}(t)$.

Il y a trois manières de résoudre ce problème potentiel :

1. Éliminer la paire critique en interdisant une des réductions du diagramme.
   - Couper la branche droite est une possibilité qui a été étudiée par Muñoz [Muñ96]. Son idée est d'interdire la distribution de la substitution dans une pile d'applications commençant par une fonction. Cependant, ceci ne peut pas être une règle du premier ordre avec la syntaxe standard, et donc chaque application est annotée avec des marques

spéciales pour distinguer les piles d'applications commençant par une variable ou une fonction. Le calcul résultant est entièrement confluent et préserve la normalisation forte, mais ne correspond pas complètement à nos spécifications, puisque sa syntaxe est plus complexe (et de fait, utilise des indices de de Bruijn).

– Maintenant si nous voulons couper la branche gauche de la paire critique, ce qui est laissé comme problème ouvert par Muñoz, nous devons exactement restreindre la règle *Beta* au cas où aucune substitution ne peut être appliquée à la fonction, donc au cas où la fonction est fermée : ainsi nous sommes au cœur de λ_{cf}. Mais nous pouvons encore autoriser certaines règles de propagation quand la substitution est ouverte, ainsi nous obtenons un ensemble de réductions légèrement plus grand que λ_{cf}. Cependant certaines règles de propagation doivent être limitées pour éviter l'α-conversion (*Lam*, $Copy_1$), ou pour assurer la normalisation des σ-règles (*Comp*, le cas échéant), de sorte que dans la plupart des cas une substitution produite ne pourra pas atteindre les variables en traversant tout le terme, puisqu'elle devra passer par une des règles restreintes. Ainsi les choix faits dans λ_{cf} de clore plus que nécessaire ne réduisent pas vraiment le nombre de stratégies efficaces disponibles dans le calcul, et conduisent à un système de récriture beaucoup plus homogène.

2. Demander que la dernière ligne du diagramme soit dérivable. Afin de dériver

$$t[u/x][v/y] =_\sigma t[v/y][u/x]$$

(avec des restrictions) dans le système, sans ajouter une règle *ad hoc*, nous devons propager les substitutions jusqu'à ce qu'elles ne soient plus imbriquées. Le problème est qu'une partie des règles de propagation demande que la substitution soit fermée (voir ci-dessus), de sorte que pour s'assurer que nous puissions réellement propager les substitutions, il faut qu'elles soient toujours fermées. C'est exactement la même chose que de dire que nous pouvons seulement produire des substitutions fermées, autrement dit nous sommes exactement dans le cas de λ_{ca}.

3. Ajouter une règle d'échange des substitutions, validant la dernière ligne du diagramme. Naturellement, si nous ajoutons grossièrement une règle :

$$t[u/x][v/y] \to t[v/y][u/x] \quad \text{pour tous } u,\, v$$

nous perdons la préservation de la normalisation forte, ce qui n'est pas ce que nous voulons. Mais nous pourrions par exemple avoir un système avec les règles de propagation closes, et une règle de permutation "close" telle que :

$$t[u/x][v/y] \to t[v/y][u/x] \quad \text{si } x, y \in \mathsf{fv}(t), \mathsf{fv}(u) \neq \varnothing, \mathsf{fv}(v) = \varnothing.$$

Alors nous pourrions permettre une règle *Beta* sans restrictions. Cette solution semble préserver la normalisation forte, résoudre le problème de confluence, et est tout à fait sensée : elle permet effectivement aux substitutions fermées de se propager davantage (à l'intérieur de termes avec des substitutions ouvertes). Cependant, ceci introduit aussi une quantité de nouvelles stratégies inutiles possibles. Au lieu de produire deux substitutions, la plus extérieure étant fermée, et de les permuter par la suite, nous préférons produire la

substitution fermée d'abord, la propager jusqu'au λ correspondant, puis créer la seconde de sorte qu'elles soient déjà dans le bon ordre. C'est moralement équivalent mais plus simple. Cependant, il peut y avoir des possibilités intéressantes alternatives pour une règle de permutation, qui ne sont pas étudiées ici.

Pour conclure, les calculs λ_{ca} et λ_{cf} apparaissent finalement de façon tout à fait naturelle comme calculs sans α-conversion avec les spécifications données.

II.4 Extensions de λ_{c}

Dans cette section nous étendons λ_{c} pour obtenir un langage de programmation fonctionnel minimaliste. Nous commençons par définir une notion de *programme*, pour laquelle nous introduisons des types.

II.4.1 Système de types pour λ_{c}

Nous considérons une variante typée de λ_{c}, et étudions plusieurs des propriétés habituelles associées à de tels calculs. Le système de type que nous donnons est commun à λ_{ca} et à λ_{cf}, et pour les propriétés nous nous concentrerons sur λ_{cf}. Cependant, tous les résultats de cette section sont également valables pour λ_{ca} (et d'ailleurs ils sont généralement plus faciles à obtenir pour λ_{ca}).

Définition II.4.1

Dans la Figure II.1, nous donnons les règles de typage, où $\Gamma \vdash t : A$ indique que le λ_{c}-terme t a le type A dans le contexte Γ, et $\top$ est une constante de type arbitraire (par exemple nat ou bool dans la Section II.4.2). Les types sont donc ici définis par la grammaire :

$$A, B ::= \top \mid A \to B.$$

Le contexte est traité comme une liste ordonnée $x_1 : A_1, \ldots, x_n : A_n$, et nous donnons une règle structurelle explicite qui permet à des éléments de la liste d'être permutés. Il est à noter que c'est la seule règle structurelle, puisque la copie et l'effacement d'éléments de la liste sont faits explicitement par les règles de typage pour *Copy* et *Erase* respectivement.

Dans ce système, plusieurs des contraintes de la syntaxe sont maintenant capturées par les règles. De plus, les contraintes sur les règles *Const* et *Var* forcent l'environnement à contenir seulement les variables libres de t, et aucune autre. Les règles de typage sont directement inspirées par le système de type de Curry pour le λ-calcul (voir la Section I.3.5 ou par exemple [Bar92]), dont les jugements sont notés $\Gamma \vdash_\lambda t : A$. Les règles *Copy* et *Erase* seraient normalement classifiées en tant que règles structurelles. Le résultat suivant indique comment les systèmes sont reliés.

Lemme II.4.2

1. Soit t un λ-terme tel que $\mathsf{fv}(t) = \{x_1, \ldots, x_n\}$, et $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, alors :

$$\Gamma \vdash_\lambda t : A \iff \Gamma \vdash [\![t]\!] : A.$$

$$\frac{\Gamma, x : A, y : B, \Delta \vdash t : C}{\Gamma, y : B, x : A, \Delta \vdash t : C}\,(\text{Exchange})$$

$$\frac{}{\vdash \star : \top}\,(\text{Const}) \qquad \frac{}{x : A \vdash x : A}\,(\text{Var})$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\,(\text{Abs}) \qquad \frac{\Gamma \vdash t : A \to B \qquad \Delta \vdash u : A}{\Gamma, \Delta \vdash t\,u : B}\,(\text{App})$$

$$\frac{\Gamma \vdash t : B}{\Gamma, x : A \vdash \epsilon_x.t : B}\,(\text{Erase}) \qquad \frac{\Gamma, x : A, y : A \vdash t : B}{\Gamma, z : A \vdash \delta_z^{x,y}.t : B}\,(\text{Copy})$$

$$\frac{\Gamma, x : A \vdash t : B \qquad \Delta \vdash v : A}{\Gamma, \Delta \vdash t[v/x] : B}\,(\text{Sub})$$

FIG. II.1 – Système de type pour λ_{c}

2. Soit t un λ_{c}-terme valide tel que $\mathsf{fv}(t) = \{x_1, \ldots, x_n\}$, et $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, alors :
$$\Gamma \vdash t : A \iff \Gamma \vdash_{\lambda} t^* : A.$$

Preuve. Par induction facile sur la dérivation de typage. □

Nous donnons maintenant quelques résultats standard sur ce calcul.

PROPOSITION II.4.3 (*Réduction du sujet*)
Si $\Gamma \vdash t : A$ et $t \to_{\mathsf{cf}} u$ alors $\Gamma \vdash u : A$.

Preuve. Nous prouvons que chaque règle préserve le type dans la Figure II.2. Le cas pour App_2 est similaire à celui pour App_1. □

Nous étudions maintenant le problème de la terminaison des termes typables. Puisque les λ-termes typables terminent, la terminaison des λ_{cf}-termes typables est une conséquence directe de la Proposition II.3.31 et du Lemme II.4.2.

PROPOSITION II.4.4 (*Terminaison*)
Si $\Gamma \vdash t : A$ dans λ_{cf}, alors t est fortement normalisable.

DÉFINITION II.4.5 (*Programmes*)
Un *programme* est un terme clos de type $\top$.

Nous pouvons montrer que, dans nos calculs, les programmes peuvent être réduits en valeurs, qui sont des termes purs. Autrement dit, dans les termes clos de type de base, il y a suffisamment de substitutions fermées, de sorte qu'elles puissent toutes être menées à bien. Par exemple, dans le contexte de la Section II.4.2, cette proposition signifie qu'un programme de type nat s'évalue toujours en un entier.

$$\dfrac{\dfrac{x : A \vdash t : B}{\vdash \lambda x.t : A \to B} \quad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x.t)\, u : B} \to \dfrac{x : A \vdash t : B \quad \Gamma \vdash u : A}{\Gamma \vdash t[u/x] : B}$$

$$\dfrac{\overline{x : A \vdash x : A} \quad \Gamma \vdash v : A}{\Gamma \vdash x[v/x] : A} \to \quad \overline{\Gamma \vdash v : A}$$

$$\dfrac{\dfrac{\Gamma, x : C \vdash t : A \to B \quad \Delta \vdash u : A}{\Gamma, \Delta, x : C \vdash t\, u : B} \quad \Theta \vdash v : C}{\Gamma, \Delta, \Theta \vdash (t\, u)[v/x] : B} \to \dfrac{\dfrac{\Gamma, x : C \vdash t : A \to B \quad \Theta \vdash v : C}{\Gamma, \Theta \vdash t[v/x] : A \to B} \quad \Delta \vdash u : A}{\Gamma, \Delta, \Theta \vdash (t[v/x])\, u : B}$$

$$\dfrac{\dfrac{\Gamma, x : C, y : A \vdash t : B}{\Gamma, x : C \vdash \lambda y.t : A \to B} \quad \vdash v : C}{\Gamma \vdash (\lambda y.t)[v/x] : A \to B} \to \dfrac{\dfrac{\Gamma, y : A, x : C \vdash t : B \quad \vdash v : C}{\Gamma, y : A \vdash t[v/x] : B}}{\Gamma \vdash \lambda y.t[v/x] : A \to B}$$

$$\dfrac{\dfrac{\Gamma, y : A, z : A \vdash t : B}{\Gamma, x : A \vdash \delta_x^{y,z}.t : B} \quad \vdash v : A}{\Gamma \vdash (\delta_x^{y,z}.t)[v/x] : B} \to \dfrac{\dfrac{\Gamma, z : A, y : A \vdash t : B \quad \vdash v : A}{\Gamma, z : A \vdash t[v/y] : B} \quad \vdash v : A}{\Gamma \vdash (t[v/y])[v/z] : B}$$

$$\dfrac{\dfrac{\Gamma, x : A, y : C, z : C \vdash t : B}{\Gamma, x' : C, x : A \vdash \delta_{x'}^{y,z}.t : B} \quad \Delta \vdash v : A}{\Gamma, \Delta, x' : C \vdash (\delta_{x'}^{y,z}.t)[v/x] : B} \to \dfrac{\dfrac{\Gamma, x : A, y : C, z : C \vdash t : B \quad \Delta \vdash v : A}{\Gamma, \Delta, y : C, z : C \vdash t[v/x] : B}}{\Gamma, \Delta, x' : C \vdash \delta_{x'}^{y,z}.t[v/x] : B}$$

$$\dfrac{\dfrac{\Gamma \vdash t : B}{\Gamma, x : A \vdash \epsilon_x.t : B} \quad \vdash v : A}{\Gamma \vdash (\epsilon_x.t)[v/x] : B} \to \quad \overline{\Gamma \vdash t : B}$$

$$\dfrac{\dfrac{\Gamma, x : A \vdash t : B}{\Gamma, x' : C, x : A \vdash \epsilon_{x'}.t : B} \quad \Delta \vdash v : A}{\Gamma, \Delta, x' : C \vdash (\epsilon_{x'}.t)[v/x] : B} \to \dfrac{\dfrac{\Gamma, x : A \vdash t : B \quad \Delta \vdash v : A}{\Gamma, \Delta \vdash t[v/x] : B}}{\Gamma, \Delta, x' : C \vdash \epsilon_{x'}.t[v/x] : B}$$

$$\dfrac{\dfrac{\Gamma, y : B \vdash t : C \;\; \Delta, x : A \vdash w : B}{\Gamma, \Delta, x : A \vdash t[w/y] : C} \quad \Theta \vdash v : A}{\Gamma, \Delta, \Theta \vdash t[w/y][v/x] : C} \to \dfrac{\Gamma, y : B \vdash t : C \qquad \dfrac{\Delta, x : A \vdash w : B \;\; \Theta \vdash v : A}{\Delta, \Theta \vdash w[v/x] : B}}{\Gamma, \Delta, \Theta \vdash t[w[v/x]/y] : C}$$

FIG. II.2 – Réduction du sujet

PROPOSITION II.4.6 (*Adéquation*)

Si t est un programme, alors $t \to_{\mathsf{cf}}^{*} \star$.

Preuve. Par réduction du sujet (Proposition II.4.3), le type du terme est préservé par la réduction. Supposons pour obtenir une contradiction que le programme t est en forme normale, et n'est pas $\star$. Puisque t est fermé, ce ne peut pas être une variable, un effacement ou une copie. Puisqu'il est de type $\top$, ce ne peut pas être une abstraction non plus. Si $t = u[v/x]$ alors v est fermé (puisque t est fermé), et donc une des règles pour la substitution devrait s'appliquer par le Lemme II.3.27. Par conséquent t est une application. Disons $t = u_1\, u_2 \dots u_n$, $n \geq 2$, tels que u_1 ne soit pas une application. Puisque t est fermé, $u_1, \dots, u_n$ le sont aussi. Par conséquent, u_1 n'est pas une variable, une copie ou un effacement. Puisque t est une forme normale, u_1 ne peut pas être une abstraction non plus (la règle *Beta* s'appliquerait). Ainsi u_1 est un terme de la forme $s[s'/x]$ où s' est fermé et x est la seule variable libre de s. Mais alors u_1 n'est pas une forme normale (Lemme II.3.27), ce qui contredit l'hypothèse. □

II.4.2 λ_{c} COMME LANGAGE DE PROGRAMMATION

Le système de type et la notion de programme ci-dessus peuvent être considérés comme la simplification d'un langage de programmation fonctionnel minimaliste, tel que PCF [Plo77], où nous avons juste une constante, et aucune fonction arithmétique ni récursion. Il n'y a aucune difficulté à étendre ce calcul au langage PCF en entier, et même au-delà, comme nous allons maintenant l'esquisser.

La syntaxe des termes et la sémantique opérationnelle pour l'évaluation de PCF sont rappelées dans la Section I.3.6. La Table II.5 donne les extensions de λ_{c} de sorte que nous puissions capturer la même syntaxe que PCF. Notons que nous avons inclus le type et les contraintes de variables, à partir desquels on peut facilement écrire les règles de typage correspondantes. Nous appelons $\lambda_{\mathsf{cf}}^{\mathsf{pcf}}$ le langage résultant. Il est à noter que la représentation des conditionnelles impose que les deux branches aient exactement les mêmes variables.

TAB. II.5 – Extensions de syntaxe

Terme	Contrainte de type	Contrainte de variables	Variables libres
$n : \mathsf{nat}$	–	–	$\varnothing$
$\mathsf{tt}, \mathsf{ff} : \mathsf{bool}$	–	–	$\varnothing$
$\mathsf{suc}(t) : \mathsf{nat}$	$t : \mathsf{nat}$	–	$\mathsf{fv}(t)$
$\mathsf{pred}(t) : \mathsf{nat}$	$t : \mathsf{nat}$	–	$\mathsf{fv}(t)$
$\mathsf{zer}(t) : \mathsf{bool}$	$t : \mathsf{nat}$	–	$\mathsf{fv}(t)$
$\mathsf{if}(b, t, u) : A$	$b : \mathsf{bool}, t : A, u : A$	$\mathsf{fv}(t) = \mathsf{fv}(u), \mathsf{fv}(t) \cap \mathsf{fv}(b) = \varnothing$	$\mathsf{fv}(b) \cup \mathsf{fv}(t)$
$\mathsf{fix}(t) : A$	$t : A \to A$	–	$\mathsf{fv}(t)$

Il est facile d'étendre la compilation (Définition II.3.2) pour capturer PCF. Nous montrons juste la compilation additionnelle pour les constantes.

DÉFINITION II.4.7 (*Compilation de PCF*)

Soit t un terme de PCF, $\mathsf{fv}(t) = \{x_1, \ldots, x_n\}$, $n \geq 0$. Sa compilation dans $\lambda_{\mathsf{cf}}^{\mathsf{pcf}}$, notée $[\![t]\!]$, est définie par : $[x_1] \ldots [x_n]\langle t \rangle$ avec $\langle \cdot \rangle$ défini comme l'extension de la fonction donnée dans la Définition II.3.2 où $\star \in \{\mathsf{tt}, \mathsf{ff}\}$ ou $\star = n$ et :

$$\begin{array}{lcll} \langle \mathsf{f}(t) \rangle & = & \mathsf{f}(\langle t \rangle) & (\mathsf{f} \in \{\mathsf{suc}, \mathsf{pred}, \mathsf{zer}, \mathsf{fix}\}) \\ \langle \mathsf{if}(b,t,u) \rangle & = & \mathsf{if}(\langle b \rangle, \epsilon_{y_1} \ldots \epsilon_{y_p}.\langle t \rangle, \epsilon_{z_1} \ldots \epsilon_{z_q}.\langle u \rangle) & (\mathsf{fv}(u) \setminus \mathsf{fv}(t) = \{y_1, \ldots, y_p\}, \\ & & & \mathsf{fv}(t) \setminus \mathsf{fv}(u) = \{z_1, \ldots, z_q\}, p \geq 0, q \geq 0) \end{array}$$

Nous ajoutons également les cas suivants à la définition de $[\cdot]\cdot$:

$$\begin{array}{lcll} [x]\mathsf{f}(t) & = & \mathsf{f}([x]t) & (\mathsf{f} \in \{\mathsf{suc}, \mathsf{pred}, \mathsf{zer}, \mathsf{fix}\}) \\ [x]\mathsf{if}(b,t,u) & = & \mathsf{if}([x]b, t, u) & (x \in \mathsf{fv}(b), x \notin \mathsf{fv}(t\,u)) \\ & = & \mathsf{if}(b, [x]t, [x]u) & (x \notin \mathsf{fv}(b), x \in \mathsf{fv}(t\,u)) \\ & = & \delta_x^{y,z}.\mathsf{if}([y](b\{x := y\}), [z](t\{x := z\}), [z](u\{x := z\})) & (x \in \mathsf{fv}(b), \mathsf{fv}(t\,u)) \\ [x]\epsilon_x.t & = & \epsilon_x.t & \end{array}$$

EXEMPLE II.4.8

Nous donnons ci-dessous deux exemples de termes compilés de PCF, spécifiquement pour mettre en évidence les détails pour la conditionnelle.

$$\begin{array}{lcl} [\![\lambda x.\mathsf{if}(x,x,x)]\!] & = & \lambda x.\delta_x^{y,z}.\mathsf{if}(y,z,z) \\ [\![\lambda x.\lambda y.\lambda z.\mathsf{if}(x,y,z)]\!] & = & \lambda x.\lambda y.\lambda z.\mathsf{if}(x, \epsilon_z.y, \epsilon_y.z). \end{array}$$

DÉFINITION II.4.9 ($\lambda_{\mathsf{cf}}^{\mathsf{pcf}}$-*réduction*)

L'ensemble des règles est identique à λ_{cf} avec les additions données dans la Table II.6, où $\mathsf{f} \in \{\mathsf{suc}, \mathsf{pred}, \mathsf{zer}, \mathsf{fix}\}$ dans Sub_1.

Nous récapitulons les propriétés principales de ce calcul. La plupart sont des prolongements faciles des résultats pour λ_{cf} typé. Rappelons que les programmes de PCF sont les termes fermés de type nat ou bool.

PROPOSITION II.4.10 (*Propriétés de* $\lambda_{\mathsf{cf}}^{\mathsf{pcf}}$)

1. Réduction du sujet : si $\Gamma \vdash t : A$ et $t \to_{\mathsf{cf}} u$, alors $\Gamma \vdash u : A$.
2. Adéquation : si $t \to^* v$ alors $[\![t]\!] \to_{\mathsf{cf}}^* [\![v]\!]$, et par conséquent si t est un programme :
   - soit $t \to_{\mathsf{cf}}^* \mathsf{tt}$, si $t : \mathsf{bool}$;
   - soit $t \to_{\mathsf{cf}}^* \mathsf{ff}$, si $t : \mathsf{bool}$;
   - soit $t \to_{\mathsf{cf}}^* n$, si $t : \mathsf{nat}$;
   - soit t diverge.

Preuve. La première partie est une extension facile du Lemme II.4.3 pour le calcul typé. Pour la deuxième partie, nous devons d'abord établir une propriété sur la substitution dans PCF,

TAB. II.6 – $\lambda^{\mathsf{pcf}}_{\mathsf{cf}}$-réduction

Nom	Réduction			Condition
Succ	$\mathsf{suc}(n)$	$\to_{\mathsf{cf}}$	$n+1$	–
$Pred_1$	$\mathsf{pred}(0)$	$\to_{\mathsf{cf}}$	0	–
$Pred_2$	$\mathsf{pred}(n+1)$	$\to_{\mathsf{cf}}$	n	–
$Iszero_1$	$\mathsf{zer}(0)$	$\to_{\mathsf{cf}}$	tt	–
$Iszero_2$	$\mathsf{zer}(n+1)$	$\to_{\mathsf{cf}}$	ff	–
$Cond_1$	$\mathsf{if}(\mathsf{tt}, t, u)$	$\to_{\mathsf{cf}}$	t	–
$Cond_2$	$\mathsf{if}(\mathsf{ff}, t, u)$	$\to_{\mathsf{cf}}$	u	–
Rec	$\mathsf{fix}(t)$	$\to_{\mathsf{cf}}$	$t\,\mathsf{fix}(t)$	$\mathsf{fv}(t) = \varnothing$
Sub_1	$\mathsf{f}(t)[v/x]$	$\to_{\mathsf{cf}}$	$\mathsf{f}(t[v/x])$	–
Sub_2	$\mathsf{if}(b,t,u)[v/x]$	$\to_{\mathsf{cf}}$	$\mathsf{if}(b[v/x],t,u)$	$x \in \mathsf{fv}(b)$
Sub_3	$\mathsf{if}(b,t,u)[v/x]$	$\to_{\mathsf{cf}}$	$\mathsf{if}(b,t[v/x],u[v/x])$	$x \in \mathsf{fv}(t), \mathsf{fv}(v) = \varnothing$

analogue au Lemme II.3.38 : soient t et w des termes de PCF tels que $\mathsf{fv}(t) = \{x_1, \ldots, x_n\}$, $n \geq 1$, et $\mathsf{fv}(w) = \varnothing$, alors $([x_1]\ldots[x_n]\langle t\rangle)[\langle w\rangle/x_1] \to^*_{\mathsf{cf}} [x_2]\ldots[x_n]\langle t\{x_1 := w\}\rangle$. La preuve de ce fait suit le même raisonnement que celle du Lemme II.3.38. En utilisant ceci, il est alors immédiat d'établir la simulation des réduction de PCF : si $t \to^* v$ alors il existe une suite de réductions $[\![t]\!] \to^*_{\mathsf{cf}} [\![v]\!]$. □

II.5 IMPLANTATION : MACHINES ABSTRAITES

Le but de cette section est de donner davantage de légitimité à notre travail, théoriquement et pratiquement. Nous utilisons les calculs précédents pour définir des stratégies, qui sont implantées et comparées expérimentalement. Ici, les Propositions II.3.33 (confluence locale) et II.2.2 (ordre des substitutions) peuvent être mises ensemble pour nous aider à comprendre la meilleure stratégie pour propager les substitutions, elles indiquent quels choix donnent les chemins de réduction les plus courts.

II.5.1 STRATÉGIES CLOSES

Les calculs définis ci-dessus sont plus restrictifs que le λ-calcul, par conséquent toute stratégie ne peut pas être définie dans notre formalisme. Cependant, il reste encore beaucoup de possibilités. Nous justifierons informellement certains des choix que nous avons faits avec des considérations d'efficacité, puis nous exhiberons deux stratégies particulières qui se comparent expérimentalement bien aux stratégies habituelles. Ce seront des stratégies pour réduire des termes fermés (programmes) en forme normale de tête faible (valeur). Avant de définir formellement ces stratégies, nous donnons quelques intuitions pour expliquer comment elles fonctionnent et pourquoi nous avons fait ces choix.

– Tout d'abord, nous voulons tirer profit de notre capacité à réduire sous des abstractions, mais pas au *top level*, puisque nous voulons nous arrêter sur une forme normale de tête

faible. En particulier, il est seulement utile d'effectuer ces réductions supplémentaires sur un terme qui sera copié, afin de partager ces réductions. Par conséquent notre définition formelle intercalera une stratégie faible avec une plus forte, appelée seulement avant une application de la règle $Copy_1$.

- Avant de copier, plus nous réduisons le terme à copier, plus le travail est partagé *a priori*. Par conséquent, nous voulons employer notre calcul le plus général, à savoir λ_{caf}. Nous donnerons cependant aussi une stratégie basée sur λ_{cf} (λ_{ca} seul est trop restrictif).
- Quand nous sommes dans une situation où nous pouvons appliquer une règle *Comp*, la Proposition II.2.2 montre qu'il vaut mieux le faire plutôt que de réduire la substitution la plus interne d'abord. Notre stratégie accordera ainsi une priorité plus élevée à *Comp*.
- Une dernière remarque est que les règles de λ_{caf} forcent la substitution à être fermée dans la règle $Copy_1$, autrement dit nous ne copions jamais un terme avec des variables libres. C'est un des aspects principaux de cette stratégie en ce qui concerne l'efficacité.

$$\frac{v \Downarrow_w w}{x[v/x] \Downarrow_w w}\ (\text{Var}) \qquad \frac{t \Downarrow_w \lambda x.r \quad r[u/x] \Downarrow_w v \quad \mathsf{fv}(t)=\varnothing \vee \mathsf{fv}(u)=\varnothing}{(t\,u) \Downarrow_w v}\ (\text{Beta})$$

$$\frac{(\lambda y.t[u/x]) \Downarrow_w v \quad \mathsf{fv}(u)=\varnothing}{(\lambda y.t)[u/x] \Downarrow_w v}\ (\text{Lam}) \qquad \frac{t \Downarrow_w v \quad v \neq \lambda x.r \vee (\mathsf{fv}(t)\neq\varnothing \wedge \mathsf{fv}(u)\neq\varnothing)}{(t\,u) \Downarrow_w (v\,u)}\ (\text{Arg})$$

$$\frac{(t[v/x])\,u \Downarrow_w w \quad x \in \mathsf{fv}(t)}{(t\,u)[v/x] \Downarrow_w w}\ (\text{App1}) \qquad \frac{t[u[v/x]/y] \Downarrow_w w \quad x \in \mathsf{fv}(u)}{t[u/y][v/x] \Downarrow_w w}\ (\text{Comp})$$

$$\frac{t\,(u[v/x]) \Downarrow_w w \quad x \in \mathsf{fv}(u)}{(t\,u)[v/x] \Downarrow_w w}\ (\text{App2}) \qquad \frac{t \Downarrow_w u \quad u[v/x] \Downarrow_w w \quad \mathsf{fv}(v) \neq \varnothing}{t[v/x] \Downarrow_w w}\ (\text{Subst})$$

$$\frac{t \Downarrow_w w \quad \mathsf{fv}(v) = \varnothing}{(\epsilon_x.t)[v/x] \Downarrow_w w}\ (\text{Erase1}) \qquad \frac{v \Downarrow_f v' \quad t[v'/y][v'/z] \Downarrow_w w \quad \mathsf{fv}(v) = \varnothing}{(\delta^{y,z}_x.t)[v/x] \Downarrow_w w}\ (\text{Copy1})$$

$$\frac{\epsilon_{x'}.(t[v/x]) \Downarrow_w w \quad x \neq x'}{(\epsilon_{x'}.t)[v/x] \Downarrow_w w}\ (\text{Erase2}) \qquad \frac{\delta^{y,z}_{x'}.(t[v/x]) \Downarrow_w w \quad x \neq x'}{(\delta^{y,z}_{x'}.t)[v/x] \Downarrow_w w}\ (\text{Copy2})$$

$$\frac{t \not\Downarrow_w \text{ par toute autre règle}}{t \Downarrow_w t}\ (\text{Axiom})$$

FIG. II.3 – Sémantique opérationnelle de la stratégie close

Dans la Figure II.3, nous donnons la sémantique opérationnelle à grands pas de la stratégie fermée $\Downarrow_w$, en utilisant une relation auxiliaire (plus forte) $\Downarrow_f$. On remarque que la règle ($Copy_1$) appelle la réduction plus forte $\Downarrow_f$, qui est définie exactement de la même manière, en remplaçant juste la règle ($Axiom$) par :

$$\frac{t \Downarrow_f v}{\lambda x.t \Downarrow_f \lambda x.v} \qquad \frac{t \Downarrow_f v}{\epsilon_x.t \Downarrow_f \epsilon_x.v} \qquad \frac{t \Downarrow_f v}{\delta^{y,z}_x.t \Downarrow_f \delta^{y,z}_x.v} \qquad \frac{t \not\Downarrow_f \text{ par une autre règle}}{t \Downarrow_f t}$$

La relation $\Downarrow_w$ est en effet ce que nous voulons : nous réduisons en forme normale de tête faible, mais nous réduisons sous les abstractions dans les sous-termes qui seront copiés. La stratégie ci-dessus appartient à λ_{caf}. Pour définir une stratégie fermée dans λ_{cf}, ce qui a quelques avantages pour l'implantation (voir Chapitre IV), il suffit de restreindre la règle *(Beta)* aux fonctions fermées.

II.5.2 Efficacité

Nos calculs vivent dans le même monde que les λ-calculs en appel par nom et en appel par valeur, au sens où ce sont des calculs de termes (par opposition à des calculs de graphes, par exemple). Chaque stratégie dans ce monde doit décider plus ou moins arbitrairement ce qu'elle doit faire avant une β-réduction (et également dans notre cas, avant la copie) : l'argument doit-il être réduit d'abord et à quel point ? L'appel par nom ne fait aucune réduction, l'appel par valeur réduit jusqu'à ce que l'argument atteigne une forme normale de tête faible, et notre stratégie close effectue davantage de réductions. Ces choix ont des conséquences fortes sur la terminaison et l'efficacité sur certaines classes de λ-termes.

Comme avec l'appel par valeur comparé à l'appel par nom, nous choisissons d'effectuer plus de travail tant que nous pouvons encore le partager. La conséquence pour l'appel par valeur est qu'il est plus efficace que l'appel par nom sur une classe de λ-termes où l'argument peut être réduit et est utilisé plusieurs fois dans le corps de la fonction, mais qu'elle termine moins souvent. La même chose s'applique pour la réduction close (par rapport à l'appel par valeur, par exemple).

Notons que ce n'est cependant pas une conséquence triviale du fait que nous effectuons davantage de travail avant de copier (ou de façon à peu près équivalente avant une β-réduction au sens implicite). Par exemple, considérons la stratégie suivante : prenons l'appel par valeur, sauf qu'avant une β-réduction, l'argument est réduit en forme normale complète. Alors cette stratégie est (expérimentalement) moins efficace que la nôtre (voir la Section IV.9), bien que davantage de travail soit effectué avant la copie, parce que certaines réductions détruisent du partage pour atteindre la forme normale complète. Par exemple, certaines variables libres peuvent être copiées. Les stratégies closes évitent ce problème.

L'efficacité des stratégies closes sera démontrée expérimentalement dans la Section IV.9. Le Chapitre IV servira à justifier que chaque pas de réduction a bien un coût algorithmique constant, et donc que le nombre de pas de réductions est une mesure de coût valable.

Il est à noter que le Chapitre IV ne propose une implantation que du calcul λ_{cf}. Or, la stratégie basée sur λ_{caf} offre naturellement un meilleur partage que λ_{cf}, donc est virtuellement plus efficace, mais nous n'avons pas de résultat d'implantation pour λ_{caf}, et nous ne sommes donc pas certains de pouvoir considérer les étapes de réductions de ce calcul comme ayant un coût constant.

II.6 Conclusion

Dans ce chapitre, nous avons proposé une famille de calculs avec substitutions explicites et sans α-conversion, dans lesquels le principe essentiel est que la réduction doit être simple et doit capturer les chemins de réduction les plus courts. Il n'y a aucune liste de substitutions

ou opérations pour les manipuler. L'effacement et la copie sont explicites, afin de guider la substitution aux endroits où elle est nécessaire et de l'effacer dès que possible quand elle ne l'est pas, conformément aux intuitions naturelles pour la propagation des substitutions.

Nous avons développé une famille de stratégies pour la réduction close basées sur ces calculs, et les résultats expérimentaux suggèrent que la réduction close se comporte mieux que les stratégies habituelles. Cependant, rien n'indique que les étapes de réduction peuvent être implantées efficacement. C'est le rôle des deux chapitres suivants de justifier ce point.

Chapitre III

Directeurs pour la récriture d'ordre supérieur

Le Chapitre II nous a donné l'occasion d'exhiber une stratégie de réduction efficace dans un certain λ-calcul avec substitutions explicites. Cependant, les règles de réduction de ce calcul font intervenir certaines conditions qu'il nous reste à savoir implanter efficacement. L'objet de ce chapitre est de proposer, dans le cadre très général de la récriture d'ordre supérieur, une représentation des variables libres qui réponde à ce problème. Cette représentation innovante, bien que très naturelle, est suffisamment abstraite pour être adaptée dans beaucoup de contextes différents et plus satisfaisante d'un point de vue opérationnel que les représentations habituelles.

III.1 Introduction

Nous nous posons le problème très spécifique d'implanter efficacement les systèmes de récriture du Chapitre II. Cependant, ce type de problème apparaît en fait de façon très générale dans de nombreux domaines de l'informatique où il existe une notion de variable et de substitution, par exemple les formalismes de récriture de toutes sortes, le λ-calcul, les langages de programmation, *etc.* La substitution est souvent l'*éminence grise*, pour citer Abadi *et al.* [ACCL91], de ces formalismes parce que sa signification intuitive (remplacer une variable par "quelque chose") cache souvent des problèmes potentiels, comme la capture de nom en présence de lieurs, et un certain coût algorithmique intrinsèque. Rendre le processus de substitution *explicite* est alors une alternative normale afin de mettre le coût algorithmique de la substitution dans le système lui-même (c'est ce que nous avons fait au Chapitre II). Mais les substitutions explicites sont seulement à la moitié du chemin et une certaine complexité peut encore être cachée dans les règles. En particulier la *représentation* des variables est d'une importance cruciale pour l'efficacité du processus de substitution, et c'est le sujet auquel nous nous intéressons dans ce chapitre. Plus précisément, nous proposons une représentation qui est à la fois efficace au sens où elle permet les opérations habituelles en temps constant et très naturelle puisqu'elle est duale (en un sens qui sera précisé ultérieurement) de la représentation habituelle. Les premières prémisses de cette idée sont dues à Kennaway et Sleep [KS88], où des chaînes de directeurs ont été utilisées pour la réduction de combinateurs. Cette idée sera alors appliquée dans le Chapitre IV à un λ-calcul avec substitutions explicites général ainsi qu'à la réduction close du Chapitre II.

Aperçu. La question que nous nous posons sera introduite avec précision en Section III.2, puis la Section III.3 sera consacrée à présenter la notion de *directeurs*, qui est la représentation des variables libres que nous proposons. La réduction doit alors tenir compte de ces directeurs, nous verrons comment lorsqu'il s'agit de systèmes de récriture de termes (TRS, Section III.4) et de systèmes de réduction combinatoire (CRS, Section III.5). Des exemples plus complexes seront développés dans la Section III.6 et nous conclurons en Section III.7.

III.2 Position du problème

Les systèmes de récriture de termes, les systèmes de récriture d'ordre supérieur, le λ-calcul et beaucoup d'autres systèmes permettent d'exprimer un certain type de calcul de manière relativement abstraite (par opposition aux machines de Turing, par exemple). L'abstraction est un aspect plaisant de ces systèmes, toutefois le vrai coût du calcul peut alors être très difficile à estimer et peut dépendre de la manière dont le système abstrait est implanté concrètement.

Il y a quelques résultats au sujet de la complexité algorithmique intrinsèque (c'est-à-dire indépendante de l'implantation) de certains systèmes. Par exemple, un théorème classique de Statman établit que le coût d'une seule étape de β-réduction dans n'importe quelle implantation du λ-calcul peut être non-élémentaire [Sta79]. Par conséquent, la β-réduction ne peut pas être considérée comme une étape atomique (par opposition à une transition dans une machine de Turing), et ceci motive une recherche d'étapes *plus atomiques*. Dans le cas du λ-calcul, une

idée très naturelle introduite par Abadi *et al.* [ACCL91] est de mettre la notion implicite, extensionnelle de substitution dans le système lui-même, en faisant par conséquent certains choix quant à sa définition et en étant plus concret, plus *explicite* quant à la façon dont les choses sont effectuées, d'un point de vue algorithmique. Ceci a motivé une grande variété de travaux dans le domaine ainsi nommé des *substitutions explicites* (par exemple [ACCL91, LRD95, DG01] sans prétention d'exhaustivité).

Plus précisément, dans le λ-calcul (traditionnel, implicite), la substitution est définie à l'aide d'*égalités* telles que (voir la Section I.3) :

$$(t\ u)\{x := v\} \triangleq (t\{x := v\})\ (u\{x := v\}).$$

Ceci est vraiment une définition extensionnelle : elle n'est pas sensée indiquer comment *calculer* la substitution, mais seulement la *définir*. En particulier, l'expression $t\{x := v\}$ n'est pas du tout dans la syntaxe des termes : la substitution et son évaluation sont toutes deux en dehors du système. Cependant, le point trompeur est que cette définition est constructive au sens où elle donne également un algorithme (intentionnel), en orientant simplement ces égalités (de gauche à droite) comme règles de récriture de termes avec substitutions explicites, de la façon suivante (où $t[x/v]$ fait partie de la syntaxe des termes *avec substitutions explicites*) :

$$(t\ u)[x/v] \to (t[x/v])\ (u[x/v]).$$

Ceci est la façon la plus naturelle (pour ne pas dire naïve) de rendre la substitution explicite dans le λ-calcul. Cependant, cette approche n'est pas forcément satisfaisante pour la raison suivante : supposons que x n'apparaisse libre que dans t et pas dans u, alors nous effectuons probablement un certain travail inutile :

- pour copier v ;
- pour propager la substitution dans $u[x/v]$;
- pour effacer finalement v quand nous réalisons qu'il n'est pas nécessaire.

Nous disons "probablement" parce que ce qui sera vraiment algorithmiquement coûteux n'est pas encore clair dans ce cadre (moins, mais toujours) abstrait. Par exemple, avec une stratégie d'appel par nom, aucune réduction ne sera effectuée à l'intérieur de v, par conséquent il n'y a aucun besoin d'effectuer réellement la duplication à ce stade, et il suffit seulement de dupliquer un pointeur vers v (en supposant qu'un tel outil soit disponible dans le cadre concret).

Ceci nous incite à exprimer la règle de récriture précédente sous la forme d'un ensemble de règles de récriture conditionnelles (c'est ce qu'on a fait dans le chapitre précédent) de la forme suivante :

$$(t\ u)[x/v] \to (t[x/v])\ u \quad \text{si } x \in \mathsf{fv}(t) \text{ et } x \notin \mathsf{fv}(u).$$

Cette solution évite les défauts précédents, mais maintenant un certain coût peut être caché dans les conditions sur les variables libres. Par exemple, calculer les ensembles de variables libres à partir de zéro à chaque fois a un coût linéaire en la taille des termes. C'est clairement trop cher pour un seul pas de récriture. Nous devons donc affiner la *représentation* des variables libres, et c'est le sujet de ce chapitre.

III.3 Des variables aux directeurs

Dans cette section, nous introduisons la notion de directeurs comme représentation des variables libres. Nous traiterons de la réduction dans les Section III.4 (TRS) et III.5 (CRS), ainsi nous nous intéressons ici principalement aux termes. Pour rendre notre propos plus concret et accessible, nous considérons des termes de premier ordre comme ceux des systèmes de récriture de termes (TRS), bien que les idées s'appliquent abstraitement à n'importe quelle algèbre de termes, et en particulier à la récriture d'ordre supérieur comme nous le montrerons dans la Section III.5 pour les systèmes de réduction combinatoire.

III.3.1 Variables d'un terme

Nous considérons une algèbre de termes $\mathcal{T} = \mathcal{T}(\mathcal{F}, \mathcal{V})$ définie par la grammaire :

$$t ::= x \mid f(t_1, \dots, t_n) \text{ où } x \in \mathcal{V} \text{ et } f \in \mathcal{F} \text{ d'arité } n.$$

L'ensemble des variables libres d'un terme est habituellement défini inductivement de la façon suivante, par une fonction $\mathsf{fv} : \mathcal{T} \to \mathcal{P}(\mathcal{V})$ (où $\mathcal{P}(X)$ est l'ensemble des parties de X) :

$$\begin{cases} \mathsf{fv}(x) = \{x\} \\ \mathsf{fv}(f(t_1, \dots, t_n)) = \mathsf{fv}(t_1) \cup \dots \cup \mathsf{fv}(t_n) \end{cases}$$

C'est une définition, mais ceci donne également un algorithme (naïf) pour calculer l'ensemble des variables libres d'un terme de façon ascendante (*bottom-up*) : partir des feuilles de l'arbre de syntaxe représentant le terme (les variables) et, à chaque nœud, prendre l'union des ensembles de variables libres dans tous les fils (sous-termes) du nœud.

Cependant, cet algorithme est linéaire en la taille du terme, ce qui est le mieux que nous pouvons espérer sans information préalable, mais ce qui est coûteux. Si les informations sur les variables libres sont d'importance (comme dans l'exemple de la Section III.2), ce ne serait certainement pas une bonne idée d'effectuer ce calcul à chaque étape de récriture. Une meilleure solution serait de conserver le terme ainsi que son ensemble de variables libres, sous la forme d'un couple $(t, \mathsf{fv}(t))$, et de s'assurer que la réduction préserve l'information, c'est-à-dire définir une réduction $\rightsquigarrow$ sur les couples de termes et d'ensembles de variables libres tels que si $t \to u$ alors $(t, \mathsf{fv}(t)) \rightsquigarrow (u, \mathsf{fv}(u))$.

En général, si $t \to u$, l'information sur $\mathsf{fv}(t)$ ne suffit pas pour calculer $\mathsf{fv}(u)$: nous avons donc besoin d'une représentation plus riche. Connaissant la règle $l \to r$ utilisée et $\mathsf{fv}(t')$ pour tout t' sous-terme de t, l'information sera facile à maintenir, parce qu'un pas de récriture est local dans le sens qu'il ne modifie pas les sous-termes suffisamment profonds. Le candidat suivant pour représenter les termes est donc : $(t, \{\mathsf{fv}(t'), t' \text{ sous-terme de } t\})$.

Maintenant c'est légèrement trop : en fait seule l'information sur les sous-termes *immédiats* est nécessaire. Si $t = f(t_1, \dots, t_n)$, nous avons seulement besoin de $\mathsf{fv}(t_i)$ pour $1 \leq i \leq n$. Si nous définissons $\upsilon_t : \{1, \dots, n\} \to \mathcal{P}(\mathcal{V})$ par $\upsilon_t(i) = \mathsf{fv}(t_i)$, alors (t, υ_t) est une bonne représentation parce que $\bigcup_{1 \leq i \leq n} \upsilon_t(i) = \mathsf{fv}(t)$ (υ_t est suffisant pour retrouver $\mathsf{fv}(t)$), et $\upsilon_t(i) = \bigcup_{1 \leq j \leq m} \upsilon_{t_i}(j)$ (pour le bon m), ainsi υ est également inductif. Il est à noter que υ_t est une fonction (au sens mathématique), et nous disons que c'est une bonne représentation parce

que son domaine est fini (et habituellement petit), par conséquent il peut être efficacement représenté par un tableau. Nous supposerons habituellement cela et omettrons l'étape finale vers l'implantation.

III.3.2 Renversons la vapeur

Nous avons maintenant une représentation sympathique des *ensembles* de variables libres d'un terme. Cela signifie qu'à chaque étape de la substitution, nous devrons traverser (un certain type de données représentant) des ensembles pour tester l'appartenance. Ce n'est pas exactement ce que nous voulons. Pour chaque position i dans un symbole f d'arité n, nous avons de l'information sur les ensembles correspondants de variables libres (du sous-terme t_i), mais en réalité nous voulons l'inverse : pour chaque variable libre x du terme $f(t_1, \ldots, t_n)$, nous voulons connaître l'ensemble (maximal) des positions i telles que x est également libre dans t_i, de sorte que nous puissions propager une substitution pour x dans ces sous-termes seulement.

Notons $\mathcal{L}_n = \mathcal{P}(\{1, \ldots, n\})$, et $\mathcal{L} = \uplus_{n\in\mathbb{N}}\mathcal{L}_n$, où $\uplus$ dénote l'union disjointe. On appelle $\mathcal{L}$ l'ensemble des *positions immédiates* ou des *localisations*. Si $S \in \mathcal{L}$, nous nous permettons d'être explicites quant au n tel que le $S \in \mathcal{L}_n$ par indiçage de la forme S_n. Nous abrégeons $\{1\}_1$ en $\downarrow$. Alors nous avons : $\{1, \ldots, n\} \to \mathcal{P}(\mathcal{V}) \simeq \{1, \ldots, n\} \times \mathcal{V} \to 2 \simeq \mathcal{V} \times \{1, \ldots, n\} \to 2 \simeq \mathcal{V} \to \mathcal{P}(\{1, \ldots, n\}) = \mathcal{V} \to \mathcal{L}_n$ (où 2 est l'ensemble à deux éléments), autrement dit υ_t définit en fait une relation entre $\{1, \ldots, n\}$ et $\mathcal{V}$ et nous pouvons prendre la relation duale. Alors c'est seulement une question de commodité (il faut toujours penser en termes de tableau) de l'orienter comme une fonction $\mathcal{V} \to \mathcal{L}_n$, que nous notons σ_t et appelons le *directeur* de t (noter que cela correspond aux *chaînes directrices* du Chapitre IV. Plus formellement, définissons la relation $\mathcal{R}$ par $i\,\mathcal{R}\,x \Leftrightarrow x \in \upsilon_t(i)$ et $\mathcal{R}^{-1}$ par $x\,\mathcal{R}^{-1}\,i \Leftrightarrow i\,\mathcal{R}\,x$. Alors σ_t est la fonction $\mathcal{V} \to \mathcal{L}_n$ telle que $x\,\mathcal{R}^{-1}\,i \Leftrightarrow i \in \sigma_t(x)$. Par conséquent si $x \in \mathcal{V}$, $\sigma_t(x)$ donne l'ensemble des positions immédiates de t où la variable x apparaît libre, et en particulier où une substitution pour x devrait être propagée.

III.3.3 Traductions

Étant donné un terme $t \in \mathcal{T}$, nous appelons *compilation* le calcul initial de σ, c'est-à-dire $\sigma_{t'}(x)$ pour chaque sous-terme t' de t et pour chaque variable x (noter que $\sigma_{t'}(x) \neq \varnothing$ seulement pour un nombre fini de x). La compilation n'est effectuée qu'une fois, puis σ est supposée être recalculée incrémentalement (c'est-à-dire en évitant de recompiler le terme entièrement) pendant la réduction ; ce sujet est discuté dans les Sections III.4 et III.5. Nous dénotons la fonction de compilation par $[\![\cdot]\!]$, de sorte que $[\![t]\!] = (t, \sigma)$. La signification de bas niveau est la suivante : à chaque nœud de l'arbre de syntaxe du terme t, nous attachons un tableau de listes d'entiers (les positions) indexé par (un sous-ensemble fini de) $\mathcal{V}$. La compilation peut être faite en calculant les ensembles de variables libres de chaque sous-terme et en prenant la relation duale correspondant à υ.

En y pensant comme à une matrice à coefficients dans $\{0, 1\}$ indexée sur $\{1, \ldots, n\}$ et $\mathcal{V}$, il est clair qu'aucun surcoût n'est introduit : il suffit simplement de lire d'abord les colonnes au lieu des lignes. Le processus entier de la compilation a ainsi une complexité linéaire, ce qui

est très acceptable puisque ceci est censé être fait seulement une fois.

Exemple III.3.1

En guise d'illustration, supposons que f et g sont des symboles fonctionnels, et x, y, z sont des variables. Soit $t = f(x, y, g(x, z))$, alors son directeur σ est donné par :

$$\begin{array}{lllll} \sigma_x : & x \mapsto \{1\} & y \mapsto \varnothing & z \mapsto \varnothing \\ \sigma_y : & x \mapsto \varnothing & y \mapsto \{1\} & z \mapsto \varnothing \\ \sigma_z : & x \mapsto \varnothing & y \mapsto \varnothing & z \mapsto \{1\} \\ \sigma_{g(x,z)} : & x \mapsto \{1\} & y \mapsto \varnothing & z \mapsto \{2\} \\ \sigma_t : & x \mapsto \{1,3\} & y \mapsto \{2\} & z \mapsto \{3\} \end{array}$$

A la fin de la réduction d'un terme annoté, nous voulons pouvoir relire le résultat comme un terme standard, c'est ce qu'on appelle la *décompilation*. Nous notons cette fonction de décompilation $(\!|\cdot|\!)$. Dans notre cas, nous préservons la structure des termes et les noms des variables, ainsi la relecture est facile : il suffit simplement d'oublier σ, par conséquent $(\!|[\![t]\!]|\!) = (\!|(t, \sigma)|\!) = t$. D'un point de vue opérationnel, nous ignorons juste l'information supplémentaire attachée à chaque nœud.

III.3.4 Remarques

Pour la clarté de l'exposé, nous avons considéré une algèbre de termes concrète au cours de cette section. Cependant, très peu de choses dépendent de cette structure concrète. En fait, notre raisonnement est assez général pour s'adapter à n'importe quel cadre muni d'une algèbre de termes définie inductivement, d'une notion de position, de variable et d'ensembles de variables libres (définis inductivement sur les termes).

Une présentation alternative, qui est celle adoptée dans le Chapitre IV, consiste à mettre les directeurs directement dans la syntaxe des termes, de la façon suivante :

$$\mathbf{t} ::= x \mid f(\mathbf{t}_1, \ldots, \mathbf{t}_n)^\sigma \text{ où } \sigma : \mathcal{V} \to \mathcal{L}_n.$$

Cela a l'avantage d'être explicite à la fois quant aux termes sur lesquels σ est calculée et à la localisation de l'information. Par commodité, nous préférons garder cet aspect implicite dans la suite de ce chapitre, bien que ce soit exactement ce que nous avons à l'esprit quand nous parlons de (t, σ).

III.4 Maintenir les directeurs

Dans la section précédente, nous avons obtenu une représentation des variables libres appropriée au genre de problèmes exposés dans la Section III.2. Cela aurait peu d'intérêt si cette représentation ne pouvait être maintenue par la réduction à un coût relativement bas. Nous démontrons dans cette section qu'il n'en est rien.

Par souci de clarté, nous continuons avec l'exemple des systèmes de récriture de termes (TRS) [DJ89, Ter03], bien que les résultats de cette section soient subsumés dans la section suivante traitant des systèmes de réduction combinatoire (CRS).

Il y a un léger paradoxe dans notre présentation : l'introduction des directeurs est motivée par la nécessité d'avoir des règles de récriture avec certaines conditions sur les variables libres de certains sous-termes. Cependant, par souci de simplicité, nous considérons des systèmes de récriture de termes non conditionnels, qui ne permettent pas d'exprimer ce genre de règles. Il sera clair qu'ajouter de telles conditions ne pose aucun problème, et que les directeurs permettent effectivement de résoudre ces conditions en temps constant.

Nous considérons des termes déjà annotés (par exemple, obtenus à partir de la compilation, mais pas nécessairement) et expliquons comment modifier la relation de réduction pour préserver des directeurs corrects.

III.4.1 Syntaxe

Les termes t et termes annotés $\mathbf{t}$ sont définis par :

$$\begin{array}{lll} t & ::= & x \mid f(t_1, \ldots, t_n) \qquad \text{où } x \in \mathcal{V}, f \in \mathcal{F} \text{ d'arité } n \\ \mathbf{t} & ::= & (t, \sigma) \qquad\qquad\qquad\quad \text{où } \sigma : \mathcal{T} \to \mathcal{V} \to \mathcal{L}. \end{array}$$

Les termes annotés $\mathbf{t}$ sont donc simplement des couples (t, σ) où $\sigma : \mathcal{T} \to \mathcal{V} \to \mathcal{L}$ est une fonction partielle (nous notons σ_t au lieu de $\sigma(t)$) telle que :

- si $x \in \mathcal{V}$, $\sigma_x(y) = \begin{cases} \downarrow & \text{si } y = x, \\ \varnothing & \text{sinon;} \end{cases}$
- si $f(t_1, \ldots, t_n) \in \mathcal{T}$, $\forall i, 1 \leq i \leq n, \forall x \in \mathcal{V}, \sigma_{t_i}(x) \neq \varnothing \Rightarrow i \in \sigma_{f(t_1,\ldots,t_n)}(x)$.

Remarque III.4.1

Une définition plus restrictive est donnée en remplaçant $\Rightarrow$ ci-dessus par $\Leftrightarrow$. Cette définition alternative permet de récupérer *exactement* l'ensemble des variables libres et donne une représentation unique des termes. En revanche, il est plus difficile de préserver cette représentation par réduction. La définition choisie est une forme d'interprétation abstraite : une variable peut soudainement disparaître en descendant dans un terme, mais au moins, nous sommes sûrs de les atteindre toutes. Nous discutons ce point plus en détails ci-dessous.

Lemme III.4.2

La compilation comme expliquée dans la section précédente donne des termes annotés valides (y compris dans le sens fort).

III.4.2 Réduction

Comme expliqué dans la Section I.2, un système de récriture de termes est constitué d'un ensemble de paires $l \to r$ de termes, appelées règles de récriture, et nous disons que t se récrit en u s'il existe une règle $l \to r$, une position p dans l'arbre de syntaxe de t et une substitution ς tels que $t|_p = l\varsigma$ et $u = t[r\varsigma]_p$, c'est-à-dire tels que le sous-terme de t à la position p est l (modulo une substitution) et tels que u est t où le sous-terme à la position p est remplacé par r (modulo la même substitution).

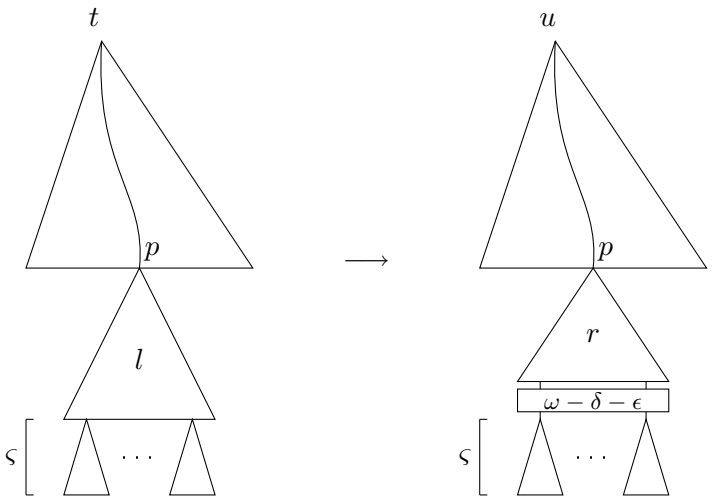


FIG. III.1 – Réduction dans un TRS

Autrement dit, la récriture est juste un remplacement local de l par r dans t, sans changer le contexte englobant l. Cela s'explique mieux graphiquement : dans la Figure III.1, la seule partie du terme qui change est celle du milieu : le contexte et la substitution sont identiques. Notons cependant que trois types de choses peuvent se produire dans la boîte marquée $\omega - \delta - \epsilon$. Les variables qui sont libres à la fois dans l et dans le support de ς peuvent apparaître dans r à des endroits différents (réarrangement), plusieurs fois (duplication), ou pas du tout (effacement). Cette observation sera cruciale, comme nous le verrons plus tard.

Dans un cadre muni de directeurs, nous devons d'abord adapter la notion de substitution.

DÉFINITION III.4.3 (*Substitution*)

> Une substitution (annotée) ς est une fonction totale de l'ensemble des variables dans l'ensemble des termes (annotés). Nous définissons habituellement une substitution sur un nombre fini de variables seulement, avec la convention qu'elle est prolongée identiquement à $\mathcal{V}$ (c'est-à-dire $\varsigma(x) = x$ par défaut). Le support de ς est l'ensemble des x tels que $\varsigma(x) \neq x$.

Nous voudrions donc définir une règle de récriture comme une paire $\mathbf{l} \rightsquigarrow \mathbf{r}$ de termes annotés et dire que $\mathbf{t}$ se récrit en $\mathbf{u}$ s'il y a une règle $\mathbf{l} \rightsquigarrow \mathbf{r}$, une position p dans l'arbre de syntaxe de t et une substitution annotée ς tels que $\mathbf{t}|_p = \mathbf{l}\varsigma$ et $\mathbf{u} = \mathbf{t}[\mathbf{r}\varsigma]_p$.

Nous devons ainsi définir le résultat de la substitution $\mathbf{t}\varsigma$ de $\mathbf{t}$ avec la substitution ς en présence de directeurs. Nous voulons évidemment vérifier la spécification suivante :

$$\mathbf{t}\varsigma = [\![(\!|\mathbf{t}|\!)\varsigma]\!]$$

où la substitution est effectuée sur les termes non étiquetés habituels, puis le terme est compilé à nouveau. Ce n'est clairement pas satisfaisant d'un point de vue opérationnel, puisque nous devons recompiler le terme et la compilation est une opération plutôt coûteuse.

Définition III.4.4

La substitution sur les termes avec directeurs peut être définie par $(t,\sigma)\varsigma = (t\varsigma,\sigma')$ tel que :

$$\begin{cases} \sigma'_x & = \sigma_{\varsigma(x)} \\ \sigma'_{f(t_1,\ldots,t_n)}(x) & = \bigcup\limits_{y,\sigma_{\varsigma(y)}(x)\neq\varnothing} \sigma_{f(t_1,\ldots,t_n)}(y). \end{cases}$$

Preuve. La définition est valide puisqu'elle satisfait la spécification $\mathbf{t}\varsigma = [\![(\!|\mathbf{t}|\!)\varsigma]\!]$:

$$\begin{cases} \sigma'_x & = \sigma_{x\varsigma} = \sigma_{\varsigma(x)} \\ \sigma'_{f(t_1,\ldots,t_n)}(x) & = \sigma_{f(t_1,\ldots,t_n)\varsigma}(x) = \sigma_{f(\mathbf{t}_1\varsigma,\ldots,\mathbf{t}_n\varsigma)}(x) \\ & = \bigcup\limits_{y,x\in\mathsf{fv}(y\varsigma)} \sigma_{f(t_1,\ldots,t_n)}(y) = \bigcup\limits_{y,\sigma_{\varsigma(y)}(x)\neq\varnothing} \sigma_{f(t_1,\ldots,t_n)}(y). \end{cases}$$

□

III.4.3 Localité

Le recalcul du directeur attaché à un nœud donné est donc possible, en utilisant l'expression ci-dessus. En regardant à nouveau la Figure III.1, il est clair qu'il faut recalculer les directeurs partout dans l'image de r dans u, mais c'est indépendant de la taille du terme. Mais devons-nous également effectuer ce recalcul partout ailleurs dans le terme ?

Les variables libres d'un terme dépendent seulement des variables libres de ses sous-termes, et il en va de même pour σ. Notre première observation est donc que les sous-termes de r qui sont dans l'image de ς (en bas de la figure) conservent les mêmes directeurs, par conséquent aucun recalcul n'est nécessaire. Le même argument vaut également pour n'importe quel nœud de la partie supérieure du terme (le contexte) qui n'est pas sur un chemin vers p.

Imaginons maintenant que seulement des réarrangements ou des duplications se produisent dans la boîte $\omega{-}\delta{-}\epsilon$ (pas d'effacement). Alors à la position p, l'ensemble de variables libres est le même dans u que dans t (parce que l'union est associative, commutative et idempotente). En d'autres termes, les directeurs ne sont pas modifiés non plus sur le chemin vers p, sauf peut-être à la position exactement p (parce que les directeurs fournissent des informations sur les fils du nœud correspondant).

En cas d'effacement, les nœuds au-dessus de p (c'est-à-dire sur le chemin jusqu'à p) doivent naturellement être mis à jour si nous voulons conserver l'information exacte sur les variables libres au niveau de ces nœuds. Ce coût est proportionnel à la profondeur de p, ce qui semble acceptable.

Cependant, nous pouvons également adopter une approche légèrement différente, comme nous le ferons dans le Chapitre IV : nous pouvons restaurer la localité de la récriture en abandonnant la mise à jour le long du chemin vers p, au prix de propager inutilement certaines substitutions qui seront effacées plus tard. A peu de choses près, c'est aussi efficace, et cela permet de compter ce coût explicitement au lieu de le cacher dans la réduction. En d'autres

termes, le coût d'un pas de réduction est indépendant de la taille du terme, ce qui est habituellement souhaitable ; mais quelques réductions supplémentaires peuvent être nécessaires.

III.5 Systèmes de réduction combinatoire avec directeurs

Nous avons donc défini la notion de directeurs dans le cadre de la récriture du premier ordre. Cependant, le type de problèmes rencontré dans la Section III.2 fait intervenir une notion de variable liée ; c'est pourquoi il est nécessaire de généraliser ce travail à la récriture d'ordre supérieur, de façon à prendre en compte dans la théorie la présence éventuelle de lieurs. Nous adaptons donc notre formalisme à celui des systèmes de réduction combinatoire (CRS, voir la Section I.4), qui constitue un cadre de récriture très général.

L'adaptation aux CRS est satisfaisante d'un point de vue théorique, mais malheureusement pas d'un point de vue pratique, comme nous le verrons. C'est pourquoi nous adapterons alors dans la Section III.5.5 la notion de directeurs au cas particulier des systèmes de réduction combinatoire avec substitutions explicites (ESCRS, voir Section I.4.3), qui constituent un cadre plus adapté pour donner un coût réaliste au calcul, tout en bénéficiant d'une notion de lieur.

III.5.1 Syntaxe

Pour adapter aux CRS le travail effectué sur les TRS, nous devons prendre en considération les deux aspects suivants (voir Section I.4) :

- l'abstraction $[x]t$;
- la distinction entre variables et métavariables.

Le premier point est facilement traité : nous devons simplement modifier les conditions sur les directeurs de façon à ce que dans un terme annoté, on ait $\sigma_{[x]t}(x) = \varnothing$ tandis que $\sigma_t(x)$ puisse être non vide (rappelons que nous travaillons sous la convention de Barendregt).

Le deuxième point est plus subtil. Notons d'abord que seuls les directeurs sur les variables sont nécessaires pour représenter les termes. Les termes avec métavariables n'ont de sens que dans les règles de récriture, ainsi le meilleur choix est d'annoter les métavariables de sorte que la substitution soit uniforme quand une règle est appliquée à un terme, c'est-à-dire de sorte que les métavariables se comportent de la même façon que les variables quand une substitution est appliquée. Les métatermes de la forme $Z(t_1, \ldots, t_n)$ sont ainsi considérés comme unaires et par exemple $Z(x, y)$ est annoté par $\{Z \mapsto \downarrow, x \mapsto \downarrow, y \mapsto \downarrow\}$. Notons que les directeurs pour les métatermes mélangent des informations sur les variables et les métavariables ; cependant, après instantiation, il y aura seulement des informations sur les variables.

Un métaterme annoté est une paire $\mathbf{t} = (t, \sigma)$ où t est un métaterme et $\sigma : \mathcal{MT} \to (\mathcal{V} \uplus \mathcal{MV}) \to \mathcal{L}$ est appelé directeur. Ici encore, nous écrivons $\sigma_t(x)$ au lieu de $\sigma(t)(x)$. Nous disons que σ_t est de rang n si pour toute variable ou métavariable α, $\sigma_t(\alpha) \in \mathcal{L}_n$. Le directeur σ doit satisfaire les conditions suivantes (α représente n'importe quelle variable ou métavariable) :

- Variables :
  - σ_x est de rang 1 ;
  - $\sigma_x(x) = \downarrow$;

  - $\sigma_x(\alpha) = \varnothing$ pour $\alpha \neq x$.
- Abstractions :
  - $\sigma_{[x]t}$ est de rang 1 ;
  - $\sigma_{[x]t}(x) = \varnothing$;
  - $\sigma_t(\alpha) \neq \varnothing \Rightarrow \sigma_{[x]t}(\alpha) = \downarrow$ pour $\alpha \neq x$.
- Symboles fonctionnels :
  - $\sigma_{f(t_1,\ldots,t_n)}$ est de rang n ;
  - $\forall i, 1 \leq i \leq n, (\sigma_{t_i}(\alpha) \neq \varnothing \Rightarrow i \in \sigma_{f(t_1,\ldots,t_n)}(\alpha))$.
- Métavariables :
  - $\sigma_{Z(t_1,\ldots,t_n)}$ est de rang 1 (et non n) ;
  - $\sigma_{Z(t_1,\ldots,t_n)}(Z) = \downarrow$;
  - $(\exists i, 1 \leq i \leq n, \sigma_{t_i}(\alpha) \neq \varnothing) \Rightarrow \sigma_{Z(t_1,\ldots,t_n)}(\alpha) = \downarrow$;
  - $(\forall i, 1 \leq i \leq n, \sigma_{t_i}(\alpha) = \varnothing) \Rightarrow \sigma_{Z(t_1,\ldots,t_n)}(\alpha) = \varnothing$ pour $\alpha \neq Z$.

La dernière condition assure par exemple que $\sigma_{Z(x)}(y) = \varnothing$ alors qu'on peut avoir $\sigma_{f(x)}(y) = \downarrow$ avec y effacé plus bas dans le terme (en arrivant sur x dans ce cas).

La compilation et la décompilation sont similaires à la Section III.3 et omis.

III.5.2 Réduction

Les règles de récriture sont de la forme $(l, \sigma) \to (r, \sigma')$. La relation de réduction est définie comme dans la section précédente à la différence que la substitution est étendue aux termes annotés par $(t, \sigma)\varsigma = (t\varsigma, \sigma')$ où :

$$\sigma'_x = \sigma_{x\varsigma} = \sigma_x, \text{d'où } \sigma'_x(\alpha) = \begin{cases} \downarrow & \text{si } \alpha = x, \\ \varnothing & \text{sinon.} \end{cases}$$

$$\sigma'_{[x]t} = \sigma_{([x]t)\varsigma} = \sigma_{[x](t\varsigma)}, \text{d'où } \sigma'_{[x]t}(\alpha) = \begin{cases} \downarrow & \text{si } \exists \beta, \sigma_t(\beta) \neq \varnothing \wedge \sigma_{\varsigma(\beta)}(\alpha) \neq \varnothing, \\ \varnothing & \text{sinon.} \end{cases}$$

La capture de variables est interdite par des conditions implicites, ainsi $\sigma_{\varsigma(\alpha)}(x) = \varnothing$ pour tout α. Il faut aussi préciser que la quantification existentielle est beaucoup plus opérationnelle qu'il n'y paraît, puisque la recherche est bornée à la fois par la taille du domaine de σ_t (c'est-à-dire le nombre de variables libres de t) et par la taille du support de ς.

$$\sigma'_{f(t_1,\ldots,t_n)} = \sigma_{f(t_1,\ldots,t_n)\varsigma} = \sigma_{f(t_1\varsigma,\ldots,t_n\varsigma)}, \text{ d'où } \sigma'_{f(t_1,\ldots,t_n)}(\alpha) = \bigcup_{\sigma_{\varsigma(\beta)}(\alpha) \neq \varnothing} \sigma_{f(t_1,\ldots,t_n)}(\beta).$$

De nouveau, l'union est finie et relativement petite de sorte que le calcul est en fait facile.

Pour les métavariables, il y a trois cas selon la substitution ς. Si $\varsigma(Z) = Z$, alors :

$$\sigma'_{Z(t_1,\ldots,t_n)} = \sigma_{Z(t_1,\ldots,t_n)\varsigma} = \sigma_{Z(t_1\varsigma,\ldots,t_n\varsigma)}$$
$$\text{d'où } \sigma'_{Z(t_1,\ldots,t_n)}(\alpha) = \begin{cases} \downarrow & \text{si } \alpha = Z \text{ ou si } \exists i, \beta, \sigma_{t_i}(\beta) \neq \varnothing \wedge \sigma_{\varsigma(\beta)}(\alpha) \neq \varnothing, \\ \varnothing & \text{sinon.} \end{cases}$$

Si $\varsigma(Z)$ est une (méta)-projection, c'est-à-dire si $\varsigma(Z) = \underline{\lambda}(x_1 \dots x_n).x_j$ pour un certain j, alors $Z(t_1, \dots, t_n) = t_j$ et :

$$\sigma'_{Z(t_1,\dots,t_n)}(\alpha) = \begin{cases} \varnothing & \text{si } \alpha = x_k \text{ pour un certain } k, \\ \sigma_{t_j}(\alpha) & \text{sinon.} \end{cases}$$

Finalement, dans le cas général, $\varsigma(Z) = \underline{\lambda}(x_1 \dots x_n).t$ (en s'assurant qu'aucun des deux cas précédents ne s'applique), de sorte que $Z(t_1, \dots, t_n) = t\{x_1 := t_1, \dots, x_n := t_n\}$.

$$\sigma'_{Z(t_1,\dots,t_n)}(\alpha) = \begin{cases} \varnothing & \text{si } \alpha = x_k \text{ pour un certain } k, \\ \sigma_t(\alpha) \ \cup \bigcup\limits_{\substack{1 \le j \le n, \\ \sigma_{t_j}(\alpha) \neq \varnothing}} \sigma_t(x_j) & \text{sinon.} \end{cases}$$

Il est clair que la connaissance locale du terme et de la substitution suffisent pour calculer σ', et le nombre de calculs élémentaires requis est borné indépendamment de la taille du terme (pour le calcul d'un seul directeur).

III.5.3 Propriétés

Nous avons défini le résultat de la substitution sur les directeurs, par conséquent nous avons également défini la réduction sur les termes avec directeurs. Par construction, les nouveaux directeurs correspondent aux anciens, par conséquent les informations sur les variables libres sont préservées par réduction. Ceci peut être formalisé de la façon suivante.

Proposition III.5.1

> La substitution comme définie ci-dessus est correcte en ce qui concerne la substitution des (méta)-termes sans directeurs (autrement dit, $\mathbf{t}\varsigma = [\![(\!|\mathbf{t}|\!)\varsigma]\!]$ avec les notations de la Section III.3).

Preuve. Il est facile de vérifier que les étapes intermédiaires dans les dérivations ci-dessus sont correctes. □

Définition III.5.2

> σ est dit *fortement correct* pour t si $x \in \mathsf{fv}(t') \Leftrightarrow \sigma_{t'}(x) \neq \varnothing$ pour tout sous-terme t' de t.

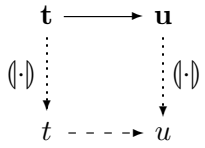
Lemme III.5.3

> - Si σ est fortement correct pour t, alors σ' (comme donné ci-dessus) est fortement correct pour $t\varsigma$.
> - Si σ est fortement correct pour t et $t \to u$, alors σ' est fortement correct pour u, où σ' est défini comme ci-dessus avec la substitution ς correspondant à l'étape de récriture $t \to u$.

Preuve. Conséquences faciles de la Proposition III.5.1. □

THÉORÈME III.5.4 (*Correction*)

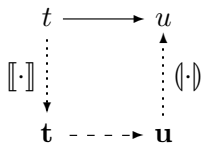
Si $\mathbf{t} \rightsquigarrow \mathbf{u}$ sur des termes annotés, alors $(\!|\mathbf{t}|\!) \to (\!|\mathbf{u}|\!)$ sur des termes standard.



Preuve. Rappelons d'abord que $(\!|\cdot|\!)$ efface seulement de l'information, c'est donc immédiat si les règles de récriture n'ont pas de conditions sur les variables libres. Sinon, cela suit du Lemme III.5.3. □

THÉORÈME III.5.5 (*Complétude*)

Si $t \to u$ sur des termes standard, alors il existe un terme annoté $\mathbf{u}$ tel que $[\![t]\!] \rightsquigarrow \mathbf{u}$ sur les termes annotés et $(\!|\mathbf{u}|\!) = u$.



Preuve. Encore une conséquence du Lemme III.5.3. □

III.5.4 LOCALITÉ

Contrairement au cas des TRS, nous ne pouvons pas garantir dans le cas présent que le nombre de directeurs qui doivent être mis à jour est petit. Cela est dû au mécanisme de substitution implicite dans les CRS, et l'on retrouve donc cette propriété dans le sous-ensemble des CRS avec substitutions explicites (ESCRS, voir section suivante).

Nous pouvons comprendre la situation plus précisément sur un exemple typique. Le λ-calcul est une instance de CRS (voir [KOR93]), dont les symboles sont λ (d'arité 1) et l'application, notée @ (d'arité 2) et dont la seule règle de récriture est (β), donnée par $@(\lambda([x]Z(x)), Y) \to Z(Y)$. Une réduction typique est illustrée par la Figure III.2. Tous les directeurs sur le chemin de la racine du terme à u (dans le terme de droite de la figure) devraient être modifiés pour tenir compte des variables libres de u. Ceci induit un coût linéaire en la taille du contexte plus la taille de t, et n'est pas limité au cas de l'effacement (contrairement au cas des TRS).

Il n'est donc pas réaliste d'utiliser des directeurs pour les CRS en général, mais ce n'est ni une surprise ni une déception : les directeurs indiquent comment diriger les substitutions, ils n'ont donc de sens que quand les substitutions sont explicites. Il est tout de même plaisant d'un point de vue théorique de pouvoir définir les directeurs dans le cadre très général des CRS, même si cela a peu d'utilité pratique.

Nous pouvons ici aussi ajouter dans les règles de récriture des conditions explicites sur les variables libres de certains sous-termes, sans créer de problème et de sorte que les directeurs

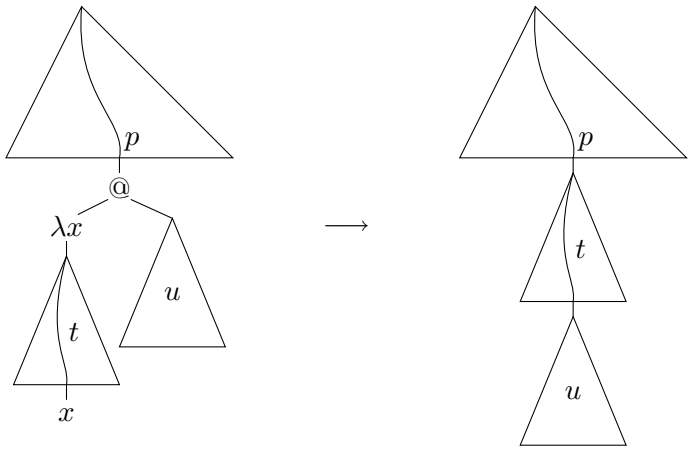


FIG. III.2 – Une β-réduction typique

permettent de résoudre ces conditions en temps constant. On peut noter que certaines conditions sont déjà implicites dans les CRS (par exemple dans $[x]Z$, nous savons que x n'est libre dans aucun terme substitué pour Z ; mais dans $[x]Z(x)$, nous ne pouvons pas être sûrs que $x \in \mathsf{fv}(t)$ si Z est substitué par t). Peut-être que cela doit être compris comme un indice qu'un meilleur cadre devrait être développé pour intégrer d'une manière plus homogène récriture et directeurs.

III.5.5 Substitutions explicites

L'étape de réduction dans un CRS n'est pas une bonne unité de mesure de la complexité algorithmique, en raison de la notion implicite de substitution qui est autorisée arbitrairement profondément dans un terme. De plus, nous n'avons pas de bonnes propriétés de localité, ce qui rend peu réaliste de maintenir des directeurs pendant la réduction. Nous avons donc besoin d'un meilleur cadre pour réaliser nos objectifs, à savoir les CRS avec substitutions explicites, définis dans la Section I.4.3. Les ESCRS sont eux-mêmes des CRS, et ils limitent l'utilisation de la substitution des CRS, de sorte que seule la connaissance locale du terme est nécessaire à chaque étape.

Les ESCRS n'utilisent pas toute la puissance des CRS, ainsi nous pouvons simplifier la définition de la relation de réduction pour cette sous-classe. En fait, nous n'utilisons pas la substitution réminiscente du λ-calcul, par conséquent nous n'avons pas besoin de substituts. Nous disons que les substitutions associent un terme à une application de métavariable de la forme $Z(x_1, \ldots, x_n)$ et nous n'avons plus à traiter les cas de la forme $Z(t_1, \ldots, t_n)$:

$$\varsigma\left(Z(x_1, \ldots, x_n)\right) = t$$

et ς est prolongée homomorphiquement aux métatermes (avec la restriction précédente) de la manière habituelle.

Naturellement, il faut encore prendre soin d'éviter les captures de variables, mais seulement en ce qui concerne les abstractions et plus les substituts (la condition de clôture assure que les variables apparaissant dans une application de métavariable sont liées par une abstraction).

La substitution est maintenant plus simple parce qu'il n'y a plus de métatermes de la forme $Z(t_1, \ldots, t_n)$. Il y a seulement un cas ici. Supposons $\varsigma(Z(x_1, \ldots, x_n)) = t$, alors, très simplement :

$$\sigma'_{Z(x_1,\ldots,x_n)} = \sigma_{Z(x_1,\ldots,x_n)\varsigma} = \sigma_t.$$

Puisque le mécanisme de substitution des CRS est interdit dans les ESCRS, nous sommes dans un cas très semblable à celui des TRS en ce qui concerne la localité : les directeurs doivent être recalculés seulement dans l'image du membre droit des règles et, en cas d'effacement, sur le chemin de la racine du terme à la position où a lieu la récriture. Là encore, cette dernière partie peut être omise au prix de propager quelques substitutions (explicites) supplémentaires plus tard.

III.6 λ-CALCUL AVEC SUBSTITUTIONS EXPLICITES

Nous pouvons dès à présent introduire les chaînes directrices pour le λ-calcul (avec substitutions explicites) qui seront décrites en détails dans le chapitre suivant, et qui entrent totalement dans le cadre des ESCRS avec directeurs développé dans la Section III.5.

Tout d'abord, puisqu'en λ-calcul pur les symboles fonctionnels sont seulement unaires ou binaires, nous utilisons les abréviations suivantes, qui donnent une meilleure intuition des directeurs :

- variable et abstraction (unaires) : $- \equiv \varnothing, \downarrow \equiv \{1\} \in \mathcal{L}_1$;
- application et substitution (binaires) : $- \equiv \varnothing, \curvearrowleft \equiv \{1\}, \curvearrowright \equiv \{2\}, \swarrow\!\!\!\searrow \equiv \{1, 2\} \in \mathcal{L}_2$.

Par exemple, l'intuition derrière un directeur $\curvearrowleft$ est : la variable correspondante apparaît seulement dans le sous-terme gauche d'une construction binaire, ou d'une manière équivalente : une substitution pour cette variable devrait être propagée seulement vers la gauche.

Dans ce contexte, pour faire le lien avec le chapitre suivant, les éléments de $\mathcal{L}$ s'appellent les *directeurs* et les listes ordonnées d'éléments de $\mathcal{L}$ s'appellent les *chaînes directrices* et sont utilisées à la place des fonctions de $\mathcal{V}$ dans $\mathcal{L}$. Ceci peut être vu comme l'utilisation d'une variante des indices de de Bruijn [Bru72] au lieu des noms, évitant de fait les problèmes de capture dus aux lieurs. C'est seulement une variante parce que, comme nous le verrons au chapitre suivant, une nouvelle variable introduite par un λ apparaît en dernier (et non en premier) dans les chaînes ; ceci est réminiscent des indices de de Bruijn inversés de Crégut [Cré90].

L'utilisation de listes fait également une différence pour l'effacement, parce que quand une variable est effacée, elle n'a plus aucun directeur correspondant dans les chaînes des sous-termes. Ceci peut être compris de la façon suivante : nous étendons les directeurs à $\mathcal{L} \cup \bot$ et imposons la condition suivante à tous les termes : $\sigma_{f(t_1,\ldots,t_n)}(x) \in \{\varnothing, \bot\} \Rightarrow \forall i.\sigma_{t_i}(x) = \bot$. Alors, en écrivant les directeurs sous forme de listes, nous omettons simplement toutes les occurrences de $\bot$.

REMARQUE III.6.1

Du point de vue de l'implantation, l'utilisation de noms et de fonctions dans les sections

précédentes correspond à une implantation en termes de tableaux, au lieu de listes.

Pour bien illustrer l'utilisation des directeurs généraux, nous développons un peu plus le système du Chapitre II (λ-calcul avec substitutions explicites) dans le cadre de ce chapitre. Nous donnerons davantage de détails dans le Chapitre IV. Nous explicitons tout d'abord l'ES-CRS conditionnel sous-jacent. Les symboles fonctionnels (avec leur arité) sont : λ^1 (abstraction), $@^2$ (application) et Σ^2 (substitution explicite). Les règles de récriture sont les suivantes.

$$
\begin{array}{lrcl}
(b) & @(\lambda([x]Z(x)), Y) & \to & \Sigma([x]Z(x), Y) \\
(v) & \Sigma([x]x, Y) & \to & Y \\
(a_1) & \Sigma([x](@(Z_1(x), Z_2)), Y) & \to & @(\Sigma([x]Z_1(x), Y), Z_2) \\
 & & & \qquad \text{si } x \in \mathsf{fv}(Z_1(x)) \\
(a_2) & \Sigma([x](@(Z_1, Z_2(x))), Y) & \to & @(Z_1, \Sigma([x]Z_2(x), Y)) \\
 & & & \qquad \text{si } x \in \mathsf{fv}(Z_2(x)) \\
(a_3) & \Sigma([x](@(Z_1(x), Z_2(x))), Y) & \to & @(\Sigma([x]Z_1(x), Y), \Sigma([x]Z_2(x), Y)) \\
 & & & \qquad \text{si } x \in \mathsf{fv}(Z_1(x)) \cap \mathsf{fv}(Z_2(x)) \\
(l) & \Sigma([x]\lambda([y]Z(x, y)), Y) & \to & \lambda([y]\Sigma([x]Z(x, y), Y)) \\
 & & & \qquad \text{si } x \in \mathsf{fv}(Z(x, y)) \\
(c) & \Sigma([x]\Sigma([y]Z(y), Y(x)), X) & \to & \Sigma([y]Z(y), \Sigma([x]Y(x), X)) \\
 & & & \qquad \text{si } x \in \mathsf{fv}(Y(x)) \\
(e) & \Sigma([x]Z, Y) & \to & Z
\end{array}
$$

A titre d'exemple, concentrons-nous sur la règle (a_1), et essayons de donner la relation de réduction correspondante d'une façon totalement explicite, comme nous le ferons au chapitre suivant. D'abord, nous reprenons la notation habituelle par souci de lisibilité. La règle (a_1) s'écrit donc comme suit :

$$(a_1) \quad (Z_1(x)\ Z_2)[x/Y] \to Z_1(x)[x/Y]\ Z_2 \qquad \text{si } x \in \mathsf{fv}(Z_1(x))$$

Et avec des directeurs :

$$
\begin{array}{rl}
(a_1') & (((Z_1(x))^{\{Z_1,x:\downarrow\}}\ Z_2^{\{Z_2:\downarrow\}})^{\{Z_1,x:\curvearrowleft;Z_2:\curvearrowright\}}[x/Y^{\{Y:\downarrow\}}])^{\{Z_1,Z_2:\curvearrowleft;Y:\curvearrowright\}} \\
 & \qquad \to (((Z_1(x))^{\{Z_1,x:\downarrow\}}[x/Y^{\{Y:\downarrow\}}])^{\{Z_1:\curvearrowleft;Y:\curvearrowright\}}\ Z_2^{\{Z_2:\downarrow\}})^{\{Z_1,Y:\curvearrowleft;Z_2:\curvearrowright\}}
\end{array}
$$

Notons que la condition $x \in \mathsf{fv}(Z_1(x))$ est réalisée au niveau des directeurs pour x (deux occurrences).

Considérons maintenant un terme annoté $\mathbf{t} = (t, \sigma)$ où $t = (u\ v)[x/w]$ et u, v, w sont des termes (pas des métatermes). Supposons que $\sigma_u(x) \neq \varnothing$ et $\sigma_v(x) = \varnothing$. Le terme $\mathbf{t}$ se récrit alors en utilisant (a_1') (sur les termes annotés) avec la substitution $\varsigma = \{Z_1 \mapsto \underline{\lambda}x.u, Z_2 \mapsto v, Y \mapsto w\}$ en (t', σ') avec $t' = (u[x/w])\ v$ et σ' est comme suit (où r désigne le membre droit de (a_1') et $\alpha \neq x$) :

$$\sigma'_{t'}(\alpha) = \bigcup_{\sigma_{\varsigma(\beta)}(\alpha)\neq\varnothing} \sigma_r(\beta) = \bigcup_{\sigma_u(\alpha)\neq\varnothing} \sigma_r(Z_1) \cup \bigcup_{\sigma_v(\alpha)\neq\varnothing} \sigma_r(Z_2) \cup \bigcup_{\sigma_w(\alpha)\neq\varnothing} \sigma_r(Y)$$
$$= \bigcup_{\sigma_u(\alpha)\neq\varnothing\vee\sigma_w(\alpha)\neq\varnothing} \curvearrowleft \cup \bigcup_{\sigma_v(\alpha)\neq\varnothing} \curvearrowright$$
$$= \begin{cases} - & \text{si } \alpha \notin \mathsf{fv}(u) \wedge \alpha \notin \mathsf{fv}(w) \wedge \alpha \in \mathsf{fv}(v), \\ \curvearrowleft & \text{si } (\alpha \in \mathsf{fv}(u) \vee \alpha \in \mathsf{fv}(w)) \wedge \alpha \notin \mathsf{fv}(v), \\ \curvearrowright & \text{si } (\alpha \notin \mathsf{fv}(u) \wedge \alpha \notin \mathsf{fv}(w)) \wedge \alpha \in \mathsf{fv}(v), \\ \downarrow & \text{si } (\alpha \in \mathsf{fv}(u) \vee \alpha \in \mathsf{fv}(w)) \wedge \alpha \in \mathsf{fv}(v). \end{cases}$$

C'est exactement la réduction que nous retrouverons au chapitre suivant. La bonne nouvelle est qu'elle est ici dérivée de façon "automatique", et que nous pouvons donc la laisser implicite. Pour être totalement convaincant dans le chapitre suivant, nous n'utiliserons cependant pas cette technique et dériverons directement la relation de réduction par une technique plus intuitive. Il est néanmoins heureux que les deux méthodes coïncident.

III.7 Conclusion

Nous avons défini un cadre très général pour la définition de systèmes de récriture avec des conditions du type de celles du Chapitre II. Ce cadre va être spécialisé dans le Chapitre IV, lui donnant une justification plus pragmatique. Il peut néanmoins avoir des applications propres, ce qui reste à explorer. L'intérêt de ce chapitre est d'apporter une meilleure compréhension aux concepts de directeurs et de variables libres, en toute généralité. En particulier, il apparaît clairement pourquoi l'information peut être maintenue localement dans un cadre avec substitutions explicites. Les substitutions explicites sont le bon cadre pour les directeurs, et les directeurs sont la bonne technique d'implantation pour les systèmes avec substitutions explicites, tels que ceux que nous considérons (cela dépend évidemment des motivations).

Chapitre IV

Chaînes directrices pour le λ-calcul

Nous utilisons les résultats du chapitre précédent pour définir un λ-calcul avec substitutions explicites sans noms ni indices, basé sur une notion généralisée de chaînes directrices. Les termes sont annotés avec de l'information, les directeurs, qui indiquent comment les substitutions doivent être propagées. Nous présentons d'abord un calcul où nous pouvons simuler des étapes arbitraires de β-réduction, et simplifions ensuite les règles pour modéliser l'évaluation des programmes fonctionnels (réduction en forme normale de tête faible). Nous montrons également que nous pouvons définir la stratégie de réduction close du Chapitre II. C'est une stratégie faible qui, contrairement aux stratégies faibles standard, permet à certaines réductions d'avoir lieu à l'intérieur des λ-abstractions offrant ainsi davantage de partage de calcul. Ce chapitre fournit un modèle d'implantation efficace pour cette stratégie, en utilisant la technologie développée au Chapitre III. Nos résultats expérimentaux confirment que, pour de grands termes construits à partir de combinateurs, nos stratégies d'évaluation faibles surpassent les évaluateurs standard. De plus, nous dérivons deux machines abstraites pour la réduction forte qui héritent de l'efficacité des évaluateurs faibles.

IV.1 Introduction

Dans le λ-calcul, l'opération de substitution utilisée dans la règle de β-réduction est définie en-dehors du système : c'est une méta-opération (voir [Bar84]). En revanche, les *calculs avec substitutions explicites* définissent la substitution avec des règles de réduction au même niveau que la β-réduction. Au cours des dernières années, une multitude de calculs avec substitutions explicites ont été proposés, en commençant par les travaux de de Bruijn [Bru78] et le $\lambda\sigma$-calcul [ACCL91]. Bien qu'il y ait beaucoup d'applications différentes pour de tels calculs, un des avantages principaux que nous voyons en décrivant le processus de propagation de la substitution au même niveau que la β-réduction est qu'il nous permet de contrôler finement le processus de substitution en vue de l'implantation.

Il y a différentes notations pour la substitution, ou plus précisément, pour les variables dans le λ-calcul. Les calculs avec substitutions explicites peuvent être classés en deux catégories :

- *nommés*, quand les variables sont dénotées par des noms comme $x, y, \ldots$;
- *sans noms* (*unnamed*, en anglais), par exemple quand des entiers (également appelés *indices*) sont utilisés.

Pendant le processus de propagation de la substitution, il peut être nécessaire d'avoir recours à l'α-conversion (c'est-à-dire de renommer certaines variables liées) pour éviter la capture de variable. Les calculs avec substitutions explicites sans noms ont donc souvent la préférence pour l'implantation (bien qu'il y ait quelques exceptions, voir par exemple [Ros96, FM99]). La notation de de Bruijn [Bru72] est sans doute devenue la syntaxe sans noms standard pour les calculs avec substitutions explicites. Le but de ce chapitre est de définir une notation alternative pour les calculs avec substitutions explicites sans noms, basée sur des *chaînes directrices*.

Les chaînes directrices ont été introduites par Kennaway et Sleep [KS88] pour la réduction de combinateurs, qui, traduite en λ-calcul, donne un système où aucune réduction ne peut être effectuée sous les abstractions (voir [ÇH98]). Dans ce chapitre, nous généralisons l'approche en définissant un calcul avec substitutions explicites et chaînes directrices dans lequel n'importe quelle stratégie de réduction du λ-calcul peut être simulée, et nous explorons les propriétés de ce calcul général avec chaînes directrices en tant que système de récriture et également en tant que moyen d'exprimer des stratégies de réduction (faibles et fortes) efficaces pour le λ-calcul. Nous considérons que cela est important pour plusieurs raisons :

- Les chaînes directrices offrent une alternative à la notation de de Bruijn [Bru72] pour les calculs sans noms. Cependant, de même qu'avec la notation de de Bruijn, la syntaxe n'est pas aussi lisible que la version nommée correspondante. Nous montrerons que la notation générale peut être simplifiée dans certains cas, par exemple, la réduction close s'avère être une restriction naturelle menant à un système de récriture très simple pour la réduction faible.
- Les chaînes directrices sont une notation naturelle pour les substitutions explicites d'un point de vue opérationnel : les termes sont annotés pour indiquer ce qu'ils devraient faire avec une substitution. Les substitutions sont seulement propagées aux endroits où elles sont nécessaires, ainsi ces calculs préservent aisément la normalisation forte (c'est-à-dire si un λ-terme est fortement normalisable, sa compilation l'est aussi). D'autre calculs préservant la normalisation forte ont été présentés dans [LRD95, DG01] (voir [Mel95, BG99] pour des contre-exemples dans $\lambda\sigma$).

– Nous fournissons une généralisation des chaînes directrices introduites par Kennaway et Sleep [KS88] pour la réduction de combinateurs. Avec nos chaînes directrices généralisées nous pouvons simuler des β-réductions arbitraires.

Nous voyons ainsi les calculs présentés dans ce chapitre comme une syntaxe alternative pour les substitutions explicites et comme une base pour des implantations plus efficaces du λ-calcul. Nous présentons trois calculs basés sur des chaînes directrices. Le premier, que nous appelons λ_o, est un système général où n'importe quelle β-réduction du λ-calcul peut être simulée. D'un point de vue théorique, λ_o a les propriétés désirées (il est confluent, préserve la normalisation forte, simule entièrement le λ-calcul). Cependant, d'une perspective d'implantation, sa généralité est un inconvénient plutôt qu'un avantage. Dans les deux autres calculs, que nous appelons λ_l et λ_c, la réduction est restreinte de sorte que seules certaines stratégies d'évaluation (dont au moins les stratégies standard et certaines stratégies efficaces) puissent être simulées. Dans ce sens, λ_l et λ_c sont faibles, mais pas aussi faibles que les calculs faibles standard. Il est bien connu que les calculs faibles avec substitutions explicites standard évitent l'α-conversion en interdisant la réduction sous les abstractions et la propagation des substitutions sous les abstractions (voir par exemple [CHL96]). En revanche, nos calculs faibles autorisent certaines réductions et certaines propagations de substitutions sous les abstractions. De cette façon davantage de réductions peuvent être partagées. De plus, nous pouvons utiliser l'information explicite fournie par les directeurs pour éviter de copier une substitution qui contient une variable libre, évitant la duplication de rédex potentiels.

Nous avons implanté une famille de machines abstraites pour la réduction faible et forte basées sur les calculs avec chaînes directrices, et le banc d'essai (donné en Section IV.9) indique que le niveau de partage obtenu est proche de la réduction optimale [Lam90, GAL92, AGN96] avec considérablement moins de surcoût administratif dans beaucoup de cas. Les applications immédiates de ce travail incluent, d'une part des évaluateurs pour le λ-calcul ou les langages fonctionnels (où la réduction faible est nécessaire), et d'autre part, l'évaluation partielle (également appelée spécialisation de programme) et les assistants de preuve (où la réduction forte est nécessaire).

ÉTAT DE L'ART. Ces travaux sont clairement liés aux travaux généraux sur les substitutions explicites, à partir du calcul séminal $\lambda C\xi\phi$ de de Bruijn [Bru78] (voir [BBLRD96] pour une présentation moderne) et du $\lambda\sigma$-calcul [ACCL91]. Cependant, ils sont davantage dans la lignée de l'utilisation des substitutions explicites pour contrôler le processus de substitution dans les implantations du λ-calcul [AFM$^+$95, Yos94, HMP98, Lan98, Nad02]. Nos calculs sont plus proche de $\lambda\upsilon$ [Les94, LRD95] que de $\lambda\sigma$ [ACCL91] au sens où nous n'avons pas de construction syntaxique pour la concaténation (parfois désignée également sous le nom de composition) de substitutions indépendantes. Les travaux de Nadathur [Nad99, Nad02] ont également des considérations d'efficacité et ont quelques points communs avec le nôtre (bien que dans un cadre tout à fait différent) : par exemple, une variante de son calcul a des conditions de clôture de certains termes.

L'efficacité et le partage dans le λ-calcul ont été des sujets importants ces vingt dernières années. Il y a dans la littérature un éventail de mécanismes utilisés pour le partage : les environnements [Yos94], les graphes de partage (*sharing graphs*) [AG98], les calculs avec adresses expli-

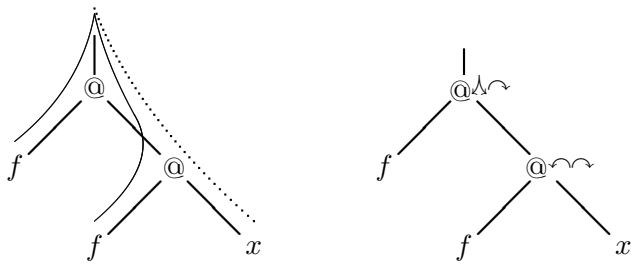
cites [BRL96, Lan98]. Nous utilisons les chaînes directrices et le mécanisme de la substitution explicite elle-même afin de réaliser le partage, autrement dit, nous n'utilisons aucune machinerie externe. Tandis que le partage optimal [Lév80] signifie le nombre optimal de β-réductions, son implantation se fonde sur les graphes de partage [Lam90, GAL92, AGN96, AG98] dans lesquels une multitude de règles coûteuses sont nécessaire (voir [LM96] par exemple). Par conséquent nous donnons ici à l'efficacité une signification plutôt algorithmique, ou plus pragmatiquement, nous compterons le nombre total d'étapes de réduction nécessaires pour atteindre une forme normale, à conditions que ces étapes puissent être vues comme élémentaires.

Aperçu. Le reste de ce chapitre est structuré comme suit. Dans la section suivante, nous donnons quelques idées fondamentales et définissons la syntaxe des chaînes directrices. Dans la Section IV.3, nous décrivons un calcul général où nous pouvons simuler une étape de β-réduction arbitraire. La Section IV.4 présente le calcul ouvert local simplifié, et le système de réduction close. Un système de type pour ces calculs est présenté dans la Section IV.6. Nous utilisons alors ces calculs pour définir plusieurs stratégies : faible (Section IV.7) et forte (Section IV.8), que nous comparons expérimentalement (Section IV.9). Nous concluons le chapitre à la Section IV.10.

IV.2 Chaînes directrices

IV.2.1 Rappels

Nous rappelons brièvement les idées fondamentales des chaînes directrices [KS88]. A titre de motivation, considérons un terme avec deux variables libres f, x et des substitutions pour chacune : $((f\,(f\,x))[F/f])[X/x]$. La meilleure manière d'effectuer ces substitutions est de les propager *seulement* aux endroits dans l'arbre syntaxique où elles sont requises. La Figure IV.1(a) montre les *chemins* que les substitutions devraient suivre dans l'arbre, où le trait plein correspond à la substitution pour f, et le trait pointillé à celle pour x.



(a) Chemins (b) Terme annoté

Fig. IV.1 – Chemins de substitution et chaînes directrices

Une façon naturelle de guider les substitutions à leur destination correcte est donnée à la Figure IV.1(b) par les chaînes directrices, qui annotent chaque nœud du graphe avec une

information indiquant où la substitution doit aller (sur les nœuds d'application, le premier symbole de type flèche, appelé directeur, correspond à f et le second à x). Quand la substitution pour f passe la racine de ce terme, une copie de F est envoyée aux deux sous-termes, et le directeur $\swarrow\!\!\!\searrow$ est effacé. La deuxième substitution peut alors passer la racine, où elle est dirigée uniquement vers la branche droite par le directeur $\curvearrowright$. Notons que les substitutions sont copiées seulement quand c'est nécessaire : s'il y a juste une occurrence d'une variable dans un terme, alors aucune duplication n'est exécutée.

Cette idée simple fonctionne bien quand la substitution est close (ne contient pas de variables libres). Si ce n'est pas le cas, nous devons ajouter les directeurs additionnels pour chaque variable libre dans la substitution à chaque passage d'une substitution ouverte par un nœud donné.

Nous finissons cette section en rappelant brièvement l'analogie entre les chaînes directrices et la réduction de combinateurs, comme décrite dans [KS88] et rappelée à la Section I.3.3. Les règles de réduction pour les combinateurs S, B, C, K sont les suivantes :

$$\begin{array}{lcl} \mathsf{S}\,x\,y\,z & \to & x\,z\,(y\,z) \\ \mathsf{B}\,x\,y\,z & \to & x\,(y\,z) \\ \mathsf{C}\,x\,y\,z & \to & x\,z\,y \\ \mathsf{K}\,x\,y & \to & x \end{array}$$

où $\mathsf{S}\,x\,y$ prend un argument et le dirige à la fois vers x et y ; $\mathsf{B}\,x\,y$ prend un argument et le dirige seulement vers y ; $\mathsf{C}\,x\,y$ prend un argument et le dirige seulement vers x ; et finalement $\mathsf{K}\,x$ prend un argument et le jette. Nous pouvons donc annoter l'application $x\,y$ avec les combinateurs, qui correspondent exactement aux directeurs : S est $\swarrow\!\!\!\searrow$, B est $\curvearrowright$, C est $\curvearrowleft$ et K est $-$ (voir ci-dessous).

IV.2.2 Syntaxe

Nous présentons maintenant plus formellement la syntaxe des termes annotés. Cette syntaxe est commune aux différents systèmes de récriture qui seront décrits plus tard.

Définition IV.2.1 (*λ-calcul avec chaînes directrices*)

Nous définissons quatre catégories syntaxiques :

Directeurs : Nous utilisons cinq symboles spéciaux, appelés directeurs, que nous désignerons par exemple par α, γ, δ :

1. $\curvearrowright$ indique que la substitution doit être propagée seulement dans la branche droite d'une construction binaire (application ou substitution, voir ci-dessous).
2. $\curvearrowleft$ indique que la substitution doit être propagée seulement dans la branche gauche d'une construction binaire.
3. $\swarrow\!\!\!\searrow$ indique que la substitution doit être propagée dans les deux branches d'une construction binaire.
4. $\downarrow$ indique que la substitution doit traverser une construction unaire (abstraction et variables, voir ci-dessous).

5. $-$ indique que la substitution doit être jetée (quand la variable en question n'apparaît pas dans un terme).

Chaînes : Une chaîne directrice ou chaîne de directeurs est soit vide, dénotée par ϵ, soit construite à partir des symboles ci-dessus (c'est-à-dire qu'elle est de la forme $\alpha_1\alpha_2\dots\alpha_n$ où les α_i sont des directeurs). Nous utilisons les lettres grecques telles que ρ, σ,... pour désigner des chaînes. La longueur d'une chaîne σ est notée $|\sigma|$. Si α est un directeur, alors α^n dénote une chaîne de α de longueur n. Si σ est une chaîne directrice de longueur n et $1 \leq i \leq j \leq n$, σ_i dénote le $i^{\text{ème}}$ directeur de σ et $\sigma_{\setminus i} = \sigma_1\dots\sigma_{i-1}\sigma_{i+1}\dots\sigma_n$ est σ où le $i^{\text{ème}}$ directeur a été enlevé. $\sigma_{i..j} = \sigma_i\dots\sigma_j$ est notre notation pour les sous-chaînes. $|\sigma|_l$ dénote le nombre de $\curvearrowleft$ et de $\swarrow$ apparaissant dans σ, $|\sigma|_r$ le nombre de $\curvearrowright$ et de $\searrow$, et $|\sigma|_+$ le nombre de directeurs qui *ne sont pas* le directeur $-$.

Prétermes : Si σ désigne les chaînes, k les entiers naturels, alors les prétermes $\mathbf{t}, \mathbf{u}$ sont définis par la grammaire suivante :

$$\mathbf{t}, \mathbf{u} ::= \square^\sigma \mid (\lambda\mathbf{t})^\sigma \mid (\mathbf{t}\ \mathbf{u})^\sigma \mid (\mathbf{t}[k/\mathbf{u}])^\sigma$$

Les prétermes sont notés $\mathbf{t}$, ou t^σ si l'on veut expliciter la chaîne directrice σ.

Termes : Les termes sont les prétermes qui satisfont récursivement les conditions de bonne formation suivantes, où $\mathcal{U} = (\downarrow |-)^*$ et $\mathcal{B} = (\curvearrowleft \mid \swarrow \mid \curvearrowright |-)^*$:

Nom	Terme	Contrainte
Variable	$\square^\sigma$	$\sigma \in \mathcal{U}, \lvert\sigma\rvert_+ = 1$
Abstraction	$(\lambda t^\rho)^\sigma$	$\sigma \in \mathcal{U},\ \lvert\rho\rvert = \lvert\sigma\rvert_+ + 1$
Application	$(t^\rho\, u^\nu)^\sigma$	$\sigma \in \mathcal{B},\ \lvert\rho\rvert = \lvert\sigma\rvert_l,\ \lvert\nu\rvert = \lvert\sigma\rvert_r$
Substitution	$(t^\rho[k/u^\nu])^\sigma$	$\sigma \in \mathcal{B},\ \lvert\rho\rvert = \lvert\sigma\rvert_l + 1,\ \lvert\nu\rvert = \lvert\sigma\rvert_r,\ 1 \leq k \leq \lvert\rho\rvert$

Remarque IV.2.2

Nous utilisons la même convention pour les termes que pour les prétermes, notant $\mathbf{t}$ ou t^σ selon que l'on veut ou non mentionner spécifiquement la chaîne directrice d'un terme annoté, comme dans la définition ci-dessus. Pour résumer les conventions de notation de ce chapitre, les lettres minuscules en caractères gras indiquent des prétermes en général (par conséquent aussi des termes bien formés), alors que les lettres minuscules en police normale indiquent des prétermes (ou des termes) dont la chaîne directrice à la racine a été supprimée, et qui ont donc besoin explicitement d'une chaîne directrice pour former un préterme ou un terme. Les lettres majuscules sont réservées pour des termes du λ-calcul habituel. Comme avec la plupart des λ-calculs, nous adopterons un certain nombre de conventions syntaxiques : nous omettrons les parenthèses à chaque fois que nous le pouvons, et omettrons la chaîne vide ϵ à moins qu'elle soit essentielle.

Passons brièvement en revue nos différentes constructions :

- $\square^\sigma$ représente les variables (c'est simplement un marqueur de position),
- $(\lambda \mathbf{t})^\sigma$ est notre λ-abstraction,
- $(\mathbf{t}\ \mathbf{u})^\sigma$ est notre application,
- finalement $(\mathbf{t}[k/\mathbf{u}])^\sigma$ est notre notation pour les substitutions explicites, signifiant que la variable correspondant au $k^{\text{ème}}$ directeur dans la chaîne directrice de $\mathbf{t}$ doit être remplacée par $\mathbf{u}$. Nous noterons souvent $(\mathbf{t}[\mathbf{u}])^\sigma$ au lieu de $(\mathbf{t}[1/\mathbf{u}])^\sigma$ quand la substitution lie la première variable de σ.

Le nom de la variable est sans intérêt puisque les chaînes directrices donnent le chemin que la substitution doit suivre à travers le terme pour arriver au bon endroit : nous avons seulement besoin d'un marqueur de position.

Contrairement à d'autres syntaxes avec substitutions explicites, la nôtre a des informations explicites sur la copie ($\curlywedge$) et l'effacement ($-$), comme dans le Chapitre II. Ceci est inspiré par des calculs pour la logique linéaire, et nous permettra un contrôle plus fin des substitutions : nous pouvons réduire un sous-terme davantage en rencontrant un $\curlywedge$, tirant de ce fait profit du mécanisme de substitution explicite pour partager certaines réductions. Il y a une présentation alternative qui combine le directeur $-$ avec l'abstraction, en utilisant la notation $(\lambda^- \mathbf{t})^\sigma$ pour indiquer que la variable liée n'apparaît pas dans le terme $\mathbf{t}$. La syntaxe résultante est plus simple et permet d'effacer les termes dès que possible. Cependant, elle ne permet pas de définir la β-réduction en toute généralité. Nous discuterons à nouveau ce choix dans la Section IV.4.

IV.2.3 Compilation et décompilation

Nous utilisons les λ-termes avec chaînes directrices comme langage intermédiaire. Il nous faut donc fournir une fonction pour compiler les λ-termes usuels dans cette syntaxe et une autre pour les décompiler. Comme d'habitude, nous considérons les termes du λ-calcul modulo α-conversion (renommage des variables liées).

La définition suivante de la compilation du λ-calcul habituel dans la syntaxe avec chaînes directrices indique précisément comment les chaînes et les termes sont construits. Nous utilisons une liste ordonnée auxiliaire $[x_1, \ldots, x_n]$ dans la fonction de compilation pour garder la trace des noms de variable correspondant à chaque directeur dans les chaînes. Chaque étape de la fonction de compilation descend un nœud dans l'arbre de syntaxe du terme, et calcule la chaîne correspondante en utilisant les fonctions auxiliaires ξ et θ. Nous dénotons par $\mathsf{fv}(M)$ l'ensemble des variables libres du λ-terme M. Nous utilisons les notations standard pour la liste vide et les opérations de construction et de concaténation ($[\,]$, $x :: \ell$ et $\ell \cdot \ell'$ respectivement) et nous abrégeons $x_1 :: \ldots :: x_n :: [\,]$ en $[x_1, \ldots, x_n]$ et même parfois en $\vec{x}$. Nous dénotons par ℓ_i le $i^{\text{ème}}$ élément d'une liste ℓ.

DÉFINITION IV.2.3 (*Compilation*)

Soit M un λ-terme avec $\mathsf{fv}(M) \subseteq \{x_1, \ldots, x_n\}$, sa compilation $[\![M]\!]_{\vec{x}}$ est définie comme suit :

$$\begin{array}{lcll} [\![x]\!]_{\vec{x}} & = & \Box^{\sigma} & \text{où } ([x], \sigma) = \xi_x(\vec{x}) \\ [\![\lambda x.M]\!]_{\vec{x}} & = & (\lambda\, [\![M]\!]_{\ell\cdot[x]})^{\sigma} & \text{où } (\ell, \sigma) = \xi_{(\lambda x.M)}(\vec{x}) \\ [\![M\ N]\!]_{\vec{x}} & = & ([\![M]\!]_{\ell}\ [\![N]\!]_{\ell'})^{\sigma} & \text{où } (\ell, \ell', \sigma) = \theta_{M,N}(\vec{x}) \end{array}$$

$$\xi_M([\,]) = ([\,], \epsilon)$$

$$\xi_M(x :: \ell) = \left\{ \begin{array}{ll} (x :: \ell', \downarrow \sigma) & \text{if } x \in \mathsf{fv}(M) \\ (\ell', -\sigma) & \text{if } x \notin \mathsf{fv}(M) \end{array} \right\} \text{ où } (\ell', \sigma) = \xi_M(\ell)$$

$$\theta_{M,N}([\,]) = ([\,], [\,], \epsilon)$$

$$\theta_{M,N}(x :: \ell) = \left\{ \begin{array}{ll} (x :: \ell', \ell'', \curvearrowleft \sigma) & \text{si } x \in \mathsf{fv}(M) \setminus \mathsf{fv}(N) \\ (\ell', x :: \ell'', \curvearrowright \sigma) & \text{si } x \in \mathsf{fv}(N) \setminus \mathsf{fv}(M) \\ (x :: \ell', x :: \ell'', \downdownarrows \sigma) & \text{si } x \in \mathsf{fv}(M) \cap \mathsf{fv}(N) \\ (\ell', \ell'', -\sigma) & \text{si } x \notin \mathsf{fv}(M) \cup \mathsf{fv}(N) \end{array} \right\} \text{ où } (\ell', \ell'', \sigma) = \theta_{M,N}(\ell)$$

Quand il n'y pas d'ambiguïté, nous utilisons la notation $[\![M]\!]_{\vec{x}}$ en supposant implicitement que $\mathsf{fv}(M) \subseteq \vec{x}$, et nous écrivons $[\![M]\!]$ quand la liste est vide.

REMARQUE IV.2.4 (*Ordre des directeurs*)

Dans une abstraction $(\lambda t^{\rho})^{\sigma}$, le *dernier* directeur dans la chaîne ρ correspond à la variable liée. Ceci est réminiscent des *indices de de Bruijn inversés* de Crégut [Cré90].

EXEMPLE IV.2.5

Nous montrons la compilation de quelques λ-termes :

$$\begin{array}{lcll} \mathsf{I} & = & [\![\lambda x.x]\!] & = & (\lambda\Box^{\downarrow})^{\epsilon} \\ \mathsf{K} & = & [\![\lambda x.\lambda y.x]\!] & = & (\lambda(\lambda\Box^{\downarrow -)^{\downarrow}})^{\epsilon} \\ \mathsf{S} & = & [\![\lambda x.\lambda y.\lambda z.(xz)(yz)]\!] & = & (\lambda(\lambda(\lambda((\Box^{\downarrow}\Box^{\downarrow})\curvearrowleft(\Box^{\downarrow}\Box^{\downarrow})\curvearrowleft)\curvearrowleft\downdownarrows)^{\downarrow\downarrow})^{\downarrow})^{\epsilon} \\ \mathsf{2} & = & [\![\lambda f.\lambda x.f(fx)]\!] & = & (\lambda(\lambda(\Box^{\downarrow}(\Box^{\downarrow}\Box^{\downarrow})\curvearrowleft)\downdownarrows\curvearrowright)^{\downarrow})^{\epsilon} \end{array}$$

Pour prouver que le résultat de la compilation est un terme bien formé, nous avons besoin de deux lemmes auxiliaires.

LEMME IV.2.6

Si $(M\ N)$ est une application avec $\mathsf{fv}(M\ N) \subseteq \{x_1, \ldots, x_n\}$, alors
$\theta_{M,N}([x_1, \ldots, x_n]) = (\ell_1, \ell_2, \sigma)$ où $\ell_1 = \mathsf{fv}(M)$, $\ell_2 = \mathsf{fv}(N)$, et $|\sigma| = n$.

Preuve. Induction immédiate sur n. □

LEMME IV.2.7 (*Longueur des chaînes*)

Soit M un λ-terme et $\mathsf{fv}(M) \subseteq \{x_1, \ldots, x_n\}$, alors :

$$[\![M]\!]_{x_1,\ldots,x_n} = u^{\sigma} \text{ où } |\sigma| = n.$$

En particulier, si M est clos alors $[\![M]\!]$ a une chaîne directrice vide (ϵ).

Preuve. Par induction sur n. Pour $n = 0$, M est soit une abstraction, auquel cas le terme compilé a la chaîne directrice $\downarrow^0 = \epsilon$ comme demandé, soit une application et $[\![M\ N]\!] = ([\![M]\!]\ [\![N]\!])^\epsilon$ puisque $\theta_{M,N}([\,]) = ([\,], [\,], \epsilon)$ par définition. Pour $n > 0$, nous raisonnons par cas selon M. Les cas variable et abstraction sont triviaux. Le cas intéressant est l'application. Dans ce cas, la propriété est une conséquence directe du Lemme IV.2.6. □

PROPOSITION IV.2.8 (*Consistence de la compilation*)

Si M est un λ-terme avec $\mathsf{fv}(M) \subseteq \vec{x}$ alors $[\![M]\!]_{\vec{x}}$ est un terme bien formé.

Preuve. Par induction sur la structure des λ-termes. Si M est une variable alors le résultat suit trivialement. Si M est une abstraction, alors le résultat suit par induction. Si M est une application, c'est une conséquence du Lemme IV.2.6, de l'hypothèse d'induction et de la construction de σ dans la définition de θ. □

Comme nous préférons penser ce calcul comme un langage intermédiaire, nous donnons aussi une fonction de décompilation, qui remet simplement les noms dans le terme.

DÉFINITION IV.2.9 (*Décompilation*)

Soit $\mathbf{t} = t^\sigma$ un terme où $|\sigma| = n$, et soient $x_1, \ldots, x_n$ n variables fraîches. Nous définissons la décompilation de $\mathbf{t}$ comme $(\!|\mathbf{t}|\!)_{[x_1,\ldots,x_n]}$, où la fonction de décompilation (qui utilise une liste auxiliaire $\vec{M}$ de λ-termes) est définie comme suit :

$$\begin{array}{lcll}
(\!|\square^\sigma|\!)_{\vec{M}} & = & M & \text{où } [M] = \kappa_\sigma(\vec{M}) \\
(\!|(\lambda\mathbf{t})^\sigma|\!)_{\vec{M}} & = & \lambda x.(\!|\mathbf{t}|\!)_{\kappa_\sigma(\vec{M}\cdot[x])} & \text{où } x \text{ est frais} \\
(\!|(\mathbf{t}\ \mathbf{u})^\sigma|\!)_{\vec{M}} & = & (\!|\mathbf{t}|\!)_\ell\ (\!|\mathbf{u}|\!)_{\ell'} & \text{où } (\ell, \ell') = \gamma_\sigma(\vec{M}) \\
(\!|(\mathbf{t}[k/\mathbf{u}])^\sigma|\!)_{\vec{M}} & = & (\!|\mathbf{t}|\!)_{[\ell_1,\ldots,\ell_{k-1},(\!|\mathbf{u}|\!)_{\ell'},\ell_k,\ldots,\ell_m]} & \text{où } (\ell, \ell') = \gamma_\sigma(\vec{M})
\end{array}$$

$$\begin{array}{lcl}
\kappa_\epsilon([\,]) & = & [\,] \\
\kappa_{\downarrow\sigma}(M :: \ell) & = & M :: \kappa_\sigma(\ell) \\
\kappa_{-\sigma}(M :: \ell) & = & \kappa_\sigma(\ell)
\end{array}$$

$$\left.\begin{array}{lcl}
\gamma_\epsilon([\,]) & = & ([\,], [\,]) \\
\gamma_{\curvearrowright\sigma}(M :: \ell) & = & (M :: \ell', \ell'') \\
\gamma_{\curvearrowright\sigma}(M :: \ell) & = & (\ell', M :: \ell'') \\
\gamma_{\curlywedge\sigma}(M :: \ell) & = & (M :: \ell', M :: \ell'') \\
\gamma_{-\sigma}(M :: \ell) & = & (\ell', \ell'')
\end{array}\right\} \text{où } (\ell', \ell'') = \gamma_\sigma(\ell)$$

On peut remarquer que dans le cas des variables $\square^\sigma$, on a nécessairement $|\sigma|_+ = 1$ puisque le terme est bien formé, d'où $\kappa_\sigma(\vec{M})$ est un singleton. Notons aussi que la liste auxiliaire $\vec{M}$ peut contenir des λ-termes arbitraires et pas forcément seulement des variables comme dans la compilation. Cela permet de mener les substitutions à destination de façon directe et élégante : il suffit de les mettre dans la liste, et les termes seront guidés aux bons endroits, grâce aux directeurs. Plus précisément :

$$(\!|\mathbf{t}|\!)_{[x_1,\ldots,x_n]}\{x_i := M_i\} = (\!|\mathbf{t}|\!)_{[x_1,\ldots,x_{i-1},M_i,x_{i+1},\ldots,x_n]}.$$

Exemple IV.2.10

Nous donnons deux petits exemples de la procédure de décompilation :

$$\begin{array}{lcl} (\!|(\lambda\Box^{\downarrow})^{\epsilon}|\!) & = & \lambda x.(\!|\Box^{\downarrow}|\!)_{[x]} = \lambda x.x \\ (\!|(\lambda(\lambda(\Box^{\downarrow}(\Box^{\downarrow}\,\Box^{\downarrow})\curvearrowright\curvearrowright)\curvearrowleft\curvearrowright)^{\downarrow})^{\epsilon}|\!) & = & \lambda x.\lambda y.(\!|(\Box^{\downarrow}(\Box^{\downarrow}\,\Box^{\downarrow})\curvearrowright\curvearrowright)\curvearrowleft\curvearrowright|\!)_{[x,y]} \\ & = & \lambda x.\lambda y.x\,(x\,y) \end{array}$$

Pour prouver que la fonction de décompilation est bien définie, nous avons besoin du lemme suivant :

Lemme IV.2.11

Si $|\sigma| = n$ et $M_1, \ldots, M_n$ sont des λ-termes, alors
- $|\kappa_\sigma([M_1, \ldots, M_n])| = |\sigma|_+$ si $\sigma \in \mathcal{U}$;
- $\gamma_\sigma([M_1, \ldots, M_n]) = (\ell, \ell')$ où $|\ell| = |\sigma|_l$ et $|\ell'| = |\sigma|_r$.

Preuve. Induction immédiate sur n. □

Lemme IV.2.12

Pendant la décompilation d'un terme bien formé, la procédure de décompilation est toujours appelée sous la forme $(\!|t^{\rho}|\!)_{[M_1,\ldots,M_n]}$ avec $|\rho| = n$.

Preuve. Par induction et le Lemme IV.2.11. Chaque étape de la procédure ajuste la longueur de la liste auxiliaire, conformément aux contraintes de bonne formation des termes (voir les Définitions IV.2.1 et IV.2.9). □

Proposition IV.2.13 (*Consistence de la décompilation*)

Si t^{σ} est un terme bien formé où $|\sigma| = n$ et $x_1, \ldots, x_n$ sont n variables fraîches alors $(\!|t^{\sigma}|\!)_{[x_1,\ldots,x_n]}$ est un λ-terme.

Preuve. Nous montrons la propriété plus générale :

Si t^{σ} est un terme bien formé, $|\sigma| = n$ et $M_1, \ldots, M_n$ sont des λ-termes, alors $(\!|t^{\sigma}|\!)_{[M_1,\ldots,M_n]}$ est bien formé.

La preuve est par induction sur t, en utilisant le Lemme IV.2.12. Le cas des variables $\Box^{\sigma}$ est trivial puisque $|\sigma|_+ = 1$ par bonne formation. Pour une abstraction, une application ou une substitution, le résultat suit directement par induction et le Lemme IV.2.12. □

Les fonctions de compilation et de décompilation sont inverses l'une de l'autre modulo α-conversion. Pour le montrer, nous utilisons le lemme auxiliaire suivant :

Lemme IV.2.14

- $\xi_M([x_1, \ldots, x_n]) = (\ell, \sigma)$ implique $\kappa_\sigma([x_1, \ldots, x_n]) = \ell$;
- $\theta_{M,N}([x_1, \ldots, x_n]) = (\ell_1, \ell_2, \sigma)$ implique $\gamma_\sigma([x_1, \ldots, x_n]) = (\ell_1, \ell_2)$.

Preuve. Induction immédiate sur n. □

PROPOSITION IV.2.15 (*Inverses*)

Si M est un λ-terme avec $\mathsf{fv}(M) \subseteq \vec{x}$, alors $(\!|[\![M]\!]_{\vec{x}}|\!)_{\vec{x}} =_\alpha M$. En particulier, si M est un λ-terme clos, $(\!|[\![M]\!]|\!) =_\alpha M$.

Preuve. Par induction sur M.

- Variable : $(\!|[\![x]\!]_{\vec{x}}|\!)_{\vec{x}} = x$ comme demandé.
- Abstraction : $(\!|[\![(\lambda x.M)]\!]_{\vec{x}}|\!)_{\vec{x}} = (\!|(\lambda [\![M]\!]_{\ell\cdot[x]})^\sigma|\!)_{\vec{x}} = \lambda y.(\!|[\![M]\!]_{\ell\cdot[x]}|\!)_{\ell\cdot[y]}$, en utilisant le Lemme IV.2.14, où $(\ell, \sigma) = \xi_{\lambda x.M}(\vec{x})$. Par induction, ceci est α-équivalent à $\lambda x.M$.
- Application : $(\!|[\![M\,N]\!]_{\vec{x}}|\!)_{\vec{x}} = (\!|([\![M]\!]_{\ell'}\,[\![N]\!]_{\ell''})^\sigma|\!)_{\vec{x}}$, où $(\ell', \ell'', \sigma) = \theta_{M,N}(\vec{x})$. Par le Lemme IV.2.14 et la définition de la décompilation : $(\!|([\![M]\!]_{\ell'}\,[\![N]\!]_{\ell''})^\sigma|\!)_{\vec{x}} = ((\!|[\![M]\!]_{\ell'}|\!)_{\ell'}\,(\!|[\![N]\!]_{\ell''}|\!)_{\ell''})$. Le résultat suit alors directement par induction.

□

IV.3 CALCUL OUVERT

Nous allons maintenant donner les règles de réduction permettant de simuler complètement le λ-calcul. Ce calcul est appelé *ouvert* (λ_o) par opposition au calcul pour la réduction close (λ_c) défini en Section IV.4.3, qui est plus simple mais qui ne simule pas complètement la β-réduction.

IV.3.1 RÈGLE *Beta*

Nous avons besoin d'une règle *Beta* pour éliminer les β-rédex et introduire une substitution explicite. Dans un terme compilé $(\lambda t^\nu)^\rho$, la variable liée par l'abstraction est déterminée par le *dernier* directeur de ν (voir la Remarque IV.2.4), et $|\nu| = |\rho|_+ + 1$ d'après les contraintes de la Définition IV.2.1. Puisque ρ peut contenir des directeurs d'effacement ($-$) qu'il faut préserver, nous les remontons au niveau de la chaîne directrice de la substitution :

$$\textit{Beta} \qquad ((\lambda \mathbf{t})^\rho\ \mathbf{u})^\sigma \rightsquigarrow_o (\mathbf{t}[|\rho|_+ + 1 \,/\, \mathbf{u}])^\tau \qquad \text{où } \tau = \psi_b(\sigma, \rho)$$

avec :

$$\begin{array}{rcl}
\psi_b(\quad \epsilon, \quad \epsilon) & = & \epsilon \\
\psi_b(\curvearrowright \sigma, \quad \rho) & = & \curvearrowright \psi_b(\sigma, \rho) \\
\psi_b(\curvearrowleft \sigma, \downarrow \rho) & = & \curvearrowleft \psi_b(\sigma, \rho) \\
\psi_b(\swarrow \sigma, \downarrow \rho) & = & \swarrow \psi_b(\sigma, \rho) \\
\psi_b(- \sigma, \quad \rho) & = & - \psi_b(\sigma, \rho) \\
\psi_b(\curvearrowleft \sigma, - \rho) & = & - \psi_b(\sigma, \rho) \\
\psi_b(\swarrow \sigma, - \rho) & = & \curvearrowright \psi_b(\sigma, \rho)
\end{array}$$

REMARQUE IV.3.1

Si la fonction est close (autrement dit, si elle a une chaîne directrice vide), alors la substitution lie la première (et seule) variable de $\mathbf{t}$, et nous avons simplement :

$$((\lambda \mathbf{t})^\epsilon\ \mathbf{u})^\sigma \rightsquigarrow_o (\mathbf{t}[\mathbf{u}])^\sigma.$$

Nous fondant sur cette idée, nous définirons un système simplifié pour l'évaluation de termes

clos en Section IV.4.

IV.3.2 Règles de propagation

Il nous faut maintenant des règles pour propager les substitutions créées par la règle *Beta* définie ci-dessus en Section IV.3.1. Les directeurs indiquent le chemin que la substitution doit suivre : nous avons besoin d'une règle par construction de terme et par directeur possible.

Pour comprendre comment les règles de propagation des substitutions sont définies, considérons un cas simple : une application $(\mathbf{t}\ \mathbf{u})^{\curvearrowleft\rho}$ avec une substitution pour la première variable, c'est-à-dire que nous considérons $((\mathbf{t}\ \mathbf{u})^{\curvearrowleft\rho}[\mathbf{v}])^{\sigma}$. La substitution doit être propagée dans la branche gauche du nœud d'application, comme l'indique le directeur $\curvearrowleft$. Nous avons donc besoin d'une règle de la forme :

$$(App_1) \qquad ((\mathbf{t}\ \mathbf{u})^{\curvearrowleft\rho}[\mathbf{v}])^{\sigma} \rightsquigarrow ((\mathbf{t}[\mathbf{v}])^{\upsilon}\ \mathbf{u})^{\tau}$$

Essayons de trouver υ et τ. Supposons qu'une substitution (close) soit appliquée aux membres gauche et droit de l'équation ci-dessus. Si, par exemple, $\sigma = \rho = \curvearrowleft$, alors la substitution est pour $\mathbf{t}$ à gauche, donc nous devons avoir $\tau = \upsilon = \curvearrowleft$ de façon à ce que la substitution soit aussi guidée vers $\mathbf{t}$ à droite. Si $\sigma = \curvearrowleft$ et $\rho = \curvearrowright$, alors elle doit être dirigée vers $\mathbf{u}$, et nous devons avoir $\tau = \curvearrowright$ et υ n'est pas concerné (disons que $\upsilon = \epsilon$ ici). Finalement, si $\sigma = \curvearrowright$, la substitution est pour $\mathbf{v}$, et nous avons $\tau = \curvearrowleft$ et $\upsilon = \curvearrowright$.

Nous obtenons la plupart des règles de propagation de cette façon. Remarquons que toutes les combinaisons de directeurs n'ont pas besoin d'être prises en considération, dans la mesure où certaines d'entre elles ne correspondent pas à des termes bien formés. La Figure IV.2 montre l'ensemble $\mathcal{P}$ des règles de propagation.

Les diverses fonctions ϕ et ψ utilisées dans les règles de propagation calculent les chaînes directrices *ad hoc*. Elles sont générées récursivement de la même façon que ci-dessus à partir des tables de la Figure IV.3.

Par exemple :

$$\begin{array}{lrcl}
\phi_l(& \epsilon, & \epsilon) & = & \epsilon \\
\phi_l(\curvearrowright\sigma, & & \rho) & = & \curvearrowright\phi_l(\sigma,\rho) \\
\phi_l(\curvearrowleft\sigma, & \curvearrowright\rho) & & = & \phi_l(\sigma,\rho) \\
\phi_l(\curvearrowleft\sigma, & \curvearrowleft\rho) & & = & \curvearrowleft\phi_l(\sigma,\rho),\ \textit{etc.}
\end{array}$$

La fonction π utilisée seulement dans la règle *Erase* ajoute un nouveau directeur $-$ pour chaque variable d'un terme effacé, et peut être paraphrasée à partir de la Figure IV.3 comme suit (α est un directeur quelconque) :

$$\begin{array}{lrcl}
\pi(& \epsilon, & \epsilon) & = & \epsilon \\
\pi(\curvearrowright\sigma, & & \rho) & = & -\,\pi(\sigma,\rho) \\
\pi(\curvearrowleft\sigma, & \alpha\,\rho) & & = & \alpha\,\pi(\sigma,\rho) \\
\pi(\curlywedge\sigma, & \alpha\,\rho) & & = & \alpha\,\pi(\sigma,\rho) \\
\pi(-\,\sigma, & & \rho) & = & -\,\pi(\sigma,\rho)
\end{array}$$

La fonction π' utilisée dans la règle *Var* réalise une tâche similaire (π' n'est pas dans la Figure IV.3 parce qu'elle ne suit pas exactement le même schéma) :

Nom	Réduction				Cond.
Var	$(\Box^{\rho}[i/v^{\nu}])^{\sigma}$	$\rightsquigarrow_o$	v^{τ}	où $\tau = \pi'(\sigma, \nu)$	$\rho_i = \downarrow$
App_1	$((\mathbf{t}\ \mathbf{u})^{\rho}[i/\mathbf{v}])^{\sigma}$	$\rightsquigarrow_o$	$((\mathbf{t}[j/\mathbf{v}])^{\upsilon}\ \mathbf{u})^{\tau}$		$\rho_i = \curvearrowleft$
où $\upsilon = \phi_l(\sigma, \rho_{\setminus i}), \tau = \psi_1(\sigma, \rho_{\setminus i}), j = \lvert\rho_{1..i}\rvert_l$					
App_2	$((\mathbf{t}\ \mathbf{u})^{\rho}[i/\mathbf{v}])^{\sigma}$	$\rightsquigarrow_o$	$(\mathbf{t}\ (\mathbf{u}[k/\mathbf{v}])^{\omega})^{\tau}$		$\rho_i = \curvearrowright$
où $\omega = \phi_r(\sigma, \rho_{\setminus i}), \tau = \psi_2(\sigma, \rho_{\setminus i}), k = \lvert\rho_{1..i}\rvert_r$					
App_3	$((\mathbf{t}\ \mathbf{u})^{\rho}[i/\mathbf{v}])^{\sigma}$	$\rightsquigarrow_o$	$((\mathbf{t}[j/\mathbf{v}])^{\upsilon}\ (\mathbf{u}[k/\mathbf{v}])^{\omega})^{\tau}$		$\rho_i = \curlywedge$
où $\upsilon = \phi_l(\sigma, \rho_{\setminus i}), \omega = \phi_r(\sigma, \rho_{\setminus i}), \tau = \psi_3(\sigma, \rho_{\setminus i}), j = \lvert\rho_{1..i}\rvert_l, k = \lvert\rho_{1..i}\rvert_r$					
Lam	$((\lambda\mathbf{t})^{\rho}[i/\mathbf{v}])^{\sigma}$	$\rightsquigarrow_o$	$(\lambda(\mathbf{t}[i/\mathbf{v}])^{\upsilon\cdot\curvearrowleft})^{\tau}$		$\rho_i = \downarrow$
où $\upsilon = \phi_d(\sigma, \rho_{\setminus i}), \tau = \psi_d(\sigma, \rho_{\setminus i})$					
Comp	$((\mathbf{t}[j/\mathbf{u}])^{\rho}[i/\mathbf{v}])^{\sigma}$	$\rightsquigarrow_o$	$(\mathbf{t}[j/(\mathbf{u}[k/\mathbf{v}])^{\omega}])^{\tau}$		$\rho_i = \curvearrowright$
où $\omega = \phi_r(\sigma, \rho_{\setminus i}), \tau = \psi_2(\sigma, \rho_{\setminus i}), k = \lvert\rho_{1..i}\rvert_r$					
Erase	$(t^{\rho}[i/\mathbf{v}])^{\sigma}$	$\rightsquigarrow_o$	t^{τ}	où $\tau = \pi(\sigma, \rho_{\setminus i})$	$\rho_i = -$

FIG. IV.2 – Règles de propagation $\mathcal{P}$

σ_1	ρ_1	ϕ_l	ϕ_r	ψ_1	ψ_2	ψ_3	π
$\curvearrowright$	ϵ	$\curvearrowright$	$\curvearrowright$	$\curvearrowleft$	$\curvearrowright$	$\curlywedge$	$-$
$\curvearrowleft$	$\curvearrowright$	ϵ	$\curvearrowleft$	$\curvearrowright$	$\curvearrowright$	$\curvearrowright$	$\curvearrowright$
$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	ϵ	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$\curvearrowleft$	$\curlywedge$	$\curvearrowleft$	$\curvearrowleft$	$\curlywedge$	$\curlywedge$	$\curlywedge$	$\curlywedge$
$\curlywedge$	$\curvearrowright$	$\curvearrowright$	$\curlywedge$	$\curlywedge$	$\curvearrowright$	$\curlywedge$	$\curvearrowright$
$\curlywedge$	$\curvearrowleft$	$\curlywedge$	$\curvearrowright$	$\curvearrowleft$	$\curlywedge$	$\curlywedge$	$\curvearrowleft$
$\curlywedge$	$\curlywedge$	$\curlywedge$	$\curlywedge$	$\curlywedge$	$\curlywedge$	$\curlywedge$	$\curlywedge$
$-$	ϵ	ϵ	ϵ	$-$	$-$	$-$	$-$
$\curvearrowleft$	$-$	ϵ	ϵ	$-$	$-$	$-$	$-$
$\curlywedge$	$-$	$\curvearrowright$	$\curvearrowright$	$\curvearrowleft$	$\curvearrowright$	$\curlywedge$	$-$

σ_1	ρ_1	ψ_d	ϕ_d	ψ_b
$\curvearrowright$	ϵ	$\downarrow$	$\curvearrowright$	$\curvearrowright$
$\curvearrowleft$	$\downarrow$	$\downarrow$	$\curvearrowleft$	$\curvearrowleft$
$\curlywedge$	$\downarrow$	$\downarrow$	$\curlywedge$	$\curlywedge$
$-$	ϵ	$-$	ϵ	$-$
$\curvearrowleft$	$-$	$-$	ϵ	$-$
$\curlywedge$	$-$	$\downarrow$	$\curvearrowright$	$\curvearrowright$

FIG. IV.3 – Fonctions utilisées dans les règles de propagation

$$\begin{array}{rcl}
\pi'(\quad \epsilon, \quad \epsilon) & = & \epsilon \\
\pi'(\curvearrowright \sigma, \alpha\, \nu) & = & \alpha\, \pi'(\sigma, \nu) \\
\pi'(\curvearrowleft \sigma, \quad \nu) & = & -\, \pi'(\sigma, \nu) \\
\pi'(\curlywedge \sigma, \alpha\, \nu) & = & \alpha\, \pi'(\sigma, \nu) \\
\pi'(- \sigma, \quad \nu) & = & -\, \pi'(\sigma, \nu)
\end{array}$$

Ces règles méritent quelques explications :

- La règle *Var* est la plus simple. Quand la substitution atteint un marqueur de variable avec comme directeur correspondant $\downarrow$, nous savons qu'il s'agit de la bonne variable (parce que la substitution a été guidée ici et n'est pas effacée). Nous savons que $\rho_{\backslash i} = -^n$ si le terme est bien formé, de sorte que $-$ et $\curvearrowleft$ dans σ doivent tous deux résulter en un directeur $-$ dans τ pour assurer que les variables effacées soient préservées. De plus, nous n'avons pas besoin d'inspecter ρ pour calculer τ.
- Les règles pour l'application sont les règles principales ici. Selon ρ_i, la substitution est guidée à gauche ou à droite, ou copiée dans App_3 seulement quand il y a plus d'une occurrence de la variable en question. Les nouvelles chaînes directrices sont calculées par des fonctions *ad hoc* à partir de σ et ρ (en omettant le $i^{\text{ème}}$ directeur de cette dernière).
- De façon surprenante, la règle qui permet à une substitution ouverte de traverser une abstraction (*Lam*) est simple. Cela mérite d'être remarqué, car ce n'est généralement pas le cas dans les calculs habituels. Par exemple, cela nécessite de l'α-conversion dans les calculs nommés.
- La règle *Comp* correspond à App_2 en remplaçant l'application par une substitution. Nous aurions pu donner des règles de composition de substitutions similaires à App_1 et App_3, mais les substitutions pourraient alors s'échanger (leur ordre ne serait pas préservé), ce qui implique que le système échouerait trivialement à préserver la normalisation forte.
- La règle *Erase* s'applique à une substitution dans n'importe quel constructeur de terme, pourvu que le directeur correspondant à la substitution soit $-$. Alors la substitution est simplement éliminée et de nouveaux directeurs $-$ sont ajoutés pour prendre en compte les variables libres du terme éliminé.

Nous avons un nombre de règles de propagation relativement petit, au regard des calculs avec substitutions explicites standard. Cependant, nos règles demandent des manipulations syntaxiques non-triviales sur les chaînes directrices quand nous considérons des substitutions arbitraires. Le système peut être drastiquement simplifié si nous imposons certaines restrictions sur les substitutions, comme nous allons le voir dans la section suivante. Notons au passage que la condition sur ρ_i dans les règles a été externalisée seulement pour améliorer la lisibilité et n'est bien sûr qu'un simple filtrage de motif.

Exemple IV.3.2

Nous montrons une suite de réductions dans ce calcul. Considérons le λ-terme $\lambda x.(\lambda y.y)\, x$ qui contient un seul rédex :

$$[\![\lambda x.(\lambda y.y)\, x]\!] = (\lambda((\lambda\Box^{\downarrow})^{\epsilon}\Box^{\downarrow})^{\curvearrowright})^{\epsilon} \rightsquigarrow_o (\lambda(\Box^{\downarrow}[\Box^{\downarrow}])^{\curvearrowright})^{\epsilon} \rightsquigarrow_o (\lambda\Box^{\downarrow})^{\epsilon} = [\![\lambda x.x]\!]$$

Notons qu'un codage à l'aide de combinateurs, grâce aux chaînes directrices présentées

dans [KS88], ne permettrait pas à ce rédex d'être contracté, et pourrait donc potentiellement être copié s'il était utilisé comme argument. En ce sens, notre calcul généralise donc les chaînes directrices de [KS88].

IV.3.3 Propriétés

Lemme IV.3.3 (*Préservation de la bonne formation*)

Si t^σ est un terme bien formé et $t^\sigma \rightsquigarrow_o u^\tau$, alors u^τ est un terme bien formé, de plus $|\sigma| = |\tau|$.

Preuve. Il faut prouver que chaque règle produit un terme bien formé avec une chaîne directrice de la même taille. La preuve est laborieuse mais sans difficultés, et nous l'omettons. □

Le processus de propagation des substitutions (c'est-à-dire l'ensemble des règles de $\mathcal{P}$) termine : chaque règle efface la substitution, ou bien la fait descendre d'un pas vers les feuilles. Formellement, nous prouvons la terminaison de $\mathcal{P}$ en utilisant une fonction d'interprétation. Cette fonction calcule les longueurs des chemins qui doivent être traversés par la substitution pour atteindre les variables correspondantes. Nous l'appelons la *distance* d'une substitution.

Définition IV.3.4 (*Distance*)

La distance associée à la substitution $[i/\mathbf{v}]$ dans le terme $(\mathbf{t}[i/\mathbf{v}])^\rho$ est calculée par la fonction $|\mathbf{t}[i/\mathbf{v}]|$ définie par induction sur $\mathbf{t}$ comme suit :

$$
\begin{array}{lcllll}
|\Box^\sigma[i/\mathbf{v}]| & = & 1 & & & \\
|(\lambda\mathbf{t})^\sigma[i/\mathbf{v}]| & = & 1 & & & (\text{si } \sigma_i = -) \\
|(\lambda\mathbf{t})^\sigma[i/\mathbf{v}]| & = & 1 + |\mathbf{t}[i/\mathbf{v}]| & & & (\text{si } \sigma_i = \downarrow) \\
|(\mathbf{t}\,\mathbf{u})^\sigma[i/\mathbf{v}]| & = & |(\mathbf{t}[j/\mathbf{u}])^\sigma[i/\mathbf{v}]| & = & 1 + |\mathbf{t}[k/\mathbf{v}]| & (\text{si } \sigma_i = \curvearrowleft) \\
|(\mathbf{t}\,\mathbf{u})^\sigma[i/\mathbf{v}]| & = & |(\mathbf{t}[j/\mathbf{u}])^\sigma[i/\mathbf{v}]| & = & 1 + |\mathbf{u}[k/\mathbf{v}]| & (\text{si } \sigma_i = \curvearrowright) \\
|(\mathbf{t}\,\mathbf{u})^\sigma[i/\mathbf{v}]| & = & |(\mathbf{t}[j/\mathbf{u}])^\sigma[i/\mathbf{v}]| & = & 1 + |\mathbf{t}[k/\mathbf{v}]| + |\mathbf{u}[k'/\mathbf{v}]| & (\text{si } \sigma_i = \swarrow\!\!\searrow) \\
|(\mathbf{t}\,\mathbf{u})^\sigma[i/\mathbf{v}]| & = & |(\mathbf{t}[j/\mathbf{u}])^\sigma[i/\mathbf{v}]| & = & 1 & (\text{si } \sigma_i = -)
\end{array}
$$

où k et k' sont calculés comme dans la Figure IV.2.

Proposition IV.3.5 (*Terminaison de la propagation*)

L'ensemble $\mathcal{P}$ des règles de propagation termine.

Preuve. Nous définissons une interprétation qui associe à chaque terme $\mathbf{t}$ un multi-ensemble avec un élément $|\mathbf{u}[i/\mathbf{v}]|$ pour chaque sous-terme $(\mathbf{u}[i/\mathbf{v}])^\rho$ de $\mathbf{t}$. Chaque application d'une règle de propagation diminue l'interprétation du terme : les règles *Var* et *Erase* effacent un élément du multi-ensemble, et dans les autre règles, un élément est remplacé par un ou deux éléments strictement plus petits. □

Il est facile de voir qu'on atteint toujours un terme pur (sans substitutions) en utilisant les règles de propagation :

Proposition IV.3.6 (*Complétion des substitutions*)

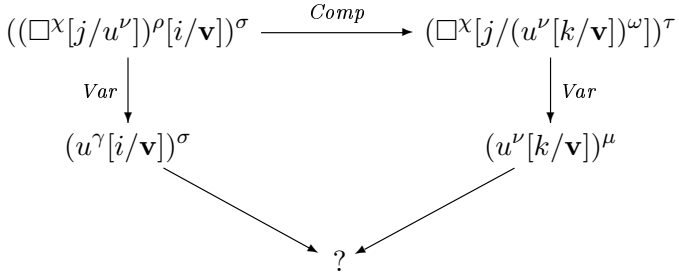
Chaque terme a une $\mathcal{P}$-forme normale qui est un terme pur.

Preuve. Nous prouvons par contradiction que tout terme de la forme $(\mathbf{u}[i/\mathbf{v}])^{\rho}$ peut être réduit par une règle de $\mathcal{P}$. Supposons qu'il existe un contre-exemple, et prenons en un de taille minimale : $(\mathbf{t}[i/\mathbf{w}])^{\sigma}$. Considérons tous les cas pour $\mathbf{t}$. Si $\mathbf{t}$ est une variable, application ou abstraction on peut évidemment appliquer une règle de $\mathcal{P}$. Si c'est une substitution $(\mathbf{t}'[j/\mathbf{u}'])^{\upsilon}$, alors elle est réductible (par minimalité du contre-exemple). □

LEMME IV.3.7 (*Confluence locale de la propagation*)

L'ensemble $\mathcal{P}$ des règles de propagation est localement confluent.

Preuve. $\mathcal{P}$ a sept paires critique, toutes avec *Comp*. Dans la version du calcul sans directeur pour l'effacement (Section IV.4), elles convergent toutes facilement. Mais ici, les paires se réduisent vers des termes où l'effacement peut avoir lieu à deux niveaux différents, et nous devons donc effectuer une étape supplémentaire avec la règle correspondante pour aplatir ces deux chaînes en une, et obtenir des termes syntaxiquement égaux. Bien sûr, cela rend la preuve particulièrement fastidieuse, du fait de la définition par cas des fonctions impliquées dans les règles, nous ne donnons donc les détails que pour la paire *Var*/*Comp*, dans la mesure où les autre paires critiques peuvent être traitées de façon similaire.



avec les conditions $\chi_j = \downarrow, \rho_i = \curvearrowright$ et les résultats intermédiaires :
$k = |\rho_{1..i}|_r, \gamma = \pi'(\rho,\nu), \tau = \psi_2(\sigma,\rho_{\setminus i}), \omega = \phi_r(\sigma,\rho_{\setminus i}), \mu = \pi'(\tau,\omega)$.

Les deux termes peuvent être syntaxiquement différents, la seule différence étant que des directeurs − peuvent avoir été ajoutés dans γ dans le terme de gauche et dans μ dans le terme de droite, donc à des niveaux différents. Heureusement, ces termes ne sont pas en $\mathcal{P}$-forme normale (grâce à la Proposition IV.3.6), et nous savons qu'une autre $\mathcal{P}$-règle peut être appliquée à la racine des deux termes. (Il y a un cas où nous ne le pouvons pas, mais alors nous avons $u^{\nu} \rightsquigarrow_o^* u'^{\nu'}$ où nous pouvons réduire à la racine et $u^{\gamma} = u^{\pi'(\rho,\nu)} \rightsquigarrow_o^* u'^{\pi'(\rho,\nu')}$, et la situation est similaire). C'est en effet la même règle pour les deux termes, parce que le terme sans annotation à gauche de la substitution est le même et $\gamma_i = \nu_k$, ce qui vient des observations suivantes :

(i) $\pi'(\rho,\nu)$ est bien défini si et seulement si $|\nu| = |\rho|_r$;

(ii) si $|\nu| = |\rho|_r$, $1 \le i \le |\rho|$ et $k = |\rho_{1..i}|_r$, $\pi'(\rho_{1..i}, \nu_{1..k})$ est un préfixe de $\pi'(\rho,\nu)$;

(iii) si $|\nu| = |\rho|_r$, $|\pi'(\rho,\nu)| = |\rho|$;

(iv) si $|\nu| = |\rho|_r$, $1 \le i \le |\rho|$ et $k = |\rho_{1..i}|_r$, $(\pi'(\rho,\nu))_{1..i} = \pi'(\rho_{1..i}, \nu_{1..k})$.

(i), (ii), (iii) sont clairs à partir de la définition de π' et (iv) est une conséquence directe de (ii) et (iii). Comme corollaire, nous avons en effet dans notre cas à partir de (iv), que $\gamma_i = \nu_k$ et $(\pi'(\rho,\nu))_{\backslash i} = \pi'(\rho_{\backslash i}, \nu_{\backslash k})$. $\quad (*)$

Maintenant, pour prouver que les termes sont égaux après cette réduction (inconnue), il suffit de montrer que $f(\sigma, \gamma_{\backslash i}) = f(\mu, \nu_{\backslash k})$ pour $f \in \{\phi_l, \phi_r, \phi_d, \psi_1, \psi_2, \psi_3, \psi_d, \pi, \pi'\}$ et $|\gamma_{1..i}|_r = |\nu_{1..k}|_r$, $|\gamma_{1..i}|_l = |\nu_{1..k}|_l$.

Cette tâche est bien sûr affreusement fastidieuse et nous l'omettons en grande partie. Nous allons cependant traiter un cas complètement, disons ψ_1, ce qui signifie que notre règle inconnue était en fait App_1, et nous voulons comparer les chaînes directrices extérieures de ces deux termes. Nous devons donc comparer $\psi_1(\sigma, \gamma_{\backslash i})$ et $\psi_1(\mu, \nu_{\backslash k})$ pour tous les cas valides, autrement dit, pour tous les cas qui correspondent à des termes initialement bien formés :

σ	$\rho_{\backslash i}$	$\nu_{\backslash k}$	$\gamma_{\backslash i}$	$\psi_1(\sigma, \gamma_{\backslash i})$	τ	ω	μ	$\psi_1(\mu, \nu_{\backslash k})$
$\curvearrowright$	ϵ	ϵ	ϵ	$\curvearrowleft$	$\curvearrowright$	$\curvearrowright$	$\curvearrowright$	$\curvearrowleft$
$\curvearrowleft$	$\curvearrowright$	$\curvearrowright$	$\curvearrowright$	$\curvearrowright$	$\curvearrowright$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowright$
$\curvearrowleft$	$\curvearrowright$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowright$	$\curvearrowleft$	$\curvearrowleft$	$\curvearrowleft$
$\curvearrowleft$	$\curvearrowright$	$-$	$-$	$-$	$\curvearrowright$	$\curvearrowleft$	$\curvearrowleft$	$-$
$\curvearrowleft$	$\curvearrowleft$	ϵ	$-$	$-$	$\curvearrowleft$	ϵ	$-$	$-$
$\curvearrowleft$	$-$	ϵ	$-$	$-$	$\curvearrowleft$	ϵ	$-$	$-$
$-$	ϵ	ϵ	ϵ	$-$	$-$	ϵ	$-$	$-$

Remarquons que l'on calcule $\gamma_{\backslash i}$ à partir de $\rho_{\backslash i}$ et $\nu_{\backslash k}$ grâce à $(*)$.

La table ci-dessus montre que si σ, ρ, ν sont comme définis dans un terme bien formé, $\psi_1(\sigma, \gamma_{\backslash i}) = \psi_1(\mu, \nu_{\backslash k})$ (les cas avec $\not\curvearrowright$ ont été omis mais sont similaires), ainsi, après l'application de la règle App_1 aux deux termes de la paire critique, leurs chaînes directrices extérieures sont les mêmes. Une preuve similaire avec ϕ_l permet de conclure que ces termes sont bien les mêmes, ce qui conclut ce cas. Tous les autres cas sont similaires. □

La confluence locale et la terminaison impliquent la confluence, grâce au Lemme de Newman I.1.4.

COROLLAIRE IV.3.8 (*Confluence de la propagation*)

L'ensemble $\mathcal{P}$ des règles de propagation est confluent sur les λ_o-termes.

Comme conséquence de la confluence (Corollaire IV.3.8) et de la terminaison (Proposition IV.3.5), l'ensemble $\mathcal{P}$ des règles de propagation définit une fonction d'un terme $\mathbf{t}$ vers son unique forme normale, dénotée $\mathcal{P}(\mathbf{t})$. Le lemme suivant montre que $\mathcal{P}$ implante la méta-opération de substitution du λ-calcul : pour calculer $M\{x := N\}$, il suffit de compiler M, compiler N, créer une substitution explicite, et la normaliser avec $\mathcal{P}$.

LEMME IV.3.9 (*Substitution*)

Soient M et N des λ-termes. Soit $[\![M]\!]_{\vec{x}} = t^{\rho}$, $[\![N]\!]_{\vec{y}} = \mathbf{u}$. Soit $\rho' = \rho-$ et $i = |\rho| + 1$ si $x \notin \vec{x}$, sinon $\rho' = \rho$ et i est la position de x dans $\vec{x}$. Alors $(\!|\mathcal{P}((t^{\rho'}[i/\mathbf{u}])^{\sigma})|\!)_{\vec{z}} = M\{x := N\}$ (où $\vec{x}, \vec{y}, \vec{z}, \sigma$ sont choisis de façon adéquate).

Preuve. Par induction sur M. Si M est une variable, nous distinguons deux cas : si $M = y \neq x$, nous obtenons le résultat requis en utilisant la règle *Erase*, sinon nous utilisons *Var*. Le cas

d'une application suit directement par induction en utilisant les règles App_1, App_2, App_3 ou *Erase*. Pour une abstraction, nous utilisons l'hypothèse d'induction et les règles *Lam* ou *Erase*. □

La réduction dans λ_o est correcte vis-à-vis du λ-calcul, dans le sens suivant.

PROPOSITION IV.3.10 (*Correction de λ_o*)

Si $\mathbf{t} \rightsquigarrow_o \mathbf{u}$, alors $(\!|\mathbf{t}|\!)_{\vec{x}} \to_\beta^* (\!|\mathbf{u}|\!)_{\vec{x}}$. En particulier :

1. Si $\mathbf{t} \rightsquigarrow_{\mathcal{P}} \mathbf{u}$, alors $(\!|\mathbf{t}|\!)_{\vec{x}} = (\!|\mathbf{u}|\!)_{\vec{x}}$, et
2. Si $\mathbf{t}$ est une $\mathcal{P}$-forme normale, alors $(\!|\mathbf{t}|\!)_{\vec{x}} \to_\beta (\!|\mathbf{u}|\!)_{\vec{x}}$.

Preuve. Le résultat est trivial si la réduction utilise une règle de $\mathcal{P}$, puisque la fonction de décompilation réalise la substitution :

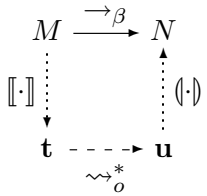
$$(\!|\mathbf{t}|\!)_{[x_1,\ldots,x_{i-1},M_i,x_{i+1},\ldots,x_n]} = (\!|\mathbf{t}|\!)_{[x_1,\ldots,x_n]}\{x_i := M_i\}.$$

Supposons que la règle *Beta* est utilisée. Nous procédons par induction sur $\mathbf{t}$. Le seul cas intéressant est quand la réduction a lieu à la racine de $\mathbf{t}$. Dans ce cas $\mathbf{t} = ((\lambda \mathbf{w})^\rho \mathbf{v})^\sigma$ et $\mathbf{u} = (\mathbf{w}[|\rho|_+ + 1/\mathbf{v}])^\tau$. Par définition de la décompilation, $(\!|\mathbf{t}|\!)_{\vec{x}} = (\lambda x.(\!|\mathbf{w}|\!)_{\vec{y}})(\!|\mathbf{v}|\!)_{\vec{z}}$, où $\vec{x}, \vec{y}, \vec{z}$ sont les listes de variables libres correspondantes. Il y a donc une étape de β-réduction : $(\!|\mathbf{t}|\!)_{\vec{x}} \to_\beta (\!|\mathbf{w}|\!)_{\vec{y}}\{x := (\!|\mathbf{v}|\!)_{\vec{z}}\} = (\!|\mathbf{u}|\!)_{\vec{x}}$ par définition de la décompilation. Dans le cas général, ce *Beta*-rédex peut apparaître dans une substitution et être copié ou effacé par la fonction de décompilation, donc $(\!|\mathbf{t}|\!)_{\vec{x}} \to_\beta^* (\!|\mathbf{u}|\!)_{\vec{x}}$. En particulier, si $\mathbf{t}$ est une $\mathcal{P}$-forme normale, alors $(\!|\mathbf{t}|\!)_{\vec{x}} \to_\beta (\!|\mathbf{u}|\!)_{\vec{x}}$ puisque $\mathbf{t}$ ne contient pas de substitutions. □

Nous avons aussi un Théorème de Simulation qui exprime la complétude de λ_o pour la β-réduction.

THÉORÈME IV.3.11 (*Simulation*)

Soit M un λ-terme avec $\mathsf{fv}(M) \subseteq \vec{x}$. Si $M \to_\beta N$, alors il existe $\mathbf{u}$ tel que $[\![M]\!]_{\vec{x}} \rightsquigarrow_o^* \mathbf{u}$ et $(\!|\mathbf{u}|\!)_{\vec{x}} = N$.



Preuve. Puisque la fonction de compilation (Définition IV.2.3) transforme les applications en applications et les abstractions en abstractions, il est clair que si M a un β-rédex alors $[\![M]\!]_{\vec{x}}$ a un *Beta*-rédex. Nous procédons par induction sur M. Le seul cas intéressant est quand la β-réduction a lieu à la racine de M. Dans ce cas, $M = (\lambda x.M')\, N'$ et $N = M'\{x := N'\}$. En utilisant la fonction de compilation, nous obtenons :

- $[\![\lambda x.M']\!]_{\vec{z}} = (\lambda \mathbf{r})^\nu$ où $\mathbf{r} = [\![M']\!]_{\vec{z}'x}$
- $[\![N']\!]_{\vec{y}} = \mathbf{w}$

– $[\![M]\!]_{\vec{x}} \rightsquigarrow_{Beta} (\mathbf{r}[i/\mathbf{w}])^{\sigma}$

En utilisant le Lemme de Substitution (Lemme IV.3.9), nous obtenons le résultat demandé :

$$(\mathbf{r}[i/\mathbf{w}])^{\sigma} \rightsquigarrow_o^* \mathbf{u} \text{ avec } (\!|\mathbf{u}|\!)_{\vec{x}} = N.$$

Les étapes de la dernière réduction sont toutes dans $\mathcal{P}$.

□

Remarque IV.3.12

En général, nous n'avons pas $\mathbf{u} = [\![N]\!]_{\vec{x}}$ (ce qui serait plus fort). A titre d'exemple, $[\![\lambda x\lambda y.((\lambda z.y)\ x)]\!] = (\lambda(\lambda((\lambda\Box^{\downarrow -})^{\downarrow}\Box^{\downarrow})\curvearrowright\curvearrowleft)^{\downarrow})^{\epsilon} \rightsquigarrow_o^* (\lambda(\lambda\Box^{-\downarrow})^{\downarrow})^{\epsilon}$, et $(\!|(\lambda(\lambda\Box^{-\downarrow})^{\downarrow})^{\epsilon}|\!) = \lambda x\lambda y.y$, alors que $[\![\lambda x\lambda y.y]\!] = (\lambda(\lambda\Box^{\downarrow})^{-})^{\epsilon}$. Intuitivement, il y a deux niveaux différents où nous pouvons dire que x doit être effacé. La fonction de compilation fait le choix canonique de placer cette information au plus haut niveau possible, c'est-à-dire juste en dessous du λ correspondant, mais il peut aussi arriver qu'une variable disparaisse au cours d'une réduction, comme dans l'exemple ci-dessus. Nous pourrions bien sûr ajouter une règle pour faire remonter les directeurs d'effacement jusqu'à ce qu'ils atteignent leur position canonique, de sorte que nous aurions effectivement $\mathbf{u} = [\![N]\!]_{\vec{x}}$. Mais c'est inutile, et c'est exactement ce que dit le théorème de simulation : nous pouvons avoir des représentations internes différentes correspondant au même terme durant la réduction, mais cela nous importe peu tant que nous pouvons toujours récupérer le résultat correct à la fin de la réduction. Ainsi, bien qu'apparemment plus faible que ce à quoi l'on pouvait s'attendre, le Théorème IV.3.11 est exactement la bonne propriété pour un langage intermédiaire qui sépare proprement la décompilation des règles de réduction.

Pour prouver la confluence de λ_o, nous avons besoin de quelques lemmes supplémentaires. Définissons $\mathbf{t} \rightarrow_B \mathbf{u}$ si $\mathbf{t}$, $\mathbf{u}$ sont en $\mathcal{P}$-forme normale et $\mathbf{t} \rightsquigarrow_{Beta} \rightsquigarrow_{\mathcal{P}}^* \mathbf{u}$.

Lemme IV.3.13

$\mathbf{t} \rightsquigarrow_o \mathbf{u} \Rightarrow \mathcal{P}(\mathbf{t}) \rightarrow_B^* \mathcal{P}(\mathbf{u})$.

Preuve. Évident si $\mathbf{t} \rightsquigarrow_{\mathcal{P}} \mathbf{u}$. Si $\mathbf{t} \rightsquigarrow_{Beta} \mathbf{u}$, le cas intéressant est quand la réduction est a la racine. Alors $((\lambda\mathbf{t})^{\rho}\,\mathbf{v})^{\sigma} \rightsquigarrow_{Beta} (\mathbf{t}[|\rho|_{+}+1/\mathbf{v}])^{\tau}$ et $((\lambda\mathcal{P}(\mathbf{t}))^{\rho}\,\mathcal{P}(\mathbf{v}))^{\sigma} \rightarrow_B \mathcal{P}((\mathcal{P}(\mathbf{t})[|\rho|_{+}+1/\mathcal{P}(\mathbf{v})])^{\tau}) = \mathcal{P}(\mathbf{u})$. □

Lemme IV.3.14

Si $\mathbf{t} \rightarrow_B \mathbf{u}$ alors $(\!|\mathbf{t}|\!)_{\vec{x}} \rightarrow_{\beta} (\!|\mathbf{u}|\!)_{\vec{x}}$ (une étape).

Preuve. Conséquence directe de la Proposition IV.3.10.

□

Nous pouvons définir une notion de résidus et de développements comme dans le λ-calcul, en utilisant la relation $\rightarrow_B$ sur les $\mathcal{P}$-formes normales de λ_o-termes (voir [Bar84]). Alors, clairement, par le Lemme IV.3.14, à un résidu de $\mathbf{t}$ par $\rightarrow_B$ correspond un résidu de $(\!|\mathbf{t}|\!)_{\vec{x}}$ par $\rightarrow_{\beta}$. Nous avons ainsi un Théorème des Développements Finis pour $\rightarrow_B$:

LEMME IV.3.15 (*FD pour* $\to_B$)

Si $\mathbf{t}$ est une $\mathcal{P}$-forme normale, tous les développements de $\mathbf{t}$ par la réduction $\to_B$ sont finis.

Preuve. Supposons qu'il y ait un développement infini de $(\mathbf{t}, \mathcal{F})$ où $\mathbf{t}$ est un λ_o-terme en $\mathcal{P}$-forme normale et $\mathcal{F}$ un ensemble d'occurrences de $\to_B$-rédex de $\mathbf{t}$:

$$\mathbf{t} \to_B \mathbf{t}_1 \to_B \cdots \to_B \mathbf{t}_n \to_B \cdots$$

Alors nous avons par le Lemme IV.3.14 :

$$(\!|\mathbf{t}|\!)_{\vec{x}} \to_\beta (\!|\mathbf{t}_1|\!)_{\vec{x}} \to_\beta \cdots \to_\beta (\!|\mathbf{t}_n|\!)_{\vec{x}} \to_\beta \cdots$$

où chaque étape est un résidu d'un rédex correspondant à $\mathcal{F}$. Puisque le λ-calcul satisfait le Théorème des Développements Finis [Bar84], c'est impossible.

□

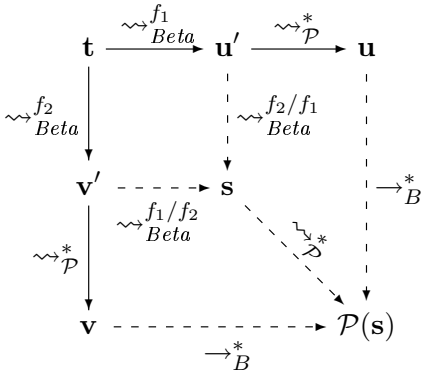
Un développement complet de $(\mathbf{t}, \mathcal{F})$ est un développement de $(\mathbf{t}, \mathcal{F})$ tel qu'il ne reste aucun résidu de $\mathcal{F}$ après lui.

LEMME IV.3.16

Tous les développements complets de $(\mathbf{t}, \mathcal{F})$ par $\to_B$ terminent sur le même terme.

Preuve. Supposons que $\mathbf{t} \to_B \mathbf{u}$ et $\mathbf{t} \to_B \mathbf{v}$ sont les premières étapes de deux développements complets de $(\mathbf{t}, \mathcal{F})$, obtenues en réduisant les rédex f_1 et f_2 respectivement. Alors, par définition de $\to_B$, $\mathbf{u}$ et $\mathbf{v}$ sont des $\mathcal{P}$-formes normales, et il y a des termes $\mathbf{u}'$ et $\mathbf{v}'$ tels que $\mathbf{t} \rightsquigarrow_{Beta} \mathbf{u}' \rightsquigarrow_{\mathcal{P}}^* \mathbf{u} = \mathcal{P}(\mathbf{u}')$ et $\mathbf{t} \rightsquigarrow_{Beta} \mathbf{v}' \rightsquigarrow_{\mathcal{P}}^* \mathbf{v} = \mathcal{P}(\mathbf{v}')$.

Si le *Beta*-rédex réduit est le même dans les deux cas, alors évidemment $\mathbf{u} = \mathbf{v}$ parce que $\rightsquigarrow_{\mathcal{P}}$ est confluent. Supposons que $f_1, f_2 \in \mathcal{F}$ soient des *Beta*-rédex différents. Alors $\mathbf{u}$ et $\mathbf{v}$ peuvent être joints grâce au Lemme IV.3.13, comme le montre le diagramme suivant, où $\rightsquigarrow_{Beta}^{f_i/f_j}$ dénote la *Beta*-réduction des résidus de f_i relativement à f_j (voir [Bar84]) :



Remarquons que la règle *Beta* est linéaire gauche et se superpose seulement trivialement avec elle-même, ce qui garantit l'existence des termes $\mathbf{s}$ et $\mathcal{P}(\mathbf{s})$ obtenus en développant $(\mathbf{t}, \{f_1, f_2\})$.

Le Lemme IV.3.15 et le Lemme de Newman [New42] permettent de terminer la preuve.

□

LEMME IV.3.17
$\rightarrow_B$ est confluent.

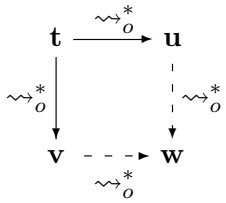
Preuve. Définissons $\mathbf{t} \rightarrowtail_B \mathbf{u}$ si $\mathbf{u}$ est un développement complet de $(\mathbf{t}, \mathcal{F})$ pour $\rightarrow_B$ pour un certain $\mathcal{F}$. Alors :

- $\rightarrow_B^* = \rightarrowtail_B^*$
- $\rightarrowtail_B$ est fortement confluent : si $\mathbf{t} \rightarrowtail_B \mathbf{t}_1$ (avec $\mathcal{F}_1$) et $\mathbf{t} \rightarrowtail_B \mathbf{t}_2$ (avec $\mathcal{F}_2$), alors soit $\mathbf{t}_3$ un développement complet de $\mathbf{t}$ avec $\mathcal{F}_1 \cup \mathcal{F}_2$. $\mathbf{t} \rightarrowtail_B \mathbf{t}_1$ provient d'un développement partiel de $(\mathbf{t}, \mathcal{F}_1 \cup \mathcal{F}_2)$, ainsi en développant complètement les résidus de $\mathcal{F}_1 \cup \mathcal{F}_2$ dans $\mathbf{t}_1$, on obtient par le Lemme IV.3.16 $\mathbf{t}_1 \rightarrowtail_B \mathbf{t}_3$. Similairement $\mathbf{t}_2 \rightarrowtail_B \mathbf{t}_3$.

□

THÉORÈME IV.3.18 (*Confluence*)
Le calcul λ_o est confluent.

Preuve. La situation est la suivante :



Nous utilisons la méthode d'interprétation. Par le Lemme IV.3.13, $\mathcal{P}(\mathbf{t}) \rightarrow_B^* \mathcal{P}(\mathbf{u})$ et $\mathcal{P}(\mathbf{t}) \rightarrow_B^* \mathcal{P}(\mathbf{v})$. Par Lemme IV.3.17, il existe $\mathbf{w}$ tel que $\mathcal{P}(\mathbf{u}) \rightarrow_B^* \mathbf{w}$ et $\mathcal{P}(\mathbf{v}) \rightarrow_B^* \mathbf{w}$. Et puisque $\rightarrow_B^* \subset \rightsquigarrow_o^*$, les dérivations $\mathbf{u} \rightsquigarrow_o^* \mathcal{P}(\mathbf{u}) \rightsquigarrow_o^* \mathbf{w}$ et $\mathbf{v} \rightsquigarrow_o^* \mathcal{P}(\mathbf{v}) \rightsquigarrow_o^* \mathbf{w}$ concluent la preuve. □

La propriété de confluence ci-dessus est parfois appelée *ground confluence* dans la terminologie des calculs avec substitutions explicites, puisqu'elle s'applique à des termes de λ_o plutôt qu'à des termes avec des métavariables du système de récriture. Une autre propriété qui a été abondamment étudiée pour les calculs avec substitution explicites est la préservation de la normalisation forte (PSN) : un calcul avec substitutions explicites préserve la normalisation forte si la compilation d'un λ-terme fortement normalisable est fortement normalisable (voir [Mel95] pour un contre-exemple pour $\lambda\sigma$, et [LRD95, DG01] pour des calculs satisfaisant la PSN). Grâce à notre forme limitée de composition (la règle *Comp* permet seulement de déplacer une substitution à l'intérieur d'une autre où la variable substituée apparaît) λ_o préserve la normalisation forte. Notre preuve de PSN est inspirée de celle de λ_v [LRD95]. Nous définissons d'abord une notion de dérivation infinie minimale et de réduction externe. Intuitivement, une dérivation est minimale si nous réduisons toujours un rédex le plus bas possible pour conserver la terminaison, et une étape de réduction est externe si elle n'a pas lieu à l'intérieur d'une substitution. Nous dénotons par $\rightsquigarrow_{Beta,p}$ une *Beta*-réduction à la position p, et par $\rightsquigarrow_{o,p}$ une λ_o-réduction arbitraire à la position p.

DÉFINITION IV.3.19 (*Dérivation minimale*)
Une suite infinie de réductions dans λ_o

$$\mathbf{t}_1 \rightsquigarrow_{Beta,p_1} \mathbf{t}'_1 \rightsquigarrow_{\mathcal{P}}^* \cdots \mathbf{t}_i \rightsquigarrow_{Beta,p_i} \mathbf{t}'_i \rightsquigarrow_{\mathcal{P}}^* \ldots$$

est *minimale* si pour toute autre dérivation infinie

$$\mathbf{t}_1 \rightsquigarrow_{Beta,p_1} \mathbf{t}'_1 \rightsquigarrow^*_{\mathcal{P}} \cdots \mathbf{t}_i \rightsquigarrow_{Beta,q} \mathbf{u} \rightsquigarrow^*_{\mathcal{P}} \cdots$$

nous avons $p_i \not\leq_{pref} q$ ($q \neq p_i p'$ pour tout p').

En d'autres termes, dans toute autre dérivation infinie, p_i et q sont disjoints ou q est au-dessus de p_i, ce qui signifie que le *Beta*-rédex que nous réduisons est l'un des plus bas.

Définition IV.3.20 (*Réduction externe*)

L'ensemble $Ext(\mathbf{t})$ des positions externes du terme $\mathbf{t}$ est définie comme suit, où Λ dénote la position racine et $x\,X$ dénote $\{x \cdot y \,|\, y \in X\}$ si X est un ensemble de positions :

$$\begin{array}{lcl} Ext(\square^\sigma) & = & \{\Lambda\} \\ Ext((\lambda\mathbf{t})^\sigma) & = & 1Ext(\mathbf{t}) \cup \{\Lambda\} \\ Ext((\mathbf{t}\ \mathbf{u})^\sigma) & = & 1Ext(\mathbf{t}) \cup 2Ext(\mathbf{u}) \cup \{\Lambda\} \\ Ext((\mathbf{t}[k/\mathbf{u}])^\sigma) & = & 1Ext(\mathbf{t}) \cup \{\Lambda\} \end{array}$$

Une étape de réduction $\mathbf{t} \rightsquigarrow_{o,p} \mathbf{u}$ est externe si $p \in Ext(\mathbf{t})$, autrement elle est interne. Nous notons $\mathbf{t} \rightsquigarrow^{ext}_o \mathbf{u}$ (resp. $\mathbf{t} \rightsquigarrow^{int}_o \mathbf{u}$) pour insister sur le fait qu'une récriture est externe (resp. interne).

Lemme IV.3.21

Si $\mathbf{t} \rightsquigarrow_{Beta,p} \mathbf{u}$ est une étape de réduction externe alors $(\!|\mathbf{t}|\!)_{\vec{x}} \to^+_\beta (\!|\mathbf{u}|\!)_{\vec{x}}$. En particulier $(\!|\mathbf{t}|\!)_{\vec{x}} \neq (\!|\mathbf{u}|\!)_{\vec{x}}$ si $(\!|\mathbf{t}|\!)_{\vec{x}}$ est fortement normalisable.

Preuve. Le terme $\mathbf{t}$ contient un *Beta*-rédex, et puisqu'il est externe, il n'est évidemment pas effacé lors de la décompilation (voir la Proposition IV.3.10). Ainsi $(\!|\mathbf{t}|\!)_{\vec{x}} \to^+_\beta (\!|\mathbf{u}|\!)_{\vec{x}}$. □

Lemme IV.3.22

S'il y a une dérivation infinie

$$\mathbf{t} \rightsquigarrow_o \mathbf{t}_1 \rightsquigarrow_o \mathbf{t}_2 \rightsquigarrow_o \mathbf{t}_3 \rightsquigarrow_o \mathbf{t}_4 \cdots$$

dans laquelle toutes les étapes de *Beta* sont internes, alors il existe une autre dérivation infinie

$$\mathbf{t} \rightsquigarrow^*_{\mathcal{P}} \mathbf{u}_1 \rightsquigarrow_o \mathbf{u}_2 \rightsquigarrow_o \mathbf{u}_3 \rightsquigarrow_o \mathbf{u}_4 \cdots$$

dans laquelle les récritures de $\mathbf{t}$ à $\mathbf{u}_1$ sont les seules externes (autrement dit, *toutes* les étapes sont internes à partir de $\mathbf{u}_1$). De plus, la transformation préserve la minimalité.

Preuve. Il suffit de prouver que si $\mathbf{u} \rightsquigarrow^{int}_o \mathbf{v} \rightsquigarrow^{ext}_o \mathbf{w}$ dans une dérivation minimale alors $\mathbf{u} \rightsquigarrow^{ext}_o \mathbf{v}' \rightsquigarrow^{int*}_o \mathbf{w}$ en préservant la minimalité. La preuve est par induction sur la structure de $\mathbf{u}$. Si la récriture externe n'a pas lieu à la position racine alors le résultat suit par induction. Si la récriture externe a lieu à la racine de $\mathbf{u}$, alors la règle appliqué est dans $\mathcal{P}$ par hypothèse, et donc $\mathbf{u}$ est un terme de la forme $(\mathbf{t}[i/\mathbf{v}])^\sigma$. Nous distinguons des cas selon la règle appliquée.

Si la règle est *Var*, App_1, App_2, *Lam*, ou *Comp* les étapes de récriture commutent. Si la règle est App_3, la récriture interne est remplacée par deux récritures internes. Si la règle est *Erase*, la récriture interne n'est pas nécessaire. La terminaison de $\mathcal{P}$ (Proposition IV.3.5) assure qu'il y a un nombre fini d'étapes externes $\mathbf{t} \rightsquigarrow_{\mathcal{P}}^{*} \mathbf{u}_1$, ce qui achève la preuve. □

Nous allons prouver un résultat un peu plus général que la préservation de la normalisation forte, qui nous sera aussi utile pour prouver la terminaison des termes typables dans la Section IV.6.

Proposition IV.3.23

Si $(\!|\mathbf{t}|\!)_{\vec{x}}$ est un λ-terme fortement normalisable alors $\mathbf{t}$ est fortement normalisable dans λ_o.

Preuve. Par l'absurde, supposons qu'il y ait une suite de réductions infinie partant de $\mathbf{t}$. Nous montrons qu'il y a une dérivation infinie pour $(\!|\mathbf{t}|\!)_{\vec{x}}$. Pour cela, nous considérons une suite de réductions infinie *minimale*, qui contient un nombre infini d'applications de *Beta* puisque les règles de propagation terminent :

$$\mathbf{t} \rightsquigarrow_{\mathcal{P}}^{*} \mathbf{t}_1 \rightsquigarrow_{Beta} \mathbf{t}_2 \rightsquigarrow_{\mathcal{P}}^{*} \mathbf{t}_3 \rightsquigarrow_{Beta} \mathbf{t}_4 \cdots$$

Par la Proposition IV.3.10 :

$$(\!|\mathbf{t}|\!)_{\vec{x}} = (\!|\mathbf{t}_1|\!)_{\vec{x}} \rightarrow_{\beta}^{*} (\!|\mathbf{t}_2|\!)_{\vec{x}} = (\!|\mathbf{t}_3|\!)_{\vec{x}} \rightarrow_{\beta}^{*} (\!|\mathbf{t}_4|\!)_{\vec{x}} \cdots$$

et puisque $(\!|\mathbf{t}|\!)_{\vec{x}}$ est fortement normalisable par hypothèse, il existe k tel que pour tout $i \geq k$, $(\!|\mathbf{t}_i|\!)_{\vec{x}} = (\!|\mathbf{t}_{i+1}|\!)_{\vec{x}}$. Ainsi, par le Lemme IV.3.21, toutes les étapes de *Beta*-réduction après ce point sont internes, et par le Lemme IV.3.22, nous pouvons supposer que *toutes* les étapes de réduction après $\mathbf{t}_k$ sont internes. Il y a donc un sous-terme $\mathbf{u}[i/\mathbf{v}]$ de $\mathbf{t}_k$ avec une suite infinie de réductions partant de $\mathbf{v}$. Mais cette substitution a été créée par une étape de *Beta* précédente, ce qui contredit l'hypothèse de minimalité. □

Théorème IV.3.24 (*PSN*)

Si M est un λ-terme fortement normalisable avec $\mathsf{fv}(M) \subseteq \vec{x}$ alors $[\![M]\!]_{\vec{x}}$ est fortement normalisable dans λ_o.

Preuve. Conséquence directe des Propositions IV.3.23 et IV.2.15. □

IV.4 Calculs simplifiés

Nous avons maintenant un cadre général pour simuler le λ-calcul avec une notation de chaînes directrices. Cependant, notre but est de rechercher de nouvelles stratégies de réduction efficaces, et en particulier de fournir un modèle d'implantation pour la stratégie close du Chapitre II, nous pouvons donc renoncer à la complétude si nous pouvons gagner en efficacité et en simplicité, à condition que nous puissions encore au moins réduire les termes clos en forme normale de tête faible, ce qui est la condition minimale largement admise pour un λ-évaluateur, comme les compilateurs et interprètes de langages fonctionnels.

Nous présenterons ainsi deux nouveaux calculs λ_l et λ_c (avec les réductions $\rightsquigarrow_l$ et $\rightsquigarrow_c$ respectivement) sur les mêmes termes, qui seront des simplifications (spécialisations) de λ_o. Par définition, la relation suivante sera valide :

$$\rightsquigarrow_c \subset \rightsquigarrow_l \subset \rightsquigarrow_o^*$$

de sorte que, par exemple, λ_l et λ_c hériteront des propriétés de terminaison de λ_o.

D'un point de vue algorithmique, les règles de récriture de λ_o ne peuvent pas être considérées comme des opérations en temps constant, parce que nous devons accéder à des directeurs à des positions arbitraires dans les chaînes, et le calcul des nouvelles chaînes directrices semble *a priori* avoir besoin d'un temps linéaire en la taille des chaînes originales. Contrairement à λ_o, le calcul λ_l effectue seulement des étapes de réduction *locales*. En d'autres termes, λ_l est une restriction de λ_o où les règles de récriture effectuent seulement des modifications locales des chaînes directrices. C'est le sous-calcul le plus général de λ_o avec une telle propriété.

Le calcul λ_c est une restriction de λ_l qui implante la *stratégie de réduction close* du Chapitre II. C'est le calcul le plus intéressant d'une perspective d'implantation, comme le montre notre banc d'essai (voir la Section IV.9).

Nous définirons d'abord la syntaxe commune pour λ_l et λ_c, puis nous donnerons les deux ensembles de règles de récriture définissant ces calculs, en nous concentrant sur les propriétés de λ_c (les preuves pour λ_l sont similaires).

IV.4.1 Syntaxe

Définition IV.4.1 (*Termes simplifiés*)

Dans cette section, la syntaxe est un peu différente :

- Les directeurs sont $\curvearrowleft$, $\curvearrowright$, $\swarrow$ et $\downarrow$ seulement ($-$ n'est plus un directeur).
- Les prétermes sont définis par la grammaire suivante :

$$\mathbf{t} ::= \Box \mid (\lambda \mathbf{t})^\sigma \mid (\lambda^- \mathbf{t})^\sigma \mid (\mathbf{t}\ \mathbf{u})^\sigma \mid (\mathbf{t}[\mathbf{u}])^\sigma$$

  où $(\lambda^- \mathbf{t})^\sigma$ est une abstraction où la variable liée n'apparaît pas dans $\mathbf{t}$ et $(\mathbf{t}[\mathbf{u}])^\sigma$ est une substitution pour la première variable de $\mathbf{t}$ seulement.
- Les termes sont les prétermes satisfaisant les contraintes données dans la Définition IV.2.1 pour λ_o, en prenant $\mathcal{U} = \downarrow^*$ et $\mathcal{B} = (\curvearrowleft \mid \swarrow \mid \curvearrowright)^*$. Remarquons que maintenant les variables n'ont plus de chaîne directrice. De plus, les abstractions ont la contrainte additionnelle suivante :

$$(\lambda^- t^\rho)^\sigma \text{ est bien formé si } \sigma \in \mathcal{U} \text{ et } |\rho| = |\sigma|.$$

Le nouvel ensemble de termes peut être vu comme un sous-ensemble des λ_o-termes à l'aide de quelques abréviations :

Proposition IV.4.2 (*Equivalence syntaxique*)

L'application suivant des termes simplifiés vers les λ_o-termes est injective.

Nom	Réduction		
Beta	$((\lambda \mathbf{t})^{\epsilon}\ \mathbf{u})^{\sigma}$	$\rightsquigarrow_l$	$(\mathbf{t}[\mathbf{u}])^{\sigma}$
BetaE	$((\lambda^{-}\mathbf{t})^{\epsilon}\ u^{\epsilon})^{\epsilon}$	$\rightsquigarrow_l$	$\mathbf{t}$
Var	$(\Box[\mathbf{v}])^{\sigma}$	$\rightsquigarrow_l$	$\mathbf{v}$
App_1	$((\mathbf{t}\ \mathbf{u})^{\curvearrowleft\rho}[\mathbf{v}])^{\curvearrowright^m\cdot\curvearrowleft^n}$	$\rightsquigarrow_l$	$((\mathbf{t}[\mathbf{v}])^{\curvearrowright^m\cdot\curvearrowleft^{\lvert\rho\rvert_l}}\ \mathbf{u})^{\curvearrowleft^m\cdot\rho}$
App_2	$((\mathbf{t}\ \mathbf{u})^{\curvearrowright\rho}[\mathbf{v}])^{\curvearrowright^m\cdot\curvearrowleft^n}$	$\rightsquigarrow_l$	$(\mathbf{t}\ (\mathbf{u}[\mathbf{v}])^{\curvearrowright^m\cdot\curvearrowleft^{\lvert\rho\rvert_r}})^{\curvearrowright^m\cdot\rho}$
App_3	$((\mathbf{t}\ \mathbf{u})^{\swarrow\rho}[\mathbf{v}])^{\curvearrowright^m\cdot\curvearrowleft^n}$	$\rightsquigarrow_l$	$((\mathbf{t}[\mathbf{v}])^{\curvearrowright^m\cdot\curvearrowleft^{\lvert\rho\rvert_l}}\ (\mathbf{u}[\mathbf{v}])^{\curvearrowright^m\cdot\curvearrowleft^{\lvert\rho\rvert_r}})^{\swarrow^m\cdot\rho}$
Lam	$((\lambda \mathbf{t})^{\downarrow\rho}[\mathbf{v}])^{\sigma}$	$\rightsquigarrow_l$	$(\lambda(\mathbf{t}[\mathbf{v}])^{\sigma\cdot\curvearrowleft})^{\downarrow^{\lvert\sigma\rvert}}$
LamE	$((\lambda^{-}\mathbf{t})^{\downarrow\rho}[\mathbf{v}])^{\sigma}$	$\rightsquigarrow_l$	$(\lambda^{-}(\mathbf{t}[\mathbf{v}])^{\sigma})^{\downarrow^{\lvert\sigma\rvert}}$
Comp	$((\mathbf{t}[\mathbf{w}])^{\curvearrowright^{n+1}\cdot\curvearrowleft^m}[\mathbf{v}])^{\curvearrowright^p\cdot\curvearrowleft^q}$	$\rightsquigarrow_l$	$(\mathbf{t}[(\mathbf{w}[\mathbf{v}])^{\curvearrowright^p\cdot\curvearrowleft^n}])^{\curvearrowright^{p+n}\cdot\curvearrowleft^m}$

FIG. IV.4 – Calcul local ouvert

- $\beth(\Box) = \Box^{\downarrow}$
- $\beth((\lambda \mathbf{t})^{\sigma}) = (\lambda \beth(\mathbf{t}))^{\sigma}$
- $\beth((\lambda^{-} t^{\rho})^{\sigma}) = (\lambda \beth(t^{\rho-}))^{\sigma}$
- $\beth((\mathbf{t}\ \mathbf{u})^{\sigma}) = (\beth(\mathbf{t})\ \beth(\mathbf{u}))^{\sigma}$
- $\beth((\mathbf{t}[\mathbf{u}])^{\sigma}) = (\beth(\mathbf{t})[1/\beth(\mathbf{u})])^{\sigma}$

Nous nous permettrons donc souvent d'identifier les termes simplifiés avec des λ_o-termes (en omettant $\beth$).

IV.4.2 Calcul local ouvert

En regardant de plus près la règle *Beta*, nous remarquons que pour un rédex dont la fonction est close, nous générons une substitution pour la première (et seule) variable du corps de la fonction (qui était liée par l'abstraction). De plus, nous savons grâce au Chapitre II que restreindre la β-réduction aux fonctions closes permet encore d'atteindre la forme normale de tête faible pour les termes clos. Dans cette section nous allons donc décrire le calcul résultant de cette restriction qui simplifie grandement les règles. Ce calcul sera appelé *local ouvert* (λ_l) parce qu'il permet toujours aux substitutions ouvertes de se propager, même à l'intérieur des abstractions (en contraste avec le système de réduction close du Chapitre II). Cela se fait sans récritures globales si nous restreignons la syntaxe aux substitutions pour le premier directeur seulement.

DÉFINITION IV.4.3 (λ_l-*calcul*)

Les règles de réduction pour le calcul local ouvert sont données dans la Figure IV.4.

Même si les règles de ce système ne s'appliquent qu'à des termes dont les chaînes directrices ont une forme particulière, le système est tout de même capable d'évaluer des termes en toute généralité, grâce à la propriété suivante.

LEMME IV.4.4 (*Complétude de la réduction*)

Dans tout réduit d'un terme clos compilé, tout sous-terme de la forme $(\mathbf{t}[\mathbf{v}])^\sigma$ a une chaîne directrice $\sigma = \curvearrowright^m \cdot \curvearrowleft^n$ pour certains entiers m et n.

Preuve. Il suffit de remarquer que les règles de propagation génèrent des substitutions de cette forme, et que la règle *Beta* fait de même : si le terme $((\lambda\mathbf{t})^\epsilon\ \mathbf{u})^\sigma$ est bien formé (et il l'est par induction) alors σ est de la forme $\curvearrowright^n$. □

Nous remarquons que, dans ce calcul, nous permettons à des substitutions même ouvertes de traverser les abstractions, sans aucune étape de réduction globale. Il s'agit à l'évidence de l'une des plus grandes qualités de ce calcul comparativement à ceux basés sur les noms ou les indices de de Bruijn.

LEMME IV.4.5 (*Préservation de la bonne formation*)

Si $\mathbf{t}$ est un terme bien formé de λ_l et $\mathbf{t} \rightsquigarrow_l \mathbf{u}$ alors $\mathbf{u}$ est un terme bien formé de λ_l.

Preuve. Inspection immédiate des règles de récriture. □

PROPOSITION IV.4.6

$\rightsquigarrow_l \subset \rightsquigarrow_o^*$.

Preuve. Nous pouvons facilement vérifier que toutes les règles données dans la Définition IV.4.3, sauf *BetaE*, sont des cas particuliers de règles de λ_o en prenant en compte la syntaxe simplifiée des termes. La règle *BetaE* peut être simulée dans λ_o avec deux réductions : *Beta* suivie de *Erase*. □

Le Théorème IV.4.7 ci-dessous résume les propriétés de λ_l. Le calcul λ_l est confluent et préserve la normalisation forte. Cependant, il ne simule pas complètement la β-réduction. A la place, nous pouvons montrer qu'il peut être utilisé pour évaluer les λ-termes clos. C'est une conséquence du fait que λ_c, qui sera prouvé être une restriction de λ_l, peut calculer la forme normale de tête faible des termes clos (voir la section suivante).

THÉORÈME IV.4.7 (*Propriétés de λ_l*)

Confluence : λ_l est confluent.

PSN : λ_l préserve la normalisation forte.

Adéquation : λ_l peut évaluer les termes clos.

Preuve. La confluence se prouve facilement en adaptant la preuve de confluence de λ_c, qui est donnée dans la section suivante. La PSN vient du Théorème IV.3.24 (PSN pour λ_o) et de la Proposition IV.4.6. L'adéquation sera évidente à partir des Théorèmes IV.4.19 et IV.4.20 (adéquation pour λ_c) et de la Proposition IV.4.11 (voir la section suivante). □

IV.4.3 CALCUL CLOS

La notion de réduction close a été présentée au Chapitre II en utilisant un calcul nommé avec substitutions explicites où les β-réductions sont effectuées quand la fonction est close

Nom	Réduction		
Beta	$((\lambda \mathbf{t})^{\epsilon}\ \mathbf{u})^{\curvearrowright^n}$	$\rightsquigarrow_c$	$(\mathbf{t}[\mathbf{u}])^{\curvearrowright^n}$
BetaE	$((\lambda^{-} \mathbf{t})^{\epsilon}\ u^{\epsilon})^{\epsilon}$	$\rightsquigarrow_c$	$\mathbf{t}$
Var	$(\Box[\mathbf{v}])^{\curvearrowright^n}$	$\rightsquigarrow_c$	$\mathbf{v}$
App_1	$((\mathbf{t}\ \mathbf{u})^{\curvearrowleft^{\rho}}[v^{\epsilon}])^{\curvearrowleft^n}$	$\rightsquigarrow_c$	$((\mathbf{t}[v^{\epsilon}])^{\curvearrowleft^{\lvert\rho\rvert_l}}\ \mathbf{u})^{\rho}$
App_2	$((\mathbf{t}\ \mathbf{u})^{\curvearrowright^{\rho}}[v^{\epsilon}])^{\curvearrowleft^n}$	$\rightsquigarrow_c$	$(\mathbf{t}\ (\mathbf{u}[v^{\epsilon}])^{\curvearrowleft^{\lvert\rho\rvert_r}})^{\rho}$
App_3	$((\mathbf{t}\ \mathbf{u})^{\curlywedge^{\rho}}[v^{\epsilon}])^{\curvearrowleft^n}$	$\rightsquigarrow_c$	$((\mathbf{t}[v^{\epsilon}])^{\curvearrowleft^{\lvert\rho\rvert_l}}\ (\mathbf{u}[v^{\epsilon}])^{\curvearrowleft^{\lvert\rho\rvert_r}})^{\rho}$
Lam	$((\lambda \mathbf{t})^{\downarrow\rho}[v^{\epsilon}])^{\curvearrowleft^n}$	$\rightsquigarrow_c$	$(\lambda(\mathbf{t}[v^{\epsilon}])^{\curvearrowleft^{n+1}})^{\downarrow^n}$
LamE	$((\lambda^{-} \mathbf{t})^{\downarrow\rho}[v^{\epsilon}])^{\curvearrowleft^n}$	$\rightsquigarrow_c$	$(\lambda^{-}(\mathbf{t}[v^{\epsilon}])^{\curvearrowleft^n})^{\downarrow^n}$
Comp	$((\mathbf{t}[\mathbf{w}])^{\curvearrowright^{n+1}}[v^{\epsilon}])^{\curvearrowleft^n}$	$\rightsquigarrow_c$	$(\mathbf{t}[(\mathbf{w}[v^{\epsilon}])^{\curvearrowleft^n}])^{\curvearrowright^n}$

FIG. IV.5 – Calcul clos

(comme ci-dessus) et les substitutions sont propagées à travers les abstractions seulement si elles sont closes, ce qui est crucial pour éviter l'α-conversion dans un cadre nommé. Ces restrictions sont exprimées par des règles de récriture avec des conditions externes sur les variables libres, et l'internalisation de ces conditions est en fait la motivation principale pour l'introduction des chaînes directrices. Nous pouvons facilement dériver un calcul pour la réduction close (λ_c) en ajoutant des restrictions aux règles de λ_l (voir la Définition IV.4.3) comme suit : *Beta*, *BetaE*, *Var* sont identiques ; dans chaque autre règle nous forçons la substitution $\mathbf{v}$ à être close, c'est-à-dire à avoir une chaîne vide (ϵ). Alors, dans $App_{1,2,3}$, $m = 0$, et dans *Comp*, $m = p = 0$ et $n = q$. Cela nous amène ainsi au système de récriture très simple suivant.

DÉFINITION IV.4.8 (*λ_c-calcul*)

Les règles de réduction pour le calcul clos sont données dans la Figure IV.5.

Ce système est complet parce qu'une propriété similaire au Lemme IV.4.4 est valide ici :

LEMME IV.4.9 (*Complétude de la réduction*)

Dans tout réduit d'un terme clos compilé, si un sous-terme est de la forme $(\mathbf{t}[\mathbf{v}])^{\sigma}$ alors soit $\sigma = {\curvearrowleft^n}$, soit $\sigma = {\curvearrowright^n}$ pour un certain entier n.

Preuve. La règle *Beta* crée une substitution annotée par $\curvearrowright^n$, les autre règles génèrent seulement des substitutions avec $\curvearrowleft^n$ pour chaîne. □

On peut remarquer que le choix entre $\curvearrowleft^n$ et $\curvearrowright^n$ est toujours évident à partir du contexte (par bonne formation). Par exemple, dans la règle *Comp*, nous réduisons un terme de la forme $((\mathbf{t}[\mathbf{w}])^{\curvearrowright^{n+1}}[v^{\epsilon}])^{\curvearrowleft^m}$, alors il est facile de voir que $m = n$.

LEMME IV.4.10 (*Préservation de la bonne formation*)

Si t^{σ} est un terme bien formé de λ_c et $t^{\sigma} \rightsquigarrow_c u^{\tau}$, alors u^{τ} est un terme bien formé de λ_c. De plus, $|\sigma| = |\tau|$.

Preuve. Inspection immédiate des règles de récriture. □

PROPOSITION IV.4.11

$\rightsquigarrow_c \subset \rightsquigarrow_l \subset \rightsquigarrow_o^*$.

Preuve. Puisque les règles de la Définition IV.4.8 sont des instances particulières des règles de λ_l (Définition IV.4.3), la relation de récriture $\rightsquigarrow_c$ est incluse dans $\rightsquigarrow_l$. La dernière inclusion a été établie dans la Proposition IV.4.6. □

Nous allons montrer que $\rightsquigarrow_c$ est confluent sur les λ_c-termes, préserve la normalisation forte et permet d'implanter les stratégies d'évaluation en appel par valeur et en appel par nom pour les λ-termes clos. Pour cela, nous montrons d'abord que les règles de propagation (c'est-à-dire toutes les règles sauf *Beta* et *BetaE*) terminent, sont confluentes et permettent d'implanter la méta-opération de substitution du λ-calcul, pourvu que les substitutions soient closes. L'ensemble des règles de propagation sera noté $\mathcal{P}_c$.

PROPOSITION IV.4.12 (*Terminaison et confluence de* $\mathcal{P}_c$)

Toutes les suites de réduction de λ_c en utilisant seulement les règles de $\mathcal{P}_c$ sont finies. De plus, $\mathcal{P}_c$ est localement confluent (et donc confluent).

Preuve. La terminaison est une conséquence directe de la Proposition IV.3.5 et du fait que $\rightsquigarrow_{\mathcal{P}_c}^* \subset \rightsquigarrow_{\mathcal{P}}^*$. La confluence locale est prouvée en montrant que les paires critiques sont joignables. Il n'y en a qu'une seule, entre *Comp* et *Var* (les conditions sur les chaînes directrices préviennent les superpositions entre *Comp* et les autres règles de propagation).

$$\begin{array}{ccc} ((\Box[\mathbf{v}])\curvearrowright^{n+1}[u^\epsilon])\curvearrowleft^n & \xrightarrow{Comp} & (\Box[(\mathbf{v}[u^\epsilon])\curvearrowleft^n])\curvearrowright^n \\ \downarrow{\scriptstyle Var} & & \downarrow{\scriptstyle Var} \\ (\mathbf{v}[u^\epsilon])\curvearrowleft^n & = & (\mathbf{v}[u^\epsilon])\curvearrowleft^n \end{array}$$

Puisque les règles de propagation terminent, le système est confluent par le Lemme de Newman I.1.4.

□

Puisque $\mathcal{P}_c$ est confluent et termine, le système définit une fonction qui associe à chaque λ_c-terme $\mathbf{t}$ sa forme normale unique, dénotée $\mathcal{P}_c(\mathbf{t})$.

LEMME IV.4.13 (*Substitutions closes*)

Soient M et N des λ-termes tels que $x \in \mathsf{fv}(M) \subseteq \vec{x}$ et $\mathsf{fv}(N) = \varnothing$. Soient $[\![M]\!]_{\vec{x}} = \mathbf{t}$, $[\![N]\!] = u^\epsilon$, $[\![M\{x := N\}]\!]_{\vec{y}} = v^\sigma$, où $\vec{y} = \vec{x}\backslash\{x\}$. Alors $\mathcal{P}_c((\mathbf{t}[u^\epsilon])^\sigma) = v^\sigma$.

Preuve. Par induction sur M. Si M est une variable (dans ce cas, c'est forcément x), alors sa compilation est simplement □ et en utilisant la règle *Var*, $\Box[u^\epsilon] \rightsquigarrow_c u^\epsilon = [\![M\{x := N\}]\!]_{\vec{y}}$ comme demandé. Si M est une application, alors une des règles App_1, App_2, App_3 s'applique, et le résultat suit directement par induction. Si M est une abstraction, alors soit *Lam*, soit *LamE* s'applique et à nouveau le résultat suit par induction. □

Comme conséquence, nous déduisons que même avec les restrictions imposées par les règles de λ_c, les substitutions closes ne restent pas bloquées : si une substitution close est crée, elle sera propagée complètement.

Le lemme suivant montre que les règles *Beta* et *BetaE* donnent aussi lieu a une relation confluente, que nous dénotons $\rightsquigarrow_B$. Dans la preuve, nous utilisons une relation auxiliaire $\Rightarrow$, qui effectue les étapes $\rightsquigarrow_B$ en parallèle. Elle est définie par induction.

DÉFINITION IV.4.14 (*Beta parallèle*)

Soit $\Rightarrow$ la relation de récriture sur les λ_c-termes définie inductivement comme suit.

1. $\mathbf{t} \Rightarrow \mathbf{t}$,
2. $((\lambda\mathbf{t})^\epsilon\,\mathbf{u})^{\frown^n} \Rightarrow (\mathbf{t}'[\mathbf{u}'])^{\frown^n}$ si $\mathbf{t} \Rightarrow \mathbf{t}'$ et $\mathbf{u} \Rightarrow \mathbf{u}'$,
3. $((\lambda^-\mathbf{t})^\epsilon\,u^\epsilon)^\epsilon \Rightarrow \mathbf{t}'$ si $\mathbf{t} \Rightarrow \mathbf{t}'$,
4. $(\lambda\mathbf{t})^\sigma \Rightarrow (\lambda\mathbf{t}')^\sigma$ si $t \Rightarrow t'$,
5. $(\mathbf{t}\ \mathbf{u})^\sigma \Rightarrow (\mathbf{t}'\,\mathbf{u}')^\sigma$ si $\mathbf{t} \Rightarrow \mathbf{t}'$ et $\mathbf{u} \Rightarrow \mathbf{u}'$,
6. $(\mathbf{t}[\mathbf{v}])^\sigma \Rightarrow (\mathbf{t}'[\mathbf{v}'])^\sigma$ si $\mathbf{t} \Rightarrow \mathbf{t}'$ et $\mathbf{v} \Rightarrow \mathbf{v}'$.

LEMME IV.4.15 (*Confluence de la* Beta-*réduction*)

$\rightsquigarrow_B$ est confluent.

Preuve. Nous montrons que $\Rightarrow$ est fortement confluent et puisque clairement $\rightsquigarrow_B \subseteq \Rightarrow$ et $\Rightarrow^*=\rightsquigarrow_B^*$, nous obtenons la confluence de $\rightsquigarrow_B$. Supposons que $\mathbf{t} \Rightarrow \mathbf{u}$ et $\mathbf{t} \Rightarrow \mathbf{v}$ ($\mathbf{u} \neq \mathbf{v}$). Nous montrons que $\exists\mathbf{w}$ tel que $\mathbf{u} \Rightarrow \mathbf{w}$ et $\mathbf{v} \Rightarrow \mathbf{w}$ par induction sur $\mathbf{t} \Rightarrow \mathbf{u}$. La preuve est standard. Le cas $\mathbf{t} \Rightarrow \mathbf{t}$ est trivial, en prenant $\mathbf{w} = \mathbf{v}$. Nous donnons les détails pour $((\lambda\mathbf{t})^\epsilon\,\mathbf{u})^{\frown^n} \Rightarrow (\mathbf{t}'[\mathbf{u}'])^{\frown^n}$, où $\mathbf{t} \Rightarrow \mathbf{t}'$ et $\mathbf{u} \Rightarrow \mathbf{u}'$. Dans ce cas, les seules alternatives sont :

- $\mathbf{v} = ((\lambda\mathbf{t})^\epsilon\,\mathbf{u})^{\frown^n}$ auquel cas nous prenons $\mathbf{w} = (\mathbf{t}'[\mathbf{u}'])^{\frown^n}$, ou bien
- $\mathbf{v} = ((\lambda\mathbf{t}'')^\epsilon\,\mathbf{u}'')^{\frown^n}$, où $\mathbf{t} \Rightarrow \mathbf{t}''$ et $\mathbf{u} \Rightarrow \mathbf{u}''$. Par induction, il existe $\mathbf{w}'$ et $\mathbf{w}''$ tels que $\mathbf{t}' \Rightarrow \mathbf{w}'$, $\mathbf{t}'' \Rightarrow \mathbf{w}'$, $\mathbf{u}' \Rightarrow \mathbf{w}''$, $\mathbf{u}'' \Rightarrow \mathbf{w}''$. Ainsi nous prenons $\mathbf{w} = (\mathbf{w}'[\mathbf{w}''])^{\frown^n}$.

Le cas $((\lambda^-\mathbf{t})^\epsilon\,u^\epsilon)^\epsilon \Rightarrow \mathbf{t}'$ où $\mathbf{t} \Rightarrow \mathbf{t}'$ est similaire. Tous les autre cas suivent directement par induction. □

Ensuite nous prouvons la commutation de $\rightsquigarrow_B$ et $\rightsquigarrow_{\mathcal{P}_c}$. Le Lemme de Hindley-Rosen I.1.7 dit que si deux relations de récriture confluentes sont indépendantes (commutent), alors leur union est confluente. Puisque $\rightsquigarrow_c=\rightsquigarrow_B \cup \rightsquigarrow_{\mathcal{P}_c}$ et nous venons de montrer que les deux relations sont confluentes, la commutation implique la confluence de $\rightsquigarrow_c$.

LEMME IV.4.16 (*Commutation*)

Si $\mathbf{t} \rightsquigarrow_{\mathcal{P}_c}^* \mathbf{a}$ et $\mathbf{t} \rightsquigarrow_B^* \mathbf{b}$, alors il existe $\mathbf{c}$ tel que $\mathbf{a} \rightsquigarrow_B^* \mathbf{c}$ et $\mathbf{b} \rightsquigarrow_{\mathcal{P}_c}^* \mathbf{c}$.

Preuve. Nous utilisons encore la relation de réduction parallèle $\Rightarrow$ (Définition IV.4.14). Puisque $\Rightarrow^*$ coïncide avec $\rightsquigarrow_B^*$, il suffit de prouver que si $\mathbf{b} \leftarrow \mathbf{t} \rightsquigarrow_{\mathcal{P}_c} \mathbf{a}$, alors il existe un terme $\mathbf{c}$ tel que $\mathbf{b} \rightsquigarrow_{\mathcal{P}_c} \mathbf{c} \leftarrow \mathbf{a}$. De cette façon nous pouvons clore le diagramme : $\mathbf{t} \rightsquigarrow_{\mathcal{P}_c}^* \mathbf{a}$ et $\mathbf{t} \rightsquigarrow_B^* \mathbf{b}$ (ce qui est équivalent à $\mathbf{b}\ {}^*\!\leftarrow \mathbf{t} \rightsquigarrow_{\mathcal{P}_c}^* \mathbf{a}$), par induction sur la longueur de la dérivation $\mathbf{t} \Rightarrow^* \mathbf{b}$.

Nous procédons par induction sur $\mathbf{t} \Rightarrow \mathbf{b}$. Nous distinguons les cas suivant :

1. $\mathbf{t} = \mathbf{b}$: alors nous prenons $\mathbf{c} = \mathbf{a}$.
2. $((\lambda \mathbf{t})^{\epsilon}\, \mathbf{u})^{\curvearrowright^n} \Rightarrow \mathbf{b} = (\mathbf{t}'[\mathbf{u}'])^{\curvearrowright^n}$ où $\mathbf{t} \Rightarrow \mathbf{t}'$ et $\mathbf{u} \Rightarrow \mathbf{u}'$: puisqu'aucune règle de $\mathcal{P}_c$ ne s'applique à la racine, ce doit être ou bien $\mathbf{t} \rightsquigarrow_{\mathcal{P}_c} \mathbf{t}''$, ou bien $\mathbf{u} \rightsquigarrow_{\mathcal{P}_c} \mathbf{u}''$. Dans le premier cas, par induction, il existe un terme $\mathbf{t}'''$ tel que $\mathbf{t}' \rightsquigarrow_{\mathcal{P}_c} \mathbf{t}'''$ et $\mathbf{t}'' \Rightarrow \mathbf{t}'''$, donc nous pouvons prendre $\mathbf{c} = (\mathbf{t}'''[\mathbf{u}'])^{\curvearrowright^n}$. Dans le second cas, par induction, il existe $\mathbf{u}'''$ tel que $\mathbf{u}' \rightsquigarrow_{\mathcal{P}_c} \mathbf{u}'''$ et $\mathbf{u}'' \Rightarrow \mathbf{u}'''$, donc nous pouvons prendre $\mathbf{c} = (\mathbf{t}'[\mathbf{u}'''])^{\curvearrowright^n}$.
3. Le cas $((\lambda^{-}\mathbf{t})^{\epsilon}\, u^{\epsilon})^{\epsilon} \Rightarrow \mathbf{b} = \mathbf{t}'$ où $\mathbf{t} \Rightarrow \mathbf{t}'$ est similaire au cas ci-dessus.
4. $(\mathbf{t}[\mathbf{v}])^{\sigma} \Rightarrow (\mathbf{t}'[\mathbf{v}'])^{\sigma}$ où $\mathbf{t} \Rightarrow \mathbf{t}'$ et $\mathbf{v} \Rightarrow \mathbf{v}'$: nous distinguons deux cas selon la position de l'étape de $\mathcal{P}_c$-réduction $(\mathbf{t}[\mathbf{v}])^{\sigma} \rightsquigarrow_{\mathcal{P}_c} \mathbf{a}$.
   - Si elle ne s'applique pas à la racine, alors la propriété suit par induction.
   - Si elle s'applique à la racine : alors il y a une substitution θ et une règle $l \to r$ de $\mathcal{P}_c$ telles que $(\mathbf{t}[\mathbf{v}])^{\sigma} = l\theta$ et $\mathbf{a} = r\theta$.
     - Si tous les $\rightsquigarrow_B$-rédex contractés lors de la réduction $(\mathbf{t}[\mathbf{v}])^{\sigma} \Rightarrow (\mathbf{t}'[\mathbf{v}'])^{\sigma} = \mathbf{b}$ sont sous des variables de l (c'est-à-dire qu'elles sont dans θ), alors ces variables sont identifiées de façon univoque (puisque l est linéaire gauche) et nous pouvons donc définir une substitution θ' telle que $\theta \Rightarrow \theta'$ et le diagramme commute : $(\mathbf{t}[\mathbf{v}])^{\sigma} = l\theta \rightsquigarrow_{\mathcal{P}_c} r\theta = \mathbf{a} \Rightarrow r\theta' = \mathbf{c}$ et $(\mathbf{t}[\mathbf{v}])^{\sigma} = l\theta \Rightarrow l\theta' = \mathbf{b} \rightsquigarrow_{\mathcal{P}_c} r\theta' = \mathbf{c}$.
     - S'il y a un $\rightsquigarrow_B$-rédex à la racine de $\mathbf{t}$ dans $(\mathbf{t}[\mathbf{v}])^{\sigma}$, alors nous avons une paire critique entre la $\mathcal{P}_c$-règle appliquée à la racine de $(\mathbf{t}[\mathbf{v}])^{\sigma}$ et la règle *Beta* ou *BetaE* appliquée à la racine de $\mathbf{t}$. Dans ce cas la $\mathcal{P}_c$-règle appliquée doit être App_2 (elle ne peut pas être App_1 ou App_3 à cause des restrictions : la fonction a ϵ pour chaîne directrice). Alors le diagramme commute comme suit : nous avons

$$(\mathbf{t}[\mathbf{v}])^{\sigma} = (((\lambda \mathbf{w})^{\epsilon}\, \mathbf{u})^{\curvearrowright^{n+1}}[v^{\epsilon}])^{\curvearrowleft^n} \rightsquigarrow_{App_2} ((\lambda \mathbf{w})^{\epsilon}\, (\mathbf{u}[v^{\epsilon}])^{\curvearrowleft^n})^{\curvearrowright^n} = \mathbf{a}$$

et

$$(\mathbf{t}[\mathbf{v}])^{\sigma} = (((\lambda \mathbf{w})^{\epsilon}\, \mathbf{u})^{\curvearrowright^{n+1}}[v^{\epsilon}])^{\curvearrowleft^n} \Rightarrow ((\mathbf{w}'[\mathbf{u}'])^{\curvearrowright^{n+1}}[\mathbf{v}'])^{\curvearrowleft^n} = \mathbf{b}$$

où $\mathbf{w} \Rightarrow \mathbf{w}', \mathbf{u} \Rightarrow \mathbf{u}', \mathbf{v} \Rightarrow \mathbf{v}'$, ainsi

$$\mathbf{a} \Rightarrow (\mathbf{w}'[(\mathbf{u}'[\mathbf{v}'])^{\curvearrowleft^n}])^{\curvearrowright^n} = \mathbf{c}$$

et

$$\mathbf{b} \rightsquigarrow_{Comp} (\mathbf{w}'[(\mathbf{u}'[\mathbf{v}'])^{\curvearrowleft^n}])^{\curvearrowright^n} = \mathbf{c}.$$

5. Dans l'autre cas, la propriété suit directement par induction.

Ceci conclut la preuve. □

THÉORÈME IV.4.17 (*Confluence*)
| λ_c est confluent.

Preuve. Conséquence de la Proposition IV.4.12, du Lemme IV.4.15 et du Lemme IV.4.16, grâce au Lemme de Hindley-Rosen I.1.7. □

THÉORÈME IV.4.18 (*PSN*)

λ_c préserve la normalisation forte.

Preuve. Conséquence du Théorème IV.3.24 (PSN pour λ_o) et de la Proposition IV.4.11. □

Les restrictions imposées sur les règles de λ_c nous permettent encore de simuler les stratégies d'évaluation habituelles pour des λ-termes clos. Nous allons montrer que les réductions en forme normale de tête faible en appel par valeur et appel par nom peuvent être implantées dans λ_c.

THÉORÈME IV.4.19 (*Evaluation en appel par nom*)

Si M est un λ-terme clos et $M \Downarrow V$ par la stratégie d'appel par nom, alors $[\![M]\!] \rightsquigarrow_c^* [\![V]\!]$.

Preuve. Par induction sur la dérivation de $M \Downarrow V$. Si M est une forme normale de tête faible, alors le théorème suit trivialement. Sinon, c'est une application $(M\,N)$ et $[\![M\,N]\!] = ([\![M]\!]\,[\![N]\!])^\epsilon$ (puisque $\mathsf{fv}(M\,N) = \varnothing$). Par induction, puisque $M \Downarrow \lambda x.M'$, nous savons que $[\![M]\!] \rightsquigarrow_c^* [\![\lambda x.M']\!]$. Maintenant, il y a une alternative.

- $[\![\lambda x.M']\!] = (\lambda [\![M']\!]_{[x]})^\epsilon$ si $x \in \mathsf{fv}(M')$. Alors $((\lambda [\![M']\!]_{[x]})^\epsilon [\![N]\!])^\epsilon \rightsquigarrow_c ([\![M']\!]_{[x]}[[\![N]\!]])^\epsilon$. Par le Lemme IV.4.13, $([\![M']\!]_{[x]}[[\![N]\!]])^\epsilon \rightsquigarrow_c^* [\![M'\{x := N\}]\!]$ et par induction ceci se réduit en $[\![V]\!]$ comme requis.
- $[\![\lambda x.M']\!] = (\lambda^- [\![M']\!])^\epsilon$ si $x \notin \mathsf{fv}(M')$. Alors $((\lambda^- [\![M']\!])^\epsilon [\![N]\!])^\epsilon \rightsquigarrow_c [\![M']\!] = [\![M'\{x := N\}]\!]$ et par induction ceci se réduit en $[\![V]\!]$ comme requis.

□

THÉORÈME IV.4.20 (*Evaluation en appel par valeur*)

Si M est un λ-terme clos et $M \Downarrow V$ par la stratégie d'appel par valeur, alors $[\![M]\!] \rightsquigarrow_c^* [\![V]\!]$.

Preuve. Similaire au Théorème IV.4.19. Dans le cas d'une application $(M\,N)$ où $M \Downarrow \lambda x.M'$ et $N \Downarrow V'$, nous obtenons : $[\![(M\,N)]\!] = ([\![M]\!]\,[\![N]\!])^\epsilon$ (puisque $\mathsf{fv}(M\,N) = \varnothing$), et par induction, $[\![M]\!] \rightsquigarrow_c^* [\![\lambda x.M']\!]$, $[\![N]\!] \rightsquigarrow_c^* [\![V']\!]$. A nouveau, il y a deux cas à considérer, qui sont identiques à ceux de la preuve du Théorème IV.4.19 ci-dessus. □

Le système a plusieurs avantages comme fondation pour une implantation d'un évaluateur du λ-calcul :

- les substitutions closes peuvent être propagées à travers les abstractions, ce qui autorise davantage de partage de calcul que dans les calculs faibles standard,
- il interdit la copie de termes ouverts, ce qui assure que nous ne dupliquons jamais un rédex potentiel.

De plus, la forme des règles de récriture montre que dans la plupart des cas, nous n'avons pas besoin de représenter les chaînes directrices par des structures complexes. Par exemple, la chaîne directrice d'une substitution peut être représentée simplement par un entier relatif, grâce au Lemme IV.4.9. Dans le même esprit, l'information sur $|\rho|_l$ et $|\rho|_r$ peut être maintenue à chaque étape. Le système peut donc être implanté d'une façon réaliste et efficace, en donnant à chaque étape de récriture un coût algorithmique constant. Ce point est détaillé à la section suivante.

Nom	Réduction		
Beta	$((\lambda \mathbf{t})^0\ \mathbf{u})^{\sigma,n,0}$	$\rightsquigarrow$	$(\mathbf{t}[\mathbf{u}])^{n,0}$
BetaE	$((\lambda^- \mathbf{t})^0\ \mathbf{u})^{\epsilon,0,0}$	$\rightsquigarrow$	$\mathbf{t}$
Var	$(\Box[\mathbf{v}])^{m,n}$	$\rightsquigarrow$	$\mathbf{v}$
App_1	$((\mathbf{t}\ \mathbf{u})^{\curvearrowleft\rho,l,r}[\mathbf{v}])^{m,n}$	$\rightsquigarrow$	$((\mathbf{t}[\mathbf{v}])^{m,l-1}\ \mathbf{u})^{\curvearrowleft^m\cdot\rho,m+l-1,r}$
App_2	$((\mathbf{t}\ \mathbf{u})^{\curvearrowright\rho,l,r}[\mathbf{v}])^{m,n}$	$\rightsquigarrow$	$(\mathbf{t}\ (\mathbf{u}[\mathbf{v}])^{m,r-1})^{\curvearrowright^m\cdot\rho,l,m+r-1}$
App_3	$((\mathbf{t}\ \mathbf{u})^{\curlywedge\rho,l,r}[\mathbf{v}])^{m,n}$	$\rightsquigarrow$	$((\mathbf{t}[\mathbf{v}])^{m,l-1}\ (\mathbf{u}[\mathbf{v}])^{m,r-1})^{\curlywedge^m\cdot\rho,m+l-1,m+r-1}$
Lam	$((\lambda \mathbf{t})^{p+1}[\mathbf{v}])^{m,n}$	$\rightsquigarrow$	$(\lambda(\mathbf{t}[\mathbf{v}])^{m,n+1})^{m+n}$
LamE	$((\lambda^- \mathbf{t})^{p+1}[\mathbf{v}])^{m,n}$	$\rightsquigarrow$	$(\lambda^-(\mathbf{t}[\mathbf{v}])^{m,n})^{m+n}$
Comp	$((\mathbf{t}[\mathbf{w}])^{n+1,m}[\mathbf{v}])^{p,q}$	$\rightsquigarrow$	$(\mathbf{t}[(\mathbf{w}[\mathbf{v}])^{p,n}])^{p+n,m}$

FIG. IV.6 – Implantation des règles ouvertes locales

IV.5 Implantation réaliste

Les calculs ouvert local et clos peuvent s'exprimer de façon plus proche d'une implantation, en explicitant les opérations sur les chaînes directrices. Ainsi, il est clair que chaque pas de réduction prend un temps algorithmiquement constant. Il s'agit essentiellement de remplacer les manipulation de listes par des opérations sur des entiers.

IV.5.1 Réduction ouverte locale

Nous changeons la syntaxe des termes : désormais, les abstractions sont seulement annotées par un entier, les substitutions par deux entiers, les applications par une chaîne directrice et deux entiers. On traduit ainsi les termes :

Terme initial	Terme traduit
$\Box$	$\Box$
$(\lambda^- \mathbf{t})^\sigma$	$(\lambda^- \mathbf{t})^{\lvert\sigma\rvert}$
$(\lambda \mathbf{t})^\sigma$	$(\lambda \mathbf{t})^{\lvert\sigma\rvert}$
$(\mathbf{t}\ \mathbf{u})^\sigma$	$(\mathbf{t}\ \mathbf{u})^{\sigma,\lvert\sigma\rvert_l,\lvert\sigma\rvert_r}$
$(\mathbf{t}[\mathbf{u}])^{\curvearrowright^m\cdot\curvearrowleft^n}$	$(\mathbf{t}[\mathbf{u}])^{m,n}$

En ce qui concerne la dernière ligne du tableau, il faut se souvenir que la chaîne directrice d'une substitution est en effet toujours de cette forme dans le cas d'une réduction avec les règles ouvertes locales à partir d'un terme compilé (Lemme IV.4.4), cas dans lequel nous nous plaçons. Il est alors facile de traduire les règles ouvertes locales, comme indiqué à la Figure IV.6.

Les règles App_i demandent de construire une chaîne directrice en la faisant précéder de n fois un même directeur. Cette opération se fait en temps constant en adaptant la structure de représentation des chaînes directrices : au lieu d'une simple liste de directeurs, on utilise une liste de couples (directeur, entier) où l'entier dénote le nombre d'occurrence du directeur. On adapte alors en conséquence les opérations d'interrogation de la tête et de la queue d'une chaîne directrice.

IV.5.2 Réduction close

On peut procéder de la même manière avec la machine close. Nous ne donnons que la traduction des termes, la traduction des règles étant alors très facile.

Terme initial	Terme traduit
$\square$	$\square$
$(\lambda^- \mathbf{t})^\sigma$	$(\lambda^- \mathbf{t})^{\lvert\sigma\rvert}$
$(\lambda \mathbf{t})^\sigma$	$(\lambda \mathbf{t})^{\lvert\sigma\rvert}$
$(\mathbf{t}\ \mathbf{u})^\sigma$	$(\mathbf{t}\ \mathbf{u})^{\sigma,\lvert\sigma\rvert_l,\lvert\sigma\rvert_r}$
$(\mathbf{t}[\mathbf{u}])^{\curvearrowleft^n}$	$(\mathbf{t}[\mathbf{u}])^n$
$(\mathbf{t}[\mathbf{u}])^{\curvearrowright^n}$	$(\mathbf{t}[\mathbf{u}])^{-n}$

La seule différence par rapport au paragraphe précédent est que l'on peut simplement coder la chaîne directrice d'une substitution par un entier relatif, grâce au Lemme IV.4.9. Pour l'application, on peut aussi économiser un entier en gardant seulement la longueur de la chaîne σ, au lieu se $|\sigma|_l$ et $|\sigma|_r$, et, lorsque cette information est requise, la retrouver grâce à la longueur de la chaîne du terme de gauche ou de droite.

On peut donc adapter les machines abstraites de la Section IV.7 de façon à ce que chaque pas de réduction soit effectivement réalisé en temps algorithmiquement constant et relativement faible (les seules opérations, en dehors de la construction des termes, sont des additions et au plus deux manipulations élémentaires de listes par pas), et où la représentation des données prend moins de place en mémoire.

En revanche, il ne paraît pas envisageable d'étendre le procédé au calcul ouvert, car, par exemple, les règles App_i demandent de parcourir entièrement les chaînes directrices. Certains pas de réductions ont donc, *a priori*, un coût linéaire en le nombre de variables libres du terme.

IV.6 Système de types pour les calculs avec chaînes directrices

Nous présentons dans cette section un système de types commun aux calculs définis ci-dessus et qui bénéficie du même genre de propriétés que le λ-calcul simplement typé habituel. Nous présentons le système avec la syntaxe de λ_o, mais il est évidemment aussi valide pour λ_l et λ_c grâce à la Proposition IV.4.2. Les types sont les types simples du λ-calcul : des variables de type $A, B \ldots$ et des types fonction $A \to B$. Nous ajoutons aussi une constante générique $\star^\sigma$ de type $\top$ (bien formée si et seulement si $\sigma = -^n$ pour un certain n). Les jugements de typage seront de la forme $\Gamma \vdash \mathbf{t} : A$ où le contexte Γ est une liste ordonnée de types.

DÉFINITION IV.6.1 (*Termes typés*)

A chaque terme $\mathbf{t}$, on assigne un type dans un contexte donné : $\Gamma \vdash \mathbf{t} : A$, comme indiqué par l'ensemble de règles suivant. Un terme est typable s'il existe un contexte tel qu'il soit typable dans ce contexte. Les règles de typage utilisent deux relations auxiliaires définies ci-dessous.

$$\frac{\Gamma \overset{\sigma}{\sim} \varnothing}{\Gamma \vdash \star^{\sigma} : \top}\,(Const) \qquad \frac{\Gamma \overset{\sigma}{\sim} A}{\Gamma \vdash \square^{\sigma} : A}\,(Ax)$$

$$\frac{\Gamma', A \vdash \mathbf{t} : B \quad \Gamma \overset{\sigma}{\sim} \Gamma'}{\Gamma \vdash (\lambda \mathbf{t})^{\sigma} : A \to B}\,(Abs)$$

$$\frac{\Gamma_1 \vdash \mathbf{t} : A \to B \qquad \Gamma_2 \vdash \mathbf{u} : A \qquad \Gamma \overset{\sigma}{\multimap} \begin{cases} \Gamma_1 \\ \Gamma_2 \end{cases}}{\Gamma \vdash (\mathbf{t}\ \mathbf{u})^{\sigma} : B}\,(App)$$

$$\frac{\Gamma_1\langle i/A\rangle \vdash \mathbf{t} : B \qquad \Gamma_2 \vdash \mathbf{u} : A \qquad \Gamma \overset{\sigma}{\multimap} \begin{cases} \Gamma_1 \\ \Gamma_2 \end{cases}}{\Gamma \vdash (\mathbf{t}[i/\mathbf{u}])^{\sigma} : B}\,(Sub)$$

où $\Gamma\langle i/A\rangle = A_1, \ldots, A_{i-1}, A, A_i, \ldots, A_n$ si $\Gamma = A_1, \ldots, A_n$.

Remarquons qu'il n'y a pas de règles structurelles, dans la mesure où le type d'une variable donnée est toujours à une position connue dans le contexte. Dans les règles de typage nous avons utilisé deux relations auxiliaires. La première ($\Gamma \overset{\sigma}{\sim} \Gamma'$) s'assure que les bons types sont effacés de Γ en fonction de σ. La seconde s'assure qu'un contexte se divise de façon adéquate selon une chaîne directrice, ce que nous notons $\Gamma \overset{\sigma}{\multimap} \begin{cases} \Gamma_1 \\ \Gamma_2 \end{cases}$.

$$\frac{}{\varnothing \overset{\epsilon}{\sim} \varnothing} \qquad \frac{\Gamma \overset{\sigma}{\sim} \Gamma'}{A, \Gamma \overset{\downarrow\sigma}{\sim} A, \Gamma'} \qquad \frac{\Gamma \overset{\sigma}{\sim} \Gamma'}{A, \Gamma \overset{-\sigma}{\sim} \Gamma'}$$

$$\frac{}{\varnothing \overset{\epsilon}{\multimap} \begin{cases} \varnothing \\ \varnothing \end{cases}} \qquad \frac{\Gamma \overset{\sigma}{\multimap} \begin{cases} \Gamma_1 \\ \Gamma_2 \end{cases}}{A, \Gamma \overset{\lambda\sigma}{\multimap} \begin{cases} A, \Gamma_1 \\ A, \Gamma_2 \end{cases}} \qquad \frac{\Gamma \overset{\sigma}{\multimap} \begin{cases} \Gamma_1 \\ \Gamma_2 \end{cases}}{A, \Gamma \overset{-\sigma}{\multimap} \begin{cases} \Gamma_1 \\ \Gamma_2 \end{cases}}$$

$$\frac{\Gamma \overset{\sigma}{\multimap} \begin{cases} \Gamma_1 \\ \Gamma_2 \end{cases}}{A, \Gamma \overset{\curvearrowright\sigma}{\multimap} \begin{cases} \Gamma_1 \\ A, \Gamma_2 \end{cases}} \qquad \frac{\Gamma \overset{\sigma}{\multimap} \begin{cases} \Gamma_1 \\ \Gamma_2 \end{cases}}{A, \Gamma \overset{\curvearrowleft\sigma}{\multimap} \begin{cases} A, \Gamma_1 \\ \Gamma_2 \end{cases}}$$

REMARQUE IV.6.2

Si un terme $\mathbf{t}$ est typable dans un contexte Γ, alors la longueur de Γ est exactement la longueur de la chaîne de $\mathbf{t}$. En particulier, un terme avec une chaîne directrice vide (par exemple la compilation d'un λ-terme clos) est typable si et seulement s'il est typable dans le contexte vide.

Exemple IV.6.3

Nous donnons deux petits exemples de dérivations de type dans ce système.

1. $(\lambda\Box^{\downarrow})^{\epsilon} : A \to A$

$$\dfrac{\dfrac{}{A \vdash \Box^{\downarrow} : A}}{\varnothing \vdash (\lambda\Box^{\downarrow})^{\epsilon} : A \to A}$$

2. $(\lambda(\lambda\Box^{\downarrow-})^{\downarrow})^{\epsilon} : A \to B \to A$

$$\dfrac{\dfrac{\dfrac{}{A, B \vdash \Box^{\downarrow-} : A}}{A \vdash (\lambda\Box^{\downarrow-})^{\downarrow} : B \to A}}{\varnothing \vdash (\lambda(\lambda\Box^{\downarrow-})^{\downarrow})^{\epsilon} : A \to B \to A}$$

Une première propriété utile est la suivante :

Proposition IV.6.4 (*Bonne formation*)

Les prétermes typables sont des termes bien formés.

Preuve. Les fonctions $\dot{\sim}$ et $\dot{-}\!\!\left\{\right.$ imposent clairement les contraintes sur les termes bien formés (Définition IV.2.1). □

Ce système de types correspond au λ-calcul simplement typé (voir la Section I.3.5), dont les jugements sont notés $\Gamma \vdash_{\lambda} M : A$, dans le sens suivant :

Lemme IV.6.5

1. $x_1 : A_1, \ldots, x_n : A_n \vdash_{\lambda} M : A \Longrightarrow A_1, \ldots, A_n \vdash [\![M]\!]_{[x_1,\ldots,x_n]} : A$
2. $A_1, \ldots, A_n \vdash \mathbf{t} : A \Longrightarrow x_1 : A_1, \ldots, x_n : A_n \vdash_{\lambda} (\!|\mathbf{t}|\!)_{[x_1,\ldots,x_n]} : A$

Preuve. Par une induction immédiate sur la dérivation de type. □

Nous avons aussi les résultats fondamentaux attendus d'un calcul simplement typé :

Théorème IV.6.6 (*Réduction du sujet*)

Si $\Gamma \vdash \mathbf{t} : A$ et $\mathbf{t} \rightsquigarrow \mathbf{u}$, alors $\Gamma \vdash \mathbf{u} : A$ (pour $\rightsquigarrow \in \{\rightsquigarrow_o, \rightsquigarrow_l, \rightsquigarrow_c\}$).

Preuve. Par la Proposition IV.4.11, les cas de λ_l et λ_c suivent du cas général pour λ_o, qui est prouvé en vérifiant que chaque règle préserve le type. Nous illustrons seulement la preuve sur le cas de App_1, les autres étant similaires.

La situation est la suivante, où $\rho_i = \curvearrowleft$, $j = |\rho_{1..i}|_l$, $\upsilon = \phi_l(\sigma, \rho_{\setminus i})$, $\tau = \psi_1(\sigma, \rho_{\setminus i})$ et $\Gamma = A_1, \ldots, A_n$:

$$\dfrac{\dfrac{\Delta_1 \vdash \mathbf{t} : A \to B \quad \Delta_2 \vdash \mathbf{u} : A \quad \Delta \overset{\rho}{\left\{ \begin{smallmatrix} \Delta_1 \\ \Delta_2 \end{smallmatrix} \right.}}{\Delta = \Gamma_1\langle i/C\rangle \vdash (\mathbf{t}\ \mathbf{u})^{\rho} : B} \qquad \Gamma_2 \vdash \mathbf{v} : C \quad \Gamma \overset{\sigma}{\left\{ \begin{smallmatrix} \Gamma_1 \\ \Gamma_2 \end{smallmatrix} \right.}}{\Gamma \vdash ((\mathbf{t}\ \mathbf{u})^{\rho}[i/\mathbf{v}])^{\sigma} : B}$$

$$\downarrow$$

$$\dfrac{\dfrac{\Omega_1\langle j/C\rangle \vdash \mathbf{t} : A \to B \quad \Omega_2 \vdash \mathbf{v} : C \quad \Theta_1 \overset{\upsilon}{\left\{ \begin{smallmatrix} \Omega_1 \\ \Omega_2 \end{smallmatrix} \right.}}{\Theta_1 \vdash (\mathbf{t}[j/\mathbf{v}])^{\upsilon} : A \to B} \qquad \Theta_2 \vdash \mathbf{u} : A \quad \Gamma \overset{\tau}{\left\{ \begin{smallmatrix} \Theta_1 \\ \Theta_2 \end{smallmatrix} \right.}}{\Gamma \vdash ((\mathbf{t}[j/\mathbf{v}])^{\upsilon}\ \mathbf{u})^{\tau} : B}$$

Alors la réduction du sujet est vérifiée pour cette règle, pourvu que

$$\begin{cases} \Omega_1\langle j/C\rangle = \Delta_1 \\ \Omega_2 = \Gamma_2 \\ \Theta_2 = \Delta_2 \end{cases}$$

Écrivons $L_\sigma = \{i \mid \sigma_i = \curvearrowleft \text{ ou } \not\curvearrowleft\}$, $R_\sigma = \{i \mid \sigma_i = \curvearrowright \text{ ou } \not\curvearrowright\}$ et $[A]_I = A_{i_1}, \ldots, A_{i_m}$ si $I = \{i_1, \ldots, i_m\}$ et $(i_j)_j$ est croissante, de sorte que $\Gamma = [A]_{\{1\ldots n\}}$. Alors nous avons :

$$\begin{array}{ll} \Gamma_1 = [A]_{L_\sigma} & \\ \Gamma_2 = [A]_{R_\sigma} & \\ \Delta_1 = [A]_{L_\sigma \cap L_\rho}\langle j/C\rangle & \text{puisque } \rho_i = \curvearrowleft \text{ et } j = |\rho_{1..i}|_l \\ \Delta_2 = [A]_{L_\sigma \cap R_\rho} & \end{array}$$

et de l'autre côté :

$$\begin{array}{ll} \Theta_1 = [A]_{L_\tau} & \\ \Theta_2 = [A]_{R_\tau} = [A]_{L_\sigma \cap R_\rho} & \text{par définition de } \psi_1 \\ \Omega_1 = [A]_{L_\tau \cap L_\upsilon} = [A]_{L_\sigma \cap L_\rho} & \text{par définition de } \phi_l \\ \Omega_2 = [A]_{L_\tau \cap R_\upsilon} = [A]_{R_\sigma} & \text{de nouveau par définition de } \phi_l. \end{array}$$

Ainsi la réduction du sujet est vraie pour App_1. Les autre cas sont similaires. □

THÉORÈME IV.6.7 (*Terminaison*)

Si $\Gamma \vdash \mathbf{t} : A$, alors $\mathbf{t}$ est fortement normalisable (dans λ_o, λ_l, λ_c).

Preuve. Puisque les λ-termes typables sont fortement normalisables, c'est une conséquence directe du Lemme IV.6.5 et de la PSN pour le calcul correspondant. □

IV.7 Machine abstraite pour l'évaluation

Dans cette section, nous allons exhiber une stratégie particulière qui se sert de l'information explicite donnée par les chaînes directrices pour réduire efficacement les termes clos en forme normale de tête faible. L'efficacité est mesurée par le nombre total d'étapes de récriture (et

pas simplement de β-réduction) et nous donnerons des comparaisons expérimentales dans la Section IV.9.

Notons que la syntaxe des chaînes directrices nous permet d'identifier le moment où nous devons copier un terme, et nous pouvons le réduire avant de le copier. En particulier, nous pouvons vouloir utiliser les règles les plus générales, afin de réduire un terme à copier en forme normale complète, évitant de ce fait de copier des rédex, quelle que soit leur forme. Cependant si nous procédons de la sorte, les substitutions ouvertes sont alors autorisées dans la règle App_3, ce qui signifie que des termes avec des variables libres, c'est-à-dire des rédex potentiels, peuvent être copiés. Nos essais expérimentaux ont confirmé qu'en restreignant juste cette règle au cas clos, nous obtenons une stratégie très semblable à la réduction close. Cela vient du fait que la propagation d'une substitution ouverte a de bonne chances d'être bloquée à un moment par cette restriction. Ainsi, la meilleure stratégie que nous avons trouvée est en fait basée sur le calcul clos, ce qui est une bonne nouvelle puisqu'il s'agit également du calcul le plus simple.

Nous ne pouvons pas espérer réduire en forme normale complète avec les règles closes, mais certains termes ouverts peuvent néanmoins être réduits. Ainsi, notre stratégie pour calculer la forme normale de tête faible d'un terme $\mathbf{t}$ peut être décrite comme suit : nous utilisons le calcul clos, qui autorise certaines réductions sous les abstractions, mais nous arrêtons la réduction dès que nous atteignons une forme normale de tête faible. Les réductions supplémentaires sont seulement effectuées quand nous réduisons un sous-terme à copier, pour partager davantage de travail que dans les stratégies habituelles.

Pour spécifier formellement cette stratégie, nous combinons une stratégie qui réduit sous les λ et une qui ne le fait pas. Nous définissons ainsi trois relations mutuellement récursives : $\Downarrow_w$, $\Downarrow_s$ et $\Downarrow$. Cette dernière sera la stratégie que nous voulons exhiber. La relation $\Downarrow_w$, définie dans la Figure IV.7, associée aux deux autres relations ci-dessous, définit la sémantique opérationnelle à grands pas que la machine abstraite close implante.

La relation de réduction $\Downarrow_w$ est utilisée comme outil pour définir les deux autres et n'a pas de sens considérée seule, dans la mesure où elle ne traite pas le cas des abstractions. Remarquons que la règle *(App_3)* appelle la réduction plus forte $\Downarrow_s$, qui est définie par :

$$\frac{\mathbf{t} \Downarrow_w \mathbf{v}}{\mathbf{t} \Downarrow_s \mathbf{v}} \qquad \frac{\mathbf{t} \Downarrow_s \mathbf{v}}{(\lambda \mathbf{t})^\sigma \Downarrow_s (\lambda \mathbf{v})^\sigma} \qquad \frac{\mathbf{t} \Downarrow_s \mathbf{v}}{(\lambda^- \mathbf{t})^\sigma \Downarrow_s (\lambda^- \mathbf{v})^\sigma}$$

$\Downarrow_s$ est la relation qui réduit sous les λ (mais pas en forme normale complète).

Finalement, $\Downarrow$ est la combinaison des deux autre relations : nous réduisons en forme normale de tête faible, mais nous réduisons davantage les sous-termes qui seront copiés.

$$\frac{\mathbf{t} \Downarrow_w \mathbf{v}}{\mathbf{t} \Downarrow \mathbf{v}} \qquad \frac{}{(\lambda \mathbf{t})^\sigma \Downarrow (\lambda \mathbf{t})^\sigma} \qquad \frac{}{(\lambda^- \mathbf{t})^\sigma \Downarrow (\lambda^- \mathbf{t})^\sigma}$$

Il peut sembler que la machine renvoie des termes qui ne sont pas des formes normales de tête faible (par exemple dans la règle *(Arg)*). En fait, la théorie assure que ce n'est pas le cas : à partir d'un terme clos, les règles closes nous permettent toujours d'atteindre une forme normale de tête faible (voir le Théorème IV.4.19). Néanmoins, la règle *(Arg)* peut être utilisée dans une réduction d'un terme à copier, elle est donc indispensable.

Les règles *(Subst)* et *(Comp)* appellent un commentaire : la restriction de *(Subst)* (v^ρ ouvert) force la règle *(Comp)* à être utilisée autant que possible avant la réduction à gauche

$$\frac{\mathbf{t} \Downarrow (\lambda \mathbf{r})^{\epsilon} \quad (\mathbf{r}[\mathbf{u}])^{\sigma} \Downarrow \mathbf{v}}{(\mathbf{t}\ \mathbf{u})^{\sigma} \Downarrow_w \mathbf{v}}\ (Beta) \qquad \frac{\mathbf{t} \Downarrow (\lambda^{-} \mathbf{v})^{\epsilon}}{(\mathbf{t}\ \mathbf{u})^{\epsilon} \Downarrow_w \mathbf{v}}\ (BetaE)$$

$$\frac{}{\Box \Downarrow_w \Box}\ (Ax) \qquad \frac{\mathbf{t} \Downarrow \mathbf{v} \quad \mathbf{v} \neq (\lambda \mathbf{r})^{\epsilon} \wedge (\mathbf{v} \neq (\lambda^{-} \mathbf{r})^{\epsilon} \vee \rho \neq \epsilon)}{(\mathbf{t}\ u^{\rho})^{\sigma} \Downarrow_w (\mathbf{v}\ u^{\rho})^{\sigma}}\ (Arg)$$

$$\frac{((\mathbf{t}[v^{\epsilon}])^{\curvearrowleft^{|\rho|_l}}\ \mathbf{u})^{\rho} \Downarrow \mathbf{w}}{((\mathbf{t}\ \mathbf{u})^{\curvearrowleft \rho}[v^{\epsilon}])^{\sigma} \Downarrow_w \mathbf{w}}\ (App1) \qquad \frac{(\mathbf{t}\ (\mathbf{u}[v^{\epsilon}])^{\curvearrowleft^{|\rho|_r}})^{\rho} \Downarrow \mathbf{w}}{((\mathbf{t}\ \mathbf{u})^{\curvearrowright \rho}[v^{\epsilon}])^{\sigma} \Downarrow_w \mathbf{w}}\ (App2)$$

$$\frac{v^{\epsilon} \Downarrow_s \mathbf{v}' \quad ((\mathbf{t}[\mathbf{v}'])^{\curvearrowleft^{|\rho|_l}}\ (\mathbf{u}[\mathbf{v}'])^{\curvearrowleft^{|\rho|_r}})^{\rho} \Downarrow \mathbf{w}}{((\mathbf{t}\ \mathbf{u})^{\curlywedge \rho}[v^{\epsilon}])^{\sigma} \Downarrow_w \mathbf{w}}\ (App3)$$

$$\frac{(\lambda(\mathbf{t}[u^{\epsilon}])^{\curvearrowleft^{|\rho|+1}})^{\rho} \Downarrow \mathbf{v}}{((\lambda \mathbf{t})^{\downarrow \rho}[u^{\epsilon}])^{\sigma} \Downarrow_w \mathbf{v}}\ (Lam) \qquad \frac{(\lambda^{-}(\mathbf{t}[u^{\epsilon}])^{\curvearrowleft^{|\rho|}})^{\rho} \Downarrow \mathbf{v}}{((\lambda^{-} \mathbf{t})^{\downarrow \rho}[u^{\epsilon}])^{\sigma} \Downarrow_w \mathbf{v}}\ (LamE)$$

$$\frac{\mathbf{v} \Downarrow \mathbf{w}}{(\Box[\mathbf{v}])^{\sigma} \Downarrow_w \mathbf{w}}\ (Var) \qquad \frac{(\mathbf{t}[(\mathbf{u}[v^{\epsilon}])^{\curvearrowleft^{|\rho|}}])^{\rho} \Downarrow \mathbf{w}}{((\mathbf{t}[\mathbf{u}])^{\curvearrowright \rho}[v^{\epsilon}])^{\sigma} \Downarrow_w \mathbf{w}}\ (Comp)$$

$$\frac{\mathbf{t} \Downarrow \mathbf{u} \quad (\mathbf{u}[v^{\rho}])^{\sigma} \Downarrow \mathbf{w} \quad \rho \neq \epsilon}{(\mathbf{t}[v^{\rho}])^{\sigma} \Downarrow_w \mathbf{w}}\ (Subst)$$

FIG. IV.7 – La relation $\Downarrow_w$

d'une substitution close. L'intuition et l'expérimentation confirment que c'est en effet le bon choix.

Notons que chaque règle correspond à une règle close (voir la Définition IV.4.8), ou bien focalise la réduction sur un sous-terme (ce qui correspond aux manipulations de pile) et peut effectivement être implantée en temps constant.

IV.8 Réduction en forme normale complète

Nous avons présenté jusqu'ici un système plutôt complexe pour simuler entièrement la β-réduction et des systèmes plus simples pour atteindre seulement la forme normale de tête faible. Si nous nous intéressons cependant au calcul des formes normales complètes, ce qui est le cas dans un certain nombre d'applications (par exemple, l'évaluation partielle ou les assistants de preuve), alors nous pouvons naturellement utiliser le cadre général. Mais ce n'est pas vraiment satisfaisant (il ne fournit aucun guidage vers une stratégie efficace et les règles sont coûteuses). D'autre part, nous avons une stratégie efficace pour réduire des termes clos en forme normale de tête faible. L'idée apparaît alors naturellement d'utiliser notre évaluateur faible efficace pour atteindre la forme normale complète, d'une façon similaire à [Cré90].

L'idée est de réduire un terme clos en forme normale de tête faible, puis de distinguer la variable liée par l'abstraction extérieure d'une manière quelconque (de la "geler"), de sorte que nous puissions de nouveau considérer le terme sous le λ comme clos, et d'appliquer récursivement le même procédé à ce sous-terme. Il y a plusieurs manières de distinguer ces variables dans la syntaxe. Ci-dessous nous présentons deux solutions naturelles.

IV.8.1 Avec des noms

Si nous choisissons de représenter les variables gelées avec des noms, nous pouvons éviter toute manipulation complexe de chaînes directrices pour maintenir les chemins vers ces variables. En conséquence, nous obtenons un système plutôt simple parce que nous pouvons utiliser les règles habituelles (par exemple les règles closes), où les variables gelées sont juste considérées comme des constantes et n'ont pas besoin de la moindre règle supplémentaire. De plus, la décompilation vers le λ-calcul nommé est alors réalisée en même temps que la réduction.

Formellement, nous étendons la syntaxe des termes de la façon suivante, où x représente des variables nommées :

$$\mathbf{t}, \mathbf{u} ::= \Box \mid (\lambda \mathbf{t})^{\sigma} \mid (\lambda^{-} \mathbf{t})^{\sigma} \mid (\mathbf{t}\ \mathbf{u})^{\sigma} \mid (\mathbf{t}[\mathbf{u}])^{\sigma} \mid x \mid \lambda' x.\mathbf{t}$$

Ainsi, nous ajoutons des variables nommées, dont la chaîne directrice implicite est ϵ, et un lieur nommé noté $\lambda' x.\mathbf{t}$. Nous n'écrivons aucune chaîne directrice pour cette abstraction, puisque nous considérerons toujours des termes clos de cette forme.

En utilisant une relation d'évaluation faible $\Downarrow$, nous pouvons alors définir la réduction en forme normale complète $\Downarrow_f$.

$$\frac{\mathbf{t} \Downarrow (\lambda \mathbf{t}')^{\epsilon} \qquad (\mathbf{t}'[x])^{\epsilon} \Downarrow_f \mathbf{t}'' \qquad x \text{ frais}}{\mathbf{t} \Downarrow_f \lambda' x.\mathbf{t}''}$$

$$\frac{\mathbf{t} \Downarrow (\lambda^{-} \mathbf{t}')^{\epsilon} \qquad \mathbf{t}' \Downarrow_f \mathbf{t}'' \qquad x \text{ frais}}{\mathbf{t} \Downarrow_f \lambda' x.\mathbf{t}''}$$

$$\frac{\mathbf{t} \Downarrow x \qquad (x \text{ variable})}{\mathbf{t} \Downarrow_f x}$$

$$\frac{\mathbf{t} \Downarrow (\mathbf{u}\ \mathbf{v})^{\epsilon} \qquad \mathbf{u} \Downarrow_f \mathbf{u}' \qquad \mathbf{v} \Downarrow_f \mathbf{v}'}{\mathbf{t} \Downarrow_f (\mathbf{u}'\ \mathbf{v}')^{\epsilon}}$$

Notons que la dernière règle est utilisée puisque nous sommes désormais dans un calcul avec constantes (les variables nommées), et la forme normale de tête faible d'un terme peut être une application (par exemple $(x\ \mathbf{t})^{\epsilon}$ où x est une variable nommée).

PROPOSITION IV.8.1 (*Correction*)

Notons $\tilde{\mathbf{u}}$ la décompilation d'un terme réduit $\mathbf{u}$, de sorte que $\tilde{\cdot}$ se contente de transformer les λ' en λ et d'effacer les chaînes directrices restantes. Alors, si M est un λ-terme clos :

$$[\![M]\!] \Downarrow_f \mathbf{u} \iff M \to_{\beta}^{*} \tilde{\mathbf{u}} \text{ en forme normale.}$$

Preuve. En supposant la correction de $\Downarrow$, la preuve est facilement adaptée de [Cré90] ou du plus récent [GL02]. □

Si nous voulons réduire un terme ouvert $\mathbf{t} = t^{\sigma}$ avec $|\sigma| = n$, nous prenons d'abord n noms de variable frais $x_1, \ldots, x_n$ et commençons la réduction à partir de

$$((\ldots((\mathbf{t}[x_1])^{\curvearrowleft^{n-1}}[x_2])^{\curvearrowleft^{n-2}} \ldots [x_{n-1}])^{\curvearrowleft}[x_n])^{\epsilon}.$$

La réduction en forme normale suit exactement la même stratégie que la réduction faible correspondante. Ainsi, pour des termes pour lesquels formes normales de tête faible et complète sont les mêmes, les deux réductions effectuent le même nombre de β et d'étapes totales. En particulier, cette stratégie est beaucoup plus efficace que la stratégie naïve habituelle.

Bien que nous devions maintenant nous occuper de noms et de variables fraîches pendant la réduction (ce qui n'était pas le cas pour la réduction en forme normale de tête faible), nous ne devons toujours pas nous soucier de la capture de noms et de l'α-conversion. En outre, la décompilation est maintenant simplifiée.

IV.8.2 Avec des directeurs

Le procédé décrit dans la section précédente exécute la décompilation pendant le calcul de la forme normale, ce qui peut être désiré, ou pas. Nous pouvons cependant aussi implanter une idée similaire en utilisant seulement des directeurs, de façon à obtenir un résultat dans cette syntaxe. Nous avons juste besoin d'une façon de distinguer les variables habituelles des

variables gelées, qui correspondent à une abstraction à l'extérieur du sous-terme que nous voulons réduire en forme normale de tête faible. Ceci peut être fait d'une façon tout à fait évidente : en introduisant un nouveau genre de directeurs correspondant à ces variables gelées. Cependant, la partie gelée des chaînes directrices peut être de n'importe quelle forme, ainsi nous devons utiliser les règles générales sur cette partie. D'un point de vue algorithmique, ceci signifie que le coût d'une étape de réduction est au plus linéaire en la profondeur des λ-abstractions du résultat en forme normale, ce qui semble encore raisonnable. Nous n'allons pas plus loin sur ce point car les idées intéressantes ont déjà été exposées dans la section précédente.

IV.9 Résultats expérimentaux

Une des motivations principales de ce travail est le désir de trouver des implantations plus efficaces du λ-calcul. Il ne s'agit pas simplement de chercher des améliorations ou des optimisations mineures des systèmes existants, mais plutôt d'essayer de repartir du début pour découvrir de nouvelles stratégies et de nouvelles techniques qui donnent des améliorations asymptotiques sur les techniques standard utilisées dans les implantations des langages fonctionnels. En conséquence il est essentiel que nos stratégies soient implantées de sorte que les idées de ce chapitre puissent être soutenues par des résultats expérimentaux et être comparées aux évaluateurs existants.

Il est difficile, sinon impossible, de trouver une mesure appropriée pour comparer des machines implantées dans des cadres différents (c'est en particulier l'une des motivations du Chapitre V). Ce point est encore plus vrai quand nous comparons des implantations de *prototypes*. Néanmoins, ce que nous pouvons faire, c'est identifier des étapes de réduction atomiques pour plusieurs évaluateurs. Bien que le coût algorithmique d'un pas puisse changer d'un évaluateur à l'autre, les accélérations asymptotiques respectives peuvent toujours être examinées. Le banc d'essai suivant suggère que les stratégies développées dans cette thèse se comportent mieux que les stratégies de réduction standard, et de plus offrent certaines statistiques surprenantes en comparaisons avec certains des meilleurs évaluateurs connus à ce jour.

Nous avons deux ensembles d'exemples : un qui appartient à $\lambda\mathsf{I}$ afin d'éviter le problème de l'effacement, et un petit ensemble de termes de $\lambda\mathsf{K}$ pour illustrer quelques comportements spécifiques. Les entiers de Church sont d'excellents moyens de produire un échantillon de grands λ-termes. Nous rappelons que les entiers de Church sont de la forme $n = \lambda f.\lambda x.f^n\, x$ et que l'application correspond à l'élévation à une puissance : $n\, m \equiv m^n$. Nous appliquons les entiers de Church dans nos exemples à $\mathsf{I}\,\mathsf{I}$, où $\mathsf{I} = \lambda x.x$, ce qui suffit à forcer la réduction en forme normale complète et nous permet de comparer les évaluateurs faibles et forts. Nous utilisons également les combinateurs $\mathsf{K} = \lambda x.\lambda y.x$ et $\mathsf{M} = \lambda x.\lambda y.(\mathsf{K}\,\mathsf{I}\,x)(\mathsf{K}\,\mathsf{I}\,x\,y)$.

Nous comparons notre machine abstraite (notée CR dans la table) à des implantations naïves de l'appel par valeur (CBV) et de l'appel par nom (CBN), ainsi qu'à l'interprète optimal (BOHM) d'Asperti *et al.* [AGN96]. Le dernier résultat fournit une comparaison avec le meilleur évaluateur connu pour de tels termes. Nous donnons également une comparaison avec CBVNF, un appel par valeur où l'argument est réduit en forme normale complète avant

la β-réduction. Nous montrons le nombre total d'étapes de ces évaluateurs (y compris les manipulations de pile). Nous donnons également le nombre de β-réductions entre parenthèses, ainsi le nombre indiqué pour BOHM est le nombre minimum de β-réductions possible. Les résultats pour les machines qui réduisent en forme normale complète ne sont pas montrés, car ils sont identiques à ceux des stratégies faibles sous-jacentes sur ces exemples.

Terme	CR	CBV	CBN	CBVNF	BOHM
2 2 I I	61(9)	78(11)	76(12)	98(11)	40(9)
2 2 2 I I	140(19)	362(42)	471(60)	342(36)	93(16)
5 5 I I	217(33)	29 723(3 913)	31 250(4 689)	22089(3153)	208(33)
5 2 2 I I	832(109)	-	-	-	847(31)
2 2 2 2 2 I I	1 507 714(196 655)	-	-	-	1 074 037 060(61)
M (5 5 I I) I	266(42)	41(8)	31(8)	34621(3161)	22(8)
K I (5 5 I I)	7(2)	29731(3915)	7(2)	34585(3155)	4(2)

Pour mettre certains de ces résultats en perspective, nous remarquons que le temps réel pour calculer, par exemple, 5 2 2 I I en utilisant OCaml est de l'ordre de 5 minutes, et de l'ordre de 3 minutes en utilisant Standard ML (les deux implantent une variante d'appel par valeur). Les résultats à la fois pour la réduction close et pour BOHM sont essentiellement instantanés.

La remarque principale que nous voulons faire avec la table ci-dessus est que la réduction close, une implantation simple du λ-calcul, a des performances clairement supérieures aux stratégies traditionnelles, telles que l'appel par valeur, et de plus est un concurrent sérieux à des implantations très sophistiquées, telles que BOHM. La comparaison avec l'appel par valeur et l'appel par nom montre qu'autoriser des réductions sous les abstractions, ce qui est particulièrement facile avec les chaînes directrices en comparaison avec les calculs habituels, est crucial pour le partage et l'efficacité.

Le point intéressant est le comportement asymptotique des stratégies quand la taille des termes augmente. Les résultats montrent que notre machine peut réduire des termes plus grands que les autres machines, et plus le terme est grand, meilleure est notre machine comparée aux autres. En fait, notre machine est tout à fait comparable au meilleur évaluateur connu pour de tels termes, à savoir BOHM [AGN96], alors même qu'elle repose sur un modèle beaucoup plus simple. Ceci autorise à penser qu'elle permet un degré de partage élevé (parce que plus le terme est grand, plus il y a de partage possible). Le dernier exemple du premier ensemble montre que notre machine explose en nombre de β-réductions par rapport à l'optimal, mais surpasse BOHM en nombre total d'étapes, ce qui est notre notion d'efficacité.

En complément à ce banc d'essai classique, nous donnons deux termes avec effacement pour illustrer les points suivants :

- La stratégie close peut effectuer plus de travail que nécessaire, quand chaque copie d'un terme sera effacée. L'occurrence de tels termes dans des situations pratiques est cependant incertaine. On pourrait également envisager de combiner la stratégie close avec l'appel par nécessité d'une façon semblable à [EPJ03] pour éviter ce problème. Une approche similaire à celle développée au Chapitre V semble également prometteuse.
- Mise à part la situation précédente, nous pouvons éviter d'effectuer du travail inutile, par opposition à l'appel par valeur par exemple, c'est-à-dire que nous avons également

certains des avantages de l'appel par nom.

IV.10 Conclusion

Nous avons présenté une syntaxe sans noms pour représenter les termes du λ-calcul avec substitutions explicites, d'une manière qui suit les intuitions habituelles quant à la sémantique opérationnelle de la propagation des substitutions. Nous avons donné un calcul général sur les chaînes directrices qui peut simuler entièrement le λ-calcul, avec des règles plutôt compliquées. Nous avons alors décrit un calcul intermédiaire, le calcul ouvert local, avec des règles très simples et permettant toujours aux substitutions ouvertes de traverser les abstractions sans récriture globale. Enfin, nous avons dérivé le calcul clos, qui internalise les conditions du système original décrit au Chapitre II.

Ces calculs ont été utilisés comme base pour décrire et implanter des machines abstraites pour la réduction faible et forte. L'efficacité était notre motivation principale, et nous avons trouvé dans la pratique que ces machines sont tout à fait efficaces sur de grands termes et autorisent un degré de partage élevé. En particulier, elles se comparent tout à fait favorablement aux évaluateurs standard, ce qui suggère que des implantations plus efficaces des langages fonctionnels et des assistants de preuve basés sur le λ-calcul sont encore possibles.

Chapitre V

Stratégies du λ-calcul et réseaux d'interaction

Les réseaux d'interaction sont un formalisme *a priori* idéal pour décrire à un niveau d'abstraction relativement élevé les implantations de langages de programmation, en particulier fonctionnels. De plus, certaines stratégies s'expriment bien dans les réseaux d'interaction alors qu'on ne sait prouvablement pas les exprimer dans un formalisme de termes, comme c'est le cas pour la stratégie optimale [Fie90, Lam90]. Mais de façon surprenante, on constate que, réciproquement, on ne sait pas non plus exprimer les stratégies usuelles dans les réseaux d'interaction. Le problème est qu'il s'agit d'un formalisme de calcul distribué. Pour synchroniser certains calculs dans le cas de la réduction optimale, et de certaines autres stratégies, il est nécessaire d'implanter un mécanisme de *boîte*, comme dans la Logique Linéaire de Girard [Gir87]. Dans ce chapitre, nous montrons que les stratégies usuelles d'appel par nom et d'appel par valeur peuvent être implantées dans les réseaux d'interaction de façon très simple, en rendant explicite le flot de contrôle au niveau syntaxique, et sans aucun besoin de boîte. Nous montrons aussi comment implanter l'appel par nécessité dans un cadre un peu plus général, mais sans perdre de propriété. Nous étendons finalement l'approche à la stratégie pleinement paresseuse, qui offre un meilleur partage que les implantations courantes des langages fonctionnels. Nous proposons donc un cadre plus uniforme entre les réseaux d'interaction et les machines abstraites traditionnelles. Ce travail peut aussi être vu comme une étape vers une approche générique pour développer des machines abstraites basées sur la récriture de graphes.

V.1 Introduction

Les réseaux d'interaction [Laf90] constituent un modèle de calcul graphique et distribué, qui permet d'exprimer de façon explicite et uniforme toutes les étapes d'un calcul, en particulier certaines étapes de copie et d'effacement habituellement cachées. Ce formalisme est inspiré par les réseaux de preuve et la logique linéaire [Gir87]. En particulier, le partage est natif (contrairement aux termes) et est traité explicitement (contrairement aux *termgraphs*). La réduction dans les réseaux d'interaction est locale et fortement confluente, ainsi les réductions peuvent avoir lieu dans n'importe quel ordre, même en parallèle (voir [Pin00]). Ces propriétés font des réseaux d'interaction un formalisme intermédiaire adapté pour les implantations de langages de programmation.

En effet, les réseaux d'interaction ont été utilisés avec succès pour l'implantation de la réduction optimale dans le λ-calcul, en partant de Lamping [Lam90], Gonthier, Abadi et Lévy [GAL92], Asperti *et al.* [AGN96] jusqu'aux travaux récents de van Oostrom *et al.* [OLZ04]. Ils ont aussi été utilisés pour un certain nombre d'autres implantations efficaces (non-optimales) du λ-calcul, par exemple par Mackie [Mac98, Mac04].

Pourtant, les réseaux d'interaction ont un statut relativement théorique : aucun des systèmes cités ci-dessus n'est utilisé dans une implantation réelle d'un langage de programmation répandu. C'est très dommageable, car les stratégies possibles dans ce cadre peuvent avoir des résultats surprenants en termes d'efficacité. Il est donc naturel d'essayer de combler le fossé entre réseaux d'interaction et formalismes plus traditionnels, tels que les machines abstraites.

Les codages ci-dessus ont en commun qu'un β-rédex est toujours traduit en paire active. Paradoxalement, alors que toutes les suites de réductions sont équivalentes, il y a donc besoin d'un interprète externe pour trouver les rédex et les gérer, ce qui est typiquement implanté à l'aide d'une pile [Pin00] ou par des méthodes *ad hoc*, non documentées, qui ne font pas partie de la théorie, donc qui sont propices aux erreurs. De plus, des réductions correspondant à des β-réductions différentes peuvent être entrelacées, ce qui impose de devoir simuler une notion de *boîte* [Gir87, GAL92] d'une façon plus ou moins complexe et coûteuse (voir [Lam90, GAL92, Mac94, Mac98, Mac04] pour divers types de tels codages).

Au contraire, dans ce chapitre, nous proposons de restreindre un peu, mais pas complètement, la liberté de nos codages. Notre traduction d'un rédex est en général inerte et nous donnons un statut explicite à la stratégie, sous la forme d'un agent standard, appelé *jeton d'évaluation*, qui traverse le réseau comme une fonction d'évaluation le ferait, en déclenchant les réductions sur son passage. Suivant cette idée, nous allons présenter successivement des codages dans les réseaux d'interaction de l'appel par nom, l'appel par valeur, l'appel par nécessité et de la stratégie pleinement paresseuse.

Ce type d'approche rappelle les travaux de Lippi [Lip02b, Lip02a]. Celui-ci décrit en effet une implantation dans les réseaux d'interaction de la réduction gauche, et propose même une description de la machine de Krivine sous forme de réseaux d'interaction. Cependant, ces codages, en ne donnant pas un statut explicite au flot d'évaluation, sont moins intuitifs et manquent de généralité. En particulier, il semble difficile d'étendre cette présentation à d'autres stratégies.

Notre idée d'un jeton qui traverse un graphe est aussi superficiellement réminiscente de la géométrie de l'interaction [Gir89], qui a été utilisée pour implanter des machines abstraites en

appel par nom [Mac95] et en appel par valeur [FM02]. Cependant les détails des approches sont très différents. En particulier, ces machines évitent autant que possible de modifier le graphe, et laissent donc moins de liberté dans la stratégie. Par exemple, l'appel par valeur est obtenu au prix d'une grande complexité. Il est relativement clair cependant que ces machines peuvent être formalisées dans notre cadre (en rendant le jeton explicite et en codant la pile avec des agents d'interaction), mais cela ne semble pas mener à une meilleure compréhension de la géométrie de l'interaction (et en particulier à une machine en appel par valeur plus satisfaisante).

Notre approche rappelle aussi la transformation par passage de continuation [Plo75], au sens où nous simulons une stratégie d'évaluation en interdisant certaines réductions tant que quelque chose ne les déclenche pas (le jeton ou la continuation). Cependant, une telle transformation suivie d'un codage traditionnel dans les réseaux d'interaction ne permettrait certainement pas de nous débarrasser des boîtes, bien qu'une seule β-réduction ne serait possible à la fois. De ce point de vue, notre approche est donc plus satisfaisante et plus simple à étendre à d'autres stratégies.

Pour résumer, nous décrivons des machines abstraites basées sur des graphes pour les stratégies usuelles de la Section I.5.2, jusqu'à la stratégie pleinement paresseuse. Notre approche est simple et, bien que certaines réductions soient séquentialisées, nous ne perdons pas tout le potentiel de parallélisme : nous rendons simplement explicite quels calculs doivent être effectués séquentiellement et lesquels peuvent être effectués en parallèle. Nous fournissons aussi un cadre plus uniforme qui relie les stratégies usuelles et les réseaux d'interaction, dans lesquels la réduction optimale peut être implantée. L'application de cette technique à la réduction close sera discutée dans le chapitre suivant.

Ce chapitre est organisé de la façon suivante. Dans la Section V.2, nous décrivons en détails le cas de l'appel par nom pour les termes clos. Les Sections V.3 et V.4 adaptent la présentation respectivement à l'appel par valeur clos et aux termes ouverts. L'appel par nécessité est à son tour traité en détails dans la Section V.5, et la question de la réduction pleinement paresseuse est abordée dans la Section V.6. Nous concluons en Section V.7.

V.2 Appel par nom

Dans cette section, nous donnons un codage de la stratégie d'appel par nom du λ-calcul dans les réseaux d'interaction. Nous présentons la stratégie avec des règles inductives, dans un modèle à grands pas, et la première étape de notre codage consiste à dériver un système de récriture plus fin. Dans cette section, nous considérons seulement des termes clos (sans variables libres). Les termes ouverts seront traités dans la Section V.4.

V.2.1 Sémantique à grands pas

La stratégie d'appel par nom pour les λ-termes clos est spécifié par les règles d'évaluation suivantes (voir la Section I.5.2) :

$$\frac{}{\lambda x.t \Downarrow \lambda x.t}\ Lam \qquad \frac{t \Downarrow \lambda x.t' \quad t'\{x := u\} \Downarrow v}{t\ u \Downarrow v}\ App$$

C'est en fait la définition inductive d'une fonction d'évaluation (également connue sous le nom de sémantique à grands pas), plutôt qu'une stratégie : nous prenons un λ-terme t en entrée et nous trouvons inductivement un terme v tel que $t \Downarrow v$. Alors, v est la forme normale de tête faible unique de t (pourvu qu'elle existe) obtenue par la stratégie d'appel par nom. Le chemin de réduction n'est pas visible dans la proposition $t \Downarrow v$, mais bien dans l'arbre de dérivation constituant la preuve de cette proposition.

Ces règles sont trop implicites, cachent trop le calcul, pour un codage direct dans les réseaux d'interaction. Dans la règle pour l'application, la procédure est appelée récursivement sur le terme gauche, et nous devons alors revenir à cette application d'une façon ou d'une autre. Dans un cadre de programmation fonctionnelle, c'est automatique mais ce n'est pas une opération gratuite : quand on entre dans une fonction, l'environnement courant est sauvegardé sur la pile ; quand elle termine, cette information est descendue de la pile. Nous formaliserons donc la stratégie d'appel par nom dans un modèle à petits pas, afin d'être plus explicite quant au flot de contrôle et de faciliter le codage dans les réseaux d'interaction.

V.2.2 Sémantique à petits pas

Nous voulons remplacer les règles inductives précédentes par un système de récriture du premier ordre, mais nous voulons également être aussi explicites quant à l'ordre d'évaluation que dans le système précédent. Intuitivement, l'idée est d'imiter avec des règles de récriture ce qui se produit en construisant l'arbre de déduction d'un jugement $t \Downarrow v$. Trois types de choses peuvent se produire : nous pouvons commencer une nouvelle branche et monter dans l'arbre de déduction (évaluer un sous-terme), nous pouvons terminer une branche (renvoyer un résultat intermédiaire) et nous pouvons descendre dans l'arbre (remettre le résultat dans le bon contexte).

Nous enrichissons donc la syntaxe des termes avec deux symboles unaires : $\Downarrow$ (correspondant à l'évaluation) et $\Uparrow$ (correspondant au retour de la fonction d'évaluation). Plus précisément, les *termes enrichis* sont définis par la grammaire suivante :

$$e ::= C[\Downarrow t] \mid C[\Uparrow t]$$

où t représente un λ-terme et $C[\,]$ un contexte à un trou sur les λ-termes (non-enrichis). Les termes enrichis ont donc une unique occurrence du symbole $\Downarrow$ ou $\Uparrow$.

Nous définissons alors le système de récriture suivant sur les termes enrichis :

$$\begin{array}{lllll} (\Downarrow Lam) & \Downarrow \lambda x.t & \rightarrow & \Uparrow \lambda x.t \\ (\Downarrow App) & \Downarrow (t\ u) & \rightarrow & (\Downarrow t)\ u \\ (\Uparrow App) & (\Uparrow \lambda x.t)\ u & \rightarrow & \Downarrow (t\{x := u\}) \end{array}$$

On voit bien intuitivement comment le système à petits pas est dérivé de celui à grands pas. Dans le cas particulier de l'appel par nom, omettre le symbole $\Uparrow$ donne un système équivalent (c'est exactement l'optimisation de récursion terminale), mais nous préférons l'inclure dès à présent ; ce point sera à nouveau discuté dans la Section V.3. A notre connaissance, c'est la première fois qu'une sémantique à petits pas est présentée de façon aussi simple pour l'appel par nom. Les présentations habituelles à petits pas de l'appel par nom et de l'appel par valeur

se fondent sur des règles inductives permettant des réductions dans une certaine classe de contextes, par conséquent elles ne rendent pas explicite le flot d'évaluation, contrairement à notre présentation, ce qui est crucial pour le codage dans les réseaux d'interaction. En un sens, notre présentation est intermédiaire entre la sémantique traditionnelle à petits pas (qui sépare autant que possible la réduction de la stratégie) et les machines abstraites (qui peuvent impliquer des structures de données complexes). Nous appelons cette présentation la *sémantique à passage de jeton* de l'appel par nom.

Un λ-terme t est toujours en forme normale pour ce système, de même que $\Uparrow t$. Pour évaluer t, nous devons commencer la réduction à partir de $\Downarrow t$. Une réduction implique toujours un $\Downarrow$ ou un $\Uparrow$, donc, par la proposition suivante, il y a toujours au plus un rédex dans un terme obtenu à partir de la réduction de $\Downarrow t$. Le flot de contrôle est donc véritablement rendu explicite au niveau syntaxique.

PROPOSITION V.2.1

Si $\Downarrow t \to^* u$, alors il existe exactement une occurrence de $\Downarrow$ ou $\Uparrow$ dans u.

Preuve. Par induction. Les deux premières règles sont faciles. Dans la dernière règle, le membre de droite peut avoir zéro ou plus d'une occurrence de u, mais u n'a pas d'occurrence de $\Downarrow$ ou $\Uparrow$ par l'hypothèse d'induction. □

Les deux systèmes se correspondent de la façon suivante :

PROPOSITION V.2.2

$t \Downarrow v \iff \Downarrow t \to^* \Uparrow v$

Preuve.

$\Rightarrow$ Par induction sur la dérivation :
- *Lam* : $\lambda x.t \Downarrow \lambda x.t$ et en effet $\Downarrow \lambda x.t \xrightarrow[\Downarrow Lam]{} \Uparrow \lambda x.t$;
- *App* : si $t\ u \Downarrow v$, alors il existe t' tel que $t \Downarrow \lambda x.t'$ et $t'\{x := u\} \Downarrow v$. Par induction, $\Downarrow t \to^* \Uparrow \lambda x.t'$ et $\Downarrow t'\{x := u\} \to^* \Uparrow v$, d'où : $\Downarrow (t\ u) \xrightarrow[\Downarrow App]{} (\Downarrow t)\ u \to^* (\Uparrow \lambda x.t')\ u \xrightarrow[\Uparrow App]{} \Downarrow t'\{x := u\} \to^* \Uparrow v$.

$\Leftarrow$ La première partie de la proposition (déjà prouvée) permet de formuler le lemme suivant : si t est un λ-terme et t a une forme normale de tête faible, alors il existe v tel que $\Downarrow t \to^* \Uparrow v$ et v est en forme normale de tête faible (conséquence du Théorème I.5.3). Nous pouvons alors procéder par induction structurelle sur t :
- Abstraction : $\Downarrow \lambda x.t \xrightarrow[\Downarrow Lam]{} \Uparrow \lambda x.t$ et en effet $\lambda x.t \Downarrow \lambda x.t$ par la règle (Lam) ;
- Application : $\Downarrow (t\ u) \xrightarrow[\Downarrow App]{} (\Downarrow t)\ u$. Par le lemme, si t a une forme normale de tête faible, il existe $\lambda x.t'$ (rappelons que tous les termes sont clos), tel que $\Downarrow t \to^* \Uparrow \lambda x.t'$. De plus, $t \Downarrow \lambda x.t'$ par induction. Alors $(\Uparrow \lambda x.t')\ u \xrightarrow[\Uparrow App]{} \Downarrow t'\{x := u\}$ et un argument similaire (lemme et induction) permet de conclure en utilisant la règle (App). Si t ou $t'\{x := u\}$ n'a pas de forme normale de tête faible, la proposition est trivialement vrai (nous n'atteignons pas un terme de la forme $\Uparrow v$).

□

Par conséquent le système de récriture précédent correspond fidèlement à la stratégie d'appel par nom. Cette étape est cruciale, car le codage dans les réseaux d'interaction suivra fidèlement le système à petits pas.

V.2.3 Codage des termes

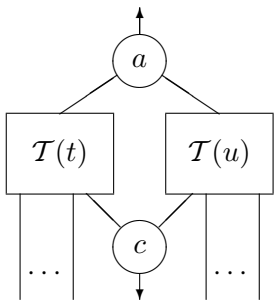
Dans cette section, nous définissons une fonction $\mathcal{T}(\cdot)$ de traduction des λ-termes dans les réseaux, qui est très naturelle. Nous utilisons la terminologie de la Section I.6. Nous représentons simplement les termes par leur arbre de syntaxe, où nous groupons ensemble plusieurs occurrences de la même variable par des agents c (correspondant à la copie) et les lions à leur nœud λ correspondant (ceci est parfois désigné sous le nom d'un *backpointer*). Les nœuds pour l'application et l'abstraction sont les agents λ et a respectivement. Ces agents ont trois ports, et leur port principal est orienté vers la racine du terme.

Dans les codages traditionnels, le port principal de l'agent d'application est orienté vers la gauche (c'est-à-dire vers la traduction du sous-terme de gauche de l'application), de sorte que l'interaction avec une abstraction (β-réduction) est toujours possible.

Ici, au contraire, les termes sont traduits en *réseaux principaux* [Laf97, Lip02b] avec un port principal libre, appelé la *racine* du réseau, et éventuellement certains ports auxiliaires libres, correspondant aux variables libres du terme. En particulier, les termes clos seront traduits en *paquets* [Laf97, Lip02b]. Les traductions de termes sont donc des *réseaux réduits*, et quelque chose devra déclencher la réduction : le *jeton d'évaluation*.

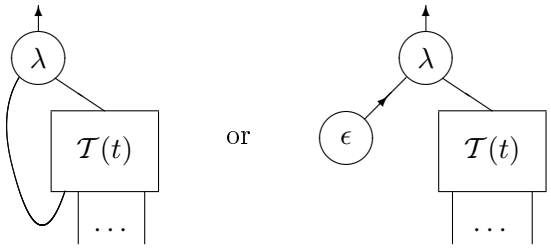
Variables. Dans cette section, nous considérons seulement des termes clos (les termes ouverts seront traités dans la Section V.4), donc les variables ne sont pas traduites en tant que telles. Elles sont simplement représentées par des arêtes entre leur lieur et leur occurrence groupée dans le corps de l'abstraction, comme expliqué ci-dessous.

Application. La traduction $\mathcal{T}(t\ u)$ d'une application $t\ u$ est simplement un agent a d'arité 2 dont le port principal pointe vers la racine, et dont le port auxiliaire gauche (appelé *port fonction*) est relié à la racine de $\mathcal{T}(t)$ et le port auxiliaire droit (*port argument*) est relié à la racine de $\mathcal{T}(u)$. Si t et u ont des variables libres en commun, alors des agents c (représentant la *copie*) les regroupent deux à deux de sorte qu'une seule occurrence de chaque variable libre apparaisse parmi les ports libres (un seul regroupement de ce type est représenté sur la figure).



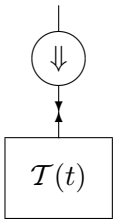
Abstraction. Pour une abstraction, $\mathcal{T}(\lambda x.t)$ est obtenue en introduisant un agent λ, et en reliant simplement son port auxiliaire droit (appelé *port corps*) à la racine de $\mathcal{T}(t)$ et son

port gauche (*port variable*) au port unique correspondant à x dans $\mathcal{T}(t)$. Si x n'apparaît pas dans t, alors le port gauche de l'agent λ est connecté à l'agent ϵ (*effacement*).



Pour résumer, nous représentons les λ-termes d'une façon très naturelle. En particulier, il n'y a aucun codage pour représenter les boîtes. Un autre point intéressant est que, en raison du lien explicite entre une variable et son agent λ correspondant, l'α-conversion est gratuite, comme c'est souvent le cas dans les représentations graphiques du λ-calcul. Jusqu'ici, nous avons seulement introduit les agents λ et a correspondant strictement au λ-calcul, ainsi que les agents ϵ et c pour la gestion de ressources explicite nécessaire (et souhaitable : nous ne voulons pas cacher des aspects aussi importants) dans les réseaux d'interaction. Remarquons en outre que la traduction d'un terme n'a aucune paire active. Par conséquent, elle est en forme normale, quelles que soient les règles d'interaction permises. De plus, elle a exactement un port principal libre, relié à la racine.

Termes enrichis. Pour commencer l'évaluation, nous devons introduire une notion de jeton d'évaluation dans nos réseaux d'interaction. Autrement dit, nous devons donner une traduction des termes enrichis. Nous introduisons donc deux nouveaux agents unaires $\Downarrow$ et $\Uparrow$. La traduction du terme enrichi $\Downarrow t$ est le réseau suivant, noté $\mathcal{T}(\Downarrow t)$ ou $\Downarrow\mathcal{T}(t)$.



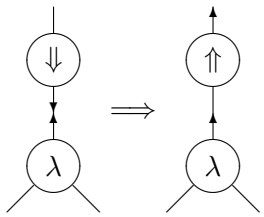
Le réseau $\mathcal{T}(\Uparrow t)$ (également noté $\Uparrow\mathcal{T}(t)$) s'obtient de manière similaire, en remplaçant l'agent $\Downarrow$ par un agent $\Uparrow$, dont le port principal est dirigé vers la racine. En particulier, $\Uparrow\mathcal{T}(t)$ est toujours un réseau en forme normale.

V.2.4 Evaluation par interaction

Dans cette section, nous donnons la dynamique du codage, c'est-à-dire, les règles d'interaction.

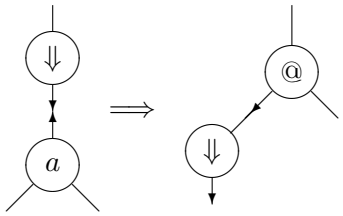
Règles linéaires

Les règles d'interaction suivent d'aussi près que possible le système de récriture de la Section V.2.2. La première est facile : quand le jeton d'évaluation atteint un λ, l'évaluation du réseau sous lui est terminée, et il peut commencer à remonter :

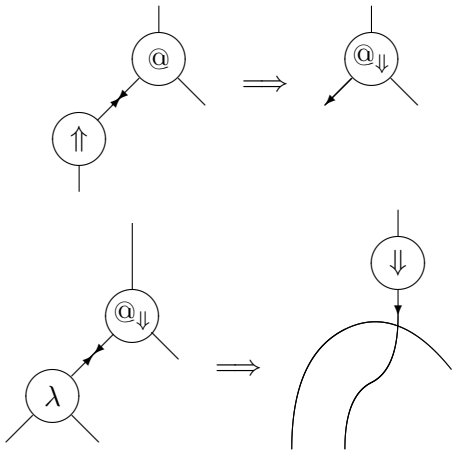


Nous profitons de cette première règle pour rappeler que les règles d'interaction ont toujours un nom implicite : par exemple, la règle ci-dessus est notée $\Downarrow \bowtie \lambda$, et la réduction par cette règle est notée $\underset{\Downarrow\bowtie\lambda}{\Longrightarrow}$.

Pour évaluer un terme dont le symbole de tête est une application, il faut d'abord évaluer son sous-terme gauche. Autrement dit, il faut déplacer le jeton d'évaluation vers le port fonction de l'application. On renomme aussi l'agent a en @, qui représente encore une application, mais dont le port principal ne pointe plus vers la racine mais vers la gauche, de sorte que l'interaction sera possible quand le jeton d'évaluation reviendra.



Finalement, quand l'agent $\Uparrow$ revient à un agent @ après l'évaluation réussie d'un sous-terme, alors nous savons avec certitude qu'il y a un λ juste sous l'agent $\Uparrow$, et une β-réduction doit être effectuée. A cause de la restriction aux interactions binaires, cela prend deux étapes :



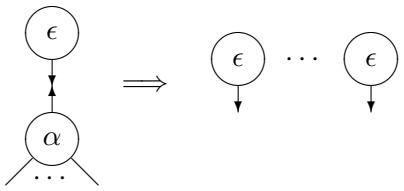
Il nous faut une étape pour dégager le chemin entre les agents @ et λ, et une autre pour relier le port variable de l'agent λ au port argument de l'agent @, ce qui initie la substitution, et pour remettre un agent $\Downarrow$ pour continuer l'évaluation du corps de l'abstraction. Nous avons introduit un nouveau nom, $@_{\Downarrow}$, pour représenter une application qui cache un jeton

d'évaluation, de façon à ce qu'il soit clair que l'unicité du jeton est conservée. En bref, nous suivons exactement le système de récriture, sauf que nous avons besoin de deux étapes au lieu d'une. Le cœur des règles d'interaction pour l'appel par nom a donc seulement quatre règles d'interaction, et aucun codage de boîtes.

On peut remarquer que, pour le moment, l'agent $\Uparrow$ n'est pas très utile, et pourrait être supprimé. C'est pourtant la clef de la généralité de notre traduction, comme nous le verrons pour les autres stratégies. On peut également noter que supprimer l'agent $\Uparrow$ correspond à l'optimisation de récursion terminale, et que notre cadre fournit un bon cadre intermédiaire pour raisonner sur ce type d'optimisations, entre la définition inductive d'une stratégie et son implantation sous forme de machine abstraite.

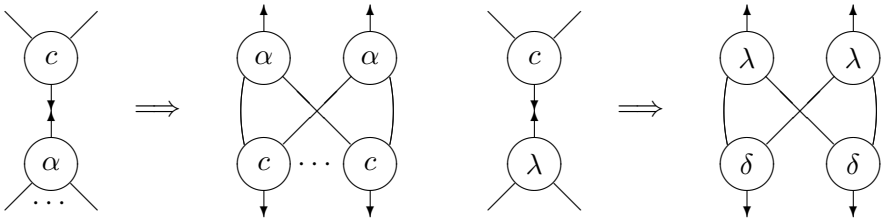
Copie et effacement

La gestion de ressources explicite typique des réseaux d'interaction est effectuée par les agents ϵ, c et δ. L'agent auxiliaire δ est introduit pour dupliquer les abstractions, comme expliqué ci-dessous. L'agent ϵ efface n'importe quel agent et se propage selon le schéma suivant (où α représente n'importe quel agent) :

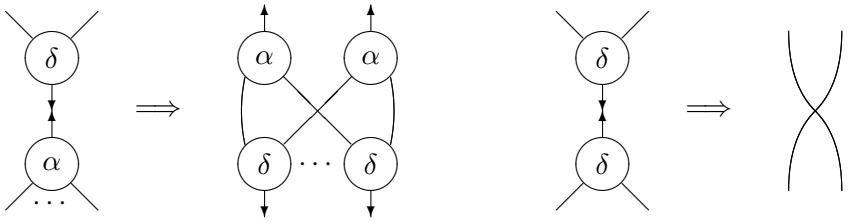


On remarque en particulier que si l'agent α est 0-aire (ce qui est le cas pour ϵ), le réseau de droite est vide. L'agent ϵ est donc responsable du glanage de cellules (*garbage collection*). Il y a deux remarques intéressantes à faire à ce sujet. D'abord, le glanage de cellules est complet (voir la Proposition V.2.3), parce que la β-réduction, donc l'effacement, ne se produit que sur des termes clos grâce au jeton d'évaluation. Ceci contraste avec d'autres implantations du λ-calcul dans les réseaux d'interaction, où il est souvent nécessaire d'évaluer un réseau (ce qui peut ne pas terminer) avant de pouvoir l'effacer. D'autre part, les interactions avec ϵ ne sont jamais nécessaires pour permettre au jeton de progresser. En particulier, les implantations sont libres d'ignorer simplement les sous-réseaux déconnectés contenant des agents ϵ ou de les effacer d'un bloc.

En général, l'agent c duplique tout agent qu'il rencontre. Pour dupliquer une abstraction, nous avons besoin d'un agent auxiliaire δ qui duplique aussi n'importe quel agent, mais qui s'arrête quand il rencontre un autre agent δ. Ce mécanisme est rendu nécessaire par le *back-pointer* des agents λ, qui correspond au fait que les λ sont des lieurs. Un agent c n'interagira donc jamais avec un autre agent c (mais ça peut être le cas d'un agent δ). Ici, α représente n'importe quel agent sauf λ.



L'agent δ duplique n'importe quel agent, sauf lui-même. S'il interagit avec lui-même, il s'annihile. Ici, α représente n'importe quel agent sauf δ.

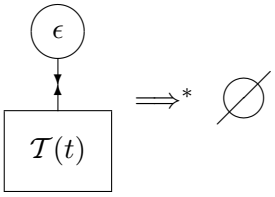


V.2.5 Propriétés

Les résultats classiques sur les paquets [Laf97, Lip02b] permettent d'établir les deux propriétés suivantes :

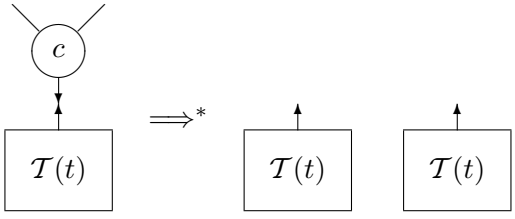
Proposition V.2.3

– si t est un λ-terme clos, alors :



(où le membre droit de la règle représente le réseau vide).

– si t est un λ-terme clos, alors :



Dans un réseau obtenu à partir de $\Downarrow\mathcal{T}(t)$, il peut y avoir plusieurs rédex impliquant des agents c, δ ou ϵ, cependant, nous avons le résultat suivant.

Proposition V.2.4

Si $\Downarrow\mathcal{T}(t) \Longrightarrow^* N$ alors, dans N, il y a exactement une occurrence de $\Downarrow$, $\Uparrow$ ou $@_{\Downarrow}$.

Preuve. Par induction, en utilisant les règles. □

Proposition V.2.5

$$t \Downarrow v \iff \Downarrow\mathcal{T}(t) \Longrightarrow^* \Uparrow\mathcal{T}(v)$$

Preuve. Il est clair que les règles d'interaction suivent fidèlement les règles de récriture de la Section V.2.2 (en utilisant la Proposition V.2.3 pour les substitutions non-linéaires), alors la Proposition V.2.2 permet de conclure. □

V.3 Appel par valeur

Dans cette section, nous montrons que nous pouvons très facilement adapter la présentation précédente à l'appel par valeur. Nous suivons la même organisation que la Section V.2, en montrant seulement les différences. Dans cette section aussi tous les termes sont supposés clos.

La stratégie d'appel par valeur pour les λ-termes clos est définie inductivement par l'ensemble de règles d'évaluation suivant (voir la Section I.5.2) :

$$\frac{}{\lambda x.t \Downarrow \lambda x.t} \qquad \frac{t \Downarrow \lambda x.t' \quad u \Downarrow v' \quad t'\{x := v'\} \Downarrow v}{t\ u \Downarrow v}$$

Nous pouvons dériver une présentation à petits pas de la stratégie d'une façon similaire à la Section V.2. Voici donc la *sémantique à passage de jeton* de l'appel par valeur :

$$\begin{array}{lcl}
\Downarrow\lambda x.t & \to & \Uparrow\lambda x.t \\
\Downarrow(t\ u) & \to & (\Downarrow t)\ u \\
(\Uparrow t)\ u & \to & t\ (\Downarrow u) \\
(\lambda x.t)\ (\Uparrow u) & \to & \Downarrow(t\{x := u\})
\end{array}$$

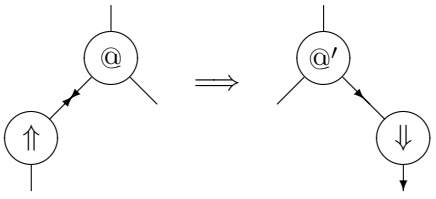
Le rôle de $\Uparrow$ est ici plus complexe qu'avec l'appel par nom : quand la partie fonction d'une application est évaluée, le contrôle est transféré à l'argument. Ensuite, quand l'argument est évalué, la β-réduction peut être effectuée.

Nous avons une propriété similaire de simulation (la preuve est aussi similaire).

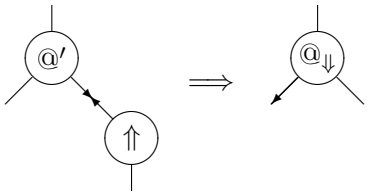
Proposition V.3.1

$$t \Downarrow v \iff \Downarrow t \to^* \Uparrow v$$

La traduction des termes en réseaux d'interaction est la même. Cependant certaines règles d'interaction doivent être légèrement modifiées, en suivant le système à petits pas. Quand le terme de gauche d'une application a fini son évaluation, nous n'effectuons plus une β-réduction immédiatement après. Au lieu de cela, nous commençons à évaluer l'argument :



Nous introduisons un nouvel agent pour l'application $@'$ dont le rôle est d'attendre que l'argument de l'application soit évalué. Quand il l'est, alors nous savons à nouveau avec certitude qu'il y a un λ à gauche, donc que nous pouvons transformer cet agent en $@_{\Downarrow}$ pour permettre à la β-réduction d'avoir lieu :



Les autres règles sont les mêmes. A nouveau, nous avons (la preuve est adaptée facilement) :

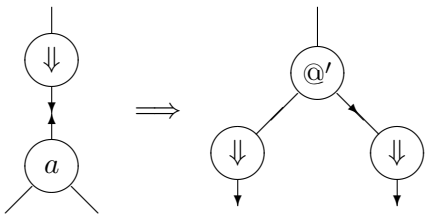
PROPOSITION V.3.2

$t \Downarrow v \iff \Downarrow\mathcal{T}(t) \Longrightarrow^* \Uparrow\mathcal{T}(v)$

Pour résumer, notre présentation s'adapte très facilement de l'appel par nom à l'appel par valeur, contrairement à certains travaux antérieurs [Lip02a]. Ce système d'interaction est très fidèle à ce qu'une fonction d'évaluation séquentielle ferait probablement. En particulier, l'agent $\Uparrow$ est nécessaire, parce que nous devons contrôler explicitement le flot de contrôle d'une manière séquentielle.

PARALLÉLISME

Naturellement, les réseaux d'interaction permettent d'évaluer la fonction et l'argument d'une application en parallèle ; c'est ce dont nous allons discuter brièvement ici. Pour garder le flot de contrôle explicite, nous devons synchroniser sur le nœud d'application quand les deux évaluations sont accomplies. Le système est obtenu à partir des mêmes règles que ci-dessus, sauf que nous remplaçons la règle d'interaction $\Downarrow \bowtie a$ par :



Maintenant dans ce système, il est encore clair que l'agent $\Uparrow$ est inutile : il n'y a aucun besoin réel de synchroniser les deux évaluations, et une β-réduction peut se produire même si l'évaluation de l'argument n'est pas encore accomplie.

Mais ce n'est pas ce qui nous intéresse. Nous préférons la version avec contrôle séquentiel et explicite (avec un jeton d'évaluation unique) parce qu'elle est vraiment plus proche d'une machine abstraite : elle est très facilement implantable sur une machine séquentielle et n'a pas besoin d'un mécanisme externe pour gérer une pile de paires actives. De plus, cette version est plus facile à généraliser à d'autres stratégies, comme nous le verrons dans les sections suivantes.

V.4 Termes ouverts

Par souci de complétude de la présentation, nous montrons comment traiter des termes ouverts. Il n'y a aucune difficulté, et aucune idée nouvelle. La présentation est faite d'une manière modulaire : nous disons seulement ce qui devrait être ajouté ou changé aux présentations de l'appel par nom et de l'appel par valeur clos pour traiter des termes ouverts.

L'évaluation en forme normale de tête faible de termes ouverts en utilisant l'appel par nom ou l'appel par valeur est faite en ajoutant au système correspondant les règles suivantes :

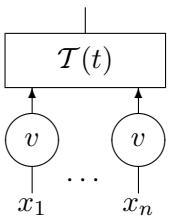
$$\frac{}{x \Downarrow x} \qquad \frac{t \Downarrow x\ u_1 \dots u_n}{t\ u \Downarrow x\ u_1 \dots u_n\ u}$$

Ou, dans un formalisme à petits pas (en conservant les autres règles) :

$$\begin{array}{lcl} \Downarrow x & \to & \Uparrow x \\ (\Uparrow x)\ u & \to & \Uparrow(x\ u) \\ (\Uparrow(v\ w))\ u & \to & \Uparrow(v\ w\ u) \end{array}$$

En ce qui concerne les réseaux d'interaction, il est clair que les variables libres vont devoir interagir avec le jeton d'évaluation, donc nous ne pouvons plus les représenter par un simple fil.

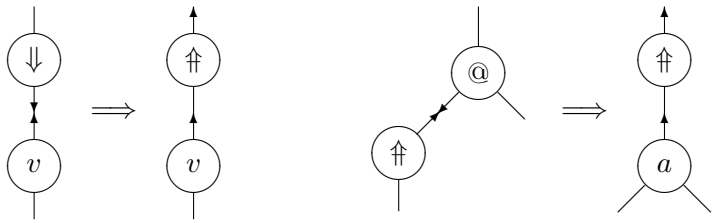
Si on applique la traduction de la Section V.2.3 à un terme t avec $\mathsf{fv}(t) = \{x_1, \dots, x_n\}$, on obtient un réseau avec n ports auxiliaires libres correspondant à $x_1, \dots, x_n$ (où les variables apparaissant plusieurs fois ont été regroupées par des agents c introduits dans la traduction des applications). On complète alors la traduction en ajoutant n agents v sur les ports auxiliaires libres, de la façon suivante :



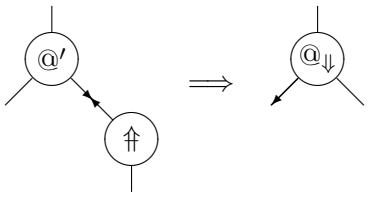
Variables. Pour résumer, si t est une variable liée, alors $\mathcal{T}(t)$ est juste un fil (à gauche). Si elle est libre dans le terme, alors $\mathcal{T}(t)$ est un agent v (à droite).



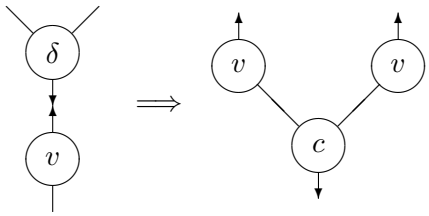
Les systèmes pour l'appel par nom et l'appel par valeur sont obtenus en ajoutant les deux règles suivantes, où nous introduisons un nouvel agent $\Uparrow\!\!\!\!\!\Uparrow$ qui est essentiellement le même que $\Uparrow$ mais qui se souvient que le terme sous lui est de la forme $x\ t_1 \dots t_n$ et pas $\lambda x.t$. C'est essentiellement la restriction aux interactions binaires qui nous oblige à introduire un tel agent.



En appel par valeur, nous devons aussi éliminer un ⇑ apparaissant à droite d'une application. Dans ce cas, ⇑ est redondant avec ⇑, donc la règle est la même :



Finalement, il peut maintenant arriver que, au cours de la duplication d'une abstraction, un agent δ rencontre un agent v. Alors, cet agent n'est jamais atteint, mais il est plus sûr de le retransformer en agent c (par exemple si l'on voulait réutiliser ce réseau) :



Un résultat similaire est alors obtenu (pour l'appel par nom ouvert et l'appel par valeur ouvert) :

Proposition V.4.1
$t \Downarrow v \iff (\Downarrow\mathcal{T}(t) \Longrightarrow^* \Uparrow\mathcal{T}(v) \text{ ou } \Uparrow\mathcal{T}(v))$

V.5 Appel par nécessité

Dans cette section, nous continuons à combler le fossé entre stratégies usuelles et réseaux d'interaction, et à plébisciter un cadre plus uniforme pour décrire les implantations du λ-calcul. Plus précisément, nous donnons une présentation de la stratégie d'appel par nécessité (voir la Section I.5.2 et ci-dessous) dans le style des sections précédentes. En comparaison avec les sections précédentes, nous améliorons le système avec la capacité de partager davantage de réductions, ce qui est un aspect majeur des réseaux d'interaction. Du point de vue de l'implantation des langages fonctionnels, nous donnons une description uniforme et entièrement formelle d'un interprète en appel par nécessité dans les réseaux d'interaction. Nous

étendons même l'approche et donnons un codage de la stratégie pleinement paresseuse (voir la Section V.6) dans les réseaux d'interaction, la rendant ainsi plus formelle et compréhensible que des présentations précédentes [Wad71, SW05]. En particulier, nous établissons un rapport avec le travail de Lang [Lan98], ce qui ne pourrait pas être fait aussi facilement sans notre cadre. Pour remettre ce travail dans le contexte adéquat, il est important de dire que, alors que l'appel par nécessité est plus ou moins devenu un standard pour les implantations des langages fonctionnels, la stratégie pleinement paresseuse, qui autorise davantage de partage de réductions, ne l'est pas. Cela montre qu'il est important d'améliorer notre compréhension de cette stratégie. Notre travail pourrait aussi constituer un pas en avant important pour mieux comprendre la réduction optimale, bien que ce ne soit pas une application immédiate.

Dans les sections précédentes, les codages sont dans les réseaux d'interaction standard, avec un agent standard appelé le jeton d'évaluation. Les réductions sont déclenchées par le jeton d'évaluation et les règles d'interaction garantissent qu'il y a une occurrence unique du jeton, la réduction est donc essentiellement déterministe. D'un autre côté, le formalisme des réseaux d'interaction impose certaines restrictions afin d'assurer la confluence forte qui ne sont plus nécessaires si les réductions sont déclenchées par un jeton unique. Dans cette section, nous devons abandonner la restriction aux réseaux d'interaction de Lafont, comme expliqué dans la Section V.5.5, et nous adoptons le formalisme d'Alexiev de réseaux d'interaction avec ports principaux multiples [Ale99], appelés plus simplement *réseaux*. Puisque la réduction est dirigée par un unique jeton, l'évaluation est toujours entièrement déterministe. Un aspect important de ce travail est donc aussi d'illustrer comment la définition des réseaux d'interaction peut être assouplie sans perdre les propriétés importantes, telles que la confluence forte. Nous avons contribué à une autre extension dans ce sens dans [SM05], orthogonale à celle que nous présentons ici, et que nous ne détaillerons pas davantage dans cette thèse.

L'appel par nécessité a été introduit par Wadsworth [Wad71]. L'idée est relativement intuitive : un sous-terme doit être évalué seulement s'il est nécessaire, et dans ce cas, il ne doit être évalué qu'une seule fois. La formulation originale est sous la forme d'un système de récriture de graphe, toutefois il y a eu plusieurs tentatives pour formaliser cette idée de différentes manières : des sémantique opérationnelles à grands pas de l'appel par nécessité ont été données indépendamment dans [Lau93] et [SI96] ; des présentations à petits pas basées sur des contextes ont été présentées dans [AFM+95, AF97, MOW98]. Contrairement à ces approches, notre formalisation est purement graphique, rendant explicite au niveau des objets la stratégie de réduction utilisée dans [Wad71]. Pour prouver la correction de notre codage, nous nous référons à une variante de la sémantique à grands pas due à Launchbury, à partir de laquelle nous dérivons une nouvelle sémantique à petits pas de l'appel par nécessité, radicalement différente des travaux précédents, et mieux adaptée à nos besoins, en suivant la même approche que dans les sections précédentes.

V.5.1 Sémantique à grands pas

La stratégie d'appel par nécessité (ou paresseuse) a été introduite par Wadsworth [Wad71] sous la forme d'un interprète graphique. Pour faciliter le raisonnement quant à l'évaluation paresseuse, Launchbury l'a exprimée sous la forme d'une sémantique naturelle (ou à grands pas) [Lau93]. Dans cette section, nous présentons une variante de cette sémantique qui est

légèrement plus simple. Dans la section suivante, nous présentons brièvement la formulation originale de Launchbury et prouvons l'équivalence des deux sémantiques.

Les environnements (par exemple Γ, Δ, Θ) sont des fonctions des variables dans les termes, de support (ou domaine) fini. On peut donc s'autoriser à les noter en extension, par exemple $(x_1 \mapsto t_1, \ldots, x_n \mapsto t_n)$. L'évaluation est définie sur des couples d'un environnement et d'un terme, de la forme $\Gamma : t$. Si $x \in \mathsf{fv}(t)$ est aussi dans le support de Γ, alors on dit que x est une variable liée dans $\Gamma : t$. Autrement dit, l'environnement se comporte comme un lieur global. Si tous les variables liées (par l'environnement ou par une λ-abstraction) dans un couple $\Gamma : t$ sont distinctes, on dira que $\Gamma : t$ est *distinctement nommé*.

Contrairement aux cas de l'appel par nom et de l'appel par valeur, l'évaluation peut modifier l'environnement comme effet secondaire. Plus précisément, quand une valeur est calculée pour une variable, cette valeur est stockée dans l'environnement pour éviter un possible recalcul si elle est requise plusieurs fois. L'évaluation est seulement définie pour les couples $\Gamma : t$ distinctement nommés et clos (si t a des variables libres, elles doivent être dans le domaine de Γ). Les jugements d'évaluation sont de la forme $\Gamma : t \Downarrow \Delta : v$, qui signifie que le terme t dans l'environnement Γ s'évalue en la valeur v avec le nouvel environnement Δ, comme défini par l'ensemble de règles de déduction suivant ($\hat{v}$ est un terme α-équivalent à v dans lequel toutes les variables liées ont été renommées en variables fraîches, voir Section I.3).

$$\frac{}{\Gamma : \lambda x.t \Downarrow \Gamma : \lambda x.t}\ \mathit{Lam}$$

$$\frac{\Gamma : t \Downarrow \Delta : \lambda x.t' \quad (\Delta, x \mapsto u) : t' \Downarrow \Theta : v}{\Gamma : t\ u \Downarrow \Theta : v}\ \mathit{App}$$

$$\frac{\Gamma : t \Downarrow \Delta : v}{(\Gamma, x \mapsto t) : x \Downarrow (\Delta, x \mapsto v) : \hat{v}}\ \mathit{Var}$$

Cette sémantique est très proche de celle de [Lau93] (voir Section V.5.2). La différence principale est que nous ne supposons pas que les termes sont précompilés dans un λ-calcul avec *lets*, de sorte que les clôtures soient déjà explicites. Nous préférons plutôt générer une nouvelle association dans l'environnement dans la règle *App* (qui correspond à la β-réduction). De plus, comme nous utilisons exactement la variable liée pour cette association, nous n'avons aucun besoin de réaliser de substitution : tout se passe grâce à l'environnement. Notre présentation est donc plus simple : il y a trois règles au lieu de quatre, il n'y aucun besoin d'un calcul avec *lets* (les définitions récursives peuvent être traitées en utilisant un point fixe standard du λ-calcul). Elle est aussi plus facile à utiliser puisque la relation d'évaluation est bien définie sur tous les termes, pas seulement sur des termes précompilés. De plus, notre sémantique est équivalente à celle de Launchbury, comme on le montrera à la Section V.5.2, ce qui justifie que c'est bien une sémantique de l'appel par nécessité et nous permet d'utiliser librement les résultats de [Lau93] au besoin. En particulier, il suffit d'effectuer l'α-conversion dans la règle *Var* seulement pour s'assurer que toutes les variables liées restent distinctes (et en particulier que l'évaluation est bien définie) :

PROPOSITION V.5.1

Si $\Gamma : t$ est clos et distinctement nommé et si $\Gamma : t \Downarrow \Delta : v$, alors $\Delta : v$ est clos et distinctement nommé.

Preuve. Preuve par induction, ou bien conséquence de la Section V.5.2 et de [Lau93]. □

V.5.2 EQUIVALENCE DES SÉMANTIQUES À GRANDS PAS

Nous rappelons la sémantique à grands pas de Launchbury pour l'appel par nécessité [Lau93] et prouvons son équivalence avec la version simplifiée présentée dans la Section V.5.1. Cette section n'est pas nécessaire pour la suite du chapitre.

DÉFINITION V.5.2

Λ_{let} est l'ensemble des λ-termes avec *lets*, définis par :

$$\begin{aligned} t, u ::= {} & x \mid \lambda x.t \mid t\ x \\ & \mid let\ x = u\ in\ t\ x \quad (\text{si } x \notin \mathsf{fv}(t)) \end{aligned}$$

DÉFINITION V.5.3

La fonction de compilation $(\cdot)^* : \Lambda \to \Lambda_{let}$ est définie par :

$$\begin{aligned} x^* &= x \\ (\lambda x.t)^* &= \lambda x.(t^*) \\ (t\ u)^* &= \begin{cases} (t^*)\ u & \text{si } u \text{ est une variable,} \\ let\ x = u\ in\ t\ x & \text{sinon } (x \text{ est une variable fraîche}). \end{cases} \end{aligned}$$

DÉFINITION V.5.4

La sémantique à grands pas de Launchbury pour l'appel par nécessité est notée $\Downarrow_L$ et est définie sur Λ_{let} comme suit. Toutes les variables liées sont supposées initialement différentes.

$$\frac{}{\Gamma : \lambda x.t \Downarrow_L \Gamma : \lambda x.t}\ Lam_L$$

$$\frac{\Gamma : t \Downarrow_L \Delta : \lambda y.t' \quad \Delta : t'\{y := x\} \Downarrow_L \Theta : v}{\Gamma : t\ x \Downarrow_L \Theta : v}\ App_L$$

$$\frac{\Gamma : t \Downarrow_L \Delta : v}{(\Gamma, x \mapsto t) : x \Downarrow_L (\Delta, x \mapsto v) : \hat{v}}\ Var_L$$

$$\frac{(\Gamma, x \mapsto u) : t \Downarrow_L \Delta : v}{\Gamma : let\ x = u\ in\ t \Downarrow_L \Delta : v}\ Let_L$$

Les termes de Λ_{let} bénéficient de certaines propriétés structurelles. D'abord, si $\Gamma : t$ est distinctement nommé et $\Gamma : t \Downarrow_L \Delta : v$, alors $\Delta : v$ est distinctement nommé [Lau93]. Autrement dit, il suffit d'effectuer l'α-conversion dans la seule règle Var_L pour garder toutes les variables liées distinctes. De plus, pour tout $t \in \Lambda_{let}$, si un sous-terme de t est une application $u\ v$, alors v est une variable, et si un sous-terme de t est de la forme $let\ x = u\ in\ v$, alors $v = w\ x$ avec $x \notin \mathsf{fv}(w)$. Il est facile de voir que la compilation $(\cdot)^*$ impose ces propriétés et que l'évaluation $\Downarrow_L$ les préserve, ce qui justifie les définitions ci-dessus, ainsi que celle-ci :

Définition V.5.5

Nous définissons la fonction de décompilation $(\cdot)^\circ : \Lambda_{let} \to \Lambda$ par :

$$\begin{aligned} x^\circ &= x \\ (\lambda x.t)^\circ &= \lambda x.(t^\circ) \\ (t\ x)^\circ &= t^\circ\ x \\ (let\ x = u\ in\ t\ x)^\circ &= t^\circ\ u^\circ \end{aligned}$$

Lemme V.5.6

Compilation et décompilation sont inverses (modulo α-conversion) :
- $(\cdot)^\circ \circ (\cdot)^* = \mathrm{id}_\Lambda$
- $(\cdot)^* \circ (\cdot)^\circ = \mathrm{id}_{\Lambda_{let}}$

Preuve. Clair. □

Nous avons aussi besoin du lemme auxiliaire suivant :

Lemme V.5.7

Si x est une variable fraîche (pour $\Gamma : t$) et u est un Λ_{let}-terme, alors $\Gamma : t \Downarrow_L \Delta : v$ si et seulement si $(\Gamma, x \mapsto u) : t \Downarrow_L (\Delta, x \mapsto u) : v$.

Preuve. Clair. □

Nous étendons $(\cdot)^*$ et $(\cdot)^\circ$ aux environnements et aux couples $\Gamma : t$ de la façon évidente. Nous identifions aussi les relations $\Downarrow$ et $\Downarrow_L$ avec leurs fonctions associées. Nous sommes désormais en position d'établir le théorème principal de cette section.

Théorème V.5.8

$\Downarrow$ et $\Downarrow_L$ sont équivalentes (modulo α-conversion) dans le sens suivant :
- $\Downarrow\ = (\cdot)^\circ \circ \Downarrow_L \circ\ (\cdot)^*$
- $\Downarrow_L = (\cdot)^* \circ \Downarrow\ \circ\ (\cdot)^\circ$

Preuve. Grâce au Lemme V.5.6, les deux points sont équivalents. Nous développons seulement le cas difficile. Supposons $\Gamma : t\ u \Downarrow \Theta : v$ et montrons que $(\Gamma : t\ u)^* \Downarrow_L \Theta' : v'$ avec $(\Theta' : v')^\circ = \Theta : v$. Par la règle *App*, il existe Δ, r tels que $\Gamma : t \Downarrow \Delta : \lambda x.r$ et $(\Delta, x \mapsto u) : r \Downarrow \Theta : v$. Par induction, il existe Δ', t' tels que $(\Gamma : t)^* \Downarrow_L \Delta' : t'$ (1) et $(\Delta' : t')^\circ = \Delta : \lambda x.r$. Par la Définition V.5.5, il existe r' tel que $t' = \lambda x.r'$ (2). Grâce au Lemme V.5.6, $\Delta' = \Delta^*$ et

$r' = r^*$ (3). Par induction encore, il existe Θ', v' tels que $((\Delta, x \mapsto u) : r)^* \Downarrow_L \Theta' : v'$ (4) et $(\Theta' : v')^\circ = \Theta : v$ (5). La dérivation suivante et le fait (5) permettent de conclure ce cas. Les deux autres cas sont faciles.

$$\dfrac{\dfrac{\dfrac{(1)+(2)}{\Gamma^* : t^* \Downarrow_L \Delta' : \lambda x.r'}}{(\Gamma^*, y \mapsto u^*) : t^* \Downarrow_L (\Delta', y \mapsto u^*) : \lambda x.r'}\,(\text{V.5.7}) \quad \dfrac{\dfrac{\dfrac{(4)}{(\Delta^*, x \mapsto u^*) : r^* \Downarrow_L \Theta' : v'}}{(\Delta', x \mapsto u^*) : r' \Downarrow_L \Theta' : v'}\,(3)}{(\Delta', y \mapsto u^*) : r'\{x := y\} \Downarrow_L \Theta' : v'}\,(=_\alpha)}{\dfrac{\dfrac{\dfrac{(\Gamma^*, y \mapsto u^*) : t^*\ y \Downarrow_L \Theta' : v'}{\Gamma^* : let\ y = u^*\ in\ t^*\ y \Downarrow_L \Theta' : v'}\,(Let_L)}{(\Gamma : t\ u)^* \Downarrow_L \Theta' : v'}\,(\text{V.5.3})}{}\,(App_L)}$$

□

V.5.3 Sémantique à petits pas

Une sémantique à grands pas fournit une description de haut niveau de la stratégie. Cependant, la stratégie elle-même n'est pas observable au niveau d'une proposition de la forme $\Gamma : t \Downarrow \Delta : v$, mais dans l'arbre de preuve de cette proposition, en utilisant les règles de déduction de la sémantique. La proposition elle-même exprime seulement une relation entre un terme d'entrée et un résultat. Dans les Sections V.5.4 et V.5.5, nous décrirons une implantation de l'appel par nécessité dans les réseaux d'interaction. Un résultat de simulation par rapport à la sémantique à grands pas ne nous semblerait pas suffisant pour prétendre que l'implantation suit effectivement la stratégie, parce qu'il pourrait y avoir des différences dans la manière dont les résultats sont obtenus qui ne peuvent pas être observées dans le résultat lui-même. Par exemple, l'appel par nom coïncide avec l'appel par nécessité, en ce qui concerne les valeurs obtenues, donc le système de la Section V.2.3 satisferait une telle propriété. C'est pourquoi nous donnons une sémantique à petits pas de l'appel par nécessité, qui est dérivée de la présentation à grands pas d'une manière systématique, de sorte qu'il est évident que les réductions du système à petits pas correspondent aux branches de déduction de celui à grands pas. Ceci facilitera également la preuve de simulation.

Nous enrichissons donc la syntaxe des termes avec les symboles $\Downarrow$ (correspondant à l'évaluation) et $\Uparrow$ (correspondant au retour de la fonction d'évaluation). Plus précisément, nous définissons l'ensemble des *termes enrichis* comme suit :

$$e ::= \Downarrow_\Gamma t \mid \Uparrow_\Gamma t \mid x \Rrightarrow e \mid C[e]$$

où t représente un λ-terme, Γ un environnement, x une variable et $C[\,]$ un contexte à un trou sur les λ-termes.

Nous pouvons maintenant définir le système de récriture suivant sur les termes enrichis :

$$\begin{array}{lllll}
(\Downarrow Lam) & \Downarrow_\Gamma \lambda x.t & \to & \Uparrow_\Gamma \lambda x.t \\
(\Downarrow App) & \Downarrow_\Gamma (t\ u) & \to & (\Downarrow_\Gamma t)\ u \\
(\Uparrow App) & (\Uparrow_\Gamma \lambda x.t)\ u & \to & \Downarrow_{(\Gamma, x \mapsto u)} t \\
(\Downarrow Var) & \Downarrow_{(\Gamma, x \mapsto t)} x & \to & x \Rrightarrow \Downarrow_\Gamma t \\
(\Uparrow Var) & x \Rrightarrow \Uparrow_\Gamma v & \to & \Uparrow_{(\Gamma, x \mapsto v)} \hat{v}
\end{array}$$

C'est un système de récriture standard : la substitution est réalisée par l'environnement et la réduction est autorisée dans n'importe quel contexte. C'est une présentation à petits pas de l'appel par nécessité, comme le montrera la Proposition V.5.10. D'autres présentations à petits pas de l'appel par nécessité incluent [AFM+95, AF97, MOW98]. La nôtre est radicalement différente, et probablement plus simple. La simplicité vient du fait que nous rendons le flot d'évaluation explicite au niveau syntaxique, tandis que les travaux cités construisent des contextes d'évaluation astucieux pour restreindre la réduction et codent les environnements sous forme de termes d'une manière peu naturelle.

Un autre point de comparaison est avec les machines abstraites telles que la machine de Krivine paresseuse [Cré90], par exemple. Notre présentation est probablement plus proche de ces travaux, bien que plus abstraite. Mais ce qui nous intéresse surtout, c'est qu'elle a exactement le degré d'abstraction adapté à nos besoins.

A notre connaissance, c'est la première fois qu'est donnée une présentation à petits pas aussi simple de l'appel par nécessité. Les présentations à petits pas habituelles utilisent plutôt des règles inductives autorisant des réductions dans une certaine classe de contextes, et par conséquent ne donnent pas un statut explicite au flot d'évaluation, contrairement à notre présentation, ce qui est crucial pour le codage dans les réseaux d'interaction. En un sens, notre présentation est intermédiaire entre la sémantique traditionnelle à petits pas (qui sépare autant que possible réduction et stratégie) et les machines abstraites (qui peuvent utiliser des structures de données complexes). Nous appelons cette présentation la *sémantique à passage de jeton* de l'appel par nécessité.

Pour évaluer t dans un contexte Γ, nous commençons la réduction à partir de $\Downarrow_\Gamma t$. En général, si Γ n'est pas défini pour certaines variables libres de t, l'évaluation en appel par nécessité de $\Gamma : t$ peut échouer. Cela correspond ici à la réduction partant de $\Downarrow_\Gamma t$ qui termine sur un terme enrichi qui n'est pas de la forme $\Uparrow_\Delta u$. Si t est clos, ce qui est généralement supposé, nous pouvons choisir Γ vide sans danger.

Ce système vérifie l'invariant fort suivant :

PROPOSITION V.5.9

Si $\Downarrow_\Gamma t \to^* u$, alors il y a exactement une occurrence de $\Downarrow_\Delta$ ou $\Uparrow_\Delta$ dans u (pour un certain Δ).

Preuve. Par induction sur la longueur de la réduction et par cas sur la dernière règle utilisée. Dans la règle $\Uparrow Var$, v est copié mais n'a pas d'occurrence de $\Downarrow$ ou $\Uparrow$ par l'hypothèse d'induction. □

Comme une réduction implique toujours un $\Downarrow_\Gamma$ ou $\Uparrow_\Gamma$, il y a donc toujours au plus un rédex dans un terme obtenu par réduction à partir de $\Downarrow_\Gamma t$ et le flot de contrôle est effectivement rendu explicite au niveau syntaxique.

PROPOSITION V.5.10

$\Gamma : t \Downarrow \Delta : v \iff \Downarrow_\Gamma t \to^* \Uparrow_\Delta v$

Preuve. Les deux implications sont systématiques.

⇒ Par induction sur la dérivation :

- $Lam : \Gamma : \lambda x.t \Downarrow \Gamma : \lambda x.t$ et en effet $\Downarrow_\Gamma \lambda x.t \xrightarrow[\Downarrow Lam]{} \Uparrow_\Gamma \lambda x.t$.

- *App* : si $\Gamma : t\ u \Downarrow \Theta : v$, alors il existe t' et Δ tels que $\Gamma : t \Downarrow \Delta : \lambda x.t'$ et $(\Delta, x \mapsto u) : t' \Downarrow \Theta : v$. Par induction, $\Downarrow_\Gamma t \to^* \Uparrow_\Delta \lambda x.t'$ et $\Downarrow_{(\Delta, x\mapsto u)} t' \to^* \Uparrow_\Theta v$, donc : $\Downarrow_\Gamma (t\ u) \xrightarrow[\Downarrow App]{} (\Downarrow_\Gamma t)\ u \to^* (\Uparrow_\Delta \lambda x.t')\ u \xrightarrow[\Uparrow App]{} \Downarrow_{(\Delta, x\mapsto u)} t' \to^* \Uparrow_\Theta v$.
- *Var* : si $(\Gamma, x \mapsto t) : x \Downarrow (\Delta, x \mapsto v) : \hat{v}$, alors $\Gamma : t \Downarrow \Delta : v$. Donc par induction, $\Downarrow_\Gamma t \to^* \Uparrow_\Delta v$, et finalement : $\Downarrow_{(\Gamma, x\mapsto t)} x \xrightarrow[\Downarrow Var]{} (x \Rrightarrow \Downarrow_\Gamma t) \to^* (x \Rrightarrow \Uparrow_\Delta v) \xrightarrow[\Uparrow Var]{} \Uparrow_{(\Delta, x\mapsto v)} \hat{v}$.

$\Leftarrow$ Par induction structurelle sur le terme t :

- Abstraction : $\Downarrow_\Gamma \lambda x.t \xrightarrow[\Downarrow Lam]{} \Uparrow_\Gamma \lambda x.t$ et en effet $\Gamma : \lambda x.t \Downarrow \Gamma : \lambda x.t$ par (Lam).
- Application : supposons $\Downarrow_\Gamma (t\ u) \to^* \Uparrow_\Theta v$. Nous avons donc $\Downarrow_\Gamma (t\ u) \xrightarrow[\Downarrow App]{} (\Downarrow_\Gamma t)\ u \to^* \Uparrow_\Theta v$. Le résultat final n'est plus une application, et la règle $\Uparrow App$ est la seule qui peut réaliser cela. Donc nous devons avoir $\Downarrow_\Gamma t \to^* \Uparrow_\Delta \lambda x.t'$ pour certains Δ et t'. De plus, $\Gamma : t \Downarrow \Delta : \lambda x.t'$ par induction. Alors $(\Uparrow_\Delta \lambda x.t')\ u \xrightarrow[\Uparrow App]{} \Downarrow_{(\Delta, x\mapsto u)} t' \to^* \Uparrow_\Theta v$, donc $(\Delta, x \mapsto u) : t' \Downarrow \Theta : v$ par induction et $\Gamma : t\ u \Downarrow \Theta : v$ par la règle (App).
- Variable : supposons $\Downarrow_{(\Gamma, x\mapsto t)} x \xrightarrow[\Downarrow Var]{} (x \Rrightarrow \Downarrow_\Gamma t) \to^* \Uparrow_{(\Delta, x\mapsto v)} \hat{v}$. La dernière règle appliquée doit être $(x \Rrightarrow \Uparrow_\Delta v) \xrightarrow[\Uparrow Var]{} \Uparrow_{(\Delta, x\mapsto v)} \hat{v}$, donc nécessairement $\Downarrow_\Gamma t \to^* \Uparrow_\Delta v$ et par induction $\Gamma : t \Downarrow \Delta : v$. Grâce à (Var), nous obtenons $(\Gamma, x \mapsto t) : x \Downarrow (\Delta, x \mapsto v) : \hat{v}$.

□

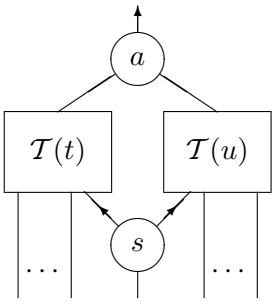
Le système de récriture ci-dessus correspond donc fidèlement à la stratégie d'appel par nécessité. En particulier, comme pour la sémantique à grands pas, si toutes les variables liées sont initialement distinctes, cela est préservé par réduction. Cette étape est cruciale, dans la mesure où le codage dans les réseaux d'interaction suivra fidèlement la sémantique à petits pas ci-dessus.

V.5.4 Codage des termes

C'est essentiellement le même codage que dans la Section V.2.3. La seule différence est que les agents c sont remplacés par des agents s représentant le partage, qui ont deux ports principaux orientés vers le haut, de sorte que la copie n'est pas possible tant qu'un agent n'arrive pas par le haut (le jeton d'évaluation aura la responsabilité d'activer un agent de partage en agent de copie). Cela contraste avec les codages habituels (incluant ceux des Sections V.2 et V.3) où un agent standard est employé, avec un seul port principal orienté vers le bas, de sorte qu'il puisse effectuer la copie immédiatement après une β-réduction. L'agent c des codages précédents sera produit par certaines règles d'interaction, voir la Section V.5.5. Nous ne donnons le codage que de l'application, les cas variable et abstraction sont strictement identiques aux cas correspondants de la Section V.2.3.

Application. Comme précédemment, la traduction d'une application est simplement un agent a dont le port principal pointe vers la racine, et dont les port auxiliaires sont reliés aux sous-termes. Les variables libres communes sont regroupées par des agents s (représentant le partage). Les agents s ont deux ports principaux orientés vers le haut, de

sorte que la copie ne commencera pas avant qu'un agent (le jeton d'évaluation) n'arrive du haut. Ce seront les seuls agents du système avec plus d'un port principal.



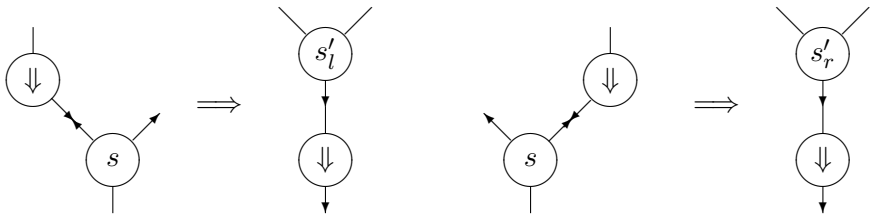
Ici aussi, la représentation des λ-termes est très simple, et sans codage de boîtes. Pour le moment, nous n'avons que quatre agents : λ et a pour les constructions linéaires ; ϵ et s pour la gestion de ressources explicite. La traduction d'un terme n'a aucune paire active, et a exactement un port principal libre, relié à la racine.

V.5.5 Evaluation par interaction

Dans cette section, nous donnons les règles d'interaction pour l'appel par nécessité. La différence entre appel par nom et appel par nécessité est seulement visible quand il est question de partage. Par conséquent, la partie linéaire des règles est identique à celle du codage de l'appel par nom de la Section V.2.4. Il n'y a pas de raison non plus de changer la façon dont la copie et l'effacement sont effectués, nous reprenons donc les règles de la Section V.2.4 et donnons quelques compléments. La vraie différence est seulement *quand* la copie doit être effectuée, c'est-à-dire quand un agent de partage doit être activé en agent de copie.

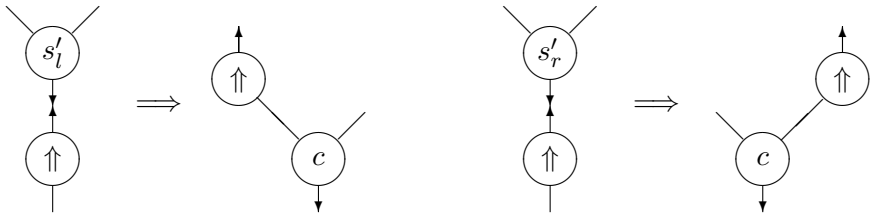
Partage

Le partage est représenté par les agents s. Quand le jeton d'évaluation atteint un agent s, cela signifie que l'évaluation du sous-terme partagé est requise. C'est fait très simplement en descendant le jeton d'évaluation sur le sous-terme partagé. L'agent s est alors renommé en un agent s' orienté vers le bas, de sorte que l'interaction sera possible quand le jeton reviendra. Nous devons aussi nous rappeler si le jeton vient de la gauche ou de la droite de l'agent s, de façon à continuer à partir de la même position. C'est pourquoi nous introduisons en fait les agents s'_l et s'_r.



Quand le jeton revient à un agent s', alors nous initions la copie avec un agent c et propageons le jeton à la position originale (gauche ou droite, comme indiqué par l'agent). Il

n'est pas nécessaire de recommencer l'évaluation (c'est-à-dire de produire un jeton $\Downarrow$) parce que le sous-terme est déjà évalué (Lemme V.5.14). (Ce ne sera pas toujours le cas pour la réduction pleinement paresseuse, voir la Section V.6).



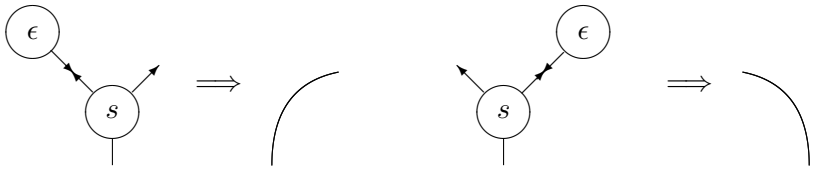
Remarque V.5.11

En raison de notre codage et parce que nous suivons une stratégie normale (nous évaluons toujours en premier le terme de gauche dans une application), le jeton atteindra souvent un agent s sur son port gauche. Cependant, ce n'est pas toujours vrai, par exemple dans le terme $(\lambda x.(\lambda y.\lambda z.z\,y)\,x\,x)\,(\lambda u.u)$. C'est pourquoi nous avons dû abandonner la restriction aux réseaux d'interaction de Lafont.

Copie et effacement

Les règles de copie et d'effacement sont essentiellement les mêmes que dans la Section V.2.4. Par souci de clarté, le schéma d'effacement $\epsilon \bowtie \alpha$ de la Section V.2.4 est ici valable pour $\alpha \in \{\lambda, a, @, \epsilon, c, \delta, s''\}$ (s'' est introduit ci-dessous).

L'effacement des agents s est, quant à lui, atypique, bien que très naturel. L'intuition en est simplement que si un terme est partagé et que l'une des deux copies n'est pas utilisée, alors le partage est inutile. De telles règles ne peuvent pas être définies dans les réseaux d'interaction standard, et cette règle est en fait souvent considérée dans la littérature comme une équivalence sur les réseaux.

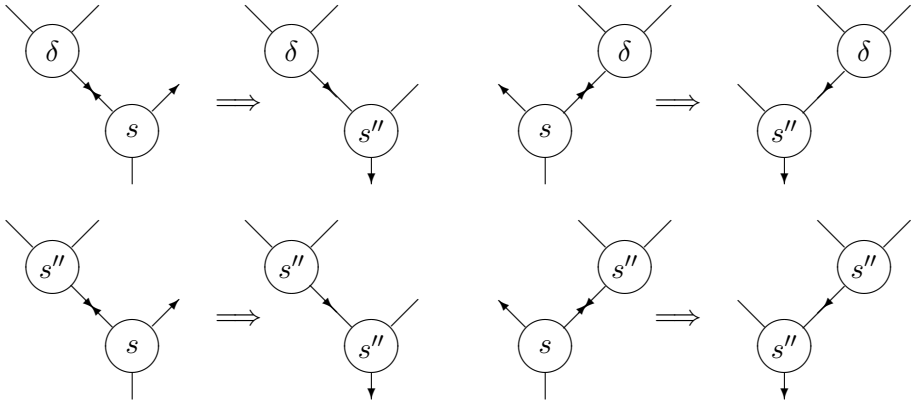


Pour la copie, le schéma $c \bowtie \alpha$ de la Section V.2.4 est ici valable pour $\alpha \in \{a, \epsilon\}$ (λ est exclu et les autres cas n'arrivent pas). Le schéma $\delta \bowtie \alpha$ est valable pour $\alpha \in \{\lambda, a, @, \epsilon, c\}$.

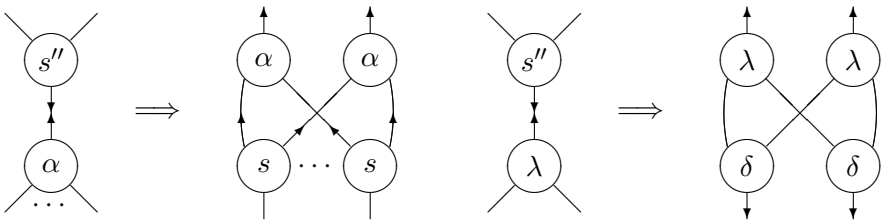
Un agent c essaie toujours de copier un sous-réseau clos et évalué, donc les agents c et s ne se rencontrent jamais (parce que l'agent s aurait d'abord été activé, voir le Lemme V.5.14). Cependant, il se peut que nous devions copier un agent s à l'intérieur d'une abstraction avec un δ. Nous ne pouvons pas utiliser la règle générique $\delta \bowtie \alpha$ avec $\alpha = s$ parce que l'agent s peut ne pas être entouré d'agents δ, et cela peut causer un "échappement" des agents δ, comme on peut le voir en réduisant le terme $(\lambda x.(\lambda y.y\,y\,x)\,(\lambda z.z\,x))\,(\lambda i.i)$.

Quand un agent δ interagit avec un agent s, l'agent s est "activé" en un nouvel agent s'' qui se comporte comme un agent c paresseux : il effectue seulement un pas de copie, et redevient

un agent s (il se "désactive"). Moralement, cela évite le problème de copier les agents s : nous ne les copions pas, nous leur ordonnons plutôt de copier. Plus précisément, les agents s'' sont introduits de la façon suivant :



Et ils interagissent de la façon suivante ($\alpha \in \{a, \epsilon, \delta\}$).



Nous n'essayons pas de préserver le partage à l'intérieur d'une abstraction (la règle $s'' \bowtie \lambda$ génère des agents δ au lieu d'agents s). Le partage obtenu est donc exactement l'appel par nécessité au sens habituel (par exemple comme en Haskell et par opposition à la réduction pleinement paresseuse) : le partage a lieu au niveau global, mais aucune réduction n'est partagée à l'intérieur d'une abstraction. Un raffinement de ce point mènera à une implantation de la stratégie pleinement paresseuse (*fully lazy*) [Wad71] dans la Section V.6.

V.5.6 Propriétés

Réduction

Dans cette section, nous donnons quelques propriétés des règles d'interaction : principalement la préservation de certaines propriétés structurelles des réseaux et la confluence forte.

Définition V.5.12

> Les règles peuvent être partitionnées en *règles d'évaluation* qui impliquent un agent $\Downarrow$, $\Uparrow$ ou $@_{\Downarrow}$, et en *règles administratives* qui impliquent les agents c, s'', δ ou ϵ. On remarque que c'est effectivement une partition. Nous notons $\underset{ev}{\Longrightarrow}$ une réduction avec une règle d'évaluation, et $\underset{adm}{\Longrightarrow}$ pour une réduction avec une règle administrative.

Définition V.5.13

Un réseau N est *valide* s'il existe un λ-terme t tel que $\Downarrow\mathcal{T}(t) \Longrightarrow^* N$.

Désormais, tous les réseaux sont supposés valides. Initialement, un réseau correspond à l'arbre de syntaxe d'un terme, donc il y a une notion naturelle d'orientation de ce réseau et de ses agents, qui peut être rendue complètement formelle en utilisant des types [Laf90]. De plus, cette orientation est préservée par réduction, au sens large pour $\lambda \bowtie @_{\Downarrow}$ (si α est en-dessous de β dans M et $M \underset{\lambda\bowtie@_{\Downarrow}}{\Longrightarrow} N$ en ne touchant pas α et β, alors, dans N, soit α est toujours en-dessous de β, soit ils sont des composantes déconnectées), strictement pour les autre règles. Nous pouvons maintenant établir quelques propriétés structurelles des réseaux valides.

Lemme V.5.14

Dans un réseau valide :

1. $\Downarrow$ et $\Uparrow$ ne sont jamais en-dessous d'un λ ou a ;
2. $\Downarrow$ et $\Uparrow$ ne sont jamais au-dessus d'un $@$;
3. $\Downarrow$ et $\Uparrow$ ne sont jamais en-dessous d'un s, c, δ ou s'' ;
4. s'il y a un $\Uparrow$, alors il y a un λ en-dessous de lui, et il ne peut y avoir que des agents c entre les deux ;
5. s'il y a un s en-dessous d'un c, alors il y a un λ entre eux ;
6. s'il y a un δ et un $\Downarrow$ ou $\Uparrow$ dans la même composante connexe, alors ce premier est en-dessous de ce dernier.

Preuve. Par induction : toutes les propriétés sont vraies pour $\Downarrow\mathcal{T}(t)$ et sont préservées par réduction (en vérifiant toutes les règles si les agents sont impliqués dans la réduction, par la remarque ci-dessus sinon). □

Dans un réseau valide, il peut y avoir plusieurs rédex administratifs, cependant, nous avons le résultat suivant.

Proposition V.5.15

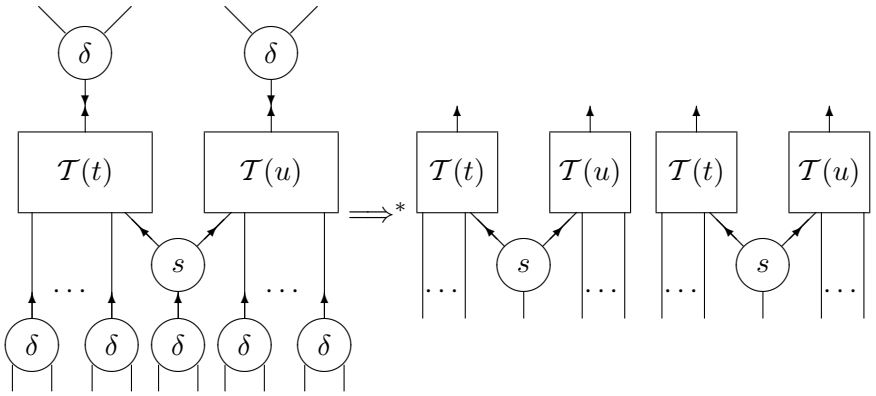
Dans un réseau valide, il y a exactement une occurrence de $\Downarrow$, $\Uparrow$ ou $@_{\Downarrow}$, donc il y a au plus un rédex d'évaluation.

Preuve. C'est vrai pour $\Downarrow\mathcal{T}(t)$ et toutes les règles préservent la propriété. □

Proposition V.5.16

La Proposition V.2.3 est encore valide ici.

Preuve. Par induction sur la structure de t. Pour le deuxième point, nous avons besoin du lemme auxiliaire suivant, facilement prouvé par induction.



□

Notre système n'est pas un système de réseau d'interaction dans le sens de Lafont à cause du seul agent s, mais c'est suffisant pour invalider la confluence forte. Cependant, on la retrouve sous deux conditions : on considère seulement des réseaux valides (donc des réseaux avec seulement un jeton) et on enlève les règles d'effacement (règles impliquant ϵ). Nous insistons à nouveau sur le fait que ces règles ne sont pas essentielles pour le progrès de l'évaluation (ne pas les appliquer ne bloquera pas le jeton d'évaluation).

Proposition V.5.17

> La réduction (en enlevant les règles d'effacement) est fortement confluente sur les réseaux valides. Autrement dit, si M est un réseau valide tel que $M \Longrightarrow P$ et $M \Longrightarrow Q$ (avec $P \neq Q$), alors il existe un réseau N tel que $P \Longrightarrow N$ et $Q \Longrightarrow N$.

Preuve. Dans un réseau valide, il y a au plus une règle d'évaluation applicable (par la Proposition V.5.15), donc au moins une des réductions est administrative. Mais un agent s ne peut pas avoir un δ sur un port et un jeton d'évaluation sur l'autre, grâce au Lemme V.5.14 et à la Proposition V.5.15. Par conséquent, il n'y a pas de chevauchement entre les règles d'évaluation et administratives : dans ce cas, les réductions sont indépendantes et la paire divergente peut être jointe en appliquant l'autre règle. Le cas restant implique donc deux règles administratives. A nouveau, si elles sont appliquées à des endroits différents, la paire est facile à joindre. Les agents c et s ne peuvent pas se rencontrer grâce au Lemme V.5.14 (point 5), donc les seuls cas restants impliquent un agent s et des agents δ ou s'' sur ses deux ports principaux. Dans toutes ces configurations, les deux réductions amènent au même réseau où l'agent s a été activé en un agent s''. Ceci conclut la preuve. □

Maintenant si nous remettons les règles d'effacement dans le système, la confluence n'est pas vraie en général en raison du chevauchement entre les règles $\epsilon \bowtie s$ et $\Downarrow \bowtie s$: si le terme correspondant au réseau en-dessous de l'agent s n'est pas faiblement normalisant, alors les réductions divergentes ne se rejoindront jamais. Ce problème pourrait être résolu en ajoutant des ports principaux à s'_l et s'_r et des règles d'interaction avec ϵ, mais cela a peu d'intérêt, au moins d'un point de vue d'implantation. Cependant, la réduction est confluente sur les λ-termes qui ont une forme normale de tête faible, comme le montre le Corollaire V.5.22 (et le contre-exemple ci-dessus montre qu'on ne peut pas espérer un meilleur résultat de confluence en général). Il est également intéressant de noter que, dans le folklore des réseaux d'interaction, un agent ϵ relié à un port auxiliaire d'un agent c est souvent considéré comme équivalent à un

simple fil [AG98]. Ceci correspond partiellement à la règle d'interaction $\epsilon \bowtie s$ dans notre cadre, ce qui est possible parce que l'agent s a deux ports principaux, mais il n'est pas surprenant que cela ne résolve pas tous les problèmes.

Une remarque plus générale au sujet des réseaux à passage de jeton est que la réduction a plus de chance d'être confluente que dans les systèmes de réseaux sans jeton, comme le montre la Proposition V.5.17 pour notre système particulier. Une autre remarque est qu'un certain coût en termes de nombre d'étapes d'interaction est dû au jeton. Cependant, il vaut mieux voir ceci comme un coût rendu explicite plutôt que comme un surcoût : dans les implantations standard de réseaux d'interaction de Lafont, il est nécessaire qu'un mécanisme externe (une machine abstraite) trouve le prochain rédex [Pin00]. Au contraire, dans les réseaux à passage de jeton, il est toujours suffisant de réduire à proximité du jeton.

Simulation

Nous voulons maintenant exhiber une propriété de simulation démontrant que le système de récriture de réseaux donné dans la Section V.5.5 est effectivement une implantation du système de récriture de la Section V.5.3, et donc de l'appel par nécessité. Cependant, il n'y a aucun intérêt à rechercher une correspondance exacte : nous ne devons pas nous soucier si les représentations des objets diffèrent, tant que les résultats des calculs correspondent. Nous considérerons donc les termes et les réseaux modulo une certaine équivalence préservant la signification des objets. Dorénavant, $\Updownarrow$ représentera $\Downarrow$ ou $\Uparrow$.

Définition V.5.18 (*Equivalence sur les termes*)

Nous définissons $\sim$ comme la plus petite relation symétrique, réflexive et transitive telle que :

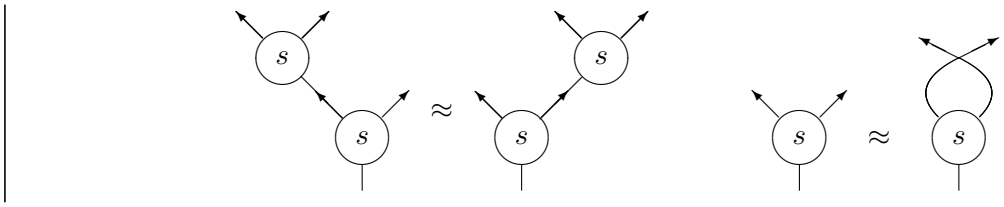
- pour tout contexte à deux trous $C[\,][\,]$ et valeur v, $C[\Updownarrow_{(\Gamma, x \mapsto v)} t][x] \sim C[\Updownarrow_{(\Gamma, x \mapsto v)} t][v]$;
- pour tout contexte $C[\,]$, si $e = C[\Updownarrow_{(\Gamma, x \mapsto u)} t]$ et $e' = C[\Updownarrow_{\Gamma} t]$, si $x \notin \mathsf{fv}(e')$ alors $e \sim e'$.

Dans la première clause de la définition ci-dessus, nous avons besoin d'un contexte avec deux trous parce que l'association $x \mapsto v$ n'est pas locale au sous-terme t, mais à tout le terme (dans la notation à grands pas $\Gamma : t$, l'environnement Γ est un lieur global). De plus, il n'y a aucun problème de capture puisque toutes les variables liées sont distinctes (on le suppose dans le terme initial et c'est préservé par réduction).

Définition V.5.19 (*Equivalence sur les réseaux*)

Nous définissons $\approx$ comme la plus petite relation symétrique, réflexive, transitive, stable par contexte de réseau (c'est-à-dire qu'en connectant deux réseaux équivalents dans le même réseau, on obtient deux réseaux équivalents) et telle que :

- $a \approx @ \approx @_{\Downarrow}$, $s \approx s'_l \approx s'_r$;
- $\underset{adm}{\Longrightarrow} \subset \approx$;
- s est associatif et commutatif, autrement dit :



Une autre chose dont nous devons prendre soin avant d'exhiber une propriété de simulation est de définir une notion raisonnable de correspondance entre les états intermédiaires pendant la réduction d'un terme et d'un réseau. Cela revient à étendre la fonction de traduction $\mathcal{T}(\cdot)$ des λ-termes (ce qui était suffisant pour commencer le processus) aux termes enrichis.

En général, un terme enrichi e est de la forme : $e = C_0[x_1 \Rrightarrow C_1[\ldots x_n \Rrightarrow C_n[\Updownarrow_\Gamma t]\ldots]]$ où t est un λ-terme, $x_1, \ldots, x_n$ sont des variables et $C_0[\,], \ldots, C_n[\,]$ sont des contextes sur les λ-termes.

Nous étendons maintenant $\mathcal{T}(\cdot)$ étape par étape :

- $\mathcal{T}(t)$ est déjà défini si t est un λ-terme clos. Si t a des variables libres $x_1, \ldots, x_n$ (qui sont en fait liées par l'environnement), nous supposons que les ports libres correspondant sont temporairement étiquetés avec la bonne variable. L'étiquette ne fera pas partie de la traduction finale : elle est seulement utilisée pour la construire. Contrairement à la Section V.4, on ne met pas d'agents v sur ces ports. Les variables sont donc simplement traduites en arêtes.
- Nous définissons $\mathcal{T}(\Updownarrow t)$ (environnement vide) comme dans la Section V.2.3 en connectant un agent $\Downarrow$ ou $\Uparrow$ à la racine de $\mathcal{T}(t)$.
- $\mathcal{T}(C[\Updownarrow_\Gamma t])$ où $\Gamma = (x_1 \mapsto t_1, \ldots, x_n \mapsto t_n)$ est défini en construisant $\mathcal{T}(C[\Updownarrow t])$ et $\mathcal{T}(t_i)$ pour tout i, et en connectant chaque port étiqueté x_i dans $\mathcal{T}(C[\Updownarrow t])$ à la racine de $\mathcal{T}(t_i)$.
- De façon similaire, pour $\mathcal{T}(C[x \Rrightarrow t])$, nous construisons $\mathcal{T}(C[x])$ et $\mathcal{T}(t)$ et connectons le port libre étiqueté x du premier à la racine du dernier.

Remarquons que $\mathcal{T}(\cdot)$ produit des réseaux qui ne sont valides que modulo $\approx$. Par exemple, $\mathcal{T}((\Downarrow_\Gamma t)\ u)$ a un agent $\Downarrow$ sous un agent a, ce qui ne peut pas arriver dans une réduction : l'agent a aurait dû être transformé en @. Toutefois, tout se passe bien modulo $\approx$, dans le sens des deux propositions suivantes.

Proposition V.5.20 (*Complétude*)

Pour deux termes enrichis t et u, si $t \to u$, alors il existe M et N tels que $\mathcal{T}(t) \approx M$, $M \Longrightarrow^* N$ et $N \approx \mathcal{T}(u)$.

Preuve. Par cas sur la règle utilisée.

- $(\Downarrow Lam)$: clair ;
- $(\Downarrow App)$: clair ;
- $(\Uparrow App)$: clair (deux pas de réduction) ;
- $(\Downarrow Var)$: disons $\Downarrow_{(\Gamma, x \mapsto t)} x \to x \Rrightarrow \Downarrow_\Gamma t$, il y a deux cas :
  - si l'occurrence de x est unique dans le terme ou si t a déjà été évalué, alors la variable x est juste représentée par un fil, et les deux traductions sont égales : $\mathcal{T}(C[\Downarrow_{(\Gamma, x \mapsto t)} x]) = \mathcal{T}(C[x \Rrightarrow \Downarrow_\Gamma t])$;
  - sinon, cela correspond à une règle $\Downarrow \bowtie s$, qui transforme s en s'_l ou s'_r (équivalents

modulo $\approx$) ;
- ($\Uparrow Var$) : il y a encore deux cas :
  - dans le cas linéaire, les traductions sont les mêmes des deux côtés ;
  - dans le cas non-linéaire, cela correspond à une règle $\Uparrow \bowtie s'_l$ ou $\Uparrow \bowtie s'_r$.

□

Nous ne pouvons pas imposer $M = \mathcal{T}(t)$ dans la proposition ci-dessus parce que les ports principaux dans $\mathcal{T}(t)$ peuvent être orientés de façon incorrecte (comme dans $\mathcal{T}((\Downarrow_\Gamma t)\ u)$), mais il y a effectivement un représentant correctement orienté dans la classe de $\mathcal{T}(t)$.

Pour l'autre direction, il y a encore un détail dont nous devons prendre soin : la règle $\Uparrow \bowtie @$ crée un réseau intermédiaire qui ne correspond pas à un terme valide jusqu'à ce que la règle $\lambda \bowtie @_\Downarrow$ correspondante soit appliquée. Nous ne raisonnons donc pas sur la réduction $\Longrightarrow$ mais plutôt sur $\Longrightarrow\!\!\!\!\!\Rightarrow$, définie comme suit :
- $M \Longrightarrow\!\!\!\!\!\Rightarrow N$ s'il existe P tel que $M \underset{\Uparrow\bowtie@}{\Longrightarrow} P \underset{\lambda\bowtie@_\Downarrow}{\Longrightarrow} N$;
- $M \Longrightarrow\!\!\!\!\!\Rightarrow N$ si $M \Longrightarrow N$ par une autre règle.

Proposition V.5.21 (*Correction*)

Si $M \Longrightarrow\!\!\!\!\!\Rightarrow N$ et $M \approx \mathcal{T}(t)$ pour un certain terme enrichi t alors il existe t' et u tels que $t \sim t'$, $t' \to^* u$ et $N \approx \mathcal{T}(u)$.

Preuve. Par cas sur la famille de règles utilisée.
- règles $\Downarrow \bowtie \lambda$, $\Downarrow \bowtie a$, $\Uparrow \bowtie @ + \lambda \bowtie @_\Downarrow$: clair (certaines règles qui déplacent des variables entre l'environnement et les constructions de la forme $x \Mapsto t$ peuvent être appliquées implicitement dans le cas linéaire) ;
- règles $\Downarrow \bowtie s$: t' est de la forme $C[\Downarrow_{(\Gamma, x\mapsto v)} x]$ (avec x non-linéaire dans t'), et en effet $\mathcal{T}(C[x \Mapsto \Downarrow_\Gamma v]) \approx N$;
- règles $\Uparrow \bowtie s'_{l/r}$: similaire ;
- règles administratives : vrai avec $t = t' = u$, puisque $\approx$ contient cette étape de $\Longrightarrow\!\!\!\!\!\Rightarrow$-réduction.

□

Corollaire V.5.22 (*Simulation de l'appel par nécessité*)

- si $\Gamma : t \Downarrow \Delta : v$, alors $\mathcal{T}(\Downarrow_\Gamma t) \Longrightarrow^* \mathcal{T}(\Uparrow_\Delta v)$;
- si $\mathcal{T}(\Downarrow_\Gamma t) \Longrightarrow^* \mathcal{T}(\Uparrow_\Delta v)$, alors il existe v', Δ' tels que $\Uparrow_\Delta v \sim \Uparrow_{\Delta'} v'$ et $\Gamma : t \Downarrow \Delta' : v'$.

Preuve. Nous rassemblons simplement les résultats.
- Par les Propositions V.5.10 et V.5.20, il y a des réseaux M et N tels que $\mathcal{T}(\Downarrow_\Gamma t) \approx M$, $N \approx \mathcal{T}(\Uparrow_\Delta v)$ et $M \Longrightarrow^* N$, et nous pouvons choisir N en forme normale sans perte de généralité. Puisque le jeton est à la racine, il n'y a aucun s' dans M ou N et il n'y a aucun danger à choisir de marquer les applications a plutôt que @. Donc, on a bien $\mathcal{T}(\Downarrow_\Gamma t) \Longrightarrow^* \mathcal{T}(\Uparrow_\Delta v)$.
- Remarquons d'abord que le réduit a un jeton, donc la $\Longrightarrow^*$-réduction est aussi une $\Longrightarrow\!\!\!\!\!\Rightarrow^*$-réduction. Nous utilisons les Propositions V.5.10 et V.5.21, et un argument similaire à celui utilisé ci-dessus permet de s'affranchir de $\approx$. Cependant, nous ne pouvons pas

nous débarrasser de $\sim$, car un réseau peut être décompilé en plusieurs termes enrichis différents.

□

On remarque que ce corollaire est plus faible que les propositions, puisque c'est seulement une propriété de simulation sur les formes normales, et par sur les réductions.

V.6 Réduction pleinement paresseuse

La stratégie d'appel par nécessité, pour laquelle nous avons donné ci-dessus une implantation utilisant la récriture de réseaux, capture seulement le partage des *valeurs*. Par exemple, l'évaluation du terme $(\lambda f.fI(fI))(\lambda w.(II)\,w)$ où $I = \lambda x.x$ évaluera le rédex II deux fois, parce que le sous-terme $\lambda w.(II)\,w$ sera partagé, puis copié en entier quand ce sera nécessaire. C'est ce qui arrive dans l'implantation ci-dessus, ainsi que dans les implantations standard de l'appel par nécessité, par exemple dans la G-machine [PJS89]. Ce n'est en général pas considéré comme un problème, parce que ce terme peut aussi être transformé en $(\lambda f.fI(fI))((\lambda z.(\lambda w.z\,w))(II))$ dans lequel le rédex II ne sera évalué qu'une fois. Cette transformation est appelée *fully lazy λ-lifting* [PJ87, Chapitre 15].

Toutefois, il est aussi possible de partager directement l'évaluation de ce rédex. Les implantations qui le permettent sont appelées *pleinement paresseuses* (*fully lazy* en anglais). Wadsworth est le premier à avoir décrit un interprète pleinement paresseux [Wad71] : il a remarqué que le rédex II ne doit pas être copié puisqu'aucune occurrence de la variable liée w n'apparaît dans ce terme. Cependant, son algorithme n'est ni intuitif ni efficace ; il est donc naturel d'essayer d'adapter le travail ci-dessus à la réduction pleinement paresseuse.

L'évaluateur proposé dans la Section V.5.5 repose d'une manière tout à fait cruciale sur les faits, d'une part, qu'à chaque fois qu'une copie est commencée sur un terme clos, elle peut toujours être menée à bien ; d'autre part, que le jeton assure que la réduction ne se produit qu'à un point où le terme est clos (le jeton ne descend pas sous les λ). Cela ne peut pas être vrai dans un évaluateur pleinement paresseux, puisque l'on veut justement pouvoir réduire des sous-termes potentiellement ouverts. La difficulté vient du fait que nous utilisons un cadre distribué : la copie est effectuée par des étapes locales, et deux processus de copie différents peuvent être intercalés, et il peut ne pas être évident de savoir quand s'arrêter.

Ce problème disparaît dans un cadre non-distribué ; par exemple, dans [SW05], un algorithme pour la réduction pleinement paresseuse est décrit dans un style impératif. L'astuce est d'effectuer deux passages pour copier : dans la première passe, les nœuds sont copiés et marqués comme tels, et un nœud déjà marqué n'est jamais copié ; dans la deuxième passe les marques sont effacées. Ce n'est clairement pas applicable ici.

Dans notre cadre le problème revient à celui de décider quand deux agents δ se rencontrent, s'ils doivent se copier mutuellement ou s'annihiler. Ce problème est bien connu dans le contexte de la réduction optimale [Lam90], où il est connu comme le problème d'implanter (efficacement) les *boîtes* de la logique linéaire. Avant même de commencer à travailler sur une solution, nous pouvons déjà dire qu'une telle solution ne sera pas complètement satisfaisante. Jusqu'ici, le jeton d'évaluation a deux avantages : il restreint l'évaluation aux rédex nécessaires et permet de contrôler la copie d'une manière très simple. Dans un cadre pleinement paresseux, ce

dernier point ne sera pas vrai : nous avons atteint une limite de l'approche à passage de jeton.

Maintenant il y a plusieurs solutions connues au problème de donner la bonne portée aux agents de copie. Le formalisme RINO, développé par Lang [Lan98], est le plus facile à adopter dans notre cas, parce qu'il n'introduit pas d'agents supplémentaires (il ajoute plutôt de la structure aux agents déjà existants) et son évaluateur est déjà (essentiellement) pleinement paresseux, bien qu'il ne prétende rien de tel. Nous utiliserons donc librement sa technologie et ses résultats. Les détails complets et les preuves peuvent être trouvés dans [Lan98].

La technologie de Lang repose sur des étiquettes données à certains agents, donc nous commençons par un peu de terminologie. Nous supposons donné un ensemble infini d'*étiquettes atomiques* (notées $x, x_1, x_2, y, \ldots$). Nous définissons alors une *étiquette* (notée $\sigma, \rho, \ldots$) comme une suite ordonnée d'étiquettes atomiques, et nous notons ϵ l'*étiquette vide*. La concaténation est notée par l'absence de symbole quand il n'y a pas de risque de confusion. Nous définissons la *longueur* d'une étiquette σ, notée $|\sigma|$ comme le nombre d'étiquettes atomiques de σ. De plus, nous notons $\rho \leq \sigma$ si ρ est un préfixe de σ.

Notre point de départ est le codage de la Section V.5.4. Les agents sont modifiés de la façon suivante.

- Les agents λ sont annotés par une étiquette atomique qui est unique dans le réseau considéré : le réseau de départ a des étiquettes atomiques uniques sur ses agents λ, et quand un agent λ est copié, les deux copies reçoivent des étiquettes atomiques fraîches (c'est-à-dire qui ne sont jamais apparues dans ce réseau), voir la Figure V.1.
- Les agents précédents c et δ sont maintenant fusionnés en un seul type d'agent c_σ, annoté par une étiquette non-atomique. Toutes les étiquettes sont initialement vides (c'est-à-dire égales à ϵ). Les agents c précédents correspondent à une étiquette vide, alors que les agents δ correspondent à c_σ avec $|\sigma| \geq 1$. C'est une conséquence de la règle $c_\sigma \bowtie \lambda$, voir la Figure V.1.
- Les agents s sont aussi annotés par une étiquette. L'agent s précédent était une version inhibée de l'agent c, utilisée pour partager des sous-termes clos. Maintenant s_σ est aussi une version inhibée de δ (dans le cas $|\sigma| \geq 1$) et peut partager des sous-termes potentiellement ouverts.
- L'agent δ avait deux fonctions duales : partager un sous-terme (en tant que *fan in*, dans la littérature) et signaler la fin du partage d'un contexte (*fan out*). Maintenant, ces deux usages sont différenciés syntaxiquement, sous la forme de c_σ pour le premier, et de $\overline{c}_\sigma$ pour le second. C'est nécessaire à cause de l'asymétrie possible de la règle $c_\sigma \bowtie \overline{c}_\rho$ (voir la Figure V.1), par opposition avec la Section V.5.5.
- Il n'y a plus d'agent s''. En effet, le comportement du nouvel agent c_σ est d'effectuer seulement une étape de copie, puis de redevenir un agent s_σ et de partager le sous-terme jusqu'à ce que le jeton rende une copie ultérieure nécessaire, de façon similaire au s'' de la Section V.5.5.
- Finalement, il y a un nouveau type de jeton, noté $\Uparrow$, utilisé quand le sous-réseau évalué sous ce jeton est de la forme : une pile d'applications commençant par un agent $\overline{c}_\sigma$. Son rôle est de remonter et de trouver un agent s'_σ (il y en a nécessairement un), contrairement au jeton standard $\Uparrow$ dont le rôle est de trouver une application. Nous utilisons la même notation que dans la Section V.4 car il y a des similitudes entre ces deux agents.

Notre interprète pleinement paresseux consiste en les règles de la Section V.5.5 (en ajoutant

des étiquettes où nécessaire), avec les modifications et additions décrites à la Figure V.1, et commentées ci-dessous.

- Un agent $\Lsh$ est créé quand le jeton atteint un $\overline{c}_\sigma$ et remonte la pile d'applications jusqu'à trouver un agent s'_σ (il y en a forcément un).
- Quand le jeton $\Lsh$ trouve un agent s'_σ, nous initions la copie avec un agent c_σ et recommençons l'évaluation à partir de la position originale (gauche ou droite, comme indiqué par l'agent).
- Quand une application est copiée, nous savons que le sous-terme de gauche sera nécessaire, donc copié. Nous préférons donc générer directement un c_σ au lieu d'un s_σ, au titre d'une optimisation facile.
- Quand un agent c_σ copie un agent λ étiqueté x, les deux copies du λ sont annotées par des étiquettes fraîches (c'est-à-dire qui n'ont pas été utilisées auparavant dans le réseau), alors que les nouveaux agents $\overline{c}$ et s sont étiqueté par σx (x suffixé à σ), voir la Figure V.1.
- Quand deux agents c_σ et $\overline{c}_\rho$ se font face, leurs étiquettes sont comparées. Si l'étiquette de l'agent de copie fermant $\overline{c}_\rho$ est un préfixe de l'autre, alors ils proviennent du même λ et doivent s'annihiler (la copie est terminée). Sinon, ils ne se correspondent pas, et doivent se copier mutuellement. Dans ce cas, la règle est asymétrique, de sorte que nous devons introduire des agents différents pour c_σ et $\overline{c}_\sigma$.

Pour comprendre le rôle des étiquettes sur un exemple, le lecteur est invité à réduire le terme $(\lambda z.z\ z)\,(\lambda x.(\lambda z.z\ z)\,(\lambda y.x\ y)))$: le corps de l'abstraction la plus interne est copié deux fois, et les étiquettes assurent que les agents de copie s'apparient correctement.

En bref, l'interprète que nous proposons est une reformulation de celui de [Lan98]. Nous ne voulons donc pas passer plus de temps sur ses propriétés. La correction de l'étiquetage et la simulation du λ-calcul sont données dans [Lan98] et sont facilement adaptées. Nous pouvons cependant noter un point commun intéressant entre les systèmes développés par Lang, Lippi et nous-mêmes : il y a deux agents distincts pour l'application, un qui est utilisé pour la β-réduction et un qui est utilisé pour la duplication, et ces agents peuvent être convertis d'un type à l'autre par certaines règles d'interaction avec d'autres agents.

L'implantation de Lang suit donc essentiellement la stratégie pleinement paresseuse, mais ce n'est pas exactement vrai puisque l'ordre normal n'est pas imposé (c'est le rôle principal du jeton ici), par conséquent certains calculs inutiles peuvent être effectués. D'autre part, il utilise des réseaux d'interaction standard (pas de ports principaux multiples), ce qui n'est pas possible avec l'approche à passage de jeton, parce que le jeton peut apparaître de n'importe quel côté d'un agent de partage. Nous pouvons brièvement énoncer la propriété principale de notre présentation :

PROPOSITION V.6.1

Le système de la Figure V.1 implante la réduction pleinement paresseuse.

Preuve. La réduction pleinement paresseuse correspond au partage des *expressions libres maximales* [PJ87]. Or, notre implantation partage *tout*, nœud par nœud, jusqu'à ce que le jeton indique que l'évaluation est requise. Notre implantation est donc *trivialement* pleinement paresseuse. □

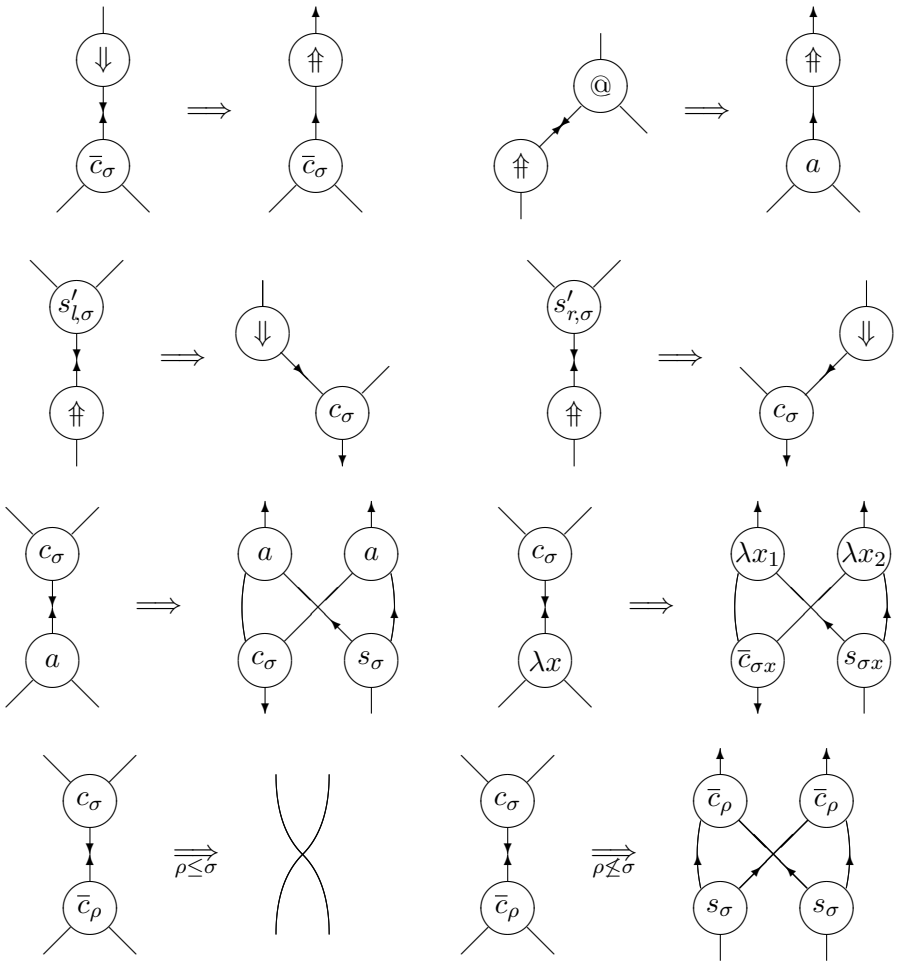


FIG. V.1 – Règles additionnelles pour la réduction pleinement paresseuse

Il serait probablement difficile de formuler précisément l'observation que l'interprète de Lang est (presque) pleinement paresseux sans notre cadre avec jeton ; d'ailleurs, Lang ne prétend rien de tel (sauf d'être expérimentalement efficace). Une fois que cette observation a été faite, il est juste de dire que l'implantation de Lang est probablement plus efficace que la nôtre, en particulier dans un vrai cadre distribué. Le jeton d'évaluation rend en effet la stratégie plus explicite, mais il peut également avoir un coût, selon la façon dont les réseaux à passage de jeton sont implantés. L'implantation de Lang n'a pas un tel surcoût (potentiel), et de plus, il emploie des réseaux d'interaction standard (pas de ports principaux multiples). Notre présentation peut être vue comme une formalisation de ce que serait probablement une implantation réelle de l'interprète de Lang dans un langage séquentiel. Il est également juste de dire que notre présentation vise la simplicité et ne considère pas de possibles optimisations. Nous pensons en effet que notre cadre pourrait être un bon candidat pour raisonner sur de telles optimisations et pour les prouver formellement.

V.7 Conclusion

Nous avons présenté une approche simple et extensible pour exprimer les stratégies les plus populaires du λ-calcul dans un cadre de récriture de réseau. La clé de ces codages est de représenter le flot de contrôle explicitement, par un agent d'interaction. L'appel par nom et l'appel par valeur peuvent s'exprimer dans les réseaux d'interaction de Lafont, alors que l'appel par nécessité requiert un cadre un peu plus général, mais aucune propriété importante n'est perdue.

L'approche est si simple que notre cadre est en fait une bonne alternative aux termes, avec l'avantage, d'une part de donner un statut et un coût explicites aux opérations de substitution et de copie, d'autre part de quotienter des différences artificielles dans les représentations sous forme de termes (α-conversion) et de couples de terme et environnement.

Pour la réduction pleinement paresseuse, le cadre est moins simple et emprunte la technologie développée par Lang, en réintroduisant certains mécanismes pour apparier les agents de copie, ce qui rappelle les boîtes de la logique linéaire. Cependant, ce n'est pas une faiblesse, mais simplement le prix à payer pour obtenir une notion de partage aussi fine. Cela montre sans doute aussi une limite de l'approche, en particulier dans le cadre d'une application à la réduction optimale. Une notion de boîte serait certainement nécessaire, mais cela peut tout de même être bénéfique de contrôler plus finement le flot d'évaluation.

Des questions et remarques similaires s'appliquent aussi à la réduction close du Chapitre II, comme on en discutera au chapitre suivant.

Enfin, nos réseaux font partie d'une certaine classe de *réseaux à passage de jeton* qui n'est pas étudiée en détails ici, mais qui semble très facile à implanter sur une machine séquentielle, sans le coût habituel pour rechercher les rédex.

Conclusions et perspectives

Contributions

Nous nous sommes donc posé la question de l'efficacité dans le λ-calcul et les langages fonctionnels, qui est un problème intrinsèquement difficile. Plus précisément, nous avons abordé les aspects cruciaux de la stratégie d'évaluation et d'un modèle d'implantation adapté. Nous avons essayé d'adopter une approche originale, mêlant aspects théoriques et pratiques.

Nous avons d'abord développé un λ-calcul avec substitutions explicites qui évite les problèmes habituels liés à la substitution et à l'α-conversion, dans lequel on peut définir les stratégies usuelles, mais aussi la stratégie close, qui autorise certaines réductions sous les abstractions, et qui est donc moins faible que les stratégies usuelles. De plus, elle évite un certain nombre des défauts habituels des stratégies fortes, en particulier celui du renommage de variables (α-conversion). Nous avons prouvé toutes les propriétés théoriques désirables pour appuyer la définition de cette stratégie.

Nous avons aussi développé une représentation des variables dans le cadre de la récriture d'ordre supérieur, que nous avons utilisée pour définir un modèle d'implantation efficace pour la réduction close. Ce calcul avec chaînes directrices est lui aussi appuyé de toutes les propriétés désirables. Des machines abstraites ont été implantées et les résultats expérimentaux confirment l'intérêt de cette stratégie pour l'implantation efficace des langages fonctionnels.

D'autre part, nous avons proposé un cadre simple permettant de faire le lien entre deux modèles de calculs traditionnellement utilisés pour l'implantation des langages fonctionnels, les machines abstraites et les réseaux d'interaction, mais radicalement opposés. Nous avons illustré ce cadre sur les stratégies usuelles, rapprochant ainsi deux modèles utilisés pour l'implantation de stratégies efficaces.

Perspectives

Réseaux

Nous avons introduit une classe de réseaux, dits à passage de jeton, plus souples que les réseaux d'interaction de Lafont, puisque certains agents peuvent avoir plusieurs ports principaux, mais avec des contraintes fortes qui permettent de retrouver certaines propriétés. Nous avons décrit une autre généralisation comparable à celle-ci dans [SM05], au sens où nous avons également donné une définition plus souple que les réseaux d'interaction, mais sans perdre de propriétés.

Cette approche nous semble intéressante, car les réseaux d'interaction ont des propriétés désirables, mais sont souvent trop contraignants. D'autres généralisations intéressantes sont sans doute possibles. De plus, les réseaux à passage de jeton semblent être plus faciles à implanter que les réseaux d'interaction standard, et il serait sans doute intéressant d'étudier davantage cet aspect.

D'autres stratégies dans les réseaux

Nos travaux, sur la réduction close et les chaînes directrices d'une part, sur les stratégies dans les réseaux à passage de jeton d'autre part, sont relativement indépendants, et il est naturel de vouloir les connecter.

Nous avons vu que, dans le cadre des réseaux à passage de jeton, il y a une frontière entre les stratégies qui ne partagent pas plus que des valeurs (appels par nom, valeur et nécessité) et la stratégie pleinement paresseuse. Cette dernière bénéficie d'un partage plus fin, mais ce partage a un prix : le codage est plus complexe.

On aurait naturellement envie d'appliquer la méthodologie du Chapitre V à la réduction close du Chapitre II. Cela permettrait en particulier de bénéficier de l'efficacité de la réduction close tout en préservant la normalisation (autrement dit, si un terme a une forme normale de tête faible, alors le système de récriture de réseau terminera et rendra cette valeur), un peu comme l'appel par nécessité par rapport à l'appel par valeur. Nous identifions deux problèmes principaux pour réaliser cela.

D'abord, comme dans le cas de la réduction pleinement paresseuse, il semble nécessaire de partager plus que des valeurs, de façon à partager certaines réductions sous des abstractions. On peut bien sûr utiliser la technologie décrite dans le Chapitre V, mais ce n'est pas forcément la meilleure solution dans ce cas, comme nous allons le voir.

Le second problème est que la réduction close repose sur certaines conditions de clôture de certains sous-termes. Le but des Chapitres III et IV était d'internaliser ces conditions sous la forme d'un système de récriture de termes non conditionnel. Il faudrait donc ici internaliser ces conditions dans un système de récriture de réseaux. La difficulté est de savoir, au niveau de la racine d'un sous-terme, s'il est clos ou non, alors que cela dépend de substitutions éventuelles au niveau des feuilles de ce sous-terme. Il y a donc un problème de localité, comme au Chapitre II, mais la solution du Chapitre IV n'est pas applicable, car la substitution n'a pas du tout la même géométrie dans les termes et les réseaux.

En réalité, une solution est proposée par Mackie [Mac04], qui décrit un évaluateur basé sur les réseaux d'interaction en utilisant certaines idées issues de la réduction close. Le choix de [Mac04] est d'introduire un lien entre les agents λ et les ports correspondant aux variables libres, de façon à faire remonter l'information quand le terme devient clos. Mais cela correspond aussi à un codage de boîte, puisque la portée des abstractions est clairement délimitée, et cela peut en particulier aussi constituer une solution au premier problème.

On pourrait donc essayer d'ajouter un jeton d'évaluation à ce codage. Cela semble possible, même si les résultats de simulation risquent d'être moins forts que ceux du Chapitre V. Cependant, cela pourrait aider à résoudre certains problèmes. Par exemple, en imposant un ordre normal sur les réductions, la stratégie pourrait effectivement devenir normalisante. Ainsi, grâce aux agents de partage et au jeton d'évaluation, on n'effectuerait que des réductions nécessaires.

Le même type de questions peut s'appliquer à la réduction optimale. Introduire un jeton permettrait très certainement de formaliser le fait que l'on veut réduire en ordre normal. Cependant, une notion de boîte serait certainement encore nécessaire. Néanmoins, en contrôlant plus finement le flot d'évaluation, on se place dans un cadre plus simple, plus prévisible, et cela pourrait très bien avoir des effets bénéfiques sur le type de boîte nécessaire.

Réduction close et optimalité(s)

Dans cette thèse, on a beaucoup parlé de partage. Il y a deux types de partage : le partage de données, et le partage de calculs. Le partage de données offre la *possibilité* de partager des calculs. Expérimentalement, nous constatons que la réduction close est meilleure que la réduction pleinement paresseuse, donc *a priori* qu'elle effectue un meilleur partage de calculs. Pourtant, ces deux stratégies bénéficient du même type de partage de données (par opposition à la réduction optimale, par exemple). De plus, c'est la stratégie pleinement paresseuse qui fait le choix de tout le temps partager les calculs, alors que la réduction close *interdit* certains calculs. On vérifie donc que le mieux est l'ennemi du bien : il peut être souhaitable de ne pas effectuer certaines réductions.

Mais nous pouvons surtout généraliser l'approche : nous avons un cadre de partage de données dans lequel vivent deux stratégies particulières. Ces stratégies font le choix, quand elles rencontrent (c'est-à-dire : quand le jeton d'évaluation rencontre) un rédex, de le réduire ou pas. La stratégie pleinement paresseuse fait toujours ce choix, la stratégie close ne le fait que s'il vérifie une certaine condition de clôture.

Nous pouvons donc très bien utiliser l'évaluateur pour la réduction pleinement paresseuse du Chapitre V, et appeler un oracle à chaque fois que nous rencontrons un rédex, pour décider s'il doit être réduit ou non. Nous avons réalisé des expérimentations en utilisant un oracle aléatoire, et nous avons procédé à un grand nombre d'exécutions. Pour un terme donné, parmi toutes ces exécutions, un certain nombre d'entre elles réalisent un nombre minimal de β-réductions. Plus précisément, pour un terme donné, il existe un nombre n, tel qu'on puisse obtenir un nombre arbitraire d'exécutions réalisant n β-réductions, et aucune qui réalise $n-1$ β-réductions. Ce nombre correspond donc a une notion d'optimalité, mais pas au sens de Lévy, puisque le partage de données autorisé est moins fort. Nous dirons donc qu'il s'agit d'une notion d'*optimalité faible*. De façon étonnante, ce nombre est très proche du nombre de β-réductions effectuées par la réduction close, expérimentalement, bien sûr.

Le prolongement de ce travail qui nous semble le plus prometteur est donc de formaliser cette notion d'optimalité faible, qui porte de meilleurs espoirs de correspondre à une notion réaliste de coût d'implantation, et d'étudier plus précisément les liens avec la réduction close.

Index

$\star$, 29
$\Box$, 90
$(\cdot)^*$, 36
$[\![\cdot]\!]$, 35, 71, 92
$(\!|\cdot|\!)$, 72, 93
$|\cdot|_+$, 90
$|\cdot|_l$, 90
$|\cdot|_r$, 90
$|\cdot|$, 90
α-conversion, 13
β-réduction, 14
ϵ, 90
$\lambda\mathsf{I}$-calcul, 12
λ-calcul, 11
  avec indices de de Bruijn, 14
  avec substitutions explicites, 16, 29
  simplement typé, 17
λ_c, 111
λ_l, 109
λ_o, 95
λ_{c}-terme, 34
λ_{ca}, 28, 37
λ_{caf}, 28, 55
λ_{cf}, 28, 43
λ_{cl}, 28, 54
$\lambda_{\mathsf{cf}}^{\mathsf{pcf}}$, 61
Λ_{c}, 34
Λ_{ca}, 37
Λ_{cf}, 44
σ, 37, 44
ξ, 21

abstracteur, 19
adéquation, 61, 62
appel
  par nécessité, 22, 144
  par nom, 22, 53, 115, 131
  par valeur, 22, 53, 115, 139
arité, 9
ARS, 8

calcul
  clos, 111
  local ouvert, 109
  ouvert, 95
chaîne directrice, 71, 88, 90
codomaine, 9
combinateurs, 15, 89
commutation, 8, 42, 50
compilation, 35, 62, 71, 92
complétion, 42
composition, 33
confluence, 8, 51, 105, 114
  forte, 8, 24, 130, 154
  locale, 8, 43, 49, 100
constante, 9, 29
contexte, 12
convention de Barendregt, 13, 30
copie, 27, 91, 137
CRS, 19

décompilation, 36, 72, 93
dérivation minimale, 47, 106
directeur, 71, 89
distance, 41, 99
domaine, 9, 10

effacement, 27, 91, 137
ESCRS, 20, 80

feuille, 9

forme normale, 8, 123
    de tête faible, 14

lemme de Hindley-Rosen, 8
lemme de Newman, 8
logique combinatoire, 15, 54

métaterme, 19
métavariable, 19

normalisation
    faible, 8
    forte, 8, 48, 59, 107, 115

optimalité, 22, 130

paire critique, 11
PCF, 18, 61
position, 9, 71
    externe, 106
préfixe, 9
préterme, 90
programme, 59
PSN, 48, 107, 115

réduction close, 27
réduction du sujet, 59, 62, 119
réseau, 24
    à passage de jeton, 130
    d'interaction, 23, 130
racine, 9
remplacement, 10

sémantique à passage de jeton, 133, 139, 148
signature, 9
simulation, 102
sous-terme, 10
stratégie, 21
    close, 63, 121
    du λ-calcul, 21
    faible, 21
    forte, 21
substitution, 10, 13, 14, 20, 74, 77
    explicite, 16, 20, 69
support, 10
système de récriture, 10
    combinatoire, 19
système de réduction abstrait, 8

terme, 9, 11, 19, 29, 90
    clos, 9, 12, 19
    enrichi, 132, 147
    fermé, 9, 12, 19
    ouvert, 12, 19, 141
    pur, 35, 99
TRS, 10
type, 58, 118

valeur, 14
variable, 9, 11, 19
    liée, 12, 19
    libre, 12, 19, 70

Bibliographie

[Abr93] Samson Abramsky. Computational interpretations of linear logic. *Theoretical Computer Science*, 111(1–2):3–57, 1993. 27, 32

[ACCL91] Martín Abadi, Luca Cardelli, Pierre-Louis Curien et Jean-Jacques Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4):375–416, octobre 1991. 17, 37, 68, 69, 86, 87

[ACM00] Andrea Asperti, Paolo Coppola et Simone Martini. (Optimal) duplication is not elementary recursive. *In Proceedings of the 27th ACM Symposium on Principles of Programming Languages (POPL'00)*, pages 96–107, 2000. 4

[AF97] Zena M. Ariola et Matthias Felleisen. The call-by-need lambda calculus. *Journal of Functional Programming*, 7(3):265–301, mai 1997. 143, 148

[AFM+95] Zena M. Ariola, Matthias Felleisen, John Maraist, Martin Odersky et Philip Wadler. The call-by-need lambda calculus. *In Proceedings of the 22nd ACM Symposium on Principles of Programming Languages (POPL'95)*, pages 233–246, San Francisco, California, janvier 1995. ACM Press. 17, 28, 87, 143, 148

[AG98] Andrea Asperti et Stefano Guerrini. *The Optimal Implementation of Functional Programming Languages*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1998. 4, 23, 87, 88, 155

[AGN96] Andrea Asperti, Cecilia Giovannetti et Andrea Naletto. The Bologna optimal higher-order machine. *Journal of Functional Programming*, 6(6):763–810, novembre 1996. 4, 23, 87, 88, 125, 126, 130

[Ale99] Vladimir Alexiev. *Non-deterministic Interaction Nets*. Thèse de doctorat, University of Alberta, 1999. 24, 143

[Bar84] Henk P. Barendregt. *The Lambda Calculus : Its Syntax and Semantics*, volume 103 de *Studies in Logic and the Foundations of Mathematics*. North-Holland Publishing Company, second, revised édition, 1984. 11, 15, 16, 19, 86, 103, 104

[Bar92] Henk P. Barendregt. Lambda calculi with types. *In* S. Abramsky, D. Gabbay et T. S. E. Maibaum, éditeurs. *Handbook of Logic in Computer Science*, volume 2, chapitre 2, pages 117–309. Oxford University Press, 1992. 58

[Baw86] Alan Bawden. Connection graphs. *In* Richard P. Gabriel, éditeur. *Proceedings of the ACM Conference on LISP and Functional Programming*, pages 258–265. ACM Press, 1986. 24

[BBLRD96] Zine-El-Abidine Benaissa, Daniel Briaud, Pierre Lescanne et Jocelyne Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5):699–722, 1996. 16, 87

[BG99] Roel Bloo et Herman Geuvers. Explicit substitution : on the edge of strong normalization. *Theoretical Computer Science*, 211(1):375–395, 1999. 86

[BLM05] Tomasz Blanc, Jean-Jacques Lévy et Luc Maranget. Sharing in the weak lambda-calculus. *In Processes, Terms and Cycles*, volume 3838 de *Lecture Notes in Computer Science*, pages 70–87. Springer, 2005. 4, 54

[BR96] Roel Bloo et Kristoffer H. Rose. Combinatory reduction systems with explicit substitution that preserve strong normalisation. *In Proceedings of Rewriting Techniques and Applications (RTA'96)*, volume 1103 de *Lecture Notes in Computer Science*, pages 169–183, 1996. 20

[BRL96] Zine-El-Abidine Benaissa, Kristoffer H. Rose et Pierre Lescanne. Modeling sharing and recursion for weak reduction strategies using explicit substitution. *In* H. Kuchen et D. Swierstra, éditeurs. *8th PLILP—Symposium on Programming Language Implementation and Logic Programming*, pages 393–407, Aachen, Germany, 1996. 88

[Bru72] N. G. de Bruijn. Lambda calculus notation with nameless dummies. *Indagationes Mathematicae*, 34:381–392, 1972. 81, 86

[Bru78] N. G. de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. Rapport technique T.H.-Report 78-WSK-03, Department of Mathematics, Eindhoven University of Technology, 1978. 16, 86, 87

[CCM87] Guy Cousineau, Pierre-Louis Curien et Michel Mauny. The categorical abstract machine. *Science of Computer Programming*, 8:173–202, 1987. 16

[ÇH98] Naim Çağman et J. Roger Hindley. Combinatory weak reduction in lambda calculus. *Theoretical Computer Science*, 198(1–2):239–249, 1998. 15, 16, 27, 28, 31, 54, 86

[CHL96] Pierre-Louis Curien, Thérèse Hardin et Jean-Jacques Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2):362–397, 1996. 87

[Chu41] Alonzo Church. *The Calculi of Lambda-Conversion.* Numéro 6 de Annals of Mathematical Studies. Princeton University Press, 1941. 2, 11

[CJ95] Hubert Comon et Jean-Pierre Jouannaud. Les termes en logique et en programmation. Notes du cours de DEA Sémantique, preuves et programmation, 1995. 9

[CK01] Horatiu Cirstea et Claude Kirchner. The rewriting calculus. *Logic Journal of the Interest Group in Pure and Applied Logics*, 9(3):427–498, mai 2001. 6

[Cré90] Pierre Crégut. An abstract machine for lambda-terms normalization. *In Lisp and Functional Programming 1990*, pages 333–340. ACM Press, 1990. 81, 92, 123, 124, 148

[Cur91] Pierre-Louis CURIEN. An abstract framework for environment machines. *Theoretical Computer Science*, 82:389–402, 1991. 16

[DG01] R. DAVID et B. GUILLAUME. A λ-calculus with explicit weakening and explicit substitution. *Mathematical Structure in Computer Science*, 11(1):169–206, 2001. 27, 69, 86, 105

[DHK01] Gilles DOWEK, Therese HARDIN et Claude KIRCHNER. HOL-$\lambda\sigma$: an intentional first-order expression of higher-order logic. *Mathematical Structures in Computer Science*, 11(1):21–45, février 2001. 26

[DJ89] Nachum DERSHOWITZ et Jean-Pierre JOUANNAUD. Rewrite Systems. *In* J. van LEEUWEN, éditeur. *Handbook of Theoretical Computer Science : Formal Models and Semantics*, volume B. North-Holland, 1989. 9, 72

[EPJ03] Robert ENNALS et Simon PEYTON JONES. Optimistic evaluation : an adaptive evaluation strategy for non-strict programs. *In* Cindy NORRIS et Jr. JAMES B. FENWICK, éditeurs. *Proceedings of the Eighth International Conference on Functional Programming (ICFP-03)*, volume 38, 9 de *ACM SIGPLAN Notices*, pages 287–298. ACM Press, 2003. 126

[Fie90] John FIELD. On laziness and optimality in lambda interpreters : Tools for specification and analysis. *In Conference Record of the 17th Annual ACM Symposium on Principles of Programming Languages (POPL '90)*, pages 1–15, San Francisco, CA, USA, janvier 1990. ACM Press. 3, 4, 23, 129

[FM99] Maribel FERNÁNDEZ et Ian MACKIE. Closed reductions in the λ-calculus. *In* J. FLUM et M. RODRÍGUEZ-ARTALEJO, éditeurs. *Proceedings of Computer Science Logic (CSL'99)*, numéro 1683 de Lecture Notes in Computer Sciences, pages 220–234. Springer-Verlag, septembre 1999. 32, 86

[FM02] Maribel FERNÁNDEZ et Ian MACKIE. Call-by-value lambda-graph rewriting without rewriting. *In Proceedings of the International Conference on Graph Transformations (ICGT'02)*, volume 2505 de *Lecture Notes in Computer Science*, Barcelona, 2002. Springer-Verlag. 131

[FMS05a] Maribel FERNÁNDEZ, Ian MACKIE et François-Régis SINOT. Closed reduction : Explicit substitutions without α-conversion. *Mathematical Structures in Computer Science*, 15(2):343–381, 2005. 5, 32

[FMS05b] Maribel FERNÁNDEZ, Ian MACKIE et François-Régis SINOT. Interaction nets vs. the ρ-calculus : Introducing bigraphical nets. *Electronic Notes in Theoretical Computer Science*, 2005. 6

[FMS05c] Maribel FERNÁNDEZ, Ian MACKIE et François-Régis SINOT. Lambda-calculus with director strings. *Journal of Applicable Algebra in Engineering, Communication and Computing*, 15(6):393–437, avril 2005. 6

[GAL92] Georges GONTHIER, Martín ABADI et Jean-Jacques LÉVY. The geometry of optimal lambda reduction. *In Proceedings of the 19th ACM Symposium on Principles of Programming Languages (POPL'92)*, pages 15–26. ACM Press, janvier 1992. 4, 23, 87, 88, 130

[Gir87] Jean-Yves GIRARD. Linear Logic. *Theoretical Computer Science*, 50(1):1–102, 1987. 32, 129, 130

[Gir89] Jean-Yves GIRARD. Geometry of interaction I : Interpretation of System F. *In* C. BONOTTO, R. FERRO, S. VALENTINI et A. ZANARDO, éditeurs. *Logic Colloquium '88*, volume 127 de *Studies in Logic and the Foundations of Mathematics*, pages 221–260, Amsterdam, 1989. North-Holland. 28, 130

[GL02] Benjamin GRÉGOIRE et Xavier LEROY. A compiled implementation of strong reduction. *In Proceedings of ICFP'02, Pittsburgh, Pennsylvania, USA*, 2002. 124

[Hin64] J. Roger HINDLEY. *The Church-Rosser Property and a Result in Combinatory Logic.* Thèse de doctorat, University of Newcastle-upon-Tyne, 1964. 8

[Hin77] J. Roger HINDLEY. Combinatory reductions and lambda reductions compared. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 23:169–180, 1977. 16, 54

[HMP98] Thérèse HARDIN, Luc MARANGET et Bruno PAGANO. Functional runtime systems within the $\lambda\sigma$-calculus. *Journal of Functional Programming*, 8(2):131–176, mars 1998. 17, 28, 87

[HO93] J. Martin E. HYLAND et C.-H. Luke ONG. Modified realizability toposes and strong normalization proofs. *In* J. F. GROOTE et M. BEZEM, éditeurs. *Typed Lambda Calculi and Applications*, volume 664 de *Lecture Notes in Computer Science*, pages 179–194. Springer-Verlag, 1993. 54

[How70] W. A. HOWARD. Assignment of ordinals to terms for primitive recursive functionals of finite type. *In Intuitionism and Proof Theory*, pages 443–458, 1970. 16, 54

[JM03] Ole H. JENSEN et Robin MILNER. Bigraphs and mobile processes. Rapport technique 570, Computer Laboratory, University of Cambridge, 2003. 6

[Kat90] Vinod KATHAIL. *Optimal interpreters for lambda-calculus based functional languages.* Thèse de doctorat, Massachusetts Institute of Technology, 1990. 23

[KL05] Delia KESNER et Stéphane LENGRAND. Extending the explicit substitution paradigm. *In* Jürgen GIESL, éditeur. *16th International Conference on Rewriting Techniques and Applications*, volume 3467 de *Lecture Notes in Computer Science*, pages 407–422. Springer-Verlag, avril 2005. 32

[Klo80] Jan-Willem KLOP. *Combinatory Reduction Systems.* Thèse de doctorat, Centre for Mathematics and Computer Science, Amsterdam, 1980. 19, 41

[KOR93] Jan-Willem KLOP, Vincent van OOSTROM et Femke van RAAMSDONK. Combinatory reduction systems : introduction and survey. *Theoretical Computer Science*, 121(1-2):279–308, décembre 1993. 19, 20, 79

[KS88] J. Richard KENNAWAY et M. Ronan SLEEP. Director strings as combinators. *ACM Transactions on Programming Languages and Systems*, 10(4):602–626, 1988. 68, 86, 87, 88, 89, 99

[Laf90] Yves LAFONT. Interaction nets. *In Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 95–108. ACM Press, janvier 1990. 3, 23, 24, 130, 153

[Laf97] Yves LAFONT. Interaction combinators. *Information and Computation*, 137(1): 69–101, août 1997. 134, 138

[Lam90] John LAMPING. An algorithm for optimal lambda calculus reduction. *In Proceedings of the 17th ACM Symposium on Principles of Programming Languages (POPL'90)*, pages 16–30. ACM Press, janvier 1990. 3, 23, 87, 88, 129, 130, 158

[Lan98] Frédéric LANG. *Modèles de la β-réduction pour les implantations.* Thèse de doctorat, École Normale Supérieure de Lyon, 1998. 17, 87, 88, 143, 159, 160

[Lau93] John LAUNCHBURY. A natural semantics for lazy evaluation. *In Conference Record of the Twentieth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 144–154, Charleston, South Carolina, janvier 1993. 143, 144, 145, 146

[Les94] Pierre LESCANNE. From $\lambda\sigma$ to $\lambda\upsilon$ a journey through calculi of explicit substitutions. *In Proceedings of the 21st ACM Symposium on Principles of Programming Languages (POPL'94).* ACM Press, 1994. 87

[Lév80] Jean-Jacques LÉVY. Optimal reductions in the lambda-calculus. *In* J. P. SELDIN et J. R. HINDLEY, éditeurs. *To H. B. Curry : Essays in Combinatory Logic, Lambda Calculus and Formalism*, pages 159–191. Academic Press, 1980. 3, 22, 88

[Lip02a] Sylvain LIPPI. Encoding left reduction in the lambda-calculus with interaction nets. *Mathematical Structures in Computer Science*, 12(6), décembre 2002. 130, 140

[Lip02b] Sylvain LIPPI. *Théorie et pratique des réseaux d'interaction.* Thèse de doctorat, Université de la Méditerranée, juin 2002. 130, 134, 138

[LM96] Julia L. LAWALL et Harry G. MAIRSON. Optimality and inefficiency : What isn't a cost model of the lambda calculus ? *In International Conference on Functional Programming*, pages 92–101, 1996. 4, 88

[LRD95] Pierre LESCANNE et Jocelyne ROUYER-DEGLI. The calculus of explicit substitutions $\lambda\upsilon$. Rapport technique RR-2222, INRIA, 1995. 47, 69, 86, 87, 105

[Mac94] Ian MACKIE. *The Geometry of Implementation.* Thèse de doctorat, Department of Computing, Imperial College of Science, Technology and Medicine, septembre 1994. 130

[Mac95] Ian MACKIE. The geometry of interaction machine. *In Proceedings of the 22nd Symposium on Principles of Programming Languages (POPL'95)*, pages 198–208, San Francisco, CA, USA, 1995. ACM Press. 131

[Mac98] Ian MACKIE. YALE : Yet another lambda evaluator based on interaction nets. *In Proceedings of the 3rd International Conference on Functional Programming (ICFP'98)*, pages 117–128. ACM Press, 1998. 130

[Mac04] Ian MACKIE. Efficient λ-evaluation with interaction nets. *In* V. van OOSTROM, éditeur. *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA'04)*, volume 3091 de *Lecture Notes in Computer Science*, pages 155–169. Springer-Verlag, juin 2004. 130, 164

[Mar91] Luc MARANGET. Optimal derivations in orthogonal term rerwiting systems and in weak lambda calculi. *In Proc. of the 1991 conference on Principles of Programming Languages*. ACM Press, 1991. 4

[Mar92] Luc MARANGET. *La stratégie paresseuse*. Thèse de doctorat, Université Paris 7, 1992. 4

[Maz05] Damiano MAZZA. Multiport interaction nets and concurrency. *In* M. ABADI et L. de ALFARO, éditeurs. *Proceedings of CONCUR'05*, volume 3653 de *Lecture Notes in Computer Sciences*, pages 21–35. Springer-Verlag, 2005. 24

[Mel95] Paul-André MELLIÈS. Typed lambda-calculi with explicit substitutions may not terminate. *In Proceedings of the Second International Conference on Typed Lambda Calculi and Applications*, volume 902 de *Lecture Notes in Computer Science*, pages 328–334. Springer-Verlag, 1995. 27, 86, 105

[MOW98] John MARAIST, Martin ODERSKY et Philip WADLER. The call-by-need lambda calculus. *Journal of Functional Programming*, 8(3):275–317, mai 1998. 143, 148

[Muñ96] César MUÑOZ. Confluence and preservation of strong normalisation in an explicit substitutions calculus (extended abstract). *In Proceedings of the Eleventh Annual IEEE Symposium on Logic in Computer Science (LICS'96)*. IEEE Computer Society Press, juillet 1996. 28, 44, 56

[Nad99] Gopalan NADATHUR. A fine-grained notation for lambda terms and its use in intensional operations. *Journal of Functional and Logic Programming*, 1999(2), mars 1999. 28, 87

[Nad02] Gopalan NADATHUR. The suspension notation for lambda terms and its use in metalanguage implementations. *In* Ruy de QUEIROZ, Luiz Carlos PEREIRA et Edward Hermann HAEUSLER, éditeurs. *Electronic Notes in Theoretical Computer Science*, volume 67. Elsevier, 2002. 17, 87

[New42] Maxwell Herman Alexander NEWMAN. On theories with a combinatorial definition of equivalence. *Annals of Mathematics*, 43(2):223–243, 1942. 8, 104

[OLZ04] Vincent van OOSTROM, Kees-Jan van de LOOIJ et Marijn ZWITSERLOOD. Lambdascope : another optimal implementation of the lambda-calculus. *In Workshop on Algebra and Logic on Programming Systems (ALPS)*, Kyoto, avril 2004. 130

[Oos01] Vinvent van OOSTROM. Net-calculus, 2001. Course notes available at `http://www.phil.uu.nl/~oostrom/typcomp/00-01/net.ps`. 32

[Pin00] Jorge Sousa PINTO. Sequential and concurrent abstract machines for interaction nets. *In* J. TIURYN, éditeur. *Proceedings of Foundations of Software Science and Computation Structures (FOSSACS)*, volume 1784 de *Lecture Notes in Computer Science*, pages 267–282. Springer-Verlag, 2000. 130, 155

[PJ87] Simon L. Peyton Jones. *The Implementation of Functional Programming Languages*. Prentice Hall International, 1987. 158, 160

[PJS89] Simon L. Peyton Jones et Jon Salkild. The spineless tagless G-machine. *In Proceedings of the Fourth International Conference on Functional Programming Languages and Computer Architecture*, pages 184–201, London, UK, septembre 1989. ACM Press. 158

[Plo75] Gordon Plotkin. Call-by-name, call-by-value, and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975. 131

[Plo77] Gordon Plotkin. LCF considered as a programming language. *Theoretical Computer Science*, 5(3):223–256, 1977. 61

[Ros73] Barry K. Rosen. Tree-manipulating systems and Church-Rosser theorems. *Journal of the ACM*, 20(1):160–187, 1973. 8

[Ros96] Kristoffer H. Rose. Explicit substitution - tutorial and survey. Lecture Series LS-96-3, BRICS, Dept. of Computer Science, University of Aarhus, Denmark, 1996. 17, 26, 28, 86

[SFM03] François-Régis Sinot, Maribel Fernández et Ian Mackie. Efficient reductions with director strings. *In* R. Nieuwenhuis, éditeur. *Proceedings of Rewriting Techniques and Applications (RTA'03)*, volume 2706 de *Lecture Notes in Computer Science*, pages 46–60. Springer-Verlag, 2003. 6

[SI96] Jill Seaman et S. Purushothaman Iyer. An operational semantics of sharing in lazy evaluation. *Science of Computer Programming*, 27(3):289–322, novembre 1996. 143

[Sin05a] François-Régis Sinot. Call-by-name and call-by-value as token-passing interaction nets. *In Proceedings of the 7th International Conference on Typed Lambda Calculi and Applications (TLCA'05)*, volume 3461 de *Lecture Notes in Computer Science*, pages 386–400. Springer-Verlag, avril 2005. 6

[Sin05b] François-Régis Sinot. Director strings revisited : A generic approach to the efficient representation of free variables in higher-order rewriting. *Journal of Logic and Computation*, 15(2):201–218, 2005. 6

[Sin06a] François-Régis Sinot. Call-by-need in token-passing nets. *Mathematical Structures in Computer Science*, 16(4), 2006. 6

[Sin06b] François-Régis Sinot. Token-passing nets : Call-by-need for free. *Electronic Notes in Theoretical Computer Science*, 135(3):129–139, 2006. 6

[SM05] François-Régis Sinot et Ian Mackie. Macros for interaction nets : A conservative extension of interaction nets. *Electronic Notes in Theoretical Computer Science*, 127(5), 2005. 6, 143, 163

[Sta79] Richard Statman. The typed λ-calculus is not elementary recursive. *Theoretical Computer Science*, 9:73–81, 1979. 4, 68

[SW05] Olin Shivers et Mitchell Wand. Bottom-up beta-reduction : uplinks and lambda-DAGs. *In Proceedings of the 14th European Symposium on Programming*

(ESOP'05), volume 3444 de *Lecture Notes in Computer Science*. Springer-Verlag, 2005. 143, 158

[Ter03] TERESE. *Term Rewriting Systems*, volume 55 de *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2003. 8, 9, 21, 72

[Wad71] Christopher P. WADSWORTH. *Semantics and Pragmatics of the Lambda-Calculus*. Thèse de doctorat, Oxford University, 1971. 143, 152, 158

[Yos94] Nobuko YOSHIDA. Optimal reduction in weak lambda-calculus with shared environments. *Journal of Computer Software*, 11(6):3–18, novembre 1994. 4, 17, 28, 87