N° d'ordre : 103
N° attribué par la bibliothèque : 98ENSL0103

ÉCOLE NORMALE SUPÉRIEURE DE LYON
Laboratoire de l'Informatique du Parallélisme

THÈSE

pour obtenir le grade de

Docteur de l'École normale supérieure de Lyon
Spécialité informatique

au titre de la formation doctorale d'informatique de Lyon

présentée et soutenue publiquement le 11 décembre 1998

par Monsieur Frédéric LANG

Titre :

Modèles de la β-réduction pour les implantations

Directeur de thèse : Monsieur Pierre LESCANNE

Devant la commission d'examen formée de :

Monsieur Roberto AMADIO, Membre
Monsieur Andrea ASPERTI, Rapporteur
Monsieur René DAVID, Membre
Madame Delia KESNER, Membre
Monsieur Pierre LESCANNE, Membre
Monsieur Michel MAUNY, Membre et Rapporteur
Madame Laurence PUEL, Membre et Rapporteur

Remerciements

Je tiens à remercier en toute sincérité, l'ensemble des personnes qui, par leur soutien scientifique, matériel, ou moral, m'ont aidé à réaliser cette thèse : Pierre, mon directeur, que je considère autant comme un ami que comme un guide scientifique ; Les membres du jury et les rapporteurs, grâce au travail desquels j'ai pu soutenir cette thèse en toute quiétude ; Zino, Kristoffer, et Luigi avec qui j'ai eu le plaisir de collaborer et qui ont enrichi ma vision de la recherche ; les membres de l'équipe Euréca, au laboratoire LORIA de Nancy où j'ai démarré ma thèse de 1995 à 1997, puis de l'équipe Plume au LIP, qui m'ont appris à prendre une part active à la communauté de recherche ; mes amis extérieurs à mes recherches, qui m'ont aidé à toujours garder un pied sur la rive ; ma famille : parents, grands-parents, sœurs, oncles et tantes, soutien affectif irremplaçable ; ma belle famille, qui m'a soutenu comme un fils, un frère ; enfin Pascale, mon épouse, et Adrien, mon fils, ma joie et mon réconfort.

Lyon, le 4 septembre 2001.

Table des matières

Table des figures

Introduction

Thèse

Le sujet principal de cette thèse est l'étude de modèles d'implantation de la β-réduction en λ-calcul. Nous y abordons des aspects théoriques et pratiques, tournant essentiellement autour des λ-calculs avec substitution explicite. Nous abordons également, mais de manière moins centrale, les modèles d'implantation dont les objets représentant les λ-termes sont des graphes non orientés connus sous le nom de réseaux d'interaction. L'objet principal de la thèse reste cependant une étude de l'impact des substitutions explicites sur des utilisations du λ-calcul : comment les substitutions explicites, enrichies de divers traits comme les adresses, les structures de données et la récursion, permettent de concevoir et démontrer de manière formelle la correction de machines abstraites ; comment le λ-calcul peut être généralisé avec des variables de réécriture tout en conservant ses principales propriétés comme la confluence et la forte normalisation d'une certaine classe de λ-termes. La partie concernant les réseaux d'interaction se justifie comme une ouverture vers d'autres techniques récentes de la modélisation du processus de β-réduction, que nous ne pourrions pas ignorer. Elle est le témoin du fait que les substitutions explicites sont seulement un choix possible de modèle opérationnel de la β-réduction. Notre objectif dans cette thèse est de mettre en évidence des situations dans lesquelles les substitutions explicites s'avèrent un bon choix pour résoudre certains types de problèmes.

Le λ-calcul

Le λ-calcul est un formalisme qui a été proposé par Alonzo Church [19] dans un but de fondation des mathématiques. Rapidement, ce but a été abandonné, et ce n'est qu'avec le développement de l'informatique qu'il connut un renouveau. Il définit les notions d'abstraction de variables et d'application d'une entité à des arguments, que l'on retrouve par exemple dans les fonctions ($f : x \mapsto f(x)$), les propositions ($\forall x\ P(x)$) ou dans les démonstrations de ces propositions (Soit x tel que $P(x), \ldots$). Le λ-calcul a donc été rapidement reconnu comme la brique de base pour modéliser toutes ces notions. Il a l'avantage d'être conceptuellement simple, et est basé sur une notion de substitution qui peut être implantée de manière automatique sur un ordinateur. L'opération principale du λ-calcul s'appelle la β-réduction et permet de réduire certains termes vers une forme canonique : la β-forme normale.

Les termes du λ-calcul (λ-termes) sont construits sur un ensemble de variables de manière inductive. Toute variable est un λ-terme ; si M et N sont des λ-termes alors leur application MN est un λ-terme ; enfin si M est un λ-terme et x une variable, alors $\lambda x.M$ est un λ-terme. Dans ce dernier cas, on parle de l'abstraction de la variable x dans le terme M. Les occurrences de x dans M sont alors dites liées. La β-réduction est définie par la fermeture reflexive, transitive et contextuelle de la règle

$$(\lambda x.M)\,N \xrightarrow[\beta]{} M\{x := N\}$$

où $M\{x := N\}$ est le résultat du remplacement de toutes les occurrences de x dans M par N. Intuitivement, cette règle exprime le résultat de l'application d'une fonction (d'un lemme, etc.) à un argument (un fait, etc.) comme l'instanciation de la variable par l'argument dans le corps de la fonction. C'est une opération classique dans de nombreux processus de calcul. La β-réduction est confluente, c'est à dire que si un λ-terme a une forme normale (un λ-terme tel que β ne peut plus être appliquée), cette forme normale est unique.

Le calcul à base de fonctions

Le λ-calcul est suffisamment expressif pour permettre de définir toute fonction que l'on peut espérer pouvoir calculer. La notion de calculabilité a été définie simultanément par Turing [80] et par Church. Turing a proposé une machine à états et à mémoire infinie, et a défini comme calculable toute fonction qui puisse être mise en œuvre sur cette machine. Church a défini la calculabilité comme la possibilité de définir une fonction par un λ-terme. Ces deux définitions sont en fait équivalentes car Turing [81] a démontré qu'une fonction est calculable par une machine de Turing si et seulement si elle est définissable par un λ-terme. C'est la thèse de Turing-Church.

Beaucoup de langages de programmation sont basés sur le modèle de fonctionnement des machines de Turing. On les appelle les *langages impératifs*. C'est le cas des langages de bas niveau (comme les langages binaires, ou les langages d'assemblage), qui doivent rester très proches de l'architecture des machines, elle même inspirée des machines de Turing. C'est aussi le cas de langages de plus haut niveau (comme le langage C), voire de très haut niveau (langages impératifs à modules, puis à objets, par exemple).

La thèse de Turing-Church suggère cependant que l'on peut définir des langages de programmation basés sur le λ-calcul. C'est le cas des langages de programmation dite fonctionnelle. Pour des raisons d'efficacité de l'exécution des programmes écrits dans ce type de langages, le λ-calcul peut être enrichi avec un opérateur de récursion, des structures de données algébriques, des constantes littérales, etc. Quoi qu'il en soit, le problème de ces langages se situe au niveau de leur implantation : ils doivent être compilés, ou interprétés, dans des langages de niveau plus bas jusqu'au langage de la machine, qui est un langage impératif. Par conséquent, il est nécessaire de bien maîtriser les mécanismes qui permettent de calculer le résultat d'un programme fonctionnel par une séquence d'instructions dans un langage impératif.

Intérêt de la programmation fonctionnelle

On peut cependant se demander pourquoi utiliser des langages de programmation fonctionnelle. La raison la plus simple que l'on puisse donner est qu'un λ-terme est un objet mathématique dont il est facile d'extraire du sens. Ce sens peut être défini simplement comme la forme normale du λ-terme ; on sait alors parfaitement définir l'équivalence (dénotationnelle) de deux programmes : ils sont équivalents s'ils se réduisent tous deux dans un troisième programme unique (leur forme normale). De même, il est aisé de travailler sur des classes particulières de termes, comme les λ-termes typables dans un certain système de type, plus ou moins expressif. Le type lui même est une information sémantique, portée par un programme, et préservée au cours de la réduction.

La programmation fonctionnelle est donc une programmation de très haut niveau. Il est possible de manipuler les fonctions comme des individus de première classe. Une fonction peut être argument d'une autre fonction (on parle souvent alors de fonctionnelle ou de fonction d'ordre supérieur) ou résultat d'une fonction. Cette possibilité offre aux langages de programmation fonctionnelle une très grande expressivité. De plus, la sémantique dénotationnelle d'un programme fonctionnel étant plus claire que celle d'un programme impératif, il est souvent beaucoup plus facile à déboguer. On peut également penser qu'il est beaucoup plus facile, et en tout cas beaucoup plus sûr, de développer des outils de transformation et d'optimisation de programmes fonctionnels. Ce sont ces aspects qui ont motivé nos recherches dans ce domaine.

On distingue au sein de la programmation fonctionnelle, deux types de langages : les langages stricts, et les langages non stricts. Un langage strict est un langage dans lequel toute fonction est stricte en ses arguments ; un langage non strict est un langage dont certaines fonctions peuvent ne pas être strictes en certains de leurs argument. Une fonction est stricte en l'un de ses arguments si, quand l'évaluation de cet argument ne termine pas alors l'évaluation de l'application de la fonction à cet argument ne termine pas. Les langages stricts sont les langages dont la stratégie d'évaluation est l'appel par valeur, c'est à dire que tout argument est évalué avant d'être appliqué à une fonction. Les langages non stricts sont les langages dont la stratégie d'évaluation est l'appel par nom, ou l'appel par nécessité. L'appel par nom est une stratégie de réduction des termes du λ-calcul, par laquelle tout argument n'est évalué que si le résultat de son évaluation est nécessaire à la poursuite du calcul. Son principal inconvénient est que chaque argument est évalué à chaque fois que son résultat est nécessaire. Une mise en œuvre plus réaliste d'un langage non strict est de partager l'évaluation des arguments utilisés à plusieurs reprises. Les stratégies basées sur ce principe sont les stratégies d'appel par nécessité. Ce ne sont pas à proprement parler des stratégies du λ-calcul, car celui-ci ne permet pas d'exprimer directement les notions de partage. Les langages en appel par nécessité, lorsqu'ils supportent les structures de données algébriques, sont appelés les langages paresseux.

Les langages paresseux permettent d'écrire des programmes avec une élégance toute particulière. Ils permettent la description de structures de données cycliques, c'est à dire des structures de données potentiellement infinies, mais qui sont dépliées en fonction des besoins. Par exemple, il est possible de définir la liste des nombres factoriels, par la pseudo expression

$$\mathsf{let}\ \ \mathrm{fact\_numbers} = \mathsf{letrec}\ \ \mathrm{facts}\ n = 1 :: (\mathrm{map}\,(\mathrm{mult}\ n)\,(\mathrm{facts}\,(n+1))\ \mathsf{in}\ \ \mathrm{facts}\ 1$$

où :: est le constructeur de listes, map est la fonctionnelle qui applique (paresseusement) une fonction aux éléments d'une liste, et mult la version currifiée de la multiplication. Calculer la factorielle d'un nombre n consiste donc à accéder au $(n+1)^{\text{ième}}$ élément de la liste fact_numbers. De plus, grâce à un mécanisme de partage, chaque nombre factoriel n'est calculé qu'une seule fois quel que soit le nombre d'accès à des nombres factoriels plus grands.

D'un autre côté, les langages en appel par valeur ne permettent pas de définir de telles structures. Cependant, ils mettent en œuvre une stratégie qui rend l'ordre d'évaluation des expressions plus clair à l'utilisateur. Il est ainsi plus facile d'enrichir les langages en appel par valeur avec des traits impératifs (ou impurs), tels l'utilisation des références, les expressions à effets de bord, ou encore les objets.

Modèles d'implantation des langages fonctionnels

Il existe donc une grande diversité de langages fonctionnels. Cependant, tous ont un point commun : l'opération de base qui permet de calculer leur résultat, la β-réduction. Du point de vue dénotationnel, si l'on considère un langage de programmation basé sur le seul λ-calcul, le résultat d'un programme est sa β-forme normale.

Cependant, le λ-calcul, dans sa généralité, n'est pas adapté pour représenter la sémantique opérationnelle de ces langages. La première raison est que la β-réduction est une opération trop forte. En effet, il n'est pas utile de calculer la forme normale (dite forte) d'un programme pour avoir une notion suffisante de son résultat. Si le résultat est une fonction, alors il est inutile de calculer la forme normale du corps de cette fonction. En particulier, si le corps de cette fonction est un terme qui n'a pas de forme normale, il est tout de même nécessaire de retourner un résultat à l'évaluation. On peut donc considérer une restriction du λ-calcul pour la programmation fonctionnelle : il s'agit de la β-réduction faible. La règle β est inchangée, mais c'est plutôt une affaire de stratégie. La réduction mise en œuvre n'est plus la fermeture contextuelle de la règle β, qui ne doit pas avoir lieu dans le corps d'une fonction, c'est à dire sous un λ. Cette restriction s'exprime sans problème dans le cadre du λ-calcul. Cependant, la β-réduction ainsi restreinte n'est plus une opération confluente. De la stratégie mise en œuvre dépend le résultat de la réduction. La seconde raison de l'inadaptation du λ-calcul pour représenter la sémantique opérationnelle des langages de programmation fonctionnelle est que la règle β ne définit pas une étape élémentaire du calcul. Rien n'est précisé quant à la façon de faire le remplacement de la variable par l'argument. Cette opération, la substitution, est définie dans la théorie uniquement par ses propriétés. De ce fait, il est sous-entendu qu'à chaque pas de β-réduction, toutes les variables sont remplacées immédiatement par l'argument correspondant. D'une part, le coût de cette opération dépend du nombre d'occurrences de la variable dans le terme, de la profondeur de ces occurrences, de la taille des termes. Il est donc impossible de mettre en relation la complexité réelle de la réduction d'un terme avec le nombre de pas de β-réduction effectués pour atteindre sa forme normale. D'autre part, dans la réalité des implantations, les choses ne se passent pas ainsi, et une telle façon de procéder mènerait à des implantations inefficaces.

Plusieurs solutions ont été mises en œuvre pour pallier ce problème. La plus ancienne est la théorie des combinateurs [76, 25, 26]. Tout λ-terme peut être traduit dans un système algébrique, le système des combinateurs, et réduit par un système de réécriture du premier ordre. Cependant, même si elle donne une complexité à la réduction des termes, cette solution n'est pas satisfaisante, car la transformation des λ-termes vers les combinateurs fait croître la taille des termes de manière quadratique, et elle provoque une duplication des calculs. C'est beaucoup plus récemment qu'est apparue une autre solution, les λ-calculs avec substitution explicite [1]. Le principe de ces calculs est de définir la substitution comme un opérateur du calcul, et un constructeur des termes. La substitution ayant le même statut que l'élimination de β-redex, les deux opérations peuvent donc cohabiter : il devient alors possible de réduire les termes partiellement. Les calculs de substitution explicite permettent donc de modéliser parfaitement les clôtures des implantations de langages de programmation fonctionnelle. Une clôture est un couple constitué d'un morceau de code (un λ-terme), accompagné de son environnement, c'est à dire de la liste des substitutions (de variables par d'autres clôtures) à opérer dans le code.

Les calculs de substitution explicite apparaissent donc comme un bon modèle pour l'implantation des langages de programmation fonctionnelle. Cependant, ils ne permettent encore pas de modéliser un aspect important des langages paresseux : le partage. En effet, ils permettent de définir les stratégies d'appel par valeur et d'appel par nom de manière un peu plus fine qu'en λ-calcul, en modélisant la notion de clôture. Mais dans la réalité des implantations de langages non stricts, il est nécessaire de mettre en œuvre un mécanisme pour partager la réduction d'une clôture qui aura été précédemment dupliquée. La stratégie qui doit être mise en œuvre est une stratégie d'appel par nécessité. Diverses tentatives ont été faites pour proposer une sémantique de cette stratégie. Launchbury [53] a proposé une sémantique à grands pas, qui n'est pas la mieux adaptée à la description des machines abstraites et ne permet pas de faire une distinction fine entre les différentes variantes de l'appel par nécessité. Ariola et al. [2] ont proposé un λ-calcul pour l'appel par nécessité, nommé λ_{let}. Ce calcul donne une description à petits pas de l'appel par nécessité. Cependant, ce calcul ne fait pas une distinction claire entre code et clôtures. D'autre part, ces sémantiques ne sont pas génériques, au sens où elles ne permettent pas de décrire les autres stratégies. $\lambda\sigma_w^a$ [15], proposé initialement par Benaissa, Rose et Lescanne, et auquel nous avons ensuite apporté notre contribution [14], est un calcul de substitution explicite qui permet une description générique des stratégies faibles de réduction des langages purs. Il fait une distinction claire entre code et clôtures, définit une notion d'adresse globale que l'on retrouve dans toutes les implantations des langages de programmation fonctionnelle paresseux, et permet une séparation claire entre le calcul et la stratégie. Par conséquent, il peut être utilisé pour modéliser aussi bien les implantations des langages stricts que des langages non stricts, et permet même de faire une distinction entre deux techniques classiques de l'implantation des langages paresseux : réducteurs de graphes et machines à environnements. De plus, $\lambda\sigma_w^a$ a été enrichi de divers traits comme les structures de données algébriques et un opérateur de récursion qui permettent de modéliser complètement les implantations.

Autres choix d'implantation

La mise en œuvre des implantations de langages de programmation fonctionnelle par les substitutions explicites représente seulement un choix, parmi d'autres possibles. En effet, comme nous l'avons déjà fait remarquer, la substitution dans le λ-calcul est définie uniquement par ses propriétés. Il existe d'autres moyens de mettre en œuvre cette opération. Une alternative aux substitutions explicites est basée sur une représentation des termes sous la forme de graphes non orientés, les réseaux d'interaction. La représentation des termes sous la forme de réseaux d'interaction est issue des travaux sur l'optimalité.

Une β-réduction optimale a été définie par Lévy [59, 60] comme une réduction qui ne duplique aucun β-redex, ni effectif, ni potentiel. (Un β-redex potentiel est une application susceptible de devenir un β-redex ; par exemple $x\,M$ est un β-redex potentiel si x peut être instancié par une abstraction, et $(M\,N)\,P$ est un redex potentiel si $M\,N$ peut se réduire en une abstraction.) Une stratégie optimale est une stratégie telle que toute réduction est optimale. Lévy a mis en évidence le fait qu'aucune des stratégies récursives classiques du λ-calcul (appels par valeur ou appel par nom) n'était optimale. Il a par ailleurs suggéré qu'il était nécessaire de mettre en œuvre des techniques de partage de clôtures pour parvenir à définir une stratégie optimale. Field [33] a cependant démontré 10 ans plus tard que le partage de clôtures n'est pas suffisant pour permettre la définition de stratégies optimales. Dans le même temps, Lamping [48] a proposé un algorithme, ou plus exactement une représentation des termes et des règles de réduction, qui permettent de rendre toute stratégie de réduction optimale. La représentation des termes permet un partage très fin de la structure. Non seulement il est possible de partager des sous-termes, mais il est également possible de partager des contextes.

Ces résultats, et plus particulièrement celui de Field, démontrent que les substitutions explicites ne permettent pas de définir des stratégies de réduction optimales. Cependant, ils ne clament pas pour autant la suprématie des implantations optimales. En effet, pour pouvoir établir une relation entre l'optimalité telle que l'aurait définie un programmeur, et l'optimalité à la Lévy, il faudrait que chaque pas de β-réduction prenne un temps de calcul constant. Hors, c'est loin d'être le cas. En effet, en plus de la règle de β-contraction (ou plus exactement ici d'élimination de β-redex), la représentation de Lamping nécessite la mise en œuvre de nombreuses autres règles de restructuration du réseau. Malheureusement, cette restructuration induit une complexité non bornée de la réduction. D'autre part, la représentation des termes sous la forme de réseaux est assez complexes à mettre en œuvre, et incompatible avec les techniques classiques de compilation. Les implantations optimales ne doivent donc pas être considérées comme les implantations suprêmes de la β-réduction, mais plutôt comme une alternative possible aux approches plus classiques, dont les applications sont encore à définir.

Le λ-calcul et la démonstration

Le λ-calcul, et la β-réduction, ne sont pas seulement la base des langages de programmation fonctionnelle. Ils sont aussi le moyen de représentation et de normalisation d'objets mathématiques : les propositions et les preuves. Pour cela, il nous faut revenir sur la notion de type d'un λ-terme.

Un système de types peut être vu comme un outil qui permet de restreindre la classe des λ-termes admissibles, afin que ces termes préservent une certaine propriété. En programmation fonctionnelle, cette propriété est qu'un programme typé ne peut pas provoquer d'erreurs de type en s'exécutant. Ce système de types peut être plus ou moins expressif, mais dans tous les cas il doit assurer une propriété de sûreté : si un terme est bien typé, alors tous ses réduits sont bien typés. C'est le cas par exemple du système de types du langage `caml`. Ce système est assez expressif, car il permet à une expression de prendre plusieurs types (des instances d'un type avec variables) en fonction de ses occurrences dans le programme.

Il existe une correspondance très fine entre systèmes de types du λ-calcul et systèmes logiques. Le type d'un terme peut être vu comme une proposition du système logique (*proposition as type*), et le terme (le programme) lui même comme une preuve de cette proposition (*proof as program*). On appelle cette correspondance la *correspondance de Curry-Howard*. Ainsi, le λ-calcul sert à coder les preuves de propositions dans des logiques dites constructives. Dans ce cadre, on peut donc fournir à la bibliothèque d'un logiciel de démonstration assitée par ordinateur comme COQ [20], des théorèmes accompagnés de leur preuve. Si la construction de preuve est un mécanisme qui nécessite une part de raisonnement humain et d'expertise, et qui est d'une grande complexité, la vérification de preuve est, par contre, un processus beaucoup plus simple et plus rapide qui peut s'effectuer de manière automatique. On voit donc bien l'intérêt d'associer des preuves aux propositions.

Dans le processus de construction de preuve d'une proposition (le pocessus qui consiste à prouver qu'il existe un terme d'un type donné, c'est à dire à le construire), il peut être très utile de manipuler des preuves incomplètes, c'est à dire des preuves utilisant d'autres preuves de lemmes qui n'ont pas encore été démontrés. Une variable (du metalangage) remplace temporairement cette preuve. Dans ce cadre, les calculs de substitution explicite peuvent être d'un grand intérêt [66]. En effet, dans le λ-calcul, où β-contraction et substitution ne sont pas définies au même niveau, il est impossible de réduire un terme avec de telles variables, la substitution n'étant pas définie sur ces variables. Avec les substitutions explicites, il est possible de suspendre la substitution, lorsqu'elle s'applique à une variable du metalangage, et à continuer la réduction par ailleurs. Le processus de substitution pourra reprendre après l'instanciation de la variable.

Un des problèmes actuels majeurs avec les preuves incomplètes est qu'il n'existe pas, à ce jour, de calcul qui assure l'unicité de la forme normale de termes équivalents, tout en préservant la terminaison de la réduction de termes pour lesquels il n'existe pas de réduction infinie dans le λ-calcul. Ce problème est abordé dans cette thèse.

Modèles pour la β-réduction forte

Dans les outils d'aide à la démonstration, il est souvent nécessaire de comparer des λ-termes entre eux, modulo la β-réduction. Pour cela, il est important de pouvoir les normaliser fortement.

Nous avons surtout mentionné jusqu'ici les modèles pour la réduction faible des λ-termes, et la programmation fonctionnelle. Les modèles d'implantation de la β-réduction forte méritent également notre attention.

Si initialement les substitutions explicites ont été introduites pour la β-réduction faible, très vite des calculs ont été proposés, qui simulent le λ-calcul dans toute sa généralité. Les substitutions sont alors un peu plus complexes, car il est nécessaire de prendre en compte les renommages qui évitent les conflits de noms. Certains calculs font l'hypothèse que ces renommages sont implicites (convention de noms de Barendregt), d'autres les gèrent de manière explicite, avec des indices qui codent les variables. Dans ce deuxième cas, il est nécessaire d'introduire de nouveaux opérateurs comme le shift ($\uparrow$) ou le lift ($\Uparrow$). Il n'est alors plus possible de considérer les substitutions comme des listes, mais ce sont des expressions algébriques beaucoup plus complexes.

La mise en œuvre d'interprètes pour la réduction forte des λ-termes n'est pas le sujet de cette thèse. On peut toutefois citer des machines abstraites telle que KNP [23, 22] ou UP [51] qui sont des généralisations des machines de réduction faible pour la réduction forte basées sur des calculs de substitution explicite. Dans ce contexte également, les interprètes optimaux à la Lamping peuvent trouver leurs applications.

Plan de la thèse

La thèse est organisée en 6 chapitres.

Le chapitre 1 est un chapitre préliminaire qui introduit les nombreux systèmes abstraits de réduction abordés dans ce document : les systèmes de réécriture de termes (TRS), qui sont le formalisme de base de nombreux calculs de substitution explicite ; le λ-calcul ; les systèmes de réduction de combinateurs (CRS), qui généralisent la notion d'abstraction du λ-calcul, et permettent de définir des systèmes de réduction d'ordre supérieur ; enfin, les CRS explicites, qui sont les CRS dans lesquels la substitution est réalisée de manière explicite, généralisant en cela l'approche par substitution explicite du λ-calcul aux CRS.

Le chapitre 2 aborde les λ-calculs avec substitution explicite dans leur généralité, les illustrant par quelques exemples. Certains problèmes relatifs à ces calculs sont introduits, notamment ceux abordés dans la suite de la thèse.

Un de ces problèmes est un problème ouvert qui est abordé dans le chapitre 3 : existe-t-il un calcul de substitution explicite qui simule le λ-calcul pas à pas, qui préserve sa forte normalisation (PSN) et qui ait la propriété de confluence sur les metatermes (metaconfluence) ? Nous tentons de répondre à cette question en définissant un calcul simulant β pas à pas, et ayant une propriété de confluence légèrement plus faible que la metaconfluence. Nous laissons ouverte la question de PSN de ce calcul. Cependant, nous ramenons cette question à celle de la PSN d'un calcul plus simple pour lequel nous espérons que la preuve de PSN se débloquera ou s'infirmera.

Un second problème introduit dans le chapitre 2 est celui de l'adéquation entre les implantations des langages de programmation fonctionnelle et ses modèles. Nous y décrivons un calcul de substitution explicite pour la réduction faible du λ-calcul, intégrant non seulement les aspects de substitution, mais également les aspects de partage, de récursion et de structures de données algébriques. Dans le chapitre 4, nous contribuons à montrer que ce modèle est adapté à la description et à la preuve de correction d'implantations, en l'utilisant pour prouver la correction d'une machine abstraite, la machine de Krivine avec partage, récursion et structures de données algébriques (KP).

Le chapitre 5 traite également de machines abstraites. Nous y proposons une alternative à la notion de clôture des langages de programmation fonctionnelle, en introduisant les super-clôtures. Une super-clôture est une structure de données qui regroupe un ensemble de clôtures partageant initialement un même environnement. Nous discutons l'utilisation des super-clôtures, proposons des machines les mettant en œuvre, et démontrons leur correction.

Enfin, dans le chapitre 6, nous discutons un autre type d'implantations de la β-réduction : les implantations par les réseaux d'interaction. Ce type d'implantations a été inventé par Lamping pour réaliser la réduction optimale des λ-termes, au sens défini par Lévy. Nous utilisons les réseaux pour définir un algorithme non optimal, nommé RINO.

Chapitre 1

Systèmes de réduction

Ce chapitre a pour but de faire les rappels nécessaires et de fixer les notations concernant les concepts utilisés tout au long de cette thèse. Il est organisé en 4 sections. Dans la section 1.1, nous introduisons les systèmes de réduction abstraits qui donnent les notions générales de réduction, confluence et terminaison. Les sections suivantes décrivent différents systèmes de réduction : les systèmes de réécriture de termes (section 1.2), certaines de leurs extensions (section 1.3), le λ-calcul (section 1.4), et les systèmes de réduction de combinateurs et de combinateurs explicites (section 1.5)

1.1 Systèmes de réduction abstraits

Les *systèmes de réduction abstraits* définissent la notion de relation abstraite qui est à la base de la plupart des processus de calcul. Nous introduisons les notions générales qui nous seront utiles par la suite au sein de quelques formalismes particuliers, comme les systèmes de réécriture de termes, les réseaux d'interaction ou le λ-calcul. La plupart des définitions de cette section sont tirées de [9].

Définition 1.1.1 (Système de réduction abstrait)

1. *Un* système de réduction abstrait *(ARS*[1]*) est un couple* $(A, \underset{R}{\rightarrow})$*, où la* réduction $\underset{R}{\rightarrow}$ *est une relation binaire sur l'ensemble* A*,* $\underset{R}{\rightarrow} \subseteq A \times A$*. On écrit généralement* $a \underset{R}{\rightarrow} b$ *au lieu de* $(a, b) \in \underset{R}{\rightarrow}$*. Lorsque cela n'introduit pas d'ambiguïté, on pourra écrire* $\rightarrow$ *au lieu de* $\underset{R}{\rightarrow}$*.*
2. *La* composition $\underset{R}{\rightarrow} \cdot \underset{S}{\rightarrow}$ *de deux ARS* $\underset{R}{\rightarrow}$ *et* $\underset{S}{\rightarrow}$ *sur* A *est l'ARS défini par* $\{(x, z) \in A \times A \mid \exists y \in A : x \underset{R}{\rightarrow} y \wedge y \underset{S}{\rightarrow} z\}$*.*
3. *Soit* $\rightarrow \subseteq A \times A$*, on définit les ARS suivants :*
   - $\xrightarrow{0} = \{(x, x) \mid x \in A\}$ *(*identité*)*
   - $\xrightarrow{i+1} = \xrightarrow{i} \cdot \rightarrow \quad (i \geq 0)$
   - $\xrightarrow{+} = \bigcup_{i>0} \xrightarrow{i}$ *(*fermeture transitive*)*
   - $\twoheadrightarrow = \xrightarrow{+} \cup \xrightarrow{0}$ *(*fermeture transitive réflexive*)*
   - $\xrightarrow{=} = \rightarrow \cup \xrightarrow{0}$ *(*fermeture réflexive*)*
   - $\leftarrow = \{(y, x) \mid x \rightarrow y\}$ *(*inverse*)*
   - $\leftrightarrow = \rightarrow \cup \leftarrow$ *(*fermeture symétrique*)*
   - $\twoheadleftarrow\!\!\!\twoheadrightarrow$ *est la fermeture transitive reflexive de* $\leftrightarrow$ *(*fermeture symétrique réflexive transitive*)*
4. *On dit que :*

[1] *Abstract Reduction System.*

- *x est R-*réductible *(ou simplement* réductible *s'il n'y a pas d'ambiguïté[2]) si et seulement s'il existe y tel que $x \underset{R}{\rightarrow} y$*
- *x est en R-*forme normale *ou est R-*irréductible *si et seulement si x n'est pas R-réductible*
- *y est une R-*forme normale *de x si et seulement si $x \underset{R}{\twoheadrightarrow} y$ et y est en R-forme normale. Si x a une et une seule R-forme normale, elle est notée $\downarrow_R (x)$.*

5. *Une relation $\rightarrow$ est dite*
   - Church-Rosser *si et seulement si $x \twoheadleftarrow\!\!\!\!\twoheadrightarrow y \Rightarrow \exists z : x \twoheadrightarrow z \twoheadleftarrow y$*
   - Confluente *si et seulement si $y_1 \twoheadleftarrow x \twoheadrightarrow y_2 \Rightarrow \exists z : y_1 \twoheadrightarrow z \twoheadleftarrow y_2$*
   - Localement confluente *si et seulement si $y_1 \leftarrow x \rightarrow y_2 \Rightarrow \exists z : y_1 \twoheadrightarrow z \twoheadleftarrow y_2$*
   - Fortement confluente *si et seulement si $y_1 \leftarrow x \rightarrow y_2 \Rightarrow \exists z : y_1 \rightarrow z \leftarrow y_2$*
   - Normalisante *si tout élément a une forme normale*
   - Fortement normalisante *s'il n'y a pas de réduction infinie*
   - Convergente *si et seulement si elle est confluente et fortement normalisante*

On représente souvent les propriétés de confluence sous la forme de diagrammes, où les parties existentielles sont représentées en pointillés. Ainsi, les propriétés de confluence, confluence locale et confluence forte sont représentées sur la figure 1.1.

$$
\begin{tikzcd}
x \arrow[r, two heads] \arrow[d, two heads] & y_1 \arrow[d, dashed, two heads] \\
y_2 \arrow[r, dashed, two heads] & z
\end{tikzcd}
\qquad
\begin{tikzcd}
x \arrow[r] \arrow[d] & y_1 \arrow[d, dashed, two heads] \\
y_2 \arrow[r, dashed, two heads] & z
\end{tikzcd}
\qquad
\begin{tikzcd}
x \arrow[r] \arrow[d] & y_1 \arrow[d, dashed] \\
y_2 \arrow[r, dashed] & z
\end{tikzcd}
$$

FIG. 1.1 – Diagrammes de confluence, confluence locale et confluence forte

Le théorème qui suit est un théorème bien connu :

Théorème 1.1.2 *$\rightarrow$ est confluente si et seulement si $\rightarrow$ est Church-Rosser.*

La propriété de confluence est très importante, puisqu'elle assure que tout élément a au plus une forme normale. De plus, si la relation est à la fois confluente et fortement normalisante (convergente), alors tout élément a une forme normale unique. Confluence et convergence ont leur importance en parallélisme et plus généralement lorsqu'un processus de calcul est mené de manière indéterministe : alors, quelle que soit la branche du calcul qui mène au résultat, ce résultat est unique.

Définition 1.1.3 *Soit $\underset{R}{\rightarrow}$ une relation sur un ensemble A, et $x \in A$. On note $x \overset{\infty}{\underset{R}{\rightarrow}}$ si et seulement s'il existe une suite $(x_i)_{i \geq 0}$ telle que $x = x_0$ et quelque soit x_i il existe x_{i+1} tel que $x_i \rightarrow x_{i+1}$. On dit que x est R-*fortement normalisable *si et seulement s'il n'existe pas une telle suite. On note* SN_R *l'ensemble des éléments de A qui sont R-fortement normalisables.*

Définition 1.1.4 (Simulation, PSN) *Soit $(\underset{R}{\rightarrow}, A_R)$ et $(\underset{S}{\rightarrow}, A_S)$ deux ARS, γ une application de A_S dans A_R, et τ une application de A_R dans A_S.*

1. *On dit que $\underset{R}{\rightarrow}$ simule $\underset{S}{\rightarrow}$ (suivant τ, γ) si et seulement si :*
   - *(a) $\tau \circ \gamma$ est l'identité sur A_S,*
   - *(b) $\forall x, y \in A_R : x \underset{R}{\rightarrow} y \Rightarrow \tau(x) \underset{S}{\twoheadrightarrow} \tau(y)$,*
   - *(c) Si x est une R-forme normale, alors $\tau(x)$ est une S-forme normale.*

[2]Pour la suite des définitions, on utilise cette même convention : le préfixe $R-$ peut être omis si cela n'introduit pas d'ambiguïté.

2. *On dit que* $\underset{R}{\rightarrow}$ *simule* $\underset{S}{\rightarrow}$ pas à pas *si* $\underset{R}{\rightarrow}$ *simule* $\underset{S}{\rightarrow}$ *et pour tout* $x_1, \ldots, x_n$ *tels que* $x_1 \underset{S}{\rightarrow} x_2 \ldots \underset{S}{\rightarrow} x_n$ *il existe une réduction* $\gamma(x_1) \underset{R}{\twoheadrightarrow} \gamma(x_2) \ldots \underset{R}{\twoheadrightarrow} \gamma(x_n)$*.*
3. *On dit que* $\underset{R}{\rightarrow}$ préserve la forte normalisation de $\underset{S}{\rightarrow}$ *si* $\underset{R}{\rightarrow}$ *simule* $\underset{S}{\rightarrow}$ *et* $\forall x \in \mathrm{SN}_S : \gamma(x) \in \mathrm{SN}_R$*.*

Nous donnons quelques définitions, lemmes et théorèmes importants utilisés dans cette thèse. Nous ne donnons pas leurs preuves qui peuvent être trouvées dans les références citées.

Notation 1.1.5 *Soit* $\underset{R}{\rightarrow}$ *et* $\underset{S}{\rightarrow}$ *deux relations. On note* $\underset{R\cup S}{\longrightarrow}$ *ou* $\underset{RS}{\rightarrow}$ *la relation* $\underset{R}{\rightarrow} \cup \underset{S}{\rightarrow}$*.*

Lemme 1.1.6 (Lemme de Newman) *Si* $\underset{R}{\rightarrow}$ *est localement confluente et fortement normalisante, alors* $\underset{R}{\rightarrow}$ *est confluente.*

La définition 1.1.7 et le lemme 1.1.8 sont dus à Yokouchi et Hikita [82], et sont d'une grande utilité pour prouver la confluence de relations qui ne sont pas fortement normalisantes. Ils sont par exemple utilisés pour la preuve de confluence de λxci au chapitre 3.

Définition 1.1.7 (Commutation de Yokouchi-Hikita) *Soit* $\underset{R}{\rightarrow}$ *et* $\underset{S}{\rightarrow}$ *des relations sur le même ensemble. On dit que* $\underset{R}{\rightarrow}$ *YH-commute sur* $\underset{S}{\rightarrow}$ *si et seulement si* $\underset{R}{\rightarrow}$ *et* $\underset{S}{\rightarrow}$ *satisfont le diagramme suivant*

$$\begin{tikzcd}
w \arrow[rrr, "S"'] \arrow[d, "R"'] & & & x \arrow[d, dashed, two heads, "R"] \\
y \arrow[r, dashed, two heads, "R"'] & \cdot \arrow[r, dashed, "S"'] & \cdot \arrow[r, dashed, two heads, "R"'] & z
\end{tikzcd}$$

Lemme 1.1.8 (Lemme de Yokouchi-Hikita) *Soit* $\underset{R}{\rightarrow}, \underset{S}{\rightarrow}, \underset{T}{\rightarrow}$ *trois relations sur le même ensemble. Si*

1. $\underset{R}{\rightarrow}$ *est fortement normalisante,*
2. $\underset{R}{\rightarrow}$ *est localement confluente,*
3. $\underset{S}{\rightarrow}$ *est fortement confluente,*
4. $\underset{R}{\rightarrow}$ *YH-commute sur* $\underset{S}{\rightarrow}$*,*

alors $\underset{R}{\twoheadrightarrow} \cdot \underset{S}{\rightarrow} \cdot \underset{R}{\twoheadrightarrow}$ *est confluente.*

On utilise généralement le lemme de Yokouchi-Hikita composé avec le lemme suivant :

Lemme 1.1.9 *Soit* $\underset{R}{\rightarrow}$ *et* $\underset{S}{\rightarrow}$ *deux relations sur le même ensemble, telles que* $\underset{S}{\rightarrow}$ *est confluente. Si* $\underset{R}{\rightarrow} \subseteq \underset{S}{\rightarrow} \subseteq \underset{R}{\twoheadrightarrow}$*, alors* $\underset{R}{\rightarrow}$ *est confluente.*

Démonstration. Si $x \underset{R}{\twoheadrightarrow} y$ et $x \underset{R}{\rightarrow} z$ alors $x \underset{S}{\twoheadrightarrow} y$ et $x \underset{S}{\twoheadrightarrow} z$ car $\underset{R}{\rightarrow} \subseteq \underset{S}{\rightarrow}$. Comme $\underset{S}{\rightarrow}$ est confluente, il existe donc w tel que $y \underset{S}{\twoheadrightarrow} w$ et $z \underset{S}{\twoheadrightarrow} w$, ou encore $y \underset{R}{\twoheadrightarrow} w$ et $z \underset{R}{\twoheadrightarrow} w$ puisque $\underset{S}{\rightarrow} \subseteq \underset{R}{\twoheadrightarrow}$. □

La définition 1.1.10 et le lemme 1.1.11 sont dûs à Dershowitz [29]. Ils sont aussi décrits dans [66], et seront très utiles dans le chapitre 3 dans le cadre de preuves de préservation de forte normalisation de λ-calculs avec substitutions explicites.

Définition 1.1.10 (Commutation de Bashmair-Dershowitz) *Soit* $\underset{R}{\rightarrow}$ *et* $\underset{S}{\rightarrow}$ *deux relations sur le même ensemble. On dit que* $\underset{R}{\rightarrow}$ *BD-commute sur* $\underset{S}{\rightarrow}$ *si et seulement si* $\underset{R}{\rightarrow}$ *et* $\underset{S}{\rightarrow}$ *satisfont le diagramme suivant*

$$\begin{tikzcd}
x \arrow[r, dashed, "S"'] \arrow[d, "R"'] & w \arrow[d, dashed, two heads, "R\cup S"] \\
y \arrow[r, "S"'] & z
\end{tikzcd}$$

Lemme 1.1.11 (Lemme de Bashmair-Dershowitz) *Soit $\underset{R}{\rightarrow}$ et $\underset{S}{\rightarrow}$ deux relations sur un ensemble A telles que*

1. *$\underset{R}{\rightarrow}$ est fortement normalisante,*
2. *$\underset{R}{\rightarrow}$ BD-commute sur $\underset{S}{\rightarrow}$.*

Quelque soit $x \in A$, si $x \in \mathrm{SN}_S$ alors $x \in \mathrm{SN}_{R \cup S}$.

1.2 Systèmes de réécriture de termes

Nous décrivons dans cette section des systèmes de réduction particuliers : les *systèmes de réécriture de termes (TRS*[3]*)* [44], basés sur une notion simple de *règle de réécriture*. Les TRS sont largement utilisés dans le cadre des λ-calculs avec substitutions explicites : $\lambda\upsilon$ et les calculs de la famille de $\lambda\sigma$ sont des TRS purs. Toutefois, les TRS ont leurs limites et ont été largement généralisés, notamment pour permettre l'utilisation de règles régies par des conditions de bord, ou encore pour permettre la représentation de termes avec variables liées. Nous décrirons dans d'autres sections de tels systèmes, qui sont à la base d'autres calculs, comme λx ou λxci.

Les systèmes de réécriture de termes sont des systèmes de réduction sur un ensemble de *termes* construits de manière inductive à partir d'une algèbre de termes, définie par un ensemble de symboles d'arité fixe (sa signature), et un ensemble infini de variables. La réduction peut ainsi être définie simplement par des règles, dites *règles de réécriture*. Nous en donnons une description rapide dans cette section. Une description plus complète peut être trouvée dans [9].

Définition 1.2.1 (Termes) *Soit Σ une signature, et X un ensemble de* variables *notées $x, y, z, \ldots$ tels que $\Sigma \cap X = \varnothing$. L'ensemble des* termes*, notés $s, t, u, \ldots$, est défini inductivement par*

- *toute variable est un terme,*
- *si f est un symbole d'arité n de Σ et $t_1, \ldots, t_n$ sont des termes, alors $f(t_1, \ldots, t_n)$ est un terme.*

On note $\mathcal{V}ar(s)$ l'ensemble des variables du terme s, c'est-à-dire l'ensemble des $x, y, z, \ldots$ qui apparaissent (qui ont une occurrence dans) s. s est dit clos si $\mathcal{V}ar(s) = \varnothing$. Les termes qui ont des variables sont appelés termes ouverts.

Si un terme t contient un sous-terme s, on note alors $t = C\{s\}$. $C\{\,\}$ est appelé un contexte.

La réécriture nécessite une notion de *substitution* (du premier ordre). Informellement, une substitution $\sigma = [x_1 \mapsto t_1; \ldots; x_n \mapsto t_n]$ où $x_1, \ldots x_n$ sont des variables et $t_1, \ldots, t_n$ sont des termes, est une application qui associe à tout terme s un autre terme s' par remplacement de toutes les occurrences des variables x_i de s par les termes t_i associés dans σ. On dit qu'un terme t est une *instance* d'un terme s si et seulement s'il existe une substitution σ telle que $\sigma(s) = t$.

Définition 1.2.2 (Réécriture)

1. *Une* règle de réécriture *est une paire (l, r) de termes telle que l n'est pas une variable et $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$. On écrit alors $l \to r$, l est appelé le* membre gauche *et r le* membre droit *de la règle de réécriture.*
2. *Un* redex[4] *est une instance du membre gauche d'une règle de réécriture. Un* Système de réécriture de termes *(TRS) est un ensemble de règles de réécritures.*
3. *Soit S un TRS. La* relation de réduction *$\underset{S}{\rightarrow}$ sur les termes est définie par $s \underset{S}{\rightarrow} t$ si et seulement s'il existe une règle de réécriture $l \to r$, un contexte $C\{\,\}$ et une substitution σ tels que $s = C\{\sigma(l)\}$ et $t = C\{\sigma(r)\}$.*

[3] *Term Rewrite System.*

[4] Abréviation anglophone de *reducible expression*.

4. Contracter *le redex signifie le remplacer par l'instance correspondante du membre droit de la règle.*

Définition 1.2.3 (Paires critiques) *Soit R un TRS.*

1. *Soit $l_i \to r_i, i = 1, 2$, deux règles de R, dont les variables ont été renommées de sorte que $\mathcal{V}ar(l_1) \cap \mathcal{V}ar(l_2) = \varnothing$. S'il existe un contexte $C\{\,\}$ et un terme s qui n'est pas une variable tels que $l_1 = C\{s\}$, et θ un unificateur plus général de s et l_2, cela détermine une* paire critique *$\langle \theta(r_1), \theta(C)\{\theta(r_2)\}\rangle$. On dit que $l_1 \to r_1$ et $l_2 \to r_2$* se chevauchent.
2. *On dit d'une paire critique $\langle t_1, t_2\rangle$ de R qu'elle est* résolue *s'il existe un terme t_3 tel que $t_1 \twoheadrightarrow_R t_3$ et $t_2 \twoheadrightarrow_R t_3$.*

Définition 1.2.4 (Linéarité, orthogonalité)

1. *Une règle de réécriture $l \to r$ est dite* linéaire à gauche *(resp.* à droite*) si aucune variable n'a plus d'une occurrence dans l (resp. r). Elle est dite* linéaire *si elle est à la fois linéaire à gauche et à droite. Par extension, un TRS est dit* linéaire *(resp.* à gauche, à droite*) si toutes ses règles sont linéaires (resp. à gauche, à droite).*
2. *Un TRS est dit* orthogonal *s'il est linéaire à gauche et sans paires critiques.*

Théorème 1.2.5 *Tout TRS orthogonal est confluent.*

Démonstration. Voir par exemple [9], corollaire 6.3.11. □

Pour prouver la terminaison d'un TRS, la technique couramment utilisée est de définir un ordre $>$ bien fondé sur les termes tel que pour toute règle de réécriture $l \to r$ on a $l > r$. On connaît un certain nombre d'ordres bien fondés sur les termes finis, notamment les *ordres récursifs sur les chemins (RPO)*. Parmi ceux-ci, on trouve l'*ordre multi-ensemble sur les chemins (MPO)* et l'*ordre lexicographique sur les chemins (LPO)*. Nous décrivons ici ce dernier, que nous utiliserons dans le chapitre 3 pour démontrer la terminaison de systèmes de substitution explicite.

Définition 1.2.6 (Ordre lexicographique sur les chemins) *Soit Σ une signature finie, et $>$ un ordre strict sur Σ. L'*ordre lexicographique sur les chemins *$>_{lpo}$ induit par $>$ est défini par $s >_{lpo} t$ si et seulement si*

- *$t \in \mathcal{V}ar(s)$ et $s \neq t$, ou*
- *$s = f(s_1, \ldots, s_m), t = g(t_1, \ldots, t_n)$, et*

$$\begin{cases} \exists i : 1 \leq i \leq m \text{ et } s_i >_{lpo} t, \ \textbf{\textit{ou}} \\ f > g, \text{ et } \forall j \text{ tel que } 1 \leq j \leq n, s >_{lpo} t_j, \ \textbf{\textit{ou}} \\ f = g, \forall j \text{ tel que } 1 \leq j \leq n, s >_{lpo} t_j, \text{ et } \exists i : 1 \leq i \leq m, s_1 = t_1, \ldots, s_{i-1} = t_{i-1}, s_i >_{lpo} t_i. \end{cases}$$

1.3 Extensions des TRS

Les TRS tels qu'ils sont définis manquent de puissance d'expressivité. Ce sont des systèmes du premier ordre, et il est par exemple impossible de définir la fonction *map* de la manière suivante :

$$map(f, Empty) \to Empty$$
$$map(f, Cons(x, l)) \to Cons(f(x), map(f, l))$$

car f apparaît à la fois comme une variable et un symbole de fonction. De même il est difficile de manipuler des termes avec variables liées, comme par exemple les λ-termes. La règle de β-réduction du λ-calcul n'est pas une règle de réécriture des TRS. Par contre, le λ-calcul peut être associé à la réécriture, soit en faisant de la réécriture *modulo les règles de conversion du λ-calcul* (ce sont les *Systèmes de réduction de combinateurs (CRS)* de Klop [43, 45] présentés à la section 1.5), soit en combinant les règles de réduction du λ-calcul et les règles de réécriture. Dans ce deuxième cas, les pas de β-réduction sont effectués explicitement et distinctement des autres pas de réécriture. Le problème est alors de démontrer quand la confluence et la terminaison sont préservées par cette combinaison. Tous ces systèmes sont appelés *Systèmes de réécriture d'ordre supérieur (HRS)*.

Une autre limite de l'expressivité des systèmes de réécriture est qu'il n'y a aucun contrôle sur l'utilisation d'une règle : considérons par exemple la fonction (qui termine[5]) suivante :

$$f(x) = \text{if } x > 1 \text{ then } \big(\text{if pair}(x) \text{ then } f(x/2) + 1 \text{ else } f(2 * (x-1) - 2)\big) \text{ else } 0$$

On peut définir f par exemple grâce au TRS suivant :

$$\begin{aligned} \mathrm{f}(0) &\to 0 \\ \mathrm{f}(1) &\to 0 \\ \mathrm{f}(x+2) &\to \mathrm{if}(\mathrm{pair}(x+2), \mathrm{f}((x+2)/2) + 1, f(2*(x+2-1)-2)) \end{aligned}$$

où if est l'opérateur conditionnel et pair le prédicat de parité des entiers naturels, définis par ailleurs, tout comme $+, -, *, /$. Les entiers sont définis de manière inductive par S et Z. Les termes clos $\mathsf{Z}, \mathsf{S}(\mathsf{Z}), \mathsf{S}(\mathsf{S}(\mathsf{Z})), \ldots$ sont abrégés par $0, 1, 2, \ldots$. Dans le membre gauche de la troisième règle, $x + 2$ est une abbréviation de $\mathsf{S}(\mathsf{S}(x))$ et non de $\mathrm{plus}(x, 2)$.

Cette définition de f pose quelques problèmes, notamment parce qu'il est possible de réduire dans la mauvaise branche du if. En particulier ce système n'est pas fortement normalisant, alors que f termine toujours ! En effet, La réduction suivante est infinie :

$$\begin{aligned} \mathrm{f}(4) &\to \mathrm{if}(\mathrm{pair}(4), \mathrm{f}(4/2) + 1, \mathrm{f}(2*3-2)) \\ &\twoheadrightarrow \mathrm{if}(\mathrm{pair}(4), \mathrm{f}(2) + 1, \underline{\mathrm{f}(4)}) \\ &\xrightarrow{\infty} \ldots \end{aligned}$$

Bien sûr, il est possible de donner une définition de f de sorte que le système soit fortement normalisant, par exemple la définition suivante

$$\begin{aligned} &\vdots \\ \mathrm{f}(x+2) &\to \mathrm{if}_f(\mathrm{pair}(x+2), x+2) \\ \mathrm{if}_f(\mathsf{true}, x) &\to \mathrm{f}(x/2) + 1 \\ \mathrm{if}_f(\mathsf{false}, x) &\to \mathrm{f}(2*(x-1)-2) \end{aligned}$$

qui nécessite l'introduction et la définition d'un nouveau symbole if_f. On préfèrera en général définir f avec une *condition de bord*.

$$\begin{aligned} \mathrm{f}(0) &\to 0 \\ \mathrm{f}(1) &\to 0 \\ \mathrm{f}(x+2) &\to \begin{cases} \mathrm{f}((x+2)/2) + 1 & \text{si pair}(x+2) \\ \mathrm{f}(2*(x+2-1)-2) & \text{sinon} \end{cases} \end{aligned}$$

[5]Ce fait est démontrable très facilement par induction sur x. On montre que $f(0), f(1), f(2)$ terminent. Considérons $N > 2$ tel que pour tout $x < N$, $f(x)$ termine. Si N est pair, alors $f(N) = f(N/2)+1$ termine car $N/2 < N$ et on peut utiliser l'hypothèse d'induction. Si N est impair, alors $f(N) = f(2*(N-1)-2) = f(N-2)+1$ car $2*(N-1)-2$ est pair et $2*(N-1)-2 > 1$. Or, $N-2 < N$ et donc $f(N)$ termine par hypothèse d'induction.

L'avantage de cette définition est d'une part qu'elle est plus concise, et d'autre part qu'elle préserve la propriété de terminaison de la fonction qu'elle veut définir.

Nous utiliserons souvent par la suite des systèmes de réécriture conditionnels (du premier ordre ou d'ordre supérieur), c'est-à-dire dont certaines règles ne pourront être instanciées que sous certaines conditions. Ces conditions doivent être définissables par un système de réécriture. De façon un peu plus formelle, une règle de réécriture de termes conditionnelle est de la forme

$$l \to r \qquad \text{si } P(x_1, \ldots, x_n)$$

où $l \to r$ est une règle de réécriture, $P()$ un prédicat calculable, et $\{x_1, \ldots, x_n\} \subseteq \mathcal{V}ar(l)$.

On trouvera souvent des règles de réécriture conditionnelles où la condition n'apparaît pas sous la forme ci-dessus. En effet, nous pourrons utiliser des notations qui nous permettent d'exprimer les conditions de manière plus compacte et plus élégante. C'est le cas par exemple pour la règle du calcul $\lambda\upsilon_c$ (voir le chapitre 3)

$$a[\uparrow]_n[s]_{n+m+1} \to a[s]_{n+m}[\uparrow_n]$$

où $n+m+1$ désigne n'importe quel entier strictement plus grand que n. Nous aurions pu également écrire cette règle de manière plus classique

$$a[\uparrow]_n[s]_m \to a[s]_{m-1}[\uparrow]_n \qquad \text{si } m > n.$$

1.4 λ-calcul

Cette section a pour but de rappeler quelques définitions et résultats du λ-calcul qui sont essentiels pour la compréhension de cette thèse. Dans un premier temps, nous présentons le λ-calcul dans un cadre non typé, et selon ses deux représentations usuelles : la représentation classique, *avec noms* [10], et la représentation *avec indices de deBruijn* [28]. Leur différence réside dans la manière de représenter les variables : l'une utilise des identificateurs, mais traite le problème des conflits de noms de manière implicite ; l'autre code les variables avec des entiers naturels, ce qui permet de traiter les problèmes de renommage de manière plus explicite. Par conséquent, l'une est plus adaptée au raisonnement humain, alors que l'autre est plus proche des implantations. En raison de ces spécificités, nous utiliserons alternativement tout au long de cette thèse les deux représentations, que l'on retrouve également dans le monde des λ-calculs avec substitutions explicites.

λ-calcul avec noms

Le λ-calcul avec noms est décrit dans [10]. La majeure partie des définitions de cette section sont issues de cet ouvrage.

Définition 1.4.1 (λ-termes) *Soit V un ensemble infini de* variables $x, y, z, x_1, \ldots, x_n, \ldots$. *L'ensemble Λ des λ-termes est défini inductivement de la manière suivante :*
- *$V \subseteq \Lambda$,*
- *Si $M \in \Lambda$ et $x \in V$ alors $(\lambda x.M) \in \Lambda$ (abstraction),*
- *Si $M \in \Lambda, N \in \Lambda$ alors $(MN) \in \Lambda$ (application).*

En général on n'écrit pas les parenthèses les plus externes et on utilise les conventions de notation suivantes :

$$\lambda x_1 x_2 \ldots x_n.M = \lambda x_1.\lambda x_2 \ldots \lambda x_n.M = (\lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.(M))\ldots)))$$
$$MN_1N_2 \ldots N_n = (\ldots((MN_1)N_2)\ldots N_n)$$

Remarque. Dans cette définition, nous manipulons deux sortes distinctes de variables : les variables notées $x, y, z, \ldots$ qui sont les variables du λ-calcul, et les *metavariables* $M, N, P, \ldots$ qui servent à décrire les termes du λ-calcul, et sont donc définies à un niveau externe de description. Il est important de distinguer les termes $\lambda x.M$ et $\lambda x.y$: le premier désigne n'importe quel terme de la forme $\lambda x.M$ où M peut être instancié par un λ-terme quelconque, par exemple par x ou par y ; on l'appellera *metaterme* ; le second est un λ-terme bien défini dont la variable y ne peut être substituée que par β-réduction (voir plus loin). Cette notion de metavariables et de metatermes est très bien traitée et généralisée dans le cadre des systèmes de réduction de combinateurs présentés à la section 1.5, qui réunissent dans un même cadre λ-calcul et réécriture. □

Définition 1.4.2 (Variables libres, termes clos, termes ouverts) *Une* variable libre *x d'un terme M est une variable qui a au moins une occurrence hors de la portée d'un λx. Les occurrences de x qui ne sont pas libres sont* liées. *Formellement, l'ensemble* $\mathrm{fv}(M)$ *des variables libres de M est défini par*

$$\begin{aligned} \mathrm{fv}(x) &= x \\ \mathrm{fv}(\lambda x.M) &= \mathrm{fv}(M) \setminus \{x\} \\ \mathrm{fv}(MN) &= \mathrm{fv}(M) \cup \mathrm{fv}(N) \end{aligned}$$

On dit que M est clos *si* $\mathrm{fv}(M) = \varnothing$, ouvert *sinon.*

Remarque. Il est important de faire la distinction entre termes clos (resp. ouverts) de la réécriture et termes clos (resp. ouverts) du λ-calcul. Dans cette section, λ-calcul et réécriture sont des formalismes sans intersection, et la confusion n'est pas gênante. Elle le sera plus dans le cadre des CRS ou des λ-calculs avec substitutions explicites. Nous introduirons dans ces cadres précis la terminologie couramment utilisée. Dans les parties de présentation des différentes théories, nous avons préféré reprendre exactement les termes usuels, tout en avertissant le lecteur. □

On introduit, d'abord de manière informelle, la *substitution* sur les λ-termes. $M\{x := N\}$ représente le résultat de la substitution de x par N dans M. Nous introduirons ensuite la β-réduction qui s'exprime par la règle de réécriture (d'ordre supérieur) $(\lambda x.M)N \underset{\beta}{\rightarrow} M\{x := N\}$. Nous devons cependant prendre certaines précautions pour la définition des ces opérations.

Définition 1.4.3 (Renommage, α-congruence)

1. *Un* renommage *dans M est le remplacement d'un sous-terme $\lambda x.N$ de M par $\lambda y.(N\{x := y\})$, où y n'a pas d'occurrence (ni libre ni liée) dans N.*
2. *M est* α-congruent *avec N, que l'on note $M \equiv_\alpha N$, si N est le résultat d'une série de renommages des variables liées de M.*

Dans la suite, les termes α-congruents sont identifiés. Ainsi, par exemple, comme $\lambda x.x \equiv_\alpha \lambda y.y$ on note $\lambda x.x \equiv \lambda y.y$ (où $\equiv$ représente l'égalité syntaxique). De même, on s'affranchit des problèmes de renommage grâce à la convention suivante :

Convention 1.4.4 (Convention de noms de Barendregt) *Si $M_1, \ldots, M_n$ apparaissent dans un certain contexte mathématique (définition, preuve), alors dans ces termes, toutes les variables liées sont choisies telles qu'elles aient des noms différents.*

La convention 1.4.4 implique un renommage implicite des variables lors des réductions. Par exemple, le terme $(\lambda x.xx)(\lambda yz.yz)$ satisfait la convention de Barendregt. Si on le réduit machinalement, c'est-à-dire sans respecter la convention en cours de réduction, on obtient la réduction suivante :

$$(\lambda x.xx)(\lambda yz.yz) \underset{\beta}{\rightarrow} (\lambda yz.yz)(\lambda yz.yz) \underset{\beta}{\rightarrow} \lambda z.(\lambda yz.yz)z \underset{\beta}{\rightarrow} \lambda z.\lambda z.zz$$

La *bonne* réduction est en fait la suivante :

$$(\lambda x.xx)(\lambda yz.yz) \underset{\beta}{\rightarrow} (\lambda yz.yz)(\lambda y'z'.y'z') \underset{\beta}{\rightarrow} \lambda z.(\lambda y'z'.y'z')z \underset{\beta}{\rightarrow} \lambda z.\lambda z'.zz'$$

> *Remarque.* À l'occasion d'exemples, et pour des raisons de clarté, nous pourrons enfreindre cette convention en ne renommant pas systématiquement certaines variables dont le lieur a été dupliqué. Toutefois nous serons bien attentifs à ce que cela n'entraine aucun conflit de noms. □

On définit à présent formellement la β-réduction $\underset{\beta}{\rightarrow}$, et d'abord la substitution :

Définition 1.4.5 (Substitution) *Le résultat de la substitution des occurrences libres de x dans M par N, $M\{x := N\}$, est défini comme suit*

$$\begin{aligned} x\{x := N\} &\equiv N \\ y\{x := N\} &\equiv y \qquad \textit{si } x \not\equiv y \\ (\lambda y.M)\{x := N\} &\equiv \lambda y.(M\{x := N\}) \\ (M_1 M_2)\{x := N\} &\equiv (M_1\{x := N\})(M_2\{x := N\}) \end{aligned}$$

Il est à noter que par la convention 1.4.4 on a forcément dans la troisième clause $y \not\equiv x$.

Cette substitution étant définie à un niveau externe par rapport à la règle de réécriture d'ordre supérieur β, elle fait partie du *metaraisonnement*. Par conséquent, nous la désignerons souvent par le terme *metasubstitution*, par opposition à la substitution *explicite* (voir le chapitre 2). En outre, il est clair que cette définition précise ce qu'est la substitution (sa sémantique dénotationnelle) sans préciser comment la calculer (sa sémantique opérationnelle).

Lemme 1.4.6 (Lemme de substitution) *Si $x \neq y$ et $x \notin \mathrm{fv}(P)$ alors*

$$M\{x := N\}\{y := P\} \equiv M\{y := P\}\big\{x := N\{y := P\}\big\}$$

Démonstration. Par induction sur la structure de M, voir [10], lemme 2.1.16. □

Nous pouvons à présent définir formellement la β-réduction.

Définition 1.4.7 (β-réduction) $\underset{\beta}{\rightarrow}$ *est la relation définie par les règles d'inférence*

$$\frac{}{(\lambda x.M)N \underset{\beta}{\rightarrow} M\{x := N\}}\ (\beta) \qquad \frac{M \underset{\beta}{\rightarrow} N}{\lambda x.M \underset{\beta}{\rightarrow} \lambda x.N}\ (\xi)$$

$$\frac{M_1 \underset{\beta}{\rightarrow} N}{M_1 M_2 \underset{\beta}{\rightarrow} N M_2}\ (\nu) \qquad \frac{M_2 \underset{\beta}{\rightarrow} N}{M_1 M_2 \underset{\beta}{\rightarrow} M_1 N}\ (\mu)$$

On dit aussi que $\underset{\beta}{\rightarrow}$ *est la* fermeture contextuelle *de la relation générée par la règle β.*

Théorème 1.4.8 $\underset{\beta}{\rightarrow}$ *est Church-Rosser.*

Démonstration. Voir [10], théorème 3.2.8. □

La β-réduction munie de Church-Rosser suffit pour définir la β-équivalence des termes normalisables : deux termes sont β-équivalents si et seulement s'ils ont la même forme normale. Une autre équivalence couramment utilisée dans le cadre du λ-calcul est la η-équivalence. Associée à la β-équivalence, elle exprime la notion d'extensionalité : deux fonctions qui sont égales en tous points de leur domaine sont équivalentes. Nous utiliserons peu la η-équivalence dans le développement de cette thèse, en fait à une seule reprise, dans le chapitre 5 dans lequel nous définirons une transformation de programmes par η-expansion.

Définition 1.4.9 (η-équivalence) *L'axiome d'η-équivalence est défini par*

$$\lambda x.M\,x =_\eta M \qquad\qquad \textit{si } x \notin \mathrm{fv}(M) \tag{η}$$

La η-équivalence est définie par la fermeture contextuelle de l'axiome (η). On parle de η-réduction lorsque cet axiome est exprimé par une règle de réécriture orientée de gauche à droite, et de η-expansion dans le sens inverse.

Définition 1.4.10 (Forme normale de tête, Forme normale de tête faible)
- *On dit qu'un λ-terme est en forme normale de tête lorsqu'il est de la forme*

$$\lambda x_1 \dots x_n.y\,N_1 \dots N_m$$

  où $n, m \geq 0$ et $N_1, \dots, N_p$ sont des λ-termes quelconques.
- *On dit qu'un λ-terme est en forme normale de tête faible lorsqu'il est de la forme $\lambda x.M$ ou x, avec M un λ-terme quelconque.*

Définition 1.4.11 (β-réduction faible) *On appelle β-réduction faible $\xrightarrow[\beta_w]{}$ la relation définie par les règles β, ν, μ du système d'inférence de la définition 1.4.7. En d'autres termes, on ne réduit pas sous les abstractions.*

Théorème 1.4.12 *La relation $\xrightarrow[\beta_w]{}$ n'est pas confluente.*

Démonstration. Le terme $(\lambda xy.xy)((\lambda z.z)(\lambda w.w))$ a deux formes normales par $\xrightarrow[\beta_w]{}$:

$$\begin{aligned}
(\lambda xy.xy)(\underline{(\lambda z.z)(\lambda w.w)}) &\xrightarrow[\beta_w]{} (\lambda xy.xy)(\lambda w.w) \\
&\xrightarrow[\beta_w]{} \lambda y.(\lambda w.w)y \\
\underline{(\lambda xy.xy)((\lambda z.z)(\lambda w.w))} &\xrightarrow[\beta_w]{} \lambda y.(\lambda z.z)(\lambda w.w)y
\end{aligned}$$

□

λ-calcul avec indices de de Bruijn

De Bruijn [28] a proposé une version du λ-calcul dans laquelle les variables ne sont pas représentées avec des noms. Son objectif était de définir un calcul permettant de résoudre de manière efficace les problèmes de renommage dans le système AUTOMATH [67].

L'idée de de Bruijn est de coder chaque occurrence d'une variable par sa profondeur d'imbrication dans le λ-terme par rapport à son lieur. Par exemple, le λ-terme $\lambda x.x$ sera codé $\lambda\,\underline{0}$, ou $\underline{0}$ signifie *la variable liée au premier λ rencontré après avoir traversé* 0 λ *en remontant vers la racine du terme.* Un autre exemple est donné par le terme $\lambda y.y(\lambda x.xy)$, qui sera codé $\lambda\,\underline{0}\,(\lambda\,\underline{0}\,\underline{1}\,)$. Cet exemple montre que deux occurrences d'une même variable (y) peuvent être codées par deux indices distincts ($\underline{0}$ et $\underline{1}$) et qu'un même indice ($\underline{0}$) peut coder des occurrences de deux variables distinctes (x et y). Les termes de de Bruijn sont définis plus formellement :

Définition 1.4.13 (Notation de de Bruijn, indices libres) *L'ensemble des termes en notation de de Bruijn est défini par* $a, b ::= \lambda a \mid a\, b \mid \underline{n}$ *avec* $n \in \mathbb{N}$. *Comme dans le* λ*-calcul avec noms, on nomme les différentes formes abstraction, application et variable. Une variable est aussi appelée* indice. *Les indices libres d'un terme sont les indices des variables libres du terme et sont donnés par* $\mathrm{fi}(a) = \mathrm{fi}_0(a)$ *où* fi_i *est défini par :*

$$\begin{array}{rcl} \mathrm{fi}_i(a\, b) & = & \mathrm{fi}_i(a) \cup \mathrm{fi}_i(b) \\ \mathrm{fi}_i(\lambda a) & = & \mathrm{fi}_{i+1}(a) \\ \mathrm{fi}_i(\underline{n}) & = & \left\{ \begin{array}{ll} \{n-i\} & n \geq i \\ \{\} & n < i \end{array} \right. \end{array}$$

On note $\lambda^n a$ *pour* $\underbrace{\lambda\lambda\ldots\lambda}_{n} a$ *et* $\lambda^0 a \equiv a$. *Les conventions d'associativité sont les mêmes que pour le* λ*-calcul avec noms.*

Remarque. Nous prenons pour convention de numéroter les indices de de Bruijn à partir de $\underline{0}$. Il n'est pas rare de trouver des présentations dans lesquelles ils sont numérotés à partir de $\underline{1}$. □

Définition 1.4.14 (β-réduction) *La* β*-réduction dans le* λ*-calcul de de Bruijn est définie comme la fermeture contextuelle de la règle*

$$(\lambda a)\, b \underset{\beta}{\rightarrow} a\{\,\underline{0} \leftarrow b\}$$

où la metasubstitution de de Bruijn est définie par

$$\begin{array}{r} (a_1\, a_2)\{\,\underline{n} \leftarrow b\} = a_1\{\,\underline{n} \leftarrow b\}\; a_2\{\,\underline{n} \leftarrow b\} \\ (\lambda a)\{\,\underline{n} \leftarrow b\} = \lambda a\{\,\underline{n+1} \leftarrow b\} \\ \underline{m}\,\{\,\underline{n} \leftarrow b\} = \left\{ \begin{array}{ll} \underline{m-1} & si\ m > n \\ \tau_n^0(b) & si\ m = n \\ \underline{m} & si\ m < n \end{array} \right. \end{array}$$

et τ_n^i *la fonction de mise à jour, ou de renommage est définie par*

$$\begin{array}{l} \tau_n^i(a\, b) = \tau_n^i(a)\, \tau_n^i(b) \\ \tau_n^i(\lambda a) = \lambda \tau_n^{i+1}(a) \\ \tau_n^i(\,\underline{m}\,) = \left\{ \begin{array}{ll} \underline{m+n} & si\ m \geq i \\ \underline{m} & si\ m < i \end{array} \right. \end{array}$$

Le λ-calcul avec noms et le λ-calcul de de Bruijn ont exactement les mêmes propriétés. On dira dans le cas du λ-calcul avec noms sous la convention de Barendregt que le renommage est *implicite*, alors que dans le λ-calcul de de Bruijn, le renommage est *explicite*. Il est possible de traduire les termes du λ-calcul de de Bruijn dans le λ-calcul avec noms grâce au système d'inférence dont la définition suit.

Définition 1.4.15 (Traduction noms/indices) *Le terme clos avec indices* a *se projette sur le terme clos avec noms* M, *que l'on note* $a \rightsquigarrow M$, *si on peut prouver* $\epsilon \vdash a \rightsquigarrow M$ *(où* ϵ *représente la* *liste vide) dans le système suivant (définition due à Rose [75]).*

$$\overline{x_0 \cdot \ldots \cdot x_n \cdot \ldots \cdot x_{n+m} \vdash \underline{n} \rightsquigarrow x_n}$$

$$\frac{x \cdot \rho \vdash a \rightsquigarrow M}{\rho \vdash \lambda a \rightsquigarrow \lambda x.M} \qquad x \notin \rho$$

$$\frac{\rho \vdash a \rightsquigarrow M \qquad \rho \vdash b \rightsquigarrow N}{\rho \vdash a\, b \rightsquigarrow M\, N}$$

Cette définition se généralise aux termes ouverts par quantification sur la liste des noms des variables libres.

Les définitions de forme normale de tête, forme normale de tête faible, réduction faible données pour le λ-calcul avec noms ont naturellement leur équivalent dans le λ-calcul de de Bruijn.

Nous utilisons dès à présent dans cette thèse plus généralement la forme du λ-calcul qui utilise des noms de variables, sous la convention de Barendregt. Nous utiliserons les indices de de Bruijn dans le cadre de λ-calculs avec substitution explicite. Les notions développées dans la suite de cette section sur le λ-calcul (résidus, stratégies, types, optimalité, etc.) sont très facilement transposables au λ-calcul à la de Bruijn.

Résidus

En réduisant des β-redex, d'autres β-redex peuvent être affectés : ils peuvent être dupliqués, éliminés, créés ou simplement modifiés. Pour formaliser cela, la notion de *résidu* a été introduite par Lévy [59]. Les résidus d'un β-redex sont en quelque sorte ce qu'il reste d'un β-redex après un certain nombre de β-réductions. La notion de résidu peut être définie formellement par un étiquetage des λ-termes qui permet de suivre l'histoire des redex.

Définition 1.4.16 (λ-calcul étiqueté) *Soit $\mathcal{L}$ un ensemble infini d'étiquettes. L'ensemble $\Lambda_{\mathcal{L}}$ des λ-termes étiquetés est défini par*

1. *Si x est une variable et $a \in \mathcal{L}$ alors $x^a \in \Lambda_{\mathcal{L}}$.*
2. *Si $M \in \Lambda_{\mathcal{L}}$ et $a \in \mathcal{L}$ alors $(\lambda x.M)^a \in \Lambda_{\mathcal{L}}$.*
3. *Si $M, N \in \Lambda_{\mathcal{L}}$ et $a \in \mathcal{L}$ alors $(MN)^a \in \Lambda_{\mathcal{L}}$.*

La β-réduction est définie par la fermeture contextuelle de la règle (β) modifiée

$$((\lambda x.M)^a\, N)^b \to M\{x := N\}$$

où la substitution est redéfinie sur les termes étiquetés, en préservant les étiquettes comme suit

$$\begin{aligned} x^a\{x := N\} &= N \\ y^a\{x := N\} &= y^a \qquad \text{si } y \not\equiv x \\ (\lambda y.M)^a\{x := N\} &= (\lambda y.M\{x := N\})^a \\ (M_1 M_2)^a\{x := N\} &= (M_1\{x := N\} M_2\{x := N\})^a \end{aligned}$$

Un étiquetage *$\mathcal{I}$ d'un λ-terme (non étiqueté) M est une fonction de Λ vers $\Lambda_{\mathcal{L}}$ qui attribue une étiquette à chaque sous-terme de M. On dit qu'un étiquetage est* initial *lorsque tous les sous-termes distincts sont étiquetés avec des étiquettes distinctes.*

Étant donné un étiquetage $\mathcal{I}$ de $M \in \Lambda$, toute réduction de M dans Λ se projette trivialement sur une réduction dans $\Lambda_{\mathcal{L}}$. La définition principale est celle de la notion exacte de résidu qui tire parti de ce fait.

Définition 1.4.17 (Résidu) *Soit $M, M' \in \Lambda$ tels que $M \twoheadrightarrow_{\beta} M'$. Soit un étiquetage initial $\mathcal{I}$ de M tel que $\mathcal{I}(M) = N$ et $N \twoheadrightarrow_{\beta} N'$ la réduction ci-dessus projetée dans $\Lambda_{\mathcal{L}}$. Soit R un sous-terme de M et R' un sous-terme de M'. On dit que R' est un* résidu *de R si et seulement si R et R' ont la même étiquette.*

Exemple 1.4.18 *Soit la réduction suivante dans Λ :*

$$\begin{aligned}\lambda yz.\underline{(\lambda x.x\,y\,x)\lambda u.(\lambda t.u)(\lambda w.w)} &\rightarrow_{\beta} \lambda yz.\underline{(\lambda u.(\lambda t.u)(\lambda w.w))y}\,\lambda u.(\lambda t.u)(\lambda w.w)\\ &\rightarrow_{\beta} \lambda yz.(\lambda t.y)(\lambda w.w)\,\lambda u.(\lambda t.u)(\lambda w.w)\end{aligned}$$

Ci-dessous, la projection de la réduction précédente sur $\Lambda_{\mathcal{L}}$ avec un étiquetage initial du terme de départ montre que $(\lambda t.y)(\lambda w.w)$ et $(\lambda t.u)(\lambda w.w)$ sont deux résidus du même redex.

$$\begin{aligned}&(\lambda y.(\lambda z.((\lambda x.((x^1\,y^2)^3\,x^4)^5)^6(\lambda u.((\lambda t.u^7)^8(\lambda w.w^9)^{10})^{11})^{12})^{13})^{14})^{15}\\ &\rightarrow_{\beta} (\lambda y.(\lambda z.(((\lambda u.((\lambda t.u^7)^8(\lambda w.w^9)^{10})^{11})^{12}y^2)^3(\lambda u.((\lambda t.u^7)^8(\lambda w.w^9)^{10})^{11})^{12})^5)^{14})^{15}\\ &\rightarrow_{\beta} (\lambda y.(\lambda z.(\underline{((\lambda t.y^2)^8(\lambda w.w^9)^{10})}^{11}(\lambda u.\underline{((\lambda t.u^7)^8(\lambda w.w^9)^{10})}^{11})^{12})^5)^{14})^{15}\end{aligned}$$

Stratégies

La règle (β) permet de définir l'équivalence de termes faiblement normalisables dans le sens où deux termes sont équivalents si et seulement s'ils ont la même β-forme normale, c'est-à-dire s'il existe pour chacun d'eux une réduction dans $\twoheadrightarrow_{\beta}$ qui mène à un terme commun. Cette définition ne dit pas s'il existe une procédure permettant de décider si deux termes sont équivalents. En effet, pour un terme qui a une forme normale, il peut exister des réductions infinies.

Exemple 1.4.19 *Soit Ω le terme non normalisable $\omega\omega$ avec $\omega = \lambda x.xx$. Les termes $(\lambda xy.y)\Omega$ et $\lambda y.y$ sont β-équivalents. Pourtant, si on considère la réduction $(\lambda xy.y)\Omega \rightarrow_{\beta} (\lambda xy.y)\Omega \rightarrow_{\beta} \ldots$, on ne peut pas décider de cette équivalence. Par contre, on le peut si on considère la réduction $(\lambda xy.y)\Omega \rightarrow_{\beta} \lambda y.y$.*

En fait, dans le cadre du λ-calcul non typé, le problème est semi- (ou partiellement) décidable, c'est-à-dire qu'on ne peut être sûr de répondre au problème de β-équivalence que si les termes ont une forme normale. Si l'un d'eux n'en a pas, la procédure de décision peut s'exécuter indéfiniment. Cependant, cette procédure de décision repose sur une notion de stratégie. En effet, comme on l'a vu dans l'exemple, il existe plusieurs manières de réduire un terme, et la manière choisie a une influence, non sur la valeur du résultat, puisque la β-réduction est confluente, mais sur l'obtention ou non de ce résultat. Une stratégie de réduction (ou plus exactement une classe de stratégies de réduction) est définie par une série de restrictions sur les règles contextuelles définissant la β-réduction. Par exemple, les stratégies *applicatives* sont définies par la restriction que N doit être une valeur pour que le redex $(\lambda x.M)N$ puisse être contracté. Nous définirons exactement la notion de valeur ultérieurement.

La notion de stratégie n'est pas seulement importante dans le cadre purement mathématique. Elle a aussi son importance dans les implantations. Par exemple, dans les langages de programmation fonctionnelle, il est important de savoir dans quel ordre vont être réduits les termes. En effet, ce choix a une influence sur l'existence ou non d'un résultat à une fonction, un programme, et une influence sur l'efficacité du calcul qui mène à ce résultat.

Par exemple, les stratégies de *réduction faible* sont définies par l'invalidation de la règle (ξ) de la définition 1.4.7. L'intersection entre les stratégies applicatives et les stratégies faibles définit les stratégies dites d'appel par valeur, que l'on retrouve dans une classe des langages de programmation fonctionnelle : les *langages applicatifs* (par exemple le langage OCAML [56] et la majeure partie des langages de la famille ML [64]). Ces stratégies donnent aux langages qui les utilisent une sémantique très différente des langages implantant l'appel par nom, comme par exemple le langage Haskell [70].

Exemple 1.4.20 *Le programme suivant, écrit dans un pseudo code que l'on peut supposer commun à tous les langages de programmation fonctionnelle*[6]*,*

$$\mathsf{letrec}\ f\,x = f\,x\ \mathsf{in}\ (\mathsf{fun}\ y \to 0)\,(f\,0)$$

est un programme qui ne termine pas en OCAML*, mais qui termine en Haskell.*

On définit plus formellement les stratégies du λ-calcul :

Définition 1.4.21 (Stratégie)

1. *Une stratégie (déterministe) est une application $F : \Lambda \to \Lambda$ telle que pour tout $M \in \Lambda : M \twoheadrightarrow_{\beta} F(M)$. F est une stratégie récursive si F est une application récursive.*
2. *Une stratégie F est une stratégie* pas à pas *si pour tout M qui n'est pas en forme normale, $M \to_{\beta} F(M)$.*
3. *F est* normalisante *si pour tout $M \in \Lambda$ qui a une forme normale, il existe $n \in \mathbb{N}$ tel que $F^n(M)$ est en forme normale.*
4. *Une stratégie non déterministe est une application F qui à tout terme M associe un ensemble de termes tels que $M \twoheadrightarrow_{\beta} N$ pour tout $N \in F(M)$.*

Définition 1.4.22 *On distingue généralement trois principales classes (non disjointes) de stratégies :*

1. *les* stratégies faibles*, c'est-à-dire telles que la réduction sous un λ soit impossible ;*
2. *les stratégies en* ordre applicatif*, c'est-à-dire telles que l'argument N soit en forme normale (selon la stratégie suivie) avant la contraction de $(\lambda x.M)N$;*
3. *les stratégies en ordre normal, qui ne font la réduction que de redex* nécessaires *(voir la définition 1.4.23).*

Une stratégie faible en ordre applicatif est dite stratégie d'appel par valeur et une stratégie faible en ordre normal est dite stratégie d'appel par nom. Il n'existe pas de stratégie à la fois en ordre applicatif et en ordre normal : en effet, un redex qui se trouve en position d'argument n'est pas nécessaire dans le cas général.

> *Remarque.* La notion de forme normale est modifiée pour les stratégies faibles. En effet, étant donné qu'il est impossible de réduire sous les λ, tout terme de la forme $\lambda x.M$ est une forme normale par une stratégie faible. On parlera généralement de *valeur* ou de *forme normale de tête faible*. Plus généralement, une valeur est un terme qui ne peut plus être réduit par la stratégie suivie. □

Les définitions et résultats suivants sont dûs à Barendregt et al. [11].

Définition 1.4.23 (Redex nécessaire) *Un redex R dans un λ-terme M est dit* nécessaire *si dans toute réduction de M en forme normale, un résidu de R est contracté.*

[6]Nous ne nous plaçons donc plus ici dans le cadre restreint du λ-calcul, mais y ajoutons un opérateur de récursion et des constantes entières.

Définition 1.4.24 (Arête, redex d'arête)

1. *Soit M un terme, on appelle l'*arête *de M la partie soulignée de* $\mathsf{hs}(M)$ *où* hs *est définie par*

$$\begin{aligned} \mathsf{hs}(x) &= \underline{x} \\ \mathsf{hs}(\lambda x.M) &= \underline{\lambda x}.\mathsf{hs}(M) \\ \mathsf{hs}(M@N) &= \mathsf{hs}(M)\underline{@}N \end{aligned}$$

   où @ est utilisé comme symbole de l'application.

2. *Un redex de M est appelé* redex d'arête *si et seulement s'il apparaît sous la forme $(\underline{\lambda x}.N)\underline{@}P$ dans* $\mathsf{hs}(M)$.

Théorème 1.4.25 *Tout redex d'arête est nécessaire.*

Démonstration. Voir [11]. □

Corollaire 1.4.26 *Les stratégies en ordre normal sont normalisantes.*

Démonstration. Elles ne réduisent que des redex nécessaires. □

Le λ-calcul permet donc de définir un certain nombre de stratégies de réduction, desquelles dépendront l'efficacité de la réduction et la possibilité ou pas de calculer systématiquement la forme normale d'un terme lorsqu'elle existe. Cependant, ces stratégies ne détaillent pas la manière dont va être réalisée la substitution : nous verrons qu'il existe différentes façons de faire la substitution, dont certaines d'ailleurs peuvent ne pas être normalisantes, quand bien même une stratégie en ordre normal est suivie pour la β-réduction. D'autre part, certains détails importants des implantations ne sont pas pris en compte par le λ-calcul, comme la possibilité de partager l'évaluation d'arguments pour les stratégies en appel par nom, donnant lieu aux stratégies d'appel par nécessité des implantations de langages fonctionnels dits paresseux.

Optimalité

La notion d'optimalité dans le λ-calcul a été définie par Lévy [59, 60], où être optimal signifie réaliser le minimum de pas de β-réduction. Intuitivement, un terme est réduit de manière optimale si dans le λ-calcul étiqueté :

1. jamais l'étiquette d'une application pouvant devenir l'étiquette d'un redex au cours de la réduction[7] n'est dupliquée ;
2. seuls des redex nécessaires sont réduits.

On parle en fait plus formellement de *familles de redex* où deux redex R_1 et R_2 dans la réduction d'un terme M appartiennent à une même famille si et seulement s'il existe une réduction (indifféremment, une autre ou la même) de M dans laquelle R_1 et R_2 sont résidus d'un même redex. Réduire de manière optimale un terme signifie ne réduire qu'une seule fois chaque famille de redex au cours de la réduction.

La question se pose donc de déterminer s'il existe des stratégies de réduction optimale. Autrement dit, existe-t-il une stratégie $\mathcal{S}$ telle que tout terme M soit réduit de manière optimale par $\mathcal{S}$? Bien entendu, les stratégies récursives classiques (en ordre applicatif ou en ordre normal) n'ont pas une telle propriété, puisqu'elles provoquent la duplication de termes, donc de redex. C'est le cas

[7] On parle de *redex potentiel*.

même pour la stratégie de réduction du redex le plus interne, quand tous les redex sont nécessaires[8]. En effet, des redex peuvent être dupliqués avant même d'être sous la forme de redex, comme le montre l'exemple suivant.

Exemple 1.4.27 *Soit le terme $M = (\lambda x.x(\lambda w.w))(\lambda y.(\lambda z.z\,z)(y\,t))$, dont toute réduction ne contracte que des redex nécessaires. Comme le montre la réduction suivante dans le λ-calcul (partiellement) étiqueté, il existe deux redex résidus d'un même sous-terme.*

$$\begin{aligned}(\lambda x.x(\lambda w.w))(\lambda y.\underline{(\lambda z.z\,z)(y\,t)^*}) &\xrightarrow[\beta]{} \underline{(\lambda x.x(\lambda w.w))(\lambda y.(y\,t)^*(y\,t)^*)}\\ &\xrightarrow[\beta]{} \underline{(\lambda y.(y\,t)^*(y\,t)^*)(\lambda w.w)}\\ &\xrightarrow[\beta]{} ((\lambda w.w)t)^*((\lambda w.w)t)^*\end{aligned}$$

Il est donc nécessaire de mettre en œuvre un partage très fin des termes pour espérer définir une telle stratégie. En fait, Field [33] a démontré en utilisant un λ-calcul avec substitution explicite étiqueté que partager des environnements, des clôtures ou des termes ne suffit pas pour définir une stratégie optimale. Lamping [48] et Kathail [42] ont proposé dans le même temps chacun une solution au problème de l'optimalité, en définissant des algorithmes dans lesquels le partage ne se limite pas à un partage de sous-termes, et en mettant en œuvre des structures de données complexes. L'algorithme de Lamping sera détaillé dans le chapitre 6.

On peut toutefois se demander si le problème de l'optimalité de la réduction du λ-calcul est une bonne approche des questions d'efficacité de la réduction. En effet, être optimal dans la définition de Lévy signifie réaliser le minimum de pas de β-contraction. Or, un pas de β-contraction n'a pas un coût constant, et pire encore, n'a pas un coût borné. Ce coût dépend entre autres du nombre d'occurrences de la variable à substituer. C'est notamment cette observation qui a donné lieu aux études sur les substitutions explicites, où l'on pense qu'être optimal signifie réaliser le minimum de pas de réduction dans un λ-calcul avec substitution explicite. Maranget [62] a d'ailleurs démontré que dans ce cadre, la stratégie paresseuse est optimale pour la réduction faible.

Typage : le cas des types simples

Il est possible de restreindre les λ-termes manipulés à une certaine catégorie de λ-termes ayant certaines propriétés de bonne formation. C'est le cas notamment des λ-termes typés dans un certain système de types. En général, de tels termes ont de bonnes propriétés, notamment ils ont un type qui est préservé par réduction, et ils sont fortement normalisables. De plus, dans le cadre de la programmation fonctionnelle, la réduction de termes typés assure qu'une certaine classe d'erreurs, les *erreurs de type*, ne peuvent pas se produire.

L'étude des systèmes de types n'est pas l'objet de cette thèse, bien que les utilisations du λ-calcul se fassent le plus souvent dans un cadre typé. Nous nous contenterons donc de présenter le plus simple des systèmes de types, le bien nommé *système des types simples*.

Définition 1.4.28 (Types simples) *Soit $\mathcal{T}$ un ensemble de* types de base*, notés $\tau, \tau_1, \ldots, \tau_n, \ldots$. Un type T est un terme défini inductivement par*

- *τ est un type simple ;*
- *Si T_1 et T_2 sont des types simples, alors $(T_1 \to T_2)$ est un type simple.*

$\to$ est associatif à droite, c'est-à-dire que $T_1 \to T_2 \to \ldots \to T_n$ est équivalent à $T_1 \to (T_2 \to (\ldots \to T_n)\ldots)$.

[8]On appelle le calcul où tous les redex sont nécessaires le calcul λI. En fait c'est une restriction du λ-calcul où toute variable liée a au moins une occurrence libre.

Définition 1.4.29 (Typage) *On dit qu'un terme M a le type T si on peut dériver une preuve de $\rho \vdash M : T$, où ρ est un contexte qui associe un type à chaque variable libre de M, et où $\rho \vdash M : T$ est défini par :*

- *si $\rho \vdash M : T_1 \to T_2$ et $\rho \vdash N : T_1$ alors $\rho \vdash MN : T_2$;*
- *si $\rho[x : T_1] \vdash M : T_2$ alors $\rho \vdash \lambda x.M : T_1 \to T_2$;*
- *Pour tout $i \in [0..n]$, on a $[x_0 : T_0; \ldots ; x_n : T_n] \vdash x_i : T_i$.*

*Si $\rho \vdash M : T_1 \to \ldots \to T_n \to \tau$ $(n \geq 0)$ on dit que M est un terme d'*arité *n, et on note* $\mathsf{ar}(M) = n$. *On dit qu'un terme M est* simplement typable *s'il existe un contexte ρ et un type T tels que $\rho \vdash M : T$.*

Théorème 1.4.30 *Tout terme simplement typable est fortement normalisable.*

Démonstration. Il existe bon nombre de preuves de forte normalisation du λ-calcul simplement typé. On retiendra cependant particulièrement la preuve simple et élégante de René David [27] en deux lemmes : Le premier lemme montre par une argumentation très simple que si M, N et $P_1, \ldots, P_n$ sont fortement normalisables, et si $M\, N\, P_1 \ldots P_n$ n'est pas fortement normalisable, alors il existe M' tel que M se réduit en $\lambda x.M'$ et $M'\{x := N\}\, P_1 \ldots P_n$ n'est pas fortement normalisable. Le second montre par une induction lexicographique sur le type de M, la β-réduction et la relation de sous-terme que si M et N sont fortement normalisables, alors $N\{x := M\}$ est fortement normalisable. Une simple induction sur la structure des termes permet de conclure que tout terme simplement typé est fortement normalisable. □

En conséquence, la β-équivalence de termes est décidable dans le λ-calcul simplement typé, quelle que soit la stratégie de réduction mise en oeuvre. On peut ajouter que la $\beta\eta$-équivalence est décidable dans le λ-calcul simplement typé. Une procédure de décisison est définie par Snyder et Gallier dans [79] qui définissent la notion de η-forme normale longue dans le λ-calcul simplement typé. Cette transformation de termes dirigée par les types est brièvement décrite dans le chapitre 5.

Le système des types simples est, comme son nom l'indique, le plus simple des systèmes de types. On souligne très souvent la correspondance (isomorphisme de Curry-Howard) entre le système des types simples, et la Déduction Naturelle de Gentzen [34]. Ainsi, il y a correspondance entre un terme (ou un programme) M de type T et la preuve M d'une proposition T. Comme il existe des systèmes logiques intuitionnistes plus riches (avec prédicats et quantificateurs notamment), il existe également des systèmes de types plus riches correspondant à ces systèmes logiques, et qui admettent une plus grande classe de termes typables. Il s'agit notamment des systèmes de types polymorphes et des systèmes de types dépendants. Nous ne les aborderons cependant pas ici, car nous nous écarterions du sujet de cette thèse.

1.5 Systèmes de réduction de combinateurs (CRS) et CRS explicites

Nous introduisons dans cette section les systèmes de réduction de combinateurs (CRS) [43, 45], un formalisme qui intègre dans un même cadre la réécriture et le λ-calcul. Différentes notions introduites dans les sections relatives à la réécriture d'une part, au λ-calcul d'autre part, sont redéfinies et renommées dans ce nouveau cadre afin d'éviter les nombreuses confusions possibles, dont le lecteur a déjà été averti à plusieurs reprises dans le développement des précédentes sections. Nous introduisons ensuite la notion de CRS explicite définie par Rose [17] et qui sera le cadre de la définition du calcul λxci du chapitre 3.

CRS

Un *système de réduction de combinateurs (CRS)* est un couple constitué d'un *alphabet* et d'un ensemble de *règles de réécriture*.

Définition 1.5.1 (Alphabet des CRS) *L'alphabet est constitué de*

1. *un ensemble infini dénombrable de* variables *notées* $x, y, z, x_1, \ldots, x_n, \ldots$,
2. *un ensemble infini dénombrable de* metavariables *notées* $X^k, Y^k, Z^k, Z_1^k, \ldots, Z_n^k, \ldots$ *où* k *est l'arité de la metavariable,*
3. *un ensemble de* symboles de fonctions *d'arité fixée,*
4. *un opérateur binaire d'abstraction noté .,*
5. *les symboles '[', ']', '(', ')' et ','.*

L'arité k *d'une metavariable* Z^k *est généralement claire dans le contexte dans lequel elle apparaît. Nous supprimerons souvent cet exposant et écrirons donc* Z *au lieu de* Z^k.

Définition 1.5.2 (Metatermes) *L'ensemble des* metatermes *sur l'alphabet de la définition 1.5.1 est défini inductivement comme suit :*

1. *les variables sont des metatermes,*
2. *si* t *est un metaterme et* x *une variable, alors* $x.t$ *est un metaterme, obtenu par* abstraction *;*
3. *si* F *est un symbole de fonction* n*-aire* ($n \geq 0$), *et* $t_1, \ldots, t_n$ *des metatermes, alors* $F(t_1, \ldots, t_n)$ *est un metaterme ;*
4. *si* $t_1, \ldots, t_k$ *sont des metatermes, alors* $Z^k[t_1, \ldots, t_k]$ *est un metaterme ; en particulier,* Z^0 *est un metaterme.*

Les metatermes sans metavariables sont des termes. *La notion de* variable liée *est la même qu'en* λ*-calcul : une occurrence de variable* x *est liée si et seulement si elle est dans le champ d'un abstracteur* (x.). *Sinon elle est libre.*

Remarque.

1. Les termes des TRS sont les metatermes des CRS dont toutes les metavariables sont d'arité nulle. Dans la suite de cette thèse, nous les appellerons donc également des metatermes.
2. Les termes du λ-calcul sont les termes des CRS dont les symboles de fonction sont l'application @ binaire et l'abstraction λ unaire. Il est important de remarquer qu'un symbole (ici λ) pour caractériser la λ-abstraction est nécessaire, notamment pour la différencier d'autres abstractions comme la μ-abstraction, par exemple.

□

Si les metavariables d'arité 0 sont les variables du premier ordre de la réécriture, nous n'avons cependant pas encore donné la signification d'une metavariable d'arité non nulle. Nous pouvons d'ores et déjà en donner une intuition avec la règle de β-réduction du λ-calcul, souvent notée de manière informelle comme suit :

$$(\lambda x.M[x])N \underset{\beta}{\rightarrow} M[N]$$

où l'on retrouve l'idée que M est en fait un contexte dans lequel la variable x peut avoir un certain nombre d'occurrences (c'est le sens de $M[x]$) et que $M[N]$ est le résultat du remplacement de cette variable x par le terme N dans M, ce qui est équivalent à $M\{x := N\}$. Ici M est une metavariable d'arité 1. Dans le terme $\lambda x.\lambda y.M[x,y]$, M est d'arité 2. $M[x,y]$ signifie que M peut contenir des occurrences des variables x et y. La règle de η-réduction est par contre notée

$$\lambda x.Mx \underset{\eta}{\rightarrow} M$$

car l'absence de x comme metaargument de la metavariable M, dans un contexte où x est explicitement lié, signifie que M ne peut être instanciée que par des termes sans occurrence de x. Cette notation permet d'exprimer un type de condition de bord simple mais fréquent (comme nous le verrons également au chapitre 3) : l'absence d'occurrence d'une variable dans un terme.

Notation 1.5.3 *$x_1.x_2.\ldots.x_n.t$ est souvent noté $x_1x_2\ldots x_n.t$. Pour un symbole de fonction unaire F on écrira souvent $Fx_1\ldots x_n.t$ au lieu de $F(x_1\ldots x_n.t)$.*

Convention 1.5.4 (convention de Barendregt) *La convention 1.4.4 du λ-calcul, nommée convention de Barendregt, est valable pour les CRS. De plus, les termes qui diffèrent seulement par un renommage des variables liées sont considérés comme syntaxiquement équivalents.*

Définition 1.5.5 (Metaterme clos) *Un metaterme est* clos *si et seulement si toutes ses variables sont liées.*

Définition 1.5.6 (Règle de réécriture des CRS) *Une* règle de réécriture *d'un CRS est un couple (l,r) noté $l \rightarrow r$, où l et r sont des metatermes tels que*

1. *l et r sont des metatermes clos ;*
2. *l a la forme $F(t_1,\ldots,t_n)$;*
3. *les metavariables qui ont une occurrence dans r ont aussi une occurrence dans l ;*
4. *les metavariables Z^k de l n'apparaissent que sous la forme $Z^k[x_1,\ldots,x_k]$ où $x_1,\ldots x_k$ sont des variables distinctes.*

Exemple 1.5.7 *$@(\lambda(x.Z[x]),Z') \rightarrow Z[Z']$ est la règle de β-réduction du λ-calcul, exprimée sous la forme d'un CRS. $\mu(x.Z[x]) \rightarrow Z[\mu(x.Z[x])]$ est une règle de μ-abstraction (opérateur de point fixe).*

Il nous reste à définir la relation de réduction associée à un CRS. Cette définition est un peu compliquée car elle nécessite l'introduction d'une metaabstraction $\underline{\lambda}$.

Définition 1.5.8 (Substitut) *Soit t un terme.*

1. *Soit $(x_1,\ldots,x_n)$ un n-uplet de variables distinctes deux à deux. L'expression $\underline{\lambda}(x_1,\ldots,x_n).t$ est un* substitut n-aire. *$\underline{\lambda}$ est un metalambda distinct de celui du λ-calcul.*
2. *Les variables $x_1,\ldots,x_n$ qui ont une occurrence libre dans t sont* liées *dans le substitut $\underline{\lambda}(x_1,\ldots,x_n).t$. Elles peuvent être renommées de la façon usuelle, à condition de ne pas introduire de conflit de nom. Les versions renommées d'un substitut sont considérées comme identiques. Les variables libres de $\underline{\lambda}(x_1,\ldots,x_n).t$ sont les variables libres de t sauf $x_1,\ldots,x_n$.*
3. *Un substitut n-aire $\underline{\lambda}(x_1,\ldots,x_n).t$ peut être appliqué à un n-uplet $(t_1,\ldots,t_n)$ de termes des CRS, résultant en la substitution simultanée suivante :*

$$(\underline{\lambda}(x_1,\ldots,x_n).t)(t_1,\ldots,t_n) = t\{x_1 := t_1,\ldots,x_n := t_n\}$$

Définition 1.5.9 (Valuation) *Une valuation σ est une application associant à chaque variable n-aire Z^n un substitut n-aire :*

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

Les valuations sont étendues homomorphiquement comme suit :

$$\begin{aligned}\sigma(x) &= x \\ \sigma(x.t) &= x.\sigma(t) \\ \sigma(F(t_1, \ldots, t_n)) &= F(\sigma(t_1), \ldots, \sigma(t_n)) \\ \sigma(Z[t_1, \ldots, t_n]) &= \sigma(Z)(\sigma(t_1), \ldots, \sigma(t_n))\end{aligned}$$

Par conséquent, si $\sigma(Z) = \underline{\lambda}(x_1, \ldots, x_n).t$ alors $\sigma(Z[t_1, \ldots, t_n]) = t\{x_1 := \sigma(t_1), \ldots, x_n := \sigma(t_n)\}$.

Définition 1.5.10 (Sûreté)

1. *Soit $l \to r$ une règle de réécriture. Un renommage de cette règle est appelé une* variante *de la règle.*
2. *Soit σ une valuation. Alors une* variante *de σ est issue d'un renommage des variables liées dans les substituts $\sigma(Z)$.*
3. *Soit $l \to r$ une règle de réécriture, et σ une valuation. Alors $l \to r$ est dite* sûre pour *σ s'il n'existe pas de metavariable Z dans l ni r telle que le substitut $\sigma(Z)$ a une occurrence de variable libre x dans une abstraction $x.\cdots$ de l ou r.*
4. *σ est dite* sûre *s'il n'y a pas de substituts $\sigma(Z)$ et $\sigma(Z')$ tels que $\sigma(Z)$ contienne une variable libre qui est également liée dans $\sigma(Z')$.*

Définition 1.5.11

1. *Soit $l \to r$ une règle de réécriture qui est sûre pour une valuation sûre σ. Alors $\sigma(l) \to \sigma(r)$ est appelée une* réécriture *ou une* contraction*. Le terme $\sigma(l)$ est appelé un* redex.
2. *Soit $\sigma(l) \to \sigma(r)$ une réécriture, et $C\{\ \}$ un contexte. Alors $C\{\sigma(l)\} \to C\{\sigma(r)\}$ est appelé un* pas de réécriture *ou* pas de réduction.

Exemple 1.5.12 *L'opérateur μ est un opérateur de récursion dont le comportement s'exprime par le CRS à une règle suivant (déjà donné dans l'exemple 1.5.7) :*

$$\mu(x.M[x]) \to M\big[\mu(x.M[x])\big]$$

Soit le terme $\lambda(y.\mu(x.x\,y))$. Une valuation sûre pour ce terme est donnée par $\sigma = M \mapsto \underline{\lambda}\,x'.x'\,y$ car

$$\sigma(\mu(x.M[x])) = \mu(\sigma(x.M[x])) = \mu(x.\sigma(M[x])) = \mu(x.\sigma(M)\,\sigma(x)) = \mu(x.(\underline{\lambda}\,x'.x'\,y)\,x) = \mu(x.x\,y)$$

Dans le contexte $\lambda(y.\Box)$, cela nous donne :

$$\lambda(y.\mu(x.x\,y)) = \lambda(y.\sigma(\mu(x.M[x]))) \to \lambda\big(y.\sigma\big(M\big[\mu(x.M[x])\big]\big)\big)$$

Or,

$$\lambda\big(y.\sigma\big(M\big[\mu(x.M[x])\big]\big)\big) = \lambda(y.\sigma(M)\,\sigma(\mu(x.M[x]))) = \lambda(y.(\underline{\lambda}\,x'.x'\,y)\,(\mu(x.x\,y))) = \lambda(y.(\mu(x.x\,y))\,y)$$

Les CRS généralisent donc les TRS en y apportant une notion d'abstraction. Ils généralisent également le λ-calcul en permettant l'expression d'autres abstractions que la λ-abstraction, et la mixité entre règles du premier ordre et ordre supérieur. Par conséquent, les CRS souffrent des mêmes inconvénients que le λ-calcul en ce qui concerne la substitution : elle est définie au niveau de la metathéorie. De plus, sa définition est un peu laborieuse. La suite décrit donc une approche similaire à l'approche des substitutions explicites pour le λ-calcul : la généralisation des substitutions explicites au cas des CRS.

CRS explicites

Les CRS sont un formalisme de réécriture d'ordre supérieur. Comme le λ-calcul (qui constitue un sous ensemble des CRS), ils sont définis grâce à une opération de substitution qui est externe au calcul, et dont le coût n'est pas borné. Bloo et Rose [17] ont donc proposé de généraliser la notion de substitution explicite du λ-calcul aux CRS.

Définition 1.5.13 (CRS explicite, ou ESCRS) *Un* CRS avec substitution explicite *(ou* CRS explicite, ESCRS*) est un CRS dont toutes les metaapplications apparaissant dans le membre droit d'une règle sont de la forme $Z[x_1, \dots, x_n]$ tel que $Z[x_1, \dots, x_n]$ apparaît également dans le membre gauche de la même règle.*

> *Remarque.* D'après cette définition, les TRS sont donc des ESCRS, puisque dans le cas des TRS il n'y a pas de metaapplication. □

Définition 1.5.14 (Explicitation des CRS [17]) *Soit R un CRS dont les symboles de fonction sont les F_i^n. L'ESCRS Rx est obtenu par les étapes suivantes (par la suite tous les vecteurs sont de taille n) :*

1. ***Extension de la syntaxe** : les* termes restreints *du CRS Rx sont définis par extension de la syntaxe de R par les clauses $\Sigma^{n+1}(\vec{x}.s, \vec{t})$ où les Σ^i $(i \geq 2)$ sont une série de symboles de fonction ajoutés à l'alphabet quand cela est nécessaire pour les règles d'introduction de la substitution ci-dessous. Les termes sans sous-termes de la forme $\Sigma^{n+1}(\vec{x}.s, \vec{t})$ sont dits* purs.
2. ***Introduction de la substitution** : Pour chaque règle (r) de R, une nouvelle règle $(r$-x$)$ de Rx est construite en remplaçant dans le membre droit toutes les metaapplications non explicites $Z^n[t_1, \dots, t_n]$ $(n \geq 1)$ par la construction*
$$\Sigma^{n+1}(\vec{x}.Z^n[x_1, \dots, x_n], \vec{t})$$
3. ***Propagation pas à pas de la substitution** : Pour chaque symbole Σ^{n+1}, une règle*
$$\Sigma^{n+1}(\vec{x}.y.Z[\vec{x}, y], \vec{X}) \to y.\Sigma^{n+1}(\vec{x}.Z[\vec{x}, y], \vec{X}) \qquad (\text{xma-}n)$$
est ajoutée, ainsi que pour chaque paire Σ^{n+1}, F^m possible, une règle
$$\Sigma^{n+1}(\vec{x}.F^m(Z_1[\vec{x}], \dots, Z_m[\vec{x}]), \vec{X}) \to F^m(\Sigma^{n+1}(\vec{x}.Z_1[\vec{x}], \vec{X}), \dots, \Sigma^{n+1}(\vec{x}.Z_m[\vec{x}], \vec{X})) \qquad (\text{x-}F^m\text{-}n)$$
4. ***Élimination de la substitution** : Pour chaque symbole Σ^{n+1}, et pour chaque $i \in \{1, \dots, n\}$, deux règles*
$$\Sigma^{n+1}(\vec{x}.x_i, \vec{X}) \to X_i \qquad (\text{xv-}n\text{-}i)$$
$$\Sigma^{n+1}(\vec{x}.Z[\vec{x'}], \vec{X}) \to \begin{cases} Z & si\ n = 1 \\ \Sigma^n(\vec{x'}.Z[\vec{x'}], \vec{X'}) & si\ n > 1 \end{cases} \qquad (\text{xgc-}n\text{-}i)$$
où
$$\vec{x'} = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$$
$$\vec{X'} = (X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_n)$$

Si la relation définie par R est notée $\rightarrow$ alors La relation définie par Rx est notée $\underset{\mathrm{ie}}{\rightarrow}$. La sous-relation définie uniquement par les règles d'introduction (de nom (r-x)*) est notée $\underset{\mathrm{i}}{\rightarrow}$, et la sous-relation ne contenant que les règles de propagation et d'élimination de la substitution (de nom* (x-$*$)*) est notée $\underset{\mathrm{e}}{\rightarrow}$.*

Les propriétés essentielles de cette transformation s'expriment par les théorèmes suivants. Leurs démonstrations peuvent être trouvées dans [17].

Théorème 1.5.15 *Pour tout CRS R, Rx est un ESCRS.*

Théorème 1.5.16 *Pour tout CRS confluent $\rightarrow$, $\underset{\mathrm{e}}{\rightarrow}$ est confluent et fortement normalisant.*

Théorème 1.5.17 *Soit s, t des R-termes : $s \underset{\mathrm{ie}}{\twoheadrightarrow} t$ si et seulement si $s \twoheadrightarrow t$.*

Théorème 1.5.18 *Si R est confluent alors Rx est confluent.*

De plus, sous certaines conditions que nous ne détaillerons pas ici, Rx préserve la forte normalisation de R. Cependant, cette propriété n'est pas vraie dans le cas général.

Chapitre 2

λ-calculs avec substitution explicite

Dans le λ-calcul, la substitution est définie à un niveau externe au calcul. De ce fait, la contraction du redex $(\lambda x.M)N$ implique, dans un seul pas de calcul, le remplacement immédiat de toutes les occurrences de x dans M par N. Un tel modèle de calcul a deux principaux inconvénients :

1. La substitution a un coût qui dépend de la structure de M, notamment du nombre d'occurrences de x et de leur profondeur dans le terme. Ce coût n'est donc pas borné. De ce fait, déterminer la complexité de la réduction d'un λ-terme comme le nombre de β-contractions effectuées n'est pas tout à fait pertinent.
2. Dans la réalité des implantations, la substitution n'est généralement que partiellement appliquée. Dans les langages de programmation fonctionnelle par exemple, les interpréteurs manipulent non pas des termes, mais des clôtures, qui sont des objets plus riches. De même, dans les systèmes de déduction, on souhaite manipuler des termes avec des variables du premier ordre, sur lesquels la substitution n'est pas définissable. Ici aussi, la notion de clôture est essentielle.

On voudrait donc utiliser des calculs qui permettent de modéliser plus finement la sémantique opérationnelle de l'évaluation des fonctions : des calculs ayant un sens opérationnel, et pour une classe plus large de fonctions, notamment les fonctions (ou les preuves) non complètement définies (incomplètes).

Ceci justifie l'utilité de décrire la substitution au même niveau que la règle de β-contraction (que nous nommerons alors règle de β-élimination ou d'élimination de β-redex). De tels systèmes de calcul nous donnent alors l'opportunité de décrire des compilateurs de langages fonctionnels très simples, et dont la preuve de correction est facilitée par la possibilité de représenter les clôtures directement dans la théorie, comme nous le montrerons au chapitre 5. Ou encore, la possibilité de manipuler des objets beaucoup plus riches que les λ-termes purs, tout en bénéficiant de propriétés essentielles et formellement établies, qui permettent de faire des raisonnements sûrs dans les systèmes de démonstration.

Nous décrivons dans cette section les λ-calculs avec substitution explicite $\lambda\mathrm{x}$ [16, 75, 74] (section 2.1) et $\lambda\upsilon$ [13] (section 2.3) qui seront utilisés par la suite. Ces deux calculs ont les mêmes propriétés, et leur différence principale se situe au niveau des choix de représentation des variables : l'un utilise des noms, l'autre des indices. C'est à la fois pour leur simplicité, et pour illustrer leurs ressemblances et leurs différences que nous présentons ces calculs. Entre temps, nous aurons défini quelques généralités et quelques propriétés des substitutions explicites à la section 2.2. Nous présentons ensuite un troisième calcul, $\lambda\sigma_{\Uparrow}$ [68] à la section 2.4, utilisant comme $\lambda\upsilon$ les indices de de Bruijn mais dont les propriétés sont différentes. Enfin, nous décrivons à la section 2.5 $\lambda\sigma_w$ [40], un calcul dédié à la description des implantations des langages de programmation fonctionnelle, et un de ses dérivés, $\lambda\sigma_w^a$ [15], qui intègre entre autres les notions d'adresse et de partage. Nous conclurons ces généralités à la section 2.6.

2.1 Un exemple de λ-calcul avec substitution explicite : λx

λx a été proposé par Kristoffer Rose dans [75]. Une de ses particularités est la représentation des variables avec des noms, sous la convention de Barendregt (convention 1.4.4).

Les premiers calculs de substitution explicite proposés dans la littérature sont des calculs avec indices de de Bruijn, donc des calculs dans lesquels les renommages sont effectués de manière explicite, et considérés dans de nombreux cas comme des substitutions. Rose propose de faire abstraction au niveau théorique des problèmes de renommage, tout comme Barendregt l'a fait pour le λ-calcul, et comme cela a été généralisé dans les CRS. C'est une façon de s'affranchir de détails souvent lourds du calcul de renommage, qui permet d'analyser certains problèmes dans un cadre plus simple, mais tout aussi riche. Vu sous un autre angle, λx est un ESCRS obtenu à partir du λ-calcul par la procédure d'explicitation décrite dans la section 1.5 et dans [17].

En outre, il est évident que la simplicité de λx en fait un outil pédagogique très intéressant pour introduire les calculs de substitution explicite. C'est aussi la raison pour laquelle nous présentons dans cette thèse λx avant les généralités sur les calculs de substitution explicite.

Les termes de λx sont les λ-termes étendus avec un nouveau constructeur : la substitution. Plus formellement :

Définition 2.1.1 *Les termes de* λx *sont définis par la grammaire suivante :*

$$M, N ::= \lambda x.M \;\mid\; MN \;\mid\; x \;\mid M\langle x := N\rangle$$

Le terme $M[x]\langle x := N\rangle$ (λx est un ESCRS, d'où la notation $M[x]$ qui signifie x peut –et non doit– être libre dans tout terme instanciant M) correspond au terme M dans lequel la substitution de x par N est explicite. Contrairement au λ-calcul, $M[x]\langle x := N\rangle$ ne représente pas le résultat de la substitution de x par N dans M, mais le terme dans lequel, dans le futur, x sera remplacé par N dans M. Le calcul λx est défini par les règles de la figure 2.1.

$$(\lambda x.M[x])N \underset{\lambda x}{\longrightarrow} M[x]\langle x := N\rangle \tag{b}$$
$$(M_1[x]\, M_2[x])\langle x := N\rangle \underset{\lambda x}{\longrightarrow} M_1[x]\langle x := N\rangle M_2[x]\langle x := N\rangle \tag{xap}$$
$$(\lambda y.M[x,y])\langle x := N\rangle \underset{\lambda x}{\longrightarrow} \lambda y.M[x,y]\langle x := N\rangle \tag{xab}$$
$$x\langle x := N\rangle \underset{\lambda x}{\longrightarrow} N \tag{xv}$$
$$M\langle x := N\rangle \underset{\lambda x}{\longrightarrow} M \tag{xgc}$$

FIG. 2.1 – Le calcul λx (comme ESCRS)

La règle (b) initialise la β-contraction. La metasubstitution de la règle β est remplacée par une substitution explicite, et les équations définissant la metasubstitution ont été orientées pour donner les règles du calcul de substitution. On peut remarquer que la règle (xgc) a une condition de bord implicite, définie par la théorie des ESCRS, selon laquelle la variable x n'a pas d'occurrence dans M, d'où la notation $M\langle x := N\rangle$ et non $M[x]\langle x := N\rangle$.

Dans ce calcul, les pas de β-contraction et de substitution peuvent donc être entremêlés, contrairement au λ-calcul où la substitution est réalisée jusqu'au bout après chaque pas d'élimination de β-redex. Cette possibilité a plusieurs conséquences.

1. Il devient possible de définir de nouvelles stratégies plus fines qu'avec le λ-calcul, la stratégie de substitution étant mise en compétition avec la stratégie de sélection du prochain β-redex (ou plus exactement, du prochain b-redex). Ainsi, il est possible de modéliser des implantations où β-contraction et substitution sont réalisées au même niveau des machines abstraites de réduction.
2. Inversement, il est possible que certaines stratégies du λ-calcul ne puissent plus être simulées par le λ-calcul avec substitution explicite. Cet aspect constitue une perte de généralité d'un calcul de substitution explicite par rapport au λ-calcul. Ce n'est pas le cas pour λx.
3. Il n'est plus certain qu'un terme fortement normalisable du λ-calcul reste fortement normalisable dans un λ-calcul avec substitution explicite donné. De plus, il n'est pas certain non plus qu'un terme ait une unique forme normale. Ce n'est pas le cas pour λx qui préserve la forte normalisation du λ-calcul et est confluent.

Exemple 2.1.2 *Il existe* 2 *chemins de β-réduction pour réduire le terme* $(\lambda x.x)((\lambda y.y)(\lambda z.z))$ *jusqu'à sa forme normale :*

$$\begin{tikzcd}
(\lambda x.x)((\lambda y.y)(\lambda z.z)) \arrow[r, "\beta"'] \arrow[d, "\beta"'] & (\lambda x.x)(\lambda z.z) \arrow[d, "\beta"'] \\
(\lambda y.y)(\lambda z.z) \arrow[r, "\beta"'] & \lambda z.z
\end{tikzcd}$$

Avec λx*, il existe* 6 *chemins de réduction du même terme jusqu'à sa forme normale :*

$$\begin{tikzcd}
(\lambda x.x)((\lambda y.y)(\lambda z.z)) \arrow[r, "\mathrm{b}"'] \arrow[d, "\mathrm{b}"'] & (\lambda x.x)(y\langle y := \lambda z.z\rangle) \arrow[r, "\mathrm{xv}"'] \arrow[d, "\mathrm{b}"'] & (\lambda x.x)(\lambda z.z) \arrow[d, "\mathrm{b}"'] \\
x\langle x := (\lambda y.y)(\lambda z.z)\rangle \arrow[r, "\mathrm{b}"'] \arrow[d, "\mathrm{xv}"'] & x\langle x := y\langle y := \lambda z.z\rangle\rangle \arrow[r, "\mathrm{xv}"'] \arrow[d, "\mathrm{xv}"'] & x\langle x := \lambda z.z\rangle \arrow[d, "\mathrm{xv}"'] \\
(\lambda y.y)(\lambda z.z) \arrow[r, "\mathrm{b}"'] & y\langle y := \lambda z.z\rangle \arrow[r, "\mathrm{xv}"'] & \lambda z.z
\end{tikzcd}$$

Il existe dans un λ-terme des substitutions qui peuvent être immédiatement éliminées par la règle (xgc). On les appelle *substitution déchet*.

Définition 2.1.3 (Substitution déchet) *On dit qu'une occurrence de la substitution $\langle x := N\rangle$ est une* substitution déchet *si et seulement si elle est appliquée à un terme dont x n'est pas une variable libre. Autrement dit, $\langle x := N\rangle$ est une substitution déchet dans $M\langle x := N\rangle$.*

Nous venons de suggérer que les λ-calculs avec substitution explicite avaient certaines propriétés de préservation par rapport au λ-calcul. Nous détaillerons ces propriétés dans la section 2.2. Nous avons introduit le calcul λx avant de définir les généralités, dans le but de l'utiliser comme illustration de ces généralités. Il sera donc l'objet de nombreux exemples de la section suivante.

2.2 Généralités et conventions

Dans cette section, nous traitons de manière générale les λ-calculs avec substitution explicite. Nous en donnons une définition formelle. Cette définition est suggérée par Rose [17].

Définition 2.2.1 (λ-calcul avec substitution explicite) *On dit qu'un système de réduction est un* λ-calcul avec substitution explicite *si c'est un ESCRS (éventuellement conditionnel) qui simule* $\xrightarrow[\beta]{}$.

Cette définition est assez large pour inclure tous les λ-calculs avec substitution explicite que l'on peut trouver dans la littérature. En effet, elle intègre des calculs qui ne gèrent pas les renommages de manière explicite, comme λx, des calculs dont les règles sont restreintes par des conditions de bord, comme λxci (voir le chapitre 3), ou des calculs qui ne simulent pas $\xrightarrow[\beta]{}$ pas à pas, comme $\lambda\zeta$ [65, 66]. Cette définition étend la définition communément admise de calcul de substitution explicite, basée sur les Catégories Cartésiennes Closes [1, 3] (systèmes de réécriture du premier ordre). Tout système de réécriture du premier ordre étant un ESCRS, il est clair que notre définition généralise la précédente. Elle n'est cependant pas trop générale, la seule meta-opération des ESCRS étant l'opération de renommage des variables suivant la convention de noms de Barendregt.

Convention 2.2.2 *Soit λq un λ-calcul avec substitution explicite quelconque*[1]. *λq possède généralement deux types de règles, comme défini dans le cadre des ESCRS : une règle d'*initialisation de la β-contraction*, généralement notée* (b)*, et des règles de* propagation de la substitution. *Lorsque c'est le cas*[2]*, on appelle* (q)*, le système constitué uniquement des règles de substitution, le* calcul de substitution (explicite) sous-jacent à λq. *La relation engendrée par le système λq pourra être notée indifféremment $\xrightarrow[\lambda q]{}$ ou $\xrightarrow[bq]{}$ en accord avec la convention de notation 1.1.5. En général,* (q) *pourra être encore décomposé en plusieurs sous-systèmes.*

Exemple 2.2.3 *On appelle* (x) *le système constitué uniquement des règles* (xv, xgc, xab, xap) *de* λx. $\xrightarrow[\mathrm{bx}]{}$ *simule* $\xrightarrow[\beta]{}$*, avec γ la fonction identité (voir la définition 1.1.4) et $\tau = \downarrow_{\mathrm{x}}$*[3].

Nous nous intéresserons aux propriétés suivantes des λ-calculs avec substitution explicite :

1. *simulation pas à pas de* $\xrightarrow[\beta]{}$,
2. *confluence sur les termes*,
3. *confluence sur les metatermes*, ou *metaconfluence*,
4. *forte normalisation du calcul de substitution q*,
5. *préservation de la forte normalisation du λ-calcul (PSN).*

Exemple 2.2.4 $\xrightarrow[\mathrm{bx}]{}$ *simule* $\xrightarrow[\beta]{}$ *pas à pas, et est confluent sur les termes.* $\xrightarrow[\mathrm{x}]{}$ *est fortement normalisant, et* $\xrightarrow[\mathrm{bx}]{}$ *est PSN. Cependant, il n'est pas confluent sur les metatermes. En effet, il existe une paire critique entre les règles* (b) *et* (xap) *qui n'est pas résolue, représentée sur la figure 2.2. Il est seulement possible de démontrer par induction sur la structure de M que les termes $M[x,y]\langle x := P\rangle\langle y := N[x]\langle x := P\rangle\rangle$ et $M[x,y]\langle y := N[x]\rangle\langle x := P\rangle$ se réduisent vers un résultat commun, c'est-à-dire que* λx *est confluent sur les termes. C'est le* lemme de substitution *des substitutions explicites.*

Un des problèmes ouverts des λ-calculs avec substitution explicite est de définir un calcul possédant simultanément les 5 propriétés énumérées ci-dessus. Les propriétés de PSN et de confluence donnent au calcul la propriété de convergence sur les termes fortement normalisables. En démonstration, le λ-calcul permet de représenter les preuves de propositions. Une des problématiques actuelles de la démonstration automatisée est de pouvoir manipuler des *preuves incomplètes* [66]. Une preuve incomplète est une preuve *à trous*, représentable par un metaterme. La propriété de

[1] q peut être instancié par x, $\upsilon, \sigma, \sigma_{\Uparrow}, \sigma_w, \zeta, \tau, \chi$, s, se, w, we, xc, xc1, xci, …, tous ceux que nous oublions, et les autres à venir.

[2] C'est le cas pour tous les calculs présentés dans cette thèse, et à notre connaissance pour tous les calculs existants.

[3] Comme nous le verrons plus loin, x est convergent, ce qui nous permet de donner cette définition.

$$\begin{tikzcd}[column sep=small]
 & & \underline{(\,\underline{(\lambda y.M[x,y])\,N[x]}\,)\langle x := P\rangle} \arrow[d, "\mathrm{xap}"] \arrow[ddddll, bend right, "\mathrm{b}"'] \\
 & & \underline{(\lambda y.M[x,y])\langle x := P\rangle}\ N[x]\langle x := P\rangle \arrow[d, "\mathrm{xab}"] \\
 & & \underline{(\lambda y.M[x,y]\langle x := P\rangle)\ N[x]\langle x := P\rangle} \arrow[d, "\mathrm{b}"] \\
 & & M[x,y]\langle x := P\rangle\big\langle y := N[x]\langle x := P\rangle\big\rangle \arrow[ddl, dotted, dash] \\
M[x,y]\langle y := N[x]\rangle\langle x := P\rangle \arrow[dr, dotted, dash] & & \\
 & ? &
\end{tikzcd}$$

FIG. 2.2 – Une réduction divergente sur les metatermes de λx

metaconfluence permet donc de s'assurer que tout chemin de réduction d'une preuve incomplète mène à une forme normale unique. Cette propriété ne doit cependant pas être obtenue au prix de la perte de PSN.

Dans le cadre des calculs de substitution explicite, nous parlerons de calculs avec *concaténation*, de calculs avec *permutation*, et de calculs avec *composition*. Il est important de définir ce que nous entendons par chacun de ces termes, car la confusion peut être grande. Informellement, un calcul de substitution explicite est un calcul avec *concaténation*, s'il existe une règle qui concatène deux substitutions pour n'en former qu'une seule. Une règle de concaténation peut être utile pour des raisons d'efficacité, car il est plus rapide de regrouper les substitutions avant de les propager que de les propager une à une. Une règle de *permutation* est une règle du calcul qui modifie la séquencialité des éléments des substitutions, c'est à dire qui provoquent un changement des positions relatives de deux termes à l'intérieur d'une séquence de substitutions. Par exemple, les calculs λx et $\lambda\upsilon$ (section 2.3) n'ont ni règle de concaténation, ni règle de permutation, alors que $\lambda\sigma_{\Uparrow}$ (section 2.4) a à la fois une règle de concaténation (Clos) et des règles de permutation (MapEnv, LiftEnv). On dit que $\lambda\sigma_{\Uparrow}$ met en œuvre la *composition* des substitutions. Le calcul λxci, présenté au chapitre 3, n'a pas de règle de concaténation, mais une règle de permutation. Il n'est *a priori* pas nécessaire d'introduire un opérateur de concaténation pour définir un calcul metaconfluent ; par contre, la définition de règles de permutation est inévitable. En effet, par exemple dans λx, si l'on veut résoudre la paire critique $\big\langle M[x,y]\langle x := N[y]\rangle\langle y := P\rangle, M[x,y]\langle y := P\rangle\langle x := N[y]\langle y := P\rangle\rangle\big\rangle$, il faudra bien que la substitution de x et la substitution de y aient, dans un sens ou dans l'autre, permuté.

Cette distinction entre concaténation, permutation et composition n'est pas si clairement établie dans l'article de Fereira, Kesner et Puel [32], où il est démontré qu'il existe des calculs avec composition (λ_d, λ_{dn}) qui préservent la forte normalisation du λ-calcul. En fait, les calculs en question sont des calculs avec concaténation, mais sans permutation des substitutions, il ne s'agit donc pas de composition au sens où nous l'entendons. Il est d'ailleurs démontré dans l'article en question qu'il existe un calcul sans concaténation, mais toujours sans permutation, nommé λ_f dont la forte normalisation est préservée par λ_d. Aucun de ces calculs n'est metaconfluent. Ce résultat intéressant va dans le même sens que notre observation : la clé du problème de la définition d'un calcul metaconfluent réside, non pas dans la possibilité de concaténer les substitutions, mais dans la possibilité de les permuter. Une règle de permutation trop générale étant la source de non PSN de plusieurs calculs, la clé du problème réside dans la manière de contrôler cette règle de permutation. C'est ce qui est fait dans le chapitre 3 de ce document.

2.3 Un calcul avec indices : $\lambda\upsilon$

$\lambda\upsilon$ [13] est un λ-calcul avec substitution explicite très proche de λx et qui possède les mêmes propriétés, mais dont les variables sont représentées par des *indices de de Bruijn*. Par conséquent, les renommages sont traités de façon explicite. Nous le présentons ici dans la perspective de son utilisation au chapitre 3, mais aussi car il fait partie des premiers calculs pour lesquels la préservation de la forte normalisation a été démontrée. La syntaxe de $\lambda\upsilon$ ainsi que les règles du calcul sont données sur la figure 2.3.

Syntaxe.

$$\begin{array}{rcll} a, b & ::= & a\,b \ \mid\ \lambda a \ \mid\ \underline{n} \ \mid\ a[s] & \text{(Termes)} \\ s, t & ::= & a/ \ \mid \Uparrow(s) \ \mid \uparrow & \text{(Substitutions)} \end{array}$$

Règles.

$$\begin{array}{rcll} (\lambda a)b & \xrightarrow[\lambda\upsilon]{} & a[b/] & \text{(B)} \\ (a\,b)[s] & \xrightarrow[\lambda\upsilon]{} & a[s]\ b[s] & \text{(App)} \\ (\lambda a)[s] & \xrightarrow[\lambda\upsilon]{} & \lambda a[\Uparrow(s)] & \text{(Lambda)} \\ \underline{0}\,[a/] & \xrightarrow[\lambda\upsilon]{} & a & \text{(FVar)} \\ \underline{n+1}\,[a/] & \xrightarrow[\lambda\upsilon]{} & \underline{n} & \text{(RVar)} \\ \underline{0}\,[\Uparrow(s)] & \xrightarrow[\lambda\upsilon]{} & \underline{0} & \text{(FVarlift)} \\ \underline{n+1}\,[\Uparrow(s)] & \xrightarrow[\lambda\upsilon]{} & \underline{n}\,[s][\uparrow] & \text{(RVarlift)} \\ \underline{n}\,[\uparrow] & \xrightarrow[\lambda\upsilon]{} & \underline{n+1} & \text{(VarShift)} \end{array}$$

FIG. 2.3 – Le calcul $\lambda\upsilon$

$\lambda\upsilon$ dispose de trois opérateurs particuliers : le *shift* ($\uparrow$), le *lift* ($\Uparrow$) et le *slash* (/). shift et lift peuvent être vues comme des applications du domaine des variables, que l'on appelle opérateurs de renommage. Intuitivement, shift peut être vu comme une application des indices vers les indices, qui a le comportement suivant :

$$\uparrow:\ \underline{n} \mapsto \underline{n+1}$$

et lift comme une application des indices vers les termes, dont le comportement est le suivant :

$$\begin{array}{lrl} \Uparrow(s): & \underline{0} & \mapsto \underline{0} \\ & \underline{1} & \mapsto s(\underline{0})[\uparrow] \\ & \underline{2} & \mapsto s(\underline{1})[\uparrow] \\ & & \ \ldots \\ & \underline{n+1} & \mapsto s(\underline{n})[\uparrow] \\ & & \ \ldots \end{array}$$

$a/$ est une application des indices vers les termes :

$$
\begin{array}{lrl}
a/: & \underline{0} & \mapsto a \\
 & \underline{1} & \mapsto \underline{0} \\
 & \underline{2} & \mapsto \underline{1} \\
 & & \dots \\
 & \underline{n+1} & \mapsto \underline{n} \\
 & & \dots
\end{array}
$$

Comme λx, $\lambda\upsilon$ préserve la forte normalisation du λ-calcul, est confluent sur les termes clos, mais n'est pas metaconfluent. En effet, $\lambda\upsilon$ ne dispose pas d'une règle de permutation de substitutions, indispensable pour la metaconfluence. Cependant, la caractéristique de $\lambda\upsilon$ est sa simplicité : c'est un TRS pur qui s'exprime en peu de règles.

2.4 Un calcul avec indices et avec composition : $\lambda\sigma_\Uparrow$

$\lambda\sigma_\Uparrow$ [68] fait partie d'une autre catégorie de λ-calculs avec substitution explicite : les $\lambda\sigma$ calculs. Contrairement à $\lambda\upsilon$ ou λx, les substitutions peuvent être composées, et sont donc des substitutions simultanées. De plus, $\lambda\sigma_\Uparrow$ dispose de règles de permutation non contrôlées. En conséquence, les propriétés de $\lambda\sigma_\Uparrow$ sont très différentes des propriétés de $\lambda\upsilon$ ou de λx : comme l'a montré Melliès, la PSN est perdue [63] au profit de la metaconfluence.

Historiquement, $\lambda\sigma_\Uparrow$ est un des héritiers directs des premiers λ-calculs avec substitution explicite [1]. Il est issu de $\lambda\sigma$, transformé afin d'obtenir la propriété de confluence sur les termes ouverts. On peut le rapprocher de λs_e [41], calcul ayant exactement les mêmes propriétés (sauf que les substitutions sont permutées mais non concaténées, donc λs_e ne met pas en œuvre la composition des substitutions) et dont la PSN a été pendant longtemps conjecturée, jusqu'à ce que Guillaume [39] démontre le contraire avec un contre exemple très similaire à celui de Melliès. La syntaxe et les règles de $\lambda\sigma_\Uparrow$ sont présentées sur la figure 2.4.

2.5 Vers les implantations de langages fonctionnels

Une des applications possibles des λ-calculs avec substitution explicite et la modélisation des implantations des langages de programmation fonctionnelle. En effet, les λ-calculs avec substitution explicite disposent des structures nécessaires pour pouvoir exprimer la notion de clôture. Cependant, comme le λ-calcul, ils souffrent de deux inconvénients majeurs :

– ils sont trop généraux en permettant un grand nombre de stratégies de réduction. En réalité, en programmation fonctionnelle, seules les stratégies faibles sont mises en œuvre.
– ils ne permettent pas de modéliser les notions de partage mises en œuvre par les langages de programmation fonctionnelle paresseuse ; ils ne permettent pas non plus de modéliser d'autres traits des langages de programmation foctionnelle comme la récursion ou les structures de données.

La conséquence du premier point est que les λ-calculs avec substitution explicite sont souvent trop complexes pour leur utilisation en tant que modèles d'implantation : ils disposent de règles et de constructeurs inutiles, comme l'opérateur de lifting ($\Uparrow$) qui n'ont pas de raison d'être alors que les substitutions n'ont pas à être propagées dans le corps des fonctions, ou comme les règles de permutation. De plus leurs propriétés ne cadrent pas avec les propriétés attendues pour modéliser les implantations, en particulier la confluence (problématique lorsqu'on se restreint à la réduction

Syntaxe.

$$
\begin{array}{rcll}
M, N & ::= & \underline{n} \;\mid\; MN \;\mid\; \lambda M \;\mid\; M[s] & \text{(Termes)} \\
s, t & ::= & \mathrm{id} \;\mid \uparrow \mid\; M \cdot s \;\mid\; s \circ t \;\mid \Uparrow(s) & \text{(Substitutions)}
\end{array}
$$

Règles.

$$
\begin{array}{rcll}
(\lambda M)\, N & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & M[N \cdot \mathrm{id}] & \text{(Beta)} \\
(M\, N)[s] & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & M[s]\, N[s] & \text{(App)} \\
(\lambda M)[s] & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & \lambda M[\Uparrow(s)] & \text{(Lambda)} \\
M[s][t] & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & M[s \circ t] & \text{(Clos)} \\
\underline{n}\,[\uparrow] & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & \underline{n+1} & \text{(VarShift1)} \\
\underline{n}\,[\uparrow \circ s] & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & \underline{n+1}\,[s] & \text{(VarShift2)} \\
\underline{0}\,[M \cdot s] & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & M & \text{(FVarCons)} \\
\underline{0}\,[\Uparrow(s)] & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & \underline{0} & \text{(FVarLift1)} \\
\underline{0}\,[\Uparrow(s) \circ t] & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & \underline{0}\,[t] & \text{(FVarLift2)} \\
\underline{n+1}\,[M \cdot s] & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & \underline{n}\,[s] & \text{(RVarCons)} \\
\underline{n+1}\,[\Uparrow(s)] & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & \underline{n}\,[s \circ \uparrow] & \text{(RVarLift1)} \\
\underline{n+1}\,[\Uparrow(s) \circ t] & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & \underline{n}\,[s \circ (\uparrow \circ t)] & \text{(RVarLift2)} \\
(s \circ t) \circ u & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & s \circ (t \circ u) & \text{(AssEnv)} \\
(M \cdot s) \circ t & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & M[t] \cdot (s \circ t) & \text{(MapEnv)} \\
\uparrow \circ (M \cdot s) & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & s & \text{(ShiftCons)} \\
\uparrow \circ \Uparrow(s) & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & s \circ \uparrow & \text{(ShiftLift1)} \\
\uparrow \circ (\Uparrow(s) \circ t) & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & s \circ (\uparrow \circ t) & \text{(ShiftLift2)} \\
\Uparrow(s) \circ \Uparrow(t) & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & \Uparrow(s \circ t) & \text{(Lift1)} \\
\Uparrow(s) \circ (\Uparrow(t) \circ u) & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & \Uparrow(s \circ t) \circ u & \text{(Lift2)} \\
\Uparrow(s) \circ (M \cdot t) & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & M \cdot (s \circ t) & \text{(LiftEnv)} \\
\Uparrow(\mathrm{id}) & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & \mathrm{id} & \text{(LiftId)} \\
\mathrm{id} \circ s & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & s & \text{(IdL)} \\
s \circ \mathrm{id} & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & s & \text{(IdR)} \\
M[\mathrm{id}] & \xrightarrow[\lambda\sigma_{\Uparrow}]{} & M & \text{(Id)}
\end{array}
$$

FIG. 2.4 – Le calcul $\lambda\sigma_{\Uparrow}$

faible), et la PSN (un calcul peut ne pas avoir PSN alors que sa restriction à la réduction faible l'a). Curien Hardin et Lévy [68] ont donc défini un λ-calcul avec substitution explicite restreint pour la β-réduction faible, ayant de bonnes propriétés et que nous présentons dans cette section. Benaissa et al. [15, 12] sont allés plus loin en enrichissant ce calcul avec des adresses permettant de modéliser le partage, un opérateur de récursion, et des structures de données, donnant un calcul qui est présenté également dans cette section et qui peut servir de base sémantique à toutes les stratégies de réduction des langages fonctionnels purs.

Réduction faible : le calcul $\lambda\sigma_w$.

Comme cela est établi par le théorème 1.4.12, la β-réduction faible n'est pas une relation confluente. C'est le cas également si l'on définit de la même manière, c'est-à-dire par une restriction relative au contexte (suppression de la règle ξ), une relation de réduction faible à partir d'un λ-calcul avec substitution explicite.

Au lieu de cela, Curien, Hardin et Lévy [68] ont défini un λ-calcul avec substitution explicite pour la réduction faible nommé $\lambda\sigma_w$ où la substitution est utilisée comme activateur de la β-réduction. En d'autres termes, si on applique une substitution particulière, la substitution identité, à un λ-terme pur, et que l'on applique les règles du calcul, alors les seules β-contractions qui peuvent être effectuées sont hors du champs d'un λ. La syntaxe et les règles de $\lambda\sigma_w$ sont présentées sur la figure 2.5.

Syntaxe.

$$
\begin{array}{rll}
V, W & ::= M[s] \mid VW \mid \underline{n} & \text{(Termes faibles)} \\
M, N & ::= \lambda M \mid MN \mid \underline{n} & \text{(Termes purs)} \\
s, t & ::= W \cdot s \mid \mathrm{id} & \text{(Substitutions)}
\end{array}
$$

Règles.

$$
\begin{array}{rll}
(\lambda M)[s]W & \xrightarrow[\lambda\sigma_w]{} M[W \cdot s] & (\mathrm{B}_w) \\
(MN)[s] & \xrightarrow[\lambda\sigma_w]{} M[s]N[s] & (\mathrm{App}) \\
\underline{0}\,[W \cdot s] & \xrightarrow[\lambda\sigma_w]{} W & (\mathrm{FVar}) \\
\underline{n+1}\,[W \cdot s] & \xrightarrow[\lambda\sigma_w]{} \underline{n}\,[s] & (\mathrm{RVar}) \\
\underline{n}\,[\mathrm{id}] & \xrightarrow[\lambda\sigma_w]{} \underline{n} & (\mathrm{VarId})
\end{array}
$$

FIG. 2.5 – Le calcul $\lambda\sigma_w$

La représentation des substitutions de $\lambda\sigma_w$ est très proche de la notion de clôture des langages de programmation fonctionnelle : on démarre la réduction dans un environnement vide (modélisé par la substitution id), et l'environnement est étendu au cours de la réduction. La classe des termes admissibles est donc restreinte par rapport au calcul $\lambda\sigma$. La règle de β-contraction est contrôlée par la substitution, de la même manière que dans une machine abstraite. La propriété principale de $\lambda\sigma_w$ est la confluence établie par le théorème suivant.

Théorème 2.5.1 $\lambda\sigma_w$ *est confluent.*

Démonstration. $\lambda\sigma_w$ est un système orthogonal. D'après le théorème 1.2.5, il est donc confluent. □

Cependant, si $\lambda\sigma_w$ est adaptée à la modélisation des clôtures des langages de programmation fonctionnelle, il ne permet pas l'expression du partage, nécessaire pour des implantations efficaces des langages paresseux. De plus, il est bien entendu qu'un langage de programmation fonctionnelle ne se satisfait pas uniquement de la β-réduction : il y manque des notions de récursion et de structures de données. Ces trois traits ont été ajoutés au calcul $\lambda\sigma_w$, donnant lieu au calcul présenté dans la suite.

Adresses, partage, récursion et structures de données : $\lambda\sigma_w^a$ et ses dérivés.

Nous présentons dans cette section le calcul $\lambda\sigma_w^a$, qui est un calcul de substitution explicite avec des adresses, proposé initialement par Zino Benaissa, Pierre Lescanne et Kristoffer Rose [15] et auquel nous avons ensuite contribué [14]. $\lambda\sigma_w^a$ est un λ-calcul avec substitution explicite issu de $\lambda\sigma_w$. Il inclut la notion d'adresse globale, et permet de modéliser les notions de partage des implantations de langages de programmation fonctionnelle paresseux. La syntaxe et les règles de $\lambda\sigma_w^a$ sont décrits sur la figure 2.6.

Les pointeurs de la forme E^a peuvent être interprétés par : la structure partagée représentant le terme E, dont la racine est située à l'adresse a. De ce fait, les réductions dans ce calcul s'effectuent sur le principe des réductions parallèles (ou simultanées), i.e. toute réduction à une adresse donnée est répercutée simultanément sur tous les termes ayant la même adresse (opération qui dans la pratique a un coût indépendant du nombre d'occurrences du redex réduit, puisqu'on manipule des références dans la mémoire). On ne considère alors dans ce calcul que des termes *admissibles*, i.e. pour lesquels à une même adresse ne peuvent pas correspondre deux structures différentes. La notation a fraîche signifie le choix d'une adresse disponible pour faire une nouvelle référence à une structure. Il ne s'agit pas forcément d'une nouvelle adresse, mais d'une adresse qui n'existe pas (ou plus) dans le terme. Les algorithmes de choix d'adresses libres (ou fraîches) ne sont pas explicités, et dépendent de l'implantation, en particulier des choix en matière de glanage de cellules.

Deux règles de $\lambda\sigma_w^a$ ont le même membre gauche : (FVarG) et (FVarE). La différence entre ces deux règles est le choix de l'adresse dans le membre droit qui est soit b (FVarG) soit a (FVarE). Ce choix conceptuel a une correspondance directe avec le problème d'implantation *duplication ou partage*. Si l'adresse est b, le membre droit est une redirection du pointeur qui fait référence à l'adresse a (où se trouve la structure représentant le terme $\underline{0}[E^b.s]$) vers l'adresse b (où se trouve la structure représentant le terme E). En conséquence, les prochaines réécritures (parallèles) de cet argument seront partagées avec toutes ses autres occurrences dans le terme. Si l'adresse est a, une copie de la racine de la structure représentant le terme E à l'adresse b est effectuée. Alors l'accès à $\underline{0}[E^b.s]$ à l'adresse a est remplacé par cette copie, et les prochaines réécritures de cet argument ne seront pas partagées. Ces deux règles introduisent un non-déterminisme. Par exemple, on peut normaliser le terme $(\,\underline{0}\,[\,\underline{n}^b]^c\ \underline{0}\,[\,\underline{n}^b]^d)^a$ de quatre manières différentes suivant les adresses. Les résultats sont $(\underline{n}^b\,\underline{n}^b)^a$, $(\underline{n}^b\,\underline{n}^d)^a$, $(\underline{n}^c\,\underline{n}^b)^a$ et $(\underline{n}^c\,\underline{n}^d)^a$. Il est clair qu'en effaçant les adresses de ces quatre termes, on obtient le même terme $\underline{n}\,\underline{n}$. La différence entre eux est le degré de partage qu'on obtient après une dérivation. L'utilisation de (FVarG) maintient le partage existant dans la structure initiale, alors que (FVarE) le fait décroître.

Outre les règles d'introduction et d'élimination des substitutions, $\lambda\sigma_w^a$ présente une règle spéciale qui permet le glanage dans les substitutions des termes qui ne sont jamais substitués, ce qui permet d'éviter l'accumulation de termes inutiles dans les substitutions.

La figure 2.6 donne une définition des pré-termes et de la pré-réduction. En raison des adresses globales, il est nécessaire de restreindre la classe des termes admissibles, *i.e.* qui représentent des graphes, et de donner une définition de réduction qui préserve la propriété d'admissibilité. Nous

Syntaxe. Les *prétermes adressés* sont definis par

$$\begin{array}{rll} T,U,V & ::= E^a \mid \bot & \text{(Pointeur)} \\ E,F & ::= M[s] \mid UV \mid \underline{n} & \text{(Contexte d'évaluation)} \\ M,N & ::= \lambda M \mid MN \mid \underline{n} & \text{(Code)} \\ s,t & ::= \mathrm{id} \mid U \cdot s & \text{(Substitution)} \end{array}$$

où a,b,c sont des *adresses* appartenant à un ensemble infini d'adresses $\mathbf{A}$.

Règle d'introduction de substitution

$$((\lambda M)[s]^b\, U)^a \to M[U \cdot s]^a \qquad (\mathrm{B}_w)$$

Système d'élimination de la substitution défini par les règles

$$\begin{array}{rll} (MN)[s]^a \to (M[s]^b\, N[s]^c)^a & b,c \text{ fraîches} & \text{(App)} \\ \underline{0}\,[E^b \cdot s]^a \to E^b & & \text{(FVarG)} \\ \underline{0}\,[E^b \cdot s]^a \to E^a & & \text{(FVarE)} \\ \underline{n+1}\,[U \cdot s]^a \to \underline{n}\,[s]^a & & \text{(RVar)} \\ \underline{n}\,[\mathrm{id}]^a \to \underline{n}^a & & \text{(VarId)} \end{array}$$

Règle de glanage de substitution.

$$M[s]^a \to M[s|_{\mathrm{fi}(M)}]^a \qquad s \neq s|_{\mathrm{fi}(M)} \qquad \text{(Collect)}$$

où la *restriction de substitution* est définie par $s|_I = s|_I^0$ où $\mathrm{id}|_I^i = \mathrm{id}$ et

$$(U \cdot s)|_I^i = \begin{cases} U \cdot s|_I^{i+1} & \text{si } i \in I \\ \bot \cdot s|_I^{i+1} & \text{sinon} \end{cases}$$

FIG. 2.6 – Le pré-calcul $\lambda\sigma_w^a$

en donnons un aperçu par la définition suivante. Une définition plus formelle peut être trouvée dans [15, 14].

Définition 2.5.2 (Termes, réduction) *Soit t un pré-terme.*

1. *On note* addr(t) *l'ensemble des adresses qui ont une occurrence dans t.*
2. *L'ensemble des sous-termes externes de t situés à l'adresse a est noté $t@a$ où $t@a = \{t_1^a, \ldots, t_n^a\}$ si et seulement s'il existe un contexte n-aire $C\{\}$ tel que $t = C\{t_1^a, \ldots, t_n^a\}$ et $a \notin \mathrm{addr}(C\{\})$.*
3. *t est un* pré-terme admissible *(un terme) si et seulement si toute adresse $a \in \mathrm{addr}(t)$ satisfait $t@a = \{s^a\}$ et $a \notin \mathrm{addr}(s)$.*
4. *On note $E^a \Vdash\!\xrightarrow{b} E'^{a'}$ la réduction simultanée de toutes les occurrences du redex situé à l'adresse b dans le terme E^a. Soit A un ensemble d'adresses, on note $E^a \Vdash\!\xrightarrow{A} E'^{a'}$ la réduction simultanée de tous les redex situés aux adresses de l'ensemble A.*

Une des particularités de $\lambda\sigma_w^a$ est d'offrir la possibilité de définir des stratégies de réduction qui incluent les aspects de partage de façon très fine. En définissant des restrictions sur la réduction, il est possible de modéliser toutes les stratégies classiques mises en œuvre dans les implantations des langages de programmation fonctionnelle. Une stratégie se caractérise par un algorithme qui détermine le choix des prochaines adresses où effectuer la réduction (ou de l'adresse dans le cas déterministe) et le choix de la règle à mettre en œuvre dans le cas où plusieurs règles peuvent s'appliquer. C'est le cas en particulier pour les règles (FVarG) et (FVarE). La sélection de cet ensemble d'adresses s'exprime par des restrictions sur les règles suivantes, où $U \vdash A$ signifie que A est le prochain ensemble d'adresses où effectuer les réductions dans U :

$$\frac{}{M[s]^a \vdash \{a\}}\ (\mathrm{Scl}) \qquad \frac{U \vdash A_1 \quad U \vdash A_2}{U \vdash A}\ A = A_1 \cup A_2 \quad (\mathrm{Par})$$

$$\frac{s \vdash A}{M[s]^a \vdash A}\ (\mathrm{Sub}) \qquad \frac{U \vdash A}{U \cdot s \vdash A}\ (\mathrm{Hd}) \qquad \frac{s \vdash A}{U \cdot s \vdash A}\ (\mathrm{Tl})$$

$$\frac{}{(UV)^a \vdash \{a\}}\ (\mathrm{Sap}) \qquad \frac{U \vdash A}{(UV)^a \vdash A}\ (\mathrm{Lap}) \qquad \frac{V \vdash A}{(UV)^a \vdash A}\ (\mathrm{Rap})$$

où on demande en plus que pour (Sap) et (Scl) une règle soit effectivement applicable.

Les différentes stratégies sont exprimées dans le tableau de la figure 2.7. CBN désigne la stratégie call by name (appel par nom), et CBV call by value (appel par valeur), les stratégies définies par Plotkin [72]. CBNeedE et CBNeedG désignent deux variantes de la stratégie call by need (appel par nécessité), celle employée dans les machines à environnement, et celle employée dans les réducteurs de graphe. La différence réside dans le choix de la règle (FVarE) ou (FVarG). Il est à noter que certaines implantations mettent en œuvre des stratégies mixtes, où les deux règles peuvent coexister. C'est le cas par exemple pour la stratégie mise en œuvre par la machine STG [71], où (FVarE) est la règle utilisée pour les accès aux codes des fonctions, mais où (FVarG) est utilisée au moment de l'empilement des arguments, afin d'éviter une ou plusieurs indirections. Une application $\underline{m}\ \underline{n}$[4]

[4]En STG les applications sont uniquement des applications de variable.

Strategie	(FVar ?)	(Scl)	(Par)	(Sub)	(Hd)	(Tl)	(Sap)	(Lap)	(Rap)
✓	E or G	✓	✓	✓	✓	✓	✓	✓	✓
CBN	E	✓	×	×			①	¬ ①	×
CBV	E	✓	×	×			① ∧ ②	¬ ①	① ∧ ¬ ②
CBNeedE	E	③	×	¬ ③	✓	×	①	¬ ①	×
CBNeedG	G	✓	×	×			①	¬ ①	×
Specul-$\|_n$	E	✓	④	✓	✓	✓	✓	✓	✓
Spine-$\|_n$	E	✓	④	✓	✓	✓	✓	✓	×

✓ : "toujours applicable ;" × : "jamais applicable ;" blanc : "sans objet ;" ① : U est une valeur ; ② : V est une valeur ; ③ : si $s = U \cdot s'$ et $M = \underline{0}$ alors U est une valeur ; et ④ : $\#A \leq n$.

FIG. 2.7 – Stratégies pour $\lambda\sigma_w^a$.

dans un environnement $[\ldots \cdot E_i^{a_i} \cdot \ldots]$ se réduit grossièrement comme suit :

$$
\begin{aligned}
(\,\underline{m}\ \underline{n}\,)[\ldots \cdot E_i^{a_i} \cdot \ldots]^b &\overset{b}{\Vdash\!\!\!\rightarrow} (\,\underline{m}\,[\ldots \cdot E_i^{a_i} \cdot \ldots]^c\,\underline{n}\,[\ldots \cdot E_i^{a_i} \cdot \ldots]^d)^b && \text{(App)}\\
&\overset{d}{\Vdash\!\!\!\twoheadrightarrow} (\,\underline{m}\,[\ldots \cdot E_i^{a_i} \cdot \ldots]^c E_n^{a_n})^b && \text{(RVar, FvarG)}\\
&\overset{c}{\Vdash\!\!\!\twoheadrightarrow} (\,\underline{0}\,[E_m^{a_m} \cdot \ldots]^c E_n^{a_n})^b && \text{(RVar)}\\
&\Vdash\!\!\!\twoheadrightarrow (\,\underline{0}\,[E'^{a'_m}_m \cdot \ldots]^c E_n^{a_n})^b \quad \text{tel que } E'^{a'_m}_m \text{ est une valeur} && (\ldots)\\
&\overset{c}{\Vdash\!\!\!\rightarrow} (E'^{c}_m E_n^{a_n})^b && \text{(FVarE)}
\end{aligned}
$$

Enfin, les deux dernières stratégies sont des stratégies parallèles.

Définition 2.5.3 *Soit $\mathcal{S}$ une stratégie définie par des restrictions sur les règles de la figure 2.7, et t un terme.*

1. *On note $t \vdash_{\mathcal{S}} A$ si et seulement si $t \vdash A$ en respectant les restrictions définies par $\mathcal{S}$.*
2. *On note $t \underset{/\mathcal{S}}{\Vdash\!\!\!\rightarrow} t'$ si et seulement si $t \vdash_{\mathcal{S}} A$ et $t \overset{A}{\Vdash\!\!\!\rightarrow} t'$.*

Le calcul $\lambda\sigma_w^a$ permet également de modéliser les structures de données algébriques et un opérateur de récursion. Pour les structures de données, la syntaxe des termes doit être étendue pour inclure

$$
\begin{aligned}
E, F \ &::= \ \ldots \ \mid \ U\star\langle \vec{C} : \vec{N} \rangle[s]\\
M, N \ &::= \ \ldots \ \mid \ C_i \ \mid \ M\star\langle \vec{C} : \vec{N} \rangle
\end{aligned}
$$

C_i dénote les éléments d'un ensemble fini de symboles distincts des autres symboles et nommés *constructeurs*. $M\star\langle \vec{C} : \vec{N} \rangle$ est une opération dont la sémantique intuitive est : si M se réduit en un constructeur C_i, alors exécuter le code N_i. $\langle \vec{C} : \vec{N} \rangle$ est une abréviation pour $\langle C_1 : N_1, \ldots, C_m : M_m \rangle$, les constructeurs C_i sont d'*arité* fixée $\mathsf{ar}(C_i)$ tels que $\mathrm{fi}(C_i) = \{0, \ldots, \mathsf{ar}(C_i) - 1\}$ et $\mathrm{fi}(M\star\langle \vec{C} : \vec{N} \rangle) = \mathrm{fi}(M) \cup \big(\bigcup_{i=1}^m \mathrm{fi}(N_i) \setminus \mathrm{fi}(C_i)\big)$. $\star$ a une plus grande précédence que la juxtaposition (application) et λ. Par conséquent, $M_1 M_2\star\langle \vec{C} : \vec{N} \rangle$ doit être compris comme $M_1(M_2\star\langle \vec{C} : \vec{N} \rangle)$ et $\lambda M\star\langle \vec{C} : \vec{N} \rangle$ comme $\lambda(M\star\langle \vec{C} : \vec{N} \rangle)$. La précédence de $\star$ sur $[\_]$ dépend du type de son membre gauche, soit terme pur, soit terme adressé. Bien qu'il ne puisse pas y avoir d'ambiguïté, nous n'écrirons pas $M\star\langle \vec{C} : \vec{N} \rangle[s]$ qui est plutôt déroutant, mais $(M\star\langle \vec{C} : \vec{N} \rangle)[s]$. Les règles (Match) et (Select) décrivent le comportement opérationnel d'une expression match ... with ... (filtrage).

$$(M\star\langle\vec{C}:\vec{N}\rangle)[s]^a \to (M[s]^b\star\langle\vec{C}:\vec{N}\rangle[s])^a \quad a \text{ fraîche} \tag{Match}$$
$$(C_i[\vec{T}\cdot s]^b\star\langle\vec{C}:\vec{N}\rangle[t])^a \to N_i[\vec{T}\cdot t]^a \tag{Select}$$

où $\vec{T}\cdot s = T_1\cdot\ldots\cdot T_{\mathsf{ar}(C_i)}\cdot s$.

La définition exprime que les seules réductions qui impliquent les données sont de déclencher la réduction de l'expression à filtrer (Match) et de sélectionner l'entrée du sélecteur de cas correspondant au C_i particulier auquel il a été appliqué (Select).

Exemple 2.5.4 *Considérons la réduction (par la stratégie CBNeedG) du terme* $M = (\lambda\,\underline{0}\star\langle \mathsf{Z}:\mathsf{Z},\mathsf{S}:\underline{0}\rangle)\,((\lambda\mathsf{S})\,\mathsf{Z})$ *où* Z *est un constructeur d'arité* 0 *et* S *un constructeur d'arité* 1. *Ce terme correspond intuitivement au programme* $(\mathsf{fun}\ x \to (\mathsf{match}\ x\ \mathsf{with}\ \mathsf{Z}\to\mathsf{Z}\ \mid\ \mathsf{S}\,y\to y))\,(\mathsf{S}\,\mathsf{Z})$.

$$\begin{array}{rlr}
M[\mathrm{id}]^a & \overset{a}{\Vdash\!\!\to} \big((\lambda\,\underline{0}\star\langle\mathsf{Z}:\mathsf{Z},\mathsf{S}:\underline{0}\rangle)[\mathrm{id}]^b\,((\lambda\mathsf{S})\,\mathsf{Z})[\mathrm{id}]^c\big)^a & \text{(App)}\\
& \overset{a}{\Vdash\!\!\to} (\underline{0}\star\langle\mathsf{Z}:\mathsf{Z},\mathsf{S}:\underline{0}\rangle)\big[((\lambda\mathsf{S})\,\mathsf{Z})[\mathrm{id}]^c\cdot\mathrm{id}\big]^a & (\mathrm{B}_w)\\
& \overset{a}{\Vdash\!\!\to} (\underline{0}\,\big[((\lambda\mathsf{S})\,\mathsf{Z})[\mathrm{id}]^c\cdot\mathrm{id}\big]^d\star\langle\mathsf{Z}:\mathsf{Z},\mathsf{S}:\underline{0}\rangle\big[((\lambda\mathsf{S})\,\mathsf{Z})[\mathrm{id}]^c\cdot\mathrm{id}\big])^a & \text{(Match)}\\
& \overset{d}{\Vdash\!\!\to} \big(((\lambda\mathsf{S})\,\mathsf{Z})[\mathrm{id}]^c\star\langle\mathsf{Z}:\mathsf{Z},\mathsf{S}:\underline{0}\rangle\big[((\lambda\mathsf{S})\,\mathsf{Z})[\mathrm{id}]^c\cdot\mathrm{id}\big]\big)^a & \text{(FVarG)}\\
& \overset{c}{\Vdash\!\!\to} \big(((\lambda\mathsf{S})[\mathrm{id}]^e\,\mathsf{Z}[\mathrm{id}]^f)^c\star\langle\mathsf{Z}:\mathsf{Z},\mathsf{S}:\underline{0}\rangle\big[((\lambda\mathsf{S})[\mathrm{id}]^e\,\mathsf{Z}[\mathrm{id}]^f)^c\cdot\mathrm{id}\big]\big)^a & \text{(App)}\\
& \overset{c}{\Vdash\!\!\to} (\mathsf{S}[\mathsf{Z}[\mathrm{id}]^f\cdot\mathrm{id}]^c\star\langle\mathsf{Z}:\mathsf{Z},\mathsf{S}:\underline{0}\rangle\big[\mathsf{S}[\mathsf{Z}[\mathrm{id}]^f\cdot\mathrm{id}]^c\cdot\mathrm{id}\big])^a & (\mathrm{B}_w)\\
& \overset{a}{\Vdash\!\!\to} \underline{0}\,[\mathsf{Z}[\mathrm{id}]^f\cdot\mathsf{S}[\mathsf{Z}[\mathrm{id}]^f\cdot\mathrm{id}]^c\cdot\mathrm{id}]^a & \text{(Select)}\\
& \overset{a}{\Vdash\!\!\to} \mathsf{Z}[\mathrm{id}]^f & \text{(FVarG)}
\end{array}$$

Les règles de stratégie de $\star$ sont communes à toutes les stratégies. Les adresses ne sont jamais permises à l'intérieur des constructions, en accord avec la tradition que les objets construits sont considérés comme des valeurs similaires aux abstractions, et donc aucune stratégie ne devrait pouvoir réduire *à l'intérieur*.

Définition 2.5.5 (Stratégie pour $\star$) *Les règles de stratégie pour $\star$ sont les suivantes :*

$$\frac{U\vdash A}{(U\star\langle\vec{C}:\vec{N}\rangle[s])^a\vdash A}\quad \textit{si } U \textit{ n'est pas une valeur} \tag{Lmw}$$

$$\frac{}{(U\star\langle\vec{C}:\vec{N}\rangle[s])^a\vdash\{a\}}\quad \textit{si } U \textit{ est une valeur} \tag{Smw}$$

Le tableau de la figure 2.7 peut être étendu avec des entrées pour ces règles, qui peuvent être permises sans restrictions pour toutes les stratégies.

La récursion est un peu plus compliquée à mettre en œuvre. Elle utilise un pointeur spécial $\bullet^a$ nommé *back-pointer* [31, 75] qui permet l'expression de graphes cycliques comme des $\lambda\sigma_w^a$-termes. Ainsi, un terme E d'adresse a peut contenir un sous-terme de la forme $\bullet^a$. L'utilisation d'un tel pointeur remet en question la définition des termes admissibles, car une même adresse peut avoir une occurrence sous son terme principal. Cette occurrence est alors un back-pointer. Informellement, un terme U est admissible si pour tout $a\in\mathrm{addr}(U)$ on a soit $U@a=\{\bullet^a\}$, soit $U@a=\{E^a\}$, E^a pouvant contenir des occurrences de $\bullet^a$, mais pas d'autres termes ayant la même adresse a. Nous considérons donc de tels termes sans donner de définition plus formelle. Dans le cas où $U@a=\{\bullet^a\}$, $\bullet^a$ est un pointeur sur lui même, ce qui dénote un sous graphe cyclique trivial.

Définition 2.5.6 (Récursion)

1. *Les termes sont enrichis par la nouvelle construction*

$$M, N ::= \dots \mid \mu M$$

où μ est appelé opérateur de récursion*, et tel que* $\mathrm{fi}(\mu M) = \{n-1 \mid n \in \mathrm{fi}(M) \text{ et } n > 0\}$*.*

2. *La règle de récursion est*

$$\mu M[s]^a \to M[\bullet^a \cdot s]^a \tag{Cycle}$$

$\bullet^a$ est une référence en arrière au terme d'adresse a qui l'englobe. Cette adresse étant susceptible de disparaître par l'utilisation de certaines règles, celles-ci doivent être redéfinies pour qu'il n'y ait pas une perte de l'information contenue dans le terme. Le back-pointer est alors remplacé par le terme auquel il faisait référence :

$$((\lambda M[s]^b)\, U)^a \to M[U \cdot (s\{\bullet^b := \lambda M[s]^b\})]^a \tag{$\mathrm{B}_w{}^\bullet$}$$

$$\underline{0}\,[E^b \cdot s]^a \to (E\{\bullet^a := \underline{0}\,[E^b \cdot s]^a\})^b \tag{FVarG$^\bullet$}$$

$$\underline{0}\,[E^b \cdot s]^a \to (E\{\bullet^b := E^b\})^a \tag{FVarE$^\bullet$}$$

$$(C_i[\vec{T} \cdot s]^b \star \langle \vec{C} : \vec{N} \rangle [t])^a \to N_i[\vec{T'} \cdot t]^a \quad \text{avec } \vec{T} \cdot s = T_1 \cdot \ldots \cdot T_{\mathsf{ar}(C_i)} \cdot s \text{ et } \vec{T'} = \vec{T}\{\bullet^b := C_i[\vec{T} \cdot s]^b\} \tag{Select$^\bullet$}$$

En remplaçant (B_w, FVarG, FVarE, Select) par ($\mathrm{B}_w{}^\bullet$, FVarG$^\bullet$, FVarE$^\bullet$, Select$^\bullet$) dans le précédent système, on obtient une réduction où la notion de terme admissible est préservée.

2.6 Conclusion

Ce chapitre nous a permis d'avoir un tour d'horizon rapide sur les calculs de substitution explicite. Nous y avons défini précisément le cadre de nos travaux dans ce domaine. Nous y avons décrit plusieurs calculs qui nous ont permis d'introduire un certain nombre de concepts et des problèmes à résoudre dans ce cadre. En particulier, nous avons donné les éléments qui permettent d'établir une classification des calculs de substitution explicite :

- En fonction de leurs propriétés : confluence, préservation de la forte normalisation, metaconfluence, simulation pas à pas et normalisation du système de substitution.
- En fonction des règles mises en œuvre : permutation, concaténation et composition.
- En fonction du degré de normalisation possible : forte ou faible.
- En fonction du traitement des renommages, explicites ou implicites.
- En fonction des extensions qu'ils mettent en œuvre.

Ces éléments de classification ne prétendent pas être exhaustifs. Cependant, ils montrent bien la diversité des possibilités qui s'offrent à l'utilisateur potentiel de ces divers calculs. Nous pensons à l'implanteur qui cherche à mettre en œuvre un algorithme de réduction dans le cadre d'un outil particulier. Nous espérons que ce chapitre peut être considéré comme un guide dans une telle situation.

Chapitre 3

Metaconfluence et PSN

Nous abordons dans ce chapitre le problème de la définition d'un λ-calcul avec substitution explicite qui ait à la fois la propriété de préservation de la forte normalisation (PSN) et la propriété de metaconfluence, ou confluence sur les metatermes. Nous décrivons deux approches : la première, détaillée à la section 3.1 est une approche naïve qui mènera très rapidement à un echec, mais qui orientera notre méthode de recherches ultérieures. La seconde, décrite section 3.2 s'appuie sur les intuitions très fortes que peut nous inspirer un calcul de substitution explicite avec noms, et nous permet d'obtenir quelques résultats intéressants. D'une part, nous ramenons le problème de PSN d'un calcul avec règle de permutation à la PSN du calcul avec permutations le plus simple que nous puissions imaginer, en accord avec notre intuition. D'autre part, nous définissons une notion de confluence légèrement plus faible que la metaconfluence, qui nous semble être un compromis raisonnable pour les implantations de calculs avec metavariables. Nous conclurons ce chapitre à la section 3.3.

3.1 Première tentative

Une première façon naïve d'essayer d'obtenir un système ayant les propriétés citées est d'étudier les paires critiques d'un système de substitution explicite, et d'introduire de nouvelles règles pour résoudre ces paires critiques (complétion). C'est ce que nous avons fait à partir de $\lambda\upsilon$, donnant ainsi lieu au calcul $\lambda\upsilon_c$ de la figure 3.1. Ce calcul est très proche de $\lambda\upsilon$, mais nous avons un peu modifié la syntaxe habituelle. Ainsi, le terme $a[b/]_n$ est équivalent au terme $a[\Uparrow^n (b/)]$ de $\lambda\upsilon$ où $\Uparrow^n (s)$ est une abbréviation pour $\underbrace{\Uparrow \dots \Uparrow}_{n}(s)$, et $\Uparrow^0 (s) = s$. De plus, les règles (rvar) et (rvarlift) ont été généralisées, donnant les règles (r) et (rlift). La variable $\underline{n}$ de $\lambda\upsilon$ sera alors codée dans $\lambda\upsilon_c$ par $\underline{0}\ \underbrace{[\uparrow]_0 \cdots [\uparrow]_0}_{n}$.

Le système $\lambda\upsilon_c$ est localement metaconfluent. Ceci se vérifie aisément par l'étude des paires critiques qui convergent toutes. Pour autant se pose le problème de la PSN, et dans un premier temps de la forte normalisation de υ_c.

Les règles qui pourraient menacer la forte normalisation du calcul sont (rlift) et (c). En effet, elles ont pour effet de faire permuter des substitutions et des renommages dans des directions opposées : ce que l'une a fait, l'autre pourrait le défaire ! Intéressons nous de plus près à ces deux règles en simplifiant le problème. Pour cela, décrivons un petit jeu, que nous appelons le *jeu des permutations* :

Syntaxe

$$\begin{array}{rll} a, b & ::= \lambda a \ \mid \ ab \ \mid \ \underline{0} \ \mid \ a[s]_n & \text{(Termes)} \\ s, t & ::= a/ \ \mid \uparrow & \text{(Substitutions)} \end{array}$$

Règles

$$\begin{array}{rcll} (\lambda a)b & \xrightarrow[\lambda\upsilon_c]{} & a[b/]_0 & \text{(b)} \\ (ab)[s]_n & \xrightarrow[\lambda\upsilon_c]{} & a[s]_n b[s]_n & \text{(app)} \\ (\lambda a)[s]_n & \xrightarrow[\lambda\upsilon_c]{} & \lambda a[s]_{n+1} & \text{(lambda)} \\ \underline{0}\,[a/]_0 & \xrightarrow[\lambda\upsilon_c]{} & a & \text{(fvar)} \\ a[\uparrow]_n[b/]_n & \xrightarrow[\lambda\upsilon_c]{} & a & \text{(r)} \\ \underline{0}\,[s]_{n+1} & \xrightarrow[\lambda\upsilon_c]{} & \underline{0} & \text{(fvarlift)} \\ a[\uparrow]_n[s]_{n+m+1} & \xrightarrow[\lambda\upsilon_c]{} & a[s]_{n+m}[\uparrow]_n & \text{(rlift)} \\ a[b/]_n[s]_{n+m} & \xrightarrow[\lambda\upsilon_c]{} & a[s]_{n+m+1}[b[s]_m/]_n & \text{(c)} \end{array}$$

FIG. 3.1 – Le calcul $\lambda\upsilon_c$

Jeu des permutations. Soit deux équipes d'effectifs quelconques, l'équipe grise et l'équipe blanche. Chaque joueur de chaque équipe choisit un *score* quelconque (un entier naturel) et les joueurs des deux équipes se placent en ligne, côte à côte, de manière quelconque également mais tous tournant la face du même côté. Ensuite, le jeu se déroule selon les règles suivantes, où chaque joueur ne peut pas être impliqué dans deux (ni plus) mouvements simultanément

- Un joueur blanc de score n peut échanger sa place avec son voisin de gauche (quelle que soit sa couleur) de score m si $m > n$. Alors, m est diminué d'un point.
- Un joueur gris de score n peut échanger sa place avec son voisin de gauche (quelle que soit sa couleur) de score m si $m \geq n$. Alors, m est incrémenté d'un point.

Le problème est : le jeu se termine-t-il, quelle que soit la configuration de départ et l'ordre des mouvements ?

En fait, il existe un exemple assez simple de non terminaison de ce jeu, trouvé par Pierre Lescanne [58] simultanément à la publication du problème dans le journal local du CRIN et INRIA-Lorraine : *Inforum*. Il est mis en évidence en seulement 5 mouvements sur la figure 3.2 : En effet, la configuration de l'étape 5 est similaire à la configuration initiale, aux acteurs près. Ce contre exemple peut être adapté au calcul υ_c :

Théorème 3.1.1 *Soit a, u_0, v_0 des termes quelconques et $u_{n+1} = v_n, v_{n+1} = u_n[\uparrow]_0[v_n/]_0$:*

$$a[\uparrow]_0[u_n/]_0[\uparrow]_0[v_n/]_0 \xrightarrow[\upsilon_c]{} \!\!\!\!\rightarrow a[\uparrow]_0[u_{n+1}/]_0[\uparrow]_0[v_{n+1}/]_0$$

Démonstration. Par une simple réduction en 4 pas :

$$
\begin{aligned}
a[\uparrow]_0\underline{[u_n/]_0[\uparrow]_0}[v_n/]_0 &\xrightarrow[v_c]{} a\underline{[\uparrow]_0[\uparrow]_1}\,[u_n[\uparrow]_0/]_0\,[v_n/]_0 && \text{(c)}\\
&\xrightarrow[v_c]{} a[\uparrow]_0[\uparrow]_0\underline{[u_n[\uparrow]_0/]_0\,[v_n/]_0} && \text{(rlift)}\\
&\xrightarrow[v_c]{} a[\uparrow]_0\underline{[\uparrow]_0[v_n/]_1}\,[u_n[\uparrow]_0[v_n/]_0/]_0 && \text{(c)}\\
&\xrightarrow[v_c]{} a[\uparrow]_0[v_n/]_0[\uparrow]_0\,[u_n[\uparrow]_0[v_n/]_0/]_0 && \text{(rlift)}\\
&= a[\uparrow]_0[u_{n+1}/]_0[\uparrow]_0[v_{n+1}/]_0
\end{aligned}
$$

□

Dans ce contre-exemple, on peut interpréter $[\uparrow]_n$ comme un joueur blanc de score n et $[a/]_n$ comme un joueur gris de score n pour retomber sur le jeu de la figure 3.2.

En fait, le même contre exemple peut également être adapté pour démontrer la non terminaison de τ, un calcul de substitution explicite dû à Riòs [73] (voir également une description dans [57]) dont nous retiendrons uniquement les 3 règles données sur la figure 3.3. La PSN de $\lambda\tau$ est restée un problème ouvert plusieurs années. En fait, τ lui même n'est pas fortement normalisant, comme le montre le théorème qui suit :

Théorème 3.1.2 *Soit u_0, v_0 des termes quelconques et $u_{n+1} = v_n, v_{n+1} = u_n[\uparrow \ \circ v_n/]$:*

$$\uparrow \ \circ (u_n/ \ \circ \ (\uparrow \ \circ v_n/)) \twoheadrightarrow_{\tau} \uparrow \ \circ (u_{n+1}/ \ \circ \ (\uparrow \ \circ v_{n+1}/))$$

Démonstration. Par une simple réduction en 3 pas.

$$
\begin{aligned}
\uparrow \ \circ (\underline{u_n/ \ \circ \ (\uparrow \ \circ v_n/)}) &\xrightarrow[\tau]{} \underline{\uparrow \ \circ (\Uparrow(\uparrow \ \circ v_n/) \ \circ \ u_n[\uparrow \ \circ v_n/]/)} && \text{(MapSl)}\\
&\xrightarrow[\tau]{} \underline{(\uparrow \ \circ v_n/) \ \circ \ (\uparrow \ \circ u_n[\uparrow \ \circ v_n/]/)} && \text{(ShiftLift2)}\\
&\xrightarrow[\tau]{} \uparrow \ \circ (\underline{v_n/ \ \circ \ (\uparrow \ \circ u_n[\uparrow \ \circ v_n/]/)}) && \text{(AssEnv)}\\
&= \uparrow \ \circ (\underline{u_{n+1}/ \ \circ \ (\uparrow \ \circ v_{n+1}/)})
\end{aligned}
$$

□

3.2 Deuxième tentative

Notre première tentative de définition d'un calcul PSN et metaconfluent s'est soldée par un échec, certes relatif car il a permis de contribuer à démontrer que $\lambda\tau$ n'avait pas les propriétés recherchées. La démarche mise en œuvre devait cependant être tentée –c'est probablement la même démarche qui avait d'ailleurs été suivie par Riòs pour définir $\lambda\tau$– mais elle s'avère un peu trop naïve. En fait, l'effet sur la PSN de l'ajout de nouvelles règles est très difficile à prévoir, et la PSN (comme la terminaison) est une propriété fragile. Un autre exemple est celui du calcul λs_e [41]. λs_e est un calcul avec substitution explicite pour lequel existait une conjecture selon laquelle metaconfluence et PSN étaient satisfaites, jusqu'à ce qu'il ait été prouvé très récemment par Guillaume grâce à un contre exemple que λs_e n'est pas PSN [39]. Il faut aussi parler de $\lambda\sigma_{\Uparrow}$ qui est metaconfluent mais n'est pas PSN, comme l'a démontré Melliès [63]. Certes, $\lambda\sigma_{\Uparrow}$ n'a pas été conçu dans le but de préserver la forte normalisation du λ-calcul, mais son exemple, comme celui de λs_e, contribue à démontrer que PSN et metaconfluence sont des propriétés difficiles à combiner.

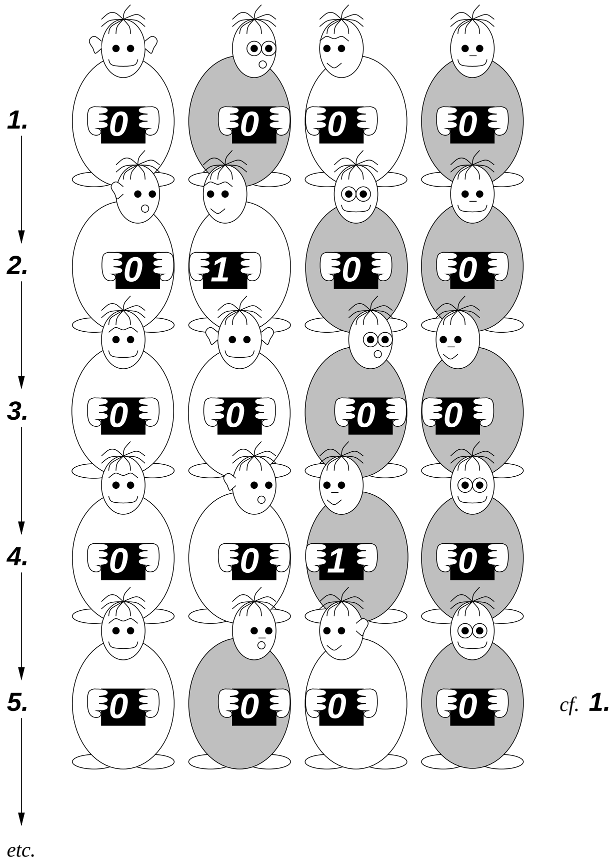


FIG. 3.2 – Non-terminaison du jeu des permutations

$$
\begin{array}{rcll}
a/ \circ s & \underset{\lambda\tau}{\rightarrow} & \Uparrow(s) \circ a[s]/ & \text{(MapSl)} \\
\uparrow \circ (\Uparrow(s) \circ t) & \underset{\lambda\tau}{\rightarrow} & s \circ (\uparrow \circ t) & \text{(ShiftLift2)} \\
(s \circ t) \circ u & \underset{\lambda\tau}{\rightarrow} & s \circ (t \circ u) & \text{(AssEnv)}
\end{array}
$$

FIG. 3.3 – Trois règles du calcul τ

Notre deuxième tentative de définition d'un λ-calcul avec substitution explicite PSN et metaconfluent s'inscrit dans une démarche différente de celle mise en œuvre précédemment. Elle part du principe déjà cité dans cette thèse qu'il est beaucoup plus facile de raisonner sur des termes avec noms de variables plutôt que des termes avec indices. En effet, dans ce cas les problèmes liés aux renommages de variables sont résolus implicitement. Autrement dit, notre démarche va être dans un premier temps de tenter de définir un λ-calcul avec noms PSN et metaconfluent. Nous espérons ainsi comprendre un peu mieux les mécanismes qui entrent en jeu dans la réduction et qui sont liés uniquement à la substitution, non au renommage. Le travail présenté dans cette section est le fruit d'une collaboration avec Kristoffer Rose.

Considérons donc à présent le calcul λx, présenté au chapitre 2[1], et souvenons nous de la paire critique non résolue $\big\langle M\langle x := N\rangle\langle y := P\rangle, M\langle y := P\rangle\langle x := N\langle y := P\rangle\rangle\big\rangle$ de la figure 2.2. Une façon de retrouver la convergence serait d'introduire la règle suivante[2] :

$$M\langle x := N\rangle\langle y := P\rangle \to M\langle y := P\rangle\big\langle x := N\langle y := P\rangle\big\rangle \qquad \text{(c)}$$

Cependant, cette règle introduit deux nouvelles sources de réduction infinie :

1. une première réduction infinie, triviale :

$$\begin{aligned} M\langle x := N\rangle\langle y := P\rangle &\to M\langle y := P\rangle\big\langle x := N\langle y := P\rangle\big\rangle && \text{(c)}\\ &\to M\big\langle x := N\langle y := P\rangle\big\rangle\big\langle y := P\langle x := N\langle y := P\rangle\rangle\big\rangle && \text{(c)}\\ &\to M\big\langle y := P\langle x := N\langle y := P\rangle\rangle\big\rangle\\ &\qquad\qquad \big\langle x := N\langle y := P\rangle\langle y := P\langle x := N\langle y := P\rangle\rangle\rangle\big\rangle && \text{(c)}\\ &\to \dots \end{aligned}$$

2. une deuxième réduction infinie plus subtile, qui a été mise en évidence dans le cas de $\lambda\sigma_{\Uparrow}$ par Melliès [63], et plus récemment dans le cas de λs_e par Guillaume [39] :

$$\begin{aligned} ((\lambda x.M)N)\langle y := (\lambda x.M)N\rangle &\twoheadrightarrow (\lambda x.M\langle y := (\lambda x.M)N\rangle)(N\langle y := (\lambda x.M)N\rangle) && \text{(xap+xab)}\\ &\to M\langle y := (\lambda x.M)N\rangle\big\langle x := N\langle y := (\lambda x.M)N\rangle\big\rangle && \text{(b)}\\ &\to M\ \langle x := N\langle y := (\lambda x.M)N\rangle\rangle\\ &\qquad\qquad \Big\langle y := \underline{((\lambda x.M)N)\langle x := N\langle y := (\lambda x.M)N\rangle\rangle}\Big\rangle && \text{(c)}\\ &\to \dots \end{aligned}$$

qui crée un motif récurrent et par conséquent une réduction infinie.

Cette règle est donc trop permissive ! Nous allons donc nous intéresser aux contraintes sur la règle (c) qui vont nous permettre d'assurer la terminaison tout en permettant de résoudre toutes les paires critiques. Avant toutes choses, nous introduisons quelques conventions de notation.

Mémoriser les permutations

Notre première tentative de définition de calcul metaconfluent et PSN a suivi la démarche suivante : *Définir un calcul metaconfluent puis prouver PSN* qui s'est soldée par un échec. Ici, nous

[1]Momentanément et pour simplifier les exemples, la convention de notation des termes de CRS n'est pas respectée dans ces exemples.

[2]Nous avions nommé cette règle (c) pour *composition*, mais cette règle est en fait une règle de *permutation*, comme nous l'avons compris plus tard.

suivrons la démarche inverse. Nous tenterons de définir un calcul avec permutation, et la contrainte la plus faible possible sur la règle de permutation, susceptible cependant de nous permettre d'obtenir PSN, selon notre intuition. Nous étudierons ensuite les contraintes supplémentaires à adopter pour obtenir la metaconfluence.

L'observation clé, tirée des contre-exemples, est la suivante : *Les réductions infinies mettent en jeu la propagation d'une substitution sous un lieur, puis la permutation de cette substitution avec une substitution de cette variable liée.* Notre solution à ce problème est simplement d'enrichir chaque substitution avec une liste des lieurs *sous lesquels* elle est passée, puis d'interdire la permutation quand la substitution interne est déjà passée sous le lieur correspondant à la substitution externe.

$$(\lambda x.M[x])N \to M[x]\langle x := N\rangle_\varnothing \tag{b}$$
$$x\langle x := N\rangle_S \to N \tag{xv}$$
$$M\langle x := N\rangle_S \to M \tag{xgc}$$
$$(\lambda x.M[x,y])\langle y := N\rangle_S \to \lambda x.M[x,y]\langle y := N\rangle_{S+x} \tag{xab}$$
$$(M_1[y]\, M_2[y])\langle y := N\rangle_S \to M_1[y]\langle y := N\rangle_S\; M_2[y]\langle y := N\rangle_S \tag{xap}$$
$$M[x,y]\langle x := N[y]\rangle_S\langle y := P\rangle_T \to M[x,y]\langle y := P\rangle_{T+x}\,\langle x := N[y]\langle y := P\rangle_T\rangle_S \tag{c1}$$

FIG. 3.4 – Le calcul λxc1 (ESCRS)

Définition 3.2.1 (λxc-termes et λxc1)

1. *Les* λxc*-termes, notés* $M, N, P, \ldots$ *sont les termes des ESCRS définis inductivement par*

$$M ::= x \mid \lambda x.M \mid MN \mid M\langle x := N\rangle_S$$

où $x, y, z, \ldots$ *sont les éléments d'un ensemble infini de variables, et* $S, T, \ldots$ *sont des ensembles (finis) de variables, appelés* ensembles bloquants. $+$ *est l'opérateur d'adjonction disjointe sur les ensembles bloquants, c'est-à-dire que dans* $S + x$*,* x *est supposé ne pas appartenir à* S*.*

2. *La réduction* λxc1 *est la fermeture contextuelle de l'union des règles de réécriture (conditionnelles) de la figure 3.4. Nous utilisons* $\xrightarrow[\mathrm{x}]{}$ *pour le sous-calcul permettant seulement les règles* (xv, xgc, xab, xap)*, et* $\xrightarrow[\mathrm{b}]{}$, $\xrightarrow[\mathrm{c1}]{}$, $\xrightarrow[\mathrm{xc1}]{}$, $\xrightarrow[\mathrm{bxc1}]{}$*, etc. pour les relations définies selon les règles du chapitre 2.*

   Remarque. Les conventions des (ES)CRS prennent en compte l'*absence* de variables libres. C'est pourquoi (xv) et (xgc) ne se chevauchent pas. Par exemple, le terme $v\langle v := \lambda w.w\rangle_\varnothing$ ne peut être réduit que par (xv) car v est libre dans v. Cela s'applique également aux ensembles bloquants : dans $\lambda y.M[x,y]\langle x := N[y]\rangle_S$ ou dans $M[x,y]\langle x := N[y]\rangle_S\langle y := P\rangle_T$, il est implicite que $y \notin S$, sinon nous aurions écrit $\ldots_{S[y]}$. Ceci illustre le fait que toutes les variables mentionnées sont soit explicitement liées, soit supposées liées dans le contexte. Par exemple, dans le metaterme $M[x]\langle x := N\rangle_{S+y}$ il est implicite que y n'est pas libre dans M. □

Le calcul xc1 est fortement normalisant, ce que nous démontrons grâce à un ordre récursif sur les chemins.

Proposition 3.2.2 $\xrightarrow[\mathrm{xc1}]{}$ *est fortement normalisant.*

Démonstration. La preuve est basée sur l'ordre récursif sur les chemins induit par l'ordre bien fondé sur les symboles (pour tout n entier)

$$\langle\rangle^{n+1} \;\triangleright\; \langle\rangle^{n} \;\triangleright\; \cdot \;\triangleright\; \lambda, *$$

où les λxc-termes sont projetés sur les termes comparables par la traduction $[\![M]\!]$, définie inductivement par

$$\begin{aligned} [\![x]\!] &= * \\ [\![\lambda x.M[x]]\!] &= \lambda[\![M[x]]\!] \\ [\![MN]\!] &= [\![M]\!] \cdot [\![N]\!] \\ [\![M[x]\langle x := N\rangle_S]\!] &= [\![M[x]]\!]\langle[\![N]\!]\rangle^c \end{aligned}$$

où c est le plus grand nombre de permutations non bloquées dans le terme :

$$c = \max\{n \;\mid\; \exists P : M[x]\langle x := N\rangle_S \xrightarrow[\mathrm{c1}]{n} P\}$$

$\square$

Cependant, et malheureusement, la PSN de λxc1 reste un problème ouvert.

Problème 3.2.3 λxc1 *est-il PSN ?*

La difficulté pour prouver la PSN d'un calcul comme λxc1 provient de la règle de permutation (c1). Par exemple, le terme $M = (\lambda x.(\lambda y.x)(\omega\, x))\,\omega$, avec $\omega = \lambda z.zz$ n'est pas β-fortement normalisable, car il peut se réduire de la façon suivante :

$$(\lambda x.(\lambda y.x)(\omega\, x))\,\omega \xrightarrow[\beta]{} (\lambda y.\omega)(\omega\,\omega) \xrightarrow[\beta]{\infty} \ldots$$

Il n'est par conséquent pas non plus bx-fortement normalisable. Cependant, M a une forme normale ω. On peut réduire M par λx pour atteindre cette forme normale, en commençant comme suit :

$$\begin{aligned} (\lambda x.(\lambda y.x)(\omega\, x))\,\omega &\xrightarrow[\mathrm{bx}]{} ((\lambda y.x)(\omega\, x))\langle x := \omega\rangle \\ &\xrightarrow[\mathrm{bx}]{} x\langle y := \omega\, x\rangle\langle x := \omega\rangle \end{aligned}$$

et tous les sous-termes et sous-termes des réduits de $x\langle y := \omega\, x\rangle\langle x := \omega\rangle$ sont fortement normalisables, car un λx-terme est bx-fortement normalisable si et seulement si sa x-forme normale est β-fortement normalisable, et chacun des termes à l'intérieur des crochets de substitutions sont bx-fortement normalisables (ce qui est le cas ici pour $\omega\, x$ et ω). Cette belle propriété est malheureusement perdue avec une règle de permutation comme (c1), qui permet des interactions entre les substitutions. En effet, $x\langle y := \omega\, x\rangle_\varnothing\langle x := \omega\rangle_\varnothing \xrightarrow[\mathrm{c1}]{} x\langle x := \omega\rangle_y\langle y := (\omega\, x)\langle x := \omega\rangle_\varnothing\rangle_\varnothing$, un terme qui n'est pas bx-fortement normalisable, puisque nous avons retrouvé un des chemins de réduction du terme initial qui pouvait mener à une réduction infinie. Comme on le voit, il est très difficile de définir un invariant et une hypothèse d'induction qui pourront nous aider à prouver la PSN pour le calcul λxc1.

Metaconfluence

xc1 n'est pas (localement) metaconfluent : il y a des paires critiques non résolues impliquant la règle (c1), en particulier une avec (xab) présentée sur la figure 3.5. Cependant, cette paire critique semble être facilement résoluble : les deux branches de la paire critique diffèrent seulement dans

$$\begin{tikzcd}
(\lambda x.M[x,y,z])\langle y := N[z]\rangle_S\langle z := P\rangle_T \arrow[d, "\mathrm{xab}^2"'] \arrow[ddr, bend left, "\mathrm{c1}"] & \\
\lambda x.M[x,y,z]\langle y := N[z]\rangle_{S+x}\langle z := P\rangle_{T+x} \arrow[dd, "\mathrm{c1}"'] & \\
 & (\lambda x.M[x,y,z])\langle z := P\rangle_{T+y}\langle y := N[z]\langle z := P\rangle_T\rangle_S \arrow[dd, "\mathrm{xab}^2"] \\
\lambda x.M[x,y,z]\langle z := P\rangle_{T+x+y}\langle y := N[z]\langle z := P\rangle_{T+x}\rangle_{S+x} \arrow[dr, dotted, no head, "?"] & \\
 & \lambda x.M[x,y,z]\langle z := P\rangle_{T+x+y}\langle y := N[z]\langle z := P\rangle_T\rangle_{S+x}
\end{tikzcd}$$

FIG. 3.5 – Paire critique non résolue entre (xab) et (c1)

l'ensemble bloquant d'une des substitutions de z. De plus, on peut noter que la variable x n'a pas d'occurrence libre dans $N[z]\langle z := P\rangle$. Nous allons donc exploiter ce fait pour tenter de résoudre la metaconfluence.

La clé est une règle d'*intersection* (i) qui résoud la paire critique introduite par la règle de permutation, sans introduire de non-terminaison. L'observation est la suivante : *Une permutation entre deux substitutions est inutile si la variable de la substitution n'a pas d'occurrence libre dans le corps de la substitution.* C'est la conséquence de la présence de la règle (xgc) dans le système : la confluence de substitutions déchet (voir la définition 2.1.3) n'est jamais un problème puisqu'elles peuvent toujours être éliminées. Cependant, comme nous travaillons avec des metatermes, nous devons introduire une notion de variables libres légèrement plus faible. En particulier, il est important de disposer d'une notion de variables libres des metavariables. En effet, une metavariable ne contient pas de variables libres, mais peut être instanciée par un terme qui contient des variables libres. Nous exprimons ce fait en définissant la notion de *variable potentiellement libre*.

Convention 3.2.4 *Pour distinguer l'utilisation normale des metavariables (où la metavariable dénote tout ce en quoi elle peut être instanciée), nous soulignons les metavariables quand nous voulons indiquer que le metaterme a une occurrence d'une metavariable donnée. Par conséquent,* $\lambda x.\underline{M}[x]$ *signifie que le metaterme est "*$\lambda x.M[x]$*" alors que* $\lambda x.M[x]$ *peut être instancié en n'importe quelle* λ*-abstraction (dont elle même, bien sûr). On appelle les metavariables de la forme* $\underline{M}[x_1;\ldots;x_n]$ *les* metavariables syntaxiques d'arité n. *La liste* $[x_1;\ldots;x_n]$ *est attachée à la metavariable syntaxique, et constante. Il s'agit des variables liées dans le contexte qui peuvent être libres dans toute instance de* $\underline{M}[x_1;\ldots;x_n]$.

Définition 3.2.5 (Variables potentiellement libres) *Soit* P *un metaterme. L'ensemble des* variables potentiellement libres *de* P*, noté* $\mathrm{pfv}(P)$*, est une généralisation des variables libres aux* *metatermes. Il est défini comme suit :*

$$\begin{aligned}
\mathrm{pfv}(MN) &= \mathrm{pfv}(M) \cup \mathrm{pfv}(N) \\
\mathrm{pfv}(\lambda x.M[x]) &= \mathrm{pfv}(M[x]) \setminus \{x\} \\
\mathrm{pfv}(x) &= \{x\} \\
\mathrm{pfv}(M[x]\langle x := N\rangle_S) &= (\mathrm{pfv}(M[x]) \setminus \{x\}) \cup \mathrm{pfv}(N) \\
\mathrm{pfv}(\,\underline{M}\,[x_1, \ldots, x_n]) &= \{x_1, \ldots, x_n\}
\end{aligned}$$

Remarque.

1. La définition de metavariable syntaxique implique qu'à toute metavariable de ce type est associée une liste de variables *potentiellement libres*. Cette contrainte est très raisonnable, voire même essentielle ! Considérons le terme $\lambda z.(\lambda x.\lambda y.x)\ \underline{M}\,[z]$ qui se réduit en $\lambda z.\lambda y.\,\underline{M}\,[z]$. Il est clair que y, ou $\lambda w.y$, ne sont pas des instances possibles de $\underline{M}\,[z]$ car y ne fait pas partie de la liste des variables potentiellement libres de $\underline{M}\,[z]$. Si $\underline{M}\,[z]$ pouvait être instancié par y, alors on aurait $\lambda z.(\lambda x.\lambda y.x)\,y \to \lambda z.\lambda y.y$, ce qui manifestement est faux, en raison d'un conflit de noms. Bien entendu on pourrait interdire une telle instanciation, car la convention de Barendregt n'est pas respectée sur l'instance de la réduction. Il n'est cependant pas commode de contraindre l'instanciation d'une metavariable dans un metaterme au vu de la réduction. La notion de variable potentiellement libre est pour cela très pratique, car il suffit d'observer le dernier terme de la réduction pour déterminer si une instantiation est possible. Par exemple, z, $\lambda x.x$, ou $\lambda t.\,\underline{N}\,[z,t]$ sont des instances possibles de $\underline{M}\,[z]$, car leurs variables potentiellement libres sont incluses dans $\{z\}$.

2. L'*élégance* des metavariables semble être perdue dans nos calculs, en raison de la distinction entre metavariable (variable du metalangage) et metavariable syntaxique (variable d'ordre supérieur). Cette distinction est cependant essentielle pour pouvoir définir l'ensemble des variables potentiellement libres. D'autre part, dans les implantations, ces deux types de variables sont à des niveaux différents : les metavariables au niveau de la description des termes (par exemple pour l'analyse syntaxique) et les metavariables syntaxiques au niveau du calcul lui même (comme constructeurs de termes). Cette distinction en entraînera bien sûr une autre : un calcul peut être confluent sur les metatermes, ou seulement confluent sur les termes avec metavariables syntaxiques. Notre étude nous restreint à ce deuxième cas.

□

Dans le but de résoudre la paire critique de la figure 3.5, nous aimerions effectuer la réduction suivante :

$$M[x]\langle x := N\rangle_{S+y} \to M[x]\langle x := N\rangle_S \qquad \text{(i)}$$

(avec la condition implicite que y n'est libre ni dans M ni dans N.) Cependant, l'introduction de cette règle est dangereuse : tout d'abord, nous introduisons des règles qui construisent un ensemble (xab, c1), puis une règle qui le détruit partiellement (i). Le risque est d'introduire de la non terminaison. En effet, prenons par exemple t le metaterme $z\langle x := M[y]\rangle_\varnothing\langle y := N\rangle_\varnothing$. Il peut être réduit infiniment comme suit :

$$
\begin{aligned}
t &\to z\langle y := N\rangle_{\{x\}}\langle x := M[y]\langle y := N\rangle_\varnothing\rangle_\varnothing && \text{(c1)}\\
&\to z\langle y := N\rangle_\varnothing\langle x := M[y]\langle y := N\rangle_\varnothing\rangle_\varnothing && \text{(i)}\\
&\to z\langle x := M[y]\langle y := N\rangle_\varnothing\rangle_{\{y\}}\langle y := \ldots\rangle_\varnothing && \text{(c1)}\\
&\to z\langle x := \ldots\rangle_\varnothing\langle y := \ldots\rangle_\varnothing && \text{(i)}\\
&\to z\langle y := \ldots\rangle_{\{x\}}\langle x := \ldots\rangle_\varnothing && \text{(c1)}\\
&\to \ldots
\end{aligned}
$$

Par conséquent, nous proposons d'éliminer cette possibilité avec une condition plus forte sur (c1) (donnant lieu à la règle (c)) : $M[x,y]\langle x := N[y]\rangle_S\langle y := P\rangle_T$ est un c-redex si (1) la substitution de y n'est pas bloquée, c'est-à-dire $y \notin S$ (comme précédemment), et (2) la variable x a une occurrence (potentielle) quelque part dans le corps du terme dans lequel elle est propagée, c'est-à-dire $y \in \mathrm{pfv}(M[x,y]\langle x := N[y]\rangle_S)$.

Définition 3.2.6 (λxci) *Le calcul* λxci *est obtenu en changeant la définition 3.2.1 pour utiliser comme réduction la fermeture contextuelle de l'union des règles de réécriture conditionnelle présentées sur la figure 3.6.*

> *Remarque.* Les calculs λxc1 et λxci utilisent des conditions qui ne sont pas tout à fait locales : dans le cas de λxc1, il s'agit de la règle (xgc) (également présente dans certaines versions du calcul λx) qui suppose de calculer l'ensemble des variables libres du terme auquel la substitution est appliquée ; dans le cas de λxci, il s'agit également des règles (c, i). □

$$
\begin{aligned}
(\lambda x.M[x])N &\to M[x]\langle x := N\rangle_\varnothing && \text{(b)}\\
x\langle x := N\rangle_S &\to N && \text{(xv)}\\
M\langle x := N\rangle_S &\to M && \text{(xgc)}\\
(\lambda x.M[x,y])\langle y := N\rangle_S &\to \lambda x.M[x,y]\langle y := N\rangle_{S+x} && \text{(xab)}\\
(M_1[y]\, M_2[y])\langle y := N\rangle_S &\to M_1[y]\langle y := N\rangle_S\ M_2[y]\langle y := N\rangle_S && \text{(xap)}\\
M[x,y]\langle x := N[y]\rangle_S\langle y := P\rangle_T &\to M[x,y]\langle y := P\rangle_{T+x}\,\langle x := N[y]\langle y := P\rangle_T\rangle_S &&\\
&\qquad \text{si } y \in \mathrm{pfv}(M[x,y]\langle x := N[y]\rangle_S) && \text{(c)}\\
M[x]\langle x := N\rangle_{S+y} &\to M[x]\langle x := N\rangle_S && \text{(i)}
\end{aligned}
$$

FIG. 3.6 – Le calcul λxci

Proposition 3.2.7 *Si* $\xrightarrow[\mathrm{bxc1}]{}$ *est PSN alors* $\xrightarrow[\mathrm{bxci}]{}$ *est PSN.*

Démonstration. L'argument principal de la démonstration est le lemme de Bashmair-Dershowitz (1.1.11). On considère les relations $\xrightarrow[\mathrm{i}]{}$ et $\xrightarrow[\mathrm{bxc}]{}$. On remarque d'une part que $\xrightarrow[\mathrm{i}]{}$ est fortement normalisante, et qu'elle BD-commute sur $\xrightarrow[\mathrm{bxc}]{}$ car elle n'élimine ni n'introduit aucun bxc-redex, et elle maintient la structure générale du terme. D'autre part, $\xrightarrow[\mathrm{bxc}]{} \subseteq \xrightarrow[\mathrm{bxc1}]{}$, donc $\mathrm{SN}_{\mathrm{bxc1}} \subseteq \mathrm{SN}_{\mathrm{bxc}}$. D'après le lemme de Bashmair-Dershowitz, $\mathrm{SN}_{\mathrm{bxc}} \subseteq \mathrm{SN}_{\mathrm{bxci}}$. Donc si $\mathrm{SN}_\beta \subseteq \mathrm{SN}_{\mathrm{bxc1}}$ (c'est-à-dire $\xrightarrow[\mathrm{bxc1}]{}$ est PSN), alors $\mathrm{SN}_\beta \subseteq \mathrm{SN}_{\mathrm{bxci}}$ (c'est-à-dire $\xrightarrow[\mathrm{bxci}]{}$ est PSN). □

Corollaire 3.2.8 $\xrightarrow[\text{xci}]{}$ *est fortement normalisante.*

Démonstration. La preuve de la proposition 3.2.7 reste valide si on omet la règle (b) : on utilise le lemme de Bashmair-Dershowitz avec $\xrightarrow[\text{i}]{}$ et $\xrightarrow[\text{xc}]{}$, et on utilise les faits $\xrightarrow[\text{xc}]{} \subseteq \xrightarrow[\text{xc1}]{}$ et $\xrightarrow[\text{xc1}]{}$ fortement normalisante. □

Proposition 3.2.9 $\xrightarrow[\text{xci}]{}$ *est confluente sur les termes avec metavariables syntaxiques.*

Démonstration. La confluence faible est vérifiée en examinant chacune des 14 paires critiques, et en montrant qu'elles ont un réduit commun. Cette preuve nécessite quelques lemmes structurels qui ne sont pas donnés ici, la preuve complète est présentée en annexe A. La confluence découle du corollaire 3.2.8 par le lemme de Newman (lemme 1.1.6). □

Proposition 3.2.10 $\xrightarrow[\text{bxci}]{}$ *est confluente sur les termes avec metavariables syntaxiques.*

Démonstration. Nous utilisons la même technique que dans [68], présentée également dans [69], qui utilise le lemme de Yokouchi-Hikita (lemme 1.1.8). Pour cela, nous définissons une relation $\xrightarrow[\text{b}\|]{}$, la b-contraction parallèle, et nous montrons qu'elle satisfait les propriétés requises pour l'application du lemme de Yokouchi-Hikita. La preuve complète est présentée en annexe A. □

Proposition 3.2.11 (Simulation pas à pas) $\xrightarrow[\text{bxci}]{*} \supseteq \xrightarrow[\beta]{}$.

Démonstration. $\xrightarrow[\text{bxci}]{*} \supseteq \xrightarrow[\text{bx}]{*}$ car toutes les règles de λx sont des règles de λxci, aux ensembles bloquants près, mais ces ensembles n'ont aucune influence sur les réductions par les règles de λx. D'autre part, $\xrightarrow[\text{bx}]{*} \supseteq \xrightarrow[\beta]{}$ a été démontré dans [75] (proposition 3.1.15). □

Vers un calcul avec renommages explicites

Dans cette section, nous abordons le problème de la définition d'un équivalent avec renommages explicites de λxci. Nous proposons λx_Bci, un calcul avec des indices de de Bruijn. Un résultat intéressant sur ce calcul est la préservation de la forte normalisation de λxci, qui peut se démontrer de manière très simple, grâce aux techniques de nommage explicite de Rose [75].

Définition 3.2.12 (λx_Bc-termes, λx_Bci)

1. *Les* λx_Bc-termes, *notés* $a, b, c, \ldots$ *sont définis inductivement par*

$$a ::= \underline{0} \ \mid\ \lambda a \ \mid\ ab \ \mid\ a \uparrow_n \ \mid\ a\langle \underline{n} := b\rangle$$

   où $n, m, p, \ldots$ *sont des entiers naturels quelconques.*

2. *Les* termes purs *sont caractérisés par le sous ensemble des* λx_B*c-termes défini par* $a ::= v \mid \lambda a \mid ab$ *où* v *représente une* variable *définie par* $v ::= \underline{0} \mid v \uparrow_0$. *Le* $n^{\text{ième}}$ *indice de de Bruijn (habituellement noté* $\underline{n-1}$*) est représenté par* $\underline{0}\,\underbrace{\uparrow_0 \cdots \uparrow_0}_{n-1}$.

3. *Les metavariables* $a, b, c, \ldots$ *peuvent également être notées* $a_n, b_n, c_n, \ldots$. *Ecrire* a_n *signifie que les* indices potentiellement libres *(voir la définition 3.2.13) de tout metaterme instanciant* a *sont strictement inférieurs à* n. *Par exemple,* a_0 *est une metavariable qui peut être instanciée par n'importe quel metaterme clos. Les metavariables syntaxiques sont toujours notées* $\underline{a}_n$, $\underline{b}_n, \ldots$, *où* n *est donné et constant.*

$$
\begin{aligned}
(\lambda a)\, b &\to a\langle\, \underline{0} \,:= b\rangle && \text{(b)}\\
a \uparrow_n \langle\, \underline{n} \,:= b\rangle &\to a && \text{(xgc1)}\\
a_n\langle\, \underline{n+m} \,:= b\rangle &\to a_n && \text{(xgc2)}\\
\underline{0}\,\langle\, \underline{0} \,:= a\rangle &\to a && \text{(xv)}\\
(\lambda a)\langle\, \underline{n} \,:= b\rangle &\to \lambda a\langle\, \underline{n+1} \,:= b\rangle && \text{(xab)}\\
(a\, b)\langle\, \underline{n} \,:= c\rangle &\to a\langle\, \underline{n} \,:= c\rangle\, b\langle\, \underline{n} \,:= c\rangle && \text{(xap)}\\
a\langle\, \underline{n} \,:= b\rangle\langle\, \underline{n+p} \,:= c\rangle &\to a\langle\, \underline{n+p+1} \,:= c\rangle\langle\, \underline{n} \,:= b\langle\, \underline{p} \,:= c\rangle\rangle &&\\
&\quad \text{si } n+p \in \mathrm{pfi}(a\langle\, \underline{n} \,:= b\rangle) && \text{(c)}\\
a \uparrow_n \langle\, \underline{n+m+1} \,:= b\rangle &\to a\langle\, \underline{n+m} \,:= b\rangle \uparrow_n && \text{(i)}\\
a \uparrow_n \uparrow_{n+p+1} &\to a \uparrow_{n+p} \uparrow_n && \text{(rc)}\\
a_n \uparrow_{n+m} &\to a_n && \text{(rgc)}\\
(\lambda a) \uparrow_n &\to \lambda a \uparrow_{n+1} && \text{(rab)}\\
(a\, b) \uparrow_n &\to a \uparrow_n\ b \uparrow_n && \text{(rap)}\\
a\langle\, \underline{n} \,:= b\rangle \uparrow_{n+p} &\to a \uparrow_{n+p+1} \langle\, \underline{n} \,:= b \uparrow_p\rangle && \text{(rx)}
\end{aligned}
$$

FIG. 3.7 – Le calcul $\lambda x_B ci$

4. *La réduction* $\lambda x_B ci$ *est la fermeture contextuelle des règles de réécriture conditionnelle présentées sur la figure 3.7. Nous utilisons* $\xrightarrow[x_B]{}$ *pour le sous-calcul permettant seulement l'utilisation des règles* (xgc1, xgc2, xv, xab, xap) *et* $\xrightarrow[\mathrm{r}]{}$ *pour le sous-calcul permettant seulement* (rc, rgc, rab, rap, rx).

Définition 3.2.13 (Indices potentiellement libres) *Soit a un metaterme. L'ensemble des* indices potentiellement libres *de a, noté* pfi(a) *est un ensemble d'entiers naturels défini par :*

$$
\begin{aligned}
\mathrm{pfi}(\,\underline{0}\,) &= \{0\}\\
\mathrm{pfi}(\lambda a) &= \{m-1 \mid m > 0 \in \mathrm{pfi}(a)\}\\
\mathrm{pfi}(ab) &= \mathrm{pfi}(a) \cup \mathrm{pfi}(b)\\
\mathrm{pfi}(a\langle\, \underline{n} \,:= b\rangle) &= \{m \mid m < n \in \mathrm{pfi}(a)\} \cup \{m-1 \mid m > n \in \mathrm{pfi}(a)\} \cup \{m+n \mid m \in \mathrm{pfi}(b)\}\\
\mathrm{pfi}(a \uparrow_n) &= \{m \mid m < n \in \mathrm{pfi}(a)\} \cup \{m+1 \mid m \geq n \in \mathrm{pfi}(a)\}\\
\mathrm{pfi}(\,\underline{a}_{\,n}) &= \{m \mid m < n\}
\end{aligned}
$$

Proposition 3.2.14 $\xrightarrow[\mathrm{r}]{}$ *est fortement normalisant.*

Démonstration. Soit $\succ$ l'ordre récursif sur les chemins induit par l'ordre bien fondé sur les symboles $\rhd$ défini par

$$\uparrow_n\ ,\ \uparrow_m\ \rhd\ \cdot\ ,\ \langle\, \underline{n} \,:=\rangle\ ,\ \langle\, \underline{m} \,:=\rangle\ ,\ \lambda\ ,\ \underline{0}$$

Il est clair que $a \xrightarrow[\mathrm{r}]{} b \Rightarrow a \succ b$ sauf pour la règle (rc) où $a = b$. Pour cette règle cependant, la somme des indices portés par les *shifts* décroît strictement dans $\mathbb{N}$. $\square$

Définition 3.2.15 (Traduction $\lambda x_B c$-λxc) *Nous définissons un système d'inférence permettant de traduire les $\lambda x_B c$-termes vers les λxc-termes, de la même manière que dans [75]. Si ρ est la liste $[x_1; \ldots ; x_{n+m}]$ alors $\rho\backslash n = [x_1; \ldots ; x_n]$ et $\rho/n = [x_{n+1}; \ldots ; x_{n+m}]$. De plus, une liste peut être considérée comme un ensemble quand cela est clair dans le contexte*[3].

$$x \cdot \rho \vdash \underline{0} \leadsto x \qquad \frac{x \cdot \rho \vdash a \leadsto M}{\rho \vdash \lambda a \leadsto \lambda x \cdot M} \ (x \notin \rho) \qquad \frac{\rho \vdash a \leadsto M \quad \rho \vdash b \leadsto N}{\rho \vdash a\, b \leadsto M\, N}$$

$$\frac{\rho\backslash n \cdot x \cdot \rho/n \vdash a \leadsto M \quad \rho/n \vdash b \leadsto N}{\rho \vdash a\langle \underline{n} := b\rangle \leadsto M\langle x := N\rangle_{\rho\backslash n}} \qquad \frac{\rho\backslash n \cdot \rho/(n+1) \vdash a \leadsto M}{\rho \vdash a \uparrow_n \leadsto M}$$

Lemme 3.2.16 *Si $a \xrightarrow[\mathrm{bx}_B\mathrm{ci}]{} b$ alors $\forall \rho : \rho \vdash a \leadsto M$ et $\rho \vdash b \leadsto N \Rightarrow M \xrightarrow[\mathrm{bxci}]{+} N$.*

Démonstration. Voir l'annexe B. □

Lemme 3.2.17 *Si $a \xrightarrow[\mathrm{r}]{} b$ alors $\forall \rho : \rho \vdash a \leadsto M \Rightarrow \rho \vdash b \leadsto M$.*

Démonstration. Voir l'annexe B. □

Proposition 3.2.18 *Si $\xrightarrow[\mathrm{bxci}]{}$ est PSN alors $\xrightarrow[\mathrm{bx}_B\mathrm{cir}]{}$ est PSN.*

Démonstration. Le diagramme suivant, donné par les lemmes 3.2.16 et 3.2.17, et la proposition 3.2.14 montrent que s'il existe une réduction infinie par $\lambda x_B ci$, alors il existe une réduction infinie par λxci.

$$\begin{tikzcd}
a_0 \arrow[r, "\mathrm{bx}_B\mathrm{ci}"'] \arrow[d, squiggly] & a'_0 \arrow[r, two heads, "\mathrm{r}"'] \arrow[d, squiggly] & a_1 \arrow[r, "\mathrm{bx}_B\mathrm{ci}"'] \arrow[d, squiggly] & \cdots \arrow[r, two heads, "\mathrm{r}"'] & a_i \arrow[r, "\mathrm{bx}_B\mathrm{ci}"'] \arrow[d, squiggly] & a'_i \arrow[r, two heads, "\mathrm{r}"'] \arrow[d, squiggly] & \cdots \\
M_0 \arrow[r, "+", "\mathrm{bxci}"'] & M'_0 \arrow[r, equal] & M_1 \arrow[r, "+", "\mathrm{bxci}"'] & \cdots \arrow[r, equal] & M_i \arrow[r, "+", "\mathrm{bxci}"'] & M'_i \arrow[r, equal] & \cdots
\end{tikzcd}$$

□

Proposition 3.2.19 *$\xrightarrow[\mathrm{bx}_B\mathrm{cir}]{}$ n'est pas metaconfluent.*

Démonstration. Il suffit de trouver une paire critique non résolue. Prenons par exemple une des paires critiques entre (rx) et (i), caractérisée par le terme $t = a\langle \underline{n} := b\rangle \uparrow_{n+p} \langle \underline{n+p+m+1} := c\rangle$. Par (rx) on a $t \to a \uparrow_{n+p+m+1} \langle \underline{n} := b \uparrow_p\rangle\langle \underline{n+p+m+1} := c\rangle$ et par (i), $t \to a\langle \underline{n} := b\rangle\langle \underline{n+p+m} := c\rangle \uparrow_{n+p}$. Si $n+p+m+1 \in \mathrm{pfi}(a \uparrow_{n+p+m+1} \langle \underline{n} := b \uparrow_p\rangle)$, *i.e.*, $n+p+m \in a\langle \underline{n} := b\rangle$, alors la paire critique converge. Par contre, dans le cas contraire, aucune règle ne peut s'appliquer sur aucun des deux termes. □

Cette dernière proposition est due au fait qu'il n'est pas possible de définir une règle strictement équivalente à la règle (xgc) de λxci. En effet, une telle règle devrait s'exprimer comme suit :

$$a\langle \underline{n} := b\rangle \to a \quad \text{si } \underline{n} \notin \mathrm{pfi}(a)$$

Or cette règle est incorrecte, car la substitution de l'indice $\underline{n}$ transporte une information de renommage nécessaire même si b est inutile pour la réduction du terme. Il existe donc des cas où ni (xgc1), ni (xgc2) ne peuvent s'appliquer, mais pour lesquels il existe un équivalent dans $\lambda xc1$ tel que (xgc) pourrait s'appliquer.

[3] Il est important de remarquer que dans ce cas la taille de la liste est identique à la taille de l'ensemble, puisque nous travaillons sous la convention de noms de Barendregt (1.4.4), c'est-à-dire que toutes les variables dans un même environnement ont des noms distincts.

λx_Bci n'est pas metaconfluent. Cependant, comme nous avons défini les metavariables syntaxiques, qui transportent avec elles l'information sur leurs variables libres, il devrait être possible de démontrer, par une induction sur les termes avec metavariables syntaxiques, que λx_Bci est confluent. Cette preuve est cependant lourde à mettre en œuvre, et nous ne sommes pas encore en mesure de la donner.

3.3 Conclusion

Nous avons étudié le problème de la définition d'un calcul PSN et metaconfluent selon deux approches : La première est une approche assez naïve qui consiste à extraire un calcul metaconfluent (un TRS conditionnel) par complétion à partir d'un calcul PSN. Nous avons vu qu'elle peut mener à des calculs dont le calcul de substitution lui même n'est pas fortement normalisant. Cette expérience nous a convaincu de la nécessité de comprendre plus finement les mécanismes en jeu dans la permutation des substitutions. La seconde approche tient compte de cette observation et des preuves de non terminaison de Melliès [63] et de Guillaume [39] respectivement pour les calculs $\lambda\sigma$ [1] et λs_e [41]. Nous faisons abstraction des problèmes liés au renommage en nous plaçant dans le cadre d'un calcul avec noms sous la convention de Barendregt, dans le style de λx, afin de nous concentrer essentiellement sur le mécanisme de la substitution.

Nous présentons donc le calcul λxc1 comme une extension de λx avec règle de permutation, et susceptible d'être PSN. Notre intuition est basée sur l'introduction des ensembles bloquants, qui devraient empécher toute réduction infinie d'un λ-terme fortement normalisable. λxc1, comme nous l'avons démontré, n'est pas metaconfluent, mais peut être étendu en un calcul confluent sur les termes avec metavariables syntaxiques, λxci, qui préserve la forte normalisation de λxc1. C'est la définition de règles de glânage de substitutions applicables à un tel type de termes, et de la notion de variable potentiellement libres, qui permettent d'obtenir ce type de confluence.

Nous revenons ensuite sur un calcul avec renommage explicite, λx_Bci, qui préserve la forte normalisation de λxci, mais qui malheureusement n'est pas metaconfluent. Cependant, la définition de ce calcul soulève une nouvelle question : est-il vraiment utile de définir un calcul de substitution explicite metaconfluent ; ou un calcul confluent sur les termes avec metavariables syntaxiques (telles que nous les avons définies dans ce chapitre) est-il suffisant pour réaliser des applications utiles en démonstration automatique ?

Malgré notre approche originale, le problème de PSN et metaconfluence reste ouvert. Nous pensons cependant que les éléments que nous avons apportés enrichissent la reflexion sur la question de l'existence et de l'intérêt d'un tel calcul.

Chapitre 4

Preuve de correction d'une machine abstraite

La machine KP [22, 23] est une machine abstraite simple qui implante la stratégie d'appel par nécessité. Elle est basée sur les règles de la machine de Krivine [46] qui implante l'appel par nom, et à laquelle des notions d'état mémoire (ou tas) et de mise à jour ont été ajoutées, dans le style de STG [71]. La correction de la machine KP a été prouvée par Crégut. De même, il a été prouvé que la stratégie de KP était une stratégie du redex le plus à gauche (et donc une stratégie sûre). Cependant, il n'a jamais été démontré que KP implantait exactement la stratégie d'appel par nécessité des machines à environnement. Autrement dit, la preuve de Crégut –comme la plupart des preuves de correction de machines abstraites, lorsqu'elles existent– n'intègre pas du tout les notions de partage. Grâce à un outil comme $\lambda\sigma_w^a$, nous pouvons à présent pallier ce manque. Les preuves de correction que nous donnons sont l'illustration de la puissance expressive de ce calcul : les substitutions explicites permettent de détailler des stratégies qui ne peuvent être exprimées en λ-calcul, notamment en permettant la représentation de la notion d'environnement ; les adresses permettent l'explicitation des notions de partages de sous-termes, et une caractérisation précise des différentes stratégies implantables par les machines abstraites des langages de programmation fonctionnelle. Cette démonstration nous servira également de base pour démontrer la correction et la stratégie suivie par une autre machine que nous introduirons au chapitre 5.

Ce chapitre est organisé en 5 sections. La section 4.1 est une section préliminaire qui nous permet de définir des notions de base utilisées par la suite. Nous décrivons à la section 4.2 les machines de Krivine et KP, et nous donnons une preuve de KP utilisant $\lambda\sigma_w^a$ à la section 4.3. La section 4.4 montre comment la preuve est modifiée lorsque KP intègre des aspects de récursion et structures de données algébriques. Nous conclurons à la section 4.5.

4.1 Préliminaires

Nous précisons dans cette section des notations qui sont locales à ce chapitre. En particulier, la notion de *mapping* qui est la représentation d'une application partielle d'éléments d'un ensemble A vers des éléments d'un ensemble B. Soit $a_1, \ldots, a_n$ des éléments (pas forcément distincts) d'un ensemble A et $b_1, \ldots, b_n$ des éléments d'un ensemble b. Un mapping est une liste ρ notée $[a_1 \mapsto b_1; \ldots; a_n \mapsto b_n]$. L'ensemble des a_i est appelé le *domaine* du mapping, que l'on note $\mathrm{dom}(\rho)$. Un mapping de domaine vide est noté $[\,]$. Un mapping peut être appliqué à des éléments de son domaine tel que $[a_1 \mapsto b_1; \ldots; a_{i+n} \mapsto b_{i+n}]\, a = b_i$ si et seulement s'il existe i tel que $a = a_i$ et quelque soit

$$\big(\lambda M[e_1]\ ;\ N[e_2]\cdot s\big) \to \big(M\big[N[e_2]\cdot e_1\big]\ ;\ s\big) \qquad \text{(Lam)}$$
$$\big(MN[e]\ ;\ s\big) \to \big(M[e]\ ;\ N[e]\cdot s\big) \qquad \text{(App)}$$
$$\big(\underline{n}\,[M_0[e_0]\ldots M_n[e_n]\cdot e]\ ;\ s\big) \to \big(M_n[e_n]\ ;\ s\big) \qquad \text{(Access)}$$

FIG. 4.1 – La machine de Krivine

$j, (i+n > j > i)$, on a $a \neq a_j$. Un mapping ρ de A vers B peut être étendu ou modifié par un mapping $\rho' = [a_0 \mapsto b_0; \ldots; a_n \mapsto b_n]$ tel que $\forall 0 \leq i \leq n : a_i \in A, b_i \in B$. Le résultat est noté $\rho[a_0 \mapsto b_0; \ldots; a_n \mapsto b_n]$ ou $\rho\rho'$.

4.2 Description des machines K et KP

La machine de Krivine est présentée sur la figure 4.1. Elle décrit des transitions d'*états* $(M[e]\ ;\ s)$ qui ont trois composants :

1. le *code* M, un λ-terme en notation de de Bruijn ;
2. l'*environnement* e, une liste de *clôtures* ;
3. la *pile des arguments* s, qui a la même structure qu'un environnement.

Une *clôture* est un couple noté $M[e]$ et composé d'un terme (son code) et d'un environnement. Quand on évalue une application (App), une clôture est construite avec le code de l'argument et l'environnement courant, puis empilée sur la pile des arguments. Une abstraction λM représente une fonction qui attend au moins un argument. Quand on évalue cette fonction dans un certain contexte (Lam), c'est la clôture qui est sur la pile qui joue le rôle d'argument. Elle est alors dépilée et liée à la variable $\underline{0}$. Toute clôture qui était liée précédemment à l'indice $\underline{n}$ dans l'environnement précédent est liée maintenant à l'indice $\underline{n+1}$.

Pour permettre le partage de l'évaluation des clôtures de la machine de Krivine, on ajoute aux états un état mémoire, ou tas (*heap*), noté h, qui est un mapping des adresses de clôtures vers les clôtures. Les clôtures sont donc à présent référencées dans le tas, c'est ce qui permet de les partager. Lorsqu'on réduit une clôture, il faut l'isoler de son contexte. C'est pourquoi on introduit également une nouvelle pile, nommée pile des mise à jour (*update stack*) et notée u. Cette pile permet de sauvegarder la pile des arguments (le contexte) ainsi que l'adresse de la clôture que l'on est en train d'évaluer, afin de pouvoir la mettre à jour avec sa valeur à la fin de l'évaluation. La présentation originale de KP par Crégut ne fait pas mention de cette deuxième pile mais utilise un système de marques pour isoler la partie de la pile qui représente le contexte. Nous avons préféré reprendre la notion de pile de mise à jour de la machine STG [71] qui nous semble être une façon plus élégante de présenter les choses et qui est tout à fait équivalente. Les règles et les structures de données de la machine KP sont présentées sur la figure 4.2.

4.3 Preuve de correction de KP

Définition 4.3.1 *Soit $\mathcal{A}$ un ensemble d'adresses infini et dénombrable dans* $\mathbf{A}$ *(Rappel :* $\mathbf{A}$ *est l'ensemble des des adresses de $\lambda\sigma_w^a$, voir la section 2.5) dont les éléments sont notés $\alpha, \ldots, \alpha_i, \ldots$,*

États.

$$\begin{aligned}
S &::= (M[e] \ ; \ s \ ; \ u \ ; \ h) && \text{(État)}\\
M, N &::= \lambda M \ \mid \ MN \ \mid \ \underline{n} && \text{(Code)}\\
e, s &::= a \cdot e \ \mid \ \epsilon && \text{(Environnement, Pile)}\\
u &::= (s, a) \cdot u \ \mid \ \epsilon && \text{(Pile des mise à jour)}\\
h &::= [\,] \ \mid \ h\big[a \mapsto M[e]\big] && \text{(Tas)}\\
& a \text{ dénote les adresses de clôtures}
\end{aligned}$$

Règles.

$$\begin{aligned}
(MN[e] \ ; \ s \ ; \ u \ ; \ h) &\to (M[e] \ ; \ a \cdot s \ ; \ u \ ; \ h\big[a \mapsto N[e]\big]) \quad a \text{ fraîche} && \text{(App)}\\
(\lambda M[e] \ ; \ a \cdot s \ ; \ u \ ; \ h) &\to (M[a \cdot e] \ ; \ s \ ; \ u \ ; \ h) && \text{(Lam)}\\
(\,\underline{n+1}\,[a \cdot e] \ ; \ s \ ; \ u \ ; \ h) &\to (\,\underline{n}\,[e] \ ; \ s \ ; \ u \ ; \ h) && \text{(Skip)}\\
(\,\underline{0}\,[a \cdot e] \ ; \ s \ ; \ u \ ; \ h) &\to (N[e'] \ ; \ \epsilon \ ; \ (s, a) \cdot u \ ; \ h) && \text{(Access)}\\
& \text{où } h\,a = N[e'] \\
(\lambda M[e] \ ; \ \epsilon \ ; \ (s, a) \cdot u \ ; \ h) &\to (\lambda M[e] \ ; \ s \ ; \ u \ ; \ h\big[a \mapsto \lambda M[e]\big]) && \text{(Update)}
\end{aligned}$$

FIG. 4.2 – La machine KP

tel que si a et b sont des adresses distinctes de clôtures de KP, alors elles représentent également des adresses de termes distinctes appartenant à $\mathbf{A} \setminus \mathcal{A}$.

La fonction de traduction ν *des états de KP vers les pré-termes de* $\lambda\sigma_w^a$ *est définie par*

$$\nu(M[e] \ ; \ s \ ; \ u \ ; \ h) = \tau(M[\gamma(e, \rho)]^\alpha, s, u, \rho) \qquad \text{où } \rho = \phi(h), \text{un mapping de } \mathbf{A} \text{ vers les } \lambda\sigma_w^a\text{–termes},$$

où τ *est la fonction définie par*

$$\begin{aligned}
\tau(E^b, a \cdot s, u, \rho) &= \tau\big((E^b\,(\rho\, a))^\alpha, s, u, \rho\big)\\
\tau(E^b, \epsilon, (s, a) \cdot u, \rho) &= \tau(\,\underline{0}\,[E^a \cdot \mathrm{id}]^b, s, u, \rho[a \mapsto E^a])\\
\tau(E^a, \epsilon, \epsilon, \rho) &= E^a
\end{aligned}$$

et où chaque adresse de $\mathcal{A}$ *est attribuée au plus une fois, i.e. chaque* $\alpha \in \mathcal{A}$ *a au plus une occurrence dans* $\nu(S)$ *pour tout état* S. ϕ *est la fonction définie par*

$$\begin{aligned}
\phi(h\big[a \mapsto N[e]\big]) &= \phi(h)\big[a \mapsto N[\gamma(e, \phi(h))]^a\big]\\
\phi([\,]) &= [\,]
\end{aligned}$$

Comme on le constate, ϕ *est la fonction qui permet de déployer le tas, le transformant en un mapping des adresses vers les termes de* $\lambda\sigma_w^a$. *Il est à noter que* ϕ *est bien définie, comme cela est démontré par le lemme 4.3.2. Enfin,* γ *est définie par*

$$\begin{aligned}
\gamma(a \cdot e, \rho) &= (\rho\, a) \cdot \gamma(e, \rho)\\
\gamma(\epsilon, \rho) &= \mathrm{id}
\end{aligned}$$

Lemme 4.3.2 *Soit M un λ_{NF}-term et S un état tel que $(M[\epsilon]\ ;\ \epsilon\ ;\ \epsilon\ ;\ [\,]) \twoheadrightarrow S$, alors $\nu(S)$ est défini.*

Démonstration. On note simplement que :

1. $\nu(M[\epsilon]\ ;\ \epsilon\ ;\ \epsilon\ ;\ [\,])$ est défini pour tout M et vaut $M[\mathrm{id}]^\alpha$.
2. Soit $S = (M[e]\ ;\ s\ ;\ u\ ;\ h) \to (M'[e']\ ;\ s'\ ;\ u'\ ;\ h') = S'$ un pas de dérivation par KP. Si $\nu(S)$ est défini, alors $\phi(h')$ est défini. Par exemple, dans la règle (App), on déduit des hypothèses que $\gamma(e, \phi(h))$ est défini, par conséquent $\phi(h\big[a \mapsto N[e]\big]) = \phi(h)\big[a \mapsto N[\gamma(e, \phi(h))]\big]$ est défini. Pour les autres règles, c'est encore plus évident.
3. Toute adresse dans un état $(M[e]\ ;\ s\ ;\ u\ ;\ h)$ est dans le domaine du mapping $\phi(h)$. C'est trivial pour l'adresse fraîche a de la règle (App), et démontrable par une induction facile pour les autres adresses et les autres règles.
4. Enfin, il est facile de démontrer que ν est défini quand toutes les conditions ci-dessus sont satisfaites.

□

Lemme 4.3.3 *Soit M un λ_{NF}-terme, $\nu(M[\epsilon]\ ;\ \epsilon\ ;\ \epsilon\ ;\ [\,])$ est admissible.*

Démonstration. $\nu(M[\epsilon]\ ;\ \epsilon\ ;\ \epsilon\ ;\ [\,]) = M[\mathrm{id}]^\alpha$ est admissible car $\mathrm{addr}(M[\mathrm{id}]^\alpha) = \{\alpha\}$ et $M[\mathrm{id}]^\alpha@\alpha = \{M[\mathrm{id}]^\alpha\}$. □

Définition 4.3.4 *Soit deux termes P et E^a, on définit $P\{a := E^a\}$ comme le terme résultant du remplacement de $P@a$ par E^a dans P. Ceci est étendu aux mappings, où $\rho[b \mapsto F^b]\{a := E^a\} = \rho\{a := E^a\}[b \mapsto F^b\{a := E^a\}]$ et $[\,]\{a := E^a\} = [\,]$.*

Lemme 4.3.5 *Soit E^a un terme et ρ un mapping tel que $\big(E^a\,(\rho\,a')\big)^c$ est admissible. Si $E^a \xrightarrow{b}_{/\mathrm{CBNeedE}} F^a$ alors $\big(E^a\,(\rho\,a')\big)^c \xrightarrow{b}_{/\mathrm{CBNeedE}} \big(F^a\,(\rho\{b := F^a@b\}\,a')\big)^c$.*

Démonstration. Il est évident que si $E_1^{a_1} \xrightarrow{b}_{/\mathrm{CBNeedE}} F^{a_1}$ alors $(E_1^{a_1}\,E_2^{a_2})^c \xrightarrow{b}_{/\mathrm{CBNeedE}} (F^{a_1}\,E_2^{a_2}\{b := F^{a_1}@b\})^c$ par définition de $\Vdash\!\!\to$. Il est également clair que $(\rho\,a)\{b := E^b\} = (\rho\{b := E^b\}\,a)$. □

Lemme 4.3.6 *Soit E^a, s, u, ρ tels que $\tau(E^a, s, u, \rho)$ est admissible. Si $E^a \xrightarrow{b}_{/\mathrm{CBNeedE}} F^a$ alors*

$$\tau(E^a, s, u, \rho) \longrightarrow_{/\mathrm{CBNeedE}} \tau(F^a, s, u, \rho\{b := F^a@b\}).$$

Démonstration. Par induction sur u. Dans la suite ρ' désigne $\rho\{b := F^a@b\}$.

– u vide : par induction sur s. Si s est vide alors $\tau(E^a, \epsilon, \epsilon, \rho) = E^a \longrightarrow_{/\mathrm{CBNeedE}} F^a = \tau(F^a, \epsilon, \epsilon, \rho')$. Sinon, $\tau(E^a, a' \cdot s, \epsilon, \rho) = \tau\big((E^a\,(\rho\,a'))^\alpha, s, \epsilon, \rho\big)$; comme $\big(E^a\,(\rho\,a')\big)^\alpha \xrightarrow{b}_{/\mathrm{CBNeedE}} \big(F^a\,(\rho'\,a')\big)^\alpha$ (lemme 4.3.5), et comme $\big(F^a\,(\rho'\,a')\big)^\alpha@b = F^a@b$, alors par hypothèse d'induction sur s :

$$\begin{aligned}\tau(E^a, a' \cdot s, \epsilon, \rho) &\longrightarrow_{/\mathrm{CBNeedE}} \tau\big((F^a\,(\rho'\,a'))^\alpha, s, \epsilon, \rho'\big)\\ &= \tau(F^a, a' \cdot s, \epsilon, \rho')\end{aligned}$$

– u non vide : par induction sur s. Si s est vide, alors soit $P = \tau(E^a, \epsilon, (a', s') \cdot u, \rho)$.

$$\begin{aligned}P &= \tau(\,\underline{0}\,[E^{a'} \cdot \mathrm{id}]^a, s', u, \rho[a' \mapsto E^{a'}])\\ &\longrightarrow_{/\mathrm{CBNeedE}} \tau(\,\underline{0}\,[F^{a'} \cdot id]^a, s', u, \rho[a' \mapsto E^{a'}]\{b := F^{a'}@b\})\\ &\qquad \text{par h.i sur } u \text{ et car } \underline{0}\,[E^{a'} \cdot \mathrm{id}]^a \longrightarrow_{/\mathrm{CBNeedE}} \underline{0}\,[F^{a'} \cdot \mathrm{id}]^a\\ &= \tau(\,\underline{0}\,[F^{a'} \cdot id]^a, s', u, \rho'[a' \mapsto F^{a'}]) \text{ car } E^{a'}\{b := F^{a'}@b\} = F^{a'}\\ &= \tau(F^a, \epsilon, (a', s') \cdot u, \rho')\end{aligned}$$

Sinon, soit $P = \tau(E^a, a' \cdot s, u, \rho)$.

$$
\begin{aligned}
P &= \tau\big((E^a\ (\rho\, a'))^\alpha, s, u, \rho\big) \\
&\xrightarrow{}_{/\mathrm{CBNeedE}} \tau\big((F^a\ (\rho'\, a'))^\alpha, s, u, h, \rho'\big) \\
&\qquad \text{d'après le lemme 4.3.5, par h.i. sur } s \text{, et car } \big(F^a\ (\rho'\, a')\big)^\alpha @ b = F^a @ b. \\
&= \tau(F^a, a' \cdot s, u, \rho')
\end{aligned}
$$

□

Corollaire 4.3.7 *Soit E^a un $\lambda\sigma_w^a$-term et s, u, ρ tels que $\tau(E^a, s, u, \rho)$ est admissible.*
Si $E^a \xrightarrow{a}_{/\mathrm{CBNeedE}} F^a$ alors $\tau(E^a, s, u, \rho) \xrightarrow{}_{/\mathrm{CBNeedE}} \tau(F^a, s, u, \rho)$.

Démonstration. On peut remarquer d'après la définition de τ que a est un α. Par conséquent il n'est attribué qu'une fois et n'a pas encore d'occurrence dans ρ, et $\rho\{\alpha := E^\alpha\} = \rho$. □

Lemme 4.3.8 *Si $M[\gamma(e,\rho)]^a$ est admissible alors $M[\gamma(e,\rho)]^a = M\big[\gamma(e, \rho[a \mapsto E^a])\big]^a$.*

Démonstration. Comme $M[\gamma(e,\rho)]^a$ est admissible, a ne peut pas être une sous-adresse de a. Par conséquent, $\rho[a \mapsto E^a]$ n'est jamais appliqué à a, et pour toute autre adresse b, $\rho\, b = \rho[a \mapsto E^a]\, b$.

□

Théorème 4.3.9 *Si $(M[e]\ ;\ s\ ;\ u\ ;\ h) \to (M'[e']\ ;\ s'\ ;\ u'\ ;\ h')$ alors $\nu(M[e]\ ;\ s\ ;\ u\ ;\ h) \xrightarrow{}_{/\mathrm{CBNeedE}} \nu(M'[e']\ ;\ s'\ ;\ u'\ ;\ h')$.*

Démonstration. Dans la suite, ρ désigne $\phi(h)$ et ρ' désigne $\rho\big[a \mapsto N[\gamma(e,\rho)]^a\big] = \phi(h\big[a \mapsto N[e]\big])$.
Soit $P = \nu(MN[e]\ ;\ s\ ;\ u\ ;\ h)$.

$$
\begin{aligned}
P &= \tau\big((MN)[\gamma(e,\rho)]^{\alpha_0}, s, u, \rho\big) \\
&\xrightarrow{}_{/\mathrm{CBNeedE}} \tau\big((M[\gamma(e,\rho)]^{\alpha_1} N[\gamma(e,\rho)]^a)^{\alpha_0}, s, u, \rho\big) \\
&\qquad \text{d'après le corollaire 4.3.7, et car } a \text{ et } \alpha_1 \text{ sont fraîches.} \\
&= \tau\big((M[\gamma(e,\rho')]^{\alpha_1} N[\gamma(e,\rho)]^a)^{\alpha_0}, s, u, \rho'\big) \\
&\qquad \text{car } a \text{ est fraîche} \\
&= \tau\big((M[\gamma(e,\rho')]^{\alpha_1}\ (\rho'\, a))^{\alpha_0}, s, u, \rho'\big) \quad \text{car par définition } \rho'\, a = N[\gamma(e,\rho)]^a \\
&= \tau(M[\gamma(e,\rho')]^{\alpha_1}, a \cdot s, u, \rho') \quad \text{par définition de } \tau \\
&= \nu(M[e]\ ;\ a \cdot s\ ;\ u\ ;\ h\big[a \mapsto N[e]\big]) \quad \text{par définition de } \nu
\end{aligned}
$$

Soit $P = \nu(\lambda M[e]\ ;\ a \cdot s\ ;\ u\ ;\ h)$.

$$
\begin{aligned}
P &= \tau\big((\lambda M)[\gamma(e,\rho)]^{\alpha_0}, a \cdot s, u, \rho\big) \\
&= \tau\big(((\lambda M)[\gamma(e,\rho)]^{\alpha_0}\ (\rho\, a))^{\alpha_1}, s, u, \rho\big) \\
&\xrightarrow{}_{/\mathrm{CBNeedE}} \tau(M[(\rho\, a) \cdot \gamma(e,\rho)]^{\alpha_1}, s, u, \rho) \text{ d'après le corollaire 4.3.7} \\
&= \tau(M[\gamma(a \cdot e, \rho)]^{\alpha_1}, s, u, \rho) \\
&= \nu(M[a \cdot e]\ ;\ s\ ;\ u\ ;\ h)
\end{aligned}
$$

Soit $P = \nu(\underline{n+1}\,[a \cdot e]\ ;\ s\ ;\ u\ ;\ h)$.

$$
\begin{aligned}
P &= \tau(\underline{n+1}\,[\gamma(a \cdot e, \rho)]^\alpha, s, u, \rho) \\
&= \tau(\underline{n+1}\,[(\rho\, a) \cdot \gamma(e,\rho)]^\alpha, s, u, \rho) \\
&\xrightarrow{}_{/\mathrm{CBNeedE}} \tau(\underline{n}\,[\gamma(e,\rho)]^\alpha, s, u, \rho) \text{ d'après le corollaire 4.3.7} \\
&= \nu(\underline{n}\,[e]\ ;\ s\ ;\ u\ ;\ h)
\end{aligned}
$$

Soit $P = \nu(\underline{0}\,[a \cdot e]\ ;\ s\ ;\ u\ ;\ h)$.

$$
\begin{aligned}
P &= \tau(\underline{0}\,[\gamma(a \cdot e, \rho)]^{\alpha}, s, u, \rho)\\
&= \tau(\underline{0}\,[(\rho\, a) \cdot \gamma(e, \rho)]^{\alpha}, s, u, \rho)\\
&\longmapsto_{/\mathrm{CBNeedE}} \tau(\underline{0}\,[(\rho\, a) \cdot \mathrm{id}]^{\alpha}, s, u, \rho) \text{ d'après le corollaire 4.3.7}\\
&= \tau(\underline{0}\,[(\rho\, a) \cdot \mathrm{id}]^{\alpha}, s, u, \rho[a \mapsto (\rho\, a)]) \text{ car } \rho[a \mapsto (\rho\, a)] = \rho\\
&= \tau(N[\gamma(e', \rho)]^{\alpha}, \epsilon, (s, a) \cdot u, \rho)\\
&\qquad\qquad\qquad \text{car } h\, a = N[e'] \text{ implique } \rho\, a = N[\gamma(e', \rho)]^{a}\\
&= \nu(N[e']\ ;\ \epsilon\ ;\ (s, a) \cdot u\ ;\ h)
\end{aligned}
$$

Notons que la règle de $\lambda\sigma_w^a$ utilisée ici est une variante de (Collect) :

$$\underline{0}\,[P \cdot s]^{a} \to \underline{0}\,[P \cdot \mathrm{id}]^{a}$$

Le lecteur peut facilement se convaincre que cette règle est correcte, car $\underline{0}\,[P{\cdot}s]^{a} \to \underline{0}\,[P{\cdot}\bot{\cdot}\ldots{\cdot}\bot{\cdot}\mathrm{id}]$ qui est évidemment équivalent à $\underline{0}\,[P \cdot \mathrm{id}]$.

Dans la suite, ρ' désigne $\rho\big[a \mapsto \lambda M[\gamma(e, \rho)]^{a}\big]$. Soit $P = \nu(\lambda M[e]\ ;\ \epsilon\ ;\ (s, a) \cdot u\ ;\ h)$.

$$
\begin{aligned}
P &= \tau\big((\lambda M)[\gamma(e, \rho)]^{\alpha}, \epsilon, (s, a) \cdot u, \rho\big)\\
&= \tau(\underline{0}\,[\lambda M[\gamma(e, \rho)]^{a} \cdot \mathrm{id}]^{\alpha}, s, u, \rho')\\
&= \tau(\underline{0}\,[\lambda M[\gamma(e, \rho')]^{a} \cdot \mathrm{id}]^{\alpha}, s, u, \rho') \text{ d'après le lemme 4.3.8}\\
&\longmapsto_{/\mathrm{CBNeedE}} \tau(\lambda M[\gamma(e, \rho')]^{\alpha}, s, u, \rho')\\
&\qquad\qquad \text{d'après le corollaire 4.3.7, et car } \lambda M[\gamma(e, \rho')]^{a} \text{ est une valeur}\\
&= \nu(\lambda M[e]\ ;\ s\ ;\ u\ ;\ h\big[a \mapsto \lambda M[e]\big])
\end{aligned}
$$

□

Théorème 4.3.10 *KP réduit les termes clos jusqu'à leur forme normale de tête.*

Démonstration. Les états d'arrêt sont de la forme $(\lambda M[e]\ ;\ \epsilon\ ;\ \epsilon\ ;\ h)$ qui représentent des $\lambda\sigma_w^a$-term $\lambda M[s]^{a}$. Notons que $(\underline{n}\,[\epsilon]\ ;\ s\ ;\ u\ ;\ h)$ n'est pas un état d'arrêt dans le cadre de la réduction de termes clos. □

4.4 Structures de données et récursion

KP telle que présentée dans [22, 23] ne prend pas en compte les structures de données et la récursion. Il nous paraît cependant intéressant de prouver la correction d'une machine ayant ces caractéristiques. Nous allons donc dans un premier temps enrichir KP avec ces traits[1], en nous aidant de $\lambda\sigma_w^a$, puis nous tenterons de démontrer la correction de notre machine.

Au niveau des états, la modification principale est l'ajout d'une pile, nommée pile des données et notée d. Cette pile contient des triplets sélecteur de cas, environnement, pile. Lorsqu'un terme est appliqué à un sélecteur de cas, l'évaluation du terme est effectuée jusqu'à atteindre un constructeur.

[1]Notre proposition sera légèrement différente de celle de Sestoft [77] dans le choix de l'implantation des constructeurs et de la récursion, notre solution privilégiant la simplicité par rapport au calcul qui nous sert de modèle plutôt que l'efficacité et des optimisations qui rendraient cette preuve plus complexe.

Une fois ce constructeur atteint, le choix de la bonne branche du sélecteur est effectué, et la pile qui avait été mémorisée est restaurée. L'introduction des constructeurs implique un nouveau cas de mises à jour (UpdateData), puisqu'ils sont également des valeurs. Il est à noter que la règle (Access) doit être modifiée pour que la pile des données soit empilée comme la pile des arguments sur la pile des mise à jour. D'autre part, la règle (App) n'est utilisée que pour des arguments qui ne sont pas des sélecteurs de cas. Pour les autres règles, la pile des données n'influence pas la réduction.

$$(M\star\langle\vec{C}:\vec{N}\rangle[e]\ ;\ s\ ;\ d\ ;\ u\ ;\ h) \to (M[e]\ ;\ \epsilon\ ;\ (\langle\vec{C}:\vec{N}\rangle[e],s)\cdot d\ ;\ u\ ;\ h) \qquad \text{(Data)}$$

$$(C_i[a_1\cdot\ldots\cdot a_n\cdot e]\ ;\ \epsilon\ ;\ (\langle\vec{C}:\vec{N}\rangle[f],s)\cdot d\ ;\ u\ ;\ h) \to (N_i[a_1\cdot\ldots\cdot a_n\cdot f]\ ;\ s\ ;\ d\ ;\ u\ ;\ h) \quad \text{où } \mathsf{ar}(C_i)=n \qquad \text{(Select)}$$

$$(C[e]\ ;\ \epsilon\ ;\ \epsilon\ ;\ (s,d,a)\cdot u\ ;\ h) \to (C[e]\ ;\ s\ ;\ d\ ;\ u\ ;\ h\big[a\mapsto C[e]\big]) \qquad \text{(UpdateData)}$$

$$(\mu M[e]\ ;\ \epsilon\ ;\ \epsilon\ ;\ (s,d,a)\cdot u\ ;\ h) \to (M[a\cdot e]\ ;\ \epsilon\ ;\ \epsilon\ ;\ (s,d,a)\cdot u, h) \qquad \text{(Rec)}$$

Définition 4.4.1 *La nouvelle fonction de traduction ν est définie par*

$$\nu(M[e]\ ;\ s\ ;\ d\ ;\ u\ ;\ h) = \tau(M[\gamma(e,\rho)]^\alpha, s, d, u, \rho)$$

où $u = (s_1,d_1,a_1)\cdot\ldots\cdot(s_n,d_n,a_n)\cdot\epsilon$, $\rho = \phi(h)[a_1\mapsto\bullet^{a_1};\ldots;a_n\mapsto\bullet^{a_n}]$, et où τ est la fonction définie par

$$\begin{aligned}
\tau(E^b, a\cdot s, d, u, \rho) &= \tau\big((E^b\,(\rho\,a))^\alpha, s, d, u, \rho\big)\\
\tau(E^a, \epsilon, (\langle\vec{C}:\vec{N}\rangle[e],s)\cdot d, u, \rho) &= \tau\big((E^a\star\langle\vec{C}:\vec{N}\rangle[\gamma(e,\rho)]^{\alpha_0})^{\alpha_1}, s, d, u, \rho\big)\\
\tau(E^b, \epsilon, \epsilon, (s,d,a)\cdot u, \rho) &= \tau(\underline{0}\,[E^a\cdot\mathrm{id}]^b, s, d, u, \rho[a\mapsto E^a])\\
\tau(E^a, \epsilon, \epsilon, \epsilon, \rho) &= E^a
\end{aligned}$$

ϕ est la fonction définie par

$$\begin{aligned}
\phi(h\big[a\mapsto N[e]\big]) &= \phi(h)\big[a\mapsto N[\gamma(e,\phi(h))]^a\big]\\
\phi([\,]) &= [\,]
\end{aligned}$$

et γ est définie par

$$\begin{aligned}
\gamma(a\cdot e,\rho) &= \begin{cases} (\rho\,a)\cdot\gamma(e,\rho) & \textit{si } a\in\mathcal{D}om(\rho)\\ \bullet^a\cdot\gamma(e,\rho) & \textit{sinon}\end{cases}\\
\gamma(\epsilon,\rho) &= \mathrm{id}
\end{aligned}$$

Les lemmes 4.3.2, 4.3.3, 4.3.5, 4.3.6 et le corollaire 4.3.7 sont toujours satisfaits avec la nouvelle définition de ν. Leur démonstration utilise la même argumentation que nous ne redonnons pas ici. Pour le lemme 4.3.6, la démonstration nécessite une induction supplémentaire (facile) sur la pile d, que nous ne donnons pas non plus ici. Nous introduisons enfin le nouveau lemme suivant :

Lemme 4.4.2 $M[\gamma(e,\rho)]^b\{\bullet^a := E^a\} = M[\gamma(e,\rho[a\mapsto E^a])]^b$.

Démonstration. On procède par induction sur e. Si $e = \epsilon$ alors l'égalité est triviale. Sinon, si $\rho\, a = \bullet^a$ ou $a \notin \mathcal{D}om(\rho)$ alors $\gamma(a \cdot e, \rho)\{\bullet^a := E^a\} = (\bullet^a \cdot \gamma(e, \rho))\{\bullet^a := E^a\} = E^a \cdot \gamma(e, \rho)\{\bullet^a := E^a\} = E^a \cdot \gamma(e, \rho[a \mapsto E^a]) = \gamma(a \cdot e, \rho[a \mapsto E^a])$ par hypothèse d'induction. Enfin, $\gamma(b \cdot e, \rho)\{\bullet^a := E^a\} = x^b \cdot \gamma(e, \rho)\{\bullet^a := E^a\} = x^b \cdot \gamma(e, \rho[a \mapsto E^a]) = \gamma(b \cdot e, \rho[a \mapsto E^a])$ par hypothèse d'induction, avec $b \neq a$ et $x^b = \rho\, b$ si $b \in \mathcal{D}om(\rho)$ ou $\bullet^b$ sinon. □

Théorème 4.4.3 *Si* $(M[e]\ ;\ s\ ;\ u\ ;\ h) \to (M'[e']\ ;\ s'\ ;\ u'\ ;\ h')$ *alors* $\nu(M[e]\ ;\ s\ ;\ u\ ;\ h) \;{}_{/\mathrm{CBNeedE}}\!\!\overset{\Downarrow}{\longmapsto}\; \nu(M'[e']\ ;\ s'\ ;\ u'\ ;\ h')$.

Démonstration. Dans la démonstration de ce théorème on prend pour convention que $u = (s_1, d_1, a_1) \cdot \ldots \cdot (s_n, d_n, a_n)$ et on note $[\vec{a} \mapsto \bullet^{\vec{a}}]$ pour $[a_1 \mapsto \bullet^{a_1}; \ldots; a_n \mapsto \bullet^{a_n}]$. Enfin, on note ρ pour $\phi(h)[\vec{a} \mapsto \bullet^{\vec{a}}]$.

- Règles (App, Lam, Skip, Access) : l'argumentation de la preuve du théorème 4.3.9 reste valable.
- Règle (Update) : l'argumentation de la démonstration du théorème 4.3.9 n'est plus valable car le lemme 4.3.8 n'est pas satisfait dans le cadre de la nouvelle définition de ν.
  Soit $P = \nu\big(\lambda M[e]\ ;\ \epsilon\ ;\ \epsilon\ ;\ (s, d, a) \cdot u, h\big)$ et $\rho' = \rho[a \mapsto \bullet^a]$.

$$\begin{aligned}
P &= \tau(\lambda M[\gamma(e, \rho)]^\alpha, \epsilon, \epsilon, (s, d, a) \cdot u, \rho') \\
&= \tau(\underline{0}\,[\lambda M[\gamma(e, \rho')]^a \cdot \mathrm{id}]^\alpha, s, d, u, \rho[a \mapsto \lambda M[\gamma(e, \rho')]^a]) \\
&{}_{/\mathrm{CBNeedE}}\!\!\overset{\Downarrow}{\longmapsto} \tau(\lambda M[\gamma(e, \rho')]^\alpha\{\bullet^a := \lambda M[\gamma(e, \rho')]^a\}, s, d, u, \rho[a \mapsto \lambda M[\gamma(e, \rho')]^a]) \\
&= \tau(\lambda M[\gamma(e, \rho[a \mapsto \lambda M[\gamma(e, \rho')]^a])]^\alpha, s, d, u, \rho[a \mapsto \lambda M[\gamma(e, \rho')]^a]) \text{ d'après le lemme 4.4.2} \\
&= \nu(\lambda M[e]\ ;\ s\ ;\ d\ ;\ u\ ;\ h\big[a \mapsto \lambda M[e]\big])
\end{aligned}$$

- Règle (Data) : soit $P = \nu(M{\star}\langle \vec{C} : \vec{N}\rangle[e]\ ;\ s\ ;\ d\ ;\ u\ ;\ h)$.

$$\begin{aligned}
P &= \tau\big((M{\star}\langle \vec{C} : \vec{N}\rangle)[\gamma(e, \rho)]^{\alpha_0}, s, d, u, \rho\big) \\
&{}_{/\mathrm{CBNeedE}}\!\!\overset{\Downarrow}{\longmapsto} \tau\big((M[\gamma(e, \rho)]^{\alpha_1}{\star}\langle \vec{C} : \vec{N}\rangle[\gamma(e, \rho)])^{\alpha_0}, s, d, u, \rho\big) \\
&= \tau(M[\gamma(e, \rho)]^{\alpha_1}, \epsilon, (\langle \vec{C} : \vec{N}\rangle[e], s) \cdot d, u, \rho) \\
&= \nu(M[e]\ ;\ \epsilon\ ;\ (\langle \vec{C} : \vec{N}\rangle[e], s) \cdot d\ ;\ u\ ;\ h)
\end{aligned}$$

- Règle (Select) : soit $P = \nu(C_i[a_1 \cdot \ldots \cdot a_n \cdot e]\ ;\ \epsilon\ ;\ (\langle \vec{C} : \vec{N}\rangle[f], s) \cdot d\ ;\ u\ ;\ h)$.

$$\begin{aligned}
P &= \tau(C_i[\gamma(a_1 \cdot \ldots \cdot a_n \cdot e, \rho)]^{\alpha_0}, \epsilon, (\langle \vec{C} : \vec{N}\rangle[f], s) \cdot d, u, \rho) \\
&= \tau\big((C_i[\gamma(a_1 \cdot \ldots \cdot a_n \cdot e, \rho)]^{\alpha_0}{\star}\langle \vec{C} : \vec{N}\rangle[\gamma(f, \rho)])^{\alpha_1}, s, d, u, \rho\big) \\
&{}_{/\mathrm{CBNeedE}}\!\!\overset{\Downarrow}{\longmapsto} \tau(N_i[\gamma(a_1 \cdot \ldots \cdot a_n \cdot f, \rho)]^{\alpha_1}\{\bullet^{\alpha_0} := C_i[\gamma(a_1 \cdot \ldots \cdot a_n \cdot e, \rho)]^{\alpha_0}\}, s, d, u, \rho) \\
&= \tau(N_i[\gamma(a_1 \cdot \ldots \cdot a_n \cdot f, \rho)]^{\alpha_1}, s, d, u, \rho) \quad \text{since } \alpha_0 \text{ was fresh} \\
&= \nu(N_i[a_1 \cdot \ldots \cdot a_n \cdot f]\ ;\ s\ ;\ d\ ;\ u\ ;\ h)
\end{aligned}$$

- Règle (UpdateData) : soit $P = \nu(C[e]\ ;\ \epsilon\ ;\ \epsilon\ ;\ (s, d, a) \cdot u\ ;\ h)$.

$$\begin{aligned}
P &= \tau(C[\gamma(e, \rho)]^\alpha, \epsilon, \epsilon, (s, d, a) \cdot u, \rho) \\
&= \tau(\underline{0}\,[C[\gamma(e, \rho)]^a \cdot \mathrm{id}]^\alpha, s, d, u, \rho\big[a \mapsto C[\gamma(e, \rho)]^a\big]) \\
&{}_{/\mathrm{CBNeedE}}\!\!\overset{\Downarrow}{\longmapsto} \tau(C[\gamma(e, \rho)]^\alpha\{\bullet^a := C[\gamma(e, \rho)]^a\}, s, d, u, \rho\big[a \mapsto C[\gamma(e, \rho)]^a\big]) \\
&= \tau(C\big[\gamma(e, \rho[a \mapsto C[\gamma(e, \rho)]^a])\big]^\alpha, s, d, u, \rho\big[a \mapsto C[\gamma(e, \rho)]^a\big]) \text{ d'après le lemme 4.4.2} \\
&= \nu(C[e]\ ;\ s\ ;\ d\ ;\ u\ ;\ h\big[a \mapsto C[e]\big])
\end{aligned}$$

– Règle (Rec) : soit $P = (\mu M[e]\ ;\ \epsilon\ ;\ \epsilon\ ;\ (s,d,a)\cdot u, h)$.

$$\begin{aligned}
P &= \tau(\mu M[\gamma(e,\rho)]^\alpha, \epsilon, \epsilon, (s,d,a)\cdot u, \rho)\\
&= \tau(\,\underline{0}\,[\mu M[\gamma(e,\rho)]^a \cdot \mathrm{id}]^\alpha, s, d, u, \rho\big[a \mapsto \mu M[\gamma(e,\rho)]^a\big])\\
&\stackrel{a}{\Vdash\!\!\rightarrow}_{/\text{CBNeedE}}\ \tau(\,\underline{0}\,\big[M[\bullet^a\cdot\gamma(e,\rho)]^a \cdot \mathrm{id}]^\alpha, s, d, u, \rho\big[a \mapsto M[\bullet^a\cdot\gamma(e,\rho)]^a\big]) \quad \text{d'après le lemme 4.3.6.}\\
&\Vdash\!\!\rightarrow_{/\text{CBNeedE}}\ \tau(M[\bullet^a\cdot\gamma(e,\rho)]^\alpha\{\bullet^a := M[\bullet^a\cdot\gamma(e,\rho)]\}, s, d, u, \rho\big[a \mapsto M[\bullet^a\cdot\gamma(e,\rho)]^a\big])\\
&= \tau(M[M[\bullet^a\cdot\gamma(e,\rho)]^a \cdot \gamma(e, \rho[a \mapsto M[\bullet^a\cdot\gamma(e,\rho)]^a])]^\alpha, s, d, u, \rho\big[a \mapsto M[\gamma(a\cdot e,\rho)]^a\big])\\
&= \tau(M[\gamma(a\cdot e,\rho)]^\alpha, s, d, u, \rho\big[a \mapsto M[\gamma(a\cdot e,\rho)]^a\big])\\
&= \tau(M[\gamma(a\cdot e,\rho)]^\alpha, \epsilon, \epsilon, (s,d,a)\cdot u, \rho)\\
&= \nu(M[a\cdot e], \epsilon, \epsilon, (s,d,a)\cdot u, \rho)
\end{aligned}$$

□

La partie restante de la preuve consiste à démontrer que la réduction s'arrête à la forme normale de tête des termes donnés en entrée. Cette preuve est très simple dans le cadre des termes sans structures de données ni récursion, car il n'existe qu'un seul cas d'arrêt. Ici, il existe une multitude de cas d'arrêt, mais il faut démontrer que tous les états correspondant représentent des valeurs ou sont inaccessibles. Il nous faut donc d'abord définir la classe restreinte des termes qui peuvent être donnés en entrée, c'est-à-dire ceux qui ont un sens en terme de programmes. Par exemple, un terme de la forme $(\mu M)\,N$ n'a pas de sens, seuls les programmes où μM est en position d'argument étant admissibles. Pour déterminer la classe des termes admissibles, nous allons donc définir un langage fonctionnel de plus haut niveau que $\lambda\sigma_w^a$, et donner les règles qui permettent de compiler les termes de $\lambda\sigma_w^a$ dans ce langage. Seuls les termes issus de la compilation de programmes écrits dans ce langage de haut niveau seront considérés comme admissibles.

Définition 4.4.4 *On définit un langage $\mathcal{L}$ dont la syntaxe abstraite est la suivante : un programme est une expression e définie par la grammaire*

$$\begin{array}{rcll}
e & ::= & e_1\, e_2 & \text{(Application)}\\
& | & \mathsf{fun}\ x \to e & \text{(Abstraction)}\\
& | & \mathsf{let}\ x = e_1\ \mathsf{in}\ e_2 & \text{(Let)}\\
& | & \mathsf{letrec}\ x = e_1\ \mathsf{in}\ e_2 & \text{(Let récursif)}\\
& | & \mathsf{match}\ e\ \mathsf{with}\ a & \text{(Filtrage)}\\
& | & x & \text{(Variables)}\\
& | & C & \text{(Constructeurs)}\\
a & ::= & \begin{array}{ll} & C_1\, x_1^1 \ldots x_{n_1}^1 \to e_1\\ \llbracket & C_2\, x_1^2 \ldots x_{n_2}^2 \to e_2\\ & \vdots\\ \llbracket & C_m\, x_1^m \ldots x_{n_m}^m \to e_m \end{array} \qquad \textit{où } n_i = \mathsf{ar}(C_i), m > 0 & \text{(Alternatives)}
\end{array}$$

Une alternative a est un sélecteur de cas où $\{C_1, \ldots, C_m\}$ sont (tous) les constructeurs d'un type de données algébrique. Hormis ces types de données, $\mathcal{L}$ est un langage non typé. Soit e une expression de $\mathcal{L}$, on définit la traduction $\mathcal{C}\llbracket e\rrbracket_{[]}$ (compilation) de cette expression vers du code de

$\lambda\sigma_w^a$ *comme suit :*

$$\begin{aligned}
\mathcal{C}[\![e_1\, e_2]\!]_\rho &= \mathcal{C}[\![e_1]\!]_\rho\, \mathcal{C}[\![e_2]\!]_\rho \\
\mathcal{C}[\![\mathsf{fun}\ x \to e]\!]_\rho &= \lambda \mathcal{C}[\![e]\!]_{x\cdot\rho} \\
\mathcal{C}[\![\mathsf{let}\ x = e_1\ \mathsf{in}\ e_2]\!]_\rho &= (\lambda \mathcal{C}[\![e_2]\!]_{x\cdot\rho})\, \mathcal{C}[\![e_1]\!]_\rho \\
\mathcal{C}[\![\mathsf{letrec}\ x = e_1\ \mathsf{in}\ e_2]\!]_\rho &= (\lambda \mathcal{C}[\![e_2]\!]_{x\cdot\rho})\, (\mu \mathcal{C}[\![e_1]\!]_{x\cdot\rho}) \\
\mathcal{C}[\![\mathsf{match}\ e\ \mathsf{with}\ a]\!]_\rho &= \mathcal{C}[\![e]\!]_\rho \star \mathcal{A}[\![a]\!]_\rho \\
\mathcal{C}[\![x]\!]_\rho &= \underline{n} \quad \textit{où } \rho = x_0 \cdot \ldots \cdot x_{n-1} \cdot x \cdot \rho' \textit{ et } \forall i : x_i \neq x \\
\mathcal{C}[\![C]\!]_\rho &= \lambda^n C \quad \textit{où } n = \mathsf{ar}(C)
\end{aligned}$$

$$\mathcal{A}\left[\!\!\left[\begin{array}{ll} & C_1\, x_1^1 \ldots x_{n_1}^1 \to e_1 \\ \| & C_2\, x_1^2 \ldots x_{n_2}^2 \to e_2 \\ & \vdots \\ \| & C_m\, x_1^m \ldots x_{n_m}^m \to e_m \end{array}\right]\!\!\right]_\rho = \langle C_1 : \mathcal{C}[\![e_1]\!]_{x_{n_1}^1\cdot\ldots\cdot x_1^1\cdot\rho}, \ldots, C_m : \mathcal{C}[\![e_m]\!]_{x_{n_m}^m\cdot\ldots\cdot x_1^m\cdot\rho}\rangle$$

Exemple 4.4.5 *Le programme* $\mathsf{letrec}\ l = \mathsf{Cons\ Zero}\ l\ \mathsf{in\ match}\ l\ \mathsf{with\ Nil} \to \mathsf{Nil} \ \mid\ \mathsf{Cons}\ x\ l' \to l'$ *est compilé en*

$$(\lambda\, \underline{0} \star \langle \mathsf{Nil} : \mathsf{Nil}, \mathsf{Cons} :\ \underline{0}\,\rangle)(\mu(\lambda\lambda\mathsf{Cons})\ \mathsf{Zero}\ \ \underline{0}\,).$$

Théorème 4.4.6 *Les termes issus de programmes écrits dans $\mathcal{L}$ et compilés par $\mathcal{C}$ sont réduits par la machine jusqu'à leur forme normale de tête pour la stratégie CBNeedE.*

Démonstration. On donne le tableau suivant qui récapitule les différentes configurations de la machine. Chaque colonne correspond à un élément des états (M pour le code, e pour l'environnement, s pour la pile des arguments, d pour la pile des données, u pour la pile des mises à jour), et chaque ligne à un type d'état. ? signifie que la pile ou l'environnement en question est quelconque, ϵ que la pile en question est vide, et $\neq \epsilon$ qu'elle est non vide. Enfin, dans la colonne e, les indications correspondent à des conditions sur la taille de l'environnement. Par exemple $> n$ signifie que l'environnement a une taille strictement supérieure à n.

	M	e	s	d	u	Règle
1	MN	?	?	?	?	(App)
2	λM	?	$\neq \epsilon$	?	?	(Lam)
3	λM	?	ϵ	ϵ	$\neq \epsilon$	(Update)
4	λM	?	ϵ	$\neq \epsilon$	?	aucune
5	λM	?	ϵ	ϵ	ϵ	aucune
6	$\underline{n}$	$> n$	?	?	?	(Access)
7	$\underline{n}$	$\leq n$	?	?	?	aucune
8	$M \star \langle \vec{C} : \vec{N} \rangle$	?	?	?	?	(Data)
9	C	?	$\neq \epsilon$	?	?	aucune
10	C	$\geq \mathsf{ar}(C)$	ϵ	$\neq \epsilon$	?	(Select)
11	C	?	ϵ	ϵ	$\neq \epsilon$	(UpdateData)
12	C	?	ϵ	ϵ	ϵ	aucune
13	C	$< \mathsf{ar}(C)$	ϵ	$\neq \epsilon$	?	aucune
14	μM	?	ϵ	ϵ	$\neq \epsilon$	(Rec)
15	μM	?	$\neq \epsilon$	?	?	aucune

Les différents cas d'arrêt sont donc les suivants :
- Cas 4 : cet état représente bien une forme normale pour la stratégie CBNeedE.
- Cas 5 : cet état représente bien une forme normale pour la stratégie CBNeedE.
- Cas 7 : cet état n'est pas atteignable dans le cas des termes clos.
- Cas 9 : cet état représente bien une forme normale pour la stratégie CBNeedE.
- Cas 12 : cet état représente bien une forme normale pour la stratégie CBNeedE.
- Cas 13 : cet état représente bien une forme normale pour la stratégie CBNeedE.
- Cas 15 : cet état n'est pas atteignable ; ce dernier point est un peu subtil : un terme de la forme μM est forcément placé en position d'argument dans le terme initial (voir le schéma de compilation). Par conséquent, il sera toujours accédé après un (Access) qui sauvegarde les piles dans un bloc de mise à jour et les remplace par des piles vides.

$\square$

4.5 Conclusion

Nous avons prouvé la correction de la machine KP enrichie avec les structures de données et la récursion. Outre celle de Pierre Crégut, d'autres preuves de correction de machines abstraites existent dans la littérature : Hardin, Maranget et Pagano ont démontré la correction de plusieurs machines abstraites grâce au calcul $\lambda\sigma_w$ [40], notamment la machine de Krivine, la SECD [49], la FAM [18] et la CAM [21]. Cependant, ces machines ne mettent en œuvre aucun partage, car elles implantent les stratégies call-by-value et call-by-name. Les démonstrations ne font donc pas état de la stratégie de partage mise en œuvre comme c'est le cas ici pour KP. D'autre part, ces machines sont présentées sans les aspects de récursion et structures de données. Sestoft [78] a donné également une preuve de KP qui se base sur la sémantique naturelle de Launchbury pour l'évaluation paresseuse [53]. Cette preuve fait donc bien état de la stratégie utilisée, mais la sémantique en question n'est pas générique comme $\lambda\sigma_w^a$, puisqu'elle ne permet pas de décrire d'autres stratégies que l'évaluation paresseuse. De fait, elle ne permet de comparer des stratégies avec la même finesse que $\lambda\sigma_w^a$, qui permet notamment de distinguer (et de définir de manière formelle) machines à environnement et réducteurs de graphes. Nous espérons avoir convaincu le lecteur que $\lambda\sigma_w^a$ est un outil adapté à une description fine des machines abstraites et des stratégies qu'elles mettent en œuvre, quelles qu'elles soient et plus particulièrement les stratégies d'appel par nécessité (call-by-need) dont la description est habituellement complexe. De plus, la simplicité de cette démonstration met en évidence la commodité de l'outil $\lambda\sigma_w^a$.

Chapitre 5

Super-clôtures

Les machines abstraites des langages de programmation fonctionnelle non stricts –c'est-à-dire soit en appel par nom [72], soit en appel par nécessité [53]– utilisent des *clôtures* pour enregistrer les arguments en attente d'évaluation [23, 24, 30, 71]. Une clôture est constituée d'un pointeur vers du code, et d'un environnement qui est une photographie – éventuellement restreinte aux éléments utiles – de l'environnement du programme tel qu'il était quand la clôture a été construite. Par conséquent, il y a autant de créations de clôtures dans un environnement particulier que d'arguments en attente d'évaluation rencontrés dans cet environnement précis. Nous pensons que ce nombre peut être élevé dans certaines applications, par exemple des programmes générés automatiquement, ou des programmes manipulant de nombreux paramètres (systèmes d'exploitation, déduction automatique) et qu'en conséquence une telle stratégie peut être très coûteuse. Nous proposons donc d'exploiter cette observation pour proposer d'éventuelles optimisations pour ces machines.

Pour présenter nos idées, nous reprenons la machine de Krivine [46], décrite au chapitre 4. C'est une machine simple qui n'a pas la prétention de servir de base à la description d'implantations réalistes et efficaces puisque par exemple le partage, les structures de données, la récursion ne sont pas décrits. Cependant sa simplicité nous permet de décrire ce qui est essentiel dans notre travail. Ensuite, nous mettons en œuvre le partage et nous discutons les problèmes qui y sont liés.

Ce chapitre est organisé en 5 sections. Après quelques préliminaires dans la section 5.1, nous présentons l'idée de base des super-clôtures dans la section 5.2. La section 5.3 montre comment la machine de Krivine peut être modifiée pour mettre en œuvre les super-clôtures. Nous verrons que cette machine soulève une difficulté, la division de super-clôtures. Dans la section 5.4, nous proposons deux solutions pour résoudre cette difficulté : la première est basée sur une transformation de programmes inspirée de l'expansion en forme normale η longue du λ-calcul. Malheureusement, cette transformation n'est applicable qu'aux termes simplement typés. Cela est trop restrictif dans le cas de langages dans lesquels le polymorphisme de types est réellement nécessaire pour une bonne utilisabilité. La seconde solution est plus satisfaisante et met en œuvre ce que nous nommons les fenêtres d'accès. Nous proposons dans la section 5.5 une version de cette dernière machine dans laquelle l'évaluation est partagée. Nous montrons enfin dans la section 5.7 que toutes nos machines sont correctes vis à vis des stratégies respectives, appel par nom et appel par valeur, et qu'elles permettent de réduire les termes jusqu'à leur forme normale de tête faible. Nous conclurons à la section 5.8.

Ce travail est le résultat d'une collaboration avec Zine-el-Abidine Benaissa [52].

5.1 Préliminaires

Nous précisons dans cette section des notations qui sont locales à ce chapitre. En particulier, nous utilisons les vecteurs. Un vecteur d'éléments de type A (typiquement $a, b, c, \ldots, a_1, \ldots, a_n$) peut être noté $\vec{a}$ (vecteur quelconque), $\{a_1; \ldots; a_n\}$ (vecteur quelconque de taille n, éventuellement vide), ou $\vec{a}_i^j$ qui est équivalent à $\{a_i; a_{i+1}; \ldots; a_j\}$ ou vide si $j < i$. Le vecteur vide peut être également noté ϵ. Les vecteurs peuvent être *concaténés* tels que $\vec{a}_0^i + \vec{a}_{i+1}^n = \vec{a}_0^n$ ou des éléments peuvent être ajoutés en tête tels que $a_0 \cdot \vec{a}_1^n = \vec{a}_0^n$. L'accès à l'élément placé en position n d'un vecteur est noté $\vec{a}(n)$ et défini par $\vec{a}_i^{i+n+m}(n) = a_{i+n}$. La modification de l'élément placé en position n dans un vecteur est notée $\vec{a}(n \leftarrow b)$ et définie par $\vec{a}_i^{i+n+m}(n \leftarrow b) = \{a_i; \ldots; a_{i+n-1}; b; a_{i+n+1}; \ldots; a_{i+n+m}\}$.

Nous utiliserons dans ce chapitre les λ-termes (de de Bruijn) en *notation vectorisée* : le λ-terme $MN_0 \ldots N_n$ où M n'est pas une application est noté $M\vec{N}_0^n$. Les abstractions sont également groupées, et le terme $\underbrace{\lambda \ldots \lambda}_{n} M$ où M n'est pas une abstraction est noté $\lambda^n M$. $\lambda^0 M$ représente donc un λ-terme M qui n'est pas une abstraction.

Enfin, nous renvoyons le lecteur a la définition de mapping donnée au chapitre 4. On peut écrire $[\vec{a}_0^n \mapsto \vec{b}_0^n]$ pour le mapping $[a_0 \mapsto b_0; \ldots; a_n \mapsto b_n]$, et $\rho\, \vec{a}_0^n = \{\rho\, a_0; \ldots; \rho\, a_n\}$.

5.2 Idée de base

Comme on peut le constater sur la machine de Krivine (voir le chapitre 4 pour une description de la machine de Krivine), une application de n arguments provoque la création de n clôtures, ce qui est illustré par la dérivation suivante :

$$(MN_1 \ldots N_n[e]\ ;\ s) \twoheadrightarrow (M[e]\ ;\ N_1[e] \cdot \ldots \cdot N_n[e] \cdot s)$$

Ces clôtures ont toutes la même structure, sauf au niveau du code. C'est également le cas dans la plupart des machines existantes, de nombreuses structures de données similaires peuvent être créées. Nous pensons que ce travail pourrait être épargné. Par exemple, dans STG, une liaison let de n expressions implique la création de n clôtures. Nous voulons donc étudier le comportement d'une machine dans laquelle ces clôtures sont groupées dans une seule structure, que nous appelons *super-clôture*. Par conséquent notre objectif principal n'est pas de partager e (même si en fait il est partiellement partagé) mais de faire de la création de n clôtures une tâche unique qui peut être effectuée dans un temps qui ne dépend pas de n.

5.3 Une première machine utilisant les super-clôtures

Définition 5.3.1 (Super-clôture) *Une* super-clôture *est une structure de données notée* $\langle \vec{N}, e \rangle$ *où* $\vec{N}$ *est un* vecteur de code *(non vide), et* e *un environnement.*

Les super-clôtures peuvent apparaître dans la pile ou dans les environnements. Dans la pile, $\langle \vec{N}_0^n, e \rangle$ peut être interprété comme $N_0[e] \cdot \ldots \cdot N_n[e]$ dans la machine de Krivine, et dans un environnement comme $N_n[e] \cdot \ldots \cdot N_0[e]$. La SC-machine est définie sur la figure 5.1. Elle réduit des λ-termes en notation vectorisée. Si on la compare avec la machine de Krivine, on remarque que seule une super-clôture est construite au lieu de n clôtures. Cela signifie que nous épargons n unités de temps. De plus, l'environnement e n'est instancié qu'une seule fois pour les n arguments.

La SC-machine est donc une machine dans laquelle les environnements sont (au moins partiellement) partagés. En cela, elle doit être comparée avec la TIM [30] qui également partage les

$$
\begin{aligned}
(M\vec{N}[e]\ ;\ s) &\to (M[e]\ ;\ \langle \vec{N}, e\rangle \cdot s) && \text{(App)}\\
(\lambda^{n+1} M[e]\ ;\ \langle \vec{N}_0^{n+m+1}, f\rangle \cdot s) &\to (M[\langle \vec{N}_0^{m}, f\rangle \cdot e]\ ;\ \langle \vec{N}_{n+1}^{n+m+1}, f\rangle \cdot s) && \text{(Split)}\\
(\lambda^{n+m+1} M[e]\ ;\ \langle \vec{N}_0^{m}, f\rangle \cdot s) &\to (\lambda^{n} M[\langle \vec{N}_0^{m}, f\rangle \cdot e]\ ;\ s) && \text{(Lam)}\\
(\,\underline{n}\,[\langle \vec{N}_0^{n+m}, f\rangle \cdot e]\ ;\ s) &\to (N_m[f]\ ;\ s) && \text{(Access)}\\
(\,\underline{n+m+1}\,[\langle \vec{N}_0^{m}, f\rangle \cdot e]\ ;\ s) &\to (\,\underline{n}\,[e]\ ;\ s) && \text{(Skip)}
\end{aligned}
$$

FIG. 5.1 – La machine SC

environnements. Cependant, il existe des différences importantes entre les deux approches. D'abord, notre machine hérite de la machine de Krivine la possibilité de réduire tous les termes alors que la TIM est restreinte à la réduction des supercombinateurs. Cependant, on peut imaginer des extensions de la TIM qui peuvent réduire des termes quelconques, en chaînant les environnement par exemple. Les deux machines resteraient toutefois fort différentes. Dans la SC-machine, les environnements sont partagés par le biais des super-clôtures, alors que dans la TIM, ils sont partagés par le biais d'un tas. C'est ce tas qui permet le partage de l'évaluation, alors que nous avons toujours besoin d'un tas de (super)-clôtures, le partage des environnements étant local aux super-clôtures. En fait, comme nous le verrons dans la section 5.5, nos environnements sont partagés avec des limitations.

L'optimisation permise par les super-clôtures pour l'appel par nom avec la machine de Krivine est donc évidente, particulièrement en terme de nombre de pas de réduction. Cependant, certains pas de réduction sont maintenant plus coûteux, puisque par exemple les règles (Lam) et (Split) impliquent qu'une abstraction n-aire vérifie si la super-clôture qui est située en sommet de pile est de taille plus petite que n. Pour la même raison, l'accès aux variables est plus coûteux, puisqu'il peut ne pas être effectué en un seul pas (Access, Skip). Enfin, nos super-clôtures ne permettent pas une optimisation par ailleurs classique qui permet d'éviter la création de clôtures du type $\underline{n}\,[\ldots \cdot M_n[e_n] \cdot \ldots]$, en les remplaçant directement par la clôture déjà allouée $M_n[e_n]$. Ce manque peut être un défaut car cette optimisation permet d'éviter beaucoup de mise à jour inutiles.

De plus, nous sommes bien conscients que notre vision est encore trop simpliste pour une bonne analyse de l'appel par nécessité. En effet, un des problèmes pour implanter le partage réside dans la règle (Split). Supposons qu'une abstraction demande moins d'arguments que ceux disponibles dans la super-clôture en sommet de pile. Cette situation est très courante lorsqu'on utilise l'ordre supérieur. La super-clôture doit être divisée (*split*) pour que seul le nombre requis d'arguments soit déplacé vers l'environnement. Le risque est que les super-clôtures soient divisée en super-clôtures dont le vecteur de code n'a qu'un élément, c'est-à-dire en clôtures ! Ce n'est pas ce que l'on veut. Nous voudrions plutôt créer des clôtures seulement quand cela s'avère nécessaire, c'est-à-dire seulement lorsqu'on a besoin d'y accèder. De plus, à cause de la règle (Split), une super-clôture ne peut pas être partagée puisqu'elle peut être modifiée après sa création, dans un environnement particulier. Nous présentons dans la section suivante deux solutions pour éviter les inconvénients d'une telle règle.

5.4 Deux solutions pour éviter la division de super-clôture

Nous proposons deux solutions pour éviter la division de super-clôture : la première est basée sur une transformation statique des programmes inspirée de l'expansion en forme normale η longue du λ-calcul. Cependant, cette solution n'est pas satisfaisante dans le cas d'un langage avec types polymorphes, et nous donnons une autre solution qui utilise les *fenêtres d'accès.*

Première tentative : transformation statique

L'idée est de faire en sorte que les programmes n'aient jamais à utiliser la règle (Split). Il suffit pour cela d'imposer aux abstractions de prendre autant ou plus d'arguments que ceux qui sont dans la super-clôture en sommet de pile. En d'autres termes, il faut interdire les termes de la forme $(\lambda^n M)\vec{N}^{n+m+1}$ où $n > 0$. Une telle transformation existe-t-elle ?

Dans le λ-calcul simplement typé, il existe une transformation nommée *expansion en forme normale η longue* [79], qui est une transformation dirigée par le type des termes. Elle est définie sur les termes en β-forme normale, de façon à permettre d'unifier les β-formes normales, modulo η-conversion. Si $\lambda^n\, \underline{m}\, \vec{N}_1^p$ est un terme simplement typé d'arité $n+q$ en β-forme normale, alors sa forme normale η longue (de tête) est le terme

$$\lambda^{n+q}\, \underline{m+q}\ \{\tau_0^q(N_1); \dots ; \tau_0^q(N_p);\ \underline{q-1}\, ; \dots ;\ \underline{0}\,\}$$

Cette transformation nous inspire pour définir une nouvelle transformation, généralisée à tous les λ-termes simplement typés, et non plus seulement aux β-formes normales. Nous l'appelons *expansion en forme normale η longue généralisée*, que nous noterons gel[1].

Définition 5.4.1 (expansion en forme normale η longue généralisée) *Soit M un terme simplement typé, on définit $\mathsf{gel}(M)$ par*

- *si M est de la forme $\lambda^n N \vec{N}_0^p$ d'arité $n+q$, alors*

$$\mathsf{gel}(M) = \lambda^{n+q}\mathsf{gel}(\tau_0^q(N))\,\{\mathsf{gel}(\tau_0^q(N_0)); \dots ; \mathsf{gel}(\tau_0^q(N_p));\ \underline{q-1}\, ; \dots ;\ \underline{0}\,\}$$

- *si M est de la forme $\lambda^{n+1}\,\underline{m}\,$ d'arité $n+q+1$, alors*

$$\mathsf{gel}(M) = \lambda^{n+q+1}\,\underline{m+q}\ \{\,\underline{q-1}\, ; \dots ;\ \underline{0}\,\}$$

- *si $M = \underline{n}\,$, alors $\mathsf{gel}(M) = \underline{n}\,$.*

où τ_i^n est la fonction de renommage définie à la section 1.4, et adaptée de manière évidente à la notation vectorisée.

> *Remarque.* Comme on peut le constater, les variables ne sont pas expansées quand elles ne constituent pas le corps d'une abstraction. Cette expansion est en effet inutile pour l'effet que nous recherchons. Une variable est simplement un élément de code qui permet d'accéder à une clôture et n'a aucun effet sur les opérations sur la pile. Par conséquent, gel n'est pas *stricto sensu* une généralisation de l'expansion en forme normale η longue. □

Lemme 5.4.2 gel *est une fonction totale sur l'ensemble des termes simplement typés.*

Démonstration. Soit M un λ-terme simplement typé. On étudie tous les cas possibles :

- si M est une application $N\vec{N}_0^n$, alors le premier cas de la définition 5.4.1 s'applique.

[1] gel pour *generalized expansion to η long normal form.*

– si M est une abstraction $\lambda^{n+1}M$, alors soit M est une variable, et le deuxième cas de la définition 5.4.1 s'applique, soit M est une application, et c'est le premier cas qui s'applique. On rappelle que d'après la définition des termes en notation vectorisée, M ne peut pas être une abstraction.
– si M est une variable, c'est le troisième cas de la définition 5.4.1 qui s'applique.

□

Exemple 5.4.3 *Soit le terme $(\lambda\,\underline{0})\{\lambda\,\underline{0};\lambda\,\underline{0}\}$ de type $\iota\to\iota$ où ι est un type de base.*

$$\begin{aligned}\mathsf{gel}((\lambda\,\underline{0})\{\lambda\,\underline{0};\lambda\,\underline{0}\}) &= \mathsf{gel}(\lambda\,\underline{0})\,\{\mathsf{gel}(\lambda\,\underline{0});\mathsf{gel}(\lambda\,\underline{0})\}\\ &= (\lambda^3\mathsf{gel}(\,\underline{2}\,)\{\,\underline{1}\,;\,\underline{0}\,\})\{\lambda^2\mathsf{gel}(\,\underline{1}\,)\{\,\underline{0}\,\};\lambda\,\underline{0}\}\\ &= (\lambda^3\,\underline{2}\,\{\,\underline{1}\,;\,\underline{0}\,\})\{\lambda^2\,\underline{1}\,\{\,\underline{0}\,\};\lambda\,\underline{0}\}\end{aligned}$$

Notre nouvelle machine, nommée SCη-machine est définie sur les termes en gel. Elle est définie sur la figure 5.2. En fait, il s'agit non moins que de la SC-machine de laquelle on a éliminé la règle (Split).

$$\begin{aligned}(M\vec{N}[e]\ ;\ s) &\to (M[e]\ ;\ \langle\vec{N},e\rangle\cdot s) && \text{(App)}\\ (\lambda^{n+m+1}M[e]\ ;\ \langle\vec{N}^{n+1},f\rangle\cdot s) &\to (\lambda^m M[\langle\vec{N}^{n+1},f\rangle\cdot e]\ ;\ s) && \text{(Lam)}\\ (\,\underline{n}\,[\langle\vec{N}_0^{n+m},e\rangle\cdot f]\ ;\ s) &\to (N_m[e]\ ;\ s) && \text{(Access)}\\ (\,\underline{n+m+1}\,[\langle\vec{N}_0^{m},e\rangle\cdot f]\ ;\ s) &\to (\,\underline{n}\,[f]\ ;\ s) && \text{(Skip)}\end{aligned}$$

FIG. 5.2 – La machine SCη

Malheureusement cette machine, ou du moins la transformation qu'elle impose n'est pas pleinement satisfaisante. En effet, comme nous l'avons déjà mentionné, elle est conçue uniquement pour réduire des termes simplement typés. Cela signifie que notre machine ne peut pas être utilisée comme base d'implantation d'un langage avec polymorphisme de type. Il est pourtant bien connu que la programmation fonctionnelle typée sans polymorphisme de types n'est pas assez expressive pour être d'un grand intérêt. (Voir [55] pour une bonne introduction au polymorphisme de type dans les langages de programmation fonctionnelle.) La question est donc à présent de savoir s'il existe une transformation similaire pour un système de types plus puissant, en particulier un système de types polymorphes ? La réponse est certainement non, puisqu'il est impossible de connaître l'arité d'une expression de type $\alpha_0\to\ldots\to\alpha_n$ où α_n est une variable de type. De plus, il est important de remarquer que toute expression de type fonctionnel (c'est-à-dire d'arité non nulle) est transformée en une abstraction, c'est-à-dire une valeur. Par conséquent son évaluation par la machine est triviale ! Cela peut, en outre, avoir un effet néfaste sur le partage, puisqu'une application $M\vec{M}_0^m$ d'arité $n+1$ dont l'évaluation pourrait être partagée va être transformée en une abstraction $\lambda^{n+1}M'(\vec{M'}_0^m+\{\vec{n};\ldots;\vec{0}\})$ qui est en forme normale de tête faible. Par conséquent, la valeur de $M\vec{M}_0^m$ ne peut plus être partagée.

Exemple 5.4.4 *La valeur de $(\lambda\,\underline{0})\{\lambda\,\underline{0};\lambda\,\underline{0}\}$ est $\lambda\,\underline{0}$. Dans la SC-machine, c'est cette valeur qui serait partagée si le terme ci-dessus était utilisé plusieurs fois. Dans le cas de la SCη-machine c'est le terme $(\lambda^3\,\underline{2}\,\{\,\underline{1}\,;\,\underline{0}\,\})\{\lambda^2\,\underline{1}\,\{\,\underline{0}\,\};\lambda\,\underline{0}\}$ (sa* gel*) qui sera partagé. Donc il sera réévalué à chacune de ses utilisations, qui dépend du contexte.*

Deuxième tentative : les fenêtres d'accès

Nous proposons ici une alternative nommée *fenêtres d'accès.* L'idée est que dans la règle (Split) il n'est pas nécessaire de diviser physiquement la super-clôture. Seul l'accès à la super-clôture doit être restreint. Par conséquent une super-clôture peut être partagée, chaque occurrence d'une super-clôture étant spécifiée par les bornes entre lesquelles l'accès est permis dans son vecteur de code. Nous appelons ces bormes la *fenêtre d'accès.* Dans ce qui suit, nous n'allons pas encore exprimer le partage, et nous définissons à nouveau une machine où *a priori* les super-clôtures sont copiées ; mais nous gardons à l'esprit que le but ultime de cette étude est l'implantation, en invitant le lecteur à voir les super-clôtures comme des pointeurs vers les super-clôtures physiques, et d'accepter de patienter jusqu'à la section 5.5 où les aspects de partage seront traités en détail.

Définition 5.4.5 (Fenêtre d'accès) *Soit $\langle \vec{N}_0^{n+i+j}, e\rangle$ une super-clôture sur laquelle nous voulons restreindre l'accès à la partie $\langle \vec{N}_i^{i+j}, e\rangle$. Nous la notons $\langle \vec{N}_0^{n+i+j}, e\rangle_i^{i+j}$ où i^{i+j} est la* fenêtre d'accès *de la super-clôture. La figure 5.3 donne les règles de la machine à fenêtres.*

$$
\begin{aligned}
(M\vec{N}_0^n[e]\ ;\ s) &\to (M[e]\ ;\ \langle \vec{N}_0^n, e\rangle_0^n \cdot s) && \text{(App)}\\
(\lambda^{n+1}M[e]\ ;\ \langle \vec{N}, f\rangle_i^{i+n+m+1} \cdot s) &\to (M[\langle \vec{N}, f\rangle_i^{i+n} \cdot e]\ ;\ \langle \vec{N}, f\rangle_{i+n+1}^{i+n+m+1} \cdot s) && \text{(Split)}\\
(\lambda^{n+m+1}M[e]\ ;\ \langle \vec{N}, f\rangle_i^{i+n} \cdot s) &\to (\lambda^m M[\langle \vec{N}, f\rangle_i^{i+n} \cdot e]\ ;\ s) && \text{(Lam)}\\
(\underline{n}\,[\langle \vec{N}, e\rangle_i^{i+n+m} \cdot f]\ ;\ s) &\to (N_{i+m}[e]\ ;\ s) && \text{(Access)}\\
(\underline{n+m+1}\,[\langle \vec{N}, f\rangle_i^{i+m} \cdot e]\ ;\ s) &\to (\underline{n}\,[e]\ ;\ s) && \text{(Skip)}
\end{aligned}
$$

FIG. 5.3 – La machine SC à fenêtres

Exemple 5.4.6 *Nous donnons l'exemple de la réduction du terme $(\lambda\,\underline{0})\{\lambda\,\underline{0}\,;\lambda\,\underline{0}\,\}$ par la SC-machine à fenêtres :*

$$
\begin{aligned}
((\lambda\,\underline{0})\{\lambda\,\underline{0}\,;\lambda\,\underline{0}\,\}[\epsilon]\ ;\ \epsilon) &\to (\lambda\,\underline{0}\,[\epsilon]\ ;\ \langle\{\lambda\,\underline{0}\,;\lambda\,\underline{0}\,\},\epsilon\rangle_0^1 \cdot \epsilon) && \text{(App)}\\
&\to (\,\underline{0}\,[\langle\{\lambda\,\underline{0}\,;\lambda\,\underline{0}\,\},\epsilon\rangle_0^0 \cdot \epsilon]\ ;\ \langle\{\lambda\,\underline{0}\,;\lambda\,\underline{0}\,\},\epsilon\rangle_1^1 \cdot \epsilon) && \text{(Split)}\\
&\to (\lambda\,\underline{0}\,[\epsilon]\ ;\ \langle\{\lambda\,\underline{0}\,;\lambda\,\underline{0}\,\},\epsilon\rangle_1^1 \cdot \epsilon) && \text{(Access)}\\
&\to (\,\underline{0}\,[\langle\{\lambda\,\underline{0}\,;\lambda\,\underline{0}\,\},\epsilon\rangle_1^1 \cdot \epsilon]\ ;\ \epsilon) && \text{(Lam)}\\
&\to (\lambda\,\underline{0}\,[\epsilon]\ ;\ \epsilon) && \text{(Access)}
\end{aligned}
$$

Les fenêtres d'accès sont sans doute une meilleure solution que la transformation en forme normale η-longue généralisée, puisqu'elles ne provoquent pas de perte dans le partage qui peut être réalisé, et qu'elles ne sont pas liées à un système de types particulier. Cependant, les fenêtre d'accès ont un coût, puisqu'elles augmentent la taille des environnements, chaque référence aux arguments d'une fonction nécessitant deux entiers (la fenêtre d'accès) en plus de la référence de la super-clôture. Il est donc évident que le gain en espace ne peut concerner que des fonctions prenant un grand nombre d'arguments.

5.5 Partage

Dans cette section, nous ajoutons le partage à la SC-machine à fenêtres. Pour cela, nous ajoutons aux états une notion d'*état mémoire*, ou *tas*, nous manipulons des adresses qui permettent de partager des arguments, avec une notion de *mise à jour*. Nous utilisons en fait la même technique que par exemple [71] ou [23], excepté que nous utilisons les super-clôtures.

Les états $(M[e] \ ; \ s \ ; \ u \ ; \ h_1 \ ; \ h_2)$ ont donc à présent 6 composants :
– le *code* M dont la structure est la même que pour les machines précédentes ;
– l'*environnement* e, un vecteur d'*adresses avec fenêtres d'accès* ;
– la *pile des arguments* s, comme l'environnement, mais en pile ;
– la *pile des mise-à-jour* u, une pile de *blocs de mise-à-jour* ;
– le *tas des super-clôtures* h_1, un mapping des adresses vers les super-clôtures ;
– le *tas des clôtures* h_2, un mapping des adresses vers les clôtures.

Les fenêtres d'accès sont maintenant associées à des *adresses* de super-clôtures. a_i^j représente la super-clôture à l'adresse a, restreinte à la fenêtre d'accès ${}_i^j$. Pour le partage, nous avons besoin de manipuler des clôtures *et* des super-clôtures, toutes stockées dans le tas. Pour des raisons d'homogénéité des structures de données, nous définissons donc deux tas : le tas des clôtures et le tas des super-clôtures. En effet, une super-clôture est une structure de données qui représente plusieurs clôtures partageant le même environnement. Quand un élément d'une super-clôture est évalué, il doit être mis à jour dans le tas. Son environnement, qui a changé suite à l'évaluation, ne peut plus être partagé avec les autres éléments de la super-clôture. Il faut donc créer une clôture, et une indirection de la super-clôture vers la clôture pour permettre le partage. Par conséquent, les super-clôtures sont maintenant un peu différentes : elles sont constituées d'un vecteur dont les éléments sont du code et des adresses de clôtures. Les éléments de cette union sont notés $l, l_1, \ldots, l_n, \ldots$

Nous avons besoin d'une opération sur les états qui fournit des adresses *fraîches*, ce qui signifie intuitivement *libres*. On ne définit pas précisément cette fonction, et on note *a fraîche* pour signifier que dans l'état courant a est une variable fraîche.

> *Remarque.* Nous aurions pu définir a fraîche dans un état $(M[e] \ ; \ s \ ; \ u \ ; \ h)$ par $a \notin \mathrm{dom}(h)$. Cependant cette définition est restrictive, car elle enlève la possibilité de réallouer une adresse devenue inutile mais à laquelle est associée une clôture (ou une super-clôture). □

Nous donnons dans un premier temps les règles simples, qui sont obtenues immédiatement par traduction des règles de la SC-machine à fenêtres sans partage :

$$(M\vec{N}_0^n[e] \ ; \ s \ ; \ u \ ; \ h_1 \ ; \ h_2) \to (M[e] \ ; \ a_0^n \cdot s \ ; \ u \ ; \ h_1[a \mapsto \langle \vec{N}_0^n, e\rangle] \ ; \ h_2) \qquad a \text{ fraîche} \qquad \text{(App)}$$

$$(\lambda^{n+1}M[e] \ ; \ a_i^{i+n+m+1} \cdot s \ ; \ u \ ; \ h_1 \ ; \ h_2) \to (M[a_i^{i+n} \cdot e] \ ; \ a_{i+n+1}^{i+n+m+1} \cdot s \ ; \ u \ ; \ h_1 \ ; \ h_2) \qquad \text{(Split)}$$

$$(\lambda^{n+m+1}M[e] \ ; \ a_i^{i+n} \cdot s \ ; \ u \ ; \ h_1 \ ; \ h_2) \to (\lambda^m M[a_i^{i+n} \cdot e] \ ; \ s \ ; \ u \ ; \ h_1 \ ; \ h_2) \qquad \text{(Lam)}$$

$$(\underline{n+m+1}\,[a_i^{i+m} \cdot e] \ ; \ s \ ; \ u \ ; \ h_1 \ ; \ h_2) \to (\underline{n}\,[e] \ ; \ s \ ; \ u \ ; \ h_1 \ ; \ h_2) \qquad \text{(Skip)}$$

Les règles suivantes sont moins triviales. Elles décrivent l'accès et la mise à jour des données. Tout d'abord, nous donnons la règle pour accéder à un argument qui partage son environnement dans une super-clôture :

$$(\,\underline{n}\,[a_i^{i+n+m}\cdot f]\ ;\ s\ ;\ u\ ;\ h_1\ ;\ h_2) \to (N[e]\ ;\ \epsilon\ ;\ (s,a,i+m)\cdot u\ ;\ h_1\ ;\ h_2) \qquad \text{(Access1)}$$
$$\text{où } h_1\ a =\ \langle\vec{l},e\rangle$$
$$\text{et } \vec{l}(i+m) = N$$

Comme l'argument peut être mis à jour, on empile sur la pile des mises à jour un bloc de mise à jour contenant la référence en mémoire de l'argument en cours d'évaluation. Cependant, on peut imaginer que la super-clôture n'est pas modifiable, c'est-à-dire que toute mise à jour est inutile. C'est le cas par exemple si la clôture est déjà une valeur, ou si on n'y accède qu'une seule fois [54]. Dans ce cas, on peut imaginer l'optimisation suivante :

$$(\,\underline{n}\,[a_i^{i+n+m}\cdot f]\ ;\ s\ ;\ u\ ;\ h_1\ ;\ h_2) \to (N[e]\ ;\ s\ ;\ u\ ;\ h_1\ ;\ h_2) \qquad \text{(Access1')}$$
$$\text{où } h_1\ a = \langle\vec{l},e\rangle$$
$$\vec{l}(i+m) = N \text{ non modifiable}$$

Quoi qu'il en soit, dans (Access1) ou dans (Acces1'), l'environnement e doit être copié, puisqu'il va être modifié par l'évaluation de l'argument N. Cela pourrait être un autre défaut de notre machine, car jusqu'à maintenant nous avons essayé d'éviter de copier des environnements. Cependant, puisque ces copies d'environnements sont effectuées *par nécessité*, nous pouvons espérer que cela arrivera moins souvent que dans la machine sans super-clôtures.

Les blocs de mise à jour doivent être dépilés de la pile des mise à jour quand l'évaluation a atteint une valeur, ce qui est décrit par la règle (Update). Soit $\lambda^{n+1}M$ la valeur de l'argument. Ici, il faut créer une clôture pour la mise à jour, car l'environnement a changé par rapport à l'environnement partagé :

$$(\lambda^{n+1}M[e]\ ;\ \epsilon\ ;\ (s,a,i)\cdot u\ ;\ h_1\ ;\ h_2) \to (\lambda^{n+1}M[e]\ ;\ s\ ;\ u\ ;\ h_1[a\mapsto\langle\vec{l'},f\rangle]\ ;\ h_2\big[a'\mapsto\lambda^{n+1}M[e]\big])$$
$$a' \text{ fraîche} \qquad \text{(Update)}$$
$$\text{où } h_1\ a = \langle\vec{l},f\rangle$$
$$\text{et } \vec{l'} = \vec{l}(i\leftarrow a')$$

Dans cette dernière règle, l'environnement est enore une fois copié puisqu'il va être modifié dans le contexte d'évaluation particulier de la fonction $\lambda^{n+1}M$. C'est une des différences principales avec la TIM, dans laquelle les environnements sont toujours maintenus partagés.

Comme cela est suggéré par (Update), il y a maintenant un deuxième cas à prendre en compte lorsqu'on accède un argument : le cas d'une indirection vers une clôture. Si une clôture est déjà créée pour l'argument à évaluer, alors on peut considérer que celui-ci est déjà mis à jour, et qu'il est une valeur. D'où la règle :

$$(\,\underline{n}\,[a_i^{i+n+m}\cdot f]\ ;\ s\ ;\ u\ ;\ h_1\ ;\ h_2) \to (N[e]\ ;\ s\ ;\ u\ ;\ h_1\ ;\ h_2) \qquad \text{(Access2)}$$
$$\text{où } h_1\ a = \langle\vec{l},e\rangle$$
$$\vec{l}(i+m) = a'$$
$$\text{et } h_2\ a' = N[e]$$

Cette dernière règle suggère donc que pour réaliser un accès, la machine doit vérifier s'il y a une indirection ou pas. Ceci peut malheureusement induire une perte de performance sévère de la machine. De plus, l'indirection elle même est coûteuse puisqu'elle nécessite deux accès au tas au lieu d'un seul.

5.6 Un exemple de l'efficacité de la machine

Nous avons jusqu'ici mis en évidence les défauts de notre machine. Cependant, il y a également des cas où les super-clôtures peuvent apporter une certaine efficacité. C'est le cas notamment de programmes dans lesquels :

1. les fonctions ont un grand nombre d'arguments ;
2. beaucoup d'arguments soit ne sont pas nécessaires, soit ne nécessitent pas de mises à jour ;
3. ils contiennent beaucoup de variables libres.

En effet, lors de l'application d'une fonction à n arguments, une seule super-clôture est créée au lieu de n clôtures. De plus, le fait que des arguments ne soient pas nécessaires ou ne nécessitent pas de mises à jour implique que peu d'indirections vers des clôtures seront nécessaires. Enfin, les arguments contenant beaucoup de variables libres, cela signifie que les environnements contiennent les valeurs de ces variables, c'est-à-dire qu'ils sont volumineux. Ainsi, le gain obtenu en ne les dupliquant pas grâce aux super-clôtures n'est pas négligeable.

Illustrons cela par un exemple. Considérons le programme suivant, écrit dans un pseudo-langage avec opérateur de récursion, sélecteurs de cas, litéraux entiers et opérateurs prédéfinis sur les entiers :

$$
\begin{array}{l}
\text{select } n\ f_1\ f_2\ f_3\ f_4\ f_5\ f_d\ x = \mathsf{match}\ n\ \mathsf{with} \\
\qquad 1 \to f_1\,x \;\mid\; 2 \to f_2\,x \;\mid\; 3 \to f_3\,x \;\mid\; 4 \to f_4\,x \;\mid\; 5 \to f_5\,x \;\mid\; \_ \to f_d\,x
\end{array}
$$

$$
\begin{array}{l}
\text{nsel } n = \mathsf{let}\ y_1 = 1, y_2 = 2, y_3 = 3, y_4 = 4\ \mathsf{in\ let} \\
\qquad g_1\,x = y_1 + y_2x + y_3x^2 + y_4x^3, \\
\qquad g_2\,x = y_1 + y_2x + y_3x^2 - y_4x^3, \\
\qquad g_3\,x = y_1 + y_2x - y_3x^2 + y_4x^3, \\
\qquad g_4\,x = y_1 - y_2x + y_3x^2 + y_4x^3, \\
\qquad g_5\,x = -y_1 + y_2x + y_3x^2 + y_4x^3, \\
\qquad g_d\,x = y_4 + y_3x + y_2x^2 + y_1x^3 \\
\mathsf{in}\ \text{select } n\,g_1\,g_2\,g_3\,g_4\,g_5\,g_d\,2
\end{array}
$$

$$
\text{main} = (\text{nsel } 1) + (\text{nsel } 2) + (\text{nsel } 3)
$$

À l'exécution avec une machine classique, une clôture serait créée pour chaque argument de la fonction select. Cependant, chacun des arguments (les g_i) étant en fait une valeur, aucune mise à jour n'a besoin d'être effectuée. Par conséquent il n'y a aucune indirection de la super-clôture vers des clôtures. En fait, seule une super-clôture est créée pour l'ensemble des arguments de select. L'environnement constitué des valeurs des y_i n'est donc pas inutilement dupliqué.

5.7 Preuves de correction des machines

Nous donnons dans cette section les preuves de correction de nos machines. Nous démontrons qu'elles réalisent bien la β-réduction, et qu'elles réduisent les termes jusqu'en forme normale de tête faible. Pour cela, nous traduisons les états vers des termes (décompilation), et nous montrons que ces propriétés sont satisfaites. Pour les machines sans partage, nous utilisons le calcul $\lambda\sigma_w$, que nous savons correct par rapport au λ-calcul.

Correction de la sc-machine

Définition 5.7.1 *Nous définissons la fonction μ qui traduit les états de la* SC*-machine vers les termes de $\lambda\sigma_w$ comme suit :*

$$\begin{aligned}\mu(M[e]\ ;\ s) &= \kappa((M)[\rho(e)], s)\\ \text{où } \kappa(M,\epsilon) &= M\\ \kappa(M, \langle\vec{N}_0^n, f\rangle\cdot s) &= \kappa(M\,\vec{N}_0^n[\rho(f)], s)\end{aligned}$$

et ρ est la fonction de traduction des environnements de la SC*-machine vers les substitutions de $\lambda\sigma_w$:*

$$\begin{aligned}\rho(\epsilon) &= \mathrm{id}\\ \rho(\langle\vec{N}_0^n, e\rangle\cdot f) &= N_n[\rho(e)]\cdot\ldots\cdot N_0[\rho(e)]\cdot\rho(f)\end{aligned}$$

De plus, nous utilisons la traduction implicite des termes en notation vectorisée vers les termes en notation classique, et dans $\lambda\sigma_w$, $M\vec{N}_0^n[s]$ est une abbréviation pour $MN_0[s]\ldots N_n[s]$.

Nous avons besoin du lemme suivant pour établir la principale propriété :

Lemme 5.7.2 *Si $M \xrightarrow[\lambda\sigma_w]{} M'$ alors $\kappa(M,s) \xrightarrow[\lambda\sigma_w]{} \kappa(M',s)$.*

Démonstration. Par induction sur s :
- $\kappa(M,\epsilon) = M \xrightarrow[\lambda\sigma_w]{} M' = \kappa(M',\epsilon)$.
- $\kappa(M,\langle\vec{N}_0^n, e\rangle\cdot s) = \kappa(M\vec{N}_0^n[\rho(e)], s) \xrightarrow[\lambda\sigma_w]{} \kappa(M'\vec{N}_0^n[\rho(e)], s) = \kappa(M', \langle\vec{N}_0^n, e\rangle\cdot s)$

□

Théorème 5.7.3

$$\textit{Si } (M[e]\ ;\ s) \to (M'[e']\ ;\ s') \textit{ alors } \mu(M[e]\ ;\ s) \xrightarrow[\lambda\sigma_w]{}\!\!\!\!\!\to \mu(M'[e']\ ;\ s').$$

Démonstration. Par vérification systématique des règles de la SC-machine.
- Règle (App) :

$$\begin{aligned}\mu(M\vec{N}[e]\ ;\ s) &= \kappa((M\vec{N})[\rho(e)], s)\\ &\xrightarrow[\lambda\sigma_w]{}\!\!\!\!\!\to \kappa(M[\rho(e)]\vec{N}[\rho(e)], s) \quad \text{(par le lemme 5.7.2)}\\ &= \kappa(M[\rho(e)], \langle\vec{N}, e\rangle\cdot s)\\ &= \mu(M[e]\ ;\ \langle\vec{N}, e\rangle\cdot s)\end{aligned}$$

- Règle (Split) :

$$\begin{aligned}\mu(\lambda^{n+1}M[e]\ ;\ \langle\vec{N}_0^{n+m+1}, f\rangle\cdot s) &= \kappa((\lambda^{n+1}M)[\rho(e)], \langle\vec{N}_0^{n+m+1}, f\rangle\cdot s)\\ &= \kappa((\lambda^{n+1}M)[\rho(e)]\vec{N}_0^{n+m+1}[\rho(f)], s)\\ &\xrightarrow[\lambda\sigma_w]{}\!\!\!\!\!\to \kappa(M\big[N_n[\rho(f)]\cdot\ldots\cdot N_0[\rho(f)]\cdot\rho(e)\big]\vec{N}_{n+1}^{n+m+1}[\rho(f)], s)\\ &= \kappa(M[\rho(\langle\vec{N}_0^n, f\rangle\cdot e)], \langle\vec{N}_{n+1}^{n+m+1}, f\rangle\cdot s)\\ &= \mu(M[\langle\vec{N}_0^n, f\rangle\cdot e]\ ;\ \langle\vec{N}_{n+1}^{n+m+1}, f\rangle\cdot s)\end{aligned}$$

– Règle (Lam) :

$$\begin{aligned}\mu(\lambda^{n+m+1}M[e]\ ;\ \langle\vec{N}_0^m, f\rangle\cdot s) &= \kappa((\lambda^{n+m+1}M)[\rho(e)], \langle\vec{N}_0^m, f\rangle\cdot s)\\ &= \kappa((\lambda^{n+m+1}M)[\rho(e)]\vec{N}_0^m[\rho(f)], s)\\ &\xrightarrow[\lambda\sigma_w]{}\!\!\!\!\rightarrow \kappa((\lambda^n M)[N_m[\rho(f)]\cdot\ldots\cdot N_0[\rho(f)]\cdot\rho(e)], s)\\ &= \kappa((\lambda^n M)[\rho(\langle\vec{N}_0^m, f\rangle\cdot e)], s)\\ &= \mu(\lambda^n M[\langle\vec{N}_0^m, f\rangle\cdot e]\ ;\ s)\end{aligned}$$

– Règle (Access) :

$$\begin{aligned}\mu(\underline{n}\,[\langle\vec{N}_0^{n+m}, f\rangle\cdot e]\ ;\ s) &= \kappa(\underline{n}\,[N_{n+m}[\rho(f)]\cdot\ldots N_0[\rho(f)]\cdot\rho(e)], s)\\ &\xrightarrow[\lambda\sigma_w]{}\!\!\!\!\rightarrow \kappa(N_m[\rho(f)], s)\\ &= \mu(N_m[f]\ ;\ s)\end{aligned}$$

– Règle (Skip) :

$$\begin{aligned}\mu(\underline{n+m+1}\,[\langle\vec{N}_0^m, f\rangle\cdot e]\ ;\ s) &= \kappa(\underline{n+m+1}\,[N_m[\rho(f)]\cdot\ldots\cdot N_0[\rho(f)]\cdot\rho(e)], s)\\ &\xrightarrow[\lambda\sigma_w]{}\!\!\!\!\rightarrow \kappa(\underline{n}\,[\rho(e)], s)\\ &= \mu(\underline{n}\,[e]\ ;\ s)\end{aligned}$$

□

Pour montrer que la SC-machine réduit les termes jusqu'à leur forme normale de tête faible, on montre d'abord le lemme suivant :

Lemme 5.7.4 *Les états d'arrêt de la* SC*-machine sont de la forme* $(\lambda^m M[e]\ ;\ \epsilon)$ *ou* $(\underline{n}\,[\epsilon]\ ;\ s)$.

Démonstration. Pour tous les autres cas $(M[e]\ ;\ s)$, il est possible d'appliquer une des règles de la machine :

- M est une application : règle (App).
- M est une abstraction, et la pile n'est pas vide : une des règles (Split) ou (Lam) s'applique.
- M est une variable et l'environnement n'est pas vide : une des règles (Access) ou (Skip) s'applique.

□

Remarque. On travaille en général avec des termes clos. Les états d'arrêt sont dans ce cas uniquement de la forme $(\lambda^m M[e]\ ;\ \epsilon)$. □

Nous pouvons déduire du lemme 5.7.4 le corollaire suivant :

Corollaire 5.7.5 *Si* $(M[\epsilon]\ ;\ \epsilon) \twoheadrightarrow (N[e]\ ;\ s)$ *alors* $\mu(N[e]\ ;\ s)$ *est la* $\lambda\sigma_w$ *forme normale de tête de* M.

Correction de la scη-machine

Théorème 5.7.6 *Si* $(M[e]\ ;\ s) \rightarrow (M'[e']\ ;\ s')$ *alors* $\mu(M[e]\ ;\ s) \xrightarrow[\lambda\sigma_w]{}\!\!\!\!\rightarrow \mu(M'[e']\ ;\ s')$.

Démonstration. Immédiat, car les règles de la SCη-machine forment un sous-ensemble des règles de la SC-machine. □

Théorème 5.7.7 *Soit* M *un terme simplement typé.* M *et* $\mathsf{gel}(M)$ *sont* η*-equivalents.*

Démonstration. Immédiat, car gel ne fait que des η-expansions. □

La seule chose à démontrer est donc que la machine réduit les termes jusqu'à leur $\lambda\sigma_w$ forme normale de tête. Pour cela, nous avons besoin de quelques lemmes intermédiaires.

Lemme 5.7.8 *Soit M un terme en* gel.

1. *Si $M = M'\vec{N}_0^n$ alors $M, N_0, \ldots, N_n$ sont en* gel.
2. *Si $M = \lambda^n M'$ alors $\lambda^0 M', \lambda^1 M', \ldots, \lambda^{n-1} M'$ sont en* gel.

Démonstration. Par définition de gel. □

Lemme 5.7.9 *Soit M un terme en* gel *et $(M[\epsilon]\ ;\ \epsilon) \twoheadrightarrow (M'[e]\ ;\ s)$.*

1. *M' est en* gel,
2. *Soit $\vec{N}_0^n$ un vecteur de $(M'[e]\ ;\ s)$: $N_0, \ldots, N_n$ sont en* gel.

Démonstration. Par induction sur la longueur de la réduction :

- Règle (App) : immédiat.
- Règle (Lam) : par hypothèse, $\mathsf{ar}(\lambda^{n+m+1}M) = n + m + 1$, donc $\mathsf{ar}(\lambda^m M) = m$, c'est-à-dire que $\lambda^m M$ est en gel.
- Règles (Access) et (Skip) : immédiat.

□

Corollaire 5.7.10 *La machine ne peut pas s'arrêter dans un état de la forme*

$$(\lambda^{n+1}M[e]\ ;\ \langle \vec{N}_0^{n+m+1}, f\rangle \cdot s)$$

Démonstration. Puisque $\lambda^{n+1}M$ est d'arité $n + 1$, il ne peut pas y avoir plus de $n + 1$ arguments sur la pile. Par conséquent, la super-clôture qui se trouve en sommet de pile est de taille maximale $n + 1$. □

Corollaire 5.7.11 *La* SCη*-machine réduit les termes en* gel *jusqu'à leur $\lambda\sigma_w$ forme normale de tête.*

Correction de la sc-machine à fenêtres

Définition 5.7.12 *ζ est la fonction de traduction des états de la* SC*-machine à fenêtres vers les états de la* SC*-machine, définie par*

$$\zeta(M[e]\ ;\ s) = (M[\xi(e)]\ ;\ \xi(s))$$

où ξ est la fonction de traduction des environnements et des piles d'une machine à l'autre :

$$\xi(\epsilon) = \epsilon$$
$$\xi(\langle \vec{N}_0^{i+j+k}, e\rangle_i^{i+j} \cdot s) = \langle \vec{N}_i^{i+j}, e\rangle \cdot \xi(s)$$

Théorème 5.7.13 *Si $(M[e]\ ;\ s) \to (M'[e']\ ;\ s')$ alors $\zeta(M[e]\ ;\ s) \to \zeta(M'[e']\ ;\ s')$.*

Démonstration. La vérification de chaque règle de la SC-machine à fenêtres est immédiate. □

Lemme 5.7.14 *Les états d'arrêt de la* SC*-machine à fenêtre sont de la forme $(\lambda^m M[e]\ ;\ \epsilon)$ ou $(\underline{n}\,[\epsilon]\ ;\ s)$.*

Démonstration. Même argument que pour la SC-machine. □

Corollaire 5.7.15 *La* SC*-machine à fenêtres réduit les termes jusqu'à leur $\lambda\sigma_w$ forme normale de tête.*

Machine avec partage

La machine à fenêtres avec partage est une optimisation de KP. Elle suit exactement la même stratégie. Nous avons déjà démontré à la section 4 que KP est correcte et suit la stratégie CBNeedE. Nous donnons ici une preuve que toute dérivation de la machine à fenêtre avec partage se projette sur une dérivation de KP.

Notation 5.7.16 *On note $\underset{\text{SCP}}{\rightarrow}$ les transitions d'état de la machine à fenêtre avec partage et $\underset{\text{KP}}{\rightarrow}$ les transitions d'état de KP.*

Définition 5.7.17 *Si a, b sont des adresses distinctes de super-clôtures de taille respectives n et m dans la machine à fenêtres, alors $a_1, \ldots, a_n, b_1, \ldots, b_m$ représentent des adresses distinctes de KP. On définit la fonction de traduction τ des états de la machine à fenêtres avec partage vers les états de KP :*

$$\tau(M[e]\ ;\ s\ ;\ u\ ;\ h_1\ ;\ h_2) = \big(M[\tau_1(e)]\ ;\ \tau_2(s)\ ;\ \tau_3(u)\ ;\ \tau_4(h_1, h_2)\big)$$

où τ_1, τ_2, τ_3 et τ_4 sont définis par

$$\begin{aligned}
\tau_1(a_0^n \cdot e) &= a_n \cdot \ldots \cdot a_0 \cdot \tau_1(e)\\
\tau_1(\epsilon) &= \epsilon\\
\tau_2(a_0^n \cdot s) &= a_0 \cdot \ldots \cdot a_n \cdot \tau_2(s)\\
\tau_2(\epsilon) &= \epsilon\\
\tau_3\big((s, a, n) \cdot u\big) &= (\tau_2(s), a_n) \cdot \tau_3(u)\\
\tau_3(\epsilon) &= \epsilon\\
\tau_4(h_1[a \mapsto \langle \vec{l}_0^n, e\rangle], h_2) &= \tau_4(h_1, h_2)[a_0 \mapsto \tau_5(l_0, e, h_2); \ldots; a_n \mapsto \tau_5(l_n, e, h_2)]\\
\tau_4([\,], h_2) &= [\,]\\
\tau_5(M, e, h_2) &= M[\tau_1(e)]\\
\tau_5(a, e, h_2) &= M[\tau_1(e')] \quad \text{où } h_2\, a = M[e']
\end{aligned}$$

Remarque : la traduction des termes en notation vectorisée vers les termes en notation classique est implicite.

Théorème 5.7.18 *Soit S, S' des états de la machine à fenêtre. Si $S \underset{\text{SCP}}{\rightarrow} S'$ alors $\tau(S) \underset{\text{KP}}{\twoheadrightarrow} \tau(S')$.*

Démonstration. On étudie chacune des règles de la machine à fenêtre :

- Règle (App) :
  Soit $S = (M\vec{N}_0^n[e]\ ;\ s\ ;\ u\ ;\ h_1\ ;\ h_2)$ et $S' = (M[e]\ ;\ a_0^n \cdot s\ ;\ u\ ;\ h_1[a \mapsto \langle \vec{N}_0^n, e\rangle]\ ;\ h_2)$.

$$\begin{aligned}
\tau(S) &= \big(M\vec{N}_0^n[\tau_1(e)]\ ;\ \tau_2(s)\ ;\ \tau_3(u)\ ;\ \tau_4(h_1, h_2)\big)\\
&\underset{\text{KP}}{\rightarrow} \big(M\vec{N}_0^{n-1}[\tau_1(e)]\ ;\ a_n \cdot \tau_2(s)\ ;\ \tau_3(u)\ ;\ \tau_4(h_1, h_2)\big[a_n \mapsto N_n[\tau_1(e)]\big]\big)\\
&\underset{\text{KP}}{\twoheadrightarrow} \big(M[\tau_1(e)]\ ;\ a_0 \cdot \ldots \cdot a_n \cdot \tau_2(s)\ ;\ \tau_3(u)\ ;\ \tau_4(h_1, h_2)\big[a_n \mapsto N_n[\tau_1(e)]; \ldots; a_0 \mapsto N_0[\tau_1(e)]\big]\big)\\
&= \tau(S')
\end{aligned}$$

- Règle (Split) :
  Soit $S = (\lambda^{n+1}M[e]\ ;\ a_i^{i+n+m+1} \cdot s\ ;\ u\ ;\ h_1\ ;\ h_2)$ et $S' = (M[a_i^{i+n} \cdot e]\ ;\ a_{i+n+1}^{i+n+m+1} \cdot s\ ;\ u\ ;\ h_1\ ;\ h_2)$.

$$\begin{aligned}
\tau(S) &= \big(\lambda^{n+1}M[\tau_1(e)]\ ;\ a_i \cdot \ldots \cdot a_{i+n+m+1} \cdot \tau_2(s)\ ;\ \tau_3(u)\ ;\ \tau_4(h_1, h_2)\big)\\
&\underset{\text{KP}}{\rightarrow} \big(\lambda^n M[a_i \cdot \tau_1(e)]\ ;\ a_{i+1} \cdot \ldots \cdot a_{i+n+m+1} \cdot \tau_2(s)\ ;\ \tau_3(u)\ ;\ \tau_4(h_1, h_2)\big)\\
&\underset{\text{KP}}{\twoheadrightarrow} \big(M[a_{i+n} \cdot \ldots \cdot a_i \cdot \tau_1(e)]\ ;\ a_{i+n+1} \cdot \ldots \cdot a_{i+n+m+1} \cdot \tau_2(s)\ ;\ \tau_3(u)\ ;\ \tau_4(h_1, h_2)\big)\\
&= \tau(S')
\end{aligned}$$

– Règle (Lam) :
Soit $S = (\lambda^{n+m+1}M[e] \; ; \; a_i^{i+n} \cdot s \; ; \; u \; ; \; h_1 \; ; \; h_2)$ et $S' = (\lambda^m M[a_i^{i+n} \cdot e] \; ; \; s \; ; \; u \; ; \; h_1 \; ; \; h_2)$.

$$\begin{aligned}
\tau(S) &= \big(\lambda^{n+m+1}M[\tau_1(e)] \; ; \; a_i \cdot \ldots \cdot a_{i+n} \cdot \tau_2(s) \; ; \; \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big) \\
&\xrightarrow[\text{KP}]{} \big(\lambda^{n+m}M[a_i \cdot \tau_1(e)] \; ; \; a_{i+1} \cdot \ldots \cdot a_{i+n} \cdot \tau_2(s) \; ; \; \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big) \\
&\xrightarrow[\text{KP}]{} \big(\lambda^{m}M[a_{i+n} \cdot \ldots \cdot a_i \cdot \tau_1(e)] \; ; \; \tau_2(s) \; ; \; \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big) \\
&= \tau(S')
\end{aligned}$$

– Règle (Skip) :
Soit $S = (\underline{n+m+1}\,[a_i^{i+m} \cdot e] \; ; \; s \; ; \; u \; ; \; h_1 \; ; \; h_2)$ et $S' = (\underline{n}\,[e] \; ; \; s \; ; \; u \; ; \; h_1 \; ; \; h_2)$.

$$\begin{aligned}
\tau(S) &= \big(\underline{n+m+1}\,[a_{i+m} \cdot \ldots \cdot a_i \cdot \tau_1(e)] \; ; \; \tau_2(s) \; ; \; \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big) \\
&\xrightarrow[\text{KP}]{} \big(\underline{n+m}\,[a_{i+m-1} \cdot \ldots \cdot a_i \cdot \tau_1(e)] \; ; \; \tau_2(s) \; ; \; \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big) \\
&\xrightarrow[\text{KP}]{} \big(\underline{n}\,[\tau_1(e)] \; ; \; \tau_2(s) \; ; \; \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big) \\
&= \tau(S')
\end{aligned}$$

– Règle (Access1) :
Soit $S = (\underline{n}\,[a_i^{i+n+m} \cdot f] \; ; \; s \; ; \; u \; ; \; h_1 \; ; \; h_2)$ et $S' = (N[e] \; ; \; \epsilon \; ; \; (s,a,i+m) \cdot u \; ; \; h_1 \; ; \; h_2)$.

$$\begin{aligned}
\tau(S) &= \big(\underline{n}\,[a_{i+n+m} \cdot \ldots \cdot a_i \cdot \tau_1(f)] \; ; \; \tau_2(s) \; ; \; \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big) \\
&\xrightarrow[\text{KP}]{} \big(\underline{0}\,[a_{i+m} \cdot \ldots \cdot a_i \cdot \tau_1(f)] \; ; \; \tau_2(s) \; ; \; \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big) \\
&\xrightarrow[\text{KP}]{} \big(N[\tau_1(e)] \; ; \; \epsilon \; ; \; (\tau_2(s), a_{i+m}) \cdot \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big) \\
&\qquad \text{car } h_1\, a = \langle \vec{l}, e\rangle \text{ et } \vec{l}(i+m) = N \text{ implique } \tau_4(h_1,h_2)\, a_{i+m} = N[\tau_1(e)] \\
&= \tau(S')
\end{aligned}$$

– Règle (Access2) :
Soit $S = (\underline{n}\,[a_i^{i+n+m} \cdot f] \; ; \; s \; ; \; u \; ; \; h_1 \; ; \; h_2)$ et $S' = (N[e] \; ; \; s \; ; \; u \; ; \; h_1 \; ; \; h_2)$.

$$\begin{aligned}
\tau(S) &= \big(\underline{n}\,[a_{i+n+m} \cdot \ldots \cdot a_i \cdot \tau_1(f)] \; ; \; \tau_2(s) \; ; \; \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big) \\
&\xrightarrow[\text{KP}]{} \big(\underline{0}\,[a_{i+m} \cdot \ldots \cdot a_i \cdot \tau_1(f)] \; ; \; \tau_2(s) \; ; \; \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big) \\
&\xrightarrow[\text{KP}]{} \big(N[\tau_1(e)] \; ; \; \epsilon \; ; \; (\tau_2(s), a_{i+m}) \cdot \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big) \\
&\qquad \text{car } h_1\, a = \langle \vec{l}, e\rangle, \vec{l}(i+n) = a' \text{ et } h_2\, a' = N[e] \text{ implique } \tau_4(h_1,h_2)\, a_{i+n} = N[\tau_1(e)] \\
&\xrightarrow[\text{KP}]{} \big(N[\tau_1(e)] \; ; \; \tau_2(s) \; ; \; \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big[a_{i+m} \mapsto N[\tau_1(e)]\big]\big) \\
&\qquad \text{car } h_2 \text{ ne contient que des valeurs, donc } N[\tau_1(e)] \text{ est une valeur} \\
&= \tau(S') \qquad \text{car } \tau_4(h_1,h_2)\big[a_{i+m} \mapsto N[\tau_1(e)]\big] = \tau_4(h_1,h_2)
\end{aligned}$$

– Règle (Update) :
Soit $S = (\lambda^{n+1}M[e] \; ; \; \epsilon \; ; \; (s,a,i) \cdot u \; ; \; h_1 \; ; \; h_2)$ et
$S' = (\lambda^{n+1}M[e] \; ; \; s \; ; \; u \; ; \; h_1[a \mapsto \langle \vec{l'}, f\rangle] \; ; \; h_2\big[a' \mapsto \lambda^{n+1}M[e]\big])$.

$$\begin{aligned}
\tau(S) &= \big(\lambda^{n+1}M[\tau_1(e)] \; ; \; \epsilon \; ; \; (\tau_2(s), a_i) \cdot \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big) \\
&\xrightarrow[\text{KP}]{} \big(\lambda^{n+1}M[\tau_1(e)] \; ; \; \tau_2(s) \; ; \; \tau_3(u) \; ; \; \tau_4(h_1,h_2)\big[a_i \mapsto \lambda^{n+1}M[\tau_1(e)]\big]\big) \\
&= \tau(S')
\end{aligned}$$

$\square$

Théorème 5.7.19 *La machine à fenêtre réduit les termes jusqu'à leur forme normale de tête.*

Démonstration. Les états d'arrêt de la machine sont analogues aux états d'arrêt de KP. □

Corollaire 5.7.20 *La machine à fenêtre est une implantation correcte de la stratégie CBNeedE.*

5.8 Conclusion

Nous avons présenté un nouveau concept utile pour les implantations des langages fonctionnels paresseux, que nous appelons *super-clôtures*. Nous avons démontré simplement que cette idée est applicable et correcte.

Cependant, nous n'avons pas totalement résolu la question de l'efficacité. Nous avons montré que notre machine peut avoir des défauts, mais nous n'avons pas comparé les gains aux pertes en pratique. Existe-t'il des programmes réels, voire des classes de programmes, où les super-clôtures apportent une optimisation non négligeable ? ne sommes pas encore capables de répondre à cette question, dont la réponse dépend fortement de la structure des programmes. Seule l'expérimentation pourrait prouver son efficacité. Il semble évident que les super-clôtures ne peuvent être efficaces que pour des programmes dont les fonctions ont un grand nombre d'arguments. Ceci nous fait penser que l'utilisation des clôtures et des super-clôtures devraient cohabiter dans une même implantation, c'est-à-dire que certaines applications devraient entraîner la création de clôtures alors que d'autres devraient entraîner la création de super-clôture. La décision pourrait être prise à la compilation, par une méthode d'analyse statique qui resterait à définir. Nous sommes conscients qu'il existe des transformations statiques de programmes qui limitent la création des clôtures, comme l'analyse de stricticité ou l'analyse de code mort. Nos travaux futurs doivent prendre ces méthodes en considération.

De plus, nous n'avons pas soulevé le problème de l'occupation de la mémoire, notamment le *trimming* et le glânage de cellules[2], qui sont des problèmes très importants dans les implantations réelles. En particulier, nous pensons que le trimming est compliqué par l'utilisation des super-clôtures. Il a été démontré par Benaissa, Rose et Lescanne [15] que la *Spineless Tagless G-machine (STG)* de Peyton Jones [71] n'a pas de fuite de mémoire. En serait-il de même avec les super-clôtures ? Et sinon, quel serait le coût sur l'occupation de la mémoire ? En effet, les super-clôtures peuvent être très néfastes pour l'occupation de la mémoire, car les environnements sont associés à des vecteurs de code. Par conséquent, il peut être très difficile, si possible, de déterminer si une addresse est utilisée ou pas, ou même si une partie d'une super-clôture reste utile. Prenons l'exemple suivant (dans le λ-calcul avec noms pour des questions de clarté). Le terme $(\lambda f.\lambda g.f)M[x]N[y]$ dans un environnement $\rho = [x \mapsto C_1; y \mapsto C_2]$ où $M[x]$ est un terme dont seul x est une variable libre et $N[y]$ un terme dont seul y est une variable libre. Une super-clôture $\langle\{M[x]; N[y]\ , \rho\rangle$ est créée, mais seul son premier composant $M[x]$ est utilisé. De plus, ρ contient toujours x et y alors que le mapping vers la valeur associée à y est inutile et restera. Maintenant, si y est liée à une grosse structure de données, cela pourrait gravement augmenter le coût en espace en introduisant des fuites de mémoire.

Nous n'avons pas traité le problème des structures de données et de la récursion. Nous pensons que ce problème est orthogonal à la création des clôtures. En effet, la récursion peut être réalisée avec l'opérateur μ qui n'implique pas de création de clôture. L'utilisation de μ est illustrée dans le chapitre 4 dans le cadre d'une extension de KP à la récursion. De plus, les super-clôtures peuvent être très facilement étendues aux structures de données algébriques, qui peuvent être manipulées

[2] *garbage collection.*

exactement comme d'autres arguments. Par conséquent nous avons choisi de ne pas traiter ces aspects dans ce chapitre.

Nous pouvons dire que les super-clôtures ne sont probablement pas une bonne optimisation dans le cas général, car elles soulèvent des problèmes dont les solutions sont coûteuses. Cependant, nous pensons qu'il peut exister des cas spécifiques pour lesquels les super-clôtures sont une optimisation, et elles méritent donc d'être étudiées plus avant. Dans tous les cas, nous avons essayé d'exploiter cette idée simple que les clôtures pourraient être groupées, et nous avons mis en évidence les problèmes que cette idée soulève. D'un autre côté, nous avons encore une fois démontré l'utilité des λ-calculs avec substitution explicite pour la description et la preuve de propriétés de machines abstraites.

Chapitre 6

λ-réducteurs basés sur les réseaux d'interaction

Dans les chapitres précédents, nous avons traité des implantations de la β-réduction par les substitutions explicites. Il existe bien entendu des alternatives, parmi lesquelles l'utilisation des réseaux d'interaction [47]. Il s'agit de manipuler les objets du calcul (les λ-termes) comme une certaine classe de graphes non orientés, les réseaux, et non plus comme des termes ou des graphes acycliques orientés. C'est un tel réseau d'interactions que nous présentons dans ce chapitre, et qui constitue une des contributions originales de cette thèse.

L'utilisation des réseaux d'interaction pour réaliser la β-réduction a été suggérée par Lamping [48], qui a proposé un algorithme capable de réduire les λ-termes en réalisant le nombre de β-contractions optimal, au sens défini par Lévy [59, 60]. Un tel algorithme n'aurait pas pu être défini grâce à une approche de substitutions explicites, comme cela a été démontré par Field [33]. Nous présentons une version simpliste de l'algorithme de Lamping dans ce chapitre. Nous nous inspirerons ensuite d'idées proposées par Lamping pour réaliser un algorithme présenté sous forme de réseau d'interactions, comme par exemple l'utilisation des fans pour exprimer un partage très fin de la structure qui représente un λ-terme. Cependant, contrairement à l'algorithme de Lamping, notre algorithme n'est pas optimal, notre objectif étant seulement de définir un algorithme simple et efficace ; en effet, la non-optimalité de notre algorithme ne préjuge en rien sur son efficacité, l'optimalité au sens de Lévy n'étant pas à comprendre dans son sens classique. (Voir la section 1.4 pour une discussion sur ce point.) La non optimalité est liée à la sémantique des fans-out, différente de celle proposée par Lamping ou de celle induite par les réseaux de preuve de la logique linéaire [36], la géométrie de l'interaction [37, 35, 61] et les systèmes d'interaction [6, 8, 7]. En contrepartie, nous évitons les problèmes d'accumulation des nœuds de contrôle et d'explosion combinatoire que connaît l'algorithme de Lamping.

Notre algorithme ne permet pas la réduction en forme normale, mais permet toutes les stratégies de réduction de tête (faible ou forte). Ce problème peut paraître troublant : en effet, certains réducteurs peuvent être appelés récursivement, une fois la forme normale de tête calculée, sur les arguments non réduits. Ici, cependant, la représentation des termes est telle qu'il n'est en général pas possible de prolonger l'évaluation sans décompilation d'un résultat intermédiaire. Ce problème sera traité plus en détail à la section 6.5.

Parmi les autres propositions de λ-réducteurs non optimaux basés sur les réseaux d'interaction, nous pouvons comparer notre travail avec celui de Ian Mackie [61]. Il définit également un algorithme qui s'arrête avant la forme normale, et qui n'a pas de problème d'accumulation des nœuds de contrôle, raison de l'inefficacité de certains algorithmes. Cependant, son objectif est de rester le plus proche possible de la logique linéaire, rendant ainsi la preuve de son système plus simple. Nous avons choisi une approche légèrement différente en nous fiant à notre intuition, ce qui nous oblige à réaliser une preuve de correction *ad hoc* effectivement plus laborieuse. Cependant, nous ne capturons pas tout à fait le même partage.

Ce chapitre est organisé en 8 sections. La section 6.1 décrit les réseaux d'interaction de Lafont, et la section 6.2 décrit brièvement l'algorithme de Lamping. Nous proposons notre algorithme, RINO, à la section 6.3, et nous donnons à la section 6.4 des éléments déterminants dans la preuve de sa correction. Nous abordons le problème du calcul de la forme normale forte des termes à la section 6.5. Les sections suivantes traitent plus particulièrement des performances de RINO : la section 6.6 traite du partage qui est mis en œuvre, et la section 6.7 donne un banc d'essai des performances de RINO. Nous conclurons à la section 6.8

Une partie de ce travail a été réalisée en collaboration avec Zine-El-Abidine Benaissa.

6.1 Réseaux d'interaction

Dans cette section, nous présentons brièvement les réseaux d'interaction, introduits par Yves Lafont [47]. Ils sont à la base des algorithmes de réduction *optimaux* du λ-calcul [48, 38]. Les réseaux d'interaction sont un nouvel exemple de systèmes de réduction abstraits.

Les réseaux d'interaction sont des graphes qui généralisent les réseaux de preuve de la logique linéaire de Girard [36]. Ils donnent une sémantique simple de la réécriture de graphes, fortement reliée à la logique. La présentation que nous faisons ici du travail de Lafont est très succincte, et n'a pour objet que la compréhension de ce chapitre. Nous ne présentons donc que les aspects qui nous sont utiles pour la suite, et en particulier nous n'aborderons pas les problèmes de *deadlock*, *cercle vicieux*, *partitionnement*, etc. Le lecteur qui voudrait en savoir plus est invité à se référer à l'article original.

Réseaux non typés

Définition 6.1.1 (Réseau) *Un* réseau *est un graphe (cyclique) non orienté dont les nœuds sont étiquetés. Un ensemble fini et fixé de* ports *est attribué à chaque étiquette ; un de ces ports est distingué et appelé* port principal*. Les nœuds sont connectés les uns aux autres par leurs ports, et chaque port est connecté à un et un seul autre port. Lorsque deux nœuds sont connectés par leurs ports principaux, on dit qu'ils forment une* paire active*.*

Convention 6.1.2 (Numérotation des ports) *On introduit la convention de numéroter les ports en fonction de leur position par rapport au port principal dans leur représentation graphique usuelle. Le port principal prend le numéro 1, et ainsi de suite lorsqu'on parcourt les ports dans le sens trigonométrique.*

Exemple 6.1.3 *La figure 6.1 donne des exemples de nœuds d'interaction qui permettent de représenter les entiers et l'addition. Leur port principal est identifié par une flêche. Le nœud étiqueté par* `Plus` *(on dira le nœud* `Plus`*) a trois ports. Les nœuds* `Succ` *et* `Zero` *ont respectivement 2 et 1 port.*

Définition 6.1.4 (Système de règles d'interaction) $a, b, c, \ldots$ *sont des* variables de ports*. Un* système de règles d'interaction *est un système de règles de réécriture (de réseaux) dont les membres gauches sont des paires actives, et qui satisfait les propriétés 6.1.6, 6.1.7 et 6.1.8.*

Exemple 6.1.5 *Le système de règles d'interactions de la figure 6.2 définit l'addition sur les entiers.*

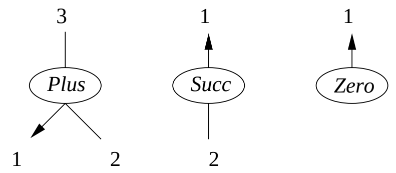


FIG. 6.1 – Nœuds d'interaction pour les entiers munis de l'addition

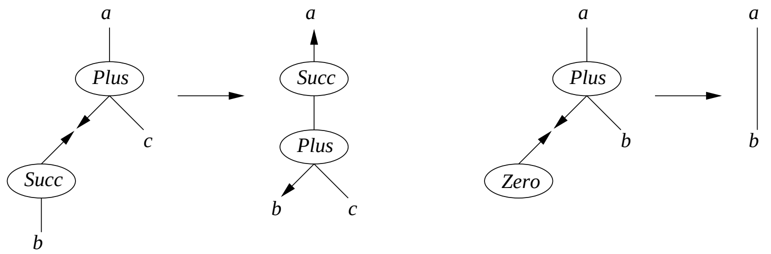


FIG. 6.2 – Règles d'interaction définissant l'addition sur les entiers

Propriété 6.1.6 (Linéarité) *Dans une règle, chaque variable a exactement deux occurrences, une dans le membre gauche et l'autre dans le membre droit.*

En conséquence de la contrainte introduite par la propriété 6.1.6, la duplication et l'effacement de nœuds doivent être explicites.

Propriété 6.1.7 (Non ambiguïté) *Il y a au plus une règle pour chaque paire active.*

Propriété 6.1.8 (Optimisation) *Les membres droits des règles ne contiennent pas de paires actives.*

Lorsqu'une paire active est réduite par une règle d'interaction, on dit qu'il y a *interaction* entre les deux éléments de la paire. La propriété principale des systèmes de règles d'interaction est la confluence forte, qui découle naturellement des propriétés.

Théorème 6.1.9 *Tout système de règles d'interaction est fortement confluent.*

Réseaux typés

D'après la définition 6.1.4, certains réseaux, comme celui de la figure 6.3 peuvent être considérés comme valides, alors qu'ils n'ont manifestement aucun sens. Afin de restreindre la validité des réseaux, il est possible d'introduire des règles de typage sur les ports.

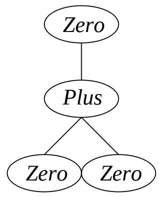


FIG. 6.3 – Un réseau qui n'a pas de sens

Définition 6.1.10 (Types) *Tout port possède un type constitué d'une* constante de type *et d'une* polarité $+$ *ou* $-$. *Un port ayant une polarité négative est une* entrée *et un port ayant une polarité positive est une* sortie. *On dit que deux ports sont* opposés *lorsqu'ils ont la même constante de type et des polarités opposées.*

Définition 6.1.11 (Réseaux et règles bien typés)

- *Un* réseau *est* bien typé *si toutes ses connexions lient des sorties à des entrées du même type.*
- *Une* règle *est* bien typée *si les ports principaux des symboles de son membre gauche sont opposés, et si le membre droit est bien typé – le type des variables étant donné par le membre gauche.*

Exemple 6.1.12 *Si on définit une constante de type* `nat`*, on peut donner les types suivants aux ports de la figure 6.1 :* `nat+` *aux ports principaux des nœuds* `Zero` *et* `Succ`*, ainsi qu'au port numéro 3 du nœud* `Plus` *;* `nat-` *à tous les autres ports. Ainsi il est facile de constater que le réseau de la figure 6.3 est mal typé.*

Définition 6.1.13 *Les nœuds dont le port principal est une sortie sont appelés* constructeurs. *Les autres sont appelés* destructeurs.

Remarque. Comme on pouvait s'y attendre, `Zero` et `Succ` sont des constructeurs, `Plus` est un destructeur. Le système de règles d'interaction proposé pour définir l'addition semble être une implantation triviale du système de réécriture de termes $S(x) + y \to S(x+y)$, $0+x \to x$ sous la forme d'un système de règles d'interaction. Cependant, dans le cas général, la traduction du système de réécriture de termes vers un système de règles d'interaction équivalent n'est pas immédiate. Par exemple, le système de réécriture de termes

$$\begin{aligned} \max(0, x) &\to x \\ \max(x, 0) &\to x \\ \max(S(x), S(y)) &\to S(\max(x, y)) \end{aligned}$$

ne peut pas être implanté sous la forme d'un système de règles d'interaction sans par exemple l'introduction d'un destructeur auxilliaire $\max'$ défini comme suit :

$$\begin{aligned} \max(0, x) &\to x \\ \max(S(x), y) &\to \max'(x, y) \\ \max'(x, 0) &\to S(x) \\ \max'(x, S(y)) &\to S(\max(x, y)) \end{aligned}$$

□

6.2 L'algorithme optimal de Lamping

Dans [48], John Lamping a présenté un des premiers λ-réducteurs optimaux[1]. Cet algorithme, malgré la vision très pragmatique de son auteur est, hormis certains détails, très proche d'un système de règles d'interaction : les objets manipulés sont des réseaux tels que définis pas Lafont.

[1] À la même époque, Vinod Kathail a proposé lui aussi un algorithme pour la β-réduction optimale des λ-termes [42].

Gonthier, Abadi et Lévy ont modifié cet algorithme [38] afin d'établir un lien plus fort entre le pragmatisme ingénieux de Lamping et la logique linéaire. Nous nous contenterons ici de décrire une version naïve (et par conséquent incorrecte) de l'algorithme de Lamping, sous la forme d'un système de règles d'interaction. Nous discuterons ensuite pourquoi cette description est erronée, et décrirons de manière moins formelle la solution – complexe – mise en œuvre par Lamping.

L'algorithme de Lamping repose sur une vision linéaire du nœud d'abstraction[2] : un nœud d'abstraction possède 3 ports : de manière informelle, il s'agit des ports liés respectivement à sa *racine*, à son *corps*, et à l'emplacement de sa *variable*. L'élimination d'un β-redex s'exprime alors tout simplement par l'interaction entre une application et une abstraction, dont le résultat est le remplacement de la variable de l'abstraction par l'argument de l'application, qui se fait simplement par une connection des ports. Cette règle est représentée sur la figure 6.5. L'implantation de la β-réduction ne serait pas plus compliquée si on ne considérait qu'un λ-calcul linéaire. Cependant, certaines variables ont plusieurs occurrences dans un terme. Il faut donc introduire un opérateur de partage explicite, nommé *fan-in*[3]. De plus, certaines variables peuvent n'avoir aucune occurrence, ce qui justifie l'introduction d'un nœud supplémentaire que nous nommerons le *nœud vide*, pour lequel il n'existe aucune règle associée, si ce n'est des règles de garbage collection (ou glanage de cellules) que nous ne donnerons pas ici. Il est ainsi possible de représenter n'importe quel λ-terme à l'aide de quatre nœuds d'interaction : application, abstraction, nœud vide et fan-in, représentés sur la figure 6.4. Tout fan est identifié par une étiquette (représentée par les variables $s, t, \ldots$), unique avant la réduction. Le nœud d'abstraction lie une variable de manière explicite, ce lien étant représenté par un arc dans le graphe. Par conséquent, il n'est pas utile de donner des noms aux variables. Cependant, nous prendrons l'habitude de leur donner des noms, qui apparaîtront dans les figures comme des anotations des nœuds d'abstraction (λx au lieu de λ). Ces anotations sont utiles à la compréhension des réseaux et peuvent être utilisées au moment de la décompilation des réseaux pour l'attribution des noms. Nous supposons qu'aucun conflit n'en résulte.

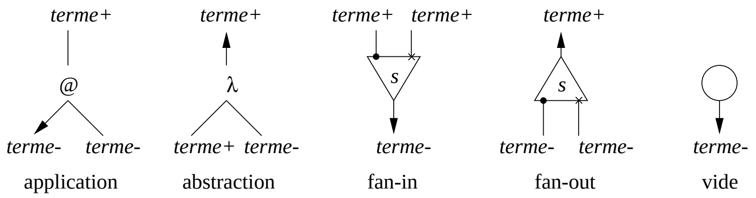


FIG. 6.4 – Nœuds d'interaction de l'algorithme simpliste de Lamping

Exemple 6.2.1 *Le terme $\lambda z.(\lambda x.x\,x)(\lambda y.z)$ peut être représenté par le réseau de la figure 6.6.*

Outre l'élimination de β-redex, d'autres règles sont nécessaires et sont présentées sur la figure 6.7. En effet, après quelques éliminations de β-redexes, des rédexes sont cachés par des fans-in, comme c'est le cas après la réduction illustrée sur la figure 6.8. Une abstraction est partagée, et fait partie de deux sous-termes distincts, dont l'un est un redex. Par conséquent, il faut créer un nouvel emplacement pour sa variable. En effet, cette variable peut prendre des valeurs différentes selon le redex auquel elle appartient. Le principe de l'algorithme repose sur l'idée de la duplication simultanée de l'abstraction et de sa variable, sans duplication du corps de l'abstraction, qui peut contenir des redexes, effectifs ou potentiels. Pour cela, un nœud dual du fan-in, le *fan-out* est introduit. Ces nœuds, fan-in et fan-out associés, délimitent la partie de la structure (du réseau) qui

[2] D'où les nombreux liens entre la logique linéaire et l'optimalité ; voir par exemple sur ce sujet les nombreux travaux de Asperti et Laneve [6, 8, 5, 7, 50].

[3] *fan* : en français, *éventail*.

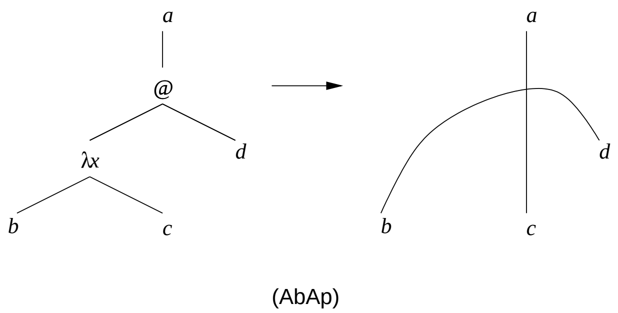


FIG. 6.5 – L'élimination de β-redex par interaction

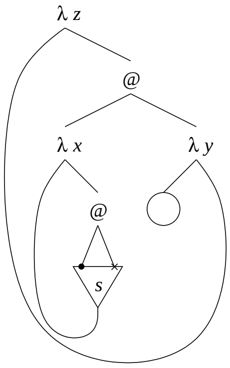


FIG. 6.6 – Représentation du terme $\lambda z.(\lambda x.x\,x)(\lambda y.z)$ par un réseau

doit rester partagée ; à ce moment de la réduction il s'agit de la partie du contexte de la variable en question qui est interne au corps de son lieur.

Nous avons appelé la règle (AbAp) règle d'*élimination de β-redex*. Elle s'apparente en ce sens à la règle (b) d'un λ-calcul avec substitutions explicites. Pourtant, contrairement à (b), (AbAp) effectue le remplacement de la variable par l'argument du redex. En ce sens, (AbAp) est similaire à la règle de β-contraction (β) du λ-calcul. Cependant, comme nous l'avons dit, cette règle induit un certain nombre d'actions décrites par les autres règles, nécessaires pour faire apparaître le prochain β-redex. La β-réduction n'est donc pas une suite d'applications de (AbAp), quand il s'agit d'une suite de β-contractions. Nous préférons donc qualifier (AbAp) de *règle d'élimination de β-redex* et ne pas faire la confusion, même s'il existe de nombreux cas où cette règle suffit à réaliser une β-contraction.

Comme le fan-in, le fan-out a lui aussi son rôle de duplication : lorsqu'une application fait partie de deux redexes distincts, il est nécessaire de la dupliquer, sans toutefois dupliquer son argument. C'est ce qui est fait dans l'interaction entre le fan-out et l'application (FoAp). La figure 6.9 donne l'exemple de la réduction en forme normale du réseau présenté à la figure 6.8.

Il existe une relation entre fans-in et fans-out : un fan-out a toujours été créé par l'interaction entre un fan-in et une abstraction, ou par duplication d'un autre fan-out. Dans notre description naïve de l'algorithme, il est sous-entendu que lorsqu'un fan-in et un fan-out interagissent, la procédure de décicision pour savoir s'ils délimitent un même morceau de contexte est simple, et qu'il suffit de donner des étiquettes aux fans. Ce n'est en réalité pas le cas, et afin d'assurer la correction de l'algorithme, il est nécessaire d'introduire d'autres nœuds et d'autres règles. Ce sont ces nœuds (*croissants, crochets*) et ces règles que Lamping lui même appelle la *bureaucratie*. Ils

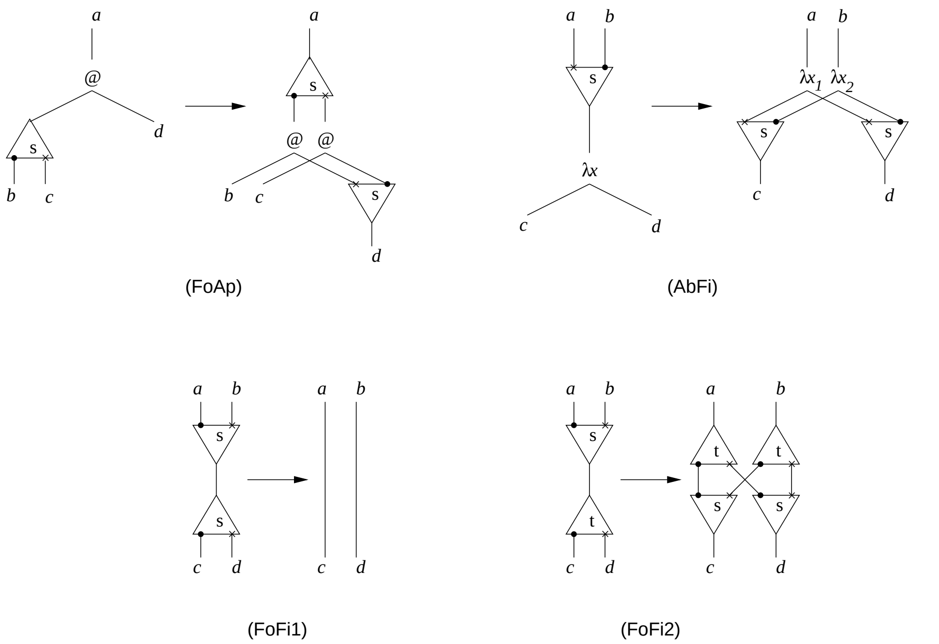


FIG. 6.7 – Autres règles d'interaction de l'algorithme simpliste de Lamping

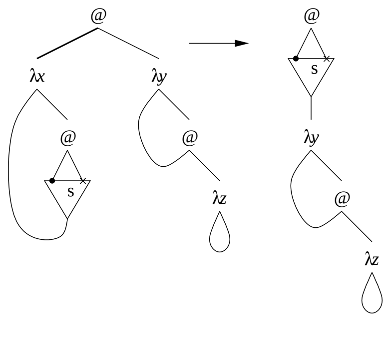


FIG. 6.8 – Exemple de redex caché après une réduction

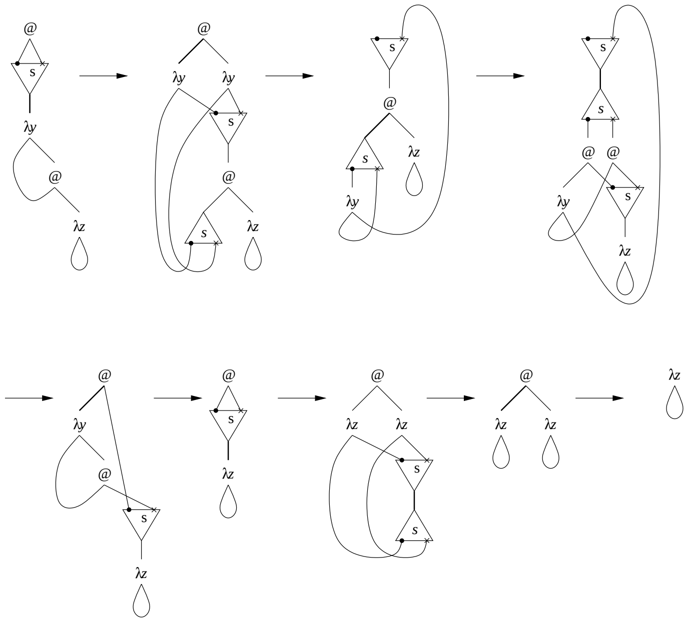


FIG. 6.9 – Prolongement de l'évaluation

induisent une complexité de réduction pouvant être très élevée, confirmant ainsi que la complexité de la réduction d'un terme ne dépend pas seulement du nombre de β-contractions.

6.3 Une autre utilisation des réseaux d'interaction : **RINO**

Nous venons de voir que l'algorithme de Lamping présente deux types de règles : la règle de β-élimination d'une part, et des règles annexes nécessaires pour faire aparaître de nouveaux redex d'autre part (il s'agit de la *bureaucratie*, selon les termes de Lamping). Il est donc important de garder à l'esprit la signification de la notion d'optimalité à la Lévy, qui n'a rien à voir avec une optimalité vue dans l'optique d'un implanteur : nous sommes dans un cadre où seules les β-contractions ont un coût, ce qui n'est bien entendu pas réaliste. D'autant plus qu'il n'existe aucune borne au nombre de règles de la bureaucratie que l'on peut appliquer à un terme suite à une β-contraction. Nous ne nous placerons donc pas ici dans une optique de définition d'un algorithme optimal au sens de Lévy.

Par contre, nous avons été particulièrement séduits par le formalisme des réseaux d'interaction qui permet de mettre en œuvre un partage qui ne peut pas être atteint avec un calcul de substitution explicite. Il nous a donc semblé attractif de proposer un algorithme original basé sur les réseaux d'interaction et qui permette de réaliser la β-réduction, tout en offrant une solution beaucoup plus simple que l'algorithme de Lamping[4] – mais encore une fois, nous ne sommes pas dans la même optique.

Notre algorithme, nommé RINO pour *Réseau d'Interaction Non Optimal*, repose sur l'idée que la complexité bureaucratique de l'algorithme de Lamping provient essentiellement du comportement de la règle d'interaction entre une application et un fan-out (règle (FoAp) de la figure 6.7) : c'est en particulier cette règle qui impose l'introduction des croissants et des crochets. Nous pensons

[4]Nous parlons bien sûr de l'algorithme avec crochets et croissants et non de l'algorithme simpliste présenté à la section 6.2 qui est très simple mais incorrect.

que nous pouvons nous épargner l'introduction de ces opérateurs en modifiant cette règle. Bien entendu, cela se paiera par la perte de l'optimalité. Nous ne permettrons donc pas la duplication d'un nœud d'application par un fan-out, et par conséquent, ce sont les fans-in qui devront avoir pour tâche de dupliquer les applications. Cependant, il n'est pas question de dupliquer les applications aveuglément. Il existe en effet un certain nombre de redex qu'il est inutile de dupliquer, notamment les redex qui ont une existence effective dans le graphe. D'autre part, nous souhaitons respecter la théorie des réseaux d'interaction afin de profiter *gratuitement* de la propriété de confluence.

Considérons un nœud d'application et un fan-in de l'algorithme de Lamping. A priori, ces nœuds ne peuvent pas interagir, car ils sont tous deux destructeurs, leur rôle principal étant l'interaction avec l'abstraction : l'un dans le cadre de la contraction d'un β-redex ; l'autre dans le cadre de la duplication nécessaire d'un lieur (*i.e.* la création d'un nouvel emplacement de variable avant une éventuelle β-contraction). Nous avons donc besoin de définir un nouveau type d'application, qui soit un constructeur : l'application duplicable (par un fan-in). Ainsi, lorsqu'une application entre en interaction avec un fan-out, elle change de statut pour devenir application duplicable. (Voir la règle (FoAp) de la figure 6.11.) Cette application peut alors être dupliquée par un fan-in (AdFi) ou provoquer la mutation d'autres nœuds d'application en applications duplicables (AdAp). Nous décrivons à présent RINO de manière plus formelle.

Définition 6.3.1 (Séquences)

1. *Soit $\mathcal{S}$ un ensemble. Une séquence sur $\mathcal{S}$ est définie inductivement par*
   - *ϵ est une séquence sur $\mathcal{S}$ (on la nomme* séquence vide*) ;*
   - *si l est une séquence sur $\mathcal{S}$ et $x \in \mathcal{S}$ alors $x \cdot l$ est une séquence sur $\mathcal{S}$ (on dit que x est l'élément de tête de $x \cdot l$).*
2. *La* longueur *d'une séquence $\mathsf{lg}(l)$ est définie par :*

$$\mathsf{lg}(\epsilon) = 0$$
$$\mathsf{lg}(x \cdot l) = 1 + \mathsf{lg}(l)$$

3. *Soit l_1 et l_2 deux séquences sur $\mathcal{S}$. $l_1 + l_2$ est la séquence sur $\mathcal{S}$ définie par :*

$$\epsilon + l = l$$
$$(x \cdot l_1) + l_2 = x \cdot (l_1 + l_2)$$

4. *On définit sur les séquences l'ordre $\leq$ par $\forall l_1, l_2 : l_1 \leq l_2 \Leftrightarrow \exists l_3 : l_2 = l_1 + l_3$. Si $l_1 \leq l_2$ on dit que l_1 est un préfixe de l_2.*

Définition 6.3.2 (Variables, étiquettes, nœuds)

- $\mathbf{V}$ *est un ensemble infini d'identificateurs notés $x, y, z, x_1, x_2, \ldots$ et appelés* variables.
- $\mathbf{L}$ *est l'ensemble des séquences sur* $\mathbf{V}$*, nommées* étiquettes*, et notées $s, t, u, \ldots$.*
- *Les différents nœuds mis en œuvre dans l'algorithme sont présentés sur la figure 6.10. On les nomme respectivement (de gauche à droite)* application*,* application duplicable*,* abstraction*,* fan-out*,* fan-in *et* nœud vide*. Toute abstraction est identifiée par une variable $x \in \mathbf{V}$, et tout fan est étiqueté par une étiquette $s \in \mathbf{L}$.*
- *Le port numéro 2 d'une abstraction est appelé* port variable*, et le réseau connecté au port numéro 3 d'une abstraction est appelé le* corps *de l'abstraction.*

Remarque. Notre but est la définition d'un système de règles d'interaction. Nous avons cependant ici fait une légère généralisation par rapport à la définition de Lafont en permettant à certains nœuds d'avoir des étiquettes appartenant à un ensemble infini, et nous le verrons plus loin, modifiables dynamiquement. Cependant, les propriétés des réseaux d'interaction sont assez solides pour résister à cette généralisation, en particulier la confluence forte. □

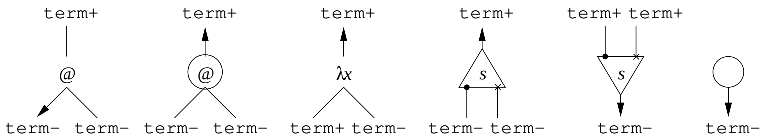


FIG. 6.10 – Nœuds d'interaction de RINO

La figure 6.11 présente RINO. Ce système n'est correct qu'avec la convention suivante :

Convention 6.3.3 *Les variables x_1 et x_2 qui apparaissent dans le membre droit de l'interaction entre une abstraction et un fan-in sont des variables* fraîches, *i.e. qui n'ont jamais eu d'occurrence dans les réseaux dont est issu (par interaction) le réseau courant.*

Remarque. Il existe un mécanisme simple et local pour la définition des variables fraîches : considérons les identificateurs de variables construits par l'expression régulière $I\{0,1\}^*$ où I est un ensemble infini d'identificateurs ne contenant pas les symboles 0 et 1, et tel qu'initialement tout identificateur du réseau appartient à I. Si x est l'identificateur d'une variable liée dans le réseau, alors $x0$ et $x1$ sont des variables *fraîches.* Ceci est dû au fait que toute variable a une occurrence unique dans un réseau. □

6.4 Compilation et décompilation

Soit un λ-terme M, il existe un réseau d'interaction qui représente M. Nous ne définirons pas formellement la transformation qui permet de passer des termes aux réseaux qui est, somme toute, assez classique et d'ailleurs quasiment identique à la compilation des termes dans les réseaux de l'algorithme de Lamping ; nous définirons seulement la transformation inverse (*read-back*). Cependant nous nous concentrerons sur quelques règles à respecter pour que toute transformation soit correcte.

Convention 6.4.1
- *Tout réseau d'interaction issu de la traduction d'un terme (on dira* initial*) est sans fan-out. Les fans-out ne peuvent donc apparaître dans le réseau que par réduction.*
- *Toutes les variables d'un réseau initial sont distinctes.*
- *Tous les fans-in d'un réseau initial sont étiquetés par ϵ.*

Remarque. La dernière contrainte est en réalité un peu forte : il n'est pas indispensable d'étiqueter initialement les fans-in particulièrement par ϵ, d'autres étiquettes pouvant faire l'affaire. Cependant, ce choix semble le plus naturel, et il nous évite d'avoir à définir précisément quelles sont les étiquettes initiales valides d'un réseau. □

Corollaire 6.4.2 *Toutes les variables d'un réseau obtenu par interactions à partir d'un réseau initial sont distinctes.*

Démonstration. Immédiat par les conventions 6.3.3 et 6.4.1. □

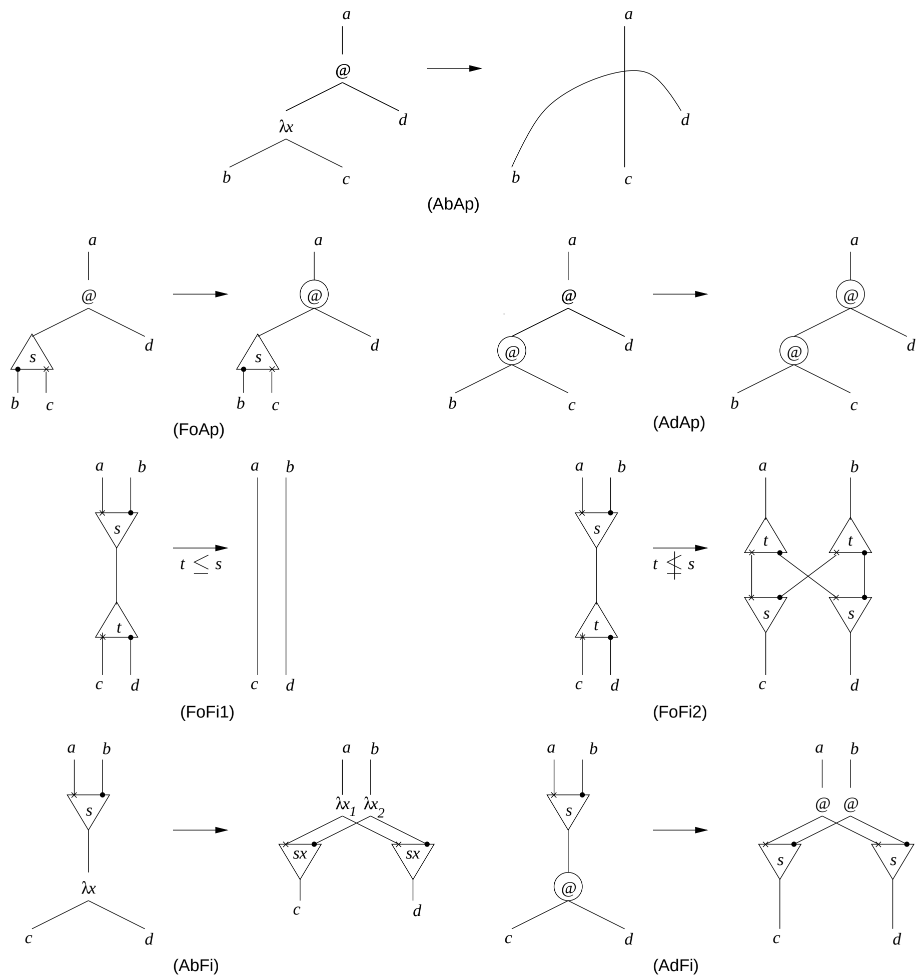
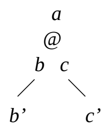
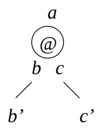
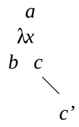
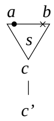


FIG. 6.11 – Règles d'interaction de RINO (β-réduction de tête)

Dans ce schéma de décompilation, a, b, c désignent les ports du nœud courant et a', b', c' les ports qui y sont respectivement connectés.

$$\mathcal{R}_{(R)}(a,p) = \big(\mathcal{R}_{(R)}(b',p)\,\mathcal{R}_{(R)}(c',p)\big)$$ avec  ou  dans R

$$\left.\begin{array}{l}\mathcal{R}_{(R)}(a,p) = \lambda x.\mathcal{R}_{(R)}(c',p)\\ \mathcal{R}_{(R)}(b,p) = x\end{array}\right\}$$ avec  dans R

$$\left.\begin{array}{l}\mathcal{R}_{(R)}(a,p) = \mathcal{R}_{(R)}\big(c',(s,\bullet)\cdot p\big)\\ \mathcal{R}_{(R)}(b,p) = \mathcal{R}_{(R)}\big(c',(s,\times)\cdot p\big)\end{array}\right\}$$ avec  dans R

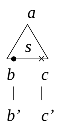
$$\left.\begin{array}{l}\mathcal{R}_{(R)}\big(a,(t,\_)\cdot p\big) = \mathcal{R}_{(R)}(a,p)\\ \mathcal{R}_{(R)}\big(a,(u,\bullet)\cdot p\big) = \mathcal{R}_{(R)}(b',p)\\ \mathcal{R}_{(R)}\big(a,(u,\times)\cdot p\big) = \mathcal{R}_{(R)}(c',p)\end{array}\right\}$$ avec  dans R, et $s \not\leq t, s \leq u$

FIG. 6.12 – Read back des réseaux de RINO

Définition 6.4.3 (Read-back) *Soit $\mathcal{M}$ l'ensemble $\{\bullet, \times\}$. Un réseau d'interaction R existant en arrière plan, on définit la procédure de* read-back[5] *qui lui est associée : $\mathcal{R}_{(R)}$ prend en argument un port positif et une pile (une séquence sur l'ensemble des couples $\{(s,m) \mid s \in \mathbf{L}, m \in \mathcal{M}\}$) et retourne un terme. Cette procédure est appelée initialement sur le port qui constitue le point d'entrée du réseau (sa racine), avec une pile vide. Elle est définie sur la figure 6.12.*

Il nous faudrait démontrer la correction de RINO, ainsi que la correction de $\mathcal{R}$. Nous nous contenterons d'en donner seulement quelques éléments convaincants : en particulier que notre système d'étiquetage convient pour identifier les fans, et que l'utilisation d'une pile pour la procédure $\mathcal{R}$ est suffisante pour que $\mathcal{R}$ soit totalement définie.

Nous commençons par prouver la correction de notre système d'étiquetage des fans. Pour cela nous définissons les notions de réplique, original, clône, géniteur et parent. Nous démontrons ensuite que la condition sur les étiquettes des fans lors de leur interaction exprime bien ce que nous attendons.

Définition 6.4.4 (Répliques d'un fan) *Soit f un fan, les* répliques immédiates *de f sont les fans de même type que f issus de son interaction avec un autre nœud. On étend la relation de réplique immédiate à celle de* réplique *par fermeture reflexive et transitive.*

Exemple 6.4.5 *Un fan-in f qui interagit avec un fan-out d'étiquette non préfixe* (FoFi2) *a trois répliques : f lui-même (qui n'a cependant plus d'existence propre dans le réseau), ainsi que ses deux copies par cette règle qui sont ses répliques immédiates.*

Définition 6.4.6 (Original, Clône) *Soit f un fan-out, on appelle l'*original *de f le fan-out dont f est une réplique, et qui n'est la réplique que de lui-même. Soit f_0 l'original de f, on appelle un* clône *de f tout fan-out d'original f_0.*

[5] ou *décompilation.*

Remarque. L'existence d'un original pour tout fan-out est due à l'absence de fan-out dans le réseau initial, et au fait qu'un fan-out ne peut être créé que par l'interaction d'un fan-in et d'une abstraction (AbFi). C'est cette même remarque qui donne lieu aux définitions suivantes. □

Définition 6.4.7 (Géniteur, Parent) *Soit un fan-out f dont l'original f_0 a été produit par interaction* (AbFi) *impliquant un fan in f'. On appelle le* géniteur *de f la réplique immédiate de f' par* (AbFi). *On appelle un* parent *de f tout fan-in réplique du géniteur de f.*

La suite du développement consiste à démontrer que les règles d'interaction entre fans de RINO sont correctes. Plus précisément, nous montrons que deux fans ne s'éliminent que s'ils sont parents. Ceci correspond bien à l'idée qu'un fan-in f en dupliquant une abstraction démarre un processus de duplication du corps entier de l'abstraction avec le fan-out créé f'. Ce processus ne se termine localement (pour un chemin donné) que lorsque des répliques de f et f' interagissent. Nous démontrons que dans ce cas, l'étiquetage que nous avons choisi et la condition de préfixe sont corrects.

Lemme 6.4.8 *Soit f et f' des fans-out d'étiquettes respectives s et s'. f et f' sont clônes l'un de l'autre si et seulement si $s = s'$.*

Démonstration.
- Si f et f' sont clônes, alors ils sont des répliques d'un même original f_0 d'étiquette s_0. Il est facile de vérifier que les répliques d'un fan-out d'étiquette t ont la même étiquette t (FoAp, FoFi2).
- Si f et f' ont la même étiquette, alors leurs originaux respectifs f_0 et f'_0 ont la même étiquette (car f et f_0 sont clônes l'un de l'autre, idem pour f' et f'_0, et donc d'après le point précédent ils ont la même étiquette), notée $s_0.x$. En effet, l'étiquette d'un fan-out n'est jamais ϵ, comme le prouve la seule règle qui génère de nouveaux fan-outs (AbFi). Leurs géniteurs respectifs sont donc des fans-in d'étiquettes s_0 qui ont interagi avec une abstraction λx. Or d'après le lemme 6.4.2, il n'existe qu'une seule abstraction de variable x et après (AbFi) cette variable disparaît. Donc, f_0 et f'_0 ont le même géniteur, *i.e.* ils sont clônes.

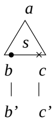
□

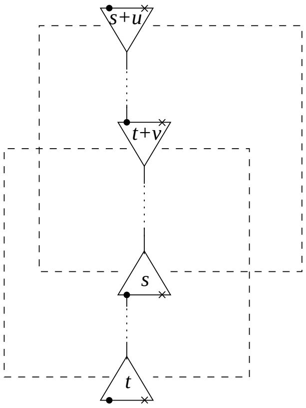
Lemme 6.4.9 *Soit f un fan-out d'étiquette s et f' un fan-in d'étiquette s'. f' est parent de f si et seulement si $s \leq s'$.*

Démonstration.
- Si f' est parent de f, alors d'après la définition 6.4.7, f' est une réplique du géniteur de f, f_0. D'après le lemme 6.4.8, f_0 a la même étiquette s que f, et il est facile de vérifier que toutes les répliques d'un fan-in d'étiquette t ont une étiquette dont t est un préfixe. Donc $s \leq s'$.
- Si $s \leq s'$, alors f' est une réplique d'un fan-in d'étiquette s. $s = s_0.x$ (car c'est l'étiquette d'un fan-out), *i.e.* il y a eu une interaction entre un fan-in dont f' est une réplique et une abstraction de la variable x (AbFi). Cette interaction n'a pu avoir lieu qu'une fois, et c'est celle qui a donné naissance à f. Donc f' est une réplique du géniteur de f, *i.e.* f' est parent de f.

□

Corollaire 6.4.10 *Un fan-in et un fan-out ne s'éliminent* (FoFi1) *que lorsqu'ils sont parents. L'étiquetage des fans et les conditions d'étiquettes sont donc corrects.*

Nous donnons à présent une idée de la correction de la procédure de read-back $\mathcal{R}$. Nous montrons que si un réseau peut-être décompilé par $\mathcal{R}$, alors tous les réduits de ce réseau peuvent également être décompilés par $\mathcal{R}$. Autrement dit, l'utilisation d'une pile est correcte, et $\mathcal{R}$ ne bloque jamais sur un cas $\mathcal{R}(a, \epsilon)$ avec  dans R (c'est le seul cas de blocage possible). Autrement dit, on ne peut pas avoir de sous réseau de la forme



Associée à la preuve de correction sur l'utilisation des étiquettes qui précède, cette partie de la preuve de correction donne des éléments qui permettent de se persuader de la correction de RINO. En effet, les règles d'interaction que nous proposons sont très intuitives et, somme toute, très proches des règles proposées par Lamping.

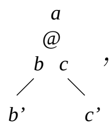
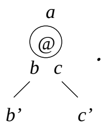
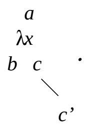
Le premier lemme établit que $\mathcal{R}$ est complètement définie sur les réseaux initiaux.

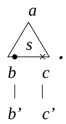
Lemme 6.4.11 *Tout réseau initial est décompilable par la procédure $\mathcal{R}$.*

Démonstration. Un tel réseau ne contient pas de fan-out, et la procédure $\mathcal{R}$ est complêtement définie, sans condition, sur le domaine des ports positifs. Par conséquent $\mathcal{R}$ termine et retourne toujours un résultat. □

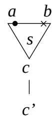
Nous définissons ensuite une notion de trace de chemins, et de chemin valide comme suit.

Définition 6.4.12 (Trace, Chemin)

1. *Une* trace *est une séquence sur* $\{\Delta_s^m \mid s \in \mathbf{L}, m \in \mathcal{M}\} \cup \{\nabla_s^m \mid s \in \mathbf{L}, m \in \mathcal{M}\}$.
2. *Un* chemin *(noté* $\mathbf{c}, \mathbf{c}', \ldots$*) d'un port positif a à un port positif b est une séquence sur les ports positifs, auquel on associe une trace :*
   – *Soit a un port positif quelconque,* $a \cdot \epsilon$ *est un chemin de trace* ϵ.
   – $a \cdot b' \cdot \epsilon$ *est un chemin de trace* ϵ *et* $a \cdot c' \cdot \epsilon$ *un chemin de trace* ϵ *dans*  *, et dans*  .
   – $a \cdot c' \cdot \epsilon$ *est un chemin de trace* ϵ *dans*  .

– *$a \cdot b' \cdot \epsilon$ est un chemin de trace $\Delta_s^\bullet \cdot \epsilon$ et $a \cdot c' \cdot \epsilon$ un chemin de trace $\Delta_s^\times \cdot \epsilon$ dans* 

.

– *$a \cdot c' \cdot \epsilon$ est un chemin de trace $\nabla_s^\bullet \cdot \epsilon$ et $b \cdot c' \cdot \epsilon$ est un chemin de trace $\nabla_s^\times \cdot \epsilon$ dans* 

.

– *Si $(a \cdot \mathbf{c}) + (b \cdot \epsilon)$ est un chemin de trace τ et $b \cdot \mathbf{c}'$ un chemin de trace τ' alors $(a \cdot \mathbf{c}) + (b \cdot \mathbf{c}')$ est un chemin de trace $\tau + \tau'$.*

3. *Un chemin de la forme $(a \cdot \mathbf{c}) + (b \cdot \epsilon)$ est dit chemin reliant a à b ou chemin de a à b. On dit également que le chemin commence en a et finit en b.*

Définition 6.4.13 (Trace valide, Chemin valide)

1. *Une trace τ est* valide *si*
   – $\tau = \epsilon$.
   – *ou $\tau = \nabla_s^m \cdot \sigma$, avec σ une trace valide.*
   – *ou $\tau = (\nabla_{s+u}^m \cdot \sigma_1) + (\Delta_s^m \cdot \sigma_2)$, avec σ_1 et σ_2 des traces valides.*
2. *Un chemin est* valide *si et seulement si sa trace est valide.*

On voit immédiatement que $\mathcal{R}$ décompile les termes en suivant les chemins valides. En effet, tout appel récursif de $\mathcal{R}$ est effectué sur un nœud qui suit le nœud de l'appel courant dans un chemin valide. Par exemple, dans la première ligne de la définition 6.4.3, b' et c' suivent a dans des chemins valides. Le fait que les chemins suivis sont valides est assuré par la pile p qui maintient l'information nécessaire sur les traces.

Définition 6.4.14 (Chemin Initial, Clos, Final, Cyclique)

1. *Un chemin est* initial *s'il relie la racine du réseau à un port quelconque.*
2. *Un chemin est* clos *s'il relie la racine du corps d'une abstraction au port de variable de cette même abstraction.*
3. *Un chemin est* final *s'il relie la racine du réseau à un port de variable.*
4. *Un chemin est* cyclique *s'il relie un port à lui même et est de longueur strictement supérieure à* 1.

Nous voulons à présent donner au lecteur une bonne intuition de la correction de RINO et de la procédure $\mathcal{R}$. Nous voulons démontrer que tout chemin initial peut aboutir à une variable, autrement dit que tout chemin initial est le préfixe d'un chemin final valide. Pour cela nous montrerons que si c'est vrai pour un réseau, alors après une étape de réduction, la propriété est toujours vraie. Clairement, si cette propriété est satisfaite, alors aucun chemin de réduction par $\mathcal{R}$ ne mène à un blocage. Il est à noter cependant que nous n'aurons pas démontré que $\mathcal{R}$ termine toujours. Il faudrait pour cela démontrer qu'il n'existe pas de chemin valide infini, c'est à dire pas de chemin valide cyclique.

Nous donnons tout d'abord quelques propriétés bien utiles sur les traces valides.

Lemme 6.4.15 *Si τ_1 et τ_2 sont des traces valides alors $\tau_1 + \tau_2$ est une trace valide.*

Démonstration. Par induction sur la structure de τ_1 : si $\tau_1 = \epsilon$, c'est trivial. Sinon τ_1 est de la forme $\sigma + \sigma'$ avec σ' valide (soit $\sigma = \nabla_s \cdot \epsilon$, soit $\sigma = (\nabla_{s+t} \cdot \sigma_1) + (\Delta_s \cdot \epsilon)$). Par hypothèse d'induction, $\sigma' + \tau_2$ est valide, et par conséquent, par application de la définition 6.4.13, $\tau_1 + \tau_2$ est valide. $\square$

Lemme 6.4.16 *Si $\tau_1 + (\nabla^m_{s+u} \cdot \tau_2) + (\Delta^m_s \cdot \tau_3)$ est une trace valide avec τ_2 valide, alors $\tau_1 + \tau_2 + \tau_3$ est une trace valide.*

Démonstration. On raisonne par induction sur la structure de τ_1 : Si $\tau_1 = \epsilon$ alors τ_3 est valide (par définition) et donc $\tau_2 + \tau_3$ est valide (lemme 6.4.15). Si $\tau_1 = \nabla^{m'}_{t+v} \cdot \sigma$ alors soit $\sigma + (\nabla^m_{s+u} \cdot \tau_2) + (\Delta^m_s \cdot \tau_3)$ est valide, et par hypothèse d'induction $\sigma + \tau_2 + \tau_3$ est valide, donc par définition $\tau_1 + \tau_2 + \tau_3$ est valide. Soit $\tau_3 = \sigma' + (\Delta^{m'}_t \cdot \sigma'')$ avec σ'' et $\sigma + (\nabla^m_{s+u} \cdot \tau_2) + (\Delta_s \cdot \sigma')$ valides. Par hypothèse d'induction on a donc $\sigma + \tau_2 + \sigma'$ valide et donc $\tau_1 + \tau_2 + \tau_3$ est valide par définition. □

Lemme 6.4.17 *Soit τ_1, τ_2, σ des traces. Si $\tau_1 + \tau_2$, $\tau_1 + \sigma$ et σ sont valides, alors $\tau_1 + \sigma + \tau_2$ est valide.*

Démonstration. On procède par induction sur la structure de τ_1 : si $\tau_1 = \epsilon$ alors la propriété est immédiatement satisfaite par le lemme 6.4.15. Si $\tau_1 = \nabla^m_s \cdot \tau'_1$, nous pouvons distinguer plusieurs cas :
- Soit τ'_1 et τ_2 sont valides. Alors la propriété est vraie par hypothèse d'induction.
- Soit τ'_1 est valide et $\tau_2 = \tau'_2 + (\Delta^m_{s'} \cdot \tau''_2)$ avec $s' \leq s$, $\tau'_1 + \tau'_2$ et τ''_2 valides. Comme σ est valide, $\tau'_1 + \sigma$ est valide, donc par hypothèse d'induction $\tau'_1 + \sigma + \tau'_2$ est valide. Par définition on a donc $\tau_1 + \sigma + \tau_2$ valide.
- Soit $\tau'_1 = \tau''_1 + (\Delta^m_{s'} \cdot \tau'''_1)$ avec $s' \leq s$ et τ''_1 valide. Par définition et d'après les hypothèses on a donc $\tau'''_1 + \tau_2$ valide et $\tau''' + \sigma$ valide. Par hypothèse d'induction, ceci nous donne $\tau'''_1 + \sigma + \tau_2$ valide, et donc comme par définition $\nabla^m_s \cdot (\tau''_1 + (\Delta^m_{s'} \cdot \epsilon))$ est valide, on a $\tau_1 + \sigma + \tau_2$ valide (lemme 6.4.15).

□

Lemme 6.4.18 *Si $\sigma = \tau_1 + (\nabla^m_s \cdot \Delta^{m'}_t \cdot \tau_2)$ est valide avec $t \not\leq s$, alors $\sigma' = \tau_1 + (\Delta^{m'}_t \cdot \nabla^m_s \cdot \tau_2)$ est valide.*

Démonstration. Comme σ est valide, on a $\tau_1 = \tau'_1 + (\nabla^{m'}_{t'} \cdot \tau''_1)$ avec $t \leq t'$ et $\tau''_1 + (\nabla^m_s \cdot \epsilon)$ valide. On a donc τ''_1 valide. On procède par induction sur τ'_1.
- $\tau'_1 = \epsilon$: alors τ_2 est valide, et par construction σ' est valide.
- $\tau'_1 = \nabla^n_u \cdot \delta$, on doit considérer plusieurs cas :
  - $\delta + (\nabla^m_s \cdot \Delta^{m'}_t \cdot \tau_2)$ est valide : alors par hypothèse d'induction $\delta + (\Delta^{m'}_t \cdot \nabla^m_s \cdot \tau_2)$ est valide et par construction, σ' est valide.
  - $\delta = \delta' + (\Delta^n_{u'} \cdot \sigma'')$ avec $u' \leq u$ et δ' valide. Alors $\sigma'' + (\nabla^m_s \cdot \Delta^{m'}_t \cdot \tau_2)$ est valide et par hypothèse d'induction $\sigma'' + (\Delta^{m'}_t \cdot \nabla^m_s \cdot \tau_2)$ est valide. Par construction on a donc σ' valide.
  - $\tau_2 = \tau'_2 + (\Delta^n_{u'} \cdot \tau''_2)$ avec $u' \leq u$ et $\delta + (\nabla^m_s \cdot \Delta^{m'}_t \cdot \tau'_2)$ valide (on a aussi τ''_2 valide). Alors par hypothèse d'induction, $\delta + (\Delta^{m'}_t \cdot \nabla^m_s \cdot \tau'_2)$ est valide, et par construction σ' est valide.

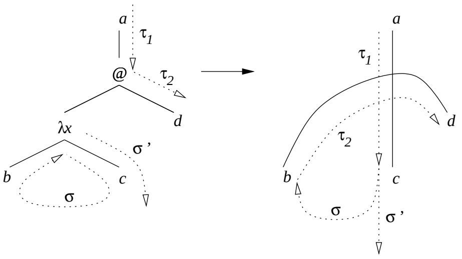
□

Théorème 6.4.19

1. *Tout chemin initial valide est le préfixe d'un chemin final valide.*
2. *S'il existe un chemin clos reliant un port a à un port b, alors il existe (au moins) un chemin clos valide reliant les mêmes ports.*

Démonstration. Ces propriétés sont satisfaites pour tout réseau initial, car tous les chemins d'un réseau initial sont valides, et tout chemin initial est le préfixe d'un chemin final car il n'existe pas de chemin cyclique[6]. Nous supposons à présent qu'elles sont satisfaites pour un réseau donné, et nous montrons qu'après une étape de réduction de ce réseau elles restent satisfaites. Pour cela nous étudions chacune des règles.

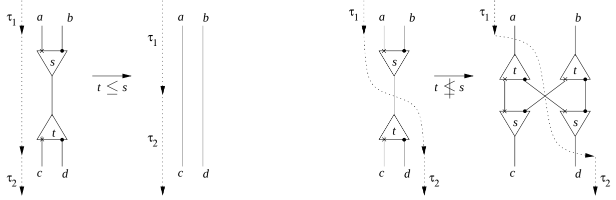
– (AbAp) :



1. Par hypothèse, si τ_1 est la trace d'un chemin initial valide, alors il existe τ_2 tel que $\tau_1+\tau_2$ est la trace d'un chemin final valide, et il existe σ et/ou σ' tels que $\tau_1+\sigma$ et/ou $\tau_1+\sigma'$ est la trace d'un chemin final valide. Si un chemin clos existe, alors σ existe et est une trace valide d'après l'hypothèse d'induction sur 2. Par conséquent, d'après le lemme 6.4.15 $\tau_1+\sigma$ est la trace d'un chemin final valide, et d'après le lemme 6.4.17 $\tau_1+\sigma+\tau_2$ est la trace d'un chemin final valide.
2. Si $\tau_1+\tau_2$ est la trace d'un chemin clos valide, alors deux cas sont à considérer : il existe σ tel que sur la figure, alors d'après le même raisonnement que pour le point précédent, on a $\tau_1+\sigma+\tau_2$ valide ; il n'existe pas de tel σ, alors il n'existe plus de chemin clos dont τ_1 est le préfixe. Si $\tau_1+\sigma'$ est la trace d'un chemin clos valide, un chemin de même trace est préservé après réduction.

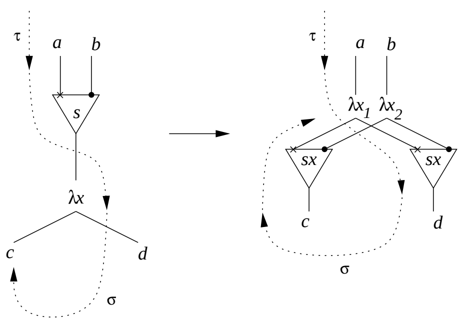
– Les règles (FoAp, AdAp) n'apportent pas de modifications aux chemins existant dans le réseau.

– (FoFi1, FoFi2) :



La démonstration utilise le fait que si $\tau_1+(\nabla_s^m \cdot \Delta_t^m \cdot \tau_2)$ est valide, alors $\tau_1+\tau_2$ est valide pour le cas $t \leq s$ (lemme 6.4.16), et utilise le lemme 6.4.18 pour l'autre cas. Les détails de la démonstration sont laissés au lecteur.

– (AbFi) :



[6] Nous n'avons pas démontré cela dans le cas général. Cependant, c'est trivialement vrai dans le cas des réseaux initiaux.

Si σ et τ sont valides, alors par définition $\tau + (\nabla_s^m \cdot \sigma)$ est valide, de même que $\tau + (\nabla_{sx}^m \cdot \sigma) + (\Delta_{sx}^m \cdot \epsilon)$.
– (AdFi) n'entraîne pas de changements sur les traces.

□

6.5 Calcul de la forme normale forte

RINO est un système extrêmement simple, et permet de calculer *au moins* la forme normale de tête du réseau donné en entrée. Cependant il ne permet pas de calculer sa forme normale forte. En effet, le calcul peut se terminer parce qu'un redex qui se trouve en position d'argument ne peut apparaître explicitement, comme illustré sur la figure 6.13. Dans le dernier réseau, un redex est caché : c'est le redex $(\lambda x_2. \dots)(\lambda u.u)$. Le blocage est dû au fait que fan-in et fan-out ne se trouvent pas tous deux sur une même *arête* (nous entendons par arête l'analogue des arêtes des termes du λ-calcul, voir définition 1.4.24). L'application située sous le fan-in appartenant à la forme normale du réseau, sa transformation en application duplicable est impossible (il n'y a pas de fan-out sous son fils gauche).

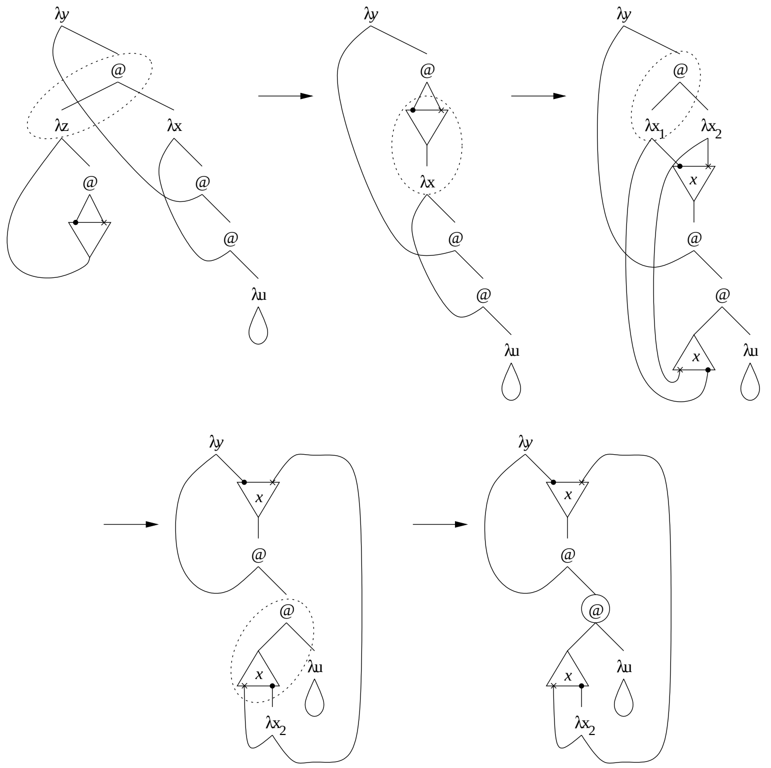


FIG. 6.13 – Réseau dont on ne sait pas calculer la forme normale forte

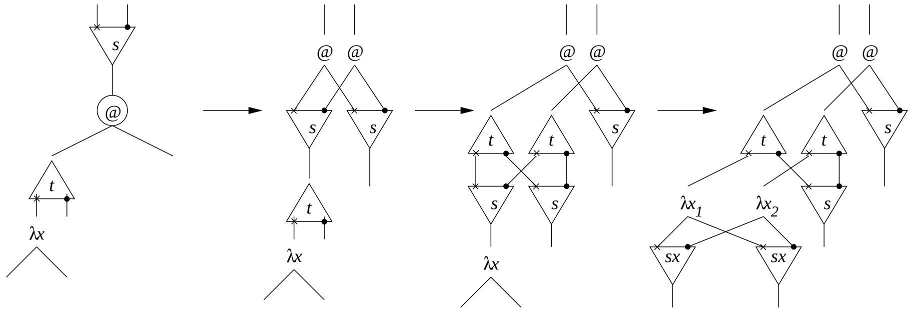
Une solution pour prolonger l'évaluation serait de décompiler le réseau obtenu comme résultat de l'évaluation de tête, puis de le recompiler sous la forme d'un réseau sans fan-out ; on peut alors calculer la forme normale de tête des arguments. On recommence le processus jusqu'à avoir atteint la forme normale forte. Bien entendu, cette solution n'est pas efficace : la décompilation et la recompilation d'un réseau sont des étapes coûteuses, et surtout elles provoquent la perte de tout le partage obtenu jusque là. On peut sans doute trouver de meilleures solutions, comme par exemple une fonction de décompilation plus fine, qui permettrait de préserver le partage lorsque cela est possible, en extrayant un graphe acyclique orienté au lieu d'un terme (ou un terme contenant des lets, ce qui revient au même). Une meilleure solution encore serait d'autoriser la duplication des symboles lorsqu'il est certain qu'ils ne feront pas partie d'un redex. C'est le cas par exemple des applications (multiples) dont le symbole de tête est une variable dont le lieur ne peut pas être impliqué dans une réduction. Dans ce cas, les applications peuvent être considérées comme duplicables, ce qui permet de débloquer la réduction. Dans le cas de la figure 6.13, la variable y en tête de l'application étant liée à un λ qui ne pourra pas être réduit, l'application en question peut être dupliquée, ce qui permettra l'interaction des fans et le prolongement de la réduction de l'argument de l'application.

Un tel algorithme pourrait s'avérer être un bon candidat pour l'implantation d'un réducteur fort dans le cadre d'un logiciel d'aide à la démonstration.

6.6 Partage

Nous discutons dans cette section le partage qui est réalisé par RINO. Nous nous restreignons pour cette discussion à la réduction de tête faible par la stratégie du redex le plus à gauche et le plus externe[7].

Comme nous l'avons déjà noté à plusieurs reprises, RINO n'est bien sûr pas optimal, au sens de Lévy, puisque des redexes potentiels peuvent être copiés. C'est le cas par exemple dans ce qui suit



où un redex potentiel $(\lambda x.\_)\_$ est copié, ses copies étant de la forme $(\lambda x_1.\_)\_$ et $(\lambda x_2.\_)\_$.

Cependant, il est clair qu'avec RINO, un redex effectif, c'est-à-dire un redex effectivement présent dans le réseau, ne peut pas être copié. En effet, lorsqu'un tel redex existe, la seule possibilité est de le réduire, par la règle (AbAp). Ceci est illustré par l'exemple suivant, qui montre que des redex effectifs ne pouvant pas être partagés dans le cadre d'un calcul de substitution explicite avec adresses peuvent l'être dans le cadre de RINO.

Exemple 6.6.1 *Réduisons le terme* $H = ((\lambda f.f\,(f I))\,G)[\mathrm{id}]^a$ *par la stratégie CBNeedG du calcul* $\lambda\sigma_w^a$*, présenté section 2.5, où nous remplaçons les indices de de Bruijn par des variables pour une*

[7] *Leftmost outermost redex.*

meilleure lisibilité.

$$
\begin{array}{rlr}
H & \xrightarrow{\text{CBNeedG}} \big((\lambda f.f\,(f\,I))[\mathrm{id}]^b\,G[\mathrm{id}]^c\big)^a & (\mathrm{App}) \\
& \xrightarrow{\text{CBNeedG}} \big(f\,(f\,I)\big)[f \mapsto G[\mathrm{id}]^c]^a & (\mathrm{B}_w) \\
& \xrightarrow{\text{CBNeedG}} \big(f[f \mapsto G[\mathrm{id}]^c]^d\,(f\,I)[f \mapsto G[\mathrm{id}]^c]^e\big)^a & (\mathrm{App}) \\
& \xrightarrow{\text{CBNeedG}} \big((\lambda x.I\,x)[\mathrm{id}]^c\,(f\,I)[f \mapsto G[\mathrm{id}]^c]^e\big)^a & (\mathrm{FVarG}) \\
& \xrightarrow{\text{CBNeedG}} (I\,x)\big[\underbrace{x \mapsto (f\,I)[f \mapsto G[\mathrm{id}]^c]^e}_{s}\big]^a & (\mathrm{B}_w{*}) \\
& \xrightarrow{\text{CBNeedG}} \big((\lambda y.y)[s]^g\,x[s]^h\big)^a & (\mathrm{App}) \\
& \xrightarrow{\text{CBNeedG}} y[y \mapsto x[s]^h; s]^a & (\mathrm{B}_w) \\
& \xrightarrow{\text{CBNeedG}} x\big[x \mapsto (f\,I)[f \mapsto G[\mathrm{id}]^c]^e\big]^h & (\mathrm{FVarG}) \\
& \xrightarrow{\text{CBNeedG}} (f\,I)[f \mapsto G[\mathrm{id}]^c]^e & (\mathrm{FVarG}) \\
& \xrightarrow{\text{CBNeedG}} \big(f[f \mapsto G[\mathrm{id}]^c]^i\,I[f \mapsto G[\mathrm{id}]^c]^j\big)^e & (\mathrm{App}) \\
& \xrightarrow{\text{CBNeedG}} \big((\lambda x.I\,x)[\mathrm{id}]^c\,I[f \mapsto G[\mathrm{id}]^c]^j\big)^e & (\mathrm{FVarG}) \\
& \xrightarrow{\text{CBNeedG}} (I\,x)\big[x \mapsto I[f \mapsto G[\mathrm{id}]^c]^j\big]^e & (\mathrm{B}_w{*}) \\
& \xrightarrow{\text{CBNeedG}} \cdots &
\end{array}
$$

Les lignes signalées par une astérisque ($*$) *montrent que le redex* $(I\,x)$ *est réduit deux fois, dans des contextes différents.*

La réduction de H *par RINO suivant la stratégie du redex le plus à gauche et le plus externe est illustrée sur la figure 6.14. Il montre bien que* $(I\,x)$ *n'est réduit qu'une seule fois (partie grisée).*

Cependant, il existe des cas, au contraire, où des redex (potentiels) qui peuvent être partagés dans le cadre d'un calcul de substitution explicite ne peuvent pas l'être dans le cadre de RINO.

Exemple 6.6.2 *Le terme* $(\lambda x.x\,x)\,(\lambda y.(\lambda z.z\,z)\,(y\,(\lambda t.t)))$ *se réduit par RINO en* 12 *éliminations de* β*-redex, et par KP en* 8 *éliminations de* β*-redex.*

6.7 Banc d'essai

Nous donnons dans cet section un banc d'essai, afin d'avoir une idée des performances de RINO. Nous utilisons pour cela le banc d'essai réalisé par Andrea Asperti pour son implantation optimisée de l'algorithme optimal de Lamping [4]. Nous récupérons ces résultats et testons les mêmes termes d'une part avec la machine KP, et d'autres part avec une implantation de RINO (évaluation en appel par nom). Les mesures relevées pour la réduction par l'implantation optimale d'Asperti et pour RINO sont le nombre d'interactions. Pour la machine KP, nous comptons le nombre de pas de réduction. Ces mesures sont difficilement exploitables pour comparer les performances absolues d'exemples ponctuels, car il n'y a pas de commune mesure entre le coût d'une interaction et le coût d'un pas de réduction par une machine abstraite. De même, le coût moyen d'une interaction de l'implantation d'Asperti est différent du coût moyen d'une interaction de RINO, en raison des détails de l'implantation. Cependant, le but de ces tests est plutôt de comparer comment évoluent ces mesures quand la complexité des termes donnés en entrée augmente. Nous n'avons pas non plus comparé les performances en temps d'exécution, en raison des grandes différences résidant dans les différentes implantations : l'implantation d'Asperti est réalisée dans un langage compilé (le langage C), alors que l'implantation de RINO est une implantation expérimentale réalisée dans un langage

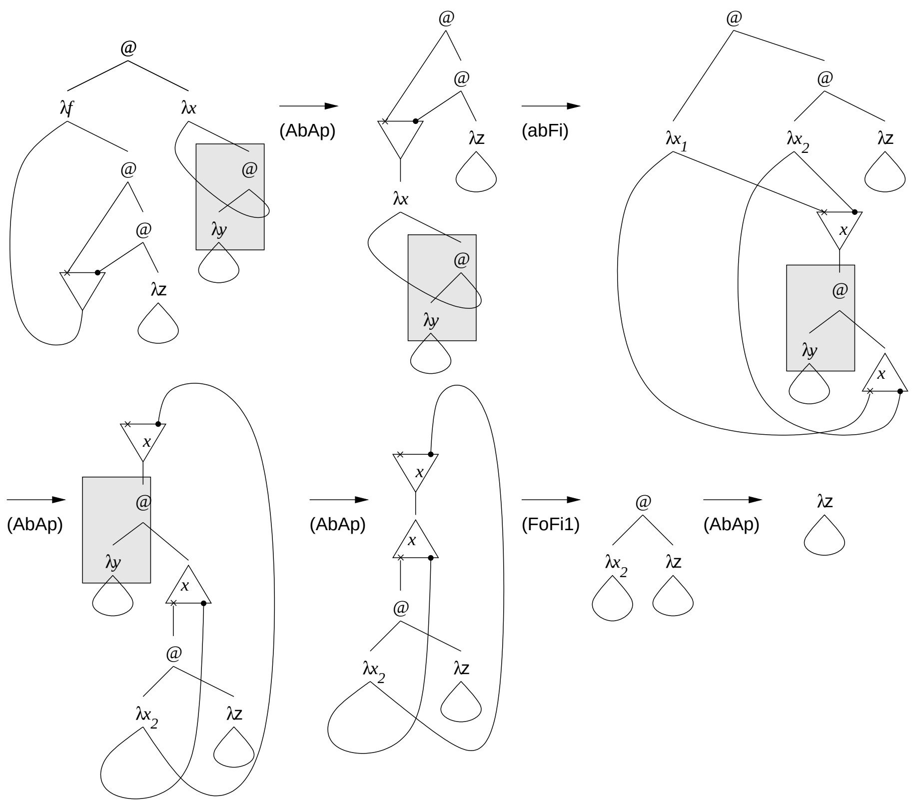


FIG. 6.14 – Partage de redex interne au corps d'une fonction

interprété (le langage Java[8]). la machine KP quand à elle est implantée en caml sous la forme d'un interprète de λ-termes.

Dans un premier temps, le banc d'essai montre les performances des trois implantations sur trois séries de termes : une série de factorielles (figure 6.15), une série de nombres de Fibonacci (figure 6.16), et une troisième série, particulièrement favorable à l'implantation optimale d'Asperti (figure 6.17). Nous verrons que cette troisième série est également très favorable à RINO.

Définition 6.7.1 *On définit les λ-termes suivants :*

$$\begin{aligned}
\mathsf{I} &= \lambda x.x \\
\mathsf{Succ} &= \lambda nxy.x\ (n\ x\ y) \\
\mathsf{Mult} &= \lambda nmx.n\ (m\ x) \\
\mathsf{Pair} &= \lambda xyz.z\ x\ y \\
\mathsf{Fst} &= \lambda xy.x \\
\mathsf{Snd} &= \lambda xy.y \\
\mathsf{Add} &= \lambda mnxy.m\ x\ (n\ x\ y) \\
\overline{0} &= \lambda fx.x \\
\overline{1} &= \lambda fx.f\ x \\
\overline{2} &= \lambda fx.f\ (f\ x) \\
&\vdots \\
\overline{n} &= \lambda fx.\underbrace{f\ (f\ (\ldots (f}_{n}\ x)))
\end{aligned}$$

(Les $\overline{n}$ sont les numéraux de Church.)

Définition 6.7.2 (Factorielle) *La factorielle est le terme défini par*

$$\mathsf{Fact} = \lambda n.n\ \mathsf{NextFact}\ (\mathsf{Pair}\ \overline{1}\ \overline{0})\ \mathsf{Fst}$$

où NextFact *est défini par*

$$\mathsf{NextFact} = \lambda p.(\lambda n_1 n_2.\mathsf{Pair}\ (\mathsf{Mult}\ n_1\ n_2)\ n_2)\ (p\ \mathsf{Fst})\ (\mathsf{Succ}\ (p\ \mathsf{Snd}))$$

Définition 6.7.3 (Fibonacci) *La fonction de Fibonacci est le terme défini par*

$$\mathsf{Fibo} = \lambda n.n\ \mathsf{NextFibo}\ (\mathsf{Pair}\ \overline{0}\ \overline{1})\ \mathsf{Fst}$$

où NextFibo *est défini par*

$$\mathsf{NextFibo} = \lambda p.(\lambda n_1 n_2.\mathsf{Pair}\ (\mathsf{Add}\ n_1\ n_2)\ n_1)\ (p\ \mathsf{Fst})\ (p\ \mathsf{Snd})$$

Les chiffres indiquent, pour RINO et l'implantation optimale d'Asperti le nombre d'interactions, et pour KP le nombre de pas de réduction. Le nombre entre parenthèses est, pour les trois implantations, le nombre de β-redexes contractés au cours de la réduction. Dans le tableau de la figure 6.15,

[8]Nous avons initialement choisi d'implanter RINO en Java pour plusieurs raisons : sa portabilité, qui nous permet de le faire tourner sur n'importe quelle plateforme ; ses aspects de langage à objets, le haut niveau permettant de réduire l'effort de programmation, les nœuds d'interaction pouvant être définis directement comme des objets ; enfin, pour sa grande popularité et le sentiment de l'auteur qu'il est vraiment indispensable de se former à un langage de programmation faisant si grand bruit ;-)

Entrée	RINO	Asperti	KP
Fact $\overline{1}$ I I	54 (30)	273 (20)	128 (30)
Fact $\overline{3}$ I I	312 (65)	983 (45)	421 (86)
Fact $\overline{5}$ I I	1 053 (127)	2 380 (74)	3 353 (550)
Fact $\overline{7}$ I I	16 068 (999)	5 625 (107)	116 175 (17 862)
Fact $\overline{9}$ I I	813 906* (46 391)	12 402 (144)	8 044 237 (1 227 494)
Fact $\overline{10}$ I I	explosion	17 916 (164)	explosion
Fact $\overline{20}$ I I		307 839 (425)	

FIG. 6.15 – Comportement de la fonction factorielle

Entrée	RINO	Asperti	KP
Fibo $\overline{1}$ I I	48 (28)	248 (20)	120 (28)
Fibo $\overline{4}$ I I	348 (72)	868 (55)	455 (93)
Fibo $\overline{7}$ I I	865 (124)	2 209 (98)	1 350 (246)
Fibo $\overline{10}$ I I	2 346 (208)	6 456 (173)	4 581 (767)
Fibo $\overline{13}$ I I	7 911 (428)	24 487 (390)	17 716 (2 848)
Fibo $\overline{16}$ I I	30 776 (1 224)	95 199 (1 177)	72 803 (11 537)
Fibo $\overline{19}$ I I	126 925 (4 460)	385 911 (4 404)	305 602 (48 218)

FIG. 6.16 – Comportement de la fonction de Fibonacci

Entrée($\overline{m}$)	RINO	Asperti	KP
$\overline{1}$	9 (7)	78 (7)	33 (7)
$\overline{4}$	54 (16)	221 (16)	244 (38)
$\overline{7}$	99 (25)	386 (25)	1 827 (265)
$\overline{10}$	144 (34)	551 (34)	14 386 (2 060)
$\overline{13}$	189 (43)	831 (49)	114 753 (16 399)
$\overline{16}$	234 (52)	973 (58)	917 584 (131 090)
$\overline{19}$	279 (61)	1 106 (67)	7 340 127 (1 048 597)

FIG. 6.17 – Comportement de $(\lambda n.n\ \overline{2}\ \mathsf{I}\ \mathsf{I})\ \overline{m}$

l'astérisque indique que le nombre inscrit correspond au nombre d'interactions réalisées avec une implantation optimisée. Le nombre d'interactions réalisées par l'implantation non optimisée devrait être légèrement supérieur, mais n'a pas pu être calculé. Dans le tableau de la figure 6.17, le nombre de β-contractions réalisées par RINO apparaît à certaines lignes comme inférieur au nombre optimal. Ce n'est pas un miracle, mais simplement dû au fait que les termes réduits par Asperti ne sont pas exactement ceux donnés en entrée du tableau. Ce sont des termes équivalents par quelques β-contractions[9].

Ces résultats montrent que sur ces exemples le partage de redexes est bien plus important par RINO que par les techniques classiques de réduction, représentées par KP. Dans tous les cas, le nombre de β-redexes contracté est inférieur pour RINO, qui réalise d'ailleurs la réduction optimale pour les termes de la famille $(\lambda n.n\ \overline{2}\ \mathsf{I}\ \mathsf{I})\ \overline{m}$, tout en réalisant un nombre d'interactions qui évolue de façon linéaire par rapport à l'entrée.

Dans le cas de la factorielle, le nombre d'interactions de RINO évolue de manière exponentielle par rapport à l'entrée, tout comme dans une implantation classique. Seule l'implantation optimale ne montre pas une courbe exponentielle. Il est à noter cependant que ni pour RINO, ni pour l'implantation optimale, le nombre d'intéractions n'augmente linéairement par rapport au nombre de β-contractions.

Dans le cas de Fibonacci, le rapport entre le nombre de β-contractions effectuées par RINO et le nombre de β-contractions effectuées par l'implantation optimale semble converger vers 1. On peut donc dire que, dans ce cas précis, RINO est très proche de l'optimalité.

Cependant, il existe des exemples pour lesquels le nombre de β-redex éliminés par RINO est beaucoup plus important que le nombre de β-redex éliminés par la machine KP. C'est le cas de la fonction suivante définie avec les numéraux de Barendregt.

Définition 6.7.4 (Test des nombres premiers) *On donne une implantation de la fonction* prime *qui teste de façon naïve si un entier (de Barendregt) est premier, comme suit :*

$$\text{letrec } \mathsf{prime} = \lambda n.\lambda m.\text{if } (m \geq n) \text{ then } \mathsf{true} \text{ else } (\text{if } (\mathsf{div}\ m\ n) \text{ then } \mathsf{false} \text{ else } (\mathsf{prime}\ n\ (\mathsf{succ}\ m)))$$

appelée initialement sous la forme prime $\hat{n}\ \hat{2}$ *où pour tout* $m \geq 0, \hat{m}$ *désigne l'entier de Barendregt* m*, et où* $\hat{n}$ *est le nombre dont on veut savoir s'il est premier. La définition en* λ*-calcul des diverses fonctions et constructions du langage utilisées est donnée en annexe C.*

Le tableau de la Figure 6.18 montre que RINO peut dupliquer de nombreux redex potentiels qui ne sont pas dupliqués par KP.

6.8 Conclusion

Nous avons proposé dans ce chapitre un algorithme simple, RINO, basé sur les réseaux d'interaction. Sa particularité est son comportement original en terme de partage des β-redex. En effet, aucun β-redex effectif initialement dans un terme ne peut être dupliqué, en raison des propriétés des réseaux d'interaction. Ce n'est pas le cas des implantations basées sur les substitutions explicites où les β-redex effectifs situés dans le corps des fonctions peuvent être dupliqués. Pourtant, il existe des situations où le partage de β-redex (potentiels) est plus important dans le cadre des substitutions explicites, d'autres où il est plus important dans le cadre des réseaux d'interaction. La raison se situe au niveau de la sémantique du partage, très différente dans un cas et dans l'autre.

[9] Ce fait nous a été confirmé par Andrea Asperti. Les termes qu'il a testés à partir de 10 sont $(\mathsf{Add}\,\overline{p}\,\overline{10})$ pour p valant $3, 6, 9$ (communication personnelle).

Entrée($\hat{m}$)	RINO	KP
$\hat{1}$	251 (48)	300 (50)
$\hat{2}$	458 (73)	484 (75)
$\hat{3}$	2 940 (361)	2 353 (332)
$\hat{5}$	13 567 (1 554)	8 150 (1 124)
$\hat{7}$	37 659 (4 076)	16 832 (2 320)
$\hat{11}$	149 060 (14 963)	42 083 (5 820)
$\hat{13}$	258 083 (24 951)	59 292 (8 216)
$\hat{17}$	explosion	100 622 (13 980)
$\hat{19}$		125 405 (17 440)
$\hat{23}$		182 380 (25 408)
$\hat{31}$		330 529 (46 172)

FIG. 6.18 – Comportement de la fonction prime

Conclusion

Nous avons décrit plusieurs modèles opérationnels pour la β-réduction, destinés à la mise en œuvre des évaluateurs dans les logiciels se basant sur une représentation des objets du calcul par des λ-termes. Il s'agit essentiellement des langages de programmation fonctionnelle, et des outils d'aide à la démonstration. La notion de réduction mise en œuvre par ces deux types d'outils est différente : les premiers se contentent de la β-réduction faible, les seconds ont besoin d'une notion de réduction plus forte. D'autre part, les propriétés attendues des modèles utilisés dépendent également des utilisations qui en sont faites.

Programmation fonctionnelle

Les calculs de substitution explicite sont un bon outil pour la description d'implantations de langages de programmation fonctionnelle. Cela a déjà été mis en évidence pour un certain nombre de machines abstraites dans [40], où il est démontré que chacune d'elle met en œuvre une stratégie du calcul $\lambda\sigma_w$. Ce calcul permet une représentation des clôtures proche de la réalité des implantations, et fait une distinction claire entre code (λ-termes purs) et données (clôtures). Cette distinction est très importante car elle permet de faire un distinguo entre la partie statique et la partie dynamique du programme.

Toutefois, aucune des machines considérées ne met en œuvre le partage, car $\lambda\sigma_w$ ne permet pas de donner une sémantique précise de l'appel par nécessité. $\lambda\sigma_w^a$ pallie ce manque : par l'introduction des adresses et d'une sémantique claire du partage, les stratégies d'appel par nécessité peuvent être exprimées dans le cadre des calculs de substitution explicite. Ce formalisme a même permis de mettre en évidence de manière claire des nuances qui existaient dans la pratique, mais pour lesquelles aucun modèle unifié n'avait été proposé. C'est le cas notamment de la règle d'accès à une variable, différente dans le cas des réducteurs de graphes et des machines à environnement.

Nous avons contribué dans cette thèse à montrer que $\lambda\sigma_w^a$ est effectivement adapté à la description des implantations de langages de programmation fonctionnelle paresseux. Nous avons donné la preuve que la machine KP met en œuvre une stratégie d'appel par nécessité, notamment l'appel par nécessité des machines à environnement. Cette preuve d'une machine somme toute assez simple (elle ne met en œuvre aucune optimisation, le code est un simple λ-terme enrichi d'un opérateur de récursion et de structures de données algébriques) donne des éléments sur une méthode à suivre pour prouver la correction d'autres machines, plus complexes, ou basées sur d'autres principes, comme les réducteurs de graphes. En particulier, il serait sans doute intéressant de donner une preuve de correction d'une machine utilisée pour l'implantation d'un langage de programmation fonctionnelle paresseux efficace, par exemple le langage Haskell avec la machine STG, ou encore de donner une preuve de la première machine efficace proposée pour la mise en œuvre de la réduction paresseuse, un réducteur de graphes, la G-machine. À l'heure actuelle, une propriété importante de STG a été démontrée formellement grâce au calcul $\lambda\sigma_w^a$: la preuve qu'elle ne provoque pas de fuites de mémoire [15, 14].

D'un point de vue pédagogique, nous pensons en outre que $\lambda\sigma_w^a$ est un excellent outil pour une description formelle, claire et précise, des principales notions utiles dans l'implantation des langages de programmation fonctionnelle. Il aurait ainsi sa place dans le cadre d'un cours approfondi sur ce sujet (cours de DEA ou école d'été, livre d'enseignement, article didactique dans un *handbook*, etc.). Nous envisageons également de réutiliser les principes de substitution explicite et adresses pour modéliser les implantations des langages à objets.

D'autre part, nous avons proposé dans cette thèse une modification de la machine KP, mettant en œuvre les super-clôtures. Les super-clôtures se basent sur l'idée que la création des clôtures peut se faire par nécessité, afin d'éviter la création de clôtures inutiles. Bien entendu, ceci est au prix de la création des super-clôtures, toutefois moins volumineuses et moins nombreuses que les clôtures. La machine proposée a été démontrée correcte vis à vis de KP. Cependant, c'est une machine purement théorique, et un grand nombre de problèmes d'implantation n'ont pas été abordés, comme notamment les problèmes de fuite de mémoire, ou d'élimination d'indirections. De plus, seule une implantation de cette machine abstraite pourrait démontrer ses performances, et permettre de mettre en évidence ses qualités réelles, ses avantages et ses inconvénients vis à vis d'une machine équivalente sans super-clôtures. De plus, cette comparaison, pour être honnête, ne doit être faite qu'avec une machine déjà très performante, comme la STG, dans laquelle les super-clôtures pourraient être mises en œuvre. On peut imaginer que cette machine, dans la pratique, ne démontre une efficacité réelle que dans certains cas particuliers. Peut être peut on considérer ce résultat comme un échec. Nous le voyons plutôt comme un renforcement de la thèse défendue dans ce document : nous avons commencé par considérer un modèle excessivement simplifié des machines à environnement, qui ne permet pas de décrire le partage des implantations de langages ; ceci nous a amené à imaginer des optimisations, qui sont réellement des optimisations pour le modèle pris en compte, mais pas dans la réalité. Un modèle plus complet comme $\lambda\sigma_w^a$ permet de mieux comprendre les mécanismes impliqués dans les implantations.

Outils d'aide à la démonstration

La seconde utilisation principale du λ-calcul et de la β-réduction se fait dans le cadre des outils d'aide à la démonstration, où les λ-termes codent les preuves. La notion de réduction utilisée est la β-réduction forte, qui permet de normaliser les preuves. Nous avons abordé dans cette thèse deux problèmes relatifs à la mise en œuvre de la β-réduction forte.

Le premier problème est un problème de représentation des preuves incomplètes. La question posée est *comment représenter une preuve incomplète dans le λ-calcul, et comment la normaliser ?* Nous n'avons pas entièrement répondu à cette question, qui est somme toute complexe, mais avons apporté des contributions non négligeables dans ce domaine. La contrainte que nous nous sommes fixée est la définition d'un calcul qui ait à la fois la propriété de préserver la forte normalisation du λ-calcul, et la propriété de confluence sur la réduction des preuves incomplètes. La préservation de la forte normalisation permet de ne pas risquer d'introduire de sources de réduction infinies qui ne soient pas celles du λ-calcul, afin de garantir la terminaison de la réduction de certains termes, notamment les termes typés. La confluence assure l'unicité de la forme normale (si elle existe) d'une preuve incomplète. Cette contrainte est accentuée par la volonté de définir un calcul qui permette de mettre en œuvre toutes les stratégies de réduction du λ-calcul. On dit que le calcul recherché simule le λ-calcul pas à pas. Plusieurs tentatives ont été faites pour la définition d'un tel calcul, qui ont toutes échoué sur une de ces trois propriétés. Nous avons défini dans cette thèse λxc1, un calcul qui peut être considéré comme le calcul de substitution explicite minimal disposant d'ingrédients essentiels pour obtenir ces propriétés : il contient λx, qui est le calcul minimal simulant β pas à pas ; il dispose d'une règle de permutation, nécessaire pour obtenir la confluence sur les preuves incomplètes ; cette règle de permutation est contrôlée, ce qui est supposé éviter de nouvelles sources de réduction infinies. La preuve de préservation de la forte normalisation pour λxc1 reste malheureusement un problème ouvert. En effet, nous ne disposons pas des outils et des techniques nécessaires pour réaliser une telle preuve. Ce travail constitue un de nos objectifs pour l'avenir. Nous espérons ainsi que, de la même manière que la preuve de PSN du calcul λx peut être adaptée à la preuve de PSN d'autres calculs sans permutations, une preuve de PSN de λxc1 pourrait être utile à la preuve de PSN d'autres calculs avec règle de permutation. Bien entendu, prouver la PSN de λxc1 étant un problème ouvert, il est tout à fait possible qu'un jour ou l'autre apparaisse un contre exemple de cette propriété. Alors, nous pensons que ce contre exemple répondra de manière négative, et définitive, à la question de l'existence d'un calcul tel que nous cherchons à le définir. Bien que nous penchions plutôt vers l'idée que λxc1 a PSN, nous n'avons pas voulu en faire une conjecture, l'histoire des substitutions explicites ayant démontré à plusieurs reprises que de telles conjectures sont risquées.

La deuxième étape de notre travail a été de définir un calcul confluent sur les preuves incomplètes, préservant la forte normalisation de λxc1. Nous avons montré qu'il est alors nécessaire de définir une notion de variables qui peuvent être libres dans les instances de la preuve : les variables potentiellement libres. De ce fait, nous avons montré qu'il est nécessaire de pouvoir différencier les variables représentant les trous dans les preuves (les metavariables dites syntaxiques) des metavariables classiques, ces deux types de metavariables ayant des statuts différents. Nous avons mis en évidence l'intérêt pour les metavariables syntaxiques de transporter de l'information sur les variables potentiellement libres. Nous avons donc défini le calcul λxci, et démontré qu'il est confluent sur les termes avec metavariables syntaxiques et qu'il préserve la forte normalisation de λxc1. L'étape suivante a été la définition d'un calcul avec renommages explicites, c'est à dire utilisant la représentation des variables de de Bruijn. Ce calcul, λ$x_{\mathcal{B}}$ci, préserve la forte normalisation de λxci (donc de λxc1). Nous n'avons cependant pas encore démontré sa confluence sur les termes avec metavariables syntaxiques, cette preuve nécessitant une démonstration quelque peu laborieuse de lemmes structurels. Réaliser cette preuve est également un de nos objectifs à court terme.

Le second problème relatif à la démonstration, et abordé dans cette thèse, est celui de la mise en œuvre efficace de la β-réduction forte. Par mise en œuvre efficace, nous entendons une mise en œuvre qui permette un bon partage des calculs, sans introduire une complexité de réduction trop importante. Nous avions déjà abordé ce problème dans le cadre du DEA [51], où les aspects de partage ont été incorporés à une machine de réduction forte en ordre normal des λ-termes. Si le partage permettait dans ce cas une amélioration importante de la complexité de la réduction, nous avions cependant mis en évidence la difficulté de mise en œuvre d'un bon partage. Nombreuses sont les duplications de redex rendues nécessaires par le formalisme des substitutions explicites. Les réseaux d'interaction apportent des solutions intéressantes pour la mise en œuvre d'un partage plus fin. Ils ont été utilisés notamment dans le cadre de la définition de réducteurs optimaux, comme celui de Lamping. Cependant, de tels réducteurs s'avèrent assez complexes à comprendre (dans leur version non simpliste) et à mettre en œuvre. De plus, la décompilation des graphes obtenus par réduction de termes par ces évaluateurs est très complexe. Or, si cette décompilation n'est pas forcément nécessaire dans le cadre de la programmation fonctionnelle[10], elle peut l'être dans le cadre de la démonstration (l'objectif est de pouvoir manipuler le terme en forme normale).

[10]Le résultat d'un programme fonctionnel est soit un objet simple (une constante, une liste, etc.), auquel cas la décompilation est aisée, soit un objet fonctionnel, auquel cas son type, connu statiquement, est suffisant comme résultat retourné à l'utilisateur, sa représentation en mémoire n'ayant pas besoin d'être décompilée.

Nous avons proposé dans cet thèse une alternative aux substitutions explicites et aux évaluateurs optimaux, basée également sur une représentation des termes par des réseaux d'interaction. Cet évaluateur permet, comme les évaluateurs optimaux, de partager non seulement des termes, mais également des contextes. Le degré de partage est bien entendu moins grand que pour les implantations optimales, mais les problèmes engendrés sont aussi moins complexes à résoudre. En particulier, l'identification des fans se base sur un système d'étiquetage assez simple, et de ce fait, la procédure de décompilation est simple également. D'autre part, le nombre de nœuds d'interaction à créer pour représenter un terme est moins élevé. Cependant, contrairement à nos attentes, RINO ne permet pas un degré de partage intermédiaire entre substitutions explicites et réducteurs optimaux. Dans certains cas l'explosion combinatoire est inévitable mais sans commune mesure avec l'exposiion combinatoire des substitutions explicites, dans d'autres cas l'évaluateur atteint l'optimalité, dans d'autres cas enfin, RINO partage beaucoup moins de redex qu'une machine basée sur les substitutions explicites. Nous n'avons pas encore pu mettre en évidence les particularités des termes qui pourraient éventuellement permettre de prévoir le comportement de telle ou telle implantation en terme de partage. Ceci constitue un de nos travaux à venir.

D'autre part, il est nécessaire pour les applications qui lui pouraient être destinées, que RINO soit généralisé pour mettre en œuvre la β-réduction forte. Nous avons décrit rapidement dans la thèse comment nous comptions nous y prendre. Ceci constitue également un de nos travaux futurs.

Pour synthétiser, nous pensons avoir obtenu des résultats nouveaux dans les domaines suivants :

- la programmation fonctionnelle, avec :
  - la démonstration que $\lambda\sigma_w^a$est un bon outil de représentation de la sémantique opérationnelle des langages fonctionnels ;
  - la proposition d'une stratégie alternative de création des clôtures ;
- la démonstration assistée par ordinateur, avec :
  - l'ébauche de la définition d'un langage de représentation des preuves incomplètes ;
  - de nouvelles perspectives pour des algorithmes efficaces de normalisation de preuves, notamment basés sur les réseaux d'interaction.

Certains ont pu penser qu'avec l'arrivée en force des langages à objets d'une part, et des démonstrateurs performants (mais hélas logiquement fragiles) d'autre part, l'étude du λ-calcul et de son implantation perdaient de son intérêt. Nous espérons avoir démontré le contraire, et ouvert de nouvelles voies de recherche.

Annexe A

Confluence du calcul λxci

A.1 Preuve de confluence locale

Le but de cette section est de montrer que λxci est localement confluent sur les termes avec metavariables syntaxiques. Pour cela, il nous faut d'abord démontrer quelques propriétés importantes et non triviales sur les λxci-termes.

Définition A.1.1 *Nous définissons la notion de variable x* liée avec ensemble bloquant S et domaine V dans un terme M *comme suit :*

- *x est liée avec ensemble bloquant $\varnothing$ et domaine* $\text{pfv}(M)$ *dans $\lambda x.M[x]$;*
- *x est liée avec ensemble bloquant S et domaine* $\text{pfv}(M)$ *dans $M[x]\langle x := N\rangle_S$;*
- *si x est liée avec ensemble bloquant S et domaine V dans M alors elle est liée avec ensemble bloquant S et domaine V dans $MN, NM, N[y]\langle y := M\rangle_T, M[y]\langle y := N\rangle_T$ et $\lambda y.M[y]$.*

Si x est liée dans un sous-terme $M[y]$ d'un metaterme N, on dit que x est liée sous y *dans N. De plus, si ni x n'est liée sous y (dans M), ni y n'est liée sous x (dans M), on dit que x et y sont* sans relation *(dans M). Enfin, on dit que y est liée entre x et z (dans M) si x est liée sous y et y est liée sous z (dans M).*

Lemme A.1.2 *Soit x liée sous y dans M_1, et $M_1 \to M_2$. Si y est liée sous x avec domaine V_2 et ensemble bloquant S_2 dans M_2, alors*

$$x \in V_2 \Rightarrow x \in S_2$$

Démonstration. Par induction sur la longueur n de la dérivation de M_1 à M_2.

- $n = 1$: la règle utilisée pour passer de M_1 à M_2 est soit (xab) soit (c) ; dans les deux cas, la propriété est triviale ;
- supposons que la propriété soit satisfaite pour toute dérivation de longueur $n \leq N$;
- Soit $M_1 \xrightarrow{N+1} M_2$ où y est liée sous x avec domaine V_2 et ensemble bloquant S_2 dans M_2, et $x \in V_2$; on peut décomposer la réduction en $M_1 \xrightarrow{N} M' \to M_2$; soit V' et S' le domaine et l'ensemble bloquant de y dans M' ; comme les domaines décroissent par réduction et comme $x \in V_2$, alors $x \in V'$; de plus :
  - soit y est liée sous x dans M', et par l'hypothèse d'induction, $x \in S'$; la seule règle qui puisse modifier l'ensemble bloquant de y sur la réduction de M' à M_2 (c'est-à-dire sans modifier les positions relatives de x et y) est la règle (i), mais comme x appartient au domaine de y, (ı) ne peut pas être utilisée ; par conséquent, $x \in S_2$;
  - soit x est toujours liée sous y, et $M' \to M_2$ est de longueur 1 ;

– il faut remarquer que d'après l'hypothèse, x et y ne peuvent pas être sans relation dans M'.

□

Lemme A.1.3 *Soit x et y des variables sans relation liées dans M_1, et $M_1 \xrightarrow{+} M_2$. Si y est liée sous x avec ensemble bloquant S_2 et domaine V_2 dans M_2, alors*

$$x \notin S_2 \text{ and } x \in V_2 \Rightarrow \forall z \text{ lié sous } y \text{ avec ensemble bloquant } S_z \text{ dans } M_2, x \notin S_z$$

Démonstration. Par induction sur la longueur n de la dérivation de M_1 à M_2.

- $n = 1$: la règle utilisée pour passer de M_1 à M_2 est soit (xab) soit (c) :
  - (xab) $(\lambda x.M[x, y, z])\langle z := N[y]\rangle_S \to \lambda x.M[x, y, z]\langle z := N[y]\rangle_{S+x}$; comme N n'est pas dans la portée du lieur de la variable x dans le membre gauche de la règle, il est évident qu'il n'y a pas de variable liée dans N dont l'ensemble bloquant contienne x.
  - (c) $M[x, y, z]\langle x := N[y, z]\rangle_S\langle z := P[y]\rangle_T \to M[x, y, z]\langle z := P[y]\rangle_{T+x}\langle x := N[y, z]\langle z := P[y]\rangle_T\rangle_S$ avec y liée sous P ; la conclusion est la même que ci-dessus ;
- supposons que la propriété soit satisfaite pour toute dérivation de taille $n \leq N$;
- soit $M_1 \xrightarrow{N} M' \to M_2$ avec y liée sous x dans M_2, $x \notin S_2$ et $x \in V_2$:
  - soit y est liée sous x dans M' ; les hypothèses sur x et y sont trivialement respectées dans M', donc par l'hypothèse d'induction, la propriété est satisfaite dans M' ; il est facile de vérifier pour chaque règle qu'elles sont toujours satisfaites dans M_2 ;
  - soit x et y sont sans relation dans M' ; donc, par l'hypothèse d'induction, la propriété est satisfaite (la dérivation de M' à M_2 est de taille 1) ;
  - soit x est liée sous y dans M' : il est facile de vérifier qu'il n'existe pas de règle permettant à x et y de changer leurs positions relatives sous la condition que $x \notin S_2$.

□

Théorème A.1.4 *Soit x, y, z liées avec ensembles bloquants S_x, S_y et S_z et domaines V_x, V_y et V_z dans R.*

$$y \in V_y \setminus S_x \text{ and } z \in V_z \setminus S_y \Rightarrow z \notin S_x$$

Démonstration. Supposons $z \in S_x$. Alors, il existe un terme R_0 dans lequel z est liée sous x, et $R_0 \xrightarrow{+} R$ (ceci est trivial). Dans R_0, soit y est liée sous z, soit z est liée sous y, soit z et y sont sans relation. Nous procédons par cas :

- y est liée sous z dans R_0 : par conséquent y est liée sous x dans R_0, c'est-à-dire que les positions relatives de y et x ont changé entre R_0 et R. Donc, d'après le lemme A.1.2 $y \in S_x$, ce qui contredit l'hypothèse ;
- z est liée sous y dans R_0 : par conséquent, les positions relatives de z et y ont changé entre R_0 et R, et d'après le lemme A.1.2 $z \in S_y$, ce qui contredit l'hypothèse ;
- z et y sont sans relation dans R_0 : d'après le lemme A.1.3, il n'y a pas de variable liée sous y dont le label contienne z dans R ; ceci contredit l'hypothèse que $z \in S_x$.

□

En instantiant R en $M[x, y, z]\langle x := N[y, z]\rangle_{S[y,z]}\langle y := P[z]\rangle_{T[z]}\langle z := Q\rangle_U$ dans le théorème, on obtient le corollaire suivant.

Corollaire A.1.5 *Si $y \in \mathrm{pfv}(M[x, y, z]\langle x := N[y, z]\rangle_{S[y,z]}) \setminus S[y, z]$ et $z \in \mathrm{pfv}(M[x, y, z]\langle x := N[y, z]\rangle_{S[y,z]}\langle y := P[z]\rangle_{T[z]}) \setminus T[z]$ alors $z \notin S[y, z]$.*

Corollaire A.1.6 *Soit* $R = M[x,y,z]\langle x := N[y,z]\rangle_{S[y]+z}\langle y := P\rangle_T$.

$$y \in \mathrm{pfv}(M[x,y,z]\langle x := N[y,z]\rangle_{S[y]+z}) \setminus (S[y]+z) \Rightarrow z \notin \mathrm{pfv}(P)$$

Démonstration. Comme z appartient à l'ensemble bloquant de x, cela signifie que x est dans la portée du lieur de z, c'est-à-dire x est liée sous z. Par conséquent, on a x liée sous y et y liée sous z (en effet, sous l'hypothèse que $y \notin (S[y]+z)$ on a $y \neq z$). $y \in V_y \setminus S_x$ où V_y est le domaine de y, c'est-à-dire $\mathrm{pfv}(M[x,y,z]\langle x := N[y,z]\rangle_{S[y]+z})$, et S_x est l'ensemble bloquant de x, c'est-à-dire $(S[y]+z)$. Définissons V_z comme le domaine de z, et supposons $z \in \mathrm{pfv}(P)$. Alors $z \in V_z$ et $z \notin T$, c'est-à-dire $z \in V_z \setminus T$. D'après le théorème A.1.4 on a donc $z \notin S_x = S[y]+z$, ce qui contredit l'hypothèse que $z \in \mathrm{pfv}(P)$. □

On montre maintenant une des propriétés importantes du système :

Théorème A.1.7 (Confluence locale) $\xrightarrow[\text{bxci}]{}$ *est localement confluente sur les termes avec meta-variables syntaxiques.*

Démonstration. Il y a 14 paires critiques entre les règles de λxci, qui sont résolues dans la suite. Les corollaires A.1.5 et A.1.6 sont d'une grande utilité pour en résoudre certaines. □

Paire critique (b)-(xap)

Dans ce qui suit, nous devons considérer deux cas : $\dashrightarrow$ n'est suivi que dans le cas où $y \in \mathrm{pfv}(M[x,y]) \cup \mathrm{pfv}(N[y])$ et $\cdots\!\!>$ dans le cas où $y \notin \mathrm{pfv}(M[x,y]) \cup \mathrm{pfv}(N[y])$.

$$\begin{tikzcd}
((\lambda x.M[x,y])\,N[y])\langle y := P\rangle_S \arrow[r, "\text{(xap)}"] \arrow[d, "\text{(b)}"'] & (\lambda x.M[x,y])\langle y := P\rangle_S\, N[y]\langle y := P\rangle_S \arrow[d, "\text{(xab)}"] \\
M[x,y]\langle x := N[y]\rangle_\varnothing\langle y := P\rangle_S \arrow[d, dotted, "\text{(xgc)}"'] \arrow[dr, dashed, "\text{(c)}"] & (\lambda x.M[x,y]\langle y := P\rangle_{S+x})\,N[y]\langle y := P\rangle_S \arrow[d, "\text{(b)}"] \\
M[x,y]\langle x := N[y]\rangle_\varnothing & M[x,y]\langle y := P\rangle_{S+x}\langle x := N[y]\langle y := P\rangle_S\rangle_\varnothing \arrow[l, dotted, "\text{(xgc)}^2"]
\end{tikzcd}$$

Paire critique (xab)-(xgc)

$$\begin{tikzcd}
(\lambda x.M[x])\langle y := N\rangle_S \arrow[r, "\text{(xab)}"] \arrow[d, "\text{(xgc)}"'] & \lambda x.M[x]\langle y := N\rangle_{S+x} \arrow[dl, "\text{(xgc)}"] \\
\lambda x.M[x] &
\end{tikzcd}$$

Paire critique (xab)-(c)

Dans ce qui suit, $z \in \mathrm{pfv}((\lambda x.M[x,y,z])\langle y := N[z]\rangle_S) \setminus S$, donc $z \in \mathrm{pfv}(M[x,y,z]\langle y := N[z]\rangle_{S+x})$. En raison de la structure du terme initial, on a aussi que $x \notin \mathrm{pfv}(N[z])$. De plus, on note N' pour $N[z]\langle z := P\rangle_T$.

$$\begin{tikzcd}
(\lambda x.M[x,y,z])\langle y := N[z]\rangle_S\langle z := P\rangle_T \arrow[d, "(\mathrm{xab})^2"'] \arrow[dr, "(\mathrm{c})"] & \\
\lambda x.M[x,y,z]\langle y := N[z]\rangle_{S+x}\langle z := P\rangle_{T+x} \arrow[d, "(\mathrm{c})"'] & (\lambda x.M[x,y,z])\langle z := P\rangle_{T+y}\langle y := N'\rangle_S \arrow[ddl, "(\mathrm{xab})^2"] \\
\lambda x.M[x,y,z]\langle z := P\rangle_{T+x+y}\langle y := N[z]\langle z := P\rangle_{T+x}\rangle_{S+x} \arrow[d, "(\mathrm{i})"'] & \\
\lambda x.M[x,y,z]\langle z := P\rangle_{T+x+y}\langle y := N'\rangle_{S+x} &
\end{tikzcd}$$

Paire critique (xab)-(i)

$$\begin{tikzcd}
(\lambda x.M[y])\langle y := N\rangle_{S+z} \arrow[r, "(\mathrm{i})"] \arrow[d, "(\mathrm{xab})"'] & (\lambda x.M[y])\langle y := N\rangle_S \arrow[d, "(\mathrm{xab})"] \\
\lambda x.M[y]\langle y := N\rangle_{S+z+x} \arrow[r, "(\mathrm{i})"'] & \lambda x.M[y]\langle y := N\rangle_{S+x}
\end{tikzcd}$$

Paire critique (xap)-(xgc)

$$\begin{tikzcd}
(M\,N)\langle x := P\rangle_S \arrow[d, "(\mathrm{xap})"'] \arrow[dr, "(\mathrm{xgc})"] & \\
M\langle x := P\rangle_S\, N\langle x := P\rangle_S \arrow[r, "(\mathrm{xgc})^2"'] & M\,N
\end{tikzcd}$$

Paire critique (xap)-(c)

Dans ce qui suit, $y \in \mathrm{pfv}((M_1[x,y]\, M_2[x,y])\langle x := P[y]\rangle_S)$.

$$\begin{tikzcd}
(M_1[x,y]\, M_2[x,y])\langle x := P[y]\rangle_S\langle y := Q\rangle_T \arrow[d, "(\mathrm{c})"'] \arrow[dr, "(\mathrm{xap})"] & \\
(M_1[x,y]\, M_2[x,y])\langle y := Q\rangle_{T+x}\langle x := P'\rangle_S \arrow[d, "(\mathrm{xap})"'] & \begin{array}{c} M_1[x,y]\langle x := P[y]\rangle_S\langle y := Q\rangle_T \\ M_2[x,y]\langle x := P[y]\rangle_S\langle y := Q\rangle_T \end{array} \\
\begin{array}{c} M_1[x,y]\langle y := Q\rangle_{T+x}\langle x := P[y]\langle y := Q\rangle_T\rangle_S \\ M_2[x,y]\langle y := Q\rangle_{T+x}\langle x := P[y]\langle y := Q\rangle_T\rangle_S \end{array} &
\end{tikzcd}$$

On note $M_i[x,y]$ pour indifféremment $M_1[x,y]$ ou $M_2[x,y]$. On doit considérer deux cas : $\dashrightarrow$ n'est suivi que sous la condition $y \in (\mathrm{pfv}(P[y]) \cup \mathrm{pfv}(M_i[x,y]))$ et $\cdots\!\rightarrow$ sous la condition $y \notin (\mathrm{pfv}(P[y]) \cup \mathrm{pfv}(M_i[x,y]))$. Pour chaque membre des applications ci-dessus on montre ce qui suit :

$$\begin{tikzcd}
M_i[x,y]\langle x := P[y]\rangle_S\langle y := Q\rangle_T \arrow[r, dashed, "(\mathrm{c})"] \arrow[dr, dotted, "(\mathrm{xgc})"'] & M_i[x,y]\langle y := Q\rangle_{T+x}\langle x := P[y]\langle y := Q\rangle_T\rangle_S \arrow[d, dotted, "(\mathrm{xgc})^2"] \\
& M_i[x,y]\langle x := P[y]\rangle_S
\end{tikzcd}$$

Paire critique (xap)-(i)

$$\begin{tikzcd}
(M[x]\,N[x])\langle x := P\rangle_{S+y} \arrow[r, "(\mathrm{i})"] \arrow[d, "(\mathrm{xap})"'] & (M[x]\,N[x])\langle x := P\rangle_{S} \arrow[d, "(\mathrm{xap})"] \\
M[x]\langle x := P\rangle_{S+y}\,N[x]\langle x := P\rangle_{S+y} \arrow[r, "(\mathrm{i})^2"'] & M[x]\langle x := P\rangle_{S}\,N[x]\langle x := P\rangle_{S}
\end{tikzcd}$$

Paire critique (xv)-(c)

In the following $y \in \mathrm{pfv}(M[y])$.

$$\begin{tikzcd}
x\langle x := M[y]\rangle_S\langle y := N\rangle_T \arrow[r, "(\mathrm{c})"] \arrow[d, "(\mathrm{xv})"'] & x\langle y := N\rangle_{T+x}\langle x := M[y]\langle y := N\rangle_T\rangle_S \arrow[d, "(\mathrm{xgc})"] \\
M[y]\langle y := N\rangle_T & x\langle x := M[y]\langle y := N\rangle_T\rangle_S \arrow[l, "(\mathrm{xv})"]
\end{tikzcd}$$

Paire critique (xv)-(i)

$$\begin{tikzcd}
x\langle x := M\rangle_{S+y} \arrow[r, "(\mathrm{i})"] \arrow[dr, "(\mathrm{xv})"'] & x\langle x := M\rangle_S \arrow[d, "(\mathrm{xv})"] \\
 & M
\end{tikzcd}$$

Paire critique (xgc)-(c)

Dans ce qui suit, $y \in \mathrm{pfv}(M[x, y]\langle x := N[y]\rangle_S)$ et $x \notin \mathrm{pfv}(M[x, y])$. Étant donné la structure du terme de départ, on a aussi $x \notin \mathrm{pfv}(P)$.

$$\begin{tikzcd}
M[x,y]\langle x := N[y]\rangle_S\langle y := P\rangle_T \arrow[r, "(\mathrm{c})"] \arrow[d, "(\mathrm{xgc})"'] & M[x,y]\langle y := P\rangle_{T+x}\langle x := N[y]\langle y := P\rangle_T\rangle_S \arrow[d, "(\mathrm{xgc})"] \\
M[x,y]\langle y := P\rangle_T & M[x,y]\langle y := P\rangle_{T+x} \arrow[l, "(\mathrm{i})"]
\end{tikzcd}$$

Paire critique (xgc)-(i)

$$\begin{tikzcd}
M\langle x := N\rangle_{S+y} \arrow[r, "(\mathrm{i})"] \arrow[dr, "(\mathrm{xgc})"'] & M\langle x := N\rangle_S \arrow[d, "(\mathrm{xgc})"] \\
 & M
\end{tikzcd}$$

Paire critique (c)-(c)

Cette paire critique est la plus laborieuse à résoudre, car il y a beaucoup de cas à examiner. En effet, le terme à réduire est $M[x, y, z]\langle x := N[y, z]\rangle_S\langle y := P[z]\rangle_T\langle z := Q\rangle_U$ dans lequel $z \in \mathrm{pfv}(M[x, y, z]\langle x := N[y, z]\rangle_S\langle y := P[z]\rangle_T)$ et $y \in \mathrm{pfv}(M[x, y, z]\langle x := N[y, z]\rangle_S)$. D'après le corollaire A.1.5 on a $z \notin S$ (ce qui justifie la notation S au lieu de $S[z]$),mais on ne sait pas si

$z \in \mathrm{pfv}(M[x,y,z]), z \in \mathrm{pfv}(N[y,z])$, ou si $z \in \mathrm{pfv}(P[z])$ (la seule chose que l'on sait est que z est au moins dans un de ces ensembles). Malheureusement, il est nécessaire d'avoir de l'information à ce sujet pour décider dans beaucoup de cas si la règle qui s'applique est (xgc) ou (c). On considère 4 cas, qui sont (1) z appartient à l'ensemble de variables potentiellement libres d'au moins deux termes parmi M, N et P et (2, 3, 4) z n'appartient qu'à respectivement $\mathrm{pfv}(M)$, $\mathrm{pfv}(N)$ ou $\mathrm{pfv}(P)$. Dans tous les cas, il existe une réduction commune que nous montrons sur le diagramme suivant. Exceptionnellement on ne mentionne pas explicitement les variables liées dans le contexte au niveau des metavariables, pour des raisons de clarté.

$$
\begin{tikzcd}
M\langle x := N\rangle_S\langle y := P\rangle_T\langle z := Q\rangle_U \arrow[d, "(c)"'] \arrow[dr, "(c)"] & \\
M\langle y := P\rangle_{T+x}\langle x := N\langle y := P\rangle_T\rangle_S\langle z := Q\rangle_U \arrow[d, "(c)"'] & M\langle x := N\rangle_S\langle z := Q\rangle_{U+y}\langle y := P\langle z := Q\rangle_U\rangle_T \\
M\langle y := P\rangle_{T+x}\langle z := Q\rangle_{U+x}\langle x := N\langle y := P\rangle_T\langle z := Q\rangle_U\rangle_S &
\end{tikzcd}
$$

On note $A = M\langle y := P\rangle_{T+x}\langle z := Q\rangle_{U+x}\langle x := N\langle y := P\rangle_T\langle z := Q\rangle_U\rangle_S$ et $B = M\langle x := N\rangle_S\langle z := Q\rangle_{U+y}\langle y := P\langle z := Q\rangle_U\rangle_T$, et on examine les différents cas.

z est potentiellement libre dans au moins deux termes parmi M, N et P

$$
\begin{aligned}
A &\to M\langle z := Q\rangle_{U+x+y}\langle y := P\langle z := Q\rangle_{U+x}\rangle_{T+x}\langle x := N\langle y := P\rangle_T\langle z := Q\rangle_U\rangle_S && \text{(c)}\\
&\to M\langle z := Q\rangle_{U+x+y}\langle y := P\langle z := Q\rangle_{U+x}\rangle_{T+x}\\
&\qquad\qquad \langle x := N\langle z := Q\rangle_{U+y}\langle y := P\langle z := Q\rangle_U\rangle_T\rangle_S && \text{(c)}\\
&\to M\langle z := Q\rangle_{U+x+y}\langle y := P\langle z := Q\rangle_U\rangle_{T+x}\\
&\qquad\qquad \langle x := N\langle z := Q\rangle_{U+y}\langle y := P\langle z := Q\rangle_U\rangle_T\rangle_S && \text{(i)}
\end{aligned}
$$

$$
\begin{aligned}
B &\to M\langle z := Q\rangle_{U+x+y}\langle x := N\langle z := Q\rangle_{U+y}\rangle_S\langle y := P\langle z := Q\rangle_U\rangle_T && \text{(c)}\\
&\to M\langle z := Q\rangle_{U+x+y}\langle y := P\langle z := Q\rangle_U\rangle_{T+x}\\
&\qquad\qquad \langle x := N\langle z := Q\rangle_{U+y}\langle y := P\langle z := Q\rangle_U\rangle_T\rangle_S && \text{(c)}
\end{aligned}
$$

z est potentiellement libre dans M mais pas dans N ni P

$$
\begin{aligned}
A &\to M\langle z := Q\rangle_{U+x+y}\langle y := P\langle z := Q\rangle_{U+x}\rangle_{T+x}\langle x := N\langle y := P\rangle_T\langle z := Q\rangle_U\rangle_S && \text{(c)}\\
&\to M\langle z := Q\rangle_{U+x+y}\langle y := P\rangle_{T+x}\langle x := N\langle y := P\rangle_T\rangle_S && \text{(xgc)}
\end{aligned}
$$

$$
\begin{aligned}
B &\to M\langle z := Q\rangle_{U+x+y}\langle x := N\langle z := Q\rangle_{U+y}\rangle_S\langle y := P\langle z := Q\rangle_U\rangle_T & \text{(c)}\\
&\to M\langle z := Q\rangle_{U+x+y}\langle y := P\langle z := Q\rangle_U\rangle_{T+x}\\
&\qquad\qquad \langle x := N\langle z := Q\rangle_{U+y}\langle y := P\langle z := Q\rangle_U\rangle_T\rangle_S & \text{(c)}\\
&\to M\langle z := Q\rangle_{U+x+y}\langle y := P\rangle_{T+x}\langle x := N\langle y := P\rangle_T\rangle_S & \text{(xgc)}
\end{aligned}
$$

z est potentiellement libre dans N mais pas dans M ni P

$$
\begin{aligned}
A &\to M\langle y := P\rangle_{T+x}\langle z := Q\rangle_{U+x}\langle x := N\langle z := Q\rangle_{U+y}\langle y := P\langle z := Q\rangle_U\rangle_T\rangle_S & \text{(c)}\\
&\to M\langle y := P\rangle_{T+x}\langle x := N\langle z := Q\rangle_{U+y}\langle y := P\rangle_T\rangle_S & \text{(xgc)}
\end{aligned}
$$

$$
\begin{aligned}
B &\to M\langle z := Q\rangle_{U+x+y}\langle x := N\langle z := Q\rangle_{U+y}\rangle_S\langle y := P\langle z := Q\rangle_U\rangle_T & \text{(c)}\\
&\to M\langle z := Q\rangle_{U+x+y}\langle y := P\langle z := Q\rangle_U\rangle_{T+x}\\
&\qquad\qquad \langle x := N\langle z := Q\rangle_{U+y}\langle y := P\langle z := Q\rangle_U\rangle_T\rangle_S & \text{(c)}\\
&\to M\langle y := P\rangle_{T+x}\langle x := N\langle z := Q\rangle_{U+y}\langle y := P\rangle_T\rangle_S & \text{(xgc)}
\end{aligned}
$$

z est potentiellement libre dans P mais pas dans M ni N

$$
\begin{aligned}
A &\to M\langle z := Q\rangle_{U+x+y}\langle y := P\langle z := Q\rangle_{U+x}\rangle_{T+x}\langle x := N\langle y := P\rangle_T\langle z := Q\rangle_U\rangle_S & \text{(c)}\\
&\to M\langle z := Q\rangle_{U+x+y}\langle y := P\langle z := Q\rangle_{U+x}\rangle_{T+x}\\
&\qquad\qquad \langle x := N\langle z := Q\rangle_{U+y}\langle y := P\langle z := Q\rangle_U\rangle_T\rangle_S & \text{(c)}\\
&\to M\langle y := P\langle z := Q\rangle_{U+x}\rangle_{T+x}\langle x := N\langle y := P\langle z := Q\rangle_U\rangle_T\rangle_S & \text{(xgc)}\\
&\to M\langle y := P\langle z := Q\rangle_U\rangle_{T+x}\langle x := N\langle y := P\langle z := Q\rangle_U\rangle_T\rangle_S & \text{(i)}
\end{aligned}
$$

$$
\begin{aligned}
B &\to M\langle x := N\rangle_S\langle y := P\langle z := Q\rangle_U\rangle_T & \text{(xgc)}\\
&\to M\langle y := P\langle z := Q\rangle_U\rangle_{T+x}\langle x := N\langle y := P\langle z := Q\rangle_U\rangle_T\rangle_S & \text{(c)}
\end{aligned}
$$

Paire critique (c)-(i) (1)

Dans ce qui suit, $y \in \mathrm{pfv}(M[x,y]\langle x := N[y]\rangle_{S+z})$. De plus, d'après le corollaire A.1.6 $z \notin \mathrm{pfv}(P)$ (ce qui justifie la notation P au lieu de $P[z]$).

$$
\begin{tikzcd}
M[x,y]\langle x := N[y]\rangle_{S+z}\langle y := P\rangle_T \arrow[r, "\text{(c)}"] \arrow[d, "\text{(i)}"'] & M[x,y]\langle y := P\rangle_{T+x}\langle x := N[y]\langle y := P\rangle_T\rangle_{S+z} \arrow[d, "\text{(i)}"] \\
M[x,y]\langle x := N[y]\rangle_S\langle y := P\rangle_T \arrow[r, "\text{(c)}"'] & M[x,y]\langle y := P\rangle_{T+x}\langle x := N[y]\langle y := P\rangle_T\rangle_S
\end{tikzcd}
$$

Paire critique (c)-(i) **(2)**

Dans ce qui suit, $y \in \mathrm{pfv}(M\langle x := N\rangle_S)$.

$$\begin{tikzcd}
M[x,y]\langle x := N[y]\rangle_S\langle y := P\rangle_{T+z} \arrow[r, "(c)"] \arrow[d, "(i)"'] & M[x,y]\langle y := P\rangle_{T+z+x}\langle x := N[y]\langle y := P\rangle_{T+z}\rangle_S \arrow[d, "(i)^2"] \\
M[x,y]\langle x := N[y]\rangle_S\langle y := P\rangle_T \arrow[r, "(c)"'] & M[x,y]\langle y := P\rangle_{T+x}\langle x := N[y]\langle y := P\rangle_T\rangle_S
\end{tikzcd}$$

Corollaire A.1.8 $\xrightarrow[\mathrm{xci}]{}$ *est localement confluent.*

Démonstration. Il suffit de vérifier que les paires critiques qui n'impliquent pas (b) (toute sauf celle entre (b) et (xap)) n'utilisent pas (b) pour refermer leur diagramme. □

A.2 Preuve de confluence

Pour prouver la confluence sur les termes avec metavariables syntaxiques de $\xrightarrow[\mathrm{bxci}]{}$, on utilise la même technique que dans [68] (également présentée dans [69]) qui utilise le lemme de Yokouchi-Hikita (lemme 1.1.8) composé avec le lemme 1.1.9). Nous utiliserons $\xrightarrow[\mathrm{xci}]{}$ comme la relation fortement normalisante et localement confluente du lemme de Yokouchi-Hikita ($\xrightarrow[R]{}$), et $\xrightarrow[\mathrm{bxci}]{}$ est la relation dont nous voulons prouver la confluence ($\xrightarrow[T]{}$). Nous devons définir une relation fortement confluente $\xrightarrow[S]{}$ telle que $\xrightarrow[\mathrm{xci}]{}$ YH-commute sur $\xrightarrow[S]{}$, et $\xrightarrow[\mathrm{bxci}]{} \subseteq \xrightarrow[\mathrm{xci}]{}\!\!\!\!\!\!\rightarrow \cdot \xrightarrow[S]{} \cdot \xrightarrow[\mathrm{xci}]{}\!\!\!\!\!\!\rightarrow \subseteq \xrightarrow[\mathrm{bxci}]{}\!\!\!\!\!\!\rightarrow$. On définit la relation de b-contraction parallèle $\xrightarrow[\mathrm{b}\|]{}$, et on montre qu'elle satisfait les propriétés requises.

Définition A.2.1 *La relation de b-contraction parallèle, notée* $\xrightarrow[\mathrm{b}\|]{}$, *est définie par :*

$$\frac{M[x] \xrightarrow[\mathrm{b}\|]{} M'[x]}{\lambda x.M[x] \xrightarrow[\mathrm{b}\|]{} \lambda x.M'[x]} \qquad \frac{M \xrightarrow[\mathrm{b}\|]{} M' \quad N \xrightarrow[\mathrm{b}\|]{} N'}{(M\,N) \xrightarrow[\mathrm{b}\|]{} (M'\,N')} \qquad \frac{}{M \xrightarrow[\mathrm{b}\|]{} M}$$

$$\frac{M[x] \xrightarrow[\mathrm{b}\|]{} M'[x] \quad N \xrightarrow[\mathrm{b}\|]{} N'}{M[x]\langle x := N\rangle_S \xrightarrow[\mathrm{b}\|]{} M'[x]\langle x := N'\rangle_S} \qquad \frac{M[x] \xrightarrow[\mathrm{b}\|]{} M'[x] \qquad N \xrightarrow[\mathrm{b}\|]{} N'}{(\lambda x.M[x])\,N \xrightarrow[\mathrm{b}\|]{} M'[x]\langle x := N'\rangle_\varnothing}$$

Lemme A.2.2 $\xrightarrow[\mathrm{b}\|]{}$ *est fortement confluente.*

Démonstration. Soit M, N, P des λxc-termes tels que $M \xrightarrow[\mathrm{b}\|]{} N$ et $M \xrightarrow[\mathrm{b}\|]{} P$. On cherche un λxc-terme Q tel que $N \xrightarrow[\mathrm{b}\|]{} Q$ et $P \xrightarrow[\mathrm{b}\|]{} Q$. On procède par induction sur la structure de M :

- Si $M = A[x]\langle x := B\rangle_S$, alors N et P sont respectivement de la forme $A'[x]\langle x := B'\rangle_S$ et $A''[x]\langle x := B''\rangle_S$ où $A[x] \xrightarrow[\mathrm{b}\|]{} A'[x]$, $A[x] \xrightarrow[\mathrm{b}\|]{} A''[x]$, $B \xrightarrow[\mathrm{b}\|]{} B'$ et $B \xrightarrow[\mathrm{b}\|]{} B''$. D'après l'hypothèse d'induction, il existe $A^{(3)}[x]$ et $B^{(3)}$ tels que $A'[x] \xrightarrow[\mathrm{b}\|]{} A^{(3)}[x]$, $A''[x] \xrightarrow[\mathrm{b}\|]{} A^{(3)}[x]$, et même chose pour B' et B'' par rapport à $B^{(3)}$. Par conséquent, $Q = A^{(3)}[x]\langle x := B^{(3)}\rangle_S$ convient.
- Si $M = \lambda x.A[x]$, on fait le même raisonnement que ci-dessus.
- Si $M = x$, c'est trivial car x ne peut se réduire qu'en x, donc $N = P = Q = x$.
- Si $M = A\,B$, on doit considérer plusieurs cas, car P et Q peuvent être soit des applications, soit des substitutions (quand A est une abstraction). Cependant, cela n'introduit pas de difficulté technique, et la preuve a la même forme que ci-dessus.

□

Lemme A.2.3 $\xrightarrow[\mathrm{xci}]{}$ *YH-commute sur* $\xrightarrow[\mathrm{b}\|]{}$*, c'est-à-dire*

$$\begin{tikzcd}
a \arrow[rrr, "\mathrm{b}\|"'] \arrow[d, "\mathrm{xci}"'] & & & b \arrow[d, dashed, two heads, "\mathrm{xci}"] \\
c \arrow[r, dashed, two heads, "\mathrm{xci}"'] & \cdot \arrow[r, dashed, "\mathrm{b}\|"'] & \cdot \arrow[r, dashed, two heads, "\mathrm{xci}"'] & d
\end{tikzcd}$$

Démonstration. On vérifie les paires critiques entre $\xrightarrow[\mathrm{b}\|]{}$ et $\xrightarrow[\mathrm{xci}]{}$. Il y en a 8, et elles convergent toutes trivialement à travers ce diagramme. Une observation importante est que si $M \xrightarrow[\mathrm{b}\|]{} M'$ alors $\mathrm{pfv}(M) = \mathrm{pfv}(M')$. □

Le lemme suivant est trivial, et le théorème est une conséquence directe des résultats précédents.

Lemme A.2.4 $\xrightarrow[\mathrm{bxci}]{} \subseteq \underset{\mathrm{xci}}{\twoheadrightarrow} \cdot \xrightarrow[\mathrm{b}\|]{} \cdot \underset{\mathrm{xci}}{\twoheadrightarrow} \subseteq \underset{\mathrm{bxci}}{\twoheadrightarrow}$.

Théorème A.2.5 $\xrightarrow[\mathrm{bxci}]{}$ *est confluente sur les termes avec metavariables syntaxiques.*

Démonstration. Conséquence directe des résultats précédents, du lemme 1.1.8 et du lemme 1.1.9. □

Annexe B

Preuves de traductions

Cette annexe présente des preuves relatives à la traduction entre le calcul λxci et λx$_B$ci, dont les lemmes sont énoncés au chapitre 3.

B.1 Preuve de traduction du calcul de substitution

Cette section présente la preuve du lemme 3.2.16. On commence par prouver le lemme suivant :

Lemme B.1.1 $\forall \rho, \rho' : x_1 \ldots x_n \cdot \rho \vdash a_n \rightsquigarrow M \Rightarrow x_1 \ldots x_n \cdot \rho' \vdash a_n \rightsquigarrow M.$

Démonstration. Comme tout indice potentiellement libre de a_n est strictement plus petit que n, il est clair que seuls $x_1 \ldots x_n$ peuvent être libres dans M. Par conséquent, le reste du contexte est sans importance. □

On examine chacune des règles de λx$_B$ci.

Règle (b)

$$\dfrac{\dfrac{x \cdot \rho \vdash a \rightsquigarrow M}{\rho \vdash \lambda a \rightsquigarrow \lambda x.M} \quad \rho \vdash b \rightsquigarrow N}{\rho \vdash (\lambda a)b \rightsquigarrow (\lambda x.M)\, N} \qquad\qquad \dfrac{x \cdot \rho \vdash a \rightsquigarrow M \quad \rho \vdash b \rightsquigarrow N}{\rho \vdash a\langle \underline{0} := b\rangle \rightsquigarrow M\langle x := N\rangle_{\varnothing}}$$

et $(\lambda x.M)\, N \to M\langle x := N\rangle_{\varnothing}$ d'après (b).

Règle (xgc1)

$$\dfrac{\dfrac{x_1 \ldots x_{n+m} \vdash a \rightsquigarrow M}{x_1 \ldots x_n \cdot x \cdot x_{n+1} \ldots x_{n+m} \vdash a \uparrow_n \rightsquigarrow M} \quad x_{n+1} \ldots x_{n+m} \vdash b \rightsquigarrow N}{x_1 \ldots x_{n+m} \vdash a \uparrow_n \langle \underline{n} := b\rangle \rightsquigarrow M\langle x := N\rangle_{\{x_1 \ldots x_n\}}}$$

$$x_1 \ldots x_{n+m} \vdash a \rightsquigarrow M$$

et $M\langle x := N\rangle_{\{x_1 \ldots x_n\}} \to M$ d'après (xgc) car $x_1 \ldots x_{n+m} \vdash a \rightsquigarrow M$ prouve que $x \notin \mathrm{pfv}(M)$ (en effet, $\forall i : x_i \neq x$).

Règle (xgc2)

$$\frac{x_1 \dots x_{n+m} \cdot x \cdot x_{n+m+1} \dots x_{n+m+p} \vdash a_n \rightsquigarrow M \quad x_{n+m+1} \dots x_{n+m+p} \vdash b \rightsquigarrow N}{x_1 \dots x_{n+m+p} \vdash a_n \langle \underline{n+m} := b \rangle \rightsquigarrow M\langle x := N\rangle_{\{x_1 \dots x_{n+m}\}}}$$

$$\frac{x_1 \dots x_{n+m} \cdot x \cdot x_{n+m+1} \dots x_{n+m+p} \vdash a_n \rightsquigarrow M}{x_1 \dots x_{n+m+p} \vdash a_n \rightsquigarrow M} \text{ d'après le lemme B.1.1}$$

et $M\langle x := N\rangle_{\{x_1 \dots x_n\}} \to M$ d'après (xgc) car $x_1 \dots x_n \vdash a \rightsquigarrow M$ prouve que $x \notin \mathrm{pfv}(M)$.

Règle (xv)

$$\frac{x \cdot \rho \vdash \underline{0} \rightsquigarrow x \quad \rho \vdash a \rightsquigarrow M}{\rho \vdash \underline{0} \langle \underline{0} := a \rangle \rightsquigarrow x\langle x := M\rangle_\varnothing} \qquad \rho \vdash a \rightsquigarrow M$$

et $x\langle x := M\rangle_\varnothing \to M$ d'après (xv).

Règle (xab)

$$\frac{\dfrac{x \cdot x_1 \dots x_n \cdot y \cdot x_{n+1} \dots x_{n+m} \vdash a \rightsquigarrow M}{x_1 \dots x_n \cdot y \cdot x_{n+1} \dots x_{n+m} \vdash \lambda a \rightsquigarrow \lambda x.M} \quad x_{n+1} \dots x_{n+m} \vdash b \rightsquigarrow N}{x_1 \dots x_{n+m} \vdash (\lambda a)\langle \underline{n} := b\rangle \rightsquigarrow (\lambda x.M)\langle y := N\rangle_{\{x_1 \dots x_n\}}}$$

$$\frac{\dfrac{x \cdot x_1 \dots x_n \cdot y \cdot x_{n+1} \dots x_{n+m} \vdash a \rightsquigarrow M \quad x_{n+1} \dots x_{n+m} \vdash b \rightsquigarrow N}{x \cdot x_1 \dots x_{n+m} \vdash a\langle \underline{n+1} := b\rangle \rightsquigarrow M\langle y := N\rangle_{\{x; x_1 \dots x_n\}}}}{x_1 \dots x_{n+m} \vdash \lambda a\langle \underline{n+1} := b\rangle \rightsquigarrow \lambda x.M\langle y := N\rangle_{\{x; x_1 \dots x_n\}}}$$

et $(\lambda x.M)\langle y := N\rangle_{\{x_1 \dots x_n\}} \to \lambda x.M\langle y := N\rangle_{\{x; x_1 \dots x_n\}}$ d'après (xab).

Règle (xap)

$$\frac{\dfrac{\rho\backslash n \cdot x \cdot \rho/n \vdash a \rightsquigarrow M \quad \rho\backslash n \cdot x \cdot \rho/n \vdash b \rightsquigarrow N}{\rho\backslash n \cdot x \cdot \rho/n \vdash ab \rightsquigarrow MN} \quad \rho/n \vdash c \rightsquigarrow P}{\rho \vdash (ab)\langle \underline{n} := c\rangle \rightsquigarrow (MN)\langle x := P\rangle_{\rho\backslash n}}$$

$$\frac{\dfrac{\rho\backslash n \cdot x \cdot \rho/n \vdash a \rightsquigarrow M \quad \rho/n \vdash c \rightsquigarrow P}{\rho \vdash a\langle \underline{n} := c\rangle \rightsquigarrow M\langle x := P\rangle_{\rho\backslash n}} \quad \dfrac{\rho\backslash n \cdot x \cdot \rho/n \vdash b \rightsquigarrow N \quad \rho/n \vdash c \rightsquigarrow P}{\rho \vdash b\langle \underline{n} := c\rangle \rightsquigarrow N\langle x := P\rangle_{\rho\backslash n}}}{\rho \vdash a\langle \underline{n} := c\rangle\, b\langle \underline{n} := c\rangle \rightsquigarrow M\langle x := P\rangle_{\rho\backslash n}\, N\langle x := P\rangle_{\rho\backslash n}}$$

et $(MN)\langle x := P\rangle_{\rho\backslash n} \to M\langle x := P\rangle_{\rho\backslash n}\, N\langle x := P\rangle_{\rho\backslash n}$ d'après (xap).

Règle (c) On note q pour $n+p$, $\vec{x}_i^j$ pour $x_i \dots x_j$ (ϵ si $i > j$) et $\{x_i^j\}$ pour $\{x_i \dots x_j\}$ ($\varnothing$ si $i > j$).

$$\frac{\dfrac{\vec{x}_1^n \cdot x \cdot \vec{x}_{n+1}^q \cdot y \cdot \vec{x}_{q+1}^{q+m} \vdash a \rightsquigarrow M \quad \vec{x}_{n+1}^q \cdot y \cdot \vec{x}_{q+1}^{q+m} \vdash b \rightsquigarrow N}{\vec{x}_1^q \cdot y \cdot \vec{x}_{q+1}^{q+m} \vdash a\langle \underline{n} := b\rangle \rightsquigarrow M\langle x := N\rangle_{\{x_1^n\}}} \quad \vec{x}_{q+1}^{q+m} \vdash c \rightsquigarrow P}{\vec{x}_1^{q+m} \vdash a\langle \underline{n} := b\rangle\langle \underline{q} := c\rangle \rightsquigarrow M\langle x := N\rangle_{\{x_1^n\}}\langle y := P\rangle_{\{x_1^q\}}}$$

$$\frac{\dfrac{\vec{x}_1^n \cdot x \cdot \vec{x}_{n+1}^q \cdot y \cdot \vec{x}_{q+1}^{q+m} \vdash a \rightsquigarrow M \quad \vec{x}_{q+1}^{q+m} \vdash c \rightsquigarrow P}{\vec{x}_1^n \cdot x \cdot \vec{x}_{n+1}^{q+m} \vdash a\langle \underline{q+1} := c\rangle \rightsquigarrow M\langle y := P\rangle_{\{x_1^q; x\}}} \quad \dfrac{\vec{x}_{n+1}^q \cdot y \cdot \vec{x}_{q+1}^{q+m} \vdash b \rightsquigarrow N \quad \vec{x}_{q+1}^{q+m} \vdash c \rightsquigarrow P}{x_{n+1}^{q+m} \vdash b\langle \underline{p} := c\rangle \rightsquigarrow N\langle y := P\rangle_{\{x_{n+1}^q\}}}}{\vec{x}_1^{q+m} \vdash a\langle \underline{q+1} := c\rangle\langle \underline{n} := b\langle \underline{p} := c\rangle\rangle \rightsquigarrow M\langle y := P\rangle_{\{x_1^q; x\}}\langle x := N\langle y := P\rangle_{\{x_{n+1}^q\}}\rangle_{\{x_1^n\}}}$$

et

$$M\langle x := N\rangle_{\{x_1^n\}}\langle y := P\rangle_{\{x_1^q\}} \to M\langle y := P\rangle_{\{x_1^q;x\}}\langle x := N\langle y := P\rangle_{\{x_1^q\}}\rangle_{\{x_1^n\}}$$

d'après (c) car $\vec{x}_1^n\cdot x\cdot \vec{x}_{n+1}^q\cdot y\cdot \vec{x}_{q+1}^{q+m} \vdash a \rightsquigarrow M$ et $\vec{x}_{n+1}^q\cdot y\cdot\vec{x}_{q+1}^{q+m}\vdash b\rightsquigarrow N$ prouvent que $y \in \mathrm{pfv}(M\langle x := N\rangle_{\{x_1^n\}})$ si et seulement si $\underline{q} \in \mathrm{pfi}(a\langle \underline{n} := b\rangle)$. De plus, $N\langle y := P\rangle_{\{x_1^q\}} \twoheadrightarrow N\langle y:=P\rangle_{\{x_{n+1}^q\}}$ par n applications de (i) car $\vec{x}_{n+1}^q\cdot y\cdot \vec{x}_{q+1}^{q+m}\vdash b\rightsquigarrow N$ prouve que $x_1\ldots x_n \notin \mathrm{pfv}(N)$.

Règle (i) On note q pour $n+m$.

$$\dfrac{\dfrac{x_1\ldots x_n\cdot x_{n+2}\ldots x_{q+1}\cdot x\cdot x_{q+2}\ldots x_{q+p+1}\vdash a\rightsquigarrow M}{x_1\ldots x_{q+1}\cdot x\cdot x_{q+2}\ldots x_{q+p+1}\vdash a\uparrow_n\rightsquigarrow M}\quad x_{q+2}\ldots x_{q+p+1}\vdash b\rightsquigarrow N}{x_1\ldots x_{q+p+1}\vdash a\uparrow_n\langle\underline{q+1}:=b\rangle\rightsquigarrow M\langle x:=N\rangle_{\{x_1\ldots x_{q+1}\}}}$$

$$\dfrac{\dfrac{x_1\ldots x_n\cdot x_{n+2}\ldots x_{q+1}\cdot x\cdot x_{q+2}\ldots x_{q+p+1}\vdash a\rightsquigarrow M\quad x_{q+2}\ldots x_{q+p+1}\vdash b\rightsquigarrow N}{x_1\ldots x_n\cdot x_{n+2}\ldots x_{q+p+1}\vdash a\langle\underline{q}:=b\rangle\rightsquigarrow M\langle x:=N\rangle_{\{x_1\ldots x_n;x_{n+2}\ldots x_{q+1}\}}}}{x_1\ldots x_{q+p+1}\vdash a\langle\underline{q}:=b\rangle\uparrow_n\rightsquigarrow M\langle x:=N\rangle_{\{x_1\ldots x_n;x_{n+2}\ldots x_{q+1}\}}}$$

et $M\langle x := N\rangle_{\{x_1\ldots x_{q+1}\}} \to M\langle x := N\rangle_{\{x_1\ldots x_n;x_{n+2}\ldots x_{q+1}\}}$ car $x_1\ldots x_n\cdot x_{n+2}\ldots x_{q+1}\cdot x\cdot x_{q+2}\ldots x_{q+p+1}\vdash a\rightsquigarrow M$ prouve que $x_{n+1}\notin\mathrm{pfv}(M)$.

B.2 Preuve de traduction du calcul de renommage

Cette section présente la preuve du lemme 3.2.17

Règle (rc)

$$\dfrac{\dfrac{x_1\ldots x_n\cdot x_{n+2}\ldots x_{n+p+1}\cdot x_{n+p+m+1}\vdash a\rightsquigarrow M}{x_1\ldots x_{n+p+1}\cdot x_{n+p+3}\ldots x_{n+p+m+1}\vdash a\uparrow_n\rightsquigarrow M}}{x_1\ldots x_{n+p+m+1}\vdash a\uparrow_n\uparrow_{n+p+1}\rightsquigarrow M}$$

$$\dfrac{\dfrac{x_1\ldots x_n\cdot x_{n+2}\ldots x_{n+p+1}\cdot x_{n+p+m+1}\vdash a\rightsquigarrow M}{x_1\ldots x_n\cdot x_{n+2}\ldots x_{n+p+m+1}\vdash a\uparrow_{n+p}\rightsquigarrow M}}{x_1\ldots x_{n+p+m+1}\vdash a\uparrow_{n+p}\uparrow_n\rightsquigarrow M}$$

Règle (rgc) $\rho\vdash a_n\uparrow_n\rightsquigarrow M$ implique $\rho\vdash a_n\rightsquigarrow M$, d'après le lemme B.1.1.

Règle (rab)

$$\dfrac{\dfrac{x\cdot x_1\ldots x_n\cdot x_{n+2}\ldots x_{n+m}\vdash a\rightsquigarrow M}{x_1\ldots x_n\cdot x_{n+2}\ldots x_{n+m}\vdash\lambda a\rightsquigarrow\lambda x.M}}{x_1\ldots x_{n+m}\vdash(\lambda a)\uparrow_n\rightsquigarrow\lambda x.M}$$

$$\dfrac{\dfrac{x\cdot x_1\ldots x_n\cdot x_{n+2}\ldots x_{n+m}\vdash a\rightsquigarrow M}{x\cdot x_1\ldots x_{n+m}\vdash a\uparrow_{n+1}\rightsquigarrow M}}{x_1\ldots x_{n+m}\vdash\lambda a\uparrow_{n+1}\rightsquigarrow\lambda x.M}$$

Règle (rap)

$$\dfrac{\dfrac{x_1 \ldots x_n \cdot x_{n+2} \ldots x_{n+m} \vdash a \leadsto M \quad x_1 \ldots x_n \cdot x_{n+2} \ldots x_{n+m} \vdash b \leadsto N}{x_1 \ldots x_n \cdot x_{n+2} \ldots x_{n+m} \vdash (ab) \leadsto MN}}{x_1 \ldots x_{n+m} \vdash (ab) \uparrow_n \leadsto MN}$$

$$\dfrac{\dfrac{x_1 \ldots x_n \cdot x_{n+2} \ldots x_{n+m} \vdash a \leadsto M}{x_1 \ldots x_{n+m} \vdash a \uparrow_n \leadsto M} \quad \dfrac{x_1 \ldots x_n \cdot x_{n+2} \ldots x_{n+m} \vdash b \leadsto N}{x_1 \ldots x_{n+m} \vdash b \uparrow_n \leadsto N}}{x_1 \ldots x_{n+m} \vdash a \uparrow_n \ b \uparrow_n \leadsto MN}$$

Règle (rx)

$$\dfrac{\dfrac{\vec{x}_1^{n} \cdot x \cdot \vec{x}_{n+1}^{n+p} \cdot \vec{x}_{n+p+2}^{n+p+m} \vdash a \leadsto M \quad \vec{x}_{n+1}^{n+p} \cdot \vec{x}_{n+p+2}^{n+p+m} \vdash b \leadsto N}{\vec{x}_1^{n+p} \cdot \vec{x}_{n+p+2}^{n+p+m} \vdash a\langle \underline{n} := b\rangle \leadsto M\langle x := N\rangle_{\{x_1^n\}}}}{\vec{x}_1^{n+p+m} \vdash a\langle \underline{n} := b\rangle \uparrow_{n+p} \leadsto M\langle x := N\rangle_{\{x_1^n\}}}$$

$$\dfrac{\dfrac{\vec{x}_1^{n} \cdot x \cdot \vec{x}_{n+1}^{n+p} \cdot \vec{x}_{n+p+2}^{n+p+m} \vdash a \leadsto M}{\vec{x}_1^{n} \cdot x \cdot \vec{x}_{n+1}^{n+p+m} \vdash a \uparrow_{n+p+1} \leadsto M} \quad \dfrac{\vec{x}_{n+1}^{n+p} \cdot \vec{x}_{n+p+2}^{n+p+m} \vdash b \leadsto N}{\vec{x}_{n+1}^{n+p+m} \vdash b \uparrow_p \leadsto N}}{\vec{x}_1^{n+p+m} \vdash a \uparrow_{n+p+1} \langle \underline{n} := b \uparrow_p\rangle \leadsto M\langle x := N\rangle_{\{x_1^n\}}}$$

Annexe C

Définition de λ-termes

Nous donnons dans cette annexe la définition des booléens et des numéraux de Barendregt, ainsi que de plusieurs fonctions qui peuvent leur être appliquées.

Booléens

Les booléens sont définis par les λ-termes suivants :

$$\begin{aligned} \mathsf{true} &\equiv \lambda xy.x \\ \mathsf{false} &\equiv \lambda xy.y \end{aligned}$$

La conditionnelle if M then N else P se traduit donc simplement par le terme MNP.

Numéraux

Les numéraux de Barendregt sont définis par la grammaire suivante :

$$\hat{n} \ ::= \hat{0} \ \mid \ (\mathsf{succ}\ \hat{n})$$

où $\hat{0} \equiv \lambda x.x$ et $\mathsf{succ} \equiv \lambda xy.y\ \mathsf{false}\ x$.
La fonction de test à zéro est définie par $\mathsf{zero} \equiv \lambda x.x\ \mathsf{true}$.
La fonction prédécesseur pred s'exprime simplement par le terme $\lambda x.x\ \mathsf{false}$.

Let et let récursif

let $x = M$ in N est une abbréviation pour $(\lambda x.N)\, M$.
letrec $x = M$ in N est une abbréviation pour let $x = \mathsf{rec}\ (\lambda x.M)$ in N où rec est le combinateur de point fixe défini par le terme $\lambda f.(\lambda x.f\ (x\ x))\,(\lambda x.f\ (x\ x))$.

Fonctions récursives

On définit enfin les fonctions récursives qui dénotent respectivement l'égalité des entiers, la relation $>$, la relation $\geq$, et la relation de diviseur. suivantes :

$$
\begin{aligned}
\mathsf{moins} &\equiv \mathsf{rec}\ (\lambda\mathsf{moins}.\lambda xy.\mathrm{if}\ (\mathsf{zero}\ y)\ \mathrm{then}\ x\ \mathrm{else}\ (\mathsf{moins}\ (\mathsf{pred}\ x)\ (\mathsf{pred}\ y)))\\
\mathsf{equals} &\equiv \mathsf{rec}\ (\lambda\mathsf{equals}.\lambda xy.\mathrm{if}\ (\mathsf{zero}\ x)\ \mathrm{then}\ (\mathrm{if}\ (\mathsf{zero}\ y)\ \mathrm{then}\ \mathsf{true}\ \mathrm{else}\ \mathsf{false})\\
&\qquad \mathrm{else}\ (\mathsf{equals}\ (\mathsf{pred}\ x)\ (\mathsf{pred}\ y)))\\
\mathsf{sup} &\equiv \mathsf{rec}\ (\lambda\mathsf{sup}.\lambda xy.\mathrm{if}\ (\mathsf{zero}\ y)\ \mathrm{then}\ (\mathrm{if}\ (\mathsf{zero}\ x)\ \mathrm{then}\ \mathsf{false}\ \mathrm{else}\ \mathsf{true})\\
&\qquad \mathrm{else}\ (\mathrm{if}\ (\mathsf{zero}\ x)\ \mathrm{then}\ \mathsf{false}\ \mathrm{else}\ (\mathsf{sup}\ (\mathsf{pred}\ x)\ (\mathsf{pred}\ y))))\\
\mathsf{supeq} &\equiv \mathsf{rec}\ (\lambda\mathsf{supeq}.\lambda xy.\mathrm{if}\ (\mathsf{zero}\ y)\ \mathrm{then}\ \mathsf{true}\ \mathrm{else}\ (\mathrm{if}\ (\mathsf{zero}\ x)\ \mathrm{then}\ \mathsf{false}\\
&\qquad \mathrm{else}\ (\mathsf{supeq}\ (\mathsf{pred}\ x)\ (\mathsf{pred}\ y))))\\
\mathsf{div} &\equiv \mathsf{rec}\ (\lambda\mathsf{div}.\lambda xy.\mathrm{if}\ (\mathsf{zero}\ y)\ \mathrm{then}\ \mathsf{true}\ \mathrm{else}\ (\mathrm{if}\ (\mathsf{sup}\ x\ y)\ \mathrm{then}\ \mathsf{false}\ \mathrm{else}\ (\mathsf{div}\ x\ (\mathsf{moins}\ y\ x))))
\end{aligned}
$$

Bibliographie

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1(4) :375–416, 1991.

[2] Z. M. Ariola, M. Felleisen, J. Maraist, M. Odersky, and P. Wadler. A call-by-need lambda calculus. In *22nd Conference on Principles of Programming Languages*, pages 233–246, San Francisco, California, Janvier 1995.

[3] A. Asperti. A categorical understanding of environment machines. *Journal of Functional Programming*, 2(1), 1992.

[4] A. Asperti. $\delta \circ ! \epsilon = 1$. In J. Hsiang, editor, *Proceedings of the 6th Rewriting Techniques and Applications Conference (RTA'95)*, Lecture Notes in Computer Science, pages 102–116. Springer Verlag, 1995.

[5] A. Asperti and C. Laneve. Paths, computations and labels in the λ-calculus. In C. Kirchner, editor, *5th International Conference on Rewriting Techniques and Applications, Montreal (Canada)*, volume 690 of *Lecture Notes in Computer Science*, pages 152–167. Springer Verlag, 1993.

[6] A. Asperti and C. Laneve. Interaction systems I : The theory of optimal reduction. *Mathematical Structures in Computer Science*, 4(4), 1994.

[7] A. Asperti and C. Laneve. Paths, computations and labels in the λ-calculus. *Theoretical Computer Science*, 142(2), 1995.

[8] A. Asperti and C. Laneve. Interaction systems II : The practice of optimal reductions. *Theoretical Computer Science*, 159(2), 1996.

[9] F. Baader and T. Nipkow. *Term Rewriting and* All That. Cambridge University Press, 1998.

[10] H. P. Barendregt. *The Lambda Calculus : Its Syntax and Semantics.* North-Holland, revised edition, 1984.

[11] H. P. Barendregt, J. R. Kennaway, J. W. Klop, and M. R. Sleep. Needed reductions and spine strategies for the lambda calculus. *Information and Computation*, 75 :191–231, 1987.

[12] Z. E.-A. Benaissa. *Les calculs de substitution explicite comme fondement des implantations des langages fonctionnels.* PhD thesis, Université Henri Poincaré, Nancy 1, 1997.

[13] Z. E.-A. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *Journal of Functional Programming*, 6(5) :699–722, september 1996.

[14] Z. E.-A. Benaissa, F. Lang, P. Lescanne, and K. H. Rose. Modeling sharing and recursion for weak reduction strategies using explicit substitution. *Journal of Functional and Logic Programming*, 1998. à paraître.

[15] Z. E.-A. Benaissa, K. H. Rose, and P. Lescanne. Modeling sharing and recursion for weak reduction strategies using explicit substitution. In H. Kuchen and D. Swierstra, editors, *8th PLILP—Symposium on Programming Language Implementation and Logic Programming*, pages 393–407, Aachen, Germany, September 1996.

[16] R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In *CSN '95 – Computer Science in the Netherlands*, pages 62–72, November 1995.

[17] R. Bloo and K. H. Rose. Combinatory reduction systems with explicit substitution that preserve strong normalisation. In H. Ganzinger, editor, *RTA '96—Rewriting Techniques and Applications*, number 1103 in Lecture Notes in Computer Science, pages 169–183, New Brunswick, New Jersey, July 1996. Rutgers University, Springer-Verlag.

[18] L. Cardelli. Compiling a functional language. In *LFP*, 1984.

[19] A. Church. *The Calculi of Lambda-Conversion.* Princeton University Press, Princeton, N. J., 1941.

[20] C. Cornes, J. Courant, J.-C. Filiâtre, G. Huet, P. Manoury, C. Muñoz, C. Murthy, C. Parent, C. Paulin-Mohring, A. Saïbi, and B. Werner. *The Coq Proof Assistant Reference Manual Version 5.10.* Rapport technique n0177, INRIA, Juillet 1995.

[21] G. Cousineau, P.-L. Curien, and M. Mauny. The categorical abstract machine. In *FPCA*, 1985.

[22] P. Crégut. An abstract machine for the normalization of λ-calculus. In *Proc. Conf. on Lisp and Functional Programming*, pages 333–340. ACM, 1990.

[23] P. Crégut. *Machines à environnement pour la réduction symbolique et l'évaluation partielle.* PhD thesis, Université de Paris 7, 1991.

[24] P.-L. Curien. An abstract framework for environment machines. *Theoretical Computer Science*, 82 :389–402, 1991.

[25] H. B. Curry. Grundlagen der kombinatorischen logik. *American Journal of Mathematics*, 52 :509–536, 789–834, 1930.

[26] H. B. Curry and Feys. *Combinatory Logic*, volume 1. Elsevier Science, Amsterdam, 1958.

[27] R. David. Normalization without reducibility. Personal communication, 1998. Article soumis au journal Annals of Pure and Applied Logic.

[28] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formule manipulation with application to the church rosser theorem. *Koninklijke Nederlandse Akademie van Wetenschappen, Series A, Mathematical Sciences*, 75 :381–392, 1972.

[29] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1) :69–116, 1987. Corrigendum : 4(3) : 409–410.

[30] J. Fairbairn and S. C. Wray. TIM : A simple, lazy abstract machine to execute supercombinators. In G. Kahn, editor, *Functional Programming and Computer Architecture*, number 274 in Lecture Notes in Computer Science, pages 34–45, Portland, Oregon, 1987.

[31] M. Felleisen and D. P. Friedman. A syntactic theory of sequential state. *Theoretical Computer Science*, 69 :243–287, 1989.

[32] M. C. F. Ferreira, D. Kesner, and L. Puel. λ-calculi with explicit substitution and composition which preserve β-strong normalization. In M. Hanus and M. Rodríguez-Artalejo, editors, *ALP '96—Fifth International Conference on Algebraic and Logic Programming*, number 1139

in Lecture Notes in Computer Science, pages 284–298, Aachen, Germany, September 1996. Springer-Verlag.

[33] J. Field. On laziness and optimality in lambda interpreters : Tools for specification and analysis. In *Proceedings of the 17th Annual ACM Symposium on Principles Of Programming Languages, Orlando (Fla., USA)*, pages 1–15, San Fransisco, 1990. ACM.

[34] G. Gentzen. Investigations into logical deduction (1931). In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland Publishing Co., 1969.

[35] J.-Y. Girard. Geometry of interaction II : Deadlock-free algorithms.

[36] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 1(50) :1–102, 1987.

[37] J.-Y. Girard. Geometry of interaction I : Interpretation of system F. In Ferro, Bonotto, Valentini, and Zanardo, editors, *Logic Colloquiom '88*, pages 221–260. Elsevier Science Publisher B.V. (North Holland), 1989.

[38] G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Proceedings of the 19th Annual ACM Symposium on Principles Of Programming Languages, Albuquerque*, pages 15–26. ACM, 1992.

[39] B. Guillaume. The λse-calculus does not preserve strong normalisation. *To appear in Journal of Functionnal Programming*, 1998. Personal communication.

[40] T. Hardin, L. Maranget, and B. Pagano. Functional back-ends within the lambda-sigma calculus. In *Proceedings of International Conference on Functional Programming*, pages 25–33, 1996.

[41] F. Kamareddine and A. Ríos. Extending a λ-calculus with explicit substitution which preserves strong normalisation into a confluent calculus on open terms. *Journal of Functional Programming*, 7(4), 1997.

[42] V. K. Kathail. *Optimal Interpreters for Lambda-calculus Based Functional Programming Languages.* PhD thesis, Massachusssets Institute of Technology, 1990.

[43] J. W. Klop. *Combinatory Reduction Systems.* Mathematical Centre Tracts 127. Mathematisch Centrum, Amsterdam, 1980.

[44] J. W. Klop. Term rewriting systems. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, chapter 1, pages 1–116. Oxford University Press, 1992.

[45] J. W. Klop, V. van Oostrom, and F. van Raamsdonk. Combinatory reduction systems : Introduction and survey. *Theoretical Computer Science*, 121 :279–308, 1993.

[46] J.-L. Krivine. Un interpréteur pour le λ-calcul. Notes de cours de DEA, Université de Paris 7, 1985.

[47] Y. Lafont. Interaction nets. In *Proceedings of the 17th Annual ACM Symposium on Principles Of Programming Languages, Orlando (Fla., USA)*, pages 95–108. ACM, 1990.

[48] J. Lamping. An algorithm for optimal lambda-calculus reduction. In *Proceedings of the 17th Annual ACM Symposium on Principles Of Programming Languages, Orlando (Fla., USA)*, pages 16–30. ACM, 1990.

[49] P. J. Landin. The mechanical evaluation of expressions. *Computer Journal*, 6(4), 1964.

[50] C. Laneve. *Optimality and concurrency in interaction systems.* PhD thesis, Università di Pisa, 1993.

[51] F. Lang. Une machine à environnement avec partage, la up-machine. Master's thesis, Université Henri Poincaré, Nancy 1, 1994.

[52] F. Lang, Z. E.-A. Benaissa, and P. Lescanne. Super-closures. In S. Diehl and P. Sestoft, editors, *Proceedings of the first Workshop on Principles of Abstract Machines*, Pisa (Italy), 1998. Université de Saarbrück. Rapport technique numéro A 02/98.

[53] J. Launchbury. A natural semantics for lazy evaluation. In *20th POPL*, pages 144–154, 1993.

[54] J. Launchbury, A. Gill, J.. Hughes, S. Marlow, S. Peyton Jones, and P. Wadler. Avoiding unnecessary updates. In *Proceedings of the Glasgow Workshop on Functional Programming*, Springer-Verlag Workshops in Computing, 1993.

[55] X. Leroy. *Typage polymorphe d'un langage algorithmique.* PhD thesis, Université de Paris 7, 1992.

[56] X. Leroy. The objective caml system release 1.01, documentation and user's manual. Technical report, INRIA, 1996.

[57] P. Lescanne. From $\lambda\sigma$ to $\lambda\upsilon$, a journey through calculi of explicit substitutions. In *Proceedings of the 21st conference on Principles of Programming Languages*, pages 60–69, 1994.

[58] P. Lescanne. Le jeu des permutations ne termine pas. Communication personnelle, 1996.

[59] J.-J. Lévy. *Réductions correctes et optimales dans le lambda-calcul.* PhD thesis, Université de Paris VII, 1978. Thèse d'État.

[60] J.-J. Lévy. Optimal reductions in the lambda-calculus. In J. P. Seldin and J. R. Hindley, editors, *To H.B. Curry : Essays on Combinators, Logic, Lambda Calculus and Formalism*, pages 159–191. Academic Press, 1980.

[61] I. Mackie. *The geometry of implementation (Applications of the geometry of interaction to language implementation).* PhD thesis, University of London, Imperial College of Science, Technology and Medicine, Department of Computing, 1994.

[62] L. Maranget. Optimal derivations in orthogonal rewriting systems and in weak lambda calculi. In *Priciples Of Programming Languages*, 1991.

[63] P.-A. Melliès. Typed λ-calculi with explicit substitution may not terminate. In M. Dezani, editor, *TLCA '95—Int. Conf. on Typed Lambda Calculus and Applications*, volume 902 of *Lecture Notes in Computer Science*, pages 328–334, Edinburgh, Scotland, April 1995. Springer-Verlag.

[64] R. Milner, M. Tofte, and R. Harper. *The definition of standard ML.* MIT Press, 1990.

[65] C. Muñoz. Confluence and preservation of strong normalisation in an explicit substitutions calculus. In *LICS '96—Eleventh Annual IEEE Symposium on Logic in Computer Science*, pages 440–447, New Brunswick, New Jersey, July 1996. Rutgers University, IEEE Computer Society Press.

[66] C. Muñoz. *Un calcul de substitutions pour la représentation de preuves partielles en théorie des types.* PhD thesis, Université Denis Diderot, Paris 7, 1997.

[67] R. P. Nederpelt, J. H. Geuvers, and R. C. de Vrijer, editors. *Selected Papers on Automath*, volume 133 of *Studies in Logic.* North Holland, 1994.

[68] Curien P.-L., T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *Journal of the ACM*, 43(2) :362–397, March 1996.

[69] B. Pagano. *Des calculs de substitution explicite et de leur application à la compilation des langages fonctionnels.* PhD thesis, Université Pierre et Marie Curie (Paris), 1998.

[70] J. Peterson, K. Hammond, L. Augustsson, B. Boutel, W. Burton, J. Fasel, A. Gordon, J. Hughes, P. Hudak, T. Johnsson, M. Jones, E. Meijer, S. Peyton Jones, A. Reid, and P. Wadler. *Haskell 1.4, A Non Strict Purely Functional Language*, April 1997. available at `http ://www.haskell.org`.

[71] S. L. Peyton Jones. Implementing lazy functional programming languages on stock hardware : the spineless tagless G-machine. *Journal of Functional Programming*, 2(2) :127–202, 1992.

[72] G. D. Plotkin. Call-by-name, call-by-value, and the λ-calculus. *Theoretical Computer Science*, 1 :125–159, 1975.

[73] A. Ríos. *Contributions à l'étude de λ-calculs avec des substitutions explicites*. PhD thesis, Université de Paris 7, 1993.

[74] K. H. Rose. Explicit substitution – tutorial & survey. Lecture Series LS–96–3, BRICS, Dept. of Computer Science, University of Aarhus, Ny Munkegade (bld.540), 8000 Aarhus C, Denmark, September 1996.

[75] K. H. Rose. *Operational Reduction Models for Functional Programming Languages*. PhD thesis, DIKU, Universitetsparken 1, DK-2100 København Ø, February 1996. DIKU report 96/1.

[76] M. Schönfinkel. Über die Bausteine der mathematischen Logik. *Math. Ann.*, 92 :305–316, 1924.

[77] P. Sestoft. *Analysis and Efficient Implementation of Functional Programs*. PhD thesis, University of Copenhagen, 1991. DIKU Research Report 92/6.

[78] P. Sestoft. Deriving a lazy abstract machine. *Journal of Functional Programming*, 7(3) :231–264, 1997.

[79] W. Snyder and J. Gallier. Higher order unification revisited : Complete sets of tranformations. *Journal of Symbolic Computation*, 8(1 & 2) :101–140, 1989.

[80] A. Turing. On computable numbers, with an application to the entscheidungsproblem. *Proc. London Math. Soc.*, 42(2) :230–265, 1936.

[81] A. Turing. Computability and λ-definability. *Journal of Symbolic Logic*, 2 :153–163, 1937.

[82] H. Yokouchi and T. Hikita. A rewriting system for categorical combinators with multiple arguments. In *RIMS Lecture Notes*, volume 655, pages 186–208, Kyoto Univ., Japan, 1988.

Index

Résumé : La thèse aborde le problème de la définition de modèles adaptés aux implantations de l'opération de β-réduction en λ-calcul. Nous nous intéressons particulièrement à deux types de modèles : les calculs de substitution explicite, et les réseaux d'interaction.

Les calculs de substitution explicite sont étudiés sous différents points de vue, d'une part dans le cadre de la démonstration automatique, où il s'avère intéressant de disposer d'un calcul confluent et préservant la forte normalisation, qui permet de représenter des preuves incomplètes. Nous discutons ces propriétés et apportons d'importantes contributions à un problème ouvert dans ce domaine. D'autre part, les calculs de substitution explicite sont appliqués à la programmation fonctionnelle, et à la réduction faible qui en est le concept fondamental. Il est alors important de disposer d'outils pour exprimer le partage mis en œuvre par les implantations, notamment pour les langages paresseux. Dans ce cadre, nous donnons une preuve de correction très précise de KP, une machine abstraite de réduction paresseuse du λ-calcul, en nous appuyant sur un calcul avec substitution explicite et adresses globales, $\lambda\sigma_w^a$. Nous proposons ensuite une optimisation pour cette machine.

Les réseaux d'interaction sont habituellement utilisés pour la définition de réducteurs optimaux, au sens de Lévy. C'est le cas de l'algorithme de Lamping. Cependant, cet algorithme s'avère complexe. Nous étudions dans cette thèse un réducteur basé sur quelques idées proposées par Lamping, mais en relachant quelque peu les contraintes qui permettaient d'obtenir l'optimalité. Nous obtenons ainsi un algorithme non optimal que nous comparons avec l'algorithme de Lamping (implantation d'Asperti) et avec KP. Les résultats sont encourageants et montrent que notre algorithme réalise un assez bon partage sans avoir la lourdeur de la gestion de ce partage que l'on trouve dans les interprètes optimaux.

Abstract : The thesis addresses the issue of defining models to implement the β-reduction in the λ-calculus. We focus especially on two kinds of models : explicit substitution calculi, and interaction nets.

Explicit substitution calculi are studied in several contexts. On the one hand, they are studied in the context of automatic provers, where it is interresting to have a calculus confluent and preserving strong normalization, that allows the representation of incomplete proofs. We discuss these properties, and give important contributions to an open problem in this field. On the other hand, explicit substitution calculi are applied to functional programming and weak reduction which is the fundamental concept. It is then essential to have tools which can express the sharing provided by implementations, particularly for lazy languages. To illustrate this, we give a very precise and formal proof of the correctness of KP, an abstract machine for lazy reduction of the λ-calculus, using a calculus of explicit substitution and global adresses, namely $\lambda\sigma_w^a$. We then propose an optimization to this machine.

Interaction nets are usually associated to optimal reduction à la Lévy, as described by Lamping's algorithm. However, this algorithm is known as rather complex. We study in our thesis a reducer based on some ideas due to Lamping, but leaving out some constraints introduced for optimality. Thus, we obtain a non optimal algorithm that we compare to Lamping's algorithm (implementation of Asperti) and KP. The results show as expected that our algorithm provides quite a good amount of sharing, without the bureaucracy that one finds in optimal interpreters.