Subsitutions explicites, logique et normalisation

Emmanuel Polonovski

Université Paris 7 – Denis Diderot — UFR d'informatique
Année 2004

Substitutions explicites, logique et normalisation

Thèse

pour l'obtention du diplôme de

Docteur de l'université Paris 7, spécialité informatique

présentée et soutenue publiquement par

Emmanuel Polonovski

le 30 juin 2004

Directeur de thèse

Pierre-Louis Curien

Jury

Pierre-Louis	Curien	Directeur de thèse
René	David	Rapporteur
Thérèse	Hardin	Rapporteur
Fairouz	Kamareddine	
Delia	Kesner	
Simona	Ronchi Della Rocca	

Remerciements

Je souhaite remercier en premier lieu Pierre-Louis Curien qui a accepté d'être mon directeur de thèse. Malgré ses nombreuses responsabilités, il a toujours pris le temps de m'encourager et de m'aider dans mes travaux. Les nombreuses discussions avec lui m'ont permis de mieux comprendre le travail du chercheur et le milieu de la recherche. Sa rigueur et sa conscience professionnelle resteront un exemple pour moi.

Je remercie chaleureusement Thérèse Hardin et René David qui ont accepté le lourd travail de rapporteurs, leurs commentaires ont bien contribué à l'enrichissement de ce manuscrit. J'en profite aussi pour remercier Thérèse de m'avoir incité à faire le DEA qui m'a amené jusqu'ici.

Fairouz Kamareddine et Simona Ronchi Della Rocca ont droit à toute ma gratitude pour avoir bien voulu faire partie de mon jury de thèse, et donc se déplacer respectivement depuis l'Écosse et l'Italie.

Il m'est impossible d'énumérer toutes les causes qui m'amènent à remercier ici Delia Kesner. Avec Roberto Di Cosmo, ils m'ont initié à la recherche durant mon stage de DEA et m'ont poussé à entreprendre cette thèse ; ils m'ont toujours encouragé et soutenu depuis. Les discussions que j'ai eues avec Delia m'ont enrichi énormément, tant pour mes travaux que pour ma compréhension des différents aspects du métier d'enseignant-chercheur.

Hugo Herbelin fut, au début de ma thèse, une source d'inspiration et d'éclaricissements pour plusieurs de mes travaux, je l'en remercie vivement.

Je voudrais aussi remercier Guy Cousineau qui m'a donné, ces deux dernières années, la possibilité de pouvoir me consacrer à l'enseignement autant que je le souhaitais. Les échanges que j'ai eus avec lui m'auront aussi beaucoup apporté.

Durant ma thèse, j'ai eu le plaisir de travailler dans des cadres agréables, aussi bien pour l'enseignement que pour la recherche. Je remercie ici les membres de PPS, du LIAFA et l'UFR d'informatique de Paris XI qui y ont contribué. L'équipe des thésards (et post-thésards) de PPS mérite ma reconnaissance : ce fut une joie de travailler auprès de Vincent Balat, Sylvain Baro, Emmanuel Beffara, Anne-Gwenn Bosser, Julien Forest, Michel Hirschowitz, Stéphane Lengrand, Benjamin Leperchey, Jean-Vincent Loddo, François Maurel et Raphaël Montelatici ; je voudrais faire une mention particulière des discussions que j'ai eues avec Stéphane et qui m'ont permis d'avancer dans mon travail.

Je remercie ma famille et mes amis qui m'ont aidé et encouragé. Mes plus profonds remerciements vont sans hésiter à Anne. Elle m'a soutenu tout au long de ces années de thèse, et elle m'a supporté pendant la période douloureuse où j'ai rédigé mon manuscrit. De plus, elle a accompli un travail considérable de relecture et de correction sur l'ensemble de la première partie, lui donnant son aspect final. Enfin, l'affection dont Anne, Amélie et Nicolas m'ont entouré, m'a aidé à tenir bon dans les moments difficiles.

Résumé

Les substitutions explicites ont été introduites comme un raffinement du λ-calcul, celui-ci étant le formalisme utilisé pour étudier la sémantique des langages de programmation. Plusieurs propriétés de ces λ-calculs avec substitutions explicites sont recherchées par les sémanticiens ; parmi celles-ci, on s'intéresse plus particulièrement, dans cette thèse, à la normalisation forte et à la préservation de la normalisation forte. Ce manuscrit rend compte de plusieurs travaux autour de ces propriétés de normalisation, regroupés en trois volets.

Le premier d'entre eux formalise une technique générale de preuve de normalisation forte utilisant la préservation de la normalisation forte. Cette technique est basée sur une remontée des substitutions et utilise parfois une simulation à l'intérieur du calcul étudié. On applique cette technique à un spectre assez large de calculs avec substitutions explicites afin de mesurer les limites de son utilisation, ainsi que les facteurs augmentant la difficulté de son application. Grâce à cette technique, on prouve un résultat nouveau : la normalisation forte du $\lambda\upsilon$-calcul simplement typé.

Le deuxième travail est l'étude d'un calcul symétrique non-déterministe issu de la logique classique formulée dans le calcul des séquents. À cause de la symétrie de ce calcul, l'utilisation de preuves par réductibilité pour établir la normalisation forte d'une version avec substitutions explicites semble vouée à l'échec. On utilise alors la technique formalisée dans le premier travail, ce qui nous demande de prouver tout d'abord la préservation de la normalisation forte. À cette fin, on utilise un fragment de la théorie de la perpétuité dans les systèmes de réécriture.

La définition d'une nouvelle version du λ_{ws}-calcul avec nom, le λ_{wsn}-calcul, constitue le troisième volet de la thèse. Son étude nous permet d'approfondir les liens entre les substitutions explicites et les réseaux de preuve de la logique linéaire. Pour prouver sa normalisation forte, on enrichit l'élimination des coupures des réseaux de preuve avec une nouvelle règle, ce qui nous oblige à prouver que cette nouvelle notion de réduction est fortement normalisante. On effectue enfin la preuve de normalisation forte du λ_{wsn}-calcul par traduction et simulation dans les réseaux de preuve.

Une dernière partie est consacrée à l'observation de relations entre quelques calculs se situant de manière intermédiaire entre le λ-calcul et les réseaux de preuve. Cela nous permet de regarder l'introduction d'opérateurs explicites dans les calculs d'un point de vue différent de celui que l'on a habituellement.

Table des matières

Première partie

Introduction

Les progrès considérables de l'informatique durant les cinquante dernières années, notamment en matière de miniaturisation des composants, lui ont offert des champs d'application toujours plus vastes. Depuis une trentaine d'années, l'apparition d'ordinateurs à bas prix a permis au grand public d'y avoir accès, et le nombre de foyers équipés ne cesse de croître. Les utilisateurs réguliers constatent vite que l'ordinateur est un outil parfois capricieux, voire même imprévisible, et certains exemples de défaillance restent célèbre pour leur coût financier (Ariane 5, plus de cinq millions d'euros). Ces défaillances peuvent avoir deux origines : elles sont dues à un problème technique lié aux composants physiques (puces, cartes, câbles, etc.), ou bien à un problème de logiciel, lui-même dû à une erreur de programmation ; c'est la cause de défaillance la plus fréquente.

Pour tenter d'éliminer ces erreurs (appelées "bugs" dans le jargon informatique — "bogues" en français), l'industrie du logiciel utilise une variété de méthodes plus ou moins efficaces. Parmi celles-ci, on peut trouver des chartes de programmation ayant pour but d'empêcher les programmeurs d'utiliser certains aspects du langage considérés comme des sources d'erreurs, des méthodologies pour "transformer" de façon rigoureuse la spécification d'un programme en son code, ou encore l'utilisation de langages de programmation plus élaborés pour lesquels le compilateur est capable de diagnostiquer un plus grand nombre d'erreurs de programmation (par exemple les langages de la famille ML (SML, O'Caml, ...), le JAVA, etc.). La technique qui reste sans doute la plus employée consiste à tester le programme quand celui-ci est achevé ; or le nombre important, quasi illimité, des données à tester contraint le programmeur à en sélectionner certaines qu'il estimera les plus représentatives de l'utilisation qui sera faite du programme. De ce fait, on n'aura jamais la certitude que toutes les données non-testées ne produisent pas de défaillances.

La sémantique des langages de programmation est une approche mathématique de l'élimination d'erreurs dans les programmes. L'une de ses applications est le développement de nouveaux langages de programmation, en particulier les langages appelés "fonctionnels" (LISP, Scheme, O'Caml, SML, etc.). Ces langages offrent en général les mêmes possibilités que les langages traditionnels mais certains apportent de nouvelles fonctionnalités qui contribuent à éviter les erreurs de programmation. Une autre de ses application consiste à extraire d'un programme terminé un objet mathématique (en général, une fonction) afin de vérifier qu'il correspond bien à la spécification du problème que ce programme est censé résoudre. Cette vérification est en effet plus simple à réaliser avec un objet mathématique qu'avec le programme, d'où l'intérêt de la méthode. Les langages fonctionnels précités, étant issus de la sémantique, sont, de ce fait, plus adaptés à l'application de cette méthode. Pour la plupart de ses applications, la sémantique utilise un formalisme mathématique : le λ-calcul.

* * *

Le λ-calcul a été conçu, dans les années 1930, pour décrire et étudier les fonctions mathématiques et leurs utilisations. Il est apparu qu'il était suffisamment expressif pour décrire les fonctions calculables ; ce sont exactement les fonctions qui nous intéressent pour la sémantique des langages de programmation et c'est pour cela qu'il sert de base à la conception des langages fonctionnels.

L'étude sémantique d'un programme peut se faire de deux manières : la sémantique dénotationnelle s'intéresse au résultat produit, tandis que la sémantique opérationnelle attache plus d'importance à ce qui touche au déroulement de l'exécution du programme. Pour l'étude dénotationnelle, le formalisme tel quel convient très bien ; en revanche, pour l'étude opérationnelle, son procédé de calcul, à savoir la β-réduction, n'est pas assez fin. Pour pouvoir répondre à des questions, concernant par exemple, la quantité de ressources nécessaires à une exécution ou encore les meilleures stratégies d'exécution, il faut raffiner le procédé de calcul. Un des raffinements possibles consiste à expliciter dans le calcul l'opération de substitution — auparavant interne à la β-réduction — ; on parle de calculs avec substitutions explicites.

Que ce soit avec ou sans substitutions explicites, et que ce soit du point de vue dénotationnel ou opérationnel, certaines propriétés dynamiques de ces calculs, comme la confluence et la normalisation

forte, intéressent les sémanticiens. Ces propriétés peuvent être étudiées si les calculs en question sont considérés comme des systèmes de réécriture, la théorie de la réécriture étant un outil qui fournit un cadre formel pour étudier une grande diversité de systèmes dynamiques (calculs, programmes, automates, etc.).

La recherche de ces propriétés de réécriture a donné naissance à une grande famille de calculs avec substitutions explicites, chacun ayant leurs avantages et leurs inconvénients. L'étude de chacun de ces calculs a permis de comprendre un peu plus en profondeur les notions fondamentales qui se cachent derrière l'exécution d'un programme.

* * *

La très grande expressivité de ces calculs est parfois une gêne dans l'étude sémantique des programmes. Elle conduit à rendre fausses certaines propriétés de réécriture pourtant importantes. Or il est très vite apparu qu'on pouvait se restreindre à des sous-ensembles de ces calculs pour lesquels les propriétés en question sont vérifiées. En particulier, la sémantique s'est beaucoup consacrée à l'étude des sous-ensembles typables, car elle permet de faire d'étroites connexions d'une part entre ces calculs et les langages de programmation, d'autre part entre ces calculs et la logique, lesquelles connexions ont des répercussions à la fois sur les langages de programmation et sur la logique.

En allant de la logique vers les langages de programmation, la sémantique a aidé à définir un nouveau paradigme de programmation : la programmation fonctionnelle. Elle a contribué à y intégrer, de manière raisonnée, des structure avancées de programmation, et même à en créer de nouvelles. Par ailleurs, de plus en plus de structures des langages existants, introduites à l'origine pour les besoins de la programmation, sont passées au crible de la sémantique pour être enrichies d'un sens logique.

En allant de la programmation vers la logique, la sémantique a poussé les logiciens à développer leurs formalismes pour une application plus informatique. Par exemple, l'isomorphisme de Curry-Howard est célèbre car il établit qu'un programme écrit dans le λ-calcul correspond à la preuve d'une formule logique et réciproquement. Les sémanticiens se sont alors mis à perfectionner le λ-calcul pour étendre cet isomorphisme à un nombre toujours plus grand de programmes écrits dans les langages usuels. Dans le même mouvement, les formalismes logiques ont été raffinés afin de pouvoir, eux-aussi, être intégrés dans une étude opérationnelle plus précise des programmes et de leur exécution. La logique linéaire en est sans doute le meilleur exemple.

Plusieurs formes de logique sont utilisées en sémantique. L'isomorphisme de Curry-Howard porte sur la logique intuitionniste (c'est un fragment de la logique classique), ce qui a conduit la plupart des systèmes de typage à être décrits dans cette logique. Cependant, il est vite apparu que la logique classique pouvait donner un sens à certaines structures de programmation, poussant la sémantique à s'intéresser à des calculs et à des systèmes de typage qui en sont issus.

Contributions de la thèse

Dans cette thèse, nous nous plaçons dans le cadre de l'étude des λ-calculs avec et sans substitutions explicites typés. Nous nous intéressons à leurs propriétés dynamiques de réécriture, et en particulier à la normalisation forte. La thèse contient trois travaux plus ou moins indépendants.

Le premier, intitulé "PSN implique SN", est consacré à l'approfondissement de connexions entre deux propriétés des calculs avec substitutions explicites : la préservation de la normalisation forte (PSN), et la normalisation forte (SN). On y formalise une technique générale de preuve (initialement suggérée par Hugo Herbelin) qui permet d'utiliser la propriété PSN afin de démontrer la propriété SN. On applique cette technique à différents calculs avec substitutions explicites, ce qui nous conduit, entre autres, à démontrer la normalisation forte de l'un de ces calculs (le $\lambda\upsilon$-calcul).

Le deuxième, intitulé "Normalisation forte du $\overline{\lambda}\mu\tilde{\mu}$-calcul avec substitutions explicites", porte sur un calcul issu de la logique classique. Afin d'en démontrer la normalisation forte, nous avons recours

à la technique formalisée dans le premier travail, ce qui nous demande de démontrer la propriété PSN pour ce calcul. Pour cela, nous utilisons la technique de perpétuité formalisée récemment dans la théorie de la réécriture par Eduardo Bonelli.

Le troisième, intitulé "Le λ_{ws}-calcul avec noms", contient la définition et l'étude d'une nouvelle version d'un calcul avec substitutions explicites. Ce nouveau calcul possède de nombreuses propriétés de réécriture, et se rapproche beaucoup de la logique linéaire. Pour en prouver la normalisation forte, nous introduisons dans celle-ci une nouvelle possibilité de calcul, et nous prouvons que cet ajout ne lui fait pas perdre sa propriété de terminaison.

Plan de la thèse

Dans la première partie, nous introduisons les différentes notions avec lesquelles nous allons travailler : les systèmes de réécriture (chapitre 1), le λ-calcul et les substitutions explicites (chapitre 2), le typage, la logique et les réseaux de preuve de la logique linéaire (chapitre 3). Les trois parties suivantes sont consacrées aux travaux mentionnés ci-dessus : "PSN implique SN" (partie II), "Normalisation forte du $\overline{\lambda}\mu\tilde{\mu}$-calcul avec substitutions explicites" (partie III), "Le λ_{ws}-calcul avec noms" (partie IV). Dans la partie V nous effectuons des corrélations entre certains calculs et les réseaux de preuves de la logique linéaire, pour observer les substitutions explicites sous un autre angle. Enfin, nous concluons en donnant quelques perspectives de travaux futurs.

Chapitre 1

Systèmes de réécriture

Nous présentons ici la notion de système de réécriture ainsi que les problématiques qui y sont liées. Pour cela, nous commençons par définir de façon intuitive puis de façon formelle ce qu'est un système de réécriture, et nous étudions ses propriétés importantes. Le lecteur qui sait déjà que l'union de deux systèmes de réécriture terminant ne termine pas forcément peut passer au chapitre suivant. Pour une présentation plus complète et plus approfondie de la théorie de la réécriture, on pourra consulter [48, 31, 54, 5].

1.1 Notion

1.1.1 Exemple : feu tricolore

On peut modéliser le fonctionnement d'un feu tricolore par un système de réécriture. Celui-ci en est alors un exemple assez simple qui va nous servir de première intuition. Observons le fonctionnement classique d'un feu tricolore. À tout moment, le feu est allumé et l'une de ses trois couleurs brille : soit le vert, soit l'orange, soit le rouge. Supposons qu'au début de notre observation le feu soit vert, on peut le voir passer à l'orange, puis au rouge. Si on formalise un peu, on se donne trois mots : **feuvert**, **feuorange** et **feurouge** qui décrivent l'état possible du feu. On représente le changement d'état par une flèche : $\rightarrow$. Ainsi, notre observation peut se noter par la séquence suivante :

$$\textbf{feuvert} \rightarrow \textbf{feuorange} \rightarrow \textbf{feurouge}.$$

Tous les changements d'états ne sont pas possibles : par exemple, le feu ne peut pas passer du vert au rouge directement, c'est-à-dire qu'on ne peut avoir la séquence **feuvert**$\rightarrow$ **feurouge**. Si on souhaite indiquer à quelqu'un qui n'aurait jamais vu de feu tricolore comment il fonctionne, on peut écrire une petite table qui résume l'ensemble des changements d'états possibles :

feuvert	$\rightarrow$	**feuorange**
feuorange	$\rightarrow$	**feurouge**
feurouge	$\rightarrow$	**feuvert**

De cette façon, il pourra toujours calculer à l'avance l'état suivant du feu à partir de son état courant. Si le feu est à l'orange, c'est-à-dire si on a **feuorange**, la deuxième ligne de la table lui indiquera que l'on obtient **feurouge**.

Avec le vocabulaire des systèmes de réécriture, on appelle *termes* les mots **feuvert**, **feuorange** et **feurouge**, et *ensemble des règles de réécriture* la table ci-dessus (qui contient trois règles). Par abus de langage, au lieu d'*ensemble des règles de réécriture* on dira *règles de réécriture* (ou encore *règles de réduction*).

Voici la définition de notre système de réécriture, que l'on va nommer **Feu** :

- $Trm_{\textbf{Feu}} = \{\textbf{feuvert}, \textbf{feuorange}, \textbf{feurouge}\}$

- $Rgl_{\mathbf{Feu}} = \{\mathbf{feuvert} \to \mathbf{feuorange}, \mathbf{feuorange} \to \mathbf{feurouge}, \mathbf{feurouge} \to \mathbf{feuvert}\}$

On appelle *système de réécriture* la donnée constituée d'un ensemble de termes et de règles de réduction. On appellera (du moins dans l'introduction) *Trm*, indicé par le nom du système de réécriture, l'ensemble des termes, et *Rgl*, indicé de la même façon, l'ensemble des règles.

Dans un système de réécriture, on effectue des *réécritures* des termes. Supposons que l'on souhaite savoir ce que devient **feuorange** après cinq changements de couleur, il est alors facile de le calculer à l'aide de nos trois règles :

$$\overbrace{\mathbf{feuorange} \to \mathbf{feurouge}}^{1} \underbrace{\to \mathbf{feuvert}}_{2} \overbrace{\to \mathbf{feuorange}}^{3} \underbrace{\to \mathbf{feurouge}}_{4} \overbrace{\to \mathbf{feuvert}}^{5}.$$

Une telle séquence est un calcul du terme **feuorange** : les autres termes ont été obtenus à partir de leur prédécesseur en utilisant une règle de notre table. On appelle *réécriture*, ou *réduction*, ou *dérivation*, ou encore *calcul* une suite ordonnée de termes telle que chaque terme puisse être obtenu à partir du précédent en utilisant une règle de réécriture. Par exemple :

1. $\mathbf{feuvert} \to \mathbf{feuorange} \to \mathbf{feurouge}$
2. $\mathbf{feuvert} \to \mathbf{feuorange} \to \mathbf{feurouge} \to \mathbf{feuvert} \to \mathbf{feuorange}$

sont deux réductions possibles du terme **feuvert**. Par contre, la séquence suivante n'est pas une réduction dans notre système de réécriture :

$$\mathbf{feuorange} \to \mathbf{feurouge} \not\to \mathbf{feuorange} \to \mathbf{feurouge} \to \mathbf{feuvert}.$$

En effet, la réduction **feurouge** $\to$ **feuorange** n'est pas possible, car la règle **feurouge** $\to$ **feuorange** n'existe pas.

1.1.2 Exemple : riz créole à la crème

Voici un deuxième exemple qui va nous permettre d'introduire d'autres notions. Nous allons nous intéresser à la réalisation d'un plat de riz créole à la crème fraîche. Pour modéliser ce problème, nous allons définir des termes ainsi que des règles de réduction. Parmi les termes, on définit des termes simples :

- du **riz** (cru),
- de l'**eau**,
- de la **crème** (fraîche),
- du **riz cuit** (nature)
- et enfin notre **accompagnement** (le produit final servi dans l'assiette).

De plus, nous aurons besoin de représenter le contenu de notre casserole, afin de connaître l'avancement de notre préparation. Pour cela, on introduit un objet **casserole** qui peut contenir d'autres termes qu'on appelle *sous-termes*, puisqu'ils sont contenus dans un autre terme. On notera, par exemple, **casserole(riz, eau)** le terme représentant la casserole contenant du riz et de l'eau, et **casserole(riz cuit, crème)** le produit final de notre recette (avant d'être servi).

Une casserole pourra parfois contenir un seul ingrédient, mais comme le terme **casserole** contient deux sous-termes, il nous faut ajouter un autre terme simple **rien** qui représentera l'absence d'ingrédient. Une casserole ne contenant que du riz se notera alors **casserole(riz, rien)**, et le point de départ de notre recette sera la casserole vide : **casserole(rien, rien)**.

Pour préparer du riz créole à la crème, il faut commencer par mettre un volume de riz cru pour trois volume d'eau dans une casserole. Il faut ensuite cuire ce mélange jusqu'à absorption complète de l'eau, et enfin ajouter la crème fraîche. En première approche, on peut donner les règles de réduction suivantes :

(1.) **casserole(rien, rien)** → **casserole(riz, rien)**
(2.) **casserole(riz, rien)** → **casserole(riz, eau)**
(3.) **casserole(riz, eau)** → **casserole(riz cuit, rien)**
(4.) **casserole(riz cuit, rien)** → **casserole(riz cuit, crème)**
(5.) **casserole(riz cuit, crème)** → **accompagnement**

On a numéroté les règles afin de pouvoir plus facilement en parler. Pour cette raison, tous les systèmes de réécriture que nous allons étudier associeront à chaque règle un nom ou un numéro. Lorsqu'on écrira une dérivation, on indiquera parfois quelle règle a été utilisée à chaque étape, de la façon suivante :

$$\textbf{casserole(riz, rien)} \to_2 \textbf{casserole(riz, eau)} \to_3 \textbf{casserole(riz cuit, eau)}$$

Voici la table récapitulant la signification de chaque règle de réduction par rapport à notre recette :

(1.) Ajout du riz dans la casserole.
(2.) Ajout de l'eau dans la casserole.
(3.) Cuisson.
(4.) On ajoute la crème fraîche.
(5.) L'accompagnement est servi dans l'assiette.

Regardons la réduction du terme **casserole(rien, rien)** qui correspond au départ de la recette. On s'aperçoit qu'il n'y a qu'une seule réduction possible de ce terme (comme dans l'exemple du feu tricolore). Cette forte contrainte sur la réduction nous fait poser une question : et si on voulait mettre d'abord l'eau avant le riz dans notre casserole ? Les règles de réduction nous l'interdisent. On va essayer de les modifier pour rendre notre système un peu plus souple, de telle sorte qu'il permette cela. Les deux règles qui nous intéressent sont la (1.) et la (2.) :

(1.) **casserole(rien, rien)** → **casserole(riz, rien)**
(2.) **casserole(riz, rien)** → **casserole(riz, eau)**

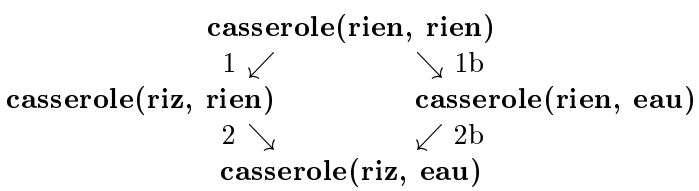
Une première façon de modifier notre système consiste à ajouter *toutes* les règles manquantes pour décrire les réductions souhaitées. Cela nous donne[1] :

(1.) **casserole(rien, rien)** → **casserole(riz, rien)**
(1b.) **casserole(rien, rien)** → **casserole(rien, eau)**
(2.) **casserole(riz, rien)** → **casserole(riz, eau)**
(2b.) **casserole(rien, eau)** → **casserole(riz, eau)**

Avec ces nouvelles règles, un phénomène nouveau se produit : nous avons deux façons de réduire **casserole(rien, rien)** vers **casserole(riz, eau)**. Il y a deux dérivations possibles :

- soit **casserole(rien, rien)** $\to_1$ **casserole(riz, rien)** $\to_2$ **casserole(riz, eau)**,
- soit **casserole(rien, rien)** $\to_{1b}$ **casserole(rien, eau)** $\to_{2b}$ **casserole(riz, eau)**.

On peut représenter ces deux dérivations dans un *graphe de dérivation* :



Dans un graphe de dérivation, on appellera *chemin* une dérivation d'un terme vers un autre.

Regardons d'un peu plus près nos nouvelles règles. Certaines se ressemblent beaucoup. Prenons, par exemple, les règles (1.) et (2b.) :

[1] On remarque que pour conserver l'obtention du terme **casserole(riz, eau)**, on a été obligé d'écrire la règle (1b.) en ajoutant l'eau à droite de la virgule. Nous reviendrons plus tard sur cela (voir section 1.1.5).

(1.) **casserole(rien, rien)** → **casserole(riz, rien)**
(2b.) **casserole(rien, eau)** → **casserole(riz, eau)**

Elle ajoutent toutes les deux **riz** à gauche de la virgule, tout en laissant ce qui est à droite inchangé. On peut se demander s'il n'est pas possible de "regrouper" ou "factoriser" ces deux règles en une seule. Celle-ci dirait : "quelque soit le terme à droite de la virgule, si le terme **rien** est à gauche, je le remplace par **riz**". Pour cela, on va utiliser une *méta-variable* dont la signification sera exactement "quelque soit le terme". On écrit la nouvelle règle de la façon suivante :

(ajouterRiz) **casserole(rien,** t**)** → **casserole(riz,** t**)**

La méta-variable t représente n'importe quel terme présent au moment du calcul. Celui-ci est laissé tel quel dans le terme obtenu. Symétriquement, on remplace les règles (1b.) et (2.) par une seule règle (ajouterEau) et les voici toutes les deux :

(ajouterRiz) **casserole(rien,** t**)** → **casserole(riz,** t**)**
(ajouterEau) **casserole(**t**, rien)** → **casserole(**t**, eau)**

Avec ces deux nouvelles règles, notre graphe de dérivation devient :

casserole(rien, rien)
ajouterRiz ↙ ↘ ajouterEau
casserole(riz, rien) **casserole(rien, eau)**
ajouterEau ↘ ↙ ajouterRiz
casserole(riz, eau)

Pour effectuer une réduction à l'aide d'une de ces règles, on prend le terme à réduire, on identifie un sous-terme qui sera représenté pas la variable t et on réécrit le terme obtenu en remplaçant t par ce sous-terme. Par exemple, pour réduire le terme **casserole(rien, eau)**, on fait d'abord l'association [t = **eau**] pour obtenir **casserole(rien,** t**)**, ensuite on réécrit vers **casserole(riz,** t**)** avec la règle (ajouterRiz), et enfin on remplace t en reprenant l'association [t = **eau**], ce qui nous donne **casserole(riz, eau)**.

On renomme les règles avec des noms plus explicites. Notre ensemble de règles de réduction $Rgl_{\mathbf{Riz}}$ est à présent le suivant :

(ajouterRiz)	**casserole(rien,** t**)**	→	**casserole(riz,** t**)**
(ajouterEau)	**casserole(**t**, rien)**	→	**casserole(**t**, eau)**
(cuire)	**casserole(riz, eau)**	→	**casserole(riz cuit, rien)**
(ajouterCrème)	**casserole(riz cuit, rien)**	→	**casserole(riz cuit, crème)**
(servir)	**casserole(riz cuit, crème)**	→	**accompagnement**

On remarque cependant que la nouvelle règle (ajouterEau) nous permet d'effectuer une réduction supplémentaire : **casserole(riz cuit, rien)** → **casserole(riz cuit, eau)**, car rien ne nous empêche d'associer à la méta-variable de cette règle le terme **riz cuit**. Nous aurons l'occasion de revenir plus tard sur cela. Ici, notre système de réécriture nous permet d'écrire le graphe complet des dérivations de **casserole(rien, rien)**. Le voici :

casserole(rien, rien)
ajouterRiz ↙ ↘ ajouterEau
casserole(riz, rien) **casserole(rien, eau)**
ajouterEau ↘ ↙ ajouterRiz
casserole(riz, eau)
cuire ↙
casserole(riz cuit, rien) $\rightarrow_{\text{ajouterEau}}$ **casserole(riz cuit, eau)**
ajouterCrème ↓
casserole(riz cuit, crème)
servir ↘
accompagnement

Regardons maintenant comment définir nos ensembles $Trm_{\mathbf{Riz}}$ et $Rgl_{\mathbf{Riz}}$. On pourrait, comme dans l'exemple du feu tricolore, énumérer tous les termes possibles. Cela nous donnerait :

$$Trm_{\mathbf{Riz}} = \{\mathbf{casserole(rien,\ rien)}, \mathbf{casserole(rien,\ eau)}, \mathbf{casserole(riz,\ rien)}, \mathbf{casserole(riz,\ eau)}, \mathbf{casserole(riz\ cuit,\ rien)}, \mathbf{casserole(riz\ cuit,\ eau)}, ...\}$$

Il y en a beaucoup, y compris certains qui n'apparaissent pas dans la recette, par exemple **casserole(riz, crème)** et **casserole(riz cuit, eau)**. C'est long et peu lisible d'écrire, sous cette forme, tous les termes possibles. Voici un moyen plus simple et surtout plus concis pour les "construire".

On prépare cette construction en deux étapes. Premièrement on sépare les termes en deux parties : les "briques de base", et les "constructions complexes". Les termes simples **riz**, **eau**, **crème**, **riz cuit**, **rien** et **accompagnement** sont les briques de bases, et l'objet **casserole** est une construction complexe. Avec le vocabulaire des systèmes de réécriture, les "briques de base" sont appelées *termes constants* (ou plus simplement *constantes*) et les constructions complexes sont appelées *symboles de fonction* [2]. **casserole** est un symbole de fonction et **casserole(?, ?)** est un terme, les deux " ?" étant ses deux sous-termes.

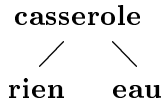
Deuxièmement on regroupe les constantes en trois catégories :

- la catégorie des constantes qui n'apparaissent jamais dans une casserole : elle ne contient que **accompagnement**,
- la catégorie **gauche** : elle contient les constantes qui peuvent apparaître, dans une casserole, à gauche de la virgule (**riz**, **riz cuit** et **rien**),
- la catégorie **droite** : elle contient les constantes qui peuvent apparaître, dans une casserole, à droite de la virgule (**eau**, **crème** et **rien**).

On peut maintenant donner l'ensemble des termes de la manière suivante. On dira qu'un terme est soit la constante **accompagnement**, soit le symbole de fonction **casserole** contenant à gauche une constante de la catégorie **gauche** et à droite une constante de la catégorie **droite**. On écrit cela, plus formellement, à l'aide d'une *grammaire* (le symbole | se lit "ou", et le symbole ::= se lit "est") :

$$\begin{array}{lcl} \mathbf{gauche} & ::= & \mathbf{riz} \mid \mathbf{riz\ cuit} \mid \mathbf{rien} \\ \mathbf{droite} & ::= & \mathbf{eau} \mid \mathbf{crème} \mid \mathbf{rien} \\ Trm_{\mathbf{Riz}} & ::= & \mathbf{accompagnement} \mid \mathbf{casserole(gauche,\ droite)} \end{array}$$

Dans la suite de notre travail, nous aurons souvent des termes qui contiennent des sous-termes. Il est parfois utile, pour guider notre intuition, de les représenter graphiquement. Cela permet de faire apparaître les contenants et les contenus dans une hiérarchie descendante : on lit de haut en bas les contenants puis les contenus. Par exemple, le terme **casserole(rien, eau)** se dessine :



1.1.3 Exemple : addition

L'addition usuelle est notre dernier exemple de système de réécriture. Tout le monde sait calculer le résultat de $2+5+3+8$, à savoir 18, ce que l'on note (à tort [3]) $2+5+3+8 = 18$. En général, lorsque l'on effectue le calcul, on procède par étape de la façon suivante :

$$2+5+3+8 \quad \rightarrow \quad 7+3+8 \quad \rightarrow \quad 10+8 \quad \rightarrow \quad 18$$

À chaque étape, on a isolé un morceau de l'addition (par exemple $2+5$) dont on a calculé la somme (7), puis on a repris ce nouveau nombre dans l'addition et on est passé à l'étape suivante. Cela ressemble fort à une dérivation dans un système de réécriture. Essayons de formaliser un tel système.

[2] par analogie avec la notation usuelle des fonctions : $f(x, y)$

[3] On peut trouver dans [33] un plaidoyer pour la différenciation des notions de calcul et d'égalité

Regardons d'abord ses termes. L'addition est une opération binaire, c'est-à-dire que dans une expression, les opérateurs + portent chacun sur deux opérandes, une opérande pouvant être un nombre ou une addition. Par exemple, dans l'expression $(2+5)+(3+8)$, le premier + porte sur les nombres 2 et 5 tandis que le deuxième + porte sur les additions $(2+5)$ et $(3+8)$; dans l'expression $2+((5+3)+8)$ le premier + porte sur le nombre 2 et l'addition $((5+3)+8)$. Remarquons que le parenthésage contraint l'ordre des étapes du calcul.

Pour les deux expressions que l'on vient d'introduire, on a les deux dérivations suivantes :

$$\begin{array}{llllllll} (2+5)+(3+8) & \to & 7+(3+8) & \to & 7+11 & \to & 18 \\ 2+((5+3)+8) & \to & 2+(8+8) & \to & 2+16 & \to & 18 \end{array}$$

On peut alors dire qu'un terme est soit un nombre, soit un + portant sur deux termes (ce sont ses sous-termes). Pour reprendre le vocabulaire des systèmes de réécriture, un nombre est une constante (choisie parmi les entiers) et un + est un symbole de fonction, c'est-à-dire que si t et t' sont deux termes alors $+(t, t')$ est un terme. Cependant, pour faciliter la lecture, on notera nos termes en plaçant le + entre ses deux sous-termes : $t + t'$.

Voici la grammaire de nos termes où $\underline{\mathrm{n}}$ est une constante (c'est-à-dire un entier) :

$$t ::= \underline{\mathrm{n}} \mid t + t$$

Ici la définition de nos termes est *récursive*, puisque t est décrit en utilisant une référence à lui-même. Ce sera souvent le cas dans les systèmes que nous étudierons. Pour construire un terme avec cette grammaire, on prend soit un nombre (choix 1), soit deux sous-termes avec un + entre eux (choix 2) et on réutilise récursivement la grammaire pour construire les deux sous-termes. Voici quelques exemples.

- Pour construire le terme 2, on utilise le choix 1.
- Pour construire le terme $2 + 3$, on commence par utiliser le choix 2 pour obtenir $t + t'$. Puis on utilise deux fois le choix 1 pour construire les sous-termes t et t' respectivement en 2 et 3.
- Pour construire le terme $((2 + 5) + 3) + 8$, on utilise :
  1. choix 2 : $t_1 + t_2$,
  2. choix 1 pour t_2 : $t_1 + 8$,
  3. choix 2 pour t_1 : $(t_3 + t_4) + 8$,
  4. choix 2 pour t_3 : $((t_5 + t_6) + t_4) + 8$,
  5. choix 1 pour t_5, t_6 et t_4 : $((2 + 5) + 3) + 8$.

Comme pour les termes avec **casserole**, on peut représenter nos termes sous forme d'arbre, par exemple pour $((2 + 5) + 3) + 8$ et $(2 + 5) + (3 + 8)$ on obtient :

Passons maintenant aux règles de réduction. On souhaite dire que si on a un + entre deux nombres, alors on réécrit ce terme vers le nombre représentant la somme de ces deux nombres. Pour cela, on peut naïvement imaginer écrire les règles de la façon suivante :

$$\begin{array}{lll} 0+0 & \to & 0 \\ 0+1 & \to & 1 \\ 1+0 & \to & 1 \\ 1+1 & \to & 2 \\ 1+2 & \to & 3 \\ \ldots & & \end{array}$$

On se rend vite compte qu'il nous faudra une infinité de règles pour notre système, ce qui ne nous convient pas.

On donne une règle qui utilise des méta-variables de nombres (notées $\underline{\mathrm{n}}$ et $\underline{\mathrm{m}}$). Celle-ci dira : "quelque soit le terme $\underline{\mathrm{n}} + \underline{\mathrm{m}}$, le terme obtenu par la réduction est la constante $\underline{\mathrm{k}}$ obtenue par la somme des nombres n et m", c'est-à-dire $k = n + m$ (n est un nombre entier et $\underline{\mathrm{n}}$ est un terme de notre système). On s'aperçoit bien qu'il y a une ambiguïté sur la compréhension du signe $+$. En effet, le $+$ de $\underline{\mathrm{n}} + \underline{\mathrm{m}}$ est un symbole de fonction, il fait partie de la *syntaxe* de nos termes : on parle du $+$ syntaxique ; le $+$ de $k = n + m$ est l'opérateur arithmétique qui effectue la somme de deux entiers, il donne le *sens* de l'addition : on parle du $+$ sémantique. Pour éviter toute confusion, on notera $\oplus$ le $+$ sémantique.

Ainsi, $5 + 3$ est le terme décrivant l'addition de 5 et de 3, alors que $5 \oplus 3$ (qui n'est pas un terme) vaut 8 (qui est un terme). Faire une réduction va alors nécessiter d'effectuer un calcul : à chaque étape, il faudra calculer les $\oplus$ pour retrouver des termes de notre système de réécriture. On aura, par exemple, la dérivation suivante :

$$(5 + 3) + 7 \quad \to \quad (5 \oplus 3) + 7 \; = 8 + 7 \quad \to \quad 8 \oplus 7 \; = 15$$

On dit que l'opérateur $\oplus$ est un opérateur *implicite*, car son calcul n'est pas décrit dans notre système : $\oplus$ n'est pas défini dans la grammaire des termes, et la façon de calculer les $n \oplus m$ n'est pas indiquée sous forme de règle. On verra par la suite qu'il peut être intéressant de rendre *explicites* certains opérateurs implicites de certains systèmes.

Voici enfin la règle de réduction de notre système :

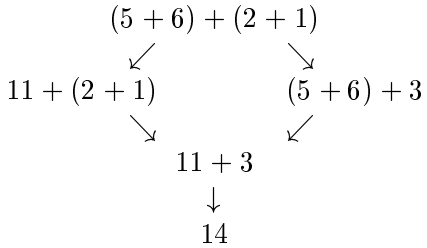
$$\underline{\mathrm{n}} + \underline{\mathrm{m}} \quad \to \quad \underline{n \oplus m}$$

On a écrit $\underline{n \oplus m}$ comme membre droit de la règle, mais il faut bien se souvenir qu'on réalise instantanément le calcul $n \oplus m$.

Pour réduire tous les termes, il y a encore un problème. En effet, on peut réduire le terme $7 + 4$ pour obtenir $7 \oplus 4$, c'est-à-dire 11, mais on ne peut pas réduire $(2 + 4) + 3$ car ce terme n'est pas de la forme : un nombre suivi du symbole $+$ suivi d'un autre nombre. Par contre, il contient un sous-terme qui est de cette forme : $2 + 4$. On voudrait donc commencer par réduire $2 + 4$ vers 6, ce qui nous donnerait le terme $6 + 3$ qui est de la bonne forme.

Comme dans la plupart des systèmes de réécriture que nous étudierons par la suite, on veut pouvoir réduire n'importe quel sous-terme. Pour cela, on va avoir besoin de la notion de *contexte*. Intuitivement, le contexte est ce qui se trouve "autour" du sous-terme considéré : par exemple, pour $(2 + 4) + 3$, le contexte de $2 + 4$ est $(\Box) + 3$ où $\Box$ marque l'emplacement du sous-terme. Grâce a cette notion, on va enrichir notre système de réécriture en disant que les règles "passent au contexte", c'est-à-dire que l'on peut réduire à tout moment n'importe quel sous-terme. C'est d'ailleurs ce qu'on a fait naturellement lorsqu'on a "isolé un morceau de l'addition dont on a calculé la somme".

Pour finir, on remarque qu'ici aussi plusieurs réductions d'un même terme sont possibles :



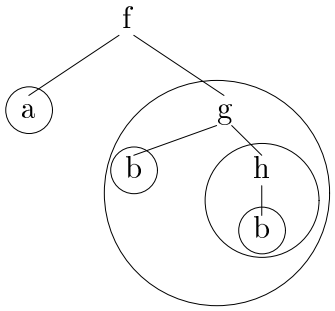
Armés de ces exemples, nous pouvons passer à la définition formelle de la notion de système de réécriture.

1.1.4 Définition

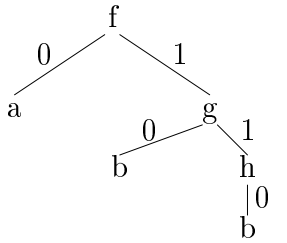
Soit $\mathcal{L}$ un *langage*, c'est-à-dire un ensemble de constantes (notées, dans nos exemples, a, b, c, etc.) et de symboles de fonction (notés, dans nos exemples, f, g, h, etc.) auxquels sont associés une *arité*, c'est-à-dire le nombre de termes qu'il contient (par exemple, l'arité de notre fonction **casserole** est 2, comme celle du +).

Définition 1.1.1 (Terme et sous-terme)

- On appelle *terme* un objet qui est soit une constante de $\mathcal{L}$, soit un symbole de fonction auquel on donne autant de termes que son arité. Autrement dit, on construit l'ensemble *Trm* de la façon suivante :
  - les constantes peuvent être des termes,
  - si $t_1, t_2, ..., t_n$ sont des termes et si f est un symbole de fonction (de $\mathcal{L}$) d'arité n, alors $f(t_1, t_2, ..., t_n)$ peut être un terme.
- Si $f(t_1, t_2, ..., t_n)$ est un terme, $\{t_1, t_2, ..., t_n\}$ sont les *sous-termes propres* de ce terme.
- Si $f(t_1, t_2, ..., t_n)$ est un terme, l'ensemble de ses *sous-termes stricts* comprend ses sous-termes propres ($\{t_1, t_2, ..., t_n\}$) ainsi que les sous-termes stricts de $t_1, t_2, ..., t_n$. Par exemple, si $f(a, g(b, h(b)))$ est un terme, l'ensemble de ses sous-termes propres est $\{a,\ g(b, h(b))\}$ et l'ensemble de ses sous-termes stricts est $\{a,\ g(b, h(b)),\ b,\ h(b)\}$. On peut aussi voir les sous-termes comme des morceaux de l'arbre représentant le terme :



  Tous les sous-termes stricts sont entourés d'un cercle. On remarque que le terme b apparaît deux fois comme sous-terme. Pour les distinguer, nous allons indiquer la position de chaque sous-terme en numérotant les arcs de notre arbre. La position d'un sous-terme sera alors donnée par la suite des numéros. Dans notre exemple :



  Les deux occurrences de b sont aux positions 10 et 110.
- L'ensemble des *sous-termes* d'un terme est l'ensemble de ses sous-terme stricts et le terme lui-même (de la même façon que l'ensemble des nombres inférieurs ou égaux à un entier sont les nombres *strictement* inférieurs et le nombre lui-même).

Pour définir nos règles de réécriture, nous avons vu qu'il nous faut des termes avec méta-variables. Voici leur définition.

Définition 1.1.2 (Terme avec méta-variables, filtrage)

- En plus de notre ensemble de termes *Trm*, on se donne un ensemble de *méta-variables* qui

peuvent prendre la place de n'importe quel terme. Plus formellement, on contruit un terme avec méta-variable en utilisant les règles de construction des termes présentées ci-dessus plus la suivante :
– Les méta-variables sont des termes.
Un terme avec méta-variable est aussi appelé *terme ouvert* par opposition aux termes sans méta-variables qui sont appelés *terme clos*.

- Le *filtrage* est le procédé qui permet d'associer un terme t avec un terme u possédant des méta-variables. Le résultat est une liste d'associations entre les méta-variables de u et des sous-termes de t. Elle permet de retrouver t si on remplace celles-ci par les sous-termes correspondants. Le filtrage peut ne pas être possible si ces deux termes sont trop différents. Regardons quelques exemples pour bien comprendre comment cela fonctionne. On se donne deux symboles de fonction f et g, trois constantes a, b, et c, et trois méta-variables x, y et z.
  – $f(x)$ peut filtrer $f(a)$ et le résultat est $[x = a]$, car si on remplace x par a dans $f(x)$, on obtient $f(a)$.
  – $f(x)$ peut filtrer $f(b)$ et le résultat est $[x = b]$.
  – $f(x)$ peut filtrer $f(g(a, b))$ et le résultat est $[x = g(a, b)]$.
  – $f(f(x))$ peut filtrer $f(f(f(a)))$ et le résultat est $[x = f(a)]$.
  – $g(x, y)$ peut filtrer $g(a, b)$ et le résultat est $[x = a, y = b]$.
  – $g(x, g(y, z))$ peut filtrer $g(f(a), g(b, f(c)))$ et le résultat est $[x = f(a), y = b, z = f(c)]$.
  – $g(x, a)$ peut filtrer $g(b, a)$ et le résultat est $[x = b]$.
  – $g(x, a)$ ne filtre pas $g(b, c)$ car même si on remplace x par b dans $g(x, a)$, on obtient $g(b, a)$ qui est différent de $g(b, c)$.
  – $g(x, a)$ ne filtre pas non plus $f(a)$.
  – x (ou n'importe quelle méta-variable) peut filtrer n'importe quel terme.
  – Un terme sans méta-variable ne peut être filtré que par lui-même, rendant comme résultat la liste vide [].

Définition 1.1.3 (Règle de réécriture)

Une *règle de réécriture* ou *règle de réduction* est un couple de termes avec méta-variables. On appelle le premier élément du couple le membre gauche (MG) et l'autre le membre droit (MD). Si t est le membre gauche et u le membre droit de la règle, on note cette règle $t \to u$. Pour être une règle de réduction, le couple (MG, MD) doit respecter les contraintes suivantes :
– le membre gauche ne doit pas être une méta-variable, c'est-à-dire qu'on ne peut pas avoir de règle, dans notre exemple créole, comme :

$$\text{t} \to \textbf{accompagnement}$$

ce qui nous autoriserait à transformer n'importe quoi en **accompagnement** ;
– l'ensemble des méta-variables du membre droit doit être inclus dans celui du membre gauche. Par exemple, si f, g et h sont des symboles d'arité 1, 2 et 3 respectivement, et si x, y, et z sont des méta-variables, les règles 1. et 2. sont autorisées, mais pas la 3. :

$$\begin{array}{lrcl} 1. & h(x, y, z) & \to & g(z, x) \\ 2. & g(x, y) & \to & g(f(y), x) \\ 3. & f(x) & \to & g(x, y) \end{array}$$

En effet, on ne peut pas calculer le résultat de l'application de la règle 3. puisqu'on ne sait pas par quoi remplacer y dans $g(x, y)$.

Comme nous l'avons vu dans l'exemple de l'addition, pour pouvoir utiliser les règles de réécriture n'importe où dans les termes, on a besoin de la notion de contexte.

Définition 1.1.4 (Contexte)

Un *contexte* est un terme qui contient exactement une fois le symbole $\Box$ à la place d'un sous-terme.

Par exemple :

- $f(\Box)$, $h(x, \Box, z)$ et $h(g(x, \Box), f(x), y)$ sont des contextes,
- $g(z, x)$ et $h(f(\Box), x, \Box)$ n'en sont pas (pas de $\Box$ dans le premier cas, et trop dans le deuxième).

On notera habituellement $C[]$, $C'[]$, etc. des contextes.

On peut *plonger* (ou *mettre*) un terme dans un contexte : cela consiste à remplacer le $\Box$ par ce terme. On note cela en insérant le terme à plonger entre les crochets. Par exemple, si $C[]$ est $f(\Box)$, alors $C[g(x, y)]$ est $f(g(x, y))$; de même si $C[] = h(g(x, \Box), f(x), y)$ alors $C[f(z)] = h(g(x, f(z)), f(x), y)$.

Voici enfin la définition d'un système de réécriture.

Définition 1.1.5 (Système de réécriture)

Un *système de réécriture* est un couple (Trm, Rgl) où Trm est un ensemble de termes, et Rgl un ensemble de règles de réécriture.

Exemple 1.1.6 (Feu tricolore)

Les termes sont les trois constantes **feuvert**, **feuorange** et **feurouge**. Voici les règles de réduction :

$$\begin{array}{lcl} \textbf{feuvert} & \to & \textbf{feuorange} \\ \textbf{feuorange} & \to & \textbf{feurouge} \\ \textbf{feurouge} & \to & \textbf{feuvert} \end{array}$$

Exemple 1.1.7 (Riz)

Les termes sont donnés par la grammaire suivante :

$$\begin{array}{lcl} \textbf{gauche} & ::= & \textbf{riz} \mid \textbf{riz cuit} \mid \textbf{rien} \\ \textbf{droite} & ::= & \textbf{eau} \mid \textbf{crème} \mid \textbf{rien} \\ \mathrm{t} & ::= & \textbf{accompagnement} \mid \textbf{casserole(gauche, droite)} \end{array}$$

Voici les règles de réduction :

$$\begin{array}{lllcl} (\text{ajouterRiz}) & \textbf{casserole(rien,} & \mathrm{t}\textbf{)} & \to & \textbf{casserole(riz,} \ \mathrm{t}\textbf{)} \\ (\text{ajouterEau}) & \textbf{casserole(}\mathrm{t}\textbf{,} & \textbf{rien)} & \to & \textbf{casserole(}\mathrm{t}\textbf{,} \ \textbf{eau)} \\ (\text{cuire}) & \textbf{casserole(riz,} & \textbf{eau)} & \to & \textbf{casserole(riz cuit,} \ \textbf{rien)} \\ (\text{ajouterCrème}) & \textbf{casserole(riz cuit,} & \textbf{rien)} & \to & \textbf{casserole(riz cuit,} \ \textbf{crème)} \\ (\text{servir}) & \textbf{casserole(riz cuit,} & \textbf{crème)} & \to & \textbf{accompagnement} \end{array}$$

Exemple 1.1.8 (Addition)

Les termes sont donnés par la grammaire suivante :

$$t ::= \underline{\mathrm{n}} \mid t + t$$

Voici la règle de réduction :

$$\underline{\mathrm{n}} + \underline{\mathrm{m}} \quad \to \quad \underline{\mathrm{n} \oplus \mathrm{m}}$$

On donne à présent la définition d'une réduction (ou réécriture), ce qui va nous permettre d'expliquer plus formellement ce que signifie *passer au contexte*.

Définition 1.1.9 (Réécriture (réduction, calcul, dérivation), redex)

- Pour effectuer une *étape de réécriture* (ou *réduction*, *calcul*, *dérivation*) à partir d'un terme

quelconque t, on procède de la façon suivante :

1. On choisit une règle de réécriture qui peut s'appliquer, c'est-à-dire pour laquelle le point 2. ci-dessous est possible.
2. On choisit un sous-terme t' qui peut être réécrit (ou réduit), c'est-à-dire qui peut être filtré par le membre gauche de la règle précédemment choisie. Ce sous-terme s'appelle alors un *redex* (pour *REDucible EXpression*, *expression réductible*) et on dit que c'est celui-là qu'on réduit.
3. On prend tout ce qui est autour de t' comme contexte, c'est-à-dire qu'on prend un contexte $C[]$ tel que $C[t'] = t$.
4. On filtre t' par le membre gauche de la règle pour obtenir une liste d'associations (voir la définition 1.1.2).
5. On utilise cette liste d'associations pour remplacer les méta-variables du membre droit de la règle par les sous-termes correspondants, ce qui nous donne un terme t''.
6. Enfin, on plonge ce terme dans le contexte d'origine et on obtient le terme final $C[t'']$.

Si t se réduit en u, on note $t \to u$. On peut préciser quelle règle de réduction a été utilisée en mettant son nom en indice de la flèche : $t \to_r u$. Les étapes 3. et 6. effectuent le *passage au contexte* : on retire le sous-terme de son contexte, puis on y replonge le sous-terme réduit.

- On note $\to^+$ la clôture transitive de $\to$, c'est-à-dire que si on a $t_1 \to t_2 \to t_3 \to ... \to t_n$, on peut noter cette réduction $t_1 \to^+ t_n$.
- On note $\to^*$ la clôture transitive et réflexive de $\to$, c'est-à-dire qu'on note $t_1 \to^* t_n$ si $t_1 \to^+ t_n$ ou si on a effectué aucune étape de réduction, auquel cas on a $t_1 = t_n$.
- On note $\to^n$ pour dénoter exactement n étapes de réduction, c'est-à-dire qu'on note $t_1 \to^n t_n$ si $t_1 \underbrace{\to t_2 \to ... \to}_{n\ fois} t_n$. Si $n = 0$, alors aucune réduction n'a eu lieu : $t_1 \to^0 t_2$ est une autre notation pour $t_1 = t_2$. Ainsi, $t_1 \to^+ t_2$ est équivalent à $t_1 \to^n t_2$ avec $n > 0$, et $t_1 \to^* t_2$ est équivalent à $t_1 \to^n t_2$ avec $n \geq 0$.
- Si $t \to^* u$, alors on dit que u est un *réduit* de t.
- L'ensemble des *redex* d'un terme est l'ensemble de ses sous-termes que l'on peut réduire.

Exemple 1.1.10 (Réductions)

- Réduisons le terme **feuvert** :
  1. C'est la règle **feuvert** → **feuorange** que nous choisissons
  2. car notre terme **feuvert** est le même terme que le membre gauche de cette règle (**feuvert**) et peut donc s'unifier avec lui.
  3. Le contexte autour de **feuvert** dans notre terme est le contexte vide : $C[] = \square$, car C[**feuvert**] = **feuvert** (autrement dit, il n'y a rien autour de **feuvert**).
  4. Le filtrage nous donne la liste vide [].
  5. Puisqu'elle est vide, nous n'avons rien à remplacer dans le membre droit de la règle (qui ne contient d'ailleurs pas de méta-variables, conformément à la restriction sur la formation des règles de réécriture), ce qui nous donne **feuorange**.
  6. On place ce dernier dans le contexte pour obtenir le terme réduit : C[**feuorange**] = **feuorange**.
- Réduisons le terme **casserole(rien, eau)** :
  1. C'est la règle ajouterRiz : **casserole(rien,** t**)** → **casserole(riz,** t**)** que nous choisissons

2. car notre terme **casserole(rien, eau)** peut s'unifier avec **casserole(rien,** t**)**.
3. Le contexte autour de **casserole(rien, eau)** dans notre terme est le contexte vide : $C[\,] = \Box$, car C[**casserole(rien, eau)**] = **casserole(rien, eau)**.
4. Le filtrage nous donne l'association [t=**eau**] car si on remplace t par **eau** dans **casserole(rien,** t**)** on obtient **casserole(rien, eau)**.
5. On utilise cette association pour remplacer t dans le membre droit de la règle, ce qui nous donne **casserole(riz, eau)**.
6. On place ce dernier dans le contexte pour obtenir le terme réduit : C[**casserole(riz, eau)**] = **casserole(riz, eau)**.

- Réduisons le terme $(5 + 3) + 4$:
  1. On prend la seule règle disponible (et elle peut s'appliquer) : $\underline{\mathrm{n}} + \underline{\mathrm{m}} \to \underline{\mathrm{n} \oplus \mathrm{m}}$.
  2. Le sous-terme $5 + 3$ de notre terme peut s'unifier avec $\underline{\mathrm{n}} + \underline{\mathrm{m}}$ (rappelons-nous que $\underline{\mathrm{n}}$ et $\underline{\mathrm{m}}$ doivent être des entiers).
  3. Le contexte autour de $5+3$ dans notre terme est le contexte $C[\,] = (\Box)+4$, car $C[5+3] = (5 + 3) + 4$.
  4. Le filtrage nous donne l'association [n=5, m=3] car si on remplace n par 5 et m par 3 dans $\underline{\mathrm{n}} + \underline{\mathrm{m}}$ on obtient $5 + 3$.
  5. On utilise cette association pour remplacer n et m dans le membre droit de la règle, ce qui nous donne $5 \oplus 3$, qui vaut 8.
  6. On place ce dernier dans le contexte pour obtenir le terme réduit : $C[8] = 8 + 4$.
- Réduisons le terme $((5 + 3) + (4 + 2)) + 6$:

$$((5 + 3) + (4 + 2)) + 6 \to (8 + (4 + 2)) + 6 \to (8 + 6) + 6 \to 14 + 6$$

On peut noter $((5 + 3) + (4 + 2)) + 6 \to^+ 14 + 6$ (et donc $((5 + 3) + (4 + 2)) + 6 \to^* 14 + 6$). Par ailleurs, on peut noter $((5 + 3) + (4 + 2)) + 6 \to^* ((5 + 3) + (4 + 2)) + 6$.

On veut, dans certains cas, comparer les réductions entre elles. Une comparaison possible est la notion de *préfixe*.

Définition 1.1.11 (Préfixe)

On dit qu'une réduction

$$t_1 \to_{r_1} t_2 \to_{r_2} \ldots \to_{r_{n-1}} t_n$$

est un *préfixe* dans une autre réduction

$$t'_1 \to_{r'_1} t'_2 \to_{r'_2} \ldots \to_{r'_{p-1}} t'_p$$

si $p > n$ et $t_1 = t'_1$ et si pour tout i compris entre 1 et $n - 1$ on a $r_i = r'_i$ (ce qui fait que l'on a aussi $t_i = t'_i$ et $t_n = t'_n$), c'est-à-dire que la deuxième réduction s'écrit :

$$t_1 \to_{r_1} t_2 \to_{r_2} \ldots \to_{r_{n-1}} t_n \to_{r'_n} t'_{n+1} \to_{r'_{n+1}} \ldots \to_{r'_{p-1}} t'_p$$

Plus simplement, les préfixes d'une réduction sont les débuts, de différentes tailles, de cette réduction. Par exemple, $(5+4)+(3+2) \to (5+4)+5 \to 9+5$ est un préfixe de $(5+4)+(3+2) \to (5+4)+5 \to 9+5 \to 14$, tout comme $(5+4)+(3+2) \to (5+4)+5$ l'est aussi.

1.1.5 Équivalences

Revenons à notre système **Addition**. On sait que l'addition usuelle est commutative, c'est-à-dire qu'on peut écrire $6 + (5 + 4) = (4 + 5) + 6$ sans même avoir besoin de calculer les résultats. Plus généralement, on dit que si e_1 et e_2 sont deux expressions arithmétiques, alors $e_1 + e_2 = e_2 + e_1$. Lorsque nous avons défini notre système de réécriture, nous avons transformé les égalités en flèches : $2 + 3 = 5$ est devenu $2 + 3 \to 5$. On peut faire de même avec l'égalité de commutativité, mais avec une conséquence inattendue. Tout d'abord, ajoutons à la définition de notre système, la règle de commutativité ; l'ensemble des règles de réduction est à présent le suivant :

$$\begin{array}{lcll} \underline{\mathrm{n}} + \underline{\mathrm{m}} & \to & \underline{\mathrm{n} \oplus \mathrm{m}} & \\ e_1 + e_2 & \to & e_2 + e_1 & \text{où } e_1 \text{ et } e_2 \text{ sont deux méta-variables.} \end{array}$$

Grâce à la nouvelle règle, on peut, pour un même terme, effectuer plus de dérivations. Par exemple, on peut réduire différemment le terme $(2 + 5) + (3 + 8)$:

$$(2 + 5) + (3 + 8) \quad \to \quad 7 + (3 + 8) \quad \to \quad (3 + 8) + 7 \quad \to \quad 11 + 7 \quad \to \quad 7 + 11 \quad \to \quad 18$$

Cette plus grande souplesse dans les réductions nous convient, mais on constate que l'ajout de cette règle de commutativité permet de faire des réductions infinies :

$$2 + 5 \quad \to \quad 5 + 2 \quad \to \quad 2 + 5 \quad \to \quad 5 + 2 \quad \to \quad 2 + 5 \quad \to \quad 5 + 2 \quad \to \quad ...$$

Nous venons de perdre une propriété importante : la *terminaison* de notre système de réécriture (voir la section suivante). Cela n'est pas satisfaisant, on souhaite avoir un moyen d'utiliser la commutativité sans pouvoir faire des réductions infinies. La solution est simple : il suffit de ne plus considérer la commutativité comme une étape de calcul. Comme en mathématiques, on va considérer cette propriété comme une *équivalence* entre nos termes.

On supprime la règle qu'on vient d'ajouter et on la convertit en équivalence :

$$e_1 + e_2 \quad \sim \quad e_2 + e_1$$

On joue sur les mots (ou plutôt sur les symboles), mais cela résout notre problème : comme on ne compte plus les étapes d'équivalence comme des réductions, on peut en effectuer autant qu'on veut, cela compte comme pour une seule. Par exemple, si on réduit le terme $((1 + 2) + 3) + (4 + 5)$ de la façon suivante :

$$2 + (4 + (5 + 6)) \quad \to \quad 2 + (4 + 11) \quad \sim \quad 2 + (11 + 4) \quad \sim \quad (11 + 4) + 2 \quad \to \quad 15 + 2$$

on pourrait écrire cette réduction :

$$2 + (4 + (5 + 6)) \quad \to \quad \underbrace{2 + (4 + 11) \quad \sim \quad (11 + 4) + 2}_{2 + (4 + 11)} \quad \to \quad 15 + 2$$

ce qui met en évidence qu'on a seulement deux étapes de réécriture. Cependant, pour bien voir les termes réduits, on écrira en général :

$$2 + (4 + (5 + 6)) \quad \to \quad 2 + (4 + 11) \quad \sim \quad (11 + 4) + 2 \quad \to \quad 15 + 2$$

La notation ci-dessus est un abus, il faudrait remplacer le $\sim$ par un $\sim^*$ pour marquer le fait qu'on a utilisé plusieurs étapes d'équivalence.

Définition 1.1.12 (Règle d'équivalence)

> Une *règle d'équivalence* ou *équivalence* est un couple de termes avec méta-variables. Si t est le membre gauche et u le membre droit de l'équivalence, on note cette règle $t \sim u$. Une règle d'équivalence peut s'appliquer dans deux sens : de t vers u ou bien de u vers t.

Définition 1.1.13 (Système de réécriture modulo)

> Un *système de réécriture modulo* est un système de réécriture qui contient en plus un ensemble de règles d'équivalences (qui passent elles-aussi au contexte).

La notion de réécriture s'étend avec les équivalences.

Définition 1.1.14 (Réécriture modulo)

> Soit $\sim$ l'équivalence engendrée par l'ensemble des règles d'équivalence. Celle-ci rend équivalents des termes $t \sim u$ qui sont liés par zéro, une ou plusieurs applications de n'importe quelle règle de cet ensemble. On peut présenter la *réécriture modulo* de deux façons, l'une plutôt mathématique, et l'autre plutôt calculatoire.
>
> - On applique les règles de réduction sur les termes de notre ensemble quotienté par l'équivalence $\sim$.
> - On dit que $t \to u$ si il existe deux termes t' et u', ainsi qu'une règle r de notre ensemble de règles tels que $t \sim t' \to_r u' \sim u$. L'équivalence $\sim$ étant une relation réflexive, on peut avoir $t = t'$ et $u = u'$.

On peut ajouter des équivalences dans de nombreux système de réécriture, la plupart des systèmes que nous étudierons dans cette thèse en comportent. En plus des exemples ci-dessus, on pourra réfléchir aux conséquences de l'ajout d'une règle d'équivalence $\mathbf{casserole}(t_1,\ t_2) \sim \mathbf{casserole}(t_2,\ t_1)$ dans le système de réécriture **Riz**.

D'autres définitions de notions seront nécessaires. Nous les donnerons en présentant les propriétés qui les utilisent.

1.2 Propriétés

Nous allons étudier deux propriétés : la confluence et la normalisation. Pour chacune d'elles, nous commencerons par donner des intuitions à partir d'exemples, puis nous en donnerons les définitions et les propriétés. Ces propriétés s'appliquent aux termes ouverts, et, par conséquence immédiate, aux termes clos.

1.2.1 Confluence

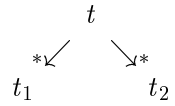
Intuition

L'intuition la meilleure nous vient de l'exemple de l'addition. Il y a une question que l'on se pose rarement car on connaît la réponse, c'est la suivante : lorsqu'on effectue une addition, est-ce que l'ordre dans lequel on procède change le résultat ? Par exemple, si on veut calculer le résultat de $(5+6)+(2+1)$ cela revient-il au même de commencer par calculer $5+6$ ou d'abord $2+1$? On peut représenter ces deux réductions, on l'a vu, sous forme d'un graphe de dérivation :

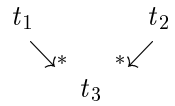
$$\begin{array}{ccc} & (5+6)+(2+1) & \\ \swarrow & & \searrow \\ 11+(2+1) & & (5+6)+3 \\ \searrow & & \swarrow \\ & 11+3 & \end{array}$$

On peut ici répondre positivement à la question : dans les deux cas, on peut obtenir le terme $11+3$. Mais constater le fait sur un exemple ne constitue pas une garantie. On se pose la question dans le cas général : partant d'un terme quelconque t, si on fait deux réductions (potentiellement très grandes ou bien nulles) de ce terme jusqu'à obtenir les termes t_1 et t_2, existe-t-il une réduction de t_1

et une réduction de t_2 qui mènent toutes deux à un même terme t_3 ? Autrement dit, si on a le graphe de dérivation suivant



Existe-t-il un terme t_3 tel que le graphe de dérivation suivant soit vérifié



Un système qui vérifierait cette propriété serait alors appelé *confluent*.

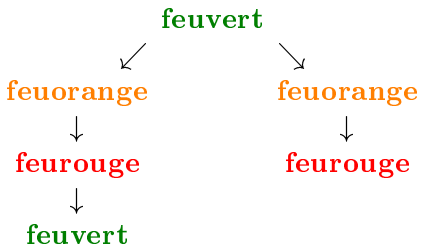
Par exemple, dans le cas de l'addition, si on a

$$\begin{array}{ccc} & ((1+2)+((3+4)+5))+(6+(7+8)) & \\ \swarrow & & \searrow \\ (3+((3+4)+5))+(6+(7+8)) & & ((1+2)+((3+4)+5))+(6+15) \\ \downarrow & & \downarrow \\ (3+((3+4)+5))+(6+15) & & ((1+2)+(7+5))+(6+15) \\ \downarrow & & \downarrow \\ (3+(7+5))+(6+15) & & ((1+2)+(7+5))+21 \\ & & \downarrow \\ & & ((1+2)+12)+21 \end{array}$$

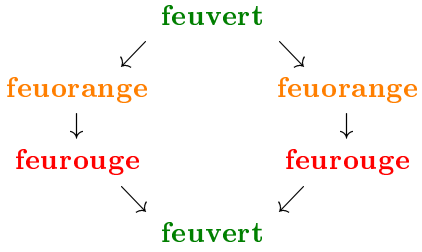
On constate qu'il existe trois réduits communs de $(3+(7+5))+(6+15)$ et $((1+2)+12)+21$: $(3+12)+21$, $15+21$ et 36. Regardons comment on peut fermer le diagramme ci-dessus avec le premier réduit commun.

$$\begin{array}{ccc} (3+(7+5))+(6+15) & & ((1+2)+12)+21 \\ \downarrow & & \downarrow \\ (3+12)+(6+15) & \rightarrow & (3+12)+21 \end{array}$$

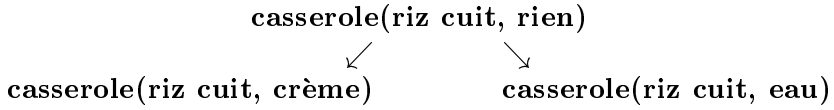
Comme on le verra plus loin, il est possible de prouver que notre système de réécriture **Addition** est confluent. Tournons-nous à présent vers les autres systèmes que l'on a étudiés au début de ce chapitre. Il n'est pas difficile de se convaincre que le système **Feu** est confluent. En effet, quelque soit les deux dérivations partant d'un même terme, l'une est préfixe de l'autre. Pour les joindre, il suffit de poursuivre comme dans la dérivation la plus longue. Par exemple :



peuvent se joindre sur le terme **feuvert**, ce qui nous donne le diagramme complet suivant :



Par contre, le système **Riz** n'est pas confluent. Par exemple :



À ce stade, il est possible de réduire **casserole(riz cuit, crème)**, mais pas vers un réduit de **casserole(riz cuit, eau)**, et aucune règle ne nous permet de réduire **casserole(riz cuit, eau)**. Il est donc impossible de joindre ces deux termes.

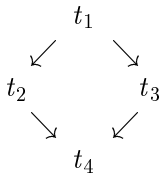
En informatique, cette question est importante; en effet, si on souhaite parfois que les programmes se comportent différemment d'une fois sur l'autre, on espère toujours que la somme indiquée au bas de sa propre feuille de paye ne dépendra pas du choix que l'ordinateur aura fait pour ce calcul.

Définition

Nous passons maintenant aux définitions formelles des différentes formes de confluence. Dans ces définitions, nous travaillons sur des termes qui peuvent conternir des méta-variables. Il existe une autre forme de confluence, plus faible, qui porte uniquement sur les termes sans méta-variables, appelés *termes clos*. La confluence englobe trivialement la confluence sur les termes clos.

Définition 1.2.1 (Propriété du diamant)

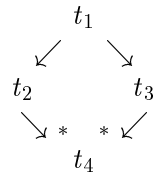
On dit qu'un système de réécriture vérifie la *propriété du diamant* si, pour tous termes t_1, t_2, t_3 tels que $t_1 \to t_2$ et $t_1 \to t_3$, il existe un terme t_4 tel que $t_2 \to t_4$ et $t_3 \to t_4$. Autrement dit, il est possible de fermer le diagramme suivant :



Cette propriété est très forte, et les systèmes la vérifient rarement.

Définition 1.2.2 (Confluence locale)

On dit qu'un système de réécriture est *localement confluent* si, pour tous termes t_1, t_2, t_3 tels que $t_1 \to t_2$ et $t_1 \to t_3$, il existe un terme t_4 tel que $t_2 \to^* t_4$ et $t_3 \to^* t_4$. Autrement dit, il est possible de fermer le diagramme suivant :

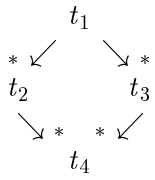


La différence avec la propriété précédente tient dans les deux étoiles qui indiquent qu'on peut à présent fermer le diagramme en zéro, une ou plusieurs étapes. Cette propriété s'observe plus souvent que la précédente.

Définition 1.2.3 (Confluence)

On dit qu'un système de réécriture est *confluent* si, pour tous termes t_1, t_2, t_3 tels que $t_1 \to^* t_2$ et $t_1 \to^* t_3$, il existe un terme t_4 tel que $t_2 \to^* t_4$ et $t_3 \to^* t_4$. Autrement dit, il est possible de fermer

le diagramme suivant :



C'est la propriété que nous avons énoncée dans le paragraphe précédent.

Preuves et exemples

Armés de ces définitions, nous allons pouvoir essayer de vérifier ces propriétés sur nos trois premiers systèmes de réécriture, ainsi que sur d'autres exemples.

Proposition 1.2.4 (Confluences de Feu)

1. Le système **Feu** vérifie la propriété du diamant.
2. Le système **Feu** est localement confluent.
3. Le système **Feu** est confluent.

Preuve :

1. Quelque soit le terme t_1, il ne peut se réduire, en une étape que d'une seule façon, ce qui nous donne $t_2 = t_3$. Dans notre système tous les termes peuvent être réduits, il existe donc t_4 tel que $t_2 = t_3 \to t_4$, et on peut conclure.
2. La preuve est exactement identique à celle du point 1.
3. Quelque soit t_1, t_2 et t_3 tels que $t_1 \to^* t_2$ et $t_1 \to^* t_3$, il est évident que l'une des réduction est un préfixe au sens large (c'est-à-dire "préfixe ou égal") de l'autre. Supposons que $t_1 \to^* t_2$ soit préfixe de $t_1 \to^* t_3$ (l'autre cas procède de la même façon), on peut alors clore le diagramme en effectuant les étapes manquantes dans la première dérivation $t_2 \to^* t_3$ et en ne faisant rien dans la deuxième $t_3 \to^* t_3$.

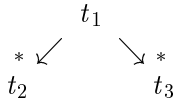
■

En fait, il était inutile de donner une preuve des points 2. et 3., car ce sont des conséquences du point 1. comme l'exprime la proposition suivante.

Proposition 1.2.5 (Héritage du diamant)

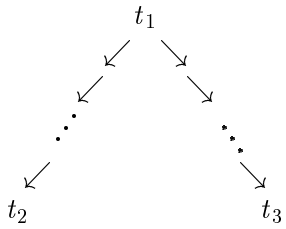
Si un système vérifie la propriété du diamant alors il est confluent.

Preuve : De la même façon, il nous faut ici fermer le diagramme suivant en effectuant, de chaque côté, zéro, une ou plusieurs réductions :

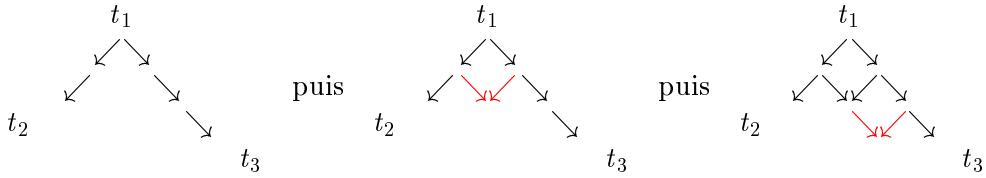


L'étoile signifie, rappelons-nous, qu'il y a eu zéro, une ou plusieurs étapes de réduction. Si on a zéro réductions d'un côté, alors le résultat est facile à obtenir, car il suffit de reproduire les réduction faites par l'autre côté. Sinon, on en a potentiellement plusieurs. On peut redessiner le diagramme ci-dessus

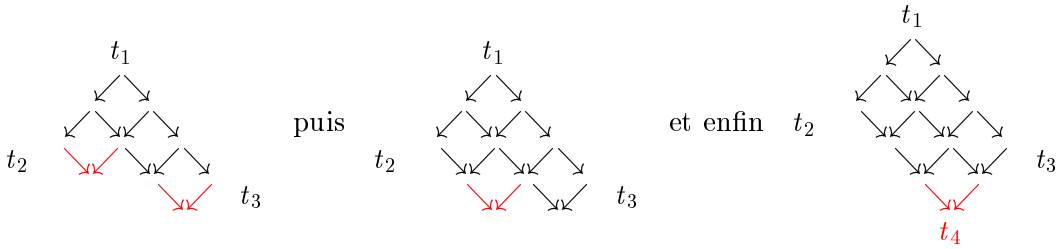
pour mieux faire apparaître ces réductions :



Il suffit de fermer le diagramme en appliquant le diamant partout, en partant du sommet pour descendre jusqu'au terme cherché. Supposons par exemple qu'on ait deux réductions à gauche et trois à droite, voici les différentes étapes nécessaires :



On peut poursuivre jusqu'à rejoindre les deux bords :



■

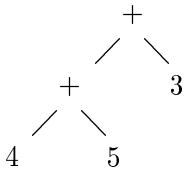
Il ne sert à rien d'essayer de vérifier la confluence de **Riz** puisque le contre-exemple donné précédemment suffit pour affirmer que ce système ne vérifie aucune des trois propriétés de confluence. On passe tout de suite au système **Addition** qui va nous permettre d'introduire une nouvelle notion.

Proposition 1.2.6 (Confluences de Addition)

Le système **Addition** vérifie la propriété du diamant.

Preuve : Prenons trois termes t_1, t_2 et t_3 tels que $t_1 \to t_2$ et $t_1 \to t_3$. Comment garantir qu'il existe un terme t_4 tel que $t_2 \to t_4$ et $t_3 \to t_4$?

On ne sait rien de t_1, ni de t_2, ni de t_3, mais on a une bonne idée de la façon dont les termes sont construits et réduits. Les termes de notre système sont composés de symboles $+$ sous forme d'un arbre, avec des entiers aux feuilles (aux extrémités) de cet arbre. Par exemple, le terme $(4+5)+3$ comporte trois feuilles 4, 5 et 3 :



Les règles de réductions nous imposent de réduire uniquement les $+$ qui ont pour sous-termes deux entiers, ce qui restreint fortement les possibilités de réduction.

Regardons la forme la plus générale que peut avoir le terme t_1. Si on suppose que $t_2 \neq t_3$ (le cas $t_2 = t_3$ étant facilement prouvé), il doit y avoir deux expressions de la forme $(n+m)$ distinctes. Le terme t_1 s'écrit : $...(n_a + m_a)...(n_b + m_b)...$ où les "..." représentent le reste du terme. Si on suppose

que t_2 est obtenu par réduction de $(n_a + m_a)$, cela nous donne $t_2 = ...n_a \oplus m_a...(n_b + m_b)...$ et $t_3 = ...(n_a + m_a)...n_b \oplus m_b...$ comme termes réduits. Pour joindre les deux réductions, il suffit, dans chaque cas, de réduire l'autre redex, cela nous donne $t_4 = ...n_1 \oplus m_1...n_2 \oplus m_2...$ comme terme final.

Par exemple, sur le terme très simple $(1+2)+(3+4)$ on peut souligner les deux redex : $(\underline{1+2}) + (\underline{3+4})$. On réduit de la façon suivante : $(\underline{1+2}) + (\underline{3+4}) \to 3 + (\underline{3+4})$ et $(\underline{1+2}) + (\underline{3+4}) \to (\underline{1+2}) + 7$, et on clôt en réduisant l'autre redex : $3 + (\underline{3+4}) \to 3 + 7$ et $(\underline{1+2}) + 7 \to 3 + 7$.

Pour effectuer la preuve, on va utiliser ce fait qu'il est toujours possible de réduire *l'autre* redex. Cette possibilité va apparaître encore plus évidemment avec l'exemple suivant. Voici un terme dont on a souligné tous les redex :

$$(((\underline{1+2}) + 3) + (((\underline{4+5}) + 6) + (7 + (\underline{8+9})))) + (\underline{10+11})$$

Il y a quelque chose que l'on remarque, c'est le fait que les soulignés ne se chevauchent jamais. Avec le vocabulaire des systèmes de réécriture, on dit que les redex sont *disjoints*. Le fait que tous les redex des termes soient nécessairement disjoints suffit à prouver que le système vérifie la propriété du diamant. ■

Nous avons besoin d'outils supplémentaires pour parler de nos systèmes de réécriture. En particulier, on souhaite pouvoir parler des interactions des redex entre eux et de la façon dont les règles de réécriture font interférer ces redex.

Définition 1.2.7 (Redex disjoints, redex superposés)

- On dit que deux redex sont *superposés* si l'un deux est un sous-terme de l'autre.
- On dit que deux redex sont *disjoints* s'ils ne sont par superposés.

Exemple 1.2.8

– Dans le système **Addition**, tous les redex sont disjoints (voir plus haut).

– Prenons un système de réécriture dont les termes sont donnés par la grammaire suivante :

$$t \ ::= \ a \mid b \mid c \mid f(t) \mid g(t,t)$$

et dont les règles sont :

$$\begin{array}{lcl} g(b,t) & \to & g(t,b) \\ f(a) & \to & c \\ a & \to & b \end{array}$$

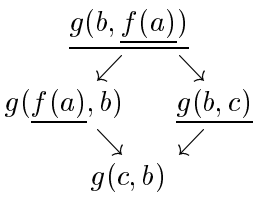
où t est une méta-variable. Dans le terme $g(a, g(a, f(b)))$ les redex sont disjoints : $g(\underline{a}, g(\underline{a}, f(b)))$. Par contre, dans le terme $g(b, f(a))$, les trois redex sont superposés : $\underline{g(b, \underline{f(\underline{a})})}$. La distinction entre redex disjoints et redex superposés est importante. En effet, lorsqu'on regarde des problèmes de confluence, leurs comportements sont différents :

∗ Les redex disjoints donnent des arbres de dérivation qui se ferment comme dans la propriété du diamant. Pour notre exemple $g(\underline{a}, g(\underline{a}, f(b)))$ on peut facilement établir le diagramme suivant :

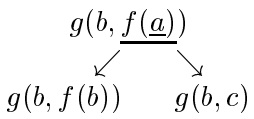
$$\begin{array}{ccc} & g(\underline{a}, g(\underline{a}, f(b))) & \\ \swarrow & & \searrow \\ g(b, g(\underline{a}, f(b))) & & g(\underline{a}, g(b, f(b))) \\ \searrow & & \swarrow \\ & g(b, g(b, f(b))) & \end{array}$$

∗ Tandis que les redex superposés se comportent de manière plus aléatoire. Pour notre exemple $\underline{g(b, \underline{f(\underline{a})})}$, le grand et le moyen redex se ferment comme dans la propriété du

diamant :



Alors que le moyen et le petit ne se ferment pas du tout :



Définition 1.2.9 (Paire critique)

Une *paire critique* est une paire de règles de réécriture telle qu'il existe un terme (avec méta-variable) contenant deux redex superposés (un de chacune des règles), qui, une fois réduits, donnent deux termes différents.

Exemple 1.2.10

- Dans le système de l'exemple précédent, il y a quatre paires critiques. Pour pouvoir les nommer plus facilement, on indice les règles de la façon suivante :

$$\begin{array}{lll} g(b,t) & \to_g & g(t,b) \\ f(a) & \to_f & c \\ a & \to_a & b \end{array}$$

  Voici les paires critiques et le méta-terme qui contient les deux redex :

  * (f, a) : le terme est $\underline{f(\underline{a})}$.
  * (g, a) : le terme est $\underline{g(b, \underline{a})}$.
  * (g, f) : le terme est $\underline{g(b, \underline{f(a)})}$.
  * (g, g) : le terme est $\underline{g(b, \underline{g(b, t)})}$, où t est une méta-variable.

- Dans le système **Riz**, les règles (ajouterRiz) et (ajouterEau) forment une paire critique, car le terme **casserole(rien, rien)** est un redex pour ces deux règles (les deux redex superposés sont deux fois le même). En revanche, les règles (cuire) et (ajouterCrème) ne forment pas une paire critique, car il n'existe pas de terme pouvant être réduit par ces deux règles, et dont les redex seraient superposés.
- Soit un système dont les termes sont les constantes a et b, les symboles de fonction sont f et g et les règles de réduction sont :

$$\begin{array}{lll} g(b) & \to_g & a \\ f(a) & \to_f & b \end{array}$$

  Ce système ne comporte aucune paire critique.

Pour terminer, voici une proposition qui établit un lien entre confluence et confluence locale.

Proposition 1.2.11 (Confluence et confluence locale)

- Un système confluent est localement confluent.
- Un système localement confluent n'est pas forcément confluent.
- Un système localement confluent et fortement normalisant est confluent.

Preuve :

- La confluence locale est un cas particulier de la confluence, du fait que $\rightarrow$ est un cas particulier de $\rightarrow^*$. En effet, si on peut trouver un réduit commun de tous termes issus en zéro, une ou plusieurs étapes d'un terme, alors on peut en trouver un pour deux termes issus en exactement une étape de ce terme.
- Voici un exemple de système de réécriture localement confluent mais non-confluent. Les termes sont uniquement des constantes : a, b, c et d. Les règles sont les suivantes :

$$\begin{array}{lcl} b & \rightarrow & a \\ b & \rightarrow & c \\ c & \rightarrow & b \\ c & \rightarrow & d \end{array}$$

  Il est localement confluent : il ne peut pas y avoir de réduction de a ou de d, on regarde les réductions possible de b et de c. On a $b \rightarrow a$ et $b \rightarrow c$, pour fermer le diagramme, on fait $c \rightarrow b \rightarrow a$:

$$\begin{array}{ccccc} & & b & & \\ & \swarrow & & \searrow & \\ a & \leftarrow & b & \leftarrow & c \end{array}$$

  On procède symétriquement pour c. Cependant, ce système n'est pas confluent, voici deux réductions de b qu'on ne peut pas joindre : $b \rightarrow^* a$ et $b \rightarrow^* d$.
- C'est le lemme de Newman.

■

1.2.2 Normalisation

Intuition

La question de la normalisation, aussi appelée terminaison, concerne la fin éventuelle des réductions. Lorsqu'on effectue une addition, on ne se pose pas la question de savoir si on a la certitude que le calcul va s'arrêter ou si cela dépend de l'ordre choisi pour l'effectuer. En effet, comme pour la confluence on répond positivement, de façon naturelle, à ces questions. Reprenons notre exemple tiré de **Addition** et regardons si son calcul se termine :

$$(5+6)+(2+1) \quad \rightarrow \quad 11+(2+1) \quad \rightarrow \quad 11+3 \quad \rightarrow \quad 14$$

En trois étapes, le calcul se *termine*. On dit qu'un calcul est terminé lorsqu'aucune règle de réduction ne peut plus s'appliquer. Le terme obtenu s'appelle une *forme normale*, c'est-à-dire un terme qui ne peut plus être réduit davantage. Dans notre exemple, comme pour tous les termes du système **Addition**, le calcul se termine quelque soit la stratégie employée. Au lieu de dire qu'il se termine, on dit souvent qu'il *normalise*, c'est-à-dire qu'il rend *normal*, mettant en *forme normale*. Un système qui normalise tous ses termes quelque soit la stratégie est appelé *fortement normalisant* ; on dit aussi qu'il *normalise fortement*. Un terme dont il n'existe aucune réduction infinie est dit *fortement normalisable* ; par abus de langage on dira aussi qu'il est fortement normalisant ou qu'il normalise fortement. Comme on le verra ci-dessous, le système **Addition** est fortement normalisant.

Le système **Riz** est lui-aussi fortement normalisant, et le fait qu'on ait pu écrire l'arbre complet des dérivations (voir paragraphe 1.1.2) en constitue une preuve. Par contre, le système **Feu** n'est pas fortement normalisant. Si on réduit le terme **feuvert**, par exemple, on constate que la réduction ne s'arrête jamais :

feuvert $\rightarrow$ **feuorange** $\rightarrow$ **feurouge** $\rightarrow$ **feuvert** $\rightarrow$ **feuorange** $\rightarrow$ **feurouge** $\rightarrow$ **feuvert** $\rightarrow$ ...

Cela vient du fait que tous les termes peuvent se réduire : il n'y a pas de formes normales dans ce système. Si on veut avoir une forme normale, on peut ajouter un terme à notre système en s'assurant qu'il est irréductible. Par exemple, lorsque le feu tombe en panne, il se met dans un état final : l'orange clignotant. On ajoute à nos termes la constante **feuorangeclignotant** et, le feu pouvant tomber en panne à partir de n'importe quel état, on ajoute les trois règles suivantes :

$$\begin{array}{lcl} \textbf{feuvert} & \to & \textbf{feuorangeclignotant} \\ \textbf{feuorange} & \to & \textbf{feuorangeclignotant} \\ \textbf{feurouge} & \to & \textbf{feuorangeclignotant} \end{array}$$

On peut maintenant avoir de nouvelles réductions pour nos termes. Par exemple, voici trois dérivations possibles du terme **feuvert** :

$$\textbf{feuvert} \to \textbf{feuorange} \to \textbf{feurouge} \to \textbf{feuvert} \to \textbf{feuorangeclignotant}$$
$$\textbf{feuvert} \to \textbf{feuorange} \to \textbf{feuorangeclignotant}$$
$$\textbf{feuvert} \to \textbf{feuorangeclignotant}$$

Le terme **feuorangeclignotant** ne pouvant pas être réduit, il est une forme normale (la seule de ce système). Tous les termes peuvent se réduire vers celui-ci, ce qui fait que toute réduction a la possibilité de s'arrêter sur ce terme : on peut, pour tout terme, trouver une dérivation qui termine. Cependant, notre système n'est toujours pas fortement normalisant, car on peut toujours effectuer la dérivation infinie précitée. On appelle *faiblement normalisant* un système vérifiant cette propriété.

En informatique, cette question est importante car si on souhaite parfois que les programmes ne se terminent pas (comme les systèmes d'exploitation, par exemple), en revanche, lorsqu'on demande à l'ordinateur d'effectuer un calcul on espère bien qu'il s'arrête un jour de calculer pour donner le résultat.

Définition

Voici les définitions formelles des deux propriétés de normalisation.

Définition 1.2.12 (Normalisation forte)

- On dit qu'un terme *normalise fortement* si il n'est pas possible d'engendrer une réduction infinie à partir de lui.
- On dit qu'un système de réécriture est *fortement normalisant* si tous ses termes normalisent fortement. Autrement dit, si chaque terme a un arbre de dérivation fini. Autrement dit, si pour tout terme toute réduction se termine.

Définition 1.2.13 (Normalisation faible)

- On dit qu'un terme *normalise faiblement* si il existe une réduction de ce terme qui termine.
- On dit qu'un système de réécriture est *faiblement normalisant* si tous ses termes normalisent faiblement. Autrement dit, si chaque terme possède, dans son arbre de dérivation, une branche finie.

Ces deux définitions sont valables même si le système de réécriture comporte des équivalences car celles-ci ne comptent pas comme des étapes de réduction. La notion de terminaison s'étend directement à la réécriture modulo à ceci près qu'un terme est en forme normale si aucune règle de réduction ne peut réduire *aucun* des termes équivalents à lui.

Notation 1.2.14

Pour un système de réécriture $\mathcal{R}$, on notera $\mathcal{SN}_\mathcal{R}$ l'ensemble de ses termes qui sont fortement normalisants[4]. Lorsqu'il n'y aura pas de confusion possible, on omettra souvent d'indiquer à quel système on fait référence, notant seulement $\mathcal{SN}$.

Preuves et exemples

Prouvons la normalisation forte de **Riz**. Cela nous permet de débuter nos démonstrations de normalisation avec une preuve directe par cas, ce genre de preuves étant en général assez simples — bien que parfois très longues — car son principe consiste à étudier chaque cas, c'est-à-dire chaque terme.

Proposition 1.2.15 (Normalisation forte de Riz)

Le système **Riz** est fortement normalisant.

Preuve : Comme l'ensemble des termes est très petit on peut procéder par cas, c'est-à-dire étudier chaque terme afin d'en établir la normalisation forte. Pour cela, on va commencer par les termes en forme normale, puis on va "remonter", c'est-à-dire parcourir dans le sens inverse, toutes les réductions qui peuvent mener à ces termes.

- Les termes **accompagnement**, **casserole(riz cuit, eau)** et **casserole(riz, crème)** ne peuvent pas être réduits. Ce sont des formes normales.
- Le terme **casserole(riz cuit, crème)** ne peut se réduire que vers **accompagnement** qui est une forme normale. Toute réduction partant de lui (il n'y en a qu'une) termine, il normalise donc fortement.
- Le terme **casserole(riz cuit, rien)** peut se réduire, soit vers **casserole(riz cuit, eau)** qui est une forme normale, soit vers **casserole(riz cuit, crème)**, qui, comme on vient de le voir, normalise fortement. Quelque soit le choix de réduction qu'on fasse, on s'arrête forcément en deux étapes au maximum. Ce terme normalise fortement.
- Le terme **casserole(riz, eau)** ne peut se réduire que vers **casserole(riz cuit, rien)** qui, comme on vient de le voir, normalise fortement. Avec le même argument que précédemment, on conclut à sa normalisation forte.
- On procède de la même façon, successivement, pour les termes **casserole(riz, rien)** et **casserole(rien, eau)**, et enfin **casserole(rien, rien)**.

■

Dans la preuve ci-dessus, on a souvent utilisé un argument sur la normalisation forte des réduits d'un terme. Voici une proposition qui établit plus abstraitement cet argument afin qu'on puisse le réutiliser dans nos preuves ultérieures.

Proposition 1.2.16 (Normalisation forte et réduits)

1. Si tous les réduits d'un terme normalisent fortement, alors ce terme normalise lui-même fortement.
2. Réciproquement, il suffit qu'un seul des réduits d'un terme ait une dérivation infinie pour que ce terme ait une dérivation infinie, *i.e.* ne normalise pas.

Preuve : Soit t le terme étudié, et $t_1, ..., t_n$ ses réduits.

1. On raisonne par l'absurde. On a par hypothèse que $t_1, ..., t_n$ sont des termes fortement normalisants. Supposons que t ne normalise pas fortement, cela signifie qu'il existe une réduction infinie partant de t. Lors de la première réduction de t, on obtient l'un des $t_1, ..., t_n$, admettons que cela soit t_i. Puisque t engendre une réduction infinie passant par t_i, cela implique que t_i ait, lui-aussi,

[4]SN pour "Strong Normalization", normalisation forte en anglais.

une dérivation infinie. Cela contredit l'hypothèse de normalisation forte de t_i, prouvant notre point.

2. Si t_i admet une dérivation infinie $t_i \to ...$, alors il existe une dérivation infinie de $t : t \to t_i \to ...$

■

Passons maintenant au système **Addition** dont la preuve de normalisation forte est plus complexe. En effet, l'ensemble des termes étant infini, on ne peut pas raisonner par cas sur chacun d'eux : on utilise une autre technique de preuve, dont l'idée est la suivante. À chaque terme on fait correspondre un entier naturel, de telle sorte qu'à chaque fois qu'on réduit un terme, l'entier correspondant diminue. Ainsi, on ne peut pas avoir de réduction infinie d'un terme, sinon cela signifierait qu'on peut faire décroître les entiers naturels infiniment. Or ceci est faux puisque les entiers naturels n sont tous plus grands ou égaux à zéro et qu'il n'y en a qu'un nombre fini entre n et 0. Cette propriété s'appelle la *bonne fondation* de l'ordre "$<$" pour les entiers. De façon générale, la propriété de bonne fondation d'un ordre pour un ensemble dit qu'il n'existe pas de suite infinie d'éléments de cet ensemble qui soient décroissant dans l'ordre considéré.

Regardons concrètement ce que cela donne. On prend comme entier la taille du terme, c'est-à-dire le nombre de constantes (les entiers) et de symboles de fonction (+) qu'on utilise pour l'écrire. Le tableau suivant donne quelques exemples de correspondances ; on remarque que le même entier peut correspondre à plusieurs termes différents.

terme	:	entier correspondant
$1+4$	:	3
2	:	1
$(7+12)+3$	:	5
$(4+6)+(1+2)$	:	7
$5+7$	:	3
$6+8$	:	3

Regardons à présent la réduction d'un terme et les entiers correspondants.

Terme :	$(1+2)+(3+4)$	$\to$	$3+(3+4)$	$\to$	$3+7$	$\to$	10
Entier :	7		5		3		1

On remarque que l'entier correspondant décroît strictement à chaque réduction : $7 > 5 > 3 > 1$. Le même phénomène a lieu pour chaque terme, ce qui nous permet de conclure à la normalisation forte.

Plus formellement, on appelle *poids* (ou *mesure*) la valeur associée à chaque terme. Voici la définition *inductive* (c'est-à-dire qui suit le principe de construction inductif des termes) du poids des termes du système **Addition** :

- si le terme est un entier, alors son poids est 1,
- sinon, le terme est de la forme $t_1 + t_2$: soit n_1 le poids de t_1 et n_2 le poids de t_2, alors le poids de $t_1 + t_2$ est l'entier correspondant à la somme de n_1 et n_2 à laquelle on ajoute 1 pour le symbole $+$.

Une telle définition est correcte, même si elle fait référence à elle-même, car elle ne le fait alors que pour des termes strictement plus petits. On donne parfois un nom à la mesure, et on note la mesure d'un terme comme une fonction : par exemple, si on choisit h comme nom pour le poids, on note $h(3+7)$ le poids du terme $3+7$. Puisque celui-ci vaut 3, on a $h(3+7) = 3$. Voici la définition formelle de la mesure h :

- si le terme t est un entier, alors $h(t) = 1$,
- sinon, t est de la forme $t = t_1 + t_2$ et on a $h(t) = h(t_1) + h(t_2) + 1$.

On établit un lemme qui met en évidence le fait que chaque réduction fait décroître strictement le poids.

Lemme 1.2.17

Pour tous termes t_1 et t_2 tels que $t_1 \to t_2$, on a $h(t_1) > h(t_2)$.

Preuve : On prouve ce lemme dans le cadre du système **Addition**.
Lorsqu'on contracte un redex, on transforme un sous-terme de la forme $x + y$, où x et y sont des entiers, en un nouvel entier z égal à la somme de x et de y. La taille de ce sous-terme était de 3, et elle passe à 1. On a donc $h(t_1) = n + 3$, où n est le poids du reste du terme, et $h(t_2) = n + 1$. Cela nous permet de conclure que $h(t_1) > h(t_2)$. ■

On peut à présent donner la proposition de normalisation forte d'**Addition** :

Proposition 1.2.18

Le système **Addition** est fortement normalisant.

Preuve : On raisonne par l'absurde. Supposons qu'il existe un terme t admettant une dérivation infinie $t \to t_1 \to t_2 \to ... \to t_p \to ....$ Par le lemme 1.2.17, on obtient une chaîne infinie d'entiers naturels décroissants $h(t) > h(t_1) > h(t_2) > ... > h(t_p) > ...$, ce qui est impossible. Le système **Addition** est donc fortement normalisant. ■

Cette technique, qui s'appelle l'*interprétation polynomiale* peut s'adapter avec des poids qui ne sont pas des entiers. La seule contrainte est d'avoir un *ordre bien fondé*. De ce fait, on ne peut pas avoir de réduction infinie, ce qui voudrait dire qu'on a une suite infinie décroissante pour l'ordre. Si un système comporte des équivalences, celles-ci doivent laisser le poids inchangé ; ainsi, on peut les appliquer un nombre quelconque de fois sans mettre en péril la preuve de terminaison.

Une autre méthode que l'on peut employer pour montrer la normalisation forte d'un système est la simulation, dont l'idée est la suivante. On souhaite montrer la normalisation forte d'un système $\mathcal{S}_1$. On choisit un système $\mathcal{S}_2$ qui est déjà fortement normalisant. On le choisit de telle sorte qu'on puisse établir une traduction $T()$ des termes de $\mathcal{S}_1$ vers les termes de $\mathcal{S}_2$. L'intérêt de cette traduction est de pouvoir rapprocher $\mathcal{S}_1$ et $\mathcal{S}_2$ pour pouvoir faire rejaillir la normalisation forte de $\mathcal{S}_2$ sur $\mathcal{S}_1$. Ce rapprochement se fait à l'aide d'une simulation, c'est-à-dire qu'on essaie de simuler les réductions de $\mathcal{S}_1$ par celles de $\mathcal{S}_2$. Plus formellement, si $t_1 \to t_2$ dans $\mathcal{S}_1$ alors on va essayer d'obtenir $T(t_1) \to^+ T(t_2)$ dans $\mathcal{S}_2$. On pourra alors facilement prouver la normalisation forte de $\mathcal{S}_1$, comme on le voit ci-dessous.

Regardons sur un exemple comment cela fonctionne. On introduit un nouveau système de réécriture, volontairement très proche du système **Addition**, dont on va très simplement pouvoir établir une traduction et une simulation dans celui-ci. On appelle ce système **Multiplication**. Voici la grammaire de ses termes :

$$t ::= \underline{\mathrm{n}} \mid t \times t$$

Voici la règle de réduction :

$$\underline{\mathrm{n}} \times \underline{\mathrm{m}} \quad \to \quad \underline{\mathrm{n} \otimes \mathrm{m}}$$

Où $\otimes$ est la multiplication sémantique. On traduit les termes de la façon suivante :

- les entiers seront traduits en 0 : pour tout x entier, sa traduction $T(x)$ vaut 0 (on note cela $T(x) = 0$),
- les multiplications seront traduites en additions : $T(t_1 \times t_2) = T(t_1) + T(t_2)$ (on prend d'abord la traduction de t_1 et t_2, puis on met un + entre les deux).

Voici un tableau montrant quelques exemples de traduction :

Terme de **Multiplication**	Traduction dans **Addition**
5	0
4×3	$0 + 0$
$(2 \times 2) \times 3$	$(0 + 0) + 0$
5×10	$0 + 0$

On remarque que deux termes peuvent avoir la même traduction (on dit que la fonction de traduction T n'est pas *injective*). Cela importe peu, ce qui compte, c'est de pouvoir effectuer une simulation des réductions de **Multiplication** par les réductions d'**Addition**. On établit le lemme de simulation suivant.

Lemme 1.2.19

Pour tous termes t_1 et t_2 de **Multiplication** tels que $t_1 \to t_2$, il existe une réduction, dans **Addition**, de $T(t_1)$ vers $T(t_2)$ en au moins une étape. Autrement dit, il est possible de fermer le diagramme suivant :

$$\begin{array}{ccc} t_1 & \to_{\textbf{Multiplication}} & t_2 \\ {\scriptstyle T}\downarrow & & \downarrow{\scriptstyle T} \\ T(t_1) & \to^{+}_{\textbf{Addition}} & T(t_2) \end{array}$$

Preuve : Si $t_1 \to t_2$ cela signifie qu'il existe un contexte $C[\,]$ et un sous-terme $x \times y$ (où x et y sont des entiers) tels que $t_1 = C[x \times y]$ et $t_2 = C[x \otimes y]$. La traduction du contexte C est un contexte C' dans lequel les entiers sont remplacés par des 0 et les $\times$ par des $+$. On a alors $T(t_1) = C'[0 + 0]$ et $T(t_2) = C'[0]$, et il y a bien une réduction du premier vers le second, en exactement une étape : il suffit de réduire ce redex $0 + 0$. ■

Ce lemme est suffisant pour prouver la normalisation forte de **Multiplication**.

Proposition 1.2.20

Le système **Multiplication** est fortement normalisant.

Preuve : On raisonne par l'absurde. Supposons qu'il existe un terme de **Multiplication**, t, qui admette une réduction infinie $t \to t_1 \to t_2 \to \ldots$ dans ce système. Grâce au lemme 1.2.19, on peut construire une réduction partant de $T(t_1)$ de la façon suivante :

$$\begin{array}{ccccccccc} t & \to_{\textbf{Multiplication}} & t_1 & \to_{\textbf{Multiplication}} & t_2 & \to_{\textbf{Multiplication}} & \cdots \\ {\scriptstyle T}\downarrow & & {\scriptstyle T}\downarrow & & {\scriptstyle T}\downarrow & & \\ T(t_1) & \to_{\textbf{Addition}} & T(t_2) & \to_{\textbf{Addition}} & T(t_3) & \to_{\textbf{Addition}} & \cdots \end{array}$$

Cette réduction est infinie elle aussi, et elle contredit le fait que le système **Addition** est fortement normalisant. Le système **Multiplication** est donc fortement normalisant lui aussi. ■

Si un système comporte des équivalences, il faut que les traductions des termes équivalents soient elles-mêmes des termes équivalents dans le système cible. Nous utiliserons de nombreuses fois cette technique de simulation dans la suite de notre travail.

Il existe bien d'autres techniques de preuve de normalisation forte, nous en présenterons d'autres au fur et à mesure de nos besoins.

Union de systèmes de réécriture et terminaison

Il arrive souvent qu'on souhaite enrichir un système de réécriture avec de nouvelles règles. On peut se demander si la normalisation forte de l'ancien système peut s'étendre au nouveau système. Regardons ce qui se passe lorsque nous faisons l'union de deux systèmes de réécriture fortement normalisants. Supposons qu'on ait les deux systèmes de réécriture suivants, ayant tous les deux comme termes les constantes a et b :

- $\mathcal{R}_1$ comporte la règle de réduction $a \to b$, et
- $\mathcal{R}_2$ comporte la règle de réduction $b \to a$.

Il est évident que ces deux systèmes terminent (en fait, la seule réduction non vide est de longueur 1 : $a \to b$ et $b \to a$ respectivement). En revanche, l'union de ces deux systèmes nous donne comme règles de réduction :

$$\begin{array}{lcl} a & \to & b \\ b & \to & a \end{array}$$

Et il est tout aussi évident que ce système ne termine pas, une réduction infinie pouvant commencer de la façon suivante : $a \to b \to a \to b \to a...$

Ce fait peut paraître anodin, mais il sera la source de problèmes que nous rencontrerons par la suite lorsque nous voudrons étendre des systèmes de réécriture, comme le λ-calcul, avec de nouvelles règles de réduction (voir section 2.2).

Chapitre 2

λ-calculs et substitutions explicites

Dans ce chapitre, nous introduisons le λ-calcul en donnant sa définition, et ses propriétés. Nous étudions ensuite les λ-calculs avec substitutions explicites : nous en donnons le principe fondateur, puis les principales propriétés, et enfin nous en faisons un tour d'horizon. Le lecteur qui sait déjà que les règles de composition de substitutions conduisent en général à la perte des propriétés de normalisation peut passer directement à la section 2.2.4. Celui qui connaît, en plus, les différences entre $\lambda\sigma$, $\lambda\upsilon$, λ_{ws} et $\lambda\mathbf{x}$ peut passer au chapitre suivant.

2.1 Le λ-calcul

2.1.1 Intuition : des fonctions mathématiques aux λ-termes

Les fonctions en mathématiques

Pour bien comprendre le λ-calcul, nous partons de la notion de fonction telle qu'elle est introduite dans l'enseignement secondaire. On y apprend qu'une *fonction* est une *relation* entre des objets d'un *ensemble de définition* et ceux d'un *ensemble image*.

Exemple 2.1.1

> On peut définir une fonction sur les entiers naturels qui fait correspondre à chaque entier celui qui lui est supérieur de 1. L'ensemble de définition est $\mathbb{N}$ (l'ensemble des entiers naturels), on peut donner sa définition de la façon suivante :
>
> $$f(x) \quad = \quad x + 1$$
>
> La fonction f ainsi définie met en relation certains entiers entre eux. Par exemple, 3 et 4 sont en relation car $f(3) = 3 + 1 = 4$. Pour définir f, nous avons utilisé une variable x qui s'appelle le paramètre de la fonction.

La plupart du temps, on se sert des fonctions non pas comme des relations, mais comme un moyen d'effectuer un calcul. La fonction f définie ci-dessus nous donne le "mode d'emploi" pour calculer, pour chaque entier, l'entier suivant dans l'ordre croissant. Si on considère la définition d'une fonction comme une définition de calcul, on préfère alors orienter l'égalité de cette définition. Plus que d'une égalité $f(3) = 4$, on est en présence d'un calcul : $f(3)$ a pour résultat 4. On aurait envie de remplacer le symbole $=$ par une flèche $\rightarrow$.

En fait, il existe déjà une notation de définition de fonction qui comporte une flèche. Reprenons la définition de f avec cette notation.

Exemple 2.1.2

> $$f : \left\{ \begin{array}{l} \mathbb{N} \rightarrow \mathbb{N} \\ x \mapsto x + 1 \end{array} \right.$$

> On lit cette définition : "soit f la fonction définie sur les entiers naturels qui associe à tout entier x l'entier $x + 1$".

On appelle l'indication $\mathbb{N} \to \mathbb{N}$ le *type* de la fonction, et nous en parlerons plus longuement dans le chapitre 3. La deuxième ligne de la définition utilise le symbole $\mapsto$ qui fait bien apparaître l'idée d'un calcul. On dit que les fonctions effectuent des calculs en prenant en *paramètre* des valeurs et en *retournant* (ou en *renvoyant*) un résultat. Par exemple, si on donne la valeur 5 comme paramètre à la fonction f, le résultat renvoyé sera 6.

Dans l'expression $f(5)$, on dit que 5 est le paramètre *effectif* de la fonction f, c'est-à-dire le paramètre qui va permettre d'effectuer le calcul. Cette appellation est à opposer à celle de paramètre *formel*, celui qui permet de formaliser la fonction : x dans notre exemple. On appelle *corps* de la fonction la partie de la définition a droite du symbole $\mapsto$: $x + 1$ dans notre exemple.

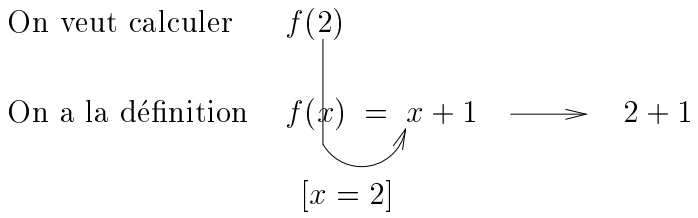
Avant de passer à la suite, on donne la définition de deux autres fonctions qui nous serviront d'exemples. Voici les définitions des fonctions h et k.

$$h : \begin{cases} \mathbb{N} \to \mathbb{N} \\ x \mapsto x + x \end{cases} \qquad\qquad k : \begin{cases} \mathbb{N} \to \mathbb{N} \\ x \mapsto 2 \end{cases}$$

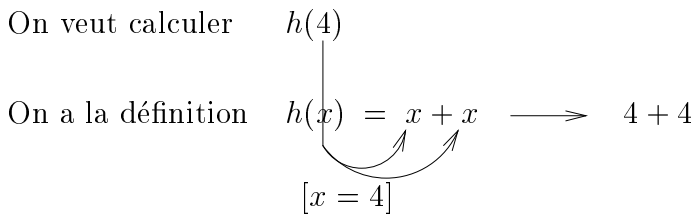
Calcul d'un résultat

On s'intéresse de plus près aux mécanismes qui entrent en jeu dans le calcul d'un résultat. Le premier est celui de l'*application*. On dit qu'on applique une fonction à un *argument* (ou paramètre effectif) lorsqu'on fournit à celle-ci une valeur afin de calculer le résultat associé. Par exemple, si on veut calculer le résultat de f pour la valeur 2, on réalise l'application de f à 2 en écrivant $f(2)$.

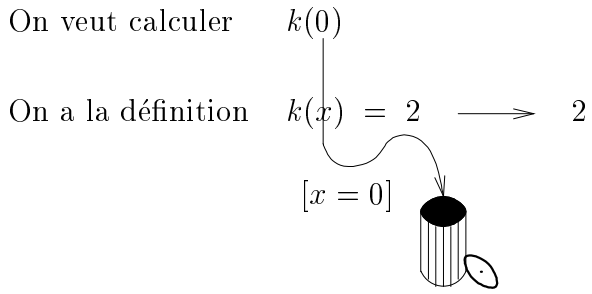
La deuxième étape est la *substitution* : on associe le paramètre effectif au paramètre formel, puis on remplace celui-ci par celui-là. Par exemple, après avoir réalisé l'application $f(2)$, on effectue l'association $[x = 2]$ qu'on utilise pour calculer la valeur du corps de la fonction : on obtient $2 + 1$. Visualisons cela à l'aide d'un dessin :



Si on veut calculer $h(4)$, il faut utiliser deux fois l'association $[x = 4]$:



En revanche, pour calculer $k(0)$, on n'utilise pas le paramètre effectif puisque le paramètre formel n'apparaît pas dans le corps de la fonction :



Enfin, dans le cas des fonctions mathématiques, une troisième étape consiste à calculer le résultat. Par exemple, dans le cas de $f(2)$, après avoir obtenu $2 + 1$, on calcule cette expression et le résultat est 3.

2.1.2 Intuition : les fonctions en informatique

Les fonctions sont apparues très tôt dans les langages de programmation structurés. Elles permettent de regrouper un ensemble d'instructions qui produisent un résultat à partir des données passées en paramètre. Voici quelques exemples de fonctions dans différents langages de programmation.

- En `C`[1] :

```
/* Cette fonction renvoie le double de l'entier pris en parametre */
int double(int n) {
  return 2*n;
}
```

  Dans l'entête de la fonction `double` : `int double(int n)`, on peut voir, entre les parenthèses, que celle-ci prend en paramètre formel un entier (INTeger en anglais) dont le nom est `n`; la mention `int` qui se trouve avant le nom de la fonction indique que le résultat renvoyé par celle-ci est aussi un entier. Dans son corps, l'instruction `return 2*n;` demande que le résultat de l'évaluation de l'expression `2*n` (c'est-à-dire le double de la valeur du paramètre effectif) soit renvoyé comme résultat de la fonction.

- En `Pascal`[2] :

```
{ Cette fonction renvoie le carré de l'entier pris en parametre }
function carre(n : integer) : integer;
begin
  carre := n*n
end;
```

  Dans l'entête de cette fonction : `function carre(n : integer) : integer;`, on peut voir, entre les parenthèses, que celle-ci prend en paramètre un entier dont le nom est `n`; la mention `integer` qui se trouve après le ":" hors des parenthèses indique que le résultat renvoyé par la fonction est aussi un entier. Dans son corps, l'instruction `carre := n*n;` demande que le résultat de l'évaluation de l'expression `n*n` (c'est-à-dire le carré de la valeur du paramètre effectif) soit affecté comme valeur de résultat de la fonction.

- En `Caml`[3] :

```
(* Cette fonction renvoie toujours l'entier 4 *)
let quatre(n) = 4;;
(* Cette fonction renvoie la somme de l'entier pris en parametre et de 3 *)
```

[1] La référence standard est [52].
[2] La plus ancienne référence semble être [71].
[3] Voir [58].

```
let plusTrois(n) = n+3;;
(* On peut aussi la definir de la facon suivante *)
let plusTrois = function n -> n+3;;
```

Ces définitions sont très proches de celles que l'on fait en mathématiques. La première dit que la valeur de la fonction est `4`, quelque soit la valeur de `n` puisque l'on ne s'en sert pas. Les deux suivantes sont deux formulations possibles qui sont offertes aux programmeur. Elle correspondent exactement aux deux définitions possibles en mathématiques dont nous avons parlé dans la section précédente. En outre, il n'est pas nécessaire de préciser le type (ensemble de définition) du paramètre et du résultat, ceux-ci seront déterminés automatiquement.

Lorsque le sémanticien veut étudier des fonctions du programme informatique, il les interprète par des λ-termes, se débarrassant ainsi de la syntaxe spécifique de chaque langage de programmation. Après avoir lu la section suivante, le lecteur n'aura pas de difficultés à trouver les λ-termes interprétant les fonctions présentées dans l'exemple ci-dessus.

2.1.3 Définition

λ-termes et β-réduction

Pour une présentation plus complète de la définition et des propriétés du λ-calcul, le lecteur pourra se référer à [14, 16, 7]. Avant tout, le λ-calcul est un langage qui nous permet d'écrire des fonctions. Voici les indications de traduction de nos fonctions dans le λ-calcul.

- Au lieu d'utiliser le symbole $\mapsto$, nous utiliserons les symboles λ et . qui encadreront le paramètre formel. Par exemple, la fonction f s'écrira $\lambda x.(x+1)$ au lieu de $x \mapsto x+1$, la fonction h s'écrira $\lambda x.(x+x)$ au lieu de $x \mapsto x+x$ et la fonction k s'écrira $\lambda x.2$ au lieu de $x \mapsto 2$. On remarque qu'on a utilisé des parenthèses pour bien identifier le corps de la fonction.
- On notera l'application d'une fonction à un argument en plaçant un espace entre le nom de la fonction et l'argument. Par exemple, l'application $f(2)$ sera notée $f\ 2$. On est libre de mettre des parenthèses, on peut ainsi écrire : $f\ (2)$, $(f\ 2)$ ou encore $(f)\ 2$.

L'intérêt du λ-calcul est qu'il formalise le calcul du résultat de l'application d'une fonction à un argument. Il fournit pour cela une règle de réécriture qui s'appelle β (on parle alors de β-réduction). Mais avant de donner cette règle, on définit les termes du λ-calcul.

Définition 2.1.3 (λ-termes)

Supposons avoir un ensemble infini de variables, $\mathcal{V}$. Les termes du λ-calcul (ou λ-termes) sont construits à l'aide de la grammaire suivante où x est une variable quelconque de $\mathcal{V}$:

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

Un terme est soit une variable x, soit une fonction comportant une variable (le paramètre formel) et un sous-terme (le corps de la fonction) $\lambda x.t$, soit l'application d'un sous-terme (la fonction, à gauche) à un autre sous-terme (le paramètre effectif, à droite) $t\ t$. L'ensemble des λ-termes est noté Λ [4].

Exemple 2.1.4

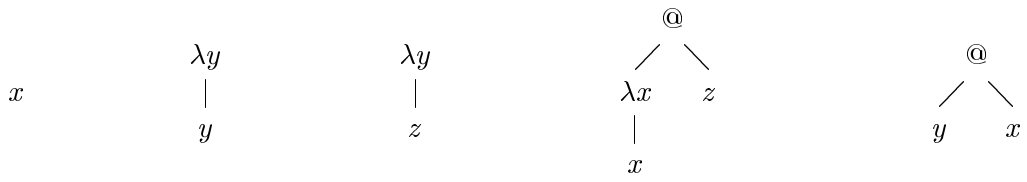
Voici quelques exemples de λ-termes :

- x : toute variable est un terme.
- $\lambda y.y$: c'est la fonction identité ; ce qu'elle prend en paramètre, elle le renvoie comme résultat.

[4] On remarque que les symboles 1, 2 et + que nous avons utilisé plus haut n'apparaissent pas dans cette grammaire. On ne peut pas écrire la fonction $\lambda x.(x+1)$. Cependant, les entiers naturels ainsi que l'addition (et bien plus encore) peuvent être définis dans le λ-calcul en utilisant la grammaire ci-dessus (pour plus de détails, consulter les références proposées au début de cette section).

- $\lambda y.z$: cette fonction n'utilise pas son paramètre.
- $(\lambda x.x)\ z$: c'est l'application de la fonction identité à la variable z.
- $y\ x$: c'est l'application de y à x (y peut devenir une fonction, comme on va le voir ci-dessous).

Comme dans l'exemple de l'addition, on peut représenter nos λ-termes par des arbres dont les nœuds sont soit des λx soit des applications (notées @) et dont les feuilles sont des variables. Voici de gauche à droite les arbres correspondant aux termes présentés ci-dessus.



Dans le λ-calcul, on dit que les fonctions sont des "citoyens de première classe" ; cela signifie que les fonctions sont des valeurs qu'on peut, par exemple, passer comme paramètre à d'autres fonctions. Cette possibilité est beaucoup plus utilisée en informatique qu'en mathématiques[5]. Par exemple, le terme $(\lambda x.x)(\lambda y.z)$ est un λ-terme : c'est l'application de la fonction $\lambda x.x$ avec comme paramètre la fonction $\lambda y.z$.

Voici la règle de réduction β, dans laquelle t et u sont des méta-variables de terme, et x est une méta-variable de variable :

$$(\lambda x.t)\ u \quad \rightarrow \quad t\{x \leftarrow u\}$$

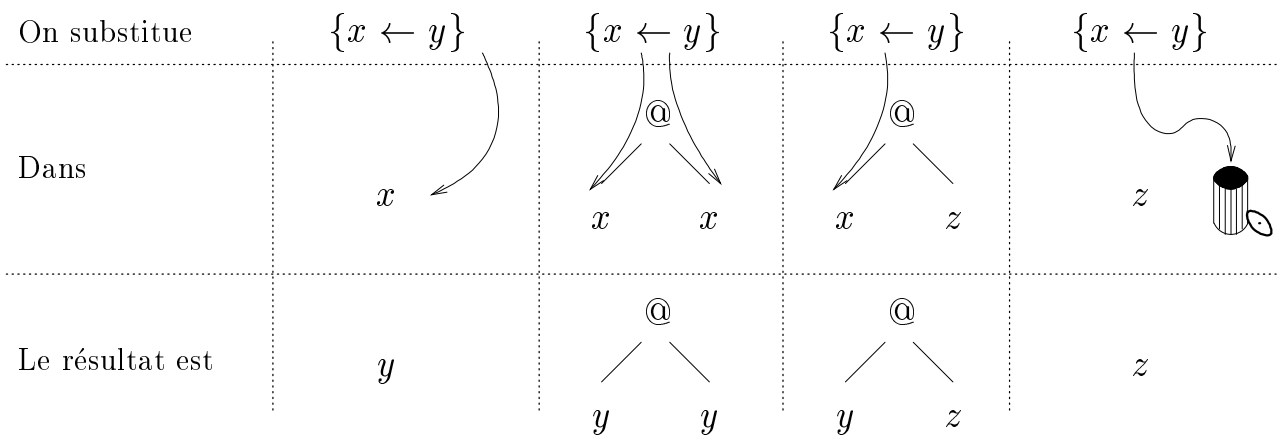
Le membre gauche de la règle correspond à l'application d'une fonction à un argument. Le filtrage (voir section 1.1.4) permet d'appliquer cette règle à tout les termes dont l'arbre commence par un nœud @ et se poursuit à gauche par un λ. Le membre droit est un peu plus complexe, nous allons l'étudier de plus près.

La notation $t\{x \leftarrow u\}$ représente un calcul implicite : celui de remplacer dans t les variables x par le terme u. Il correspond au mécanisme dont nous avons parlé pour le calcul du résultat dans la section 2.1.1. Pour calculer le membre droit de la règle, on substitue les occurrences de x par u. Comme dans l'exemple de l'addition (voir 1.1.3) ce calcul implicite est effectué instantanément avant de poursuivre la réécriture. On l'appelle la *substitution implicite*. Son calcul peut être décomposé en deux parties : premièrement, il faut *propager* la substitution jusqu'aux variables, et deuxièmement, il faut effectuer la substitution des variables concernées.

Substitution implicite, problèmes de variables et α-équivalence

Pour mieux comprendre comment cette substitution implicite fait son travail, on peut regarder les quelques exemples suivants : $(\lambda x.x)\ y \rightarrow x\{x \leftarrow y\}$, $(\lambda x.(x\ x))\ y \rightarrow (x\ x)\{x \leftarrow y\}$, $(\lambda x.(x\ z))\ y \rightarrow (x\ z)\{x \leftarrow y\}$ et $(\lambda x.z)\ y \rightarrow z\{x \leftarrow y\}$. Le tableau suivant montre la façon dont est calculé le résultat des substitutions.

[5]Les langages fonctionnels (ML, Lisp, etc.) sont bâtis sur le λ-calcul et, de ce fait, offrent cette possibilité de façon primitive. Certains langages impératifs proposent des mécanismes similaires.



On peut formaliser complètement cette *méta-opération* (c'est-à-dire opération implicite) de substitution de la façon suivante, par induction sur les termes :

$$\begin{array}{lcll} x\{x \leftarrow u\} & = & u & \\ y\{x \leftarrow u\} & = & y & \text{si } x \neq y \\ (t\ u)\{x \leftarrow v\} & = & (t\{x \leftarrow v\}\ u\{x \leftarrow v\}) & \\ (\lambda y.t)\{x \leftarrow u\} & = & \text{problème} & \\ (\lambda x.t)\{x \leftarrow u\} & = & \text{problème} & \end{array}$$

Les deux dernières lignes nécessitent plus de précautions, comme le montrent les deux exemples suivants :

- $(\lambda x.x)\{x \leftarrow y\}$ est un exemple de substitution qui pose problème pour la cinquième ligne de la formalisation ci-dessus. La variable x de la substitution correspond au paramètre formel d'une fonction qui n'apparaît plus dans le terme ; elle n'a rien à voir avec la variable x de la fonction $\lambda x.x$. La fonction $(\lambda x.x)$ ne doit donc pas être modifiée par la substitution.

  $(\lambda x.(\lambda x.x))\ y$ est un terme qui peut conduire à $(\lambda x.x)\{x \leftarrow y\}$. Cela signifie qu'on a effectué la réduction suivante :

  $$(\lambda x.(\lambda x.x))\ y \rightarrow (\lambda x.x)\{x \leftarrow y\}$$

  Si on procède naïvement au calcul de la substitution, on obtient :

  $$(\lambda x.x)\{x \leftarrow y\} = \lambda x.(x\{x \leftarrow y\}) = \lambda x.y$$

  ce qui serait une erreur.

  Le problème vient du fait que, dans le terme d'origine, $(\lambda x.(\lambda x.x))\ y$, on a employé deux fois la variable x comme paramètre formel.

- $(\lambda y.x)\{x \leftarrow y\}$ est un exemple de substitution qui pose problème pour la quatrième ligne de la formalisation ci-dessus. Comme dans l'exemple précédent, la variable y de $\lambda y.x$ est le paramètre formel de cette fonction ; elle n'a donc de sens que dans le corps de celle-ci. Par contre, la variable y de la substitution est extérieure, et donc indépendante.

  Voici un terme qui peut conduire à $(\lambda y.x)\{x \leftarrow y\}$: $(\lambda x.(\lambda y.x))\ y$. On a alors effectué la réduction suivante :

  $$(\lambda x.(\lambda y.x))\ y \rightarrow (\lambda y.x)\{x \leftarrow y\}$$

  Si on effectue le remplacement, on obtient le terme $\lambda y.y$, dans lequel la variable y correspond maintenant au paramètre formel de la fonction, ce qui serait une erreur.

  Le problème vient du fait que, dans le terme d'origine $(\lambda x.(\lambda y.x))\ y$ on a employé la variable y à la fois comme une simple variable et comme le paramètre d'une fonction.

Pour résoudre ces deux problèmes, nous devons étudier de plus près les différents types de variables qui se trouvent dans nos termes. Une notion fondamentale, autant en informatique qu'en mathématiques, est celle de *variable liée* (ou *muette* pour les mathématiciens). Son concept est le suivant.

Lorsqu'on écrit la définition d'une fonction en mathématiques, on choisit une variable pour être le paramètre formel. Par exemple, dans la définition suivante, on a choisi x :

$$f(x) = x + 1 \qquad \text{ou} \qquad f : x \mapsto x + 1$$

On aurait tout aussi bien pu choisir y ou z, ce qui nous aurait donné :

$$f(y) = y + 1 \qquad \text{ou} \qquad f(z) = z + 1$$

Dans tous les cas la fonction f reste la même ; elle renvoie toujours le même résultat lorsqu'on lui applique une même valeur. Cette variable ne sert qu'à définir la fonction, elle n'a pas d'autre signification : elle est muette, c'est une variable liée (*i.e.* liée à la définition).

Dans le λ-calcul, les variables liées sont celles introduites entre un λ et un ., comme par exemple le x dans $\lambda x.y$. Dans le terme $\lambda x.x$, on dit que l'occurrence de x dans le corps de la fonction est liée par le $\lambda x.$ qui se trouve juste avant. C'est le symbole λ qui sert à lier les variables, on l'appelle donc un *lieur*. Ainsi les termes $\lambda x.x$ et $\lambda y.y$, qui représentent la même fonction (l'identité), ne diffèrent que par le changement, ou le *renommage*, de la variable liée.

La particularité des variables liées réside dans le fait qu'on peut les changer sans modifier le sens de la fonction. Pour formaliser cela dans notre système de réécriture, on introduit une équivalence entre les termes. Celle-ci devra rendre équivalents les termes qui ne diffèrent que par un renommage des variables liées. Regardons quelques exemples de termes qui seraient équivalents :

- $\lambda x.x$ et $\lambda y.y$,
- $\lambda x.(x\ y)$ et $\lambda z.(z\ y)$,
- $(\lambda x.\lambda y.(x\ y))\ (\lambda z.z)$ et $(\lambda u.\lambda z.(u\ z))\ (\lambda z.z)$.

La notion symétrique de variable liée est celle de *variable libre*. Dans un terme, une variable est libre si elle n'est pas dans la *portée* d'un lieur pour cette variable. Par exemple, dans le terme x, la variable x est libre. Si on raisonne sur nos termes sous forme d'arbre, une variable x sera libre si il n'y a pas, au dessus d'elle, un λx dont elle serait une feuille (même lointaine). Voici quelques exemples de termes pour lesquels on indique quelles sont les variables liées et libres.

Exemple 2.1.5

- Dans le terme $\lambda x.(x\ y)$, x est liée et y est libre.
- Dans le terme $\lambda x.\lambda y.(y\ x)$, x et y sont liées.
- Dans le terme $\lambda x.(y\ z)$, y et z sont libres.
- Dans le terme $(\lambda x.x)\ x$, x apparaît à la fois comme un variable liée (dans $\lambda x.x$) et comme une variable libre (à droite de l'application).

Nous pouvons à présent regarder le cas où le renommage d'une variable liée pose problème. Dans le terme $\lambda x.(x\ y)$, x est liée, mais y est libre. Si on choisit de renommer x en y, on obtient $\lambda y.(y\ y)$; dans ce cas les deux occurrences de y sont liées. En renommant la variable liée, nous avons changé le sens de la fonction : si on lui applique la variable z, par exemple, on obtient deux résultats différents $(\lambda x.(x\ y))\ z \to z\ y$ et $(\lambda y.(y\ y))\ z \to z\ z$. On en déduit qu'il faut choisir soigneusement la nouvelle variable pour éviter ce genre de problèmes. Cependant, puisqu'on a un ensemble infini de variables, il n'est pas difficile d'en trouver une qui convient. Pour que tout se passe bien, on peut simplement choisir une variable qui n'apparaît pas du tout dans le terme, on appellera cela une *variable fraîche*.

On peut à présent définir notre équivalence, que l'on appelle α-équivalence : on dit que deux termes t et t' sont α-équivalents s'ils ne diffèrent que par un renommage de leurs variables liées (on le note $t \sim_\alpha t'$) . On définit aussi l'α-conversion qui effectue le renommage : pour tout terme, on effectue son α-conversion en prenant tous ses sous-termes de la forme $\lambda x.t$ et en les remplaçant par $\lambda y.t'$ où y est une variable fraîche et t' est le terme t dans lequel on a remplacé toutes les variables x (liées par ce λ) par y. On dit qu'on raisonne *modulo α-équivalence* quand on considère que l'on procède à des α-conversions dès qu'on en a besoin.

Revenons à notre substitution implicite et aux deux problèmes qui nous avaient arrêtés. On peut résoudre le second problème facilement en raisonnant modulo α-équivalence. Puisqu'on peut renommer les variables liées, toutes les fois où on a $(\lambda x.t)\{x \leftarrow u\}$, on α-convertit $\lambda x.t$ pour obtenir $\lambda y.t'$, et $(\lambda y.t')\{x \leftarrow u\}$ est calculé en prenant l'avant-dernière ligne de la formalisation. Pour bien comprendre le premier problème, regardons le terme suivant et sa β-réduction :

$$(\lambda x.\lambda y.(x\ y))\ y \ \rightarrow_\beta \ (\lambda y.(x\ y))\{x \leftarrow y\} \ = \ \lambda y.(y\ y)$$

Dans le premier terme, la variable y la plus à droite est libre, mais elle devient liée par le λy une fois substituée à x. C'est ce qu'on appelle une *capture* de variable. Cela change le sens de notre terme. Là encore, l'α-équivalence résout notre problème car on peut renommer la variable liée y pour s'assurer que la capture n'aura pas lieu : $(\lambda x.\lambda y.(x\ y))\ y$ est α-équivalent à $(\lambda x.\lambda z.(x\ z))\ y$ et on a

$$(\lambda x.\lambda z.(x\ z))\ y \ \rightarrow_\beta \ (\lambda z.(x\ z))\{x \leftarrow y\} \ = \ \lambda z.(y\ z)$$

La formalisation de la substitution implicite est alors complète.

Définition 2.1.6 (Substitution implicite)

La substitution implicite est définie, modulo α-équivalence, de la façon suivante, par induction sur la structure des termes :

$$\begin{array}{lcll} x\{x \leftarrow u\} & = & u & \\ y\{x \leftarrow u\} & = & y & \text{si } x \neq y \\ (t\ u)\{x \leftarrow v\} & = & (t\{x \leftarrow v\}\ u\{x \leftarrow v\}) & \\ (\lambda y.t)\{x \leftarrow u\} & = & \lambda y.(t\{x \leftarrow u\}) & \text{avec } x \neq y \text{ et } y \text{ non libre dans } u \end{array}$$

Exemple 2.1.7 (Substitutions)

Voici quelques exemples de substitutions (on fait apparaître quelques étapes du calcul) :

- $(x\ y)\{y \leftarrow z\} = (x\{y \leftarrow z\}\ y\{y \leftarrow z\}) = (x\ z)$
- $(\lambda x.y)\{y \leftarrow z\} = \lambda x.(y\{y \leftarrow z\}) = \lambda x.z$
- $(\lambda x.x)\{x \leftarrow z\} = (\lambda y.y)\{x \leftarrow z\} = \lambda y.(y\{x \leftarrow z\}) = \lambda y.y \sim_\alpha \lambda x.x$
- $(\lambda x.(x\ y))\{y \leftarrow x\} = (\lambda z.(z\ y))\{y \leftarrow x\} = \lambda z.((z\ y)\{y \leftarrow x\}) = \lambda z.(z\ x)$

On peut aussi définir formellement l'α-conversion en utilisant la substitution implicite.

Définition 2.1.8 (α-conversion, α-équivalence)

Pour réaliser l'α-conversion (ou α-équivalence) d'un terme, on remplace tous ses sous-termes de la forme $\lambda x.t$ par $\lambda y.(t\{x \leftarrow y\})$ où y est une variable fraîche. Plus formellement, on définit l'α-équivalence de la manière suivante, par induction sur la structure des termes :

$$\begin{array}{lcll} x & \sim_\alpha & x & \\ t\ u & \sim_\alpha & t'\ u' & \text{avec } t \sim_\alpha t' \text{ et } u \sim_\alpha u' \\ \lambda x.t & \sim_\alpha & \lambda y.(t'\{x \leftarrow y\}) & \text{où } y \text{ est une variable fraîche et } t \sim_\alpha t' \end{array}$$

Remarque 2.1.9

Modulo α-équivalence, les variables liées ne sont jamais touchées par une substitution : dans le terme $t\{x \leftarrow u\}$, si on a un sous-terme $\lambda x.v$, le renommage remplacera x par une variable fraîche. Ainsi, aucune variable liée ne s'appellera x.(Voir l'exemple $(\lambda x.x)\{x \leftarrow z\}$ présenté plus haut).

On aura souvent besoin de calculer l'ensemble des variables libres d'un terme. Cela se fait facilement de façon inductive.

Définition 2.1.10 (Ensemble des variables libres d'un terme)

L'ensemble des variables libres d'un terme t, noté $FV(t)$ (pour "Free Variables" en anglais) se calcule par induction sur la structure du terme, de la façon suivante :
- si le terme est une variable x, alors $FV(x) = \{x\}$,
- si le terme est une application $t\ u$, on fait l'union des variables libres de t et u : $FV(t\ u) = FV(t) \cup FV(u)$,
- si le terme est une fonction $\lambda x.t$, on prend les variables libres de t et on y enlève x qui est liée par ce λ : $FV(\lambda x.t) = FV(t) \setminus \{x\}$.

Exemple 2.1.11 (Ensemble des variables libres)

Voici quelques exemples (on fait apparaître quelques étapes de calcul sur les ensembles) :
- $FV(x\ y) = \{x, y\}$
- $FV(\lambda x.x) = \{x\} \setminus \{x\} = \emptyset$
- $FV(\lambda x.(x\ y)) = \{x, y\} \setminus \{x\} = \{y\}$
- $FV(\lambda x.\lambda y.(x\ y)) = \emptyset$
- $FV((\lambda x.(x\ y))\ (x\ z)) = (\{x, y\} \setminus \{x\}) \cup \{x, z\} = \{x, y, z\}$

Nous aurons souvent plusieurs symboles de fonction, et plusieurs lieurs, auxquels cas le calcul de l'ensemble des variables libres est à adapter en conséquence. Pour terminer avec cette section, on donne quelques exemples de β-réductions.

Exemple 2.1.12 (β-réductions)

- Soit la réduction suivante :

$$(\lambda x.(x\ x))\ (\lambda x.x) \to (x\ x)\{x \leftarrow (\lambda x.x)\} = (\lambda x.x)\ (\lambda x.x) \to x\{x \leftarrow (\lambda x.x)\} = (\lambda x.x)$$

  On remarque que dans le premier terme, il n'y a qu'un seul redex (ou β-redex), et qu'après l'avoir réduit, c'est un nouveau redex qui apparaît. En général, on ne fait pas apparaître les étapes de substitution implicite dans les dérivations. La réduction du terme s'écrit alors :

$$(\lambda x.(x\ x))\ (\lambda x.x) \to (\lambda x.x)\ (\lambda x.x) \to (\lambda x.x)$$

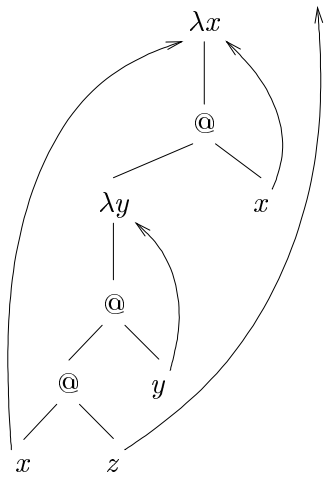
- Plusieurs réductions d'un même terme sont possibles :

$$\begin{array}{ccc} & (\lambda x.x)\ ((\lambda y.y)\ z) & \\ \swarrow & & \searrow \\ ((\lambda y.y)\ z) & & (\lambda x.x)\ z \end{array}$$

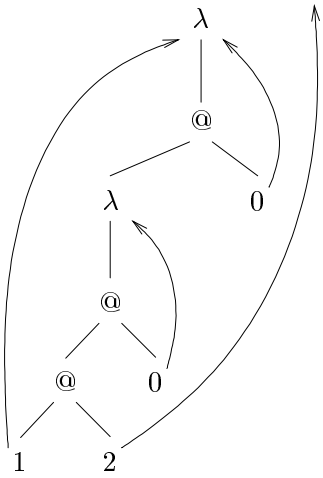
2.1.4 Les indices de de Bruijn

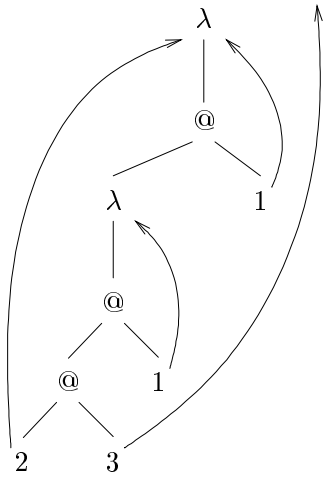
Présentation et définition

L'α-conversion et la notion de variable fraîche impliquent l'utilisation de procédés coûteux (en temps de calcul) si on souhaite implanter (c'est-à-dire programmer sur ordinateur) la β-réduction des λ-termes. Pour les supprimer, une nouvelle notation pour les λ-termes a été introduite [29, 30]. Celle-ci n'utilise plus de variables *nommées* (x, ou y, ou z, etc.), mais des entiers qui indiquent à quel λ la variable correspond. Ces entiers s'appellent des *indices de de Bruijn*. Pour comprendre comment ils sont définis, regardons le λ-terme $\lambda x.((\lambda y.((x\ z)\ y))\ x)$ sous sa forme d'arbre :



Chaque flèche relie une variable à son lieur (sauf pour z, sans lieur). Comme c'est cette information qui nous intéresse, l'idée consiste à remplacer le nom de la variable par un entier qui indique l'emplacement de son lieur. Il y a deux possibilités : soit on indique le nombre de λ qu'il faut traverser (en remontant l'arbre du terme) avant d'atteindre celui de la variable, auquel cas les indices commencent à 0, soit on indique qu'il s'agit du n-ième lieur au dessus (toujours en remontant l'arbre du terme) en mettant directement n, auquel cas les indices commencent à 1. Dans les deux cas, on numérote les variables libres en utilisant des nombres suffisamment grands pour qu'ils ne référencent aucun lieur. Par exemple, le terme $\lambda x.(x\ y)$ deviendra $\lambda(0\ 1)$ avec la première notation, et $\lambda(1\ 2)$ avec la deuxième. Pour notre terme $\lambda x.((\lambda y.((x\ z)\ y))\ x)$, on obtient soit $\lambda((\lambda((1\ 2)\ 0))\ 0)$, soit $\lambda((\lambda((2\ 3)\ 1))\ 1)$, les voici sous forme d'arbres.





Cela supprime le problème des variables nommées, mais, en contrepartie, on obtient un système dans lequel les termes sont difficilement lisibles. Par exemple, dans le terme qu'on vient d'obtenir : $\lambda((\lambda((\underline{1}\ 2)\ 0))\ \underline{0})$, il n'est pas évident de voir que les deux variables soulignées sont les mêmes (elles

n'ont pas le même numéro, mais elles font référence au même lieur), tandis que les deux variables d'indice 0 sont différentes (elles ont le même numéro, mais elles ne font pas référence au même lieur). Cela va rejaillir sur le reste des définitions : l'opération de substitution, en particulier, devra effectuer des calculs plus complexes afin de garder la cohérence des indices de de Bruijn.

On considère, dans le reste de cette section, qu'on adopte la deuxième présentation des indices, celle qui les fait commencer à 1.

Définition 2.1.13 (λ-termes avec indices de de Bruijn)

Les termes du λ-calcul avec indices de de Bruijn sont construits à l'aide de la grammaire suivante où n est un entier naturel :

$$t \quad ::= \quad n \mid \lambda t \mid t\ t$$

Exemple 2.1.14

Voici quelques exemples de λ-termes avec indices de de Bruijn. Pour chacun d'eux, on donne un terme correspondant dans le λ-calcul avec noms de variable :

- n : tout indice est un terme (x).
- $\lambda 1$: c'est la fonction identité ($\lambda x.x$).
- $\lambda 2$: cette fonction n'utilise pas son paramètre ($\lambda x.y$).
- $(\lambda 1)\ 1$: c'est l'application de la fonction identité à la variable libre 1 ($(\lambda x.x)\ y$).
- $1\ 2$: c'est l'application de 1 à 2 ($y\ x$).

Voici la règle de réduction β, dans laquelle t et u sont des méta-variables de terme :

$$(\lambda t)\ u \quad \to \quad t\{1 \leftarrow u\}$$

Substitution implicite avec indices de de Bruijn

On s'intéresse à présent à la nouvelle substitution implicite. Le terme $\lambda 1$ peut voir arriver à lui une substitution telle que $(\lambda 1)\{1 \leftarrow t\}$. Les deux problèmes dont on a parlé pour la substitution implicite du calcul nommé se posent aussi dans le calcul avec indices : une substitution semble vouloir remplacer une variable liée, et il faut faire attention à la capture des variables. Pour résoudre le premier problème, regardons le terme suivant :

$$(\lambda\lambda(1\ 2))1 \to (\lambda(1\ 2))\{1 \leftarrow 1\}$$

Ce terme peut s'écrire, dans le calcul nommé, $(\lambda x.\lambda y.(y\ x))\ z$ et il s'y réduit vers $(\lambda y.(y\ x))\{x \leftarrow z\}$, c'est-à-dire $(\lambda y.(y\ z))$. On voit bien que dans le terme $(\lambda(\underline{1}\ 2))\{1 \leftarrow 1\}$ ce n'est pas la variable soulignée qui doit être remplacée, même si elle porte le bon numéro, mais sa voisine qui porte le numéro 2. Lorsque la substitution va entrer dans le corps de la fonction l'indice à substituer ne sera plus 1, mais 2. En regardant attentivement, on constate que c'est normal puisqu'entre la substitution et sa variable, il y a alors un λ de moins.

Par contre, après être passées sous le λ, les variables du corps de la substitution (ce qui est à droite de la flèche) ont un λ de plus à traverser pour atteindre leur lieur. C'est le problème de la capture de variable qui apparaît ici. Cependant, l'indice à remplacer a comme valeur le nombre de lieurs traversés depuis la création de la substitution, il suffira donc d'ajouter ce nombre à toutes les variables libres du corps de la substitution. Sans entrer plus avant dans les détails, on donne la définition suivante de la substitution implicite du λ-calcul avec indices de de Bruijn.

Définition 2.1.15 (Substitution implicite (indices de de Bruijn))

La substitution implicite est définie de la façon suivante, par induction sur la structure des termes :

$$\begin{array}{lcll} n\{n \leftarrow u\} & = & \mathcal{U}_0^n(u) & \\ m\{n \leftarrow u\} & = & m & \text{si } m < n \\ m\{n \leftarrow u\} & = & m-1 & \text{si } m > n \\ (t\ u)\{n \leftarrow v\} & = & (t\{n \leftarrow v\}\ u\{n \leftarrow v\}) & \\ (\lambda t)\{n \leftarrow u\} & = & \lambda(t\{n+1 \leftarrow u\}) & \end{array}$$

La fonction $\mathcal{U}_i^n(t)$ effectue la mise à jour des indices pour éviter la capture des variables. Voici sa définition par induction structurelle :

$$\begin{array}{lcll} \mathcal{U}_i^n(t\ u) & = & \mathcal{U}_i^n(t)\ \mathcal{U}_i^n(u) & \\ \mathcal{U}_i^n(\lambda t) & = & \lambda\mathcal{U}_{i+1}^n(t) & \\ \mathcal{U}_i^n(m) & = & m & \text{si } m \leq i \\ \mathcal{U}_i^n(m) & = & m+n-1 & \text{si } m > i \end{array}$$

2.1.5 Propriétés

Voici un bref rappel de deux propriétés du λ-calcul : sa terminaison et sa confluence. Ces propriétés sont valables aussi bien pour la version nommée que pour les versions avec indices de de Bruijn.

Terminaison

Le λ-calcul n'est pas fortement normalisant et voici le contre-exemple le plus connu. On appelle Δ le terme $\lambda x.(x\ x)$ et Ω le terme $\Delta\ \Delta = (\lambda x.(x\ x))\ (\lambda x.(x\ x))$. Il n'y a qu'un seul redex dans ce terme, on peut faire une β-réduction :

$$\Delta\ \Delta \ = \ (\lambda x.(x\ x))\ (\lambda x.(x\ x)) \ \rightarrow \ (x\ x)\{x \leftarrow (\lambda x.(x\ x))\} \ = \ (\lambda x.(x\ x))\ (\lambda x.(x\ x)) \ = \ \Delta\ \Delta$$

Après une étape de réduction, on obtient à nouveau le même terme. On peut alors recommencer à l'infini :

$$\Delta\ \Delta \ \rightarrow \ \Delta\ \Delta \ \rightarrow \ \Delta\ \Delta \ \rightarrow \ \Delta\ \Delta \ \rightarrow \ \Delta\ \Delta \ \rightarrow \ ...$$

On verra plus loin (voir section 3.1) qu'il est possible d'identifier un sous-ensemble de λ-termes qui sont fortement normalisants.

Confluence

Théorème 2.1.16 (Confluence du λ-calcul)

Le λ-calcul est confluent.

Preuve : Voir [7]. Pour la version avec indices de de Bruijn, on trouvera une preuve dans [50] qui utilise des traductions entre les termes avec indices et les termes avec noms. ■

Dans l'étude de la confluence, on peut s'intéresser au terme $(\lambda y.((\lambda x.t)\ u))\ v$ dont voici deux réductions possibles. Le chemin qui part à gauche correspond à la réduction du redex $(\lambda x.t)\ u$ tandis que celui de droite correspond à la réduction du redex $(\lambda y.t')\ v$ (où $t' = ((\lambda x.t)\ u)$). Remarquons au passage que, modulo α-conversion, la variable x n'est pas libre dans v puisqu'on peut la renommer

dans $\lambda x.t$ par une variable fraîche.

$$\begin{array}{ccc} & (\lambda y.((\lambda x.t)\ u))\ v & \\ \swarrow & & \searrow \\ (\lambda y.(t\{x \leftarrow u\}))\ v & & ((\lambda x.t)\ u)\{y \leftarrow v\} \\ & & = \\ \downarrow & & (\lambda x.(t\{y \leftarrow v\}))\ (u\{y \leftarrow v\}) \\ & & \downarrow \\ t\{x \leftarrow u\}\{y \leftarrow v\} & & t\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} \end{array}$$

Pour obtenir la confluence, il faut vérifier que ces termes sont égaux. C'est l'objet du lemme suivant. Regardons de plus près chaque terme afin de se donner quelques intuitions.

- Dans le terme $t\{x \leftarrow u\}\{y \leftarrow v\}$, on remplace d'abord tous les x par u puis, dans le terme obtenu, on remplace tous les y par v (y compris dans les u précédemment substitués).
- Dans le terme $t\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\}$, on commence par remplacer tous les y par v dans t. Ensuite, dans le terme obtenu, on remplace tous les x par le terme u dans lequel on a remplacé tous les y par v. Comme x n'est pas dans v, la substitution $\{x \leftarrow u\{y \leftarrow v\}\}$ ne concerne, dans $t\{y \leftarrow v\}$ que les variables x présentes dans t.

Comme on va le voir dans le lemme suivant, cette égalité est aisément vérifiée. Cependant, on trouvera une configuration similaire dans la section suivante qui sera plus difficile à gérer.

Lemme 2.1.17 (Lemme de substitution)

Soient t, u et v des λ-termes tels que la variable x n'est pas libre dans v ($x \notin FV(v)$). On a

$$t\{x \leftarrow u\}\{y \leftarrow v\} \quad = \quad t\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\}$$

Preuve : Par induction sur la structure des termes :

- Si t est une variable, trois cas sont possibles :
  - soit $t = x$, alors on a
  $$x\{x \leftarrow u\}\{y \leftarrow v\} \quad = \quad u\{y \leftarrow v\}$$
  et
  $$x\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} \quad = \quad x\{x \leftarrow u\{y \leftarrow v\}\} \quad = \quad u\{y \leftarrow v\},$$
  - soit $t = y$, alors on a
  $$y\{x \leftarrow u\}\{y \leftarrow v\} \quad = \quad y\{y \leftarrow v\} \quad = \quad v$$
  et
  $$y\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} \quad = \quad v\{x \leftarrow u\{y \leftarrow v\}\} \quad = \quad v$$
  car x n'est pas libre dans v,
  - soit $t = z$ (avec $z \neq x$ et $z \neq y$), alors on a
  $$z\{x \leftarrow u\}\{y \leftarrow v\} \quad = \quad z\{y \leftarrow v\} \quad = \quad z$$
  et
  $$z\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} \quad = \quad z\{x \leftarrow u\{y \leftarrow v\}\} \quad = \quad z.$$
- Si t est une application $t_1\ t_2$, alors on a
$$(t_1\ t_2)\{x \leftarrow u\}\{y \leftarrow v\} \quad = \quad (t_1\{x \leftarrow u\}\{y \leftarrow v\}\ t_2\{x \leftarrow u\}\{y \leftarrow v\})$$
et
$$(t_1\ t_2)\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} \quad = \quad (t_1\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\}\ t_2\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\}).$$

Par hypothèse d'induction, on a

$$t_1\{x \leftarrow u\}\{y \leftarrow v\} \quad = \quad t_1\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\}$$

et

$$t_2\{x \leftarrow u\}\{y \leftarrow v\} \quad = \quad t_2\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\},$$

ce qui prouve ce point.

- Si t est une abstraction $\lambda z.t'$ (modulo α-conversion), alors on a

$$(\lambda z.t')\{x \leftarrow u\}\{y \leftarrow v\} \quad = \quad \lambda z.(t'\{x \leftarrow u\}\{y \leftarrow v\})$$

et

$$(\lambda z.t')\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\} \quad = \quad \lambda z.(t'\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\}).$$

Par hypothèse d'induction, on a

$$t'\{x \leftarrow u\}\{y \leftarrow v\} \quad = \quad t'\{y \leftarrow v\}\{x \leftarrow u\{y \leftarrow v\}\},$$

ce qui prouve ce point.

■

2.2 Les λ-calculs avec substitutions explicites

Comme nous l'avons vu précédemment, la substitution est une opération *implicite* qui n'est pas comptée dans les étapes de réécriture. Si c'est suffisant d'un point de vue théorique, c'est en revanche insatisfaisant d'un point de vue opérationnel ou calculatoire. Lorsqu'on veut observer la réduction d'un λ-terme afin de mesurer la quantité de calcul nécessaire, le nombre de β-réductions effectuées n'est qu'une piètre mesure. Prenons par exemple un terme $(\lambda x.t)\ u$, sa réduction vers $t\{x \leftarrow u\}$ est considérée comme une unique étape de calcul. Cependant, supposons que la variable x apparaisse un très grand nombre de fois dans le terme t et que le terme u ait une très grande taille, la substitution pourrait nécessiter un temps de calcul considérable. En rendant l'opération de substitution interne au système de réécriture du λ-calcul, on peut combler ce manque de précision.

Les calculs avec substitutions explicites rendent, comme leur nom l'indique, l'opération de substitution *explicite*. Ils étendent la grammaire des termes afin d'y intégrer la substitution, et ils ajoutent de nouvelles règles de réduction pour gérer ces substitutions - essentiellement leur propagation jusqu'aux variables. Cela fait, on dispose d'un outil plus fin pour analyser la réduction des termes : on peut choisir quelle substitution propager et à quel moment le faire. Cependant, dès l'arrivée des premiers calculs avec substitutions explicites, il est apparu que certaines propriétés étaient difficiles à établir, voire perdues, dans ces nouveaux systèmes.

Lorsqu'un calcul avec substitutions explicites est défini, on appelle *terme pur* un terme sans substitution. Par extension, on appelle *calcul pur* l'ancien calcul (sans substitution explicite) qui sera souvent le λ-calcul (dit λ-calcul pur).

2.2.1 Approche simple : le λx-calcul

Le plus simple calcul avec substitutions explicites est sans conteste le $\lambda\mathtt{x}$-calcul [68, 11], même s'il n'a pas été historiquement le premier proposé. Il se contente de transformer les accolades des substitutions en crochets, et de les ajouter dans les termes. La figure 2.1 présente sa grammaire et ses règles de réduction. Dans ce calcul, $(\lambda x.t)[y \leftarrow u]$ est un terme à part entière. On peut constater que les règles ajoutées sont exactement celles qui définissent la substitution implicite. On note parfois la substitution $[t/x]$ au lieu de $[x \leftarrow t]$.

Voici la grammaire des termes :

$$t \ ::= \ x \mid \lambda x.t \mid t \ t \mid t[x \leftarrow t]$$

Voici les règles de réduction :

$$\begin{array}{lll}
(\lambda x.t) \ u & \rightarrow_{Beta} & t[x \leftarrow u] \\
 & & \\
(t \ u)[x \leftarrow v] & \rightarrow_{App} & (t[x \leftarrow v] \ u[x \leftarrow v]) \\
(\lambda x.t)[y \leftarrow u] & \rightarrow_{Lambda} & \lambda x.(t[y \leftarrow u]) \\
x[x \leftarrow t] & \rightarrow_{Var1} & t \\
y[x \leftarrow t] & \rightarrow_{Var2} & y
\end{array}$$

FIG. 2.1 – Définition du $\lambda\mathtt{x}$-calcul

La règle *Beta* ressemble beaucoup à la règle β du λ-calcul. C'est elle qui crée les substitutions. On trouvera une règle de ce genre dans tous les autres calculs avec substitutions explicites. Voici un exemple de réduction dans le $\lambda\mathtt{x}$-calcul :

$$(\lambda x.((\lambda y.((x \ z) \ y)) \ x)) \ z \quad \rightarrow_{Beta} \quad ((\lambda y.((x \ z) \ y)) \ x)[x \leftarrow z]$$

Dans le λ-calcul, la substitution serait automatiquement propagée jusqu'aux variables. Ici, on peut choisir de réduire différemment, par exemple en s'occupant de l'autre *Beta*-redex.

$$((\lambda y.((x \ z) \ y)) \ x)[x \leftarrow z] \quad \rightarrow_{Beta} \quad ((x \ z) \ y)[y \leftarrow x][x \leftarrow z]$$

En revanche, arrivé à ce point, on ne peut plus choisir de propager la substitution $[x \leftarrow z]$ tant qu'on a pas propagé un peu $[y \leftarrow x]$, car aucune règle de réduction ne permet de gérer le cas où deux substitutions se rencontrent. On reparlera de cela plus loin. Terminons la réduction (à chaque étape, on souligne la substitution que l'on propage) :

$$\begin{array}{c}
((x \ z) \ y)\underline{[y \leftarrow x]}[x \leftarrow z] \\
\downarrow App \\
((x \ z)[y \leftarrow x] \ y[y \leftarrow x])\underline{[x \leftarrow z]} \\
\downarrow App \\
((x \ z)[y \leftarrow x][x \leftarrow z] \ y\underline{[y \leftarrow x]}[x \leftarrow z]) \\
\downarrow Var1 \\
((x \ z)[y \leftarrow x][x \leftarrow z] \ x\underline{[x \leftarrow z]}) \\
\downarrow Var1 \\
((x \ z)\underline{[y \leftarrow x]}[x \leftarrow z] \ z) \\
\downarrow App \\
((x[y \leftarrow x] \ z[y \leftarrow x])\underline{[x \leftarrow z]} \ z) \\
\downarrow App \\
((x[y \leftarrow x][x \leftarrow z] \ z\underline{[y \leftarrow x]}[x \leftarrow z]) \ z) \\
\downarrow Var2 \\
((x[y \leftarrow x][x \leftarrow z] \ z\underline{[x \leftarrow z]}) \ z) \\
\downarrow Var2 \\
((x\underline{[y \leftarrow x]}[x \leftarrow z] \ z) \ z) \\
\downarrow Var2 \\
((x\underline{[x \leftarrow z]} \ z) \ z) \\
\downarrow Var1 \\
((z \ z) \ z)
\end{array}$$

Dans le vocabulaire des substitutions explicites, on appelle *calcul des substitutions* le sous-ensemble des règles de réduction qui s'occupent de la propagation des substitutions, c'est-à-dire les règles qui ont des substitutions dans leur membre gauche. Pour le λx-calcul, le calcul des substitutions, appelé x-calcul, est composé des règles *App*, *Lambda*, *Var1*, et *Var2*. Pour bien visualiser celui-ci dans la définition des règles de réduction, on le sépare souvent du reste des règles en insérant une ligne vide.

2.2.2 Le dilemme de la composition

Dans la section suivante, nous présenterons les propriétés qui sont habituellement recherchées pour les calculs avec substitutions explicites. La confluence, comme nous l'avons vu au chapitre 1, est une propriété qui porte sur les termes pouvant contenir des méta-variables. La présence de ces méta-variables empêche de réduire les termes à partir d'un certain point. Par exemple, si le terme $x[y \leftarrow z]$ peut se réduire vers x (par la règle *Var2*), par contre le terme $a[y \leftarrow z]$, où a est une méta-variable, ne peut plus être réduit car on ne sait rien de a. Si c'est une variable, il faudra utiliser l'une des règles *Var1* ou *Var2*, si c'est une application ou une abstraction, il faudra respectivement utiliser la règle *App* ou *Lambda*.

Les termes avec méta-variables sont importants pour certaines applications des substitutions explicites. La confluence est donc une propriété très recherchée. Mais lorsqu'on cherche à l'établir, on se retrouve avec la paire critique dont nous avons parlé à l'occasion de la confluence du λ-calcul (avec l'étude du terme $(\lambda y.((\lambda x.t)\ u))\ v$). Nous avions vu qu'avec le lemme de substitution on pouvait fermer le diagramme de réduction de ce terme (lemme 2.1.17). Regardons ce qui se passe si nous réduisons ce terme en prenant t, u et v comme des méta-variables.

$$
\begin{array}{ccc}
 & (\lambda y.((\lambda x.t)\ u))\ v & \\
\swarrow Beta & & \searrow Beta \\
(\lambda y.(t[x \leftarrow u]))\ v & & ((\lambda x.t)\ u)[y \leftarrow v] \\
 & & \downarrow App \\
 & & (\lambda x.t)[y \leftarrow v]\ (u[y \leftarrow v]) \\
Beta \downarrow & & \downarrow Lambda \\
 & & (\lambda x.(t[y \leftarrow v]))\ (u[y \leftarrow v]) \\
 & & \downarrow Beta \\
t[x \leftarrow u][y \leftarrow v] & & t[y \leftarrow v][x \leftarrow u[y \leftarrow v]]
\end{array}
$$

t étant une méta-variable, on ne peut plus réduire nos termes et il n'est pas possible de fermer le diagramme. La preuve du lemme de substitution pourra s'appliquer dès lors que t sera remplacé par un vrai terme. Le système n'est donc pas confluent.

Pourtant, ces deux termes finals : $t[y \leftarrow v][x \leftarrow u[y \leftarrow v]]$ et $t[x \leftarrow u][y \leftarrow v]$ semblent très proches. On a l'impression qu'ils donnent un moyen de gérer l'interaction entre deux substitutions, ce qu'on appelle la *composition*. Si dans le terme $t[x \leftarrow u][y \leftarrow v]$ on veut continuer à propager la substitution $[y \leftarrow v]$, on pourrait dupliquer la substitution, en envoyer un premier exemplaire à t et un second à u. Cela semble tout à fait raisonnable d'avoir la règle de réduction suivante :

$$t[x \leftarrow u][y \leftarrow v] \quad \rightarrow_{Sub} \quad t[y \leftarrow v][x \leftarrow u[y \leftarrow v]]$$

Cette règle nous permet de fermer directement notre diagramme. Si on ajoute cette règle à notre système de réécriture, alors on s'aperçoit qu'elle a des effets néfastes : certains termes qui étaient fortement normalisants, y compris dans le λ-calcul, ne le sont plus. Par exemple, le terme $((\lambda x.\lambda y.z)\ t)\ u$,

dont la forme normale dans le λ-calcul est z, peut se réduire de la façon suivante :

$$\begin{array}{c}((\lambda x.\lambda y.z)\ t)\ u\\ \downarrow\\ (\lambda y.z)[x \leftarrow t]\ u\\ \downarrow\\ (\lambda y.(z[x \leftarrow t]))\ u\\ \downarrow\\ z[x \leftarrow t][y \leftarrow u]\\ \downarrow\\ z[y \leftarrow u][x \leftarrow t[y \leftarrow u]]\\ \downarrow\\ z[x \leftarrow t[y \leftarrow u]][y \leftarrow u[x \leftarrow t[y \leftarrow u]]]\\ \downarrow\\ \vdots\end{array}$$

Pour gagner la confluence, il semble que nous soyons contraints de perdre une propriété de normalisation : c'est le dilemme de la composition.

2.2.3 Propriétés

On s'intéresse à présent aux propriétés des calculs avec substitutions explicites. Voici un récapitulatif de celles qui sont le plus souvent recherchées. Pour chacune d'elles on mentionne ce qui se passe dans le λx-calcul.

Terminaison, normalisation forte (SN)

Cette propriété n'est quasiment jamais vérifiée, pour la simple raison que le λ-calcul lui-même ne la vérifie pas. On peut, dans chaque calcul avec substitutions explicites, reproduire le contre-exemple que l'on a déjà donné. Voici ce que donne, dans le λx-calcul, la réduction du terme $\Delta\ \Delta$:

$$\begin{array}{c}(\lambda x.(x\ x))\ (\lambda x.(x\ x))\\ \downarrow \mathit{Beta}\\ (x\ x)[x \leftarrow (\lambda x.(x\ x))]\\ \downarrow \mathit{App}\\ x[x \leftarrow (\lambda x.(x\ x))]\ x[x \leftarrow (\lambda x.(x\ x))]\\ \downarrow \mathit{Var1}\\ (\lambda x.(x\ x))\ x[x \leftarrow (\lambda x.(x\ x))]\\ \downarrow \mathit{Var1}\\ (\lambda x.(x\ x))\ (\lambda x.(x\ x))\end{array}$$

Néanmoins, comme dans le cas du λ-calcul, il sera possible de prouver qu'un sous-ensemble des termes est fortement normalisant. En général, ce sera par rapport à un système de typage (voir section 3.1). C'est cette propriété qui sera recherchée, et non la terminaison du système tout entier. On formalisera cette propriété lorsque l'on aura vu le typage.

Simulation de β

Puisque le but des substitutions explicites est de raffiner le calcul en le décomposant en étapes plus élémentaires, il est normal de demander à un calcul avec substitutions explicites de trouver les mêmes résultats que la β-réduction. Si un terme t se réduit en un terme t' dans le λ-calcul (avec la β-réduction) on veut que ce même terme t admette aussi une réduction vers t' dans le calcul avec substitutions explicites.

Propriété 2.2.1 (Simulation de β)

> Soit t et t' deux termes du λ-calcul. Si $t \to_\beta t'$ alors $t \to^+ t'$ dans le calcul avec substitutions explicites.

Le $\lambda\mathtt{x}$-calcul vérifie cette propriété.

Proposition 2.2.2

> Soit t et t' deux termes du λ-calcul. Si $t \to_\beta t'$ alors $t \to^+_{\lambda\mathtt{x}} t'$.

Preuve : L'idée est la suivante : si $t \to_\beta t'$, alors on utilise la règle *Beta* pour réduire le même redex, puis on utilise le calcul des substitutions pour propager la substitution ainsi créée jusqu'aux variables, où elle est alors soit effacée, soit utilisée. On montre qu'on obtient le terme t'. Voir [11]. ■

Préservation de la normalisation forte (PSN)

Cette propriété porte, comme la précédente, sur les relations entre le λ-calcul et le calcul avec substitutions explicites considéré. Elle dit que si un terme est fortement normalisant dans le λ-calcul, alors il l'est aussi dans le calcul avec substitutions explicites. C'est cette propriété que l'on perd souvent lorsqu'on ajoute des règles de composition de substitutions.

Propriété 2.2.3 (PSN)

> Soit t un terme du λ-calcul, si t est fortement normalisant alors il est aussi fortement normalisant dans le calcul avec substitutions explicites :
>
> $$t \in \mathcal{SN}_\lambda \;\Rightarrow\; t \in \mathcal{SN}_C$$
>
> C étant le calcul considéré.

Dans le cas de $\lambda\mathtt{x}$, la première version de règles proposée (figure 2.1) vérifie la propriété PSN.

Proposition 2.2.4

> $t \in \mathcal{SN}_\lambda \;\Rightarrow\; t \in \mathcal{SN}_{\lambda\mathtt{x}}$.

Preuve : Voir [12]. ■

Par contre, la deuxième version, avec la règle supplémentaire *Sub*, ne la vérifie pas. Un contre-exemple est exhibé dans [12].

Confluence

Comme nous l'avons mentionné dans la section précédente, il y a deux formes de confluence : la confluence, et la confluence sur les termes clos. La confluence sur les termes clos est vérifiée par la plupart des systèmes, tandis que la confluence est souvent en conflit avec la propriété PSN. Par exemple, pour $\lambda\mathtt{x}$, la version qui a PSN n'est pas confluente, et réciproquement.

Propriétés du calcul des substitutions

Pour obtenir les propriétés ci-dessus, il faut en général commencer par prouver des propriétés similaires pour le calcul des substitutions. Ces propriétés incluent la normalisation forte et la confluence, ainsi que le fait que ses formes normales sont des termes purs. Certaines d'entre elles sont prouvées, pour $\lambda\mathtt{x}$, dans la section 12.2 ; ce sont des rappels de [11].

Voici la grammaire des termes :

$$t ::= n \mid (t\ t) \mid \lambda t \mid t[s]$$
$$s ::= a/ \mid \Uparrow (s) \mid \uparrow$$

Voici les règles de réduction :

$(\lambda t)u$	$\rightarrow_{B}$	$t[u/]$
$(t\ u)[s]$	$\rightarrow_{App}$	$(t[s])\ (u[s])$
$(\lambda t)[s]$	$\rightarrow_{Lambda}$	$\lambda(t[\Uparrow (s)])$
$1[t/]$	$\rightarrow_{FVar}$	t
$n+1[t/]$	$\rightarrow_{RVar}$	n
$1[\Uparrow (s)]$	$\rightarrow_{FVarLift}$	1
$n+1[\Uparrow (s)]$	$\rightarrow_{RVarLift}$	$n[s][\uparrow]$
$n[\uparrow]$	$\rightarrow_{VarShift}$	$n+1$

FIG. 2.2 – Définition du $\lambda\upsilon$-calcul

2.2.4 Tour d'horizon

Le $\lambda\upsilon$-calcul

Ce calcul [59, 9] est au λ-calcul avec indices de de Bruijn ce que $\lambda\mathbf{x}$ est au λ-calcul. Là encore, il s'agit de rendre la substitution explicite, sans ajouter d'autres ingrédients. À cause des opérations sur les indices de de Bruijn, sa définition est bien plus complexe que celle de $\lambda\mathbf{x}$. La figure 2.2 indique la définition du $\lambda\upsilon$-calcul. On remarque que les substitutions ne comportent pas d'indication directe sur la variable qu'elles doivent substituer. Cette information est cachée dans les $\Uparrow$ qui sont ajoutés à chaque traversée d'un λ. La mise à jour des indices après passage de la substitution est assurée par l'opérateur $\uparrow$, comme le montre bien la règle *VarShift*.

Le calcul des substitutions est fortement normalisant, confluent sur les termes clos et ses formes normales sont des termes purs ; le $\lambda\upsilon$-calcul vérifie la simulation de β, la confluence, et la préservation de la normalisation forte (voir [9]). Comme il n'a pas de règle de composition, il ne peut pas être confluent.

Le $\lambda\sigma$-calcul

C'est historiquement le premier calcul avec substitutions explicites [1], ses racines sont dans [20, 42]. Il propose de gérer les substitutions explicites sous forme de listes plutôt que par unités séparées. Comme $\lambda\upsilon$, c'est un calcul avec indices de de Bruijn. La figure 2.3 montre la définition du $\lambda\sigma$-calcul. Les substitutions regroupent plusieurs associations variable-terme : par exemple, $t[1 \cdot u \cdot 3 \cdot v \cdot id]$ correspond, dans le λ-calcul, au terme $t[2 \leftarrow u][4 \leftarrow v]$. Il n'y a que le 1 comme indice de de Bruijn, les autres sont construits avec les substitutions $[\uparrow] : 2 = 1[\uparrow]$, $3 = 1[\uparrow][\uparrow]$, etc. La constante id marque la fin d'une liste de substitutions.

Ce calcul offre une plus grande dynamique que $\lambda\upsilon$ car il possède des règles pour gérer la composition de substitutions (*Clos* et *Map* essentiellement). Cependant, il n'est pas confluent. Une version légèrement modifiée de ce calcul, le $\lambda\sigma_{\Uparrow}$-calcul [21], a été proposée peu de temps après pour combler ce manque. Ces deux calculs simulent β, et aucun des deux ne vérifie PSN [61]. Le calcul des substitutions est fortement normalisant, confluent et ses formes normales sont des termes purs (voir [1, 22, 43, 67, 72]).

Voici la grammaire des termes :

$$t ::= 1 \mid (t\ t) \mid \lambda t \mid t[s]$$
$$s ::= id \mid \uparrow \mid t \cdot s \mid s \circ s$$

Voici les règles de réduction :

$(\lambda t)u$	$\rightarrow_B$	$t[u \cdot id]$
$(t\ u)[s]$	$\rightarrow_{App}$	$(t[s])\ (u[s])$
$(\lambda t)[s]$	$\rightarrow_{Lambda}$	$\lambda(t[1 \cdot (s \circ \uparrow)])$
$1[id]$	$\rightarrow_{VarId}$	1
$1[t \cdot s]$	$\rightarrow_{VarCons}$	t
$t[s][s']$	$\rightarrow_{Clos}$	$t[s \circ s']$
$id \circ s$	$\rightarrow_{IdL}$	s
$\uparrow \circ id$	$\rightarrow_{ShiftId}$	$\uparrow$
$\uparrow \circ (t \cdot s)$	$\rightarrow_{ShiftCons}$	s
$(t \cdot s) \circ s'$	$\rightarrow_{Map}$	$t[s'] \cdot (s \circ s')$
$(s_1 \circ s_2) \circ s_3$	$\rightarrow_{Ass}$	$s_1 \circ (s_2 \circ s_3)$

FIG. 2.3 – Définition du $\lambda\sigma$-calcul

Voici la grammaire des termes :

$$t ::= x \mid (t\ t) \mid \lambda x.t \mid t[s]$$
$$s ::= id \mid (t/x) \cdot s \mid s \circ s$$

Voici les règles de réduction :

$(\lambda x.t)u$	$\rightarrow_B$	$t[(u/x) \cdot id]$	
$(t\ u)[s]$	$\rightarrow_{App}$	$(t[s])\ (u[s])$	
$(\lambda x.t)[s]$	$\rightarrow_{Lambda}$	$\lambda y.(t[(y/x) \cdot s])$	avec y variable fraîche
$x[id]$	$\rightarrow_{VarId}$	x	
$x[(t/x) \cdot s]$	$\rightarrow_{VarCons1}$	t	
$x[(t/y) \cdot s]$	$\rightarrow_{VarCons2}$	$x[s]$	$(x \neq y)$
$t[s][s']$	$\rightarrow_{Clos}$	$t[s \circ s']$	
$id \circ s$	$\rightarrow_{IdL}$	s	
$((t/x) \cdot s) \circ s'$	$\rightarrow_{Map}$	$(t[s']/x) \cdot (s \circ s')$	
$(s_1 \circ s_2) \circ s_3$	$\rightarrow_{Ass}$	$s_1 \circ (s_2 \circ s_3)$	

FIG. 2.4 – Définition du $\lambda\sigma_n$-calcul

Le $\lambda\sigma_n$-calcul

Dans [1] est présentée une version nommée du $\lambda\sigma$-calcul. Il a les mêmes propriétés, ainsi que les mêmes défauts, que sa version avec indices. La figure 2.4 montre sa définition.

Voici la grammaire des termes :

$$t ::= n \mid (t\ t) \mid \lambda t \mid [n/t, n]t \mid \langle n \rangle t$$

Voici les règles de réduction :

b_1	$(\lambda t)\ u$	$\to$	$[0/u, 0]t$	
b_2	$(\langle k \rangle \lambda t)\ u$	$\to$	$[0/u, k]t$	
f	$[i/u, j]\lambda t$	$\to$	$\lambda[i+1/u, j]t$	
a	$[i/u, j](t\ v)$	$\to$	$([[i/u, j]t)\ ([i/u, j]v))$	
e_1	$[i/u, j]\langle k \rangle t$	$\to$	$\langle j+k-1 \rangle t$	si $i < k$
e_2	$[i/u, j]\langle k \rangle t$	$\to$	$\langle k \rangle [i-k/u, j]t$	si $i \geq k$
n_1	$[i/u, j]k$	$\to$	k	si $i > k$
n_2	$[i/u, j]i$	$\to$	$\langle i \rangle u$	
n_3	$[i/u, j]k$	$\to$	$j+k-1$	si $i < k$
c_1	$[i/u, j][k/v, l]t$	$\to$	$[k/[i-k/u, j]v, j+l-1]t$	si $k \leq i < k+l$
c_2	$[i/u, j][k/v, l]t$	$\to$	$[k/[i-k/u, j]v, l][i-l+1/u, j]t$	si $i \geq k+l$
d	$\langle i \rangle \langle j \rangle t$	$\to$	$\langle i+j \rangle t$	

FIG. 2.5 – Définition du λ_{ws}-calcul

Le λ_{ws}-calcul

Ce calcul [26, 41, 27] est le premier à posséder toutes les propriétés recherchées. Pour garantir PSN en présence d'une règle de composition, il propose d'ajouter des informations sur les variables absentes dans certains sous-termes. Ainsi, on évite de propager une substitution à l'intérieur d'un terme dans lequel la variable substituée n'est pas libre. De plus, ce mécanisme d'effacement permet d'économiser des étapes de réduction inutiles. Ces informations, appelées à l'origine des *étiquettes*, correspondent à des affaiblissements explicites (voir section 3.1). Comme λv, c'est un calcul avec indices de de Bruijn. La figure 2.5 montre la définition du λ_{ws}-calcul (pour une introduction plus complète de la notion d'étiquette, voir le chapitre 13).

Cette notion d'étiquette vient s'ajouter aux substitutions explicites. Cependant, comme elle n'est pas présente dans le λ-calcul, ce n'est plus par rapport à celui-ci qu'on établit les propriétés de simulation de β et de PSN. Pour cela, un λ-calcul avec étiquettes est introduit : le λ_w-calcul.

Le calcul des substitutions est fortement normalisant, confluent, et ses formes normales sont des termes purs (avec étiquettes). Le λ_{ws}-calcul simule β (du calcul avec étiquettes λ_w), est confluent, et possède la propriété PSN (voir [27]).

Le λ_{wsn}-calcul

Lors d'un travail précédent [18], nous avons proposé une version nommée du λ_{wsn}-calcul. Sa définition est donnée figure 2.6. Les étiquettes sont devenues des ensembles de variables (notés Γ dans la grammaire des termes). Une présentation plus détaillée de ce calcul sera donnée dans la partie IV de la thèse, laquelle est consacrée exclusivement à son étude.

Le λlxr-calcul

C'est sans doute le calcul avec substitutions explicites le plus récent [53]. Il s'appuie sur la relation étroite qui a été établie, dans [18], entre les substitutions explicites et les réseaux de preuve de la logique linéaire, pour rendre explicite un autre mécanisme caché dans le λ-calcul usuel. Il introduit

Voici la grammaire des termes :

$$t ::= x \mid (t\ t) \mid \lambda x.t \mid \Gamma t \mid t[x, t, \Gamma, \Gamma]$$

Voici les règles de réduction :

$$\begin{array}{lrcll}
(b_1) & (\lambda x.t)u & \to & t[x,u,\emptyset,\emptyset] & \\
(b_2) & (\Delta(\lambda x.t))u & \to & t[x,u,\emptyset,\Delta] & \\
 & & & & \\
(f) & (\lambda y.t)[x,u,\Gamma,\Delta] & \to & \lambda y.t[x,u,\Gamma\cup\{y\},\Delta] & \\
(a) & (t\ u)[x,v,\Gamma,\Delta] & \to & (t[x,v,\Gamma,\Delta]\ u[x,v,\Gamma,\Delta]) & \\
(e_1) & (\Lambda t)[x,u,\Gamma,\Delta] & \to & (\Delta\cup(\Lambda\setminus\{x\}))t & x\in\Lambda \\
(e_2) & (\Lambda t)[x,u,\Gamma,\Delta] & \to & (\Gamma\cap\Lambda)t[x,u,\Gamma\setminus\Lambda,\Delta\cup(\Lambda\setminus\Gamma)] & x\notin\Lambda \\
(n_1) & y[x,t,\Gamma,\Delta] & \to & \Delta y & x\neq y \\
(n_2) & x[x,t,\Gamma,\Delta] & \to & \Gamma t & \\
(c_1) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & & x\in\Phi\setminus\Lambda \\
 & & & t[y,u[x,v,\Gamma\setminus\Lambda,\Delta\cup(\Lambda\setminus\Gamma)],\Lambda\cap\Gamma,\Delta\cup(\Phi\setminus\{x\})] & \\
(c_2) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[x,v,(\Gamma\setminus\Phi)\cup\{y\},\Delta\cup(\Phi\setminus\Gamma)] & \\
 & & & [y,u[x,v,\Gamma\setminus\Lambda,\Delta\cup(\Lambda\setminus\Gamma)],\Lambda\cap\Gamma,\Gamma\cap\Phi] & x\notin\Phi\cup\Lambda \\
(d) & \Gamma\Delta t & \to & (\Gamma\cup\Delta)t &
\end{array}$$

FIG. 2.6 – Définition du λ_{wsn}-calcul

dans le langage un opérateur de contraction explicite qui lui permet de décomposer les β-réductions de façon encore plus fine et de se rapprocher un peu plus des réseaux de preuve. En outre, il possède toutes les propriétés recherchées. La figure 2.7 montre la définition du λlxr-calcul.

De la même façon que pour le λ_{ws}-calcul, une traduction est nécessaire pour passer des termes de λlxr aux λ-termes, mais il faut aussi une traduction pour passer des λ-termes aux λlxr-termes. Cette dernière est nécessaire afin de garantir certaines propriétés que les termes de λlxr doivent vérifier.

Le calcul des substitutions est fortement normalisant, confluent, et ses formes normales sont des termes purs (avec affaiblissements et contractions explicites). Le λlxr-calcul simule β (en considérant la traduction des termes), est confluent, et possède la propriété PSN (voir [53]). Nous reviendrons sur ce calcul, ainsi que sur ses rapports avec le λ_{wsn}-calcul, dans la partie V.

Les autres

Il serait trop long de présenter tous les calculs avec substitutions explicites existants. Entre le $\lambda\sigma$-calcul, qui est historiquement le premier, et le λ_{ws}-calcul, qui possède toutes les propriétés attendues, de nombreux calculs ont été proposés comme des tentatives pour obtenir ces propriétés. Voici, sans ordre ni comparaison, une liste non-exhaustive de calculs qu'on peut trouver dans la littérature spécialisée[6] : le plus ancien calcul qui introduit une notion d'explicitation de la substitution est le $C\lambda\xi\phi$ [30] ; le calcul des combinateurs catégoriques [20] qui a donné naissance au $\lambda\sigma$-calcul ; les calculs : λ_d et λ_{dn} [35], λs_e [49], etc.

2.2.5 Notations

Dans la suite de cette thèse, nous utiliserons les notations suivantes.

Notation 2.2.5
On notera :

[6]Par exemple, [51] et [35] proposent une bonne introduction aux substitutions explicites et à leurs applications dans les différents domaines de la sémantique.

- Λ l'ensemble des termes d'un calcul pur. La surcharge de cette notation avec celle de l'ensemble des λ-termes n'est pas ambiguë dans la mesure où, pour la plupart des calculs étudiés, c'est effectivement celui-ci qui correspond au calcul pur.
- Λ_{SN} le sous-ensemble de Λ contenant les termes fortement normalisants.
- Λ^X l'ensemble des termes d'un calcul avec substitutions explicites.
- Λ^X_{SN} le sous-ensemble de Λ^X contenant les termes fortement normalisants.

Voici la grammaire des termes :

$$t ::= x \mid \lambda x.t \mid t\ t \mid t\langle x = t\rangle \mid W_x(t) \mid C_x^{y,z}(t)$$

Voici les règles de réduction :

(B)	$(\lambda x.t)\ u$	$\to$	$t\langle x = u\rangle$	
(Abs)	$(\lambda y.t)\langle x = u\rangle$	$\to$	$\lambda y.t\langle x = u\rangle$	
$(App1)$	$(t\ v)\langle x = u\rangle$	$\to$	$t\langle x = u\rangle\ v$	$x \in FV(t)$
$(App2)$	$(t\ v)\langle x = u\rangle$	$\to$	$t\ v\langle x = u\rangle$	$x \in FV(v)$
(Var)	$x\langle x = u\rangle$	$\to$	u	
$(Weak1)$	$W_x(t)\langle x = u\rangle$	$\to$	$W_{FV(u)}(t)$	
$(Weak2)$	$W_y(t)\langle x = u\rangle$	$\to$	$W_y(t\langle x = u\rangle)$	$x \neq y$
$(Cont1)$	$C_x^{y,z}(t)\langle x = u\rangle$	$\to$	$C_\Phi^{\Delta,\Pi}(t\langle y = u_1\rangle\langle z = u_2\rangle)$	où $\Phi = FV(u)$, $u_1 = R_\Delta^\Phi(u)$ et $u_2 = R_\Pi^\Phi(u)$
$(Cont2)$	$C_w^{y,z}(t)\langle x = u\rangle$	$\to$	$C_w^{y,z}(t\langle x = u\rangle)$	$x \neq w$
$(Comp)$	$t\langle y = v\rangle\langle x = u\rangle$	$\to$	$t\langle y = v\langle x = u\rangle\rangle$	$x \in FV(v)$
$(WAbs)$	$\lambda x.W_y(t)$	$\to$	$W_y(\lambda x.t)$	$x \neq y$
$(WApp1)$	$W_y(u)\ v$	$\to$	$W_y(uv)$	
$(WApp2)$	$u\ W_y(v)$	$\to$	$W_y(uv)$	
$(Merge)$	$C_w^{y,z}(W_y(t))$	$\to$	$R_w^z(t)$	
$(Cross)$	$C_w^{y,z}(W_x(t))$	$\to$	$W_x(C_w^{y,z}(t))$	$x \neq y$ et $x \neq z$
$(CAbs)$	$C_w^{y,z}(\lambda x.t)$	$\to$	$\lambda x.C_w^{y,z}(t)$	
$(CApp1)$	$C_w^{y,z}(t\ u)$	$\to$	$C_w^{y,z}(t)\ u$	$y, z \in FV(t)$
$(CApp2)$	$C_w^{y,z}(t\ u)$	$\to$	$t\ C_w^{y,z}(u)$	$y, z \in FV(u)$
(A)	$C_w^{x,v}(C_x^{y,z}(t))$	$\sim$	$C_w^{x,y}(C_x^{z,v}(t))$	
$(C1_c)$	$C_x^{y,z}(t)$	$\sim$	$C_x^{z,y}(t)$	
$(C2_c)$	$C_{x'}^{y',z'}(C_x^{y,z}(t))$	$\sim$	$C_x^{y,z}(C_{x'}^{y',z'}(t))$	si $x \neq y', z'$ et $x' \neq y, z$
(C_w)	$W_x(W_y(t))$	$\sim$	$W_y(W_x(t))$	
(S)	$t\langle x = u\rangle\langle y = v\rangle$	$\sim$	$t\langle y = v\rangle\langle x = u\rangle$	si $y \notin FV(u)$ et $x \notin FV(v)$

FIG. 2.7 – Définition du λlxr-calcul

Chapitre 3

Typage, logique, et réseaux de preuve

Dans ce chapitre, nous présentons d'abord la notion de typage et la logique, puis la logique linéaire et les réseaux de preuve. Le lecteur qui sait déjà quel est le type de la fonction identité est dispensé de la lecture de la section 3.1. Celui qui peut donner le réseau de preuve correspondant à cette même fonction n'est pas obligé de lire la section 3.2.

3.1 Typage et logique

3.1.1 Intuition

Comme nous l'avons vu dans le chapitre précédent, les fonctions, que ce soit en informatique ou en mathématiques, sont accompagnées d'une information appelée *type*.

Les types en informatique

Pour un informaticien, le type d'une fonction sert à indiquer au compilateur du langage de programmation quels objets seront passés comme paramètres à la fonction et quel objet sera renvoyer par celle-ci. Reprenons notre exemple du `C` pour se rafraîchir la mémoire :

```
/* Cette fonction renvoie le double de l'entier pris en parametre */
int double(int n) {
  return 2*n;
}
```

Grâce aux types, le compilateur sait que la fonction `double` prendra comme paramètre un entier et renverra comme résultat un entier. Il les utilise pour compiler le programme, c'est-à-dire le réécrire sous une forme compréhensible par le micro-processeur de l'ordinateur. L'information de typage peut aussi servir, lors de la compilation, à détecter des erreurs de programmation. Par exemple, si le programmeur essaie de passer un nombre réel comme paramètre à la fonction `double`, le compilateur lui signalera son erreur. Cependant, comme le langage `C` n'est pas fortement typé, certaines erreurs de ce genre ne seront pas mentionnées, sauf sur demande expresse au compilateur.

L'approche est très différente dans les langages fonctionnels, comme par exemple en `Caml`, où les types des fonctions peuvent être *inférés*, c'est-à-dire calculés par le compilateur, à partir de leur définition. Reprenons une partie de l'exemple donné dans le chapitre précédent :

```
(* Cette fonction renvoie la somme de l'entier pris en parametre et de 3 *)
let plusTrois(n) = n+3;;
```

Dans cette définition, aucune information de type n'apparaît. Le compilateur est capable d'inférer le type de la fonction `plusTrois`. Voici, de manière simplifiée, comment il procède. Premièrement, le résultat de la fonction est `n+3`, or l'opérateur `+` opère sur deux entiers et renvoie comme résultat un entier ; le résultat de la fonction est donc un entier. Deuxièmement, le paramètre de la fonction est `n` ; il est utilisé dans l'addition précitée, ce qui permet au compilateur de conclure qu'il doit forcément être lui-aussi un entier. Si on programme en `Caml`, en mode interprété, l'interpréteur affiche le type qu'il a inféré pour la fonction. Il indique que la fonction `plusTrois` est de type `int -> int` :

```
# let plusTrois(n) = n+3;;
val plusTrois : int -> int = <fun>
```

Comme `Caml` est un langage fortement typé, le compilateur n'acceptera que des programmes complètement et correctement typés.

Les types en mathématiques

Reprenons la définition de la fonction f du chapitre précédent :

$$f : \begin{cases} \mathbb{N} \to \mathbb{N} \\ x \mapsto x + 1 \end{cases}$$

L'information de type nous indique que l'ensemble de départ ainsi que l'ensemble d'arrivée de la fonction f est l'ensemble des entiers naturels $\mathbb{N}$. On dit que la fonction associe un entier (x) à un autre entier $(x + 1)$, ou encore qu'elle prend un entier en paramètre et renvoie un entier comme résultat. Cette information est capitale pour étudier les langages de programmation.

L'origine des types en mathématiques remonte au début du siècle dernier et c'est dans la logique que cette notion trouve ses fondements [1]. À cette époque, les logiciens essayent de formaliser les mathématiques sous formes de théories logiques. Cependant, un paradoxe revient de manière récurrente dans ces théories : le paradoxe de Russel. Ce célèbre paradoxe remonte à l'antiquité ; en effet, le philosphe crétois Épiménide (VI$^{\text{ème}}$ siècle avant J.-C.) l'exprime simplement en affirmant "les crétois sont tous menteurs". Lui-même étant crétois, s'il dit la vérité, cela entre en contradiction avec son affirmation d'être menteur. Si il ment, alors son affirmation est fausse, cela signifie que les crétois disent la vérité et cela entre en contradiction avec le fait qu'il mente.

Pour éviter ce paradoxe, une possibilité consiste à contraindre les théories avec une notion de type ; c'est l'origine de la théorie des types. Dans cette théorie, il y a deux grandes catégories d'objets : les objets typés et les objets non-typés [2], avec la garantie que le paradoxe ne se trouve pas dans les termes typés.

Dans le λ-calcul non-typé, le paradoxe de Russel correspond à un terme que nous avons déjà rencontré dans le chapitre précédent : $(\lambda x.(x\ x))\ (\lambda x.(x\ x))$. Comme on va le voir ci-dessous, ce terme n'est pas typable.

Les types en sémantique

On sent bien que les notions de typage en informatique et en mathématiques sont liées. Les sémanticiens associent naturellement ces deux informations lorsqu'ils tentent de trouver la sémantique d'un programme. Par exemple, le sémanticien donne comme interprétation à la fonction `double` le λ-terme $\lambda x.(2 \times x)$ et comme type de ce λ-terme $\mathbb{N} \to \mathbb{N}$.

[1] L'introduction historique donnée ici parle des travaux [69, 66, 46, 15] (ordre chronologique). Pour une présentation historique plus complète, on pourra consulter [70].

[2] Ceux-ci sont appelés "purs" mais comme dans cette thèse on utilise la dénomination "pur" pour les termes sans substitutions explicites, on préfère ne pas surcharger ce mot pour éviter d'introduire une ambiguïté.

Le typage présente deux intérêts supplémentaires pour la sémantique. Premièrement on peut prouver que les λ-termes typables sont tous fortement normalisants. Cette propriété est très importante, car le typage d'un terme est facile et rapide à obtenir [3], tandis que la preuve de sa forte normalisation peut être difficile et longue à établir. Lorsqu'on extrait d'un programme un λ-terme correspondant à une fonction, il suffit de vérifier que celui-ci est typable pour affirmer que l'exécution de la fonction programmée s'arrêtera en produisant un résultat. Deuxièmement, le typage en tant qu'intermédiaire entre le monde de la logique et celui de la programmation permet d'établir une correspondance étroite entre ces derniers, formalisé par l'isomorphisme de Curry-Howard [47]. Celui-ci permet d'associer un λ-terme à toute preuve écrite en logique intuitionniste, et réciproquement. Cette association est très intéressante car elle permet de programmer de façon mathématique : pour écrire un programme implantant une fonction f, on commence par prouver son existence en logique intuitionniste, puis on utilise l'isomorphisme pour extraire de cette preuve un λ-terme, c'est-à-dire un programme. Le fait que le programme soit extrait de la preuve d'existence de la fonction garantit le fait que celui-ci correspond à sa spécification.

La logique

Les mathématiciens effectuent des démonstrations dans le but de prouver des propriétés, des lemmes ou des théorèmes. La logique est la branche des mathématiques qui observe les démonstrations comme des objets dont on peut étudier les propriétés et la manière dont ils sont construits. Les origines de la logique remontent aux philosophes grecs (comme Aristote, IV$^{\text{ème}}$ siècle avant J.-C.), mais les formalisations de théories logiques sont apparues avec Frege [37] (fin XIX$^{\text{ème}}$). Plusieurs formalismes logiques existent, chacun ayant leurs avantages et leurs inconvénients. Ils proposent tous des notations pour écrire, de manière formelle, les démonstrations de propositions.

Tous les hommes sont mortels, or Socrate est un homme, donc Socrate est mortel est un exemple célèbre de démonstration. Avant le XIX$^{\text{ème}}$ siècle, cette démonstration devait s'écrire dans la langue usuelle dont les phrases peuvent être ambigües, et c'est cette ambiguïté qui a fait naître la motivation d'une formalisation de la logique.

Les démonstrations doivent être rigoureuses ; il faut alors introduire un cadre formel pour les écrire. Dans la démonstration ci-dessus, *Tous les hommes sont mortels*, *Socrate est un homme*, et *Socrate est mortel* sont trois *propositions* ; les deux premières sont des hypothèses tandis que la dernière est la conclusion. Dans la proposition *Socrate est un homme*, *Socrate* et *homme* font référence à deux notions différentes : *homme* est un ensemble, alors que *Socrate* n'en est pas un ; par contre, ces notions sont liées par la proposition qui établit que *Socrate* est un élément de l'ensemble *homme*. Dans l'autre hypothèse, on voit apparaître l'ensemble *mortel*.

Formalisons un peu : on note *soc* l'élément *Socrate*, H l'ensemble des hommes, et M l'ensemble des mortels. De même, on note $\in$ l'appartenance à un ensemble. La proposition *Socrate est un homme* s'écrit $soc \in H$ et se lit "l'élément *soc* appartient à l'ensemble H". La proposition *Tous les hommes sont mortels* est d'une autre nature ; elle établit une relation entre le fait d'être un homme et le fait d'être mortel en disant : "si on est un homme, alors on est mortel", autrement dit "si un élément est dans l'ensemble H, alors il est aussi dans l'ensemble M". On note la construction logique "si ... alors ..." par le symbole $\Rightarrow$; c'est un *connecteur logique*, puisqu'il connecte deux propositions, et s'appelle *implication*. On dit que "le fait qu'un élément x soit dans H implique qu'il soit aussi dans M" et on note cela : $x \in H \Rightarrow x \in M$. On peut maintenant écrire notre démonstration de manière formelle :

Tous les hommes sont mortels, or	*Socrate est un homme, donc*	*Socrate est mortel.*
$x \in H \Rightarrow x \in M$	$soc \in H$	$soc \in M$

Les mots *or* et *donc* sont les conjonctions de la démonstration informelle. On peut s'en passer, dans la démonstation formelle, si on se fixe comme règle de lire les propositions de gauche à droite.

[3] Le typage d'un terme peut même être inféré automatiquement, comme nous l'avons dit précédemment.

Jusqu'ici, nous avons seulement écrit nos propositions sous forme mathématique ; il faut à présent formaliser le raisonnement en lui-même. Le raisonnement qu'on a utilisé dans la démonstration précédente s'appelle le *modus ponens* [4] : il permet, à partir d'une proposition $A \Rightarrow B$ et d'une proposition A, de déduire la proposition B. On utilise une représentation sous forme d'arbre pour noter les démonstrations, en utilisant des *règles d'inférence* qui sont constituées d'une barre horizontale avec du texte écrit au-dessus et en-dessous [5]. On met au-dessus les hypothèses et en-dessous les conclusions :

$$\frac{\text{hypothèses}}{\text{conclusion}}$$

On peut écrire notre démonstration sous cette forme :

$$\frac{\textit{Tous les hommes sont mortels} \quad \textit{Socrate est un homme}}{\textit{Socrate est mortel.}} \qquad \frac{x \in H \Rightarrow x \in M \quad soc \in H}{soc \in M}$$

La règle du *modus ponens* s'écrit, de façon générale, pour n'importe quelles propositions A et B :

$$\frac{A \Rightarrow B \quad A}{B}$$

Avec ces intuitions, nous pouvons aborder les définitions formelles.

3.1.2 Définitions : logique

Nous rappelons ici la définition de la déduction naturelle et du calcul des séquents. La déduction naturelle nous servira pour le typage du λ-calcul et de tous les autres calculs sauf le $\overline{\lambda}\mu\tilde{\mu}$-calcul (et sa version avec substitutions explicites) pour lequel on utilisera le calcul des séquents (voir la partie III et plus particulièrement le chapitre 9). De ces deux formalismes nous ne nous intéressons qu'aux parties qui concernent le connecteur implication, car c'est le seul connecteur dont nous aurons besoin dans la suite de notre travail. Nous poursuivons avec le typage des fonctions, c'est-à-dire des λ-termes. (La présentation ci-dessous est inspirée de [34] et, pour une présentation plus complète de la théorie des types, et en particulier du λ-calcul simplement typé, le lecteur pourra se référer à [40, 55, 60, 4, 8].)

La déduction naturelle

La notion fondatrice de la déduction naturelle est l'introduction d'hypothèse dans le raisonnement logique. Pour démontrer que A implique B, c'est-à-dire "si A est vraie, alors B est vraie", il semble naturel de vouloir prouver B avec *pour hypothèse* que A soit vraie. Ce procédé n'est pas possible dans la logique de Frege et Hilbert avec laquelle nous avons écrit notre démonstration ci-dessus. Il faut un mécanisme d'introduction d'hypothèse dans le raisonnement. La solution consiste à considérer qu'on ne démontre plus une proposition toute seule, mais sous un ensemble d'hypothèses. Au lieu de démontrer A, on démontre $\Gamma \vdash A$, qui se lit "Γ thèse A" et qui signifie "Sous les hypothèses Γ, la proposition A est vraie". Ce nouvel objet, $\Gamma \vdash A$, s'appelle un *séquent*.

Définition 3.1.1 (Séquents)

> Un séquent est une paire constituée d'un ensemble de propositions Γ (les hypothèses, $\Gamma = A_1, A_2, ..., A_n$) et d'une proposition A (celle qu'on veut démontrer sous les hyopothèses Γ).

Nous donnons ci-dessous une première version des règles de la déduction naturelle. Celle-ci comporte trois règles.

[4] On a aussi utilisé une règle d'élimination du quantificateur universel, mais cela sort du cadre logique dont on a besoin dans cette thèse.

[5] Les règles d'inférence sont une notation utilisée dans plusieurs formalismes. Nous en trouverons dans les différents formalismes logiques (déduction naturelle, calcul des séquents et logique linéaire) ainsi que pour le typage des termes des calculs que nous étudierons (λ-calcul, calculs avec substitutions explicites et autres).

Définition 3.1.2 (Règles de la déduction naturelle)

- La règle *Axiome* est celle qui permet d'utiliser une hypothèse pour démontrer la proposition qui lui est égale. Par exemple, si j'ai comme hypothèse "*Socrate est un homme*" et que je veux démontrer, sous un ensemble d'hypothèses qui contient celle-ci, la proposition "*Socrate est un homme*", je n'ai qu'à faire appel à elle. Dit plus formellement cela donne : "Si la proposition A fait partie de mon ensemble d'hypothèse, alors la proposition A est démontrable de façon immédiate". On peut noter cela par la règle d'inférence suivante :

$$\frac{}{\Gamma, A \vdash A}\ \mathit{Axiome}$$

- La règle d'introduction d'hypothèse, qu'on a présentée ci-dessus, dit : "Prouver $A \Rightarrow B$ sous l'ensemble d'hypothèses Γ revient à prouver B sous l'ensemble d'hypothèses Γ auquel on a ajouté A". Cela donne la règle d'inférence suivante :

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\ \mathit{Intr.\ Hyp.}$$

- La règle du *modus ponens* est simplement adaptée aux séquents. Elle dit : "Si j'ai démontré $A \Rightarrow B$ d'une part, et A d'autre part, alors j'en déduis B". Elle s'écrit :

$$\frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \mathit{Mod.\ Pon.}$$

Remarque 3.1.3

On a donné des noms aux règles de déduction afin de rendre les démonstrations plus simples à comprendre. Cependant, comme cela surcharge un peu leur écriture, nous ne les utiliserons pas systématiquement par la suite.

Exemple 3.1.4 (Démonstrations en déduction naturelle)

- Sous l'ensemble d'hypothèse vide, on peut démontrer $A \Rightarrow A$:

$$\dfrac{\dfrac{}{A \vdash A}\ \mathit{Axiome}}{\vdash A \Rightarrow A}\ \mathit{Intr.\ Hyp.}$$

- Sous l'hypothèse B, on peut démontrer $A \Rightarrow B$:

$$\dfrac{\dfrac{}{A, B \vdash B}\ \mathit{Axiome}}{B \vdash A \Rightarrow B}\ \mathit{Intr.\ Hyp.}$$

- On peut démontrer $\Gamma, A \vdash (A \Rightarrow B) \Rightarrow B$:

$$\dfrac{\dfrac{\dfrac{}{\Gamma, A, A \Rightarrow B \vdash A \Rightarrow B}\ \mathit{Axiome} \quad \dfrac{}{\Gamma, A, A \Rightarrow B \vdash A}\ \mathit{Axiome}}{\Gamma, A, A \Rightarrow B \vdash B}\ \mathit{Mod.\ Pon.}}{\Gamma, A \vdash (A \Rightarrow B) \Rightarrow B}\ \mathit{Intr.\ Hyp.}$$

On remarque qu'on ne s'est pas servi de Γ.

- Enfin, on peut écrire la démonstration suivante de $A \vdash A$:

$$\dfrac{\dfrac{\dfrac{}{A, A \vdash A}\ \textit{Axiome}}{A \vdash A \Rightarrow A}\ \textit{Intr. Hyp.} \quad \dfrac{}{A \vdash A}\ \textit{Axiome}}{A \vdash A}\ \textit{Mod. Pon.}$$

On peut donner une autre version des règles de la déduction naturelle, avec un changement pour la règle *Axiome*. Dans l'ancienne règle : $\Gamma, A \vdash A$, on peut s'interroger sur le sens de l'ensemble de propositions Γ. Celui-ci ne sert à rien pour effectuer la démonstration puisqu'on ne regarde que si A est présent. Observons les deux démonstrations (ou règles) suivantes :

1. $\dfrac{}{A \vdash A}$ 2. $\dfrac{}{\Gamma, A \vdash A}$

Dans les deux cas, on prouve la proposition A grâce au fait qu'elle est présente comme hypothèse. Prenons deux exemples d'énoncés (presque) mathématiques pour sentir le rôle de Γ :

1. Si x est pair, alors x est pair.
2. Si x est pair et s'il fait beau, alors x est pair.

Pour prouver les deux énoncés, on utilise la règle *Axiome* :

1. $\dfrac{}{\text{“}x \text{ est pair”} \vdash \text{“}x \text{ est pair”}}$ 2. $\dfrac{}{\text{“}x \text{ est pair”, “il fait beau”} \vdash \text{“}x \text{ est pair”}}$

Ces séquents sont tous les deux démontrables, mais leur application n'est pas la même. Pour pouvoir utiliser le premier, il suffit que x soit pair, tandis que pour utiliser le deuxième, il faut en plus qu'il fasse beau. Le deuxième énoncé s'applique moins souvent ; il est plus *faible* car l'hypothèse “il fait beau” a affaibli le séquent. Plus on ajoute d'hypothèses dans un séquent, plus on l'affaiblit. On peut formaliser cette notion d'*affaiblissement*[6] en donnant une règle explicite pour l'utiliser :

$$\dfrac{\Gamma \vdash A}{\Gamma, B \vdash A}\ \textit{Affaiblissement}$$

Cette nouvelle règle dit : “Si je peux prouver A en utilisant les hypothèses Γ, alors je peux toujours prouver A si j'ajoute une hypothèse supplémentaire”. Si on utilise cette nouvelle règle, on peut remplacer l'ancienne règle *Axiome* par la règle suivante :

$$\dfrac{}{A \vdash A}\ \textit{Axiome}$$

Dans la suite de notre travail, on prendra ce deuxième système de règles. Cependant, pour éviter d'agrandir la taille des preuves, on utilisera toujours l'ancienne règle *Axiome* dans nos démonstrations. On peut toujours convertir une application de l'ancienne règle en plusieurs applications successives de la règle *Affaiblissement*, puis en utilisant la nouvelle règle. Pour tout ensemble de propositions $A_1, A_2, ..., A_n$, on fait la transformation suivante :

$$\dfrac{}{A_1, A_2, ..., A_n, A \vdash A}\ \textit{Axiome} \quad \rightsquigarrow \quad \dfrac{\dfrac{\dfrac{}{A \vdash A}\ \textit{Axiome}}{A_n, A \vdash A}\ \textit{Affaiblissement}}{\vdots}}{\dfrac{A_2, ..., A_n, A \vdash A}{A_1, A_2, ..., A_n, A \vdash A}\ \textit{Affaiblissement}}$$

[6] “Weakening” en anglais, d'où le w dans le λ_{ws}-calcul qui incorpore un affablissement explicite.

Les coupures dans les démonstrations

Une coupure est un détour dans une démonstration. Pour effectuer la preuve d'un théorème mathématique, il est parfois plus clair de la découper en plusieurs lemmes établissant chacun un résultat intermédiaire. Par exemple, supposons qu'on ait le lemme et le théorème suivants :

```
Lemme 1 : La proposition A est vraie.
Preuve : ...preuve que A est vraie...

Théorème 1 : La proposition B est vraie.
Preuve : Pour que la proposition B soit vraie, il suffit
  que la proposition A soit vraie.
  Le lemme 1 établit que la proposition A est vraie,
  on peut donc conclure
  que la proposition B est vraie.
```

On aurait pu écrire directement la preuve du théorème, sans utiliser de lemme :

```
Théorème 1bis : La proposition B est vraie.
Preuve : Pour que la proposition B soit vraie, il suffit
  que la proposition A soit vraie. Prouvons que la propriété
  A est vraie : ...preuve que A est vraie...
  Puisqu'on vient de montrer que la proposition A est vraie,
  on peut donc conclure
  que la proposition B est vraie.
```

Le "détour" qu'on a fait en établissant le `Lemme 1` pour s'en servir dans le `théorème 1` correspond exactement à la notion de coupure.

Le lecteur attentif aura remarqué que la dernière démonstration des exemples présentés plus haut comporte un détour.

$$\dfrac{\dfrac{\dfrac{}{A, A \vdash A}\ \textit{Axiome}}{A \vdash A \Rightarrow A}\ \textit{Intr. Hyp.} \quad \dfrac{}{A \vdash A}\ \textit{Axiome}}{A \vdash A}\ \textit{Mod. Pon.}$$

En effet, le séquent $A \vdash A$ est démontrable directement par la règle *Axiome* :

$$\dfrac{}{A \vdash A}\ \textit{Axiome}$$

La première preuve comporte une *coupure*. La forme générale d'une coupure est :

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Gamma, B \vdash A}}{\Gamma \vdash B \Rightarrow A}\ \textit{Intr. Hyp.} \quad \dfrac{\pi_2}{\Gamma \vdash B}}{\Gamma \vdash A}\ \textit{Mod. Pon.}$$

Au lieu de prouver $\Gamma \vdash A$ sans détours (`Théorème 1bis`), on commence par supposer qu'on a l'hypothèse B pour démontrer $\Gamma, B \vdash A$ (`Théorème 1`). Par ailleurs, on démontre $\Gamma \vdash B$ (`Lemme 1`). On termine la démonstration en utilisant le *modus ponens*.

Il est toujours possible de transformer une preuve quelconque d'un séquent pour obtenir une preuve sans coupure du même séquent. Ce processus, appelé *élimination des coupures*, est la composante opérationnelle de la logique. On peut voir une démonstration comme un objet d'un système de

réécriture dont la règle est l'élimination des coupures. Les preuves sans coupures en sont les formes normales, car l'élimination des coupures est fortement normalisante. Comme on va le voir ci-dessous, l'isomorphisme de Curry-Howard nous dit que l'élimination des coupures correspond à la réduction d'un λ-terme jusqu'à sa forme normale. L'introduction des substitutions explicites a permis de constater que les règles d'élimination des coupures correspondent (plus ou moins) à des règles de réduction des calculs avec substitutions explicites (voir [18] et les parties IV et V de la thèse).

Le calcul des séquents

Les noms *Intr. Hyp.* et *Mod. Pon.* que nous avons donné à nos règles expliquent leur rôle du point de vue du raisonnement. Le plus souvent, les noms qui sont donnés aux règles expliquent leur rôle du point de vue des connecteurs logiques. Il s'agit de l'implication dans notre cas précis : la règle *Intr. Hyp.* l'introduit dans le séquent du bas, tandis que la règle *Mod. Pon.* l'élimine du séquent du haut. On appelle plus courament la première règle $\Rightarrow$-intro et la deuxième $\Rightarrow$-élim.

Il apparaît que les règles de la déduction naturelle sont dissymétriques[7] et, de ce fait, difficiles à utiliser pour la recherche automatique de preuves. En revanche, même si le calcul des séquents est un formalisme moins intuitif que la déduction naturelle, il a l'avantage d'y remédier. Voici, en ce qui nous concerne, les différences que l'on trouvera :

- Au lieu de règles d'introduction et d'élimination de l'implication, on aura deux règles d'introduction de celle-ci, l'une pour la partie droite du séquent, et l'autre pour la partie gauche. D'une certaine façon, une implication dans la partie gauche d'un séquent représente le contraire d'une implication dans la partie droite.
- On ajoute une règle explicite de coupure. On aura une coupure si une proposition apparaît à gauche dans une démonstration et à droite dans une autre.

Définition 3.1.5 (Règles du calcul des séquents)

- La règle *Axiome* est inchangée (et on peut aussi utiliser l'ancienne règle qui intègre les affaiblissements) :
$$\frac{}{A \vdash A}\ \textit{Axiome}$$
- La règle *Affaiblissement* est inchangée :
$$\frac{\Gamma \vdash A}{\Gamma, B \vdash A}\ \textit{Affaiblissement}$$
- Voici la règle d'introduction de l'implication à droite, qui est exactement la règle *Intr. Hyp.* :
$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\ \Rightarrow\text{-droite}$$
- La règle d'introduction de l'implication à gauche correspond à son élimination dans la déduction naturelle :
$$\frac{\Gamma \vdash A \quad \Gamma, B \vdash C}{\Gamma, A \Rightarrow B \vdash C}\ \Rightarrow\text{-gauche}$$
- La règle de coupure est la suivante :
$$\frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B}\ \textit{Coupure}$$

[7] Cela apparaît de manière plus évidente si on regarde toutes les règles de la déduction naturelle.

Exemple 3.1.6 (Démonstrations en calcul des séquents)

Reprenons nos exemples de proposition démontrées à l'aide de la déduction naturelle pour les démontrer à l'aide du calcul des séquents :

- Sous l'ensemble d'hypothèse vide, on peut démontrer $A \Rightarrow A$:

$$\dfrac{\dfrac{}{A \vdash A}\ \textit{Axiome}}{\vdash A \Rightarrow A}\ \textit{Intr. Hyp.}$$

  C'est la même démonstration.

- Sous l'hypothèse B, on peut démontrer $A \Rightarrow B$:

$$\dfrac{\dfrac{}{A, B \vdash B}\ \textit{Axiome}}{B \vdash A \Rightarrow B}\ \Rightarrow\text{-droite}$$

  C'est la même démonstration (seul le nom de la règle change).

- On peut démontrer $\Gamma, A \vdash (A \Rightarrow B) \Rightarrow B$:

$$\dfrac{\dfrac{\dfrac{}{\Gamma, A \vdash A}\ \textit{Axiome} \quad \dfrac{}{\Gamma, A, B \vdash B}\ \textit{Axiome}}{\Gamma, A, A \Rightarrow B \vdash B}\ \Rightarrow\text{-gauche}}{\Gamma, A \vdash (A \Rightarrow B) \Rightarrow B}\ \Rightarrow\text{-droite}$$

- Enfin, on peut aussi effectuer une preuve avec coupure de $A \vdash A$:

$$\dfrac{\dfrac{}{A \vdash A}\ \textit{Axiome} \quad \dfrac{}{A, A \vdash A}\ \textit{Axiome}}{A \vdash A}\ \textit{Coupure}$$

Comme pour la déduction naturelle, il est possible d'éliminer les coupures des démonstrations.

3.1.3 Définitions : typage

Le λ-calcul simplement typé

Nous allons à présent nous intéresser au typage des λ-termes. Celui-ci est très naturel et correspond aux intuitions qu'on extrait du typage des expressions dans les langages de programmation. Puisqu'on travaille sur des λ-termes, on a trois catégories de termes à typer : les variables, les λ-abstractions, et les applications. Étudions cela sur l'exemple de la fonction `plusTrois`, présenté plus haut :

- Si on veut typer cette fonction, on calcule le type du corps de la fonction en supposant que le paramètre est un entier ; on trouve que le corps de la fonction est un entier. On en conclut alors que le type de `plusTrois` est `int -> int`.
- Pour pouvoir typer la variable `n` dans le corps de la fonction, il faut utiliser l'hypothèse que le paramètre de la fonction est un entier.
- On souhaite typer l'application de cette fonction à `5`, qui est un entier. Puisqu'on sait que `plusTrois` est de type `int -> int`, on en déduit que si on donne à cette fonction un entier, elle nous rendra comme résultat un entier. Le type de `plusTrois(5)` est donc `int`.

Pour typer un terme, on recours souvent à un ensemble d'hypothèses qui permet de typer les variables libres (ou les constantes) de ce terme. Dans l'exemple précédent, on a les hypothèses : "la constante `5` est un entier, et la constante `3` est un entier". Notons Γ cet ensemble d'hypothèses initial et reprenons plus formellement les typages ci-dessus :

- Pour typer la fonction `plusTrois(n) = n+3`, on type `n+3` en ajoutant comme hypothèse que `n` est un entier. L'expression `n+3` correspond à l'application de la fonction d'addition (notée `+` de

façon infixe) aux paramètres `n` et `3`. On sait que l'addition prend deux entiers comme paramètres et renvoie un entier comme résultat. On a comme hypothèse dans Γ que `3` est un entier, et on a ajouté comme hypothèse que `n` est un entier. On en déduit que `n+3` est de type entier. Donc, la fonction `plusTrois` prend un entier en paramètre et renvoie un entier comme résultat : son type est `int -> int`.

- Pour pouvoir typer la variable `n` de type entier, nous avons utilisé l'hypothèse que `n` est de type entier.
- On souhaite typer `plusTrois(5)`. On a vu que `plusTrois` est de type `int -> int`. Puisque dans Γ on a l'hypothèse que `5` est un entier, on en déduit que `5` est un entier. On peut alors conclure que l'application de `plusTrois` à `5` nous donne un entier.

Ce qui peut s'écrire en utilisant des règles d'inférence :

$$\dfrac{\dfrac{\dfrac{\vdots}{\text{Sous les hyp. : } \Gamma \text{ et } \texttt{n} \text{ est de type } \texttt{int}\text{, alors le type de } \texttt{n+3} \text{ est } \texttt{int}}}{\text{Sous les hyp. : } \Gamma\text{, alors le type de } \texttt{plusTrois(n)=n+3} \text{ est } \texttt{int -> int}} \quad \dfrac{}{\text{Sous les hyp. : } \Gamma\text{, alors le type de } \texttt{5} \text{ est } \texttt{int}}}{\text{Sous les hyp. : } \Gamma\text{, alors le type de } \texttt{plusTrois(5)} \text{ est } \texttt{int}}$$

Si on regarde uniquement les types, on voit apparaître un objet familier :

$$\dfrac{\dfrac{\dfrac{\vdots}{\ldots\ \Gamma \text{ et } \ldots\ \texttt{int}, \ldots \texttt{int}}}{\ldots\ \Gamma, \ldots\ \texttt{int -> int}} \quad \dfrac{}{\ldots\ \Gamma, \ldots\ \texttt{int}}}{\ldots\ \Gamma, \ldots\ \texttt{int}}$$

On reconnait la forme d'une démonstration en déduction naturelle. Pour accentuer ce fait, on remplace "Sous les hyp. : X" par "$X \vdash$" et "alors le type de X est Y" par "$X : Y$". Cela nous donne la *dérivation de typage* suivante :

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, \texttt{n} : \texttt{int} \vdash \texttt{n} + 3 : \texttt{int}}}{\Gamma \vdash \texttt{plusTrois(n)} = \texttt{n} + 3 : \texttt{int -> int}} \quad \dfrac{}{\Gamma \vdash 5 : \texttt{int}}}{\Gamma \vdash \texttt{plusTrois(5)} : \texttt{int}}$$

Pour typer les λ-termes, on a besoin d'un ensemble de types. Voici sa définition.

Définition 3.1.7 (Types simples)

L'ensemble des *types simples* est défini inductivement de la façon suivante :

- Les types de base sont des types simples. Nous ne nous occuperons pas ici de savoir ce que sont ces types de base, mais le type `int` de l'exemple ci-dessus peut être vu comme un type de base.
- Si A et B sont deux types, alors $A \to B$ est un type.

Définition 3.1.8 (Typage des λ-termes)

L'ensemble des hypothèses qui servent à typer un terme s'appelle l'*environnement de typage* de ce terme. Ces hypothèses sont données sous la forme d'une association entre une variable x et son type A, ce que l'on note $x : A$. Les termes sont toujours typés avec un environnement initial, mais celui-ci peut être vide. Voici les règles de typage des λ-termes :

- Si $x : A$ est dans l'environnement de typage, alors la variable x est de type A. On écrit cela sous forme d'une règle d'inférence :

$$\dfrac{}{\Gamma, x : A \vdash x : A}\ \textit{Variable}$$

- Pour typer l'abstraction $\lambda x.t : A \to B$, il faut typer t de type B en supposant que le paramètre est de type A, c'est-à-dire en ajoutant $x : A$ à l'environnement de typage :

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ \textit{Abstraction}$$

- Pour typer l'application[8] $(t\ u) : B$, il faut que t ait le type d'une fonction, $A \to B$, et que u ait le type du paramètre[9] A, :

$$\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash u : A}{\Gamma \vdash (t\ u) : B}\ \textit{Application}$$

Exemple 3.1.9

Regardons quelques typages de λ-termes :

- Le terme $\lambda x.x$ peut être typé dans l'environnement vide :

$$\frac{\dfrac{}{x : A \vdash x : A}}{\vdash \lambda x.x : A \to A}\ \begin{matrix}\textit{Variable}\\ \textit{Abstraction}\end{matrix}$$

- Pour typer le terme $\lambda x.(y\ x)$, on a besoin d'avoir, dans l'environnement de typage, le type de y :

$$\frac{\dfrac{\dfrac{}{y : A \to B, x : A \vdash y : A \to B}\ \textit{Variable} \quad \dfrac{}{y : A \to B, x : A \vdash x : A}\ \textit{Variable}}{y : A \to B, x : A \vdash y\ x : B}\ \textit{Application}}{y : A \to B \vdash \lambda x.(y\ x) : A \to B}\ \textit{Abstraction}$$

- Pour typer le terme $(\lambda x.x)\ y$, on a besoin d'avoir, dans l'environnement de typage, le type de y :

$$\frac{\dfrac{\dfrac{}{y : A, x : A \vdash x : A}\ \textit{Variable}}{y : A \vdash \lambda x.x : A \to A}\ \textit{Abstraction} \quad \dfrac{}{y : A \vdash y : A}\ \textit{Variable}}{y : A \vdash (\lambda x.x)\ y : A}\ \textit{Application}$$

Dans la section des intuitions, nous avons annoncé que le terme $\Delta\ \Delta = (\lambda x.(x\ x))\ (\lambda x.(x\ x))$ n'est pas typable. Pour vérifier cela, essayons simplement de typer le sous-terme $x\ x$. Si celui-ci n'est pas typable, alors $\Delta\ \Delta$ ne le sera pas non plus. Commençons la dérivation de typage, sans se soucier de l'environnement, en supposant que $x\ x$ est de type A :

$$\frac{\dfrac{}{\Gamma \vdash x : B \to A}\ \textit{Variable} \quad \dfrac{}{\Gamma \vdash x : B}\ \textit{Variable}}{\Gamma \vdash x\ x : A}\ \textit{Application}$$

On voit bien qu'il y a un problème pour typer la variable x : soit x a le type $B \to A$, soit il a le type B, mais il ne peut pas avoir les deux à la fois.

L'isomorphisme de Curry-Howard

Comme on l'a remarqué plus haut, les dérivations de typage ressemblent fort à des preuves de la déduction naturelle. Prenons la dernière dérivation de nos exemples, et supprimons tout ce qui fait

[9] Dans la règle de typage de l'application, on utilise implicitement la règle de contraction sur les formules de Γ, dupliquant cet environnement pour typer les deux sous-termes.

[9] Dans l'inférence automatique du type, on a besoin d'une notion supplémentaire, l'unification, qui est proche de celle de filtrage.

référence aux λ-termes, ainsi que les symboles ":". Cela nous donne :

$$\dfrac{\dfrac{\dfrac{}{A, A \vdash A}\ \textit{Variable}}{A \vdash A \to A}\ \textit{Abstraction} \quad \dfrac{}{A \vdash A}\ \textit{Variable}}{A \vdash A}\ \textit{Application}$$

Comparons la dérivation précédente avec la preuve, en déduction naturelle, du séquent $A \vdash A$ que nous avons donnée dans l'exemple 3.1.4 :

$$\dfrac{\dfrac{\dfrac{}{A, A \vdash A}\ \textit{Axiome}}{A \vdash A \Rightarrow A}\ \textit{Intr. Hyp.} \quad \dfrac{}{A \vdash A}\ \textit{Axiome}}{A \vdash A}\ \textit{Mod. Pon.}$$

Ce sont les mêmes à quelques détails près. Le nom des règles change, et, surtout, la flèche du type des fonctions ($\to$) correspond a l'implication ($\Rightarrow$). C'est la base de l'isomorphisme de Curry-Howard. La correspondance des noms est la suivante :

- La règle *Axiome* correspond à la règle *Variable*.
- La règle *Intr. Hyp.* correspond à la règle *Abstraction*.
- La règle *Mod. Pon.* correspond à la règle *Application*.

Sans entrer dans le détail des conséquences de l'isomorphisme de Curry-Howard, nous allons présenter ce qui se passe pour l'élimination des coupures. Nous avons dit que l'élimination des coupures représente la composante opérationnelle de la logique. L'isomorphisme nous permet d'aller plus loin en disant que c'est sa composante *calculatoire*, car elle va correspondre à un calcul.

Ce n'est pas par hasard que nous avons choisi notre exemple de dérivation de typage ci-dessus. La preuve en déduction naturelle qui lui correspond contient une coupure, tandis que le λ-terme typé contient un β-redex. En déduction naturelle, une coupure est une application de la règle *Mod. Pon.* suivie d'une application de la règle *Intr. Hyp.*, tandis que dans le λ-calcul, un redex est constitué par l'*Application* d'une fonction (une *Abstraction*) à un argument. Une coupure correspond donc à un β-redex et l'élimination des coupures correspond à la β-réduction. L'isomorphisme de curry-Howard établit une connexion très étroite entre le monde de la logique et la monde des fonctions, c'est-à-dire de la programmation.

Typages des calculs avec substitutions explicites

Nous verrons, dans la suite de la thèse, des exemples de règles de typage pour les calculs avec substitutions explicites. Elles correspondent, dans l'isomorphisme de Curry-Howard, à la description pas-à-pas des étapes nécessaires pour éliminer les coupures des preuves.

3.1.4 Propriétés

Trois propriétés nous intéressent. La première se préoccupe de savoir si le typage d'un terme est décidable. La seconde concerne les relations entre typage et réduction. La troisième est la normalisation forte.

Définition 3.1.10 (Décidabilité du typage)

> On dit que le typage est *décidable* si, pour tout terme, on peut soit en donner un type, soit dire qu'il n'est pas typable.

Pour le λ-calcul, ainsi que pour les calculs avec substitutions explicites, les systèmes de typage que l'on donnera vérifieront tous cette propriété.

Définition 3.1.11 (Préservation du typage par réduction)

> Supposons que A soit le type d'un terme t. Si t se réduit en t' dans un calcul considéré, on souhaite que t' ait alors le même type A. Plus formellement, on dit qu'un ensemble de règles de réduction *préserve le typage par réduction* si, pour tout terme typé $\Gamma \vdash t : A$, on a $t \to t'$ implique $\Gamma \vdash t' : A$.

La préservation du typage est souvent fastidieuse à prouver, mais pas réellement difficile. Dans la partie IV on prouve cette propriété, par la méthode usuelle, pour le λ_{wsn}-calcul.

Le typage nous protège, en logique, contre les paradoxes. Par l'isomorphisme de Curry-Howard, il nous protège contre les dérivations infinies. La propriété suivante est sans doute la plus importante de celles que nous avons vu jusqu'à présent, et ses conséquences sont essentielles pour la sémantique.

Définition 3.1.12 (Normalisation forte)

> On dit qu'un calcul avec typage est fortement normalisant si tout terme typable est fortement normalisant. Autrement dit, le typage nous donne un sous-ensemble de termes fortement normalisants.

Théorème 3.1.13 (Normalisation forte du λ-calcul simplement typé)

> Le λ-calcul simplement typé est fortement normalisant.

Il existe plusieurs techniques de preuves de normalisation forte. Ici, nous présentons la technique de *réductibilité* qu'on utilisera dans la partie III.

Preuve : La technique de réductibilité consiste à découper le problème en deux parties articulées autour de la définition d'ensembles de termes ; ce sont des *ensembles de candidats de réductibilité*, notés ECR par la suite. Au lieu de démontrer directement que tout λ-terme typable est fortement normalisable, on procède en deux étapes :

1. On prouve que tout terme dans un ECR est fortement normalisable.
2. On prouve que tout λ-terme typable est dans un ECR.

On essaie de choisir astucieusement la définition des ECR afin d'obtenir facilement la première propriété. Dans le vocabulaire spécifique de cette technique, la deuxième propriété s'appelle l'*adéquation*. La preuve de celle-ci est parfois découpée en plusieurs preuves plus petites, de propriétés intermédiaires. En particulier, on trouve souvent une propriété qui s'appelle *fermeture par réduction* et qui établit que si un terme est dans un ECR alors ses réduits sont dans le même ECR.

On trouvera la preuve complète dans [55] et [40]. ■

Voici les résultats de normalisation forte des calculs avec substitutions explicites présentés dans le chapitre précédent.

Théorème 3.1.14

> 1. Le $\lambda\mathtt{x}$-calcul simplement typé est fortement normalisant (pour la version sans composition des substitutions).
> 2. Le $\lambda\upsilon$-calcul simplement typé est fortement normalisant.
> 3. Le λ_{ws}-calcul simplement typé est fortement normalisant.
> 4. Le λ_{wsn}-calcul simplement typé est fortement normalisant.
> 5. Le λlxr-calcul simplement typé est fortement normalisant.

Preuve :

1. Voir, par exemple, [32].
2. Voir chapitre 6.
3. Voir [18, 28].
4. Voir [18] et la partie IV pour la nouvelle version.

5. Voir [53].

■

Dans la suite de cette thèse, nous utiliserons les notations suivantes, qui font en partie suite aux notations de la section 2.2.5.

Notation 3.1.15

On notera :

- Λ_T le sous-ensemble de Λ contenant les termes typés par un système de typage T.
- Λ_T^X le sous-ensemble de Λ^X contenant les termes typés par un système de typage T.

3.1.5 Affaiblissement et contrôle des variables

Dans la correspondance que nous avons faite entre les règles de typage et les règles de déduction, nous n'avons pas parlé de la règle d'affaiblissement. Nous avons vu, en déduction naturelle, que celle-ci pouvait être ajoutée pour modifier la règle *Axiome*. L'affaiblissement consiste à ajouter une proposition supplémentaire dans les hypothèses :

$$\frac{\Gamma \vdash A}{\Gamma, B \vdash A}\ \textit{Affaiblissement}$$

Dans les dérivations de typage, par l'isomorphisme de Curry-Howard, les propositions en hypothèses constituent l'environnement de typage et donnent le type des variables libres d'un terme. Dans la règle *Variable*, on utilise de façon implicite l'affaiblissement, comme pour l'ancienne règle axiome de la déduction naturelle. Regardons à quel endroit le typage nécessite cet affaiblissement implicite. En reprenant le typage du terme $(\lambda x.x)y$, on procède en plusieurs étapes successives :

1. On peut typer $\lambda x.x$ de type $A \to A$ dans l'environnement vide :

$$\dfrac{\dfrac{}{x : A \vdash x : A}\ \textit{Variable}}{\vdash \lambda x.x : A \to A}\ \textit{Abstraction}$$

2. Pour typer, y de type A, on a besoin que $y : A$ soit dans l'environnement de typage :

$$\frac{}{y : A \vdash y : A}\ \textit{Variable}$$

3. On souhaite à présent utiliser la règle *Application*, mais cela n'est pas possible :

$$\dfrac{\dfrac{\dfrac{}{x : A \vdash x : A}\ \textit{Variable}}{\vdash \lambda x.x : A \to A}\ \textit{Abstraction} \qquad \dfrac{}{y : A \vdash y : A}\ \textit{Variable}}{?}\ \textit{Application}$$

   Ici, la règle *Application* ne peut pas s'appliquer, car elle demande que les environnements de typage des deux sous-termes de l'application soient identiques. Or l'environnement de typage de $\lambda x.x$ est vide, tandis que celui de y contient $y : A$. La solution consiste à ajouter $y : A$ dans l'environnement de typage de $\lambda x.x$, c'est un affaiblissmement.

L'affaiblissement implicite permet d'effectuer le typage d'un terme. C'est essentiel pour garantir la typabilité des termes, mais cela crée une ambiguïté. En effet, lorsqu'une variable apparaît dans un environnement de typage, on ne peut plus savoir si elle est réellement libre dans le terme, ou bien si elle correspond à un affaiblissement. Dans le λ-calcul, cela n'a aucune conséquence, mais dans les calculs avec substitutions explicites, cette ambiguïté mène à la perte de la normalisation forte en présence de règles de composition. C'est pour cela que dans [26, 41, 27] (puis [18]), le fait de noter explicitement dans les termes, l'endroit où l'on utilise l'affaiblissement permet de conserver la normalisation forte.

3.2 Logique linéaire et réseaux de preuve

Lorsqu'en mathématiques on a prouvé $A \Rightarrow B$ et $A \Rightarrow C$, on peut en déduire $A \Rightarrow B \wedge C$ (le $\wedge$ se lit "et"). Cela se lit, en français, "Si, à partir de l'hypothèse A, je peux prouver que la proposition B est vraie, et si, à partir de l'hypothèse A, je peux prouver que la proposition C est vraie, alors, à partir de l'hypothèse A, je peux prouver que les propositions B et C sont toutes les deux vraies".

- Dans la vie courante, à partir des phrases "Si il fait beau, alors je me promène" et "Si il fait beau, alors je suis content", on peut déduire que "Si il fait beau, alors je me promène et je suis content".
- En mathématiques, à partir des proposition $(x = 3) \Rightarrow (2 \times x = 6)$ et $(x = 3) \Rightarrow (3 \times x = 9)$, on peut déduire que $(x = 3) \Rightarrow (2 \times x = 6) \wedge (3 \times x = 9)$.

Jusque là, tout va bien, mais prenons cet autre exemple de la vie courante :

- À partir de "Si j'ai 1 euro, alors je peux acheter une baguette" et "Si j'ai 1 euro, alors je peux acheter un croissant", on peut déduire que "Si j'ai 1 euro, alors je peux acheter une baguette et je peux acheter un croissant".

On devine bien qu'il y a un problème. L'hypothèse "j'ai 1 euro" est insuffisante par ce qu'il faudrait "2 euros", c'est-à-dire deux fois cette hypothèse pour acheter à la fois une baguette et un croissant. En fait, dans toutes ces démonstrations, on a utilisé *deux fois* l'hypothèse sans le savoir.

Au niveau des démonstrations mathématiques cela nous convient parfaitement, mais, dans la sémantique, on s'intéresse de plus près à la gestion des ressources disponibles. C'est à cette fin que la logique linéaire [38] décompose plus finement les connecteurs de la logique, de la même façon que les substitutions explicites décomposent plus finement la β-réduction.

La logique linéaire propose en plus une notation graphique, appellée *réseau de preuve*, pour représenter ses démonstrations ($\mathcal{PN}$, pour Proof Net en anglais). La logique linéaire possède elle aussi une élimination des coupures dont la contrepartie dans les réseaux de preuve est un système de réécriture de graphe qui transforme les réseaux, de la même manière que l'élimination des coupures tranforme les démonstrations [10].

3.2.1 Définition

Pour une présentation plus complète de la logique linéaire et des réseaux de preuves, le lecteur pourra se reporter à [38]. On se place ici dans un fragment de la logique linéaire qui sera suffisant pour notre travail : la logique linéaire multiplicative et exponentielle (MELL [11]).

Séquents et connecteurs

La formulation de la logique linéaire est basée sur le calcul des séquents. Cela signifie que nous aurons une règle de coupure ainsi que des règles d'introduction à gauche et à droite des séquents. Cependant nous adoptons une présentation de la logique linéaire dans laquelle il n'y a aucune proposition à gauche du symbole $\vdash$ des séquents : toutes les propositions sont mises à droite. En logique linéaire, faire passer les propositions d'un côté à l'autre du symbole $\vdash$ correspond à la négation linéaire de la proposition. Cette négation linéaire est notée par le symbole $^{\perp}$. Par exemple, voici à gauche la règle *Axiome* telle qu'on l'a vue en calcul des séquents et à droite sa présentation en logique linéaire :

$$\frac{}{A \vdash A}\ \textit{Axiome} \qquad\qquad \frac{}{\vdash A^{\perp}, A}\ \textit{Axiome}$$

Pour résoudre le problème du nombre d'utilisation d'une hypothèse, la logique linéaire propose de séparer les hypothèses d'un séquent en deux catégories. Les hypothèses que l'on souhaite pouvoir

[10] Les réseaux de preuve ont servi de base aux recherches sur la géométrie de l'interaction [39, 3, 25], menant aux travaux sur les réductions optimales [2, 56].

[11] Multipicative and Exponential Linear Logic.

utiliser autant de fois que l'on veut seront "marquées" avec le symbole ? (qui se lit "pourquoi pas" en français, et "why not" en anglais). Les autres n'auront pas ce symbole. Par exemple, dans le séquent $\vdash ?A^{\perp}, B^{\perp}, C$ on peut utiliser autant de fois que l'on veut l'hypothèse $?A^{\perp}$ tandis qu'on ne peut utiliser qu'une seule fois l'hypothèse $B^{\perp}$.

Il y a quatre connecteurs logiques dans MELL : le ! (qui se lit "bien sûr" en français et "of course" en anglais), le "tenseur" $\otimes$ et le "par" $\parr$. La négation linéaire transforme le ? en !, le $\otimes$ en $\parr$, et réciproquement. Les équations suivantes indiquent comment la négation linéaire procède :

$$\begin{array}{ll} (A \otimes B)^{\perp} & = A^{\perp} \parr B^{\perp} \\ (?A)^{\perp} & = !(A^{\perp}) \\ (A \parr B)^{\perp} & = A^{\perp} \otimes B^{\perp} \\ (!A)^{\perp} & = ?(A^{\perp}) \end{array}$$

Propositions et règles

Pour le typage, nous avions besoin de *types de base* à partir desquels on pouvait construire nos types. De la même façon, nous avons besoin ici d'*atomes* à partir desquels on pourra construire nos propositions. La présence de la négation linéaire nous oblige à avoir deux catégories d'atomes, les atomes *positifs* et les atomes *négatifs* [12], car elle n'est pas un connecteur et ne fait que transformer les propositions. Si on veut utiliser la règle *Axiome* pour prouver une proposition atomique A :

$$\frac{}{\vdash A^{\perp}, A}\ \textit{Axiome}$$

il faut qu'on ait, dans le séquent, la proposition atomique $A^{\perp}$. Les atomes négatifs seront ceux qui possèdent le symbole $^{\perp}$.

Définition 3.2.1 (Propositions de la logique linéaire)

> Les propositions de MELL sont données par la grammaire suivante, où a est une proposition atomique (une formule) issue d'un ensemble non vide $\mathcal{A}$. De plus, $\mathcal{A}$ est l'union de deux ensembles disjoints $\mathcal{P}$ et $\mathcal{P}^{\perp}$, correspondant aux atomes dits *positifs* p et aux atomes dits *négatifs* $p^{\perp}$. En particulier, $p^{\perp}$ est considéré comme la *négation linéaire* de p et *vice versa*.
>
> $$F ::= a \mid F \otimes F \text{ (tenseur)} \mid F \parr F \text{ (par)} \mid !F \text{ (bien-sûr)} \mid ?F \text{ (pourquoi-pas)}$$

Nous avons dit que les hypothèses $?A$ peuvent être utilisées un nombre quelconque de fois. On intègre cela dans les règles de la logique linéaire avec la règle *Contraction* :

$$\frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\ \textit{Contraction}$$

Cette règle nous permet de dupliquer une hypothèse afin de s'en servir deux fois dans le reste de la preuve. En utilisant cette règle à nouveau, on peut obtenir un nombre quelconque de $?A$ dans le séquent. On peut aussi vouloir ne plus utiliser cette hypothèse, ce qui correspond à l'affaiblissement :

$$\frac{\vdash \Gamma}{\vdash \Gamma, ?A}\ \textit{Weakening}$$

Enfin, on peut transformer une hypothèse $?A$ en hypothèse A, perdant la possibilité de la dupliquer à nouveau ou de l'affaiblir. Cette règle s'appelle la *Dereliction* :

$$\frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\ \textit{Dereliction}$$

[12] Pour une description plus complète et plus précise de la polarisation en logique, on pourra consulter [57].

Définition 3.2.2

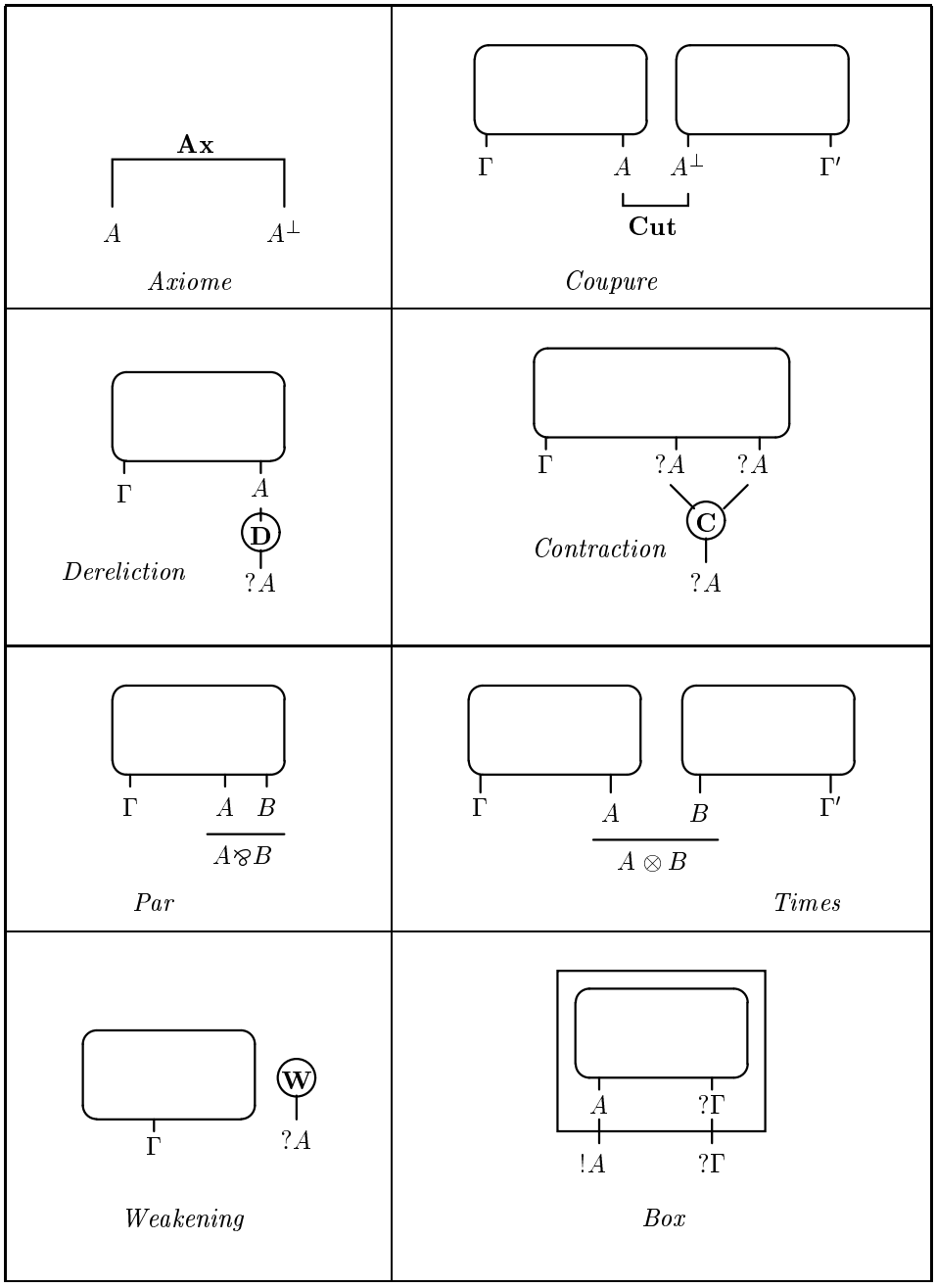
Voici les règles de la logique linéaire :

$\dfrac{}{\vdash A, A^{\perp}}$ *Axiome*	$\dfrac{\vdash \Gamma, A \quad \vdash A^{\perp}, \Delta}{\vdash \Gamma, \Delta}$ *Coupure*
$\dfrac{\vdash \Gamma, A}{\vdash \Gamma, ?A}$ *Dereliction*	$\dfrac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}$ *Contraction*
$\dfrac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}$ *Par*	$\dfrac{\vdash \Gamma, A \quad \vdash B, \Gamma'}{\vdash \Gamma, A \otimes B, \Gamma'}$ *Times*
$\dfrac{\vdash \Gamma}{\vdash \Gamma, ?A}$ *Weakening*	$\dfrac{\vdash A, ?\Gamma}{\vdash !A, ?\Gamma}$ *Box*

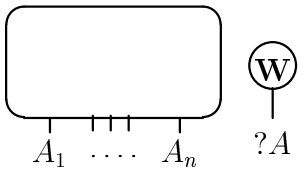
La règle *Par* correspond, dans la déduction naturelle, à l'introduction de l'implication, et, dans le calcul des séquents, à son introduction à droite. La règle *Times* correspond, dans la déduction naturelle, à l'élimination de l'implication, et, dans le calcul des séquents, à son introduction à gauche.

Réseaux de preuve

Comme nous l'avons dit, un des avantages de MELL est qu'il est possible de représenter un arbre de preuve sous la forme d'un graphe non-séquentiel. Par non-séquentiel, on veut dire que lorsque l'ordre d'application des règles n'est pas significatif, les réseaux de preuve "oublient" cet ordre. Plusieurs preuves correspondent donc à un même réseau. Voici les règles de construction inductive des réseaux de preuve :

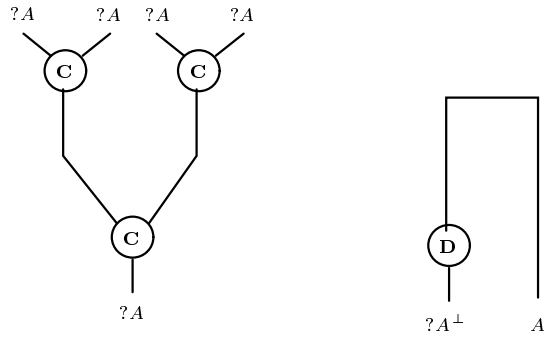


On s'est permis d'écrire directement Γ au bout d'un fil unique, alors qu'en réalité on devrait avoir autant de fils que de propositions dans Γ. Par exemple, le réseaux correspondant à la règle *Weakening* (où $\Gamma = A_1, ..., A_n$) devrait s'écrire :

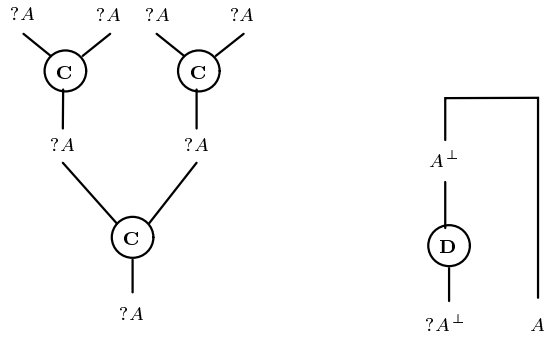


Pour rendre les dessins moins confus, on préfère conserver la notation avec Γ. Une autre simplification des dessins qu'on peut trouver consiste à ne pas écrire toutes les propositions dans certaines parties des réseaux. En particulier, si on a un arbre de nœuds correspondants à plusieurs applications successives de la règle *contraction*, on connecte directement ceux-ci sans indiquer la répétition de la formule contractée. De même, on a souvent l'application de la règle *Dereliction* juste après celle de

la règle *Axiome*. Dans ce cas, on ne fait pas apparaître l'hypothèse qui se trouve entre les deux. Par exemple, les réseaux suivants :



correspondent en réalité aux réseaux suivants :



On peut constater que le réseau une fois dessiné, ces informations sont superflues.

Élimination des coupures dans les réseaux de preuve

Comme dans le calcul des séquents, l'élimination des coupures consiste à supprimer les applications de la règle *Coupure* dans les démonstrations. Comme exemple d'élimination de coupure, on peut regarder le cas de la coupure avec un axiome. Prenons la preuve suivante :

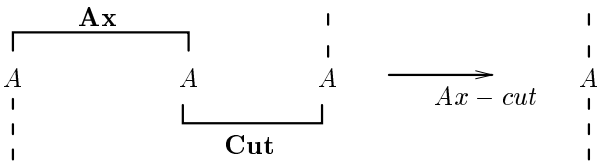
$$\dfrac{\dfrac{\pi}{\vdash A} \quad \dfrac{}{\vdash A^{\perp}, A}\ \textit{Axiome}}{\vdash A}\ \textit{Coupure}$$

Il y a un détour inutile, car on peut prouver le même séquent $\vdash A$ à partir de la preuve π sans appliquer la règle *Coupure*. Cela équivaut donc à la preuve :

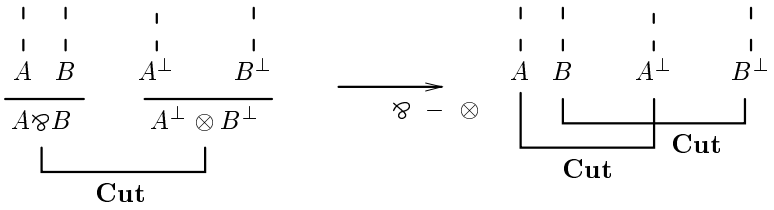
$$\dfrac{\pi}{\vdash A}$$

On peut refléter le processus d'élimination des coupures dans les réseaux de preuves, comme cela a été proposé dans [38], à l'aide des règles de réécriture ci-dessous. Elles constituent un système de réécriture de graphes. Les lignes de tirets qui apparaissent dans les règles représentent le reste du réseau qui se trouve autour du redex considéré. Cela correspond à la notion de contexte.

- Voici l'élimination de la coupure $Ax - cut$, qui élimine l'axiome :

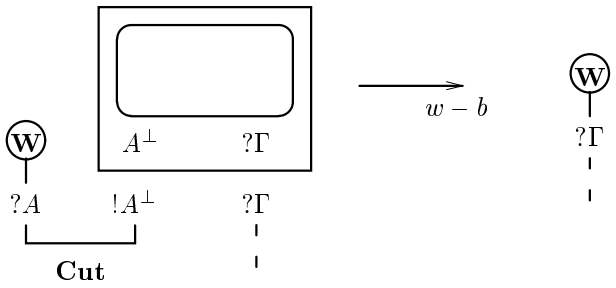


- Voici l'élimination de la coupure $\parr - \otimes$:

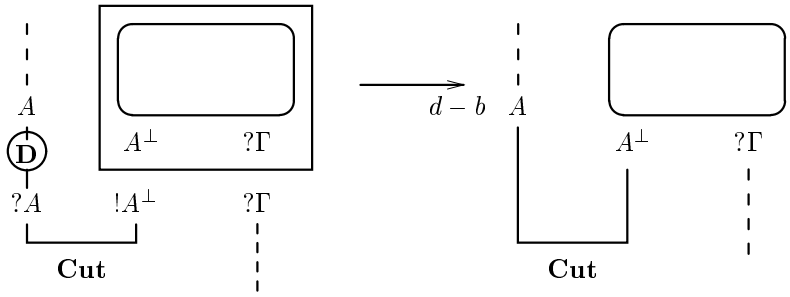


Cette règle fait apparaître deux nouvelles coupures, mais celles-ci portent sur des propositions plus petites.

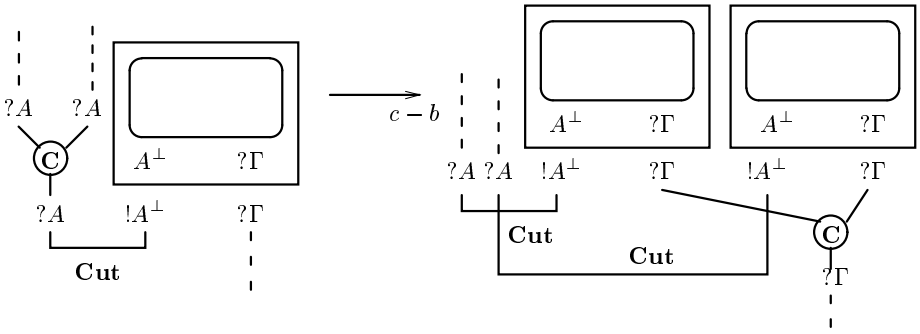
- Voici l'élimination de la coupure $w - b$, qui élimine une boîte :



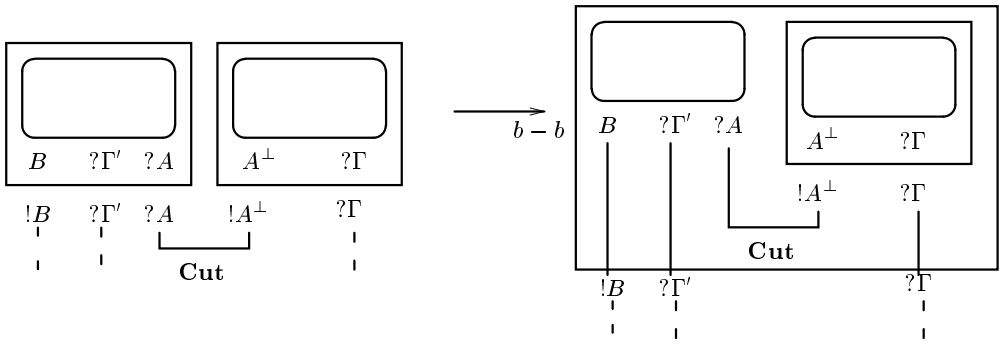
- Voici l'élimination de la coupure $d - b$, qui ouvre une boîte :



- Voici l'élimination de la coupure $c - b$, qui duplique une boîte :



- Voici l'élimination de la coupure $b - b$, qui fait entrer une boîte dans une autre :



Dans les réseaux de preuve, certaines preuves apparaissent différentes alors que l'on souhaiterait pouvoir les confondre. Des détails insignifiants tels que l'ordre suivant lequel on applique les règles *Contraction*, ou encore l'application de la règle *Contraction* à l'intérieur ou à l'extérieur d'une boîte, différencient des réseaux correspondant à des preuves très similaires. Pour nous débarasser de ces différences, nous utilisons une relation d'équivalence sur les réseaux, proposée dans [17]. Cette relation est composée de deux équivalences :

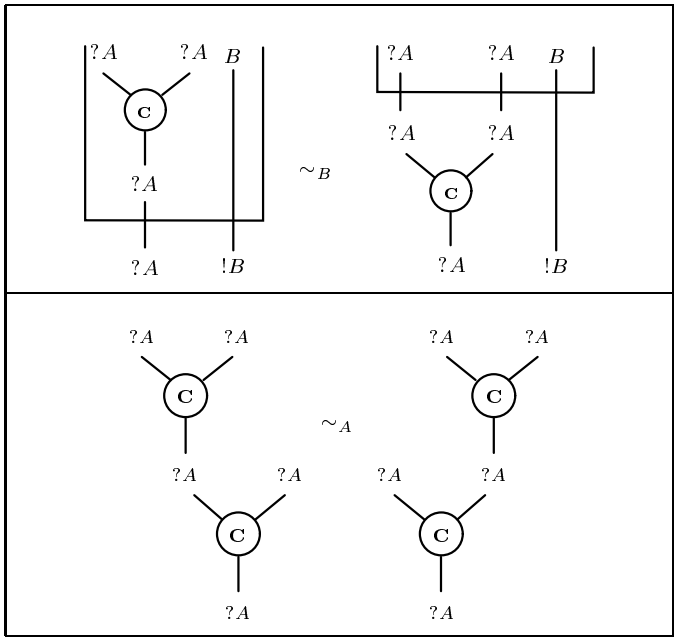
- La première est une équivalence de traversée des boîtes, (notée B). Dans MELL, les deux preuves ci-dessous sont équivalentes dans le sens où elles prouvent le même séquent :

$$\dfrac{\dfrac{\vdash ?A, ?A, B}{\vdash ?A, B}\ \textit{Contraction}}{\vdash ?A, !B}\ \textit{Box} \qquad \dfrac{\dfrac{\vdash ?A, ?A, B}{\vdash ?A, ?A, !B}\ \textit{Box}}{\vdash ?A, !B}\ \textit{Contraction}$$

- La deuxième est une équivalence d'associativité (noté A). Dans MELL, les deux preuves ci-dessous sont équivalentes (dans le séquent du milieu, on a mis entre crochet la formule sur laquelle on fait la contraction du dessus) :

$$\dfrac{\dfrac{\vdash ?A, ?A, ?A}{\vdash [?A], ?A}\ \textit{Contraction}}{\vdash ?A}\ \textit{Contraction} \qquad \dfrac{\dfrac{\vdash ?A, ?A, ?A}{\vdash ?A, [?A]}\ \textit{Contraction}}{\vdash ?A}\ \textit{Contraction}$$

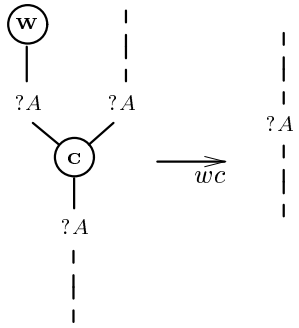
On ajoute donc les deux règles suivantes sur les réseaux de preuve :



Pour effectuer nos preuves de normalisation forte des calculs avec substitutions explicites, nous aurons besoin de deux règles de réduction supplémentaires. La deuxième, *wb*, sera présentée en détail dans la partie IV. La première, *wc* (introduite dans [18]), sert à éliminer les liens *Contraction* – *Weakening* qui sont en trop. Dans MELL, la preuve ci-dessous à gauche comporte un détour inutile par rapport à la preuve de droite :

$$\dfrac{\dfrac{\dfrac{\pi}{\vdash ?A}}{\vdash ?A, ?A}\ \textit{Weakening}}{\vdash ?A}\ \textit{Contraction} \qquad\qquad \dfrac{\pi}{\vdash ?A}$$

On en déduit la règle *wc* :



La figure 3.1 donne un exemple de réduction dans les réseaux de preuve.

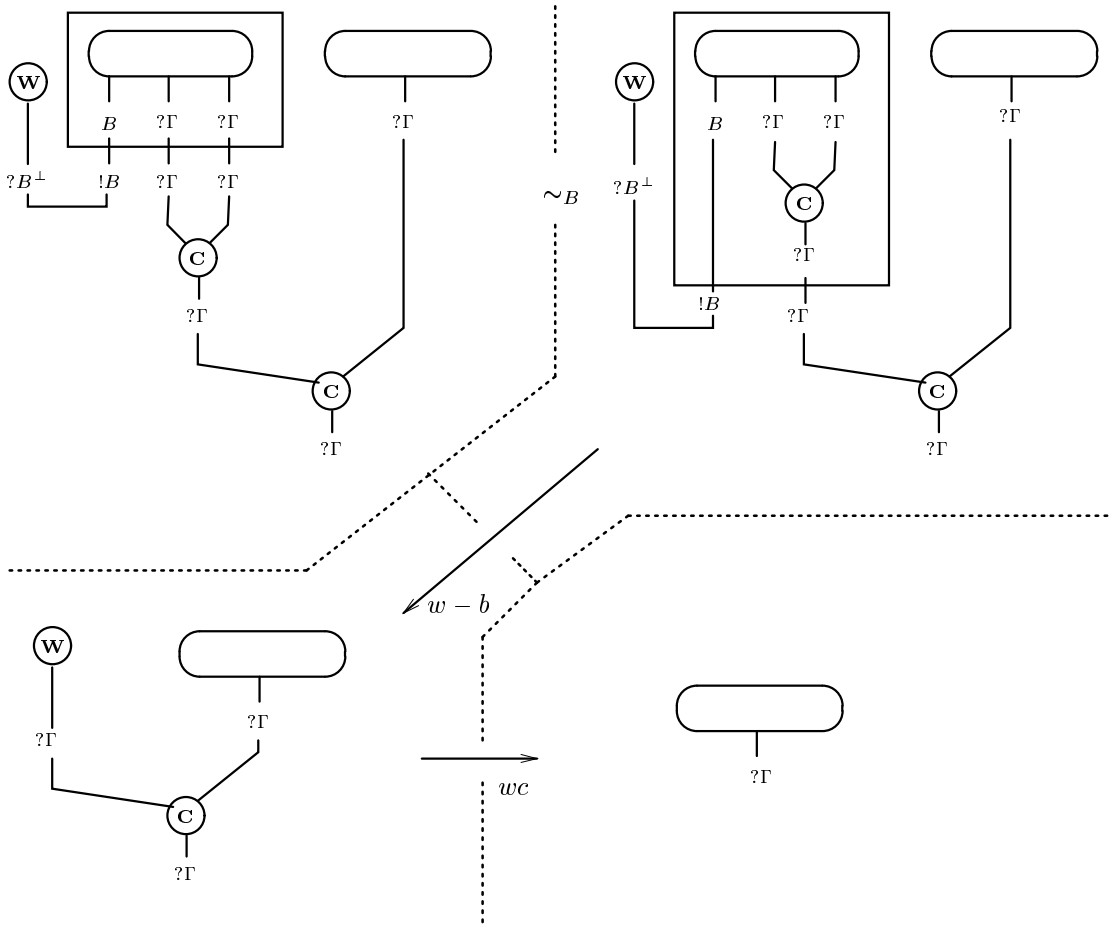


FIG. 3.1 – Exemple de réduction

Nous donnons les notations suivantes correspondant aux différents systèmes présentés ci-dessus.

Notation 3.2.3

On notera :

- $\mathcal{PN}$ l'élimination des coupures dans les réseaux de preuve de la logique linéaire,
- $\mathcal{W}$ le système composé des règles wc et wb (cette dernière sera présentée dans la partie IV),
- $\mathcal{R}$ l'union de $\mathcal{PN}$ et de $\mathcal{W}$.
- $\sim_E$ l'équivalence engendrée par l'union des équivalences $\sim_A$ et $\sim_B$,
- $\mathcal{PN}_E$ le système $\mathcal{PN}$ modulo $\sim_E$,
- $\mathcal{W}_E$ le système $\mathcal{W}$ modulo $\sim_E$,
- $\mathcal{R}_E$ l'union de $\mathcal{PN}_E$ et de $\mathcal{W}_E$.

3.2.2 Propriétés

Voici un bref rappel des propriétés de l'élimination des coupures dans les réseaux de preuve dont nous aurons besoin dans la suite de notre travail.

Théorème 3.2.4 (Normalisation forte et confluence de $\mathcal{PN}$)

> L'élimination des coupures dans les réseaux de preuve de la logique linéaire multiplicative et exponentielle est fortement normalisante et confluente.

Preuve : Voir [38]. ■

Théorème 3.2.5 (Normalisation forte de $\mathcal{PN}_E$)

> L'élimination des coupures dans les réseaux de preuve de la logique linéaire multiplicative et exponentielle modulo les équivalences A et B est fortement normalisante.

Preuve : Voir [17]. ■

Deuxième partie

PSN implique SN

Comme on l'a vu dans les chapitres 2 et 3, deux notions de terminaison coexistent pour les calculs avec substitutions explicites :

- préservation de la normalisation forte (PSN), qui dit que si un terme du calcul pur (Λ) est fortement normalisant (Λ_{SN}), alors il est aussi fortement normalisant (Λ^X_{SN}) dans le calcul avec substitutions explicites (Λ^X) étudié.
- normalisation forte (SN), qui dit que, étant donné un système de typage T, les termes typés du calcul avec substitutions explicites (Λ^X_T) sont fortement normalisants.

Ces deux propriétés ne sont pas redondantes, comme le montre la figure 3.2. PSN établit que le rectangle hachuré horizontalement et en diagonale est inclus dans le rectangle hachuré en diagonale, tandis que SN dit que le rectangle hachuré verticalement est inclus dans le rectangle hachuré en diagonale. Elle parlent donc de deux ensembles différents, mais il y a une partie commune : le rectangle hachuré verticalement et horizontalement, qui correspond aux termes typés du calcul pur. On utilisera ce fait par la suite.

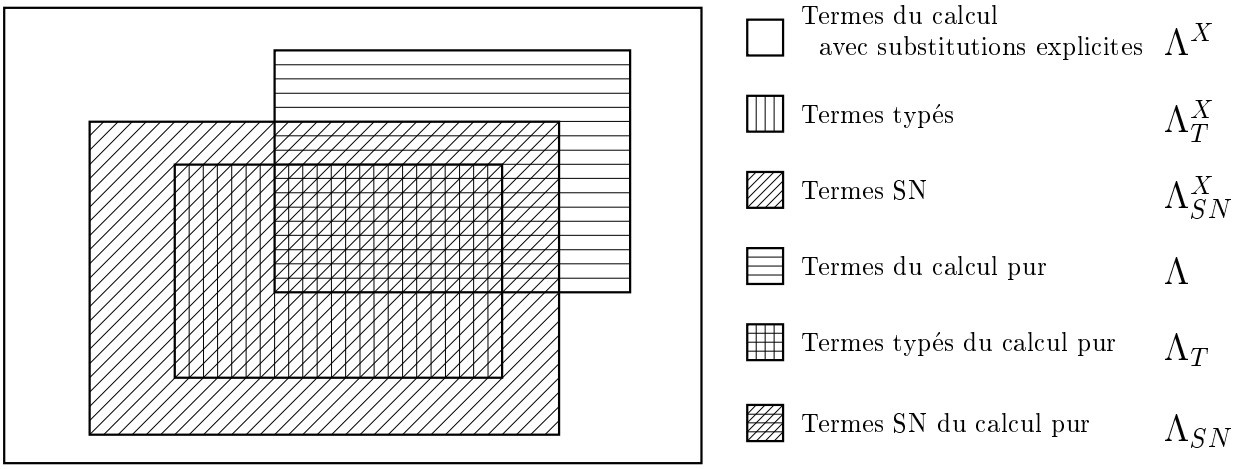


FIG. 3.2 – Termes du λ-calcul avec substitutions explicites et propriétés de normalisation

Voici les différentes relations qui existent entre les ensembles représentés figure 3.2. Toutes sauf la dernière sont évidentes par définition.

- Les termes purs typés sont des termes purs : $\Lambda_T \subset \Lambda$.
- Les termes purs fortement normalisants sont des termes purs : $\Lambda_{SN} \subset \Lambda$.
- Les termes avec substitutions typés sont des termes purs avec substitutions : $\Lambda^X_T \subset \Lambda^X$.
- Les termes purs avec substitutions fortement normalisants sont des termes purs avec substitutions : $\Lambda^X_{SN} \subset \Lambda^X$.
- Les termes purs font partie des termes avec substitutions : $\Lambda \subset \Lambda^X$.
- Les termes purs typés font partie des termes avec substitutions typés : $\Lambda_T \subset \Lambda^X_T$.
- Les termes purs typés sont fortement normalisants : $\Lambda_T \subset \Lambda_{SN}$. C'est le théorème 3.1.13.

Le diagramme présenté figure 3.3 récapitule ces relations. Le symbole d'appartenance $\subset$ y est remplacé par la flèche ($A \subset B$ est noté $A \to B$). Pour fermer le diagramme, il reste à prouver PSN : $\Lambda_{SN} \subset \Lambda^X_{SN}$ et SN : $\Lambda^X_T \subset \Lambda^X_{SN}$, ce qui clôt le diagramme, nous donnant celui de la figure 3.4.

Même si SN et PSN sont deux propriétés de terminaison, il n'y a pas toujours de relation claire entre leurs preuves : SN est parfois prouvé indépendamment de PSN (voir, par exemple, [18, 28]), et parfois la preuve de SN utilise PSN (voir, par exemple, [11]). On présente, dans cette partie, la formalisation d'une technique de preuve de SN *via* PSN suggérée initialement par H. Herbelin.

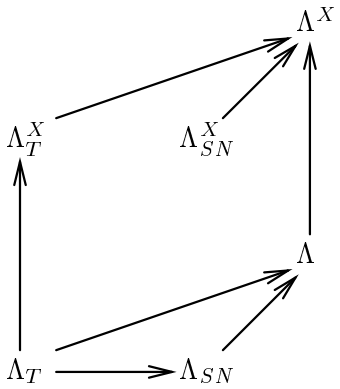


FIG. 3.3 – Propriétés de normalisation en notation ensembliste

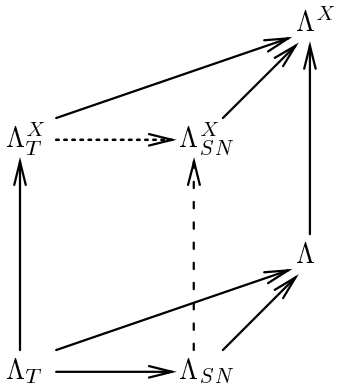


FIG. 3.4 – Propriétés de normalisation en notation ensembliste

Le chapitre 4 présente la technique et ses différents modes d'utilisation. Dans les chapitres suivants, on applique cette technique à différents calculs.

Une présentation courte de ce travail a été acceptée pour publication dans [65].

Chapitre 4

Technique de démonstration

Dans ce chapitre, on explicite en détail la technique de démonstration que l'on souhaite introduire. On commence par donner les bases et l'intuition de la technique, puis on regarde différents cas de démonstrations.

4.1 L'antiréduction *Ateb*

L'idée de cette technique est la suivante : étant donné un terme t de Λ_T^X, on construit, à l'aide de sa dérivation de typage, un terme t' appartenant à Λ_T. Pour cela, on "remonte" les substitutions du terme t pour recréer les $Beta$-redex (on nomme la fonction effectuant cette remontée $Ateb$). t' peut donc se réduire par des applications de la règle $Beta$ vers t. Puisque t' appartient à Λ_T, il appartient aussi à Λ_{SN}. Par la propriété PSN on a $t' \in \Lambda_{SN}^X$, et comme toutes les réductions de t' terminent, y compris celle qui passe par t, on obtient $t \in \Lambda_{SN}^X$.

En pratique, le scénario de démonstration ne se déroulera pas de façon aussi idéale. Nous allons étudier cela dans les paragraphes suivants.

4.2 Démonstration directe

Le cas le plus agréable est le suivant : il existe une fonction $Ateb$ de Λ_T^X vers Λ_T qui vérifie la propriété suivante.

Propriété 4.2.1 (Initialisation)

> Pour tout $t \in \Lambda_T^X$, $Ateb(t) \to^* t$.

On peut alors conclure directement à la forte normalisation du calcul, c'est-à-dire $\Lambda_T^X \subset \Lambda_{SN}^X$. Ce scénario fonctionnera avec les calculs $\lambda\mathbf{x}$ et λ_{wsn}.

4.3 Démonstration par simulation

Si il n'existe pas de fonction qui vérifie la propriété précédente, on cherche une fonction $Ateb$ qui permette d'initialiser une simulation : on essaie de trouver un terme $u' \in \Lambda_T$ tel que u' se réduit vers u et tel qu'il existe une simulation des réductions de t par des réductions de u. Si c'est possible, on peut alors conclure à la forte normalisation de t grâce à la forte normalisation que u obtient de u'.

Pour effectuer la simulation, on procède de la façon suivante. On commence par séparer les règles de réduction du calcul en deux parties $R_1 \cup R_2$ telles que R_2 soit fortement normalisant. On construit ensuite une relation $\mathcal{R}$ sur les termes de Λ_T^X qui vérifie les propriétés suivantes.

Propriété 4.3.1 (Initialisation)

Pour tout $t \in \Lambda_T^X$, il existe $u \in \Lambda_T^X$ tel que $Ateb(t) \to^* u$ et $u\mathcal{R}t$.

$$\begin{array}{cc} & t \\ & \swarrow \ \mathcal{R} \\ Ateb(t) \to^* & u \end{array}$$

Propriété 4.3.2 (Simulation 1)

Pour tout $t \in \Lambda_T^X$, si $t \to_{R_1} t'$, alors pour tout $u\mathcal{R}t$ il existe u' tel que $u \to^+ u'$ et $u'\mathcal{R}t'$.

$$\begin{array}{ccc} t & \to_{R_1} & t' \\ \mathcal{R} & & \mathcal{R} \\ u & \to^+ & u' \end{array}$$

Propriété 4.3.3 (Simulation 2)

Pour tout $t \in \Lambda_T^X$, si $t \to_{R_2} t'$, alors pour tout $u\mathcal{R}t$ il existe u' tel que $u \to^* u'$ et $u'\mathcal{R}t'$.

$$\begin{array}{ccc} t & \to_{R_2} & t' \\ \mathcal{R} & & \mathcal{R} \\ u & \to^* & u' \end{array}$$

Un cas particulier de cette technique de simulation est celui où la relation $\mathcal{R}$ est une fonction. Les preuves des lemmes de simulation sont nettement simplifiées car le "pour tout $u\mathcal{R}t$" est remplacé par "pour l'unique $u = \mathcal{R}(t)$". Cependant, pour les calculs que nous étudions dans la suite de notre travail, ce cas particulier ne peut pas être appliqué.

4.4 Le théorème principal

On énonce à présent les théorèmes qui mettent en place cette technique de démonstration.

Théorème 4.4.1 (Démonstration directe)

Pour tout système de typage T tel que $\Lambda_T \subset \Lambda_{SN}$, si il existe une fonction $Ateb$ de Λ_T^X vers Λ_T vérifiant la propriété 4.2.1 alors $\Lambda_{SN} \subset \Lambda_{SN}^X \Rightarrow \Lambda_T^X \subset \Lambda_{SN}^X$.

Preuve : Pour tout terme $t \in \Lambda_T^X$, $Ateb(t) \in \Lambda_T$. Par hypothèse (SN du λ-calcul typé), on a $Ateb(t) \in \Lambda_{SN}$. Par hypothèse (PSN) on a $Ateb(t) \in \Lambda_{SN}^X$. Par la propriété 4.2.1, on a $Ateb(t) \to^* t$, ce qui nous donne directement $t \in \Lambda_{SN}^X$. ■

Théorème 4.4.2 (Démonstration par simulation)

Pour tout système de typage T tel que $\Lambda_T \subset \Lambda_{SN}$, si il existe une fonction $Ateb$ de Λ_T^X vers Λ_T vérifiant la propriété 4.3.1, si les règles de réductions peuvent être séparées en $R_1 \cup R_2$ avec R_2 fortement normalisant, et si il existe une relation $\mathcal{R}$ vérifiant les propriétés 4.3.2 et 4.3.3, alors $\Lambda_{SN} \subset \Lambda_{SN}^X \Rightarrow \Lambda_T^X \subset \Lambda_{SN}^X$.

Preuve : On effectue une preuve par l'absurde. Supposons qu'il existe un terme $t \in \Lambda_T^X$ d'où parte une suite infinie de réductions. Par la propriété 4.3.1 il existe $u \in \Lambda_T^X$ tel que $Ateb(t) \to^* u$. Par définition, $Ateb(t) \in \Lambda_T$, et par hypothèse (SN du λ-calcul typé), on a $Ateb(t) \in \Lambda_{SN}$. Par hypothèse (PSN) on a $Ateb(t) \in \Lambda_{SN}^X$ ce qui nous donne $u \in \Lambda_{SN}^X$. La suite de réductions infinie partant de t peut être décomposée en une suite infinie de réductions alternant R_1 et R_2. Par hypothèse, R_2 est fortement normalisant, ce qui implique que la suite de réduction infinie doit être de la forme :

$$t \quad \rightarrow^*_{R_2} \quad t_1 \quad \rightarrow^+_{R_1} \quad t_2 \quad \rightarrow^*_{R_2} \quad t_3 \quad \rightarrow^+_{R_1} \quad t_4 \quad \rightarrow^*_{R_2} \quad t_5 \quad \rightarrow^+_{R_1} \quad t_6 \quad \rightarrow ...$$

Grâce à la propriété 4.3.1, on a $u\mathcal{R}t$ et grâce aux propriétés 4.3.2 et 4.3.3 on peut construire une suite de réductions partant de u de la façon suivante :

$$\begin{array}{rccccccccccccccc}
 & t & \rightarrow^*_{R_2} & t_1 & \rightarrow^+_{R_1} & t_2 & \rightarrow^*_{R_2} & t_3 & \rightarrow^+_{R_1} & t_4 & \rightarrow^*_{R_2} & t_5 & \rightarrow^+_{R_1} & t_6 & \rightarrow ... \\
\swarrow & \mathcal{R} & & \mathcal{R} & & \mathcal{R} & & \mathcal{R} & & \mathcal{R} & & \mathcal{R} & & \mathcal{R} & \\
Ateb(t) \rightarrow^* & u & \rightarrow^* & u_1 & \rightarrow^+ & u_2 & \rightarrow^* & u_3 & \rightarrow^+ & u_4 & \rightarrow^* & u_5 & \rightarrow^+ & u_6 & \rightarrow ...
\end{array}$$

La suite de réductions partant de u est infinie, ce qui contredit $u \in \Lambda^X_{SN}$. ■

Pour chacun des calculs auxquels nous appliquerons cette technique de démonstration, nous commencerons par rappeler brièvement leur définition.

Chapitre 5

Application à $\lambda\mathtt{x}$, λ_{wsn}, λ_{ws} et $\overline{\lambda}\mu\tilde{\mu}\mathtt{x}$

On regroupe dans ce chapitre les applications faciles de la technique, ainsi que l'échec en ce qui concerne λ_{ws}.

5.1 Application à $\lambda\mathtt{x}$

Le calcul $\lambda\mathtt{x}$ [68, 11] est le plus simple calcul avec substitutions explicites. Il se contente de rendre explicite l'opérateur implicite de substitution, sans ajouter d'autres règles de réduction.

5.1.1 Présentation du calcul

Les termes du $\lambda\mathtt{x}$-calcul sont donnés par la grammaire suivante :

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

Voici l'ensemble des règles de réduction :

$$\begin{array}{rll} (\lambda x.t)u & \rightarrow_{Beta} & t[u/x] \\ (t\ u)[v/x] & \rightarrow_{App} & (t[v/x])\ (u[v/x]) \\ (\lambda x.t)[u/y] & \rightarrow_{Lambda} & \lambda x.(t[u/y]) \\ x[t/x] & \rightarrow_{Var1} & t \\ y[t/x] & \rightarrow_{Var2} & y \end{array}$$

La règle *Lambda* s'effectue modulo α-conversion sur la variable liée x. Regardons les règles de typage :

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma \vdash u : B \quad \Gamma, x : B \vdash t : A}{\Gamma \vdash t[u/x] : A}$$

$$\frac{\Gamma \vdash t : B \rightarrow A \quad \Gamma \vdash u : B}{\Gamma \vdash (t\ u) : A} \qquad \frac{\Gamma, x : B \vdash t : A}{\Gamma \vdash \lambda x.t : B \rightarrow A}$$

5.1.2 Preuve de normalisation forte

On définit la fonction $Ateb$ de la façon suivante :

$$\begin{array}{lcl} Ateb(x) & = & x \\ Ateb(t\ u) & = & Ateb(t)\ Ateb(u) \\ Ateb(\lambda x.t) & = & \lambda x.Ateb(t) \\ Ateb(t[u/x]) & = & (\lambda x.Ateb(t))\ Ateb(u) \end{array}$$

On remarque que la fonction $Ateb$ effectue de la réécriture inverse pour la règle $Beta$. Il est évident que si $t' = Ateb(t)$ alors $t' \rightarrow^*_{Beta} t$. Il est évident aussi que pour tout t, $Ateb(t)$ ne contient pas de substitutions. Il nous faut vérifier que le terme obtenu est typable.

Lemme 5.1.1

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash Ateb(t) : A$$

Preuve : On procède par induction sur la dérivation de typage de t. Le seul cas non immédiat est celui de la substitution. On a $t = u[v/x]$ et

$$\frac{\Gamma \vdash v : B \quad \Gamma, x : B \vdash u : A}{\Gamma \vdash u[v/x] : A}$$

Par hypothèse d'induction, on a $\Gamma, x : B \vdash Ateb(u) : A$ et $\Gamma \vdash Ateb(v) : B$. On peut typer $Ateb(t) = (\lambda x.Ateb(u))\ Ateb(v)$ de la façon suivante

$$\frac{\dfrac{\Gamma, x : B \vdash Ateb(u) : A}{\Gamma \vdash \lambda x.Ateb(u) : B \to A} \quad \Gamma \vdash Ateb(v) : B}{\Gamma \vdash (\lambda x.Ateb(u))\ Ateb(v) : A}$$

■

On peut établir le théorème principal de ce chapitre.

Théorème 5.1.2

Le $\lambda\mathbf{x}$-calcul est fortement normalisant.

Preuve : Comme on a les propriétés de PSN pour $\lambda\mathbf{x}$ [12] et de SN du λ-calcul simplement typé [55], on peut utiliser le théorème 4.4.1. ■

5.2 Application à λ_{wsn}

Le λ_{wsn}-calcul est une version avec noms de variables du λ_{ws}-calcul [26, 41, 28], introduite dans [18], et qui sera présenté dans la quatrième partie de cette thèse. Par rapport à $\lambda\mathbf{x}$, il apporte des règles de réécriture pour gérer la composition des substitutions, ainsi qu'un mécanisme d'affaiblissement explicite (appelé étiquette) qui permet de préserver la propriété PSN avec ces nouvelles règles. De plus, les affaiblissements explicites permettent de se passer d'α-conversion.

5.2.1 Présentation du calcul

Les termes du λ_{wsn}-calcul sont donnés par la grammaire suivante :

$$t ::= x \mid (t\ t) \mid \lambda x.t \mid t[x, t, \Gamma, \Gamma] \mid \Gamma t$$

où Γ est un ensemble de variables. Une version des règles de réduction est présenté Fig. 5.1.

Voici les règles de typage :

$$\frac{}{x : A \vdash x : A}\ Ax \qquad \frac{\Gamma \setminus \Delta \vdash t : A \quad \Delta \subset \Gamma}{\Gamma \vdash \Delta t : A}\ Weak$$

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

$$\frac{\Pi \setminus \Gamma \vdash u : A \quad \Pi \setminus \Delta, x : A \vdash t : B \quad (\Gamma \cup \Delta) \subset \Pi}{\Pi \vdash t[x, u, \Gamma, \Delta] : B}\ Sub$$

$$
\begin{array}{lrcll}
(b) & (\Delta(\lambda x.t))\ (\Gamma u) & \to & t[x,u,\Gamma,\Delta] & \\
 & & & & \\
(a) & (t\ u)[x,v,\Gamma,\Delta] & \to & (t[x,v,\Gamma,\Delta]\ u[x,v,\Gamma,\Delta]) & \\
(e_1) & (\Lambda t)[x,u,\Gamma,\Delta] & \to & (\Delta \cup (\Lambda \setminus \{x\}))t & x \in \Lambda \\
(n_1) & y[x,t,\Gamma,\Delta] & \to & \Delta y & x \neq y \\
(n_2) & x[x,t,\Gamma,\Delta] & \to & \Gamma t & \\
(c_1) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & & x \in \Phi \setminus \Lambda \\
 & \multicolumn{3}{r}{t[y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})]} & \\
(c_2) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)] & \\
 & & & [y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi] & x \notin \Phi \cup \Lambda \\
(c_3) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[y,u,(\Lambda \setminus \{x\}) \cup \Delta, (\Phi \setminus \{x\}) \cup \Delta] & x \in \Phi \cap \Lambda \\
 & & & & \\
(f) & (\lambda y.t)[x,u,\Gamma,\Delta] & \sim & \lambda y.t[x,u,\Gamma \cup \{y\}, \Delta] & \\
(e_2) & (\Lambda t)[x,u,\Gamma,\Delta] & \sim & (\Gamma \cap \Lambda)t[x,u,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] & x \notin \Lambda \\
(d) & \Gamma\Delta t & \sim & (\Gamma \cup \Delta)t & \\
(\emptyset) & \emptyset t & \sim & t & \\
 & & & & \\
(c_4) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \sim & & x \in \Lambda \setminus \Phi \\
 & \multicolumn{3}{r}{t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y,u,\Delta \cup (\Lambda \setminus \{x\}), \Gamma \cap \Phi]} &
\end{array}
$$

FIG. 5.1 – Règles de réduction du λ_{wsn}-calcul

5.2.2 PSN implique SN

On définit la fonction $Ateb$ de la façon suivante :

$$
\begin{array}{lcl}
Ateb(x) & = & x \\
Ateb(t\ u) & = & Ateb(t)\ Ateb(u) \\
Ateb(\lambda x.t) & = & \lambda x.Ateb(t) \\
Ateb(\Gamma t) & = & \Gamma Ateb(t) \\
Ateb(t[x,u,\Gamma,\Delta]) & = & (\Delta(\lambda x.Ateb(t)))\ (\Gamma Ateb(u))
\end{array}
$$

Remarque 5.2.1

La fonction $Ateb$ envoie les termes du λ_{wsn}-calcul vers le λ-calcul avec affaiblissement explicite. Il faudrait avoir les résultats de normalisation pour ce calcul avant d'appliquer le théorème.

De la même façon que pour $\lambda\mathtt{x}$, la fonction $Ateb$ effectue de la réécriture inverse pour la règle b. Il est évident que si $t' = Ateb(t)$ alors $t' \to_b^* t$. Il est évident aussi que pour tout t, $Ateb(t)$ ne contient pas de substitutions. Il nous faut vérifier que le terme obtenu est typable.

Lemme 5.2.2

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash Ateb(t) : A$$

Preuve : On procède par induction sur la dérivation de typage de t. Le seul cas non immédiat est celui de la substitution. On a $t = u[x,v,\Gamma,\Delta]$ et

$$\frac{\Pi \setminus \Gamma \vdash v : B \quad \Pi \setminus \Delta, x : B \vdash u : A}{\Pi \vdash u[x,v,\Gamma,\Delta] : A}$$

Par hypothèse d'induction, on a $\Pi \setminus \Delta, x : B \vdash Ateb(u) : A$ et $\Pi \setminus \Gamma \vdash Ateb(v) : B$. On peut typer $Ateb(t) = (\Delta(\lambda x.Ateb(u)))\ \Gamma Ateb(v)$ de la façon suivante

$$\dfrac{\dfrac{\dfrac{\Pi \setminus \Delta, x : B \vdash Ateb(u) : A}{\Pi \setminus \Delta \vdash \lambda x.Ateb(u) : B \to A} \quad \dfrac{\Pi \setminus \Gamma \vdash Ateb(v) : B}{\Pi \vdash \Gamma Ateb(v) : B}}{\Pi \vdash \Delta(\lambda x.Ateb(u)) : B \to A}}{\Pi \vdash (\Delta(\lambda x.Ateb(u)))\ \Gamma Ateb(v) : A}$$

■

On peut donc appliquer le théorème 4.4.1. Néanmoins, on n'en tire pas de conclusion de normalisation forte puisque PSN n'a pas encore été démontrée pour ce calcul.

5.3 Application à λ_{ws}

Nous allons maintenant nous attaquer au calcul avec indices. Des difficultés vont venir de la rigidité de l'environnement de typage dans les séquents des dérivations de typage des termes avec indices. En effet, comme les indices sont des entiers qui se rapportent à un ordre dans l'environnement de typage, nous ne pourrons pas modifier ce dernier sans précautions. On commence par rappeler les règles de réduction (Fig. 5.2) et de typage (Fig. 5.3) du λ_{ws}-calcul où l'on suppose que $|\Gamma| = i$ et $|\Delta| = j$.

b_1	$(\lambda t u)$	$\to$	$[0/u, 0]t$	
b_2	$(\langle k\rangle \lambda t u)$	$\to$	$[0/u, k]t$	
f	$[i/u, j]\lambda t$	$\to$	$\lambda[i+1/u, j]t$	
a	$[i/u, j](t\ v)$	$\to$	$(([i/u, j]t)\ ([i/u, j]v))$	
e_1	$[i/u, j]\langle k\rangle t$	$\to$	$\langle j + k - 1\rangle t$	$si\ i < k$
e_2	$[i/u, j]\langle k\rangle t$	$\to$	$\langle k\rangle [i - k/u, j]t$	$si\ i \geq k$
n_1	$[i/u, j]k$	$\to$	k	$si\ i > k$
n_2	$[i/u, j]i$	$\to$	$\langle i\rangle u$	
n_3	$[i/u, j]k$	$\to$	$j + k - 1$	$si\ i < k$
c_1	$[i/u, j][k/v, l]t$	$\to$	$[k/[i - k/u, j]v, j + l - 1]t$	$si\ k \leq i < k + l$
c_2	$[i/u, j][k/v, l]t$	$\to$	$[k/[i - k/u, j]v, l][i - l + 1/u, j]t$	$si\ i \geq k + l$
d	$\langle i\rangle\langle j\rangle t$	$\to$	$\langle i + j\rangle t$	

FIG. 5.2 – Règles de réduction de λ_{ws}

$$\dfrac{}{\Gamma, A, \Delta \vdash i : A}\ Axiome$$

$$\dfrac{B, \Gamma \vdash t : C}{\Gamma \vdash \lambda t : B\ C}\ Lambda \qquad \dfrac{\Gamma \vdash t : B\ A \quad \Gamma \vdash u : B}{\Gamma \vdash (tu) : A}\ App$$

$$\dfrac{\Delta, \Pi \vdash u : A \quad \Gamma, A, \Pi \vdash t : B}{\Gamma, \Delta, \Pi \vdash [i/u, j]t : B}\ Subst \qquad \dfrac{\Delta \vdash t : B}{\Gamma, \Delta \vdash \langle i\rangle t : B}\ Weak$$

FIG. 5.3 – Règles de typage de λ_{ws}

Lors du typage de l'abstraction, le type du paramètre vient se mettre *devant* le reste de l'environnement de typage. Or, pour calquer le typage de la substitution, on voudrait que le type du paramètre s'insère *entre* Γ et Π. Le seul cas où cela fonctionne est celui où $i = 0$. Il faudra donc commencer par définir une fonction qui transforme tous les termes $[i/u, j]t$ en $[0/u', j]t'$ où t' et u' prennent en compte les modification de l'indice i en 0. Cependant, pour écrire une telle fonction, il faut penser au cas où t est de la forme $\langle k\rangle v$. En effet, supposons que k soit inférieur à i, si la transformation de t donne $\langle k'\rangle v'$, alors la réduction $[i/u, j]\langle k\rangle t \to \langle k\rangle [i - k/u, j]t$ n'est plus possible dans le terme

transformé $[0/u', j]\langle k'\rangle v'$ où k' est forcément plus grand que 0. Cela signifie qu'on doit tout d'abord se débarrasser des étiquettes (ainsi que mettre à 0 la partie droite des substitutions qui correspond à une étiquette). Supposons l'existence d'une fonction F qui réalise ce travail, il nous faut à présent une fonction G qui mette à 0 les indices de la partie gauche des substitutions. La fonction $Ateb$ n'a plus qu'à appeler ces deux fonctions et à "remonter" les substitutions qui sont désormais de la bonne forme.

Le problème vient de la simulation des réductions du terme initial par le terme obtenu par la fonction $Ateb$. En effet, à cause de la règle f, les substitutions ne conserveront pas un 0 en partie gauche après réduction. À cause de la règle n_2, des étiquettes vont apparaître dans les termes. Et lorsque nous essaierons de simuler l'application de la règle e_1, c'est-à-dire $[i/u, j]\langle k\rangle t \to \langle k\rangle[i-k/u, j]t$ avec $i \geq k$, nous serons démuni face à la question : "Dans tout terme de la forme $[i'/u', j']\langle k'\rangle t'$ en relation avec $[i/u, j]\langle k\rangle t$, a-t-on $i' \geq k'$?" L'établissement d'une telle propriété, qui, de plus, soit préservée par réduction, semble difficile et c'est la pierre d'achoppement de notre tentative.

5.4 Application au $\overline{\lambda}\mu\tilde{\mu}$-calcul avec substitutions explicites

Le $\overline{\lambda}\mu\tilde{\mu}$-calcul est un calcul symétrique non-déterministe issu de la logique classique. Il permet de représenter sous forme de termes les preuves de la logique classique dans le calcul des séquents. On peut lui ajouter des substitutions explicites "à la" $\lambda\mathtt{x}$. La troisième partie de cette thèse présente ce calcul et applique notre technique afin d'en démontrer la normalisation forte.

Chapitre 6

Application à $\lambda\upsilon$

Le calcul $\lambda\upsilon$ [59] est un calcul avec indices de De Bruijn et substitutions explicites. Il est presque aussi simple que $\lambda\mathbf{x}$ dans le sens ou $\lambda\upsilon$ ne possède pas de règles pour la composition des substitutions. Néanmoins, nous aurons plus de difficultés à appliquer notre méthode à cause des indices de De Bruijn.

6.1 Présentation du calcul

Les termes du $\lambda\upsilon$-calcul sont données par la grammaire suivante :

$$\begin{array}{l} t ::= n \mid (t\ t) \mid \lambda t \mid t[s] \\ s ::= a/ \mid \Uparrow (s) \mid \uparrow \end{array}$$

On remarque qu'une substitution est toujours composée d'une liste de $\Uparrow$ (parfois vide) suivie soit d'un $t/$, soit d'un $\uparrow$. On se permettra d'écrire les substitutions sous cette forme généralisée : soit $t[\Uparrow^i (t/)]$, soit $t[\Uparrow^i (\uparrow)]$, où $\Uparrow^i (s)$ est une notation pour $\underbrace{\Uparrow (\Uparrow (...(\Uparrow}_{i}(s))...))$. Voici l'ensemble des règles de réduction :

$$\begin{array}{lll} (\lambda t)u & \rightarrow_B & t[u/] \\ (t\ u)[s] & \rightarrow_{App} & (t[s])\ (u[s]) \\ (\lambda t)[s] & \rightarrow_{Lambda} & \lambda(t[\Uparrow (s)]) \\ 1[t/] & \rightarrow_{FVar} & t \\ n+1[t/] & \rightarrow_{RVar} & n \\ 1[\Uparrow (s)] & \rightarrow_{FVarLift} & 1 \\ n+1[\Uparrow (s)] & \rightarrow_{RVarLift} & n[s][\uparrow] \\ n[\uparrow] & \rightarrow_{VarShift} & n+1 \end{array}$$

Remarque 6.1.1

> Pour typer les substitutions nous introduisons une nouvelle forme de séquent. Les substitutions explicites modifient l'environnement de typage, soit en ajoutant une formule (correspondant au type du substituant), soit en supprimant un formule (avec le $\uparrow$), soit en permutant des formules (avec le $\Uparrow$). Nous écrirons $\Gamma \vdash s \triangleright \Gamma'$ pour dire que, dans l'environnement Γ, s produit un nouvel environnement Γ'. Cette notation se retrouvera dans le typage du $\lambda\sigma$-calcul et du $\lambda\sigma_n$-calcul.

Regardons les règles de typage (où $n = |\Gamma| + 1$) :

$$\frac{}{\Gamma, A, \Delta \vdash n : A} \qquad \frac{\Gamma \vdash s \triangleright \Gamma' \quad \Gamma' \vdash t : A}{\Gamma \vdash t[s] : A}$$

$$\frac{\Gamma \vdash t : B \rightarrow A \quad \Gamma \vdash u : B}{\Gamma \vdash (t\ u) : A} \qquad \frac{B, \Gamma \vdash t : A}{\Gamma \vdash \lambda t : B \rightarrow A}$$

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t/ \triangleright A, \Gamma} \qquad \frac{}{A, \Gamma \vdash \uparrow \triangleright \Gamma} \qquad \frac{\Gamma \vdash s \triangleright B, \Gamma}{A, \Gamma \vdash \Uparrow (s) \triangleright A, B, \Gamma}$$

6.2 Preuve de normalisation forte

On définit la fonction $Ateb$ de la façon suivante :

$$\begin{array}{lcl} Ateb(n) & = & n \\ Ateb(t\ u) & = & Ateb(t)\ Ateb(u) \\ Ateb(\lambda t) & = & \lambda Ateb(t) \\ Ateb(t[u/]) & = & (\lambda Ateb(t))\ Ateb(u) \\ Ateb(t[\Uparrow^i (u/)]) & = & (\lambda I_i(Ateb(t)))\ K_i(Ateb(u)) \\ Ateb(t[\Uparrow^i (\uparrow)]) & = & J_i(Ateb(t)) \end{array}$$

Exemple 6.2.1

Par exemple, si on suppose que pour tout x parmi t, u, v, w on a $x = Ateb(x)$, alors on a

$$Ateb((t[u/]\ v[\Uparrow (\Uparrow (\Uparrow (w/)))])[\Uparrow (\Uparrow (\uparrow))]) \quad = \quad J_2(((\lambda t)u)\ ((\lambda I_3(v))K_3(w)))$$

Où $I_i(t))$, $K_i(t)$ et $J_i(t)$ sont des fonctions que nous allons définir dans la suite. Toutes les substitutions effectuent du réindiçage dans le terme auquel elles sont appliquées. L'idée de ces trois fonctions est d'anticiper ce réindiçage. Pour avoir une bonne intuition de la nécessité de ces fonctions, il faut regarder les dérivations de typage des termes. Commençons par $t[\Uparrow^i (u/)]$, où $\Delta = D_i, ..., D_1$ $(i = |\Delta|)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash u : B}{\Gamma \vdash u/ \triangleright B, \Gamma}}{D_1, \Gamma \vdash \Uparrow (u/) \triangleright D_1, B, \Gamma}}{\vdots}}{D_{i-1}, ..., D_1, \Gamma \vdash \Uparrow^{i-1} (u/) \triangleright D_{i-1}, ..., D_1, B, \Gamma}}{D_i, D_{i-1}, ..., D_1, \Gamma \vdash \Uparrow^i (u/) \triangleright D_i, D_{i-1}, ..., D_1, B, \Gamma} \quad \Delta, B, \Gamma \vdash t : A}{\Delta, \Gamma \vdash t[\Uparrow^i (u/)] : A}$$

On souhaite pouvoir typer un terme de la forme $(\lambda t')u'$, c'est-à-dire

$$\frac{\dfrac{B, \Delta, \Gamma \vdash t' : A}{\Delta, \Gamma \vdash \lambda t' : B \to A} \quad \Delta, \Gamma \vdash u' : B}{\Delta, \Gamma \vdash (\lambda t')u' : A}$$

Le problème est donc d'arriver à construire un terme t' à partir de t qui soit typable dans l'environnement B, Δ, Γ au lieu de Δ, B, Γ et un terme u' à partir de u qui soit typable dans l'environnement Δ, Γ au lieu de Γ. C'est justement le travail des fonctions $I_i(\ )$ et $K_i(\ )$ respectivement. Regardons à présent la dérivation de typage de $t[\Uparrow^i (\uparrow)]$, où $\Delta = D_i, ..., D_1$ $(i = |\Delta|)$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{B, \Gamma \vdash \uparrow \triangleright \Gamma}{D_1, B, \Gamma \vdash \Uparrow (\uparrow) \triangleright D_1, \Gamma}}{\vdots}}{D_{i-1}, ..., D_1, B, \Gamma \vdash \Uparrow^{i-1} (\uparrow) \triangleright D_{i-1}, ..., D_1, \Gamma}}{D_i, D_{i-1}, ..., D_1, B, \Gamma \vdash \Uparrow^i (\uparrow) \triangleright D_i, D_{i-1}, ..., D_1, \Gamma} \quad \Delta, \Gamma \vdash t : A}{\Delta, B, \Gamma \vdash t[\Uparrow^i (\uparrow)] : A}$$

Le problème est ici d'arriver à construire un terme t' à partir de t qui soit typable dans l'environnement Δ, B, Γ au lieu de Δ, Γ. C'est ce que fait la fonction $J_i(\ )$. On peut donc dès à présent poser les propriétés que ces fonctions devront vérifier.

Propriété 6.2.2

Pour tout terme t on a (avec $i = |\Delta|$) :

- $\Gamma \vdash t : A \quad \Rightarrow \quad \Delta, \Gamma \vdash K_i(t) : A$
- $\Delta, B, \Gamma \vdash t : A \quad \Rightarrow \quad B, \Delta, \Gamma \vdash I_i(t) : A$
- $\Delta, \Gamma \vdash t : A \quad \Rightarrow \quad \Delta, B, \Gamma \vdash J_i(t) : A$

Armés de cette proposition, on peut vérifier que le terme résultant de l'application de la fonction $Ateb$ est typable.

Lemme 6.2.3

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash Ateb(t) : A$$

Preuve : On procède par induction sur la dérivation de typage de t.

- $t = n$ et

$$\frac{}{\Gamma, A, \Delta \vdash n : A}$$

On a alors $Ateb(t) = n$ et la dérivation de typage est la même.

- $t = (u\ v)$ et

$$\frac{\Gamma \vdash u : B \to A \quad \Gamma \vdash v : B}{\Gamma \vdash (u\ v) : A}$$

Par hypothèse d'induction, on a $\Gamma \vdash Ateb(u) : B \to A$ et $\Gamma \vdash Ateb(v) : B$. On peut typer $Ateb(t) = Ateb(u)\ Ateb(v)$ de la façon suivante

$$\frac{\Gamma \vdash Ateb(u) : B \to A \quad \Gamma \vdash Ateb(v) : B}{\Gamma \vdash (Ateb(u)\ Ateb(v)) : A}$$

- $t = \lambda u$ et

$$\frac{B, \Gamma \vdash u : A}{\Gamma \vdash \lambda u : B \to A}$$

Par hypothèse d'induction, on a $\Gamma, x : B \vdash Ateb(u) : A$. On peut typer $Ateb(t) = \lambda Ateb(u)$ de la façon suivante

$$\frac{B, \Gamma \vdash Ateb(u) : A}{\Gamma \vdash \lambda Ateb(u) : A}$$

- Les cas $t = u[\Uparrow^i (v/)]$ et $t = u[\Uparrow^i (\uparrow)]$ sont traités dans l'explication ci-dessus, en utilisant la propriété 6.2.2.

■

Enfin, il est évident que pour tout t, $Ateb(t)$ ne contient pas de substitutions.

6.2.1 Définition des fonctions

La fonction $J_i(t)$ effectue le réindiçage du terme t comme si on avait propagé une substitution $[\Uparrow^i (\uparrow)]$. Comme elle prend en paramètre un terme résultat de la fonction $Ateb$, elle ne travaille que sur des termes sans substitutions. Voici sa définition :

$$\begin{array}{lcll} J_i(n) & = & n+1 & \text{si } n > i \\ J_i(n) & = & n & \text{si } n \leq i \\ J_i(t\ u) & = & J_i(t)\ J_i(u) & \\ J_i(\lambda t) & = & \lambda J_{i+1}(t) & \end{array}$$

La fonction $K_i(t)$ effectue le réindiçage du terme t comme si on avait propagé un paquet de i substitutions $[\uparrow]$. On peut la définir en fonction de la fonction $J_i(t)$:

$$K_i(t) = \underbrace{J_0(J_0(...J_0(t)))}_{i}$$

Lorsque cette fonction est appliquée à une variable, on obtient $K_i(n) = n + i$. La fonction $I_i(t)$ prépare le terme t à recevoir une substitution qui a perdu ses $\Uparrow$. Elle aussi ne s'applique qu'aux termes sans substitutions. Voici sa définition :

$$\begin{array}{lcll} I_i(n) & = & n & \text{si } n > i+1 \\ I_i(n) & = & 1 & \text{si } n = i+1 \\ I_i(n) & = & n+1 & \text{si } n \leq i \\ I_i(t\ u) & = & I_i(t)\ I_i(u) & \\ I_i(\lambda t) & = & \lambda I_{i+1}(t) & \end{array}$$

Les indices sont transformés de la façon suivante. Puisqu'on a supprimé $i \Uparrow$, l'indice $i+1$ doit devenir le nouveau 1. Pour contrebalancer cette modification, il faut que tout les indices plus petits que $i+1$ soient incrémentés de 1. Les autres sont laissés inchangés.

Voici quelques propriétés utiles.

Propriété 6.2.4

Pour tout t, u, i, j, on a

$$K_i(t) = K_j(u) \quad \Rightarrow \quad K_{i+1}(t) = K_{j+1}(u)$$

En effet,

$$K_i(t) = K_j(u) \quad \Rightarrow \quad J_0(K_i(t)) = J_0(K_j(u))$$

Voici d'autres propriétés utiles :

Propriété 6.2.5

Pour tout n et i, on a

$$J_{i+1}(n) \quad = \quad K_1(J_i(n-1))$$

Preuve : On calcule les valeurs en fonction de n et i.
- si $n > i+1$ alors $J_{i+1}(n) = n$, $J_i(n-1) = n-1$ et $K_1(n-1) = n$.
- si $n \leq i+1$ alors $J_{i+1}(n) = n+1$, $J_i(n-1) = n$ et $K_1(n) = n+1$.

■

Propriété 6.2.6

Pour tout $n > 1$ et i, on a

$$I_{i+1}(n) \quad = \quad J_1(I_i(n-1))$$

Preuve : On calcule les valeurs en fonction de n et i.
- si $n > i+2$ alors $I_{i+1}(n) = n$, $I_i(n-1) = n-1$ et $J_1(n-1) = n$.
- si $n = i+2$ alors $I_{i+1}(n) = 1$, $I_i(n-1) = 1$ et $J_1(1) = 1$.
- si $n < i+2$ alors $I_{i+1}(n) = n+1$, $I_i(n-1) = n$ et $J_1(n) = n+1$.

■

Exemple 6.2.7

On reprend notre exemple. On avait calculé

$$Ateb((t[u/]\ v[\Uparrow\ (\Uparrow\ (\Uparrow\ (w/)))])[\Uparrow\ (\Uparrow\ (\uparrow))]) \quad = \quad J_2(((\lambda t)u)\ ((\lambda I_3(v))K_3(w)))$$

On peut propager les opérateurs, ce qui nous donne

$$J_2(((\lambda t)u)\ ((\lambda I_3(v))K_3(w))) \quad = \quad ((\lambda J_3(t))J_2(u))\ ((\lambda J_3(I_3(v)))J_2(K_3(w)))$$

On peut maintenant prouver la propriété 6.2.2.
Preuve :

- $\Delta, \Gamma \vdash t : A \quad \Rightarrow \quad \Delta, B, \Gamma \vdash J_i(t) : A$. On procède par induction sur t.
  - $t = n$ avec $n \leq i$: $J_i(t) = n$. On a

$$\frac{}{\Delta_1, A, \Delta_2, \Gamma \vdash n : A}$$

    Avec $n = |\Delta_1| + 1$. On conclut en typant

$$\frac{}{\Delta_1, A, \Delta_2, B, \Gamma \vdash n : A}$$

  - $t = n$ avec $n > i$: $J_i(t) = n+1$. On a

$$\frac{}{\Delta, \Gamma_1, A, \Gamma_2 \vdash n : A}$$

    Avec $n = |\Delta| + |\Gamma_1| + 1$. On a alors $n+1 = |\Delta| + |\Gamma_1| + 1 + 1$ et

$$\frac{}{\Delta, B, \Gamma_1, A, \Gamma_2 \vdash n : A}$$

  - $t = (u\ v)$: $J_i(t) = (J_i(u)\ J_i(v))$ et on conclut en appliquant deux fois l'hypothèse d'induction.
  - $t = \lambda u$ (avec $A = C \to D$) : $J_i(t) = \lambda J_{i+1}(u)$. On a

$$\frac{C, \Delta, \Gamma \vdash u : D}{\Delta, \Gamma \vdash \lambda u : C \to D}$$

    Par hypothèse d'induction, on a $C, \Delta, B, \Gamma \vdash J_{i+1}(u) : D$ et on peut construire la dérivation de typage suivante

$$\frac{C, \Delta, B, \Gamma \vdash J_{i+1}(u) : D}{\Delta, B, \Gamma \vdash \lambda J_{i+1}(u) : C \to D}$$

- $\Gamma \vdash t : A \quad \Rightarrow \quad \Delta, \Gamma \vdash K_i(t) : A$. On procède par induction sur t.

- $t = n : K_i(n) = \underbrace{J_0(J_0(...J_0(n)))}_{i} = n + i$. On a

$$\overline{\Gamma_1, A, \Gamma_2 \vdash n : A}$$

Avec $n = |\Gamma_1| + 1$. Puisque $i = |\Delta|$, on a $n + i = |\Gamma_1| + |\Delta| + 1$ et

$$\overline{\Delta, \Gamma_1, A, \Gamma_2 \vdash n + i : A}$$

- $t = (u\ v) : K_i(t) = (K_i(u)\ K_i(v))$ et on conclut en appliquant deux fois l'hypothèse d'induction.
- $t = \lambda u$ (avec $A = C \to D$) : $K_i(t) = \underbrace{J_0(J_0(...J_0(\lambda u)))}_{i} = \lambda \underbrace{J_1(J_1(...J_1(u)))}_{i}$. On a

$$\frac{C, \Gamma \vdash u : D}{\Gamma \vdash \lambda u : C \to D}$$

Par le point précédent, on a

$$\begin{array}{c} C, \Gamma \vdash u : D \\ \Downarrow \\ C, E_1, \Gamma \vdash J_1(u) : D \\ \Downarrow \\ C, E_2, E_1, \Gamma \vdash J_1(J_1(u)) : D \\ \Downarrow \\ \vdots \\ \Downarrow \\ C, E_i, ..., E_1, \Gamma \vdash \underbrace{J_1(J_1(...J_1(u)))}_{i} \end{array}$$

Avec $\Delta = E_i, ..., E_1$. On peut alors typer

$$\frac{C, \Delta, \Gamma \vdash \underbrace{J_1(J_1(...J_1(u)))}_{i} : D}{\Delta, \Gamma \vdash K_i(\lambda u) : C \to D}$$

- $\Delta, B, \Gamma \vdash t : A \quad \Rightarrow \quad B, \Delta, \Gamma \vdash I_i(t) : A$. On procède par induction sur t.
  - $t = n$ avec $n > i + 1 : I_i(t) = n$. On a

$$\overline{\Delta, B, \Gamma_1, A, \Gamma_2 \vdash n : A}$$

Avec $n = |\Delta| + 1 + |\Gamma_1| + 1$. On conclut en typant

$$\overline{B, \Delta, \Gamma_1, A, \Gamma_2 \vdash n : A}$$

  - $t = n$ avec $n = i + 1 : I_i(t) = 1$. On a

$$\overline{\Delta, B, \Gamma \vdash n : B}$$

Avec $n = i + 1 = |\Delta| + 1$. On conclut en typant

$$\overline{B, \Delta, \Gamma \vdash 1 : A}$$

  - $t = n$ avec $n \leq i : I_i(t) = n + 1$. On a

$$\overline{\Delta_1, A, \Delta_2, B, \Gamma \vdash n : A}$$

Avec $n = |\Delta_1| + 1$. On a alors $n + 1 = |\Delta_1| + 1 + 1$ et

$$\overline{B, \Delta_1, A, \Delta_2, \Gamma \vdash n : A}$$

– $t = (u\ v) : I_i(t) = (I_i(u)\ I_i(v))$ et on conclut en appliquant deux fois l'hypothèse d'induction.
– $t = \lambda u$ (avec $A = C \to D$) : $I_i(t) = \lambda I_{i+1}(u)$. On a

$$\frac{C, \Delta, B, \Gamma \vdash u : D}{\Delta, B, \Gamma \vdash \lambda u : C \to D}$$

Par hypothèse d'induction, on a $C, \Delta, \Gamma \vdash I_{i+1}(u) : D$ et on peut construire la dérivation de typage suivante

$$\frac{C, \Delta, \Gamma \vdash I_{i+1}(u) : D}{\Delta, \Gamma \vdash \lambda I_{i+1}(u) : C \to D}$$

■

6.2.2 Définition de la relation $\ll$

La fonction *Ateb* efface, entre autres, les substitutions $[\Uparrow^i (\uparrow)]$ et on ne pourra pas les retrouver en réduisant le terme obtenu, comme l'illustre l'exemple suivant.

Exemple 6.2.8

Si on reprend notre exemple, on est parti du terme

$$(t[u/]\ v[\Uparrow (\Uparrow (\Uparrow (w/)))])[\Uparrow (\Uparrow (\uparrow))]$$

La fonction *Ateb* nous a donné le terme

$$((\lambda J_3(t))J_2(u))\ ((\lambda J_3(I_3(v)))J_2(K_3(w)))$$

Celui-ci peut se réduire en deux étapes, recréant deux des trois substitutions d'origine :

$$\begin{array}{c} ((\lambda J_3(t))J_2(u))\ ((\lambda J_3(I_3(v)))J_2(K_3(w))) \\ \to_B \to_B \\ J_3(t)[J_2(u)/]\ J_3(I_3(v))[J_2(K_3(w))/] \end{array}$$

La troisième substitution d'origine $[\Uparrow (\Uparrow (\uparrow))]$ a été complètement convertie en opérations de décalage, et il en a été de même pour les $\Uparrow$ de la substitution $[\Uparrow (\Uparrow (\Uparrow (w/)))]$.

On est donc dans le cas de la démonstration par simulation. Pour effectuer cette simulation, nous allons définir une nouvelle fonction $\overline{t}$ qui met à plat les modifications d'indices d'un terme t.

$$\begin{array}{lcl} \overline{\overline{n}} & = & n \\ \overline{\overline{t\ u}} & = & \overline{t}\ \overline{u} \\ \overline{\overline{\lambda t}} & = & \lambda \overline{t} \\ \overline{\overline{t[u/]}} & = & \overline{t}[\overline{u}/] \\ \overline{\overline{t[\Uparrow^i (u/)]}} & = & I_i(\overline{t})[K_i(\overline{u})/] \\ \overline{\overline{t[\Uparrow^i (\uparrow)]}} & = & J_i(\overline{t}) \end{array}$$

Cette fonction opérera sur des termes avec substitutions. Il nous faut donc étendre les définitions de nos fonctions précédentes pour qu'elles puissent gérer les substitutions. Cependant, puisque la fonction $\overline{t}$ "enlève" de t les $\Uparrow$ et les $\uparrow$, on pourra se restreindre au seul cas de la substitution simple :

$$\begin{array}{lcl} J_i(t[u/]) & = & J_{i+1}(t)[J_i(u)/] \\ I_i(t[u/]) & = & I_{i+1}(t)[I_i(u)/] \end{array}$$

La fonction $\overline{\cdot}$ commute avec les autres fonctions définies ci-dessus, comme l'expriment les lemmes suivants.

Lemme 6.2.9

Pour tout i et t (sans $\Uparrow$ ni $\uparrow$) on a

$$\overline{J_i(t)} \quad = \quad J_i(\overline{t})$$

Preuve : On raisonne par induction sur la structure du terme.
- Si $t = n$, alors $J_i(t) = n'$, $\overline{n'} = n'$ d'une part, et $\overline{n} = n$ d'autre part.
- Dans tous les autres cas, on conclut par hypothèse d'induction.

■

Lemme 6.2.10

Pour tout i et t (sans $\Uparrow$ ni $\uparrow$) on a

$$\overline{I_i(t)} \quad = \quad I_i(\overline{t})$$

Preuve : On raisonne par induction sur la structure du terme.
- Si $t = n$, alors $I_i(t) = n'$, $\overline{n'} = n'$ d'une part, et $\overline{n} = n$ d'autre part.
- Dans tous les autres cas, on conclut par hypothèse d'induction.

■

Lemme 6.2.11

Pour tout i et t (sans $\Uparrow$ ni $\uparrow$) on a

$$\overline{K_i(t)} \quad = \quad K_i(\overline{t})$$

Preuve : C'est une conséquence directe du lemme 6.2.9. ■

On peut vérifier que cette fonction nous convient pour les termes de notre exemple.

Exemple 6.2.12

Regardons le terme final de notre exemple ci-dessus :

$$\begin{gathered}\overline{J_3(t)[J_2(u)/]\ J_3(I_3(v))[J_2(K_3(w))/]} \\ = \\ J_3(\overline{t})[J_2(\overline{u})/]\ J_3(I_3(\overline{v}))[J_2(K_3(\overline{w}))/]\end{gathered}$$

Puis le terme d'origine :

$$\begin{gathered}\overline{(t[u/]\ v[\Uparrow(\Uparrow(\Uparrow(w/)))])[\Uparrow(\Uparrow(\uparrow))]} \\ = \\ J_2(\overline{t}[\overline{u}/]\ I_3(\overline{v})[K_3(\overline{w})/]) \\ = \\ J_3(\overline{t})[J_2(\overline{u})/]\ J_3(I_3(\overline{v}))[J_2(K_3(\overline{w}))/]\end{gathered}$$

Nous avons aussi besoin de définir une relation d'ordre sur le squelette des termes. On souhaite dire que $t \preccurlyeq t'$ si et seulement si t ne contient des $[\uparrow]$ et des $\Uparrow$ qu'aux endroits où t' en contient aussi. On peut formaliser cette définition de la façon suivante :

$$\begin{array}{ccc} \text{pour tout } n \text{ et } m & & n \preccurlyeq m \\ t \preccurlyeq t' \text{ et } u \preccurlyeq u' & \Rightarrow & (t\ u) \preccurlyeq (t'\ u') \\ t \preccurlyeq t' & \Rightarrow & \lambda t \preccurlyeq \lambda t' \\ t \preccurlyeq t' & \Rightarrow & t \preccurlyeq t'[\uparrow] \\ t \preccurlyeq t' \text{ et } s \preccurlyeq s' & \Rightarrow & t[s] \preccurlyeq t'[s'] \end{array}$$

$$\begin{array}{rcc} & & \uparrow \preccurlyeq \uparrow \\ t \preccurlyeq t' & \Rightarrow & t/ \preccurlyeq t'/ \\ s \preccurlyeq s' & \Rightarrow & \Uparrow(s) \preccurlyeq \Uparrow(s') \\ s \preccurlyeq s' & \Rightarrow & s \preccurlyeq \Uparrow(s') \end{array}$$

Exemple 6.2.13

On a $t[\Uparrow(t'/)] \preccurlyeq t[\uparrow][\Uparrow(\Uparrow(\Uparrow(t'/)))]$.

À l'aide de cette relation et de la fonction $\overline{t}$, on peut définir une relation pour effectuer notre simulation. On note cette relation $\lessdot$ et on la définit de la façon suivante :

$$t \lessdot t' \iff \overline{t} = \overline{t'} \text{ et } t \preccurlyeq t'$$

On remarque que l'on a toujours $t \lessdot t$. On peut maintenant passer au lemme initial de notre simulation.

Lemme 6.2.14 (Initialisation)

Pour tout t, il existe u tel que $Ateb(t) \rightarrow_B^* u$ et $u \lessdot t$.

Preuve : On raisonne par induction sur la structure de t.

- Si $t = n$, alors $Ateb(t) = n$ et il suffit de prendre $u = n$.
- Si $t = t_1\ t_2$, alors $Ateb(t) = Ateb(t_1)\ Ateb(t_2)$. Par hypothèse d'induction, il existe u_1 et u_2 tels que $Ateb(t_1) \rightarrow_B^* u_1$ et $Ateb(t_2) \rightarrow_B^* u_2$ avec $u_1 \lessdot t_1$ et $u_2 \lessdot t_2$. On prend alors $u = u_1\ u_2$ et on conclut directement.
- Si $t = \lambda t'$, alors on procède comme ci-dessus en appliquant l'hypothèse d'induction sur t'.
- Si $t = t'[\Uparrow^i(\uparrow)]$, alors $Ateb(t) = J_i(Ateb(t'))$. Par hypothèse d'induction, il existe u' tel que $Ateb(t') \rightarrow_B^* u'$ et $u' \lessdot t'$. On prend $u = J_i(u')$ et on vérifie $u \lessdot t$, c'est-à-dire $\overline{u} = \overline{t}$ et $u \preccurlyeq t$. Cette dernière condition est trivialement vérifiée en se souvenant que $u' \preccurlyeq t'$. On calcule $\overline{u} = \overline{J_i(u')}$, qui est égal à $J_i(\overline{u'})$ par le lemme 6.2.9. D'autre part, $\overline{t} = \overline{t'[\Uparrow^i(\uparrow)]} = J_i(\overline{t'})$, ce qui nous permet de conclure en se souvenant que $\overline{u'} = \overline{t'}$.
- Si $t = t_1[\Uparrow^i(t_2/)]$, alors $Ateb(t) = (\lambda I_i(Ateb(t_1)))K_i(Ateb(t_2))$. Par hypothèse d'induction, il existe u_1 et u_2 tels que $Ateb(t_1) \rightarrow_B^* u_1$ et $Ateb(t_2) \rightarrow_B^* u_2$ avec $u_1 \lessdot t_1$ et $u_2 \lessdot t_2$. On prend alors $u' = (\lambda I_i(u_1))K_i(u_2)$ pour lequel il est clair que $Ateb(t) \rightarrow_B^* u'$. On a $u' \rightarrow_B I_i(u_1)[K_i(u_2)/]$, on prend ce dernier terme comme u et on vérifie que $u \lessdot t$, c'est-à-dire $\overline{u} = \overline{t}$ et $u \preccurlyeq t$. Cette dernière condition est trivialement vérifiée en se souvenant que $u_1 \preccurlyeq t_1$ et $u_2 \preccurlyeq t_2$. On calcule à présent $\overline{u} = \overline{I_i(u_1)[K_i(u_2)/]}$, qui est égal à $I_i(\overline{u_1})[K_i(\overline{u_2})/]$ par les lemmes 6.2.10 et 6.2.11. D'autre part, $\overline{t} = \overline{t_1[\Uparrow^i(t_2/)]} = I_i(\overline{t_1})[K_i(\overline{t_2})/]$, ce qui nous permet de conclure en se souvenant que $\overline{u_1} = \overline{t_1}$ et $\overline{u_2} = \overline{t_2}$.

■

6.2.3 Lemmes de simulation

On va établir quelques lemmes pour la simulation des réductions de $\lambda\upsilon$. On sépare l'ensemble des règles de réduction en deux : on appelle R_1 l'ensemble contenant la seule règle B et R_2 l'ensemble contenant les autres règles. On a bien R_2 fortement normalisant (voir [9]). On veut établir les diagrammes suivants :

$$\begin{array}{ccc} t & \rightarrow_B & t' \\ \forall & & \forall \\ u & \rightarrow_{\lambda\upsilon}^+ & u' \end{array} \qquad\qquad \begin{array}{ccc} t & \rightarrow_{R_2} & t' \\ \forall & & \forall \\ u & \rightarrow_{\lambda\upsilon}^* & u' \end{array}$$

On commence par regarder la simulation de B, puis celle des autres règles. Les deux propriétés suivantes correspondent aux lemmes généraux 4.3.2 et 4.3.3.

Lemme 6.2.15

Pour tout $t \to_B t'$, pour tout $u \lessdot t$ il existe u' tel que $u \to_B u'$ et $u' \lessdot t'$.

$$\begin{array}{ccc} t & \to_B & t' \\ \vdots & & \vdots \\ u & \to_B & u' \end{array}$$

Preuve : Soit $t = (\lambda v)w$ et $(\lambda v)w \to_B v[w/]$ tous les termes $u \lessdot t$ sont de la forme $(\lambda v')w'$ avec $v' \lessdot v$ et $w' \lessdot w$ on peut alors réduire $(\lambda v')w' \to_B v'[w'/]$ et la conclusion est immédiate. ■

Lemme 6.2.16

Pour tout $t \to_{R_2} t'$, pour tout $u \lessdot t$ il existe u' tel que $u \to^*_{\lambda v} u'$ et $u' \lessdot t'$.

$$\begin{array}{ccc} t & \to_{R_2} & t' \\ \vdots & & \vdots \\ u & \to^*_{\lambda v} & u' \end{array}$$

Preuve : On procède par cas sur la règle de R_2.

- $FVar : 1[v/] \to v$. Tous les termes $u \lessdot 1[v/]$ sont de la forme $1[v'/]$ avec $v' \lessdot v$ et $1[v'/] \to_{FVar} v'$.
- $RVar : n+1[v/] \to n$. Tous les termes $u \lessdot n+1[v/]$ sont de la forme $n+1[v'/]$ avec $v' \lessdot v$ et $n+1[v'/] \to_{RVar} n$.
- $App : t = (v\ w)[s] \to (v[s])\ (w[s]) = t'$. On procède par cas sur la forme de s.
  - si $s = \Uparrow^i (\uparrow)$ alors les termes $u \lessdot (v\ w)[\Uparrow^i (\uparrow)]$ peuvent avoir deux formes différentes :
    - soit $u = (v'\ w')[\Uparrow^j (\uparrow)]$ avec $v' \lessdot v$, $w' \lessdot w$, $j \leq i$ et $\overline{u} = \overline{t}$, c'est-à-dire :

$$\begin{array}{ccc} \overline{(v'\ w')[\Uparrow^j (\uparrow)]} & = & \overline{(v\ w)[\Uparrow^i (\uparrow)]} \\ = & & = \\ J_j(\overline{v'}\ \overline{w'}) & = & J_i(\overline{v}\ \overline{w}) \\ = & & = \\ J_j(\overline{v'})\ J_j(\overline{w'}) & = & J_i(\overline{v})\ J_i(\overline{w}) \end{array}$$

      Ce qui implique $J_j(\overline{v'}) = J_i(\overline{v})$ et $J_j(\overline{w'}) = J_i(\overline{w})$. Dans ce cas, $(v'\ w')[\Uparrow^j (\uparrow\ )\ ] \to_{App} (v'[\Uparrow^j (\uparrow)])\ (w'[\Uparrow^j (\uparrow)])$ et on peut conclure aisément que $(v'[\Uparrow^j (\uparrow)])\ (w'[\Uparrow^j (\uparrow)]) \lessdot (v[\Uparrow^i (\uparrow)])\ (w[\Uparrow^i (\uparrow)])$.
    - soit $u = (v'\ w')$ avec $v' \lessdot v$, $w' \lessdot w$, et $\overline{u} = \overline{t}$, c'est-à-dire :

$$\begin{array}{ccc} \overline{v'\ w'} & = & \overline{(v\ w)[\Uparrow^i (\uparrow)]} \\ = & & = \\ \overline{v'}\ \overline{w'} & = & J_i(\overline{v}\ \overline{w}) \\ = & & = \\ \overline{v'}\ \overline{w'} & = & J_i(\overline{v})\ J_i(\overline{w}) \end{array}$$

      Ce qui implique $\overline{v'} = J_i(\overline{v})$ et $\overline{w'} = J_i(\overline{w})$. Dans ce cas, $(v'\ w')$ ne se réduit pas et on peut conclure que $(v'\ w') \lessdot (v[\Uparrow^i (\uparrow)])\ (w[\Uparrow^i (\uparrow)])$.
  - si $s = \Uparrow^i (r/)$ alors tous les termes $u \lessdot (v\ w)[\Uparrow^i (r/)]$ sont de la forme $(v'\ w')[\Uparrow^j (r'/)]$ avec $v' \lessdot v$, $w' \lessdot w$, $r' \lessdot r$, $j \leq i$ et $\overline{u} = \overline{t}$, c'est-à-dire :

$$\begin{array}{ccc} \overline{(v'\ w')[\Uparrow^j (r'/)]} & = & \overline{(v\ w)[\Uparrow^i (r/)]} \\ = & & = \\ I_j(\overline{v'}\ \overline{w'})[K_j(\overline{r'})/] & = & I_i(\overline{v}\ \overline{w})[K_i(\overline{r})/] \\ = & & = \\ (I_j(\overline{v'})\ I_j(\overline{w'}))[K_j(\overline{r'})/] & = & (I_i(\overline{v})\ I_i(\overline{w}))[K_i(\overline{r})/] \end{array}$$

Ce qui implique $J_j(\overline{v'}) = J_i(\overline{v})$, $J_j(\overline{w'}) = J_i(\overline{w})$ et $K_j(\overline{r'}) = K_i(\overline{r})$. Dans ce cas, $(v'\ w')[\Uparrow^j (r'/)] \rightarrow_{App} (v'[\Uparrow^j (r'/)])\ (w'[\Uparrow^j (r'/)])$ et on peut conclure aisément que $(v'[\Uparrow^j (r'/)])\ (w'[\Uparrow^j (r'/)]) \lessdot (v[\Uparrow^i (r/)])\ (w[\Uparrow^i (r/)])$.

- *Lambda* : $t = (\lambda v)[s] \rightarrow \lambda(v[\Uparrow (s)]) = t'$. On procède par cas sur la forme de s.
  - si $s = \Uparrow^i (\uparrow)$ alors les termes $u \lessdot (\lambda v)[\Uparrow^i (\uparrow)]$ peuvent avoir deux formes différentes :
    - soit $u = (\lambda v')[\Uparrow^j (\uparrow)]$ avec $v' \lessdot v$, $j \leq i$ et $\overline{u} = \overline{t}$, c'est-à-dire :

$$\begin{array}{ccc} \overline{(\lambda v')[\Uparrow^j (\uparrow)]} & = & \overline{(\lambda v)[\Uparrow^i (\uparrow)]} \\ = & & = \\ J_j(\overline{\lambda v'}) & = & J_i(\overline{\lambda v}) \\ = & & = \\ \lambda J_{j+1}(\overline{v'}) & = & \lambda J_{i+1}(\overline{v}) \end{array}$$

      Ce qui implique $J_{j+1}(\overline{v'}) = J_{i+1}(\overline{v})$. Dans ce cas, $(\lambda v')[\Uparrow^j (\uparrow)] \rightarrow_{Lambda} \lambda(v'[\Uparrow^{j+1} (\uparrow)])$ et on peut conclure aisément que $\lambda(v'[\Uparrow^{j+1} (\uparrow)]) \lessdot \lambda(v[\Uparrow^{i+1} (\uparrow)])$.
    - soit $u = \lambda v'$ avec $v' \lessdot v$, et $\overline{u} = \overline{t}$ c'est-à-dire :

$$\begin{array}{ccc} \overline{\lambda v'} & = & \overline{(\lambda v)[\Uparrow^i (\uparrow)]} \\ = & & = \\ \lambda\overline{v'} & = & J_i(\overline{\lambda v}) \\ = & & = \\ \lambda\overline{v'} & = & \lambda J_{i+1}(\overline{v}) \end{array}$$

      Ce qui implique $\overline{v'} = J_{i+1}(\overline{v})$. Dans ce cas, $\lambda v'$ ne se réduit pas et on peut conclure que $\lambda v' \lessdot \lambda(v[\Uparrow^{i+1} (\uparrow)])$.
  - si $s = \Uparrow^i (r/)$ alors tous les termes $u \lessdot (\lambda v)[\Uparrow^i (r/)]$ sont de la forme $(\lambda v')[\Uparrow^j (r'/)]$ avec $v' \lessdot v$, $r \lessdot r'$, $j \leq i$ et $\overline{u} = \overline{t}$, c'est-à-dire :

$$\begin{array}{ccc} \overline{(\lambda v')[\Uparrow^j (r'/)]} & = & \overline{(\lambda v)[\Uparrow^i (r/)]} \\ = & & = \\ I_j(\overline{\lambda v'})[K_j(\overline{r'})/] & = & I_i(\overline{\lambda v})[K_i(\overline{r})/] \\ = & & = \\ \lambda I_{j+1}(\overline{v'})[K_j(\overline{r'})/] & = & \lambda I_{i+1}(\overline{v})[K_i(\overline{r})/] \end{array}$$

    Ce qui implique $J_{j+1}(\overline{v'}) = J_{i+1}(\overline{v})$ et $K_j(\overline{r'}) = K_i(\overline{r})$. Dans ce cas, $(\lambda v')[\Uparrow^j (r'/)] \rightarrow_{Lambda} \lambda(v'[\Uparrow^{j+1} (r'/)])$ et on peut conclure aisément que $\lambda(v'[\Uparrow^{j+1} (r'/)]) \lessdot \lambda(v[\Uparrow^{i+1} (r/)])$ grâce à la propriété 6.2.4.
- *VarShift* : $n[\uparrow] \rightarrow n+1$. Les deux seuls termes $u \lessdot n[\uparrow]$ sont $n[\uparrow]$ et $n+1$, ce qui nous permet de conclure avec éventuellement une étape de *VarShift*.
- *FVarLift* : $t = 1[\Uparrow (s)] \rightarrow_{FVarLift} 1 = t'$. On procède par cas sur la forme de s.
  - si $s = \Uparrow^i (\uparrow)$ alors les termes $u \lessdot 1[\Uparrow (\Uparrow^i (\uparrow))]$ peuvent avoir deux formes différentes :
    - soit $u = 1[\Uparrow (\Uparrow^j (\uparrow))]$ avec $j \leq i$ et $\overline{u} = \overline{t}$. On a alors $1[\Uparrow (\Uparrow^j (\uparrow))] \rightarrow_{FVarLift} 1$ et on conclut aisément.
    - soit $u = 1$ avec $\overline{u} = \overline{t}$ et on peut conclure tout aussi aisément.
  - si $s = \Uparrow^i (r/)$ alors tous les termes $u \lessdot 1[\Uparrow (\Uparrow^i (r/))]$ sont de la forme $1[\Uparrow (\Uparrow^j (r'/))]$ avec $r' \lessdot r$, $j \leq i$ et $\overline{u} = \overline{t}$. On a alors $1[\Uparrow (\Uparrow^j (r'/))] \rightarrow_{FVarLift} 1$ et on conclut.
- *RVarLift* : $t = n+1[\Uparrow (s)] \rightarrow n[s][\uparrow] = t'$. On procède par cas sur la forme de s.
  - si $s = \Uparrow^i (\uparrow)$ alors les termes $u \lessdot n+1[\Uparrow (\Uparrow^i (\uparrow))]$ peuvent avoir deux formes différentes :
    - soit $u = n'[\Uparrow^j (\uparrow)]$ avec $j \leq i+1$ et $\overline{u} = \overline{t}$, c'est-à-dire :

$$\begin{array}{ccc} \overline{n'+1[\Uparrow (\Uparrow^j (\uparrow))]} & = & \overline{n+1[\Uparrow (\Uparrow^i (\uparrow))]} \\ = & & = \\ J_{j+1}(n') & = & J_{i+1}(n+1) \end{array}$$

On déduit facilement de l'égalité précédente que n' ne peut être plus petit que n et qu'il est donc supérieur à 1. Dans ce cas, $n'[\Uparrow\ (\Uparrow^j\ (\uparrow))] \rightarrow_{RVarLift} n'-1[\Uparrow^j\ (\uparrow)][\uparrow]$ et il faut vérifier que

$$\begin{array}{ccc} \overline{n[\Uparrow^i\ (\uparrow)][\uparrow]} & = & \overline{n'-1[\Uparrow^j\ (\uparrow)][\uparrow]} \\ = & & = \\ K_1(J_i(n)) & = & K_1(J_j(n'-1)) \end{array}$$

Par la propriété 6.2.5, on a $K_1(J_i(n)) = J_{i+1}(n+1)$ et $K_1(J_j(n'-1)) = J_{j+1}(n')$, ce qui nous permet de conclure que $n'-1[\Uparrow^j\ (\uparrow)][\uparrow] \lessdot n[\Uparrow^i\ (\uparrow)][\uparrow]$.

– soit $u = n'$, et $\overline{u} = \overline{t}$ c'est-à-dire :

$$\begin{array}{ccc} \overline{n'} & = & \overline{n+1[\Uparrow\ (\Uparrow^i\ (\uparrow))]} \\ = & & = \\ n' & = & J_{i+1}(n+1) \end{array}$$

Dans ce cas, u ne se réduit pas et il faut vérifier que

$$\begin{array}{ccc} \overline{n[\Uparrow^i\ (\uparrow)][\uparrow]} & = & \overline{n'} \\ = & & = \\ K_1(J_i(n)) & = & n' \end{array}$$

On conclut que $n' \lessdot n[\Uparrow^i\ (\uparrow)][\uparrow]$ grâce à la propriété 6.2.5.

- si $s = \Uparrow^i\ (r/)$ alors tous les termes $u \lessdot n+1[\Uparrow\ (\Uparrow^i\ (r/))]$ sont de la forme $n'[\Uparrow^j\ (r'/)]$ avec $r \lessdot r'$, $j \leq i$ et $\overline{u} = \overline{t}$, c'est-à-dire :

$$\begin{array}{ccc} \overline{n'[\Uparrow^j\ (r'/)]} & = & \overline{n+1[\Uparrow\ (\Uparrow^i\ (r/))]} \\ = & & = \\ I_j(n')[K_j(\overline{r'})/] & = & I_{i+1}(n+1)[K_{i+1}(\overline{r})/] \end{array}$$

Ce qui implique $I_j(n') = I_{i+1}(n+1)$ et $K_j(\overline{r'}) = K_{i+1}(\overline{r})$. Deux cas sont possibles suivant la valeur de j.

– $j = 0$: on a alors $I_0(n') = n' = I_{i+1}(n+1)$, $K_0(\overline{r'}) = \overline{r'} = K_{i+1}(\overline{r})$ et il faut vérifier que

$$\begin{array}{ccc} \overline{n[\Uparrow^i\ (r/)][\uparrow]} & = & \overline{n'[r'/]} \\ = & & \\ K_1(I_i(n)[K_i(r)/]) & & \\ = & & \\ J_0(I_i(n)[K_i(r)/]) & = & \\ = & & \\ J_1(I_i(n))[J_0(K_i(r))/] & & \\ = & & \\ J_1(I_i(n))[K_{i+1}(r)/] & = & n'[\overline{r'}] \end{array}$$

On peut conclure que $J_1(I_i(n)) = I_{i+1}(n+1)n'$ grâce à la propriété 6.2.6.

– $j > 0$: $n'[\Uparrow^j\ (r'/)]$ se réduit vers $n'[\Uparrow^{j-1}\ (r'/)][\uparrow]$ et il faut vérifier que

$$\begin{array}{ccc} \overline{n[\Uparrow^i\ (r/)][\uparrow]} & = & \overline{n'[\Uparrow^{j-1}\ (r'/)][\uparrow]} \\ = & & = \\ K_1(I_i(n)[K_i(r)/]) & & K_1(I_{j-1}(n')[K_{j-1}(r')/]) \\ = & & = \\ J_0(I_i(n)[K_i(r)/]) & & J_0(I_{j-1}(n')[K_{j-1}(r')/]) \\ = & & = \\ J_1(I_i(n))[J_0(K_i(r))/] & & J_1(I_{j-1}(n'))[J_0(K_{j-1}(r'))/] \\ = & & = \\ J_1(I_i(n))[K_{i+1}(r)/] & = & J_1(I_{j-1}(n'))[K_j(r')/] \end{array}$$

Et on conclut directement grâce à la propriété 6.2.6.

■

6.2.4 Simulation

La fonction *Ateb* et la relation $\lessdot$ vérifient les hypothèses du théorème 4.4.2. On peut donc l'appliquer et en tirer la conclusion attendue.

Théorème 6.2.17
Le $\lambda\upsilon$-calcul est fortement normalisant.

Preuve : Comme on a les propriétés de PSN pour $\lambda\upsilon$ [9] et de SN du λ-calcul avec simplement typé [55] (qu'on étend facilement au calcul avec indices de De Bruijn), on peut utiliser le théorème 4.4.1.

■

Chapitre 7

Application à $\lambda\sigma$

Le calcul $\lambda\sigma$ [1] est un calcul avec indices de De Bruijn et substitutions explicites multiples. La difficulté supplémentaire par rapport à $\lambda\upsilon$ réside dans ces substitutions multiples. Notre application à ce calcul est seulement un exercice puisqu'il ne possède pas la propriété PSN. Cela ne nous empêchera pas d'utiliser notre théorème, établissant qu'il n'y a pas d'autre source de non-terminaison dans $\lambda\sigma$ que celle qui invalide PSN. De plus, ce travail pourra sans doute servir à montrer SN pour des stratégies de $\lambda\sigma$ qui auraient PSN.

7.1 Présentation du calcul

Les termes du $\lambda\sigma$-calcul sont donnés par la grammaire suivante :

$$\begin{array}{l} t ::= 1 \mid (t\ t) \mid \lambda t \mid t[s] \\ s ::= id \mid \uparrow \mid t \cdot s \mid s \circ s \end{array}$$

On étend la syntaxe avec les entiers $2, 3, ..., n$ définis de la façon suivante : $n = 1\underbrace{[\uparrow]...[\uparrow]}_{n-1}$. Lorsqu'on parlera des substitutions des termes, on considérera que les indices n n'en contiennent pas.

Voici l'ensemble des règles de réduction :

$$\begin{array}{lll} (\lambda t)u & \rightarrow_{B} & t[u \cdot id] \\ \\ (t\ u)[s] & \rightarrow_{App} & (t[s])\ (u[s]) \\ (\lambda t)[s] & \rightarrow_{Lambda} & \lambda(t[1 \cdot (s \circ \uparrow)]) \\ 1[id] & \rightarrow_{VarId} & 1 \\ 1[t \cdot s] & \rightarrow_{VarCons} & t \\ t[s][s'] & \rightarrow_{Clos} & t[s \circ s'] \\ \\ id \circ s & \rightarrow_{IdL} & s \\ \uparrow \circ id & \rightarrow_{ShiftId} & \uparrow \\ \uparrow \circ (t \cdot s) & \rightarrow_{ShiftCons} & s \\ (t \cdot s) \circ s' & \rightarrow_{Map} & t[s'] \cdot (s \circ s') \\ (s_1 \circ s_2) \circ s_3 & \rightarrow_{Ass} & s_1 \circ (s_2 \circ s_3) \end{array}$$

Regardons les règles de typage (voir la remarque 6.1.1 en ce qui concerne le symbole $\triangleright$) :

$$\frac{}{A, \Gamma \vdash 1 : A} \qquad \frac{\Gamma \vdash s \triangleright \Gamma' \quad \Gamma' \vdash t : A}{\Gamma \vdash t[s] : A}$$

$$\frac{\Gamma \vdash t : B \to A \quad \Gamma \vdash u : B}{\Gamma \vdash (t\ u) : A} \qquad \frac{B, \Gamma \vdash t : A}{\Gamma \vdash \lambda t : B \to A}$$

$$\frac{}{\Gamma \vdash id \triangleright \Gamma} \qquad \frac{}{A, \Gamma \vdash \uparrow \triangleright \Gamma}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash s \triangleright \Gamma'}{\Gamma \vdash t \cdot s \triangleright A, \Gamma'} \qquad \frac{\Gamma \vdash s' \triangleright \Gamma'' \quad \Gamma'' \vdash s \triangleright \Gamma'}{\Gamma \vdash s \circ s' \triangleright \Gamma'}$$

On peut ajouter une règle de typage supplémentaire pour les variables n, afin de raccourcir les dérivations. Voici la nouvelle règle et sa justification (avec $n = |\Gamma| + 1$ et $\Gamma = C_1, ..., C_{n-1}$) :

$$\frac{\dfrac{\dfrac{}{C_{n-1}, A, \Delta \vdash \uparrow \triangleright A, \Delta} \quad \dfrac{}{A, \Delta \vdash 1 : A}}{\vdots}}{\dfrac{\Gamma, A, \Delta \vdash \uparrow \triangleright C_2, ..., C_{n-1}, A, \Delta \quad C_2, ..., C_{n-1}, A, \Delta \vdash 1\underbrace{[\uparrow]...[\uparrow]}_{n-2} : A}{\Gamma, A, \Delta \vdash 1\underbrace{[\uparrow]...[\uparrow]}_{n-1} : A}} \qquad \frac{}{\Gamma, A, \Delta \vdash n : A}$$

La remontée des substitutions et certaines des fonctions définies dans la suite de notre travail s'inspirent fortement de [24].

7.2 PSN implique SN

On va procéder d'une façon similaire à celle de la section 6.2. On définit la fonction $Ateb$ de la façon suivante :

$$\begin{array}{lcl} Ateb(n) & = & n \\ Ateb(t\ u) & = & Ateb(t)\ Ateb(u) \\ Ateb(\lambda t) & = & \lambda Ateb(t) \\ Ateb(t[id]) & = & Ateb(t) \\ Ateb(t[\uparrow]) & = & \mathcal{U}_0^1(Ateb(t)) \\ Ateb(t[s \circ s']) & = & Ateb(t[s][s']) \\ Ateb(t[t' \cdot s]) & = & Ateb((\lambda t)[s])\ Ateb(t') \end{array}$$

Où $\mathcal{U}_i^j(t)$ est une fonction que nous allons définir dans la suite. L'idée de cette fonction est d'anticiper la propagation de la substitution $[\uparrow]$ et d'effectuer le réindiçage à l'avance. Pour avoir une bonne intuition de la nécessité de ces fonctions, il faut regarder la dérivation de typage de $t[\uparrow]$.

$$\frac{\dfrac{}{B, \Gamma \vdash \uparrow \triangleright \Gamma} \quad \Gamma \vdash t : A}{B, \Gamma \vdash t[\uparrow] : A}$$

Exemple 7.2.1

Par exemple, en supposant que pour tout x parmi t, u, v on a $x = Ateb(x)$, alors on a

$$Ateb((t[u \cdot id]\ v[1 \cdot 1 \cdot 5 \cdot \uparrow])[\uparrow]) \quad = \quad \mathcal{U}_0^1(((\lambda t)u)\ (((\mathcal{U}_0^1(\lambda\lambda\lambda v)5)1)1))$$

Le calcul $\mathcal{U}_0^1(t)$ va donc incrémenter de 1 les variables libres de t afin de permettre le typage de celui-ci dans l'environnement B, Γ. On peut donc dès à présent poser la propriété que cette fonction devra vérifier.

Propriété 7.2.2

Pour tout terme t sans substitution on a : $\Gamma \vdash t : A \Rightarrow B, \Gamma \vdash \mathcal{U}_0^1(t) : A$.

Il est évident que pour tout t, $Ateb(t)$ ne contient pas de substitutions. On peut vérifier que le terme résultant de l'application de la fonction $Ateb$ est typable.

Lemme 7.2.3

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash Ateb(t) : A$$

Preuve : On procède par induction sur le nombre d'étapes du calcul de $Ateb(t)$.

- $t = 1$ et

$$\frac{}{A, \Delta \vdash 1 : A}$$

On a alors $Ateb(t) = 1$ et la dérivation de typage est la même.

- $t = (u\ v)$ et

$$\frac{\Gamma \vdash u : B \to A \quad \Gamma \vdash v : B}{\Gamma \vdash (u\ v) : A}$$

Par hypothèse d'induction, on a $\Gamma \vdash Ateb(u) : B \to A$ et $\Gamma \vdash Ateb(v) : B$. On peut typer $Ateb(t) = Ateb(u)\ Ateb(v)$ de la façon suivante

$$\frac{\Gamma \vdash Ateb(u) : B \to A \quad \Gamma \vdash Ateb(v) : B}{\Gamma \vdash (Ateb(u)\ Ateb(v)) : A}$$

- $t = \lambda u$ et

$$\frac{B, \Gamma \vdash u : A}{\Gamma \vdash \lambda u : B \to A}$$

Par hypothèse d'induction, on a $B, \Gamma \vdash Ateb(u) : A$. On peut typer $Ateb(t) = \lambda Ateb(u)$ de la façon suivante

$$\frac{B, \Gamma \vdash Ateb(u) : A}{\Gamma \vdash \lambda Ateb(u) : B \to A}$$

- $t = u[id]$ et

$$\frac{\Gamma \vdash id \triangleright \Gamma \quad \Gamma \vdash u : A}{\Gamma \vdash u[id] : A}$$

On conclut directement par hypothèse d'induction.

- $t = u[\uparrow]$ et

$$\frac{B, \Gamma \vdash \uparrow \triangleright \Gamma \quad \Gamma \vdash u : A}{B, \Gamma \vdash u[\uparrow] : A}$$

On conclut par hypothèse d'induction et par la propriété 7.2.2.

- $t = u[v \cdot s]$ et

$$\frac{\dfrac{\Gamma \vdash s \triangleright \Gamma' \quad \Gamma \vdash v : B}{\Gamma \vdash v \cdot s \triangleright B, \Gamma'} \qquad B, \Gamma' \vdash u : A}{\Gamma \vdash u[v \cdot s] : A}$$

Par hypothèse d'induction, on a $\Gamma \vdash Ateb((\lambda u)[s]) : B \to A$ et $\Gamma \vdash Ateb(v) : B$. On conclut en typant

$$\frac{\Gamma \vdash Ateb((\lambda u)[s]) : B \to A \quad \Gamma \vdash Ateb(v) : B}{\Gamma \vdash Ateb((\lambda u)[s])\ Ateb(v) : A}$$

- $t = u[s \circ s']$, alors on conclut directement par hypothèse d'induction.

■

7.2.1 Définition des fonctions

La fonction $\mathcal{U}_i^j(t)$ effectue le réindiçage du terme t comme si on avait propagé une substitution $[\uparrow]$. Comme elle prend en paramètre un terme résultat de la fonction $Ateb$, elle ne travaille que sur des termes sans substitutions, cependant, nous aurons besoin, plus tard, de l'utiliser sur des termes avec substitutions mais sans $\uparrow$. Lorsqu'elle est appliquée à une substitution, elle renvoie comme résultat un couple formé d'un entier et d'une substitution, sinon elle renvoie un terme. Voici sa définition complète :

$$\begin{array}{lcll}
\mathcal{U}_i^j(n) & = & n + j & \text{si } n > i \\
\mathcal{U}_i^j(n) & = & n & \text{si } n \leq i \\
\mathcal{U}_i^j(t\ u) & = & \mathcal{U}_i^j(t)\ \mathcal{U}_i^j(u) & \\
\mathcal{U}_i^j(\lambda t) & = & \lambda \mathcal{U}_{i+1}^j(t) & \\
 & & & \\
\mathcal{U}_i^j(t[s]) & = & \text{soit } i', s' = \mathcal{U}_i^j(s) & \\
 & & \text{dans } \mathcal{U}_{i'}^j(t)[s'] & \\
\mathcal{U}_i^j(id) & = & i, id & \\
\mathcal{U}_i^j(t \cdot s) & = & \text{soit } i', s' = \mathcal{U}_i^j(s) & \\
 & & \text{dans } i' + 1, \mathcal{U}_i^j(t) \cdot s' & \\
\mathcal{U}_i^j(s_1 \circ s_2) & = & \text{soit } i'_2, s'_2 = \mathcal{U}_i^j(s_2) & \\
 & & \text{et } i'_1, s'_1 = \mathcal{U}_{i'_2}^j(s_1) & \\
 & & \text{dans } i'_1, s'_1 \circ s'_2 &
\end{array}$$

La modification de l'indice i (et la valeur de l'entier du couple résultat dans le cas des substitutions) reflète le nombre de $\cdot$ traversés, ceux-ci se comportant comme des λ.

On donne à présent la preuve de la propriété 7.2.2.

Preuve : Il nous faut prouver que, pour tout t sans substitutions, $\Gamma \vdash t : A \Rightarrow B, \Gamma \vdash \mathcal{U}_0^1(t) : A$. En fait on va prouver un résultat plus général, à savoir $\Gamma, \Delta \vdash t : A \Rightarrow \Gamma, B, \Delta \vdash \mathcal{U}_i^1(t) : A$ où $i = |\Gamma|$. On procède par induction sur t.

– $t = n$ avec $n \leq i : \mathcal{U}_i^1(t) = n$. On a

$$\overline{\Gamma_1, A, \Gamma_2, \Delta \vdash n : A}$$

Avec $n = |\Gamma_1| + 1$. On peut conclure en typant

$$\overline{\Gamma_1, A, \Gamma_2, B, \Delta \vdash n : A}$$

– $t = n$ avec $n > i : \mathcal{U}_i^1(t) = n + 1$. On a

$$\overline{\Gamma, \Delta_1, A, \Delta_2 \vdash n : A}$$

Avec $n = |\Gamma| + |\Delta_1| + 1$. On peut conclure en typant

$$\overline{\Gamma, B, \Delta_1, A, \Delta_2 \vdash n + 1 : A}$$

– $t = (u\ v) : \mathcal{U}_i^1(t) = (\mathcal{U}_i^1(u)\ \mathcal{U}_i^1(v))$. On conclut en appliquant deux fois l'hypothèse d'induction.
– $t = \lambda u$ (avec $A = C \to D$) : $\mathcal{U}_i^1(t) = \lambda \mathcal{U}_{i+1}^1(u)$. On a

$$\frac{C, \Gamma, \Delta \vdash u : D}{\Gamma, \Delta \vdash \lambda u : C \to D}$$

Par hypothèse d'induction, on a $C, \Gamma, B, \Delta \vdash \mathcal{U}_{i+1}^1(u) : D$, ce qui nous permet de construire la dérivation suivante

$$\frac{C, \Gamma, B, \Delta \vdash \mathcal{U}_{i+1}^1(u) : D}{\Gamma, B, \Delta \vdash \lambda \mathcal{U}_{i+1}^1(u) : C \to D}$$

■

Voici une propriété utile.

Propriété 7.2.4

Pour tout t, i, j, l, on a

$$\mathcal{U}_i^j(\mathcal{U}_i^l(t)) \quad = \quad \mathcal{U}_i^{j+l}(t)$$

Preuve : Par induction sur le terme t, le seul cas intéressant est celui de l'indice et il est immédiat. Les autres cas se montrent par application directe de l'hypothèse d'induction. ■

Exemple 7.2.5

On reprend notre exemple. On avait calculé

$$Ateb((t[u \cdot id]\ v[1 \cdot 1 \cdot 5 \cdot \uparrow])[\uparrow]) \quad = \quad \mathcal{U}_0^1(((\lambda t)u)\ (((\mathcal{U}_0^1(\lambda\lambda\lambda v)5)1)1))$$

On peut propager l'opérateur, ce qui nous donne

$$\mathcal{U}_0^1(((\lambda t)u)\ (\lambda((\lambda(\mathcal{U}_0^1(\lambda v)w))1))1) \quad = ((\lambda\mathcal{U}_1^1(t))\mathcal{U}_0^1(u))\ ((((\lambda\lambda\lambda\mathcal{U}_3^2(v))6)2)2)$$

7.2.2 Définition de la relation $\lessdot$

La fonction $Ateb$ appliquée à un terme t renvoie un nouveau terme t' qui ne peut pas se réduire vers t. En effet, les termes $\uparrow$ disparaissent et l'information qu'ils portaient est propagée dans t'. Les réduits de t' ne pourront donc pas comporter ces termes. Ceci est illustré par l'exemple suivant.

Exemple 7.2.6

Si on reprend notre exemple, on a

$$\underline{((\lambda\mathcal{U}_1^1(t))\mathcal{U}_0^1(u))}\ ((((\lambda\lambda\lambda\mathcal{U}_3^2(v))6)2)2)$$
$$\to^*$$
$$\mathcal{U}_1^1(t)[\mathcal{U}_0^1(u) \cdot id]\ \mathcal{U}_3^2(v)[2 \cdot 2 \cdot 6 \cdot id]$$

On remarque bien que l'information de réindiçage du $\uparrow$ du terme initial a été propagée dans le terme : la substitution $[1 \cdot 1 \cdot 5 \cdot \uparrow]$ est devenue $[2 \cdot 2 \cdot 6 \cdot id]$.

Le terme d'origine est

$$(t[u \cdot id]\ v[1 \cdot 1 \cdot 5 \cdot \uparrow])[\uparrow]$$

Comme dans le chapitre précédent, la fonction $Ateb$ a propagé les informations de réindiçage, le terme qu'elle nous renvoie ne peut donc pas se réduire vers le terme d'origine.

On va donc être obligé de simuler les réductions de notre terme original par le terme obtenu. On pourrait penser définir les fonctions et la relation $\lessdot$ comme ce qui suit. On se rendra compte que cela ne fonctionne pas et on verra une solution alternative.

Pour effectuer la simulation, on pourrait procéder comme dans le chapitre précédent, à savoir définir une fonction $\overline{t}$ qui mette à plat les modifications d'indices d'un terme t et y supprime les $[id]$ solitaires ainsi qu'une relation d'ordre sur le squelette des termes, notée $\ll$. On souhaite que cette dernière dise que $t \preccurlyeq t'$ si et seulement si t' ne contient des $\uparrow$ et des $[id]$ qu'aux endroits où t en contient aussi. La figure 7.1 présente leur définition.

$\overline{n}$	$=$	n	pour tout n et m		$n \ll m$
$\overline{t\ u}$	$=$	$\overline{t}\ \overline{u}$	$t \ll t'$ et $u \ll u'$	$\Rightarrow$	$(t\ u) \ll (t'\ u')$
$\overline{\lambda t}$	$=$	$\lambda\overline{t}$	$t \ll t'$	$\Rightarrow$	$\lambda t \ll \lambda t'$
$\overline{t[s]}$	$=$	soit $n, s' = \overline{s}$ dans	$t \ll t'$	$\Rightarrow$	$t \ll t'[\uparrow]$
		$\mathcal{U}_0^n(\overline{t})[s']$ si $s' \neq \emptyset$	$t \ll t'$	$\Rightarrow$	$t \ll t'[id]$
		$\mathcal{U}_0^n(\overline{t})$ sinon	$t \ll t'$ et $s \ll s'$	$\Rightarrow$	$t[s] \ll t'[s']$
$\overline{\uparrow}$	$=$	$1, \emptyset$			
$\overline{id}$	$=$	$0, \emptyset$			$\uparrow \ll \uparrow$
$\overline{t \cdot s}$	$=$	soit $n, s' = \overline{s}$ dans			$id \ll id$
		$n, \overline{t}[s']$ si $s' \neq \emptyset$			$id \ll \uparrow$
		$n, \overline{t} \cdot id$ sinon	$t \ll t'$ et $s \ll s'$	$\Rightarrow$	$t \cdot s \ll t' \cdot s'$
$\overline{s_1 \circ s_2}$	$=$	soit $n_1, s_1' = \overline{s_1}$	$s \ll s'$	$\Rightarrow$	$s \ll s' \circ id$
		et $n_2, s_2' = \overline{s_2}$ dans	$s \ll s'$	$\Rightarrow$	$s \ll id \circ s'$
		$n_1 + n_2, \emptyset$ si $s_1' = s_2' = \emptyset$	$s \ll s'$	$\Rightarrow$	$s \ll s' \circ \uparrow$
		$n_1 + n_2, \mathcal{U}_0^{n_2}(s_1')$ si $s_2' = \emptyset$	$s \ll s'$	$\Rightarrow$	$s \ll \uparrow \circ s'$
		$n_1 + n_2, s_2'$ si $s_1' = \emptyset$	$s_1 \ll s_1'$ et $s_2 \ll s_2'$	$\Rightarrow$	$s_1 \circ s_2 \ll s_1' \circ s_2'$
		$n_1 + n_2, \mathcal{U}_0^{n_2}(s_1') \circ s_2'$ sinon			

FIG. 7.1 – Définition de la fonction $\overline{t}$ et de l'ordre $\ll$

On pourrait ensuite définir la relation $\mathcal{R}$, pour effectuer notre simulation, de la façon suivante :

$$t\mathcal{R}t' \iff \overline{t} = \overline{t'} \text{ et } t \ll t'$$

Cependant, il est inutile d'aller plus loin car cette relation ne conviendra pas pour la simulation. En effet, un problème survient pour la simulation de la règle $Abs : (\lambda t)[s] \to \lambda(t[1 \cdot (s \circ \uparrow)]$. Si $s = id$ (ou $\uparrow$), alors un terme $u\mathcal{R}(\lambda t)[id]$ peut être λt qui ne vérifie pas $\lambda t \mathcal{R} \lambda(t[1 \cdot (s \circ \uparrow)]$. Il faudrait que notre relation puisse s'étendre à cette id étendue, ainsi qu'à toutes ses extensions possibles (et de la même façon pour $\uparrow$). On va donc recommencer à définir une relation qui prenne en compte ce problème.

Pour résoudre ce problème, on va choisir d'égaliser les termes dont la σ-forme normale est égale. On appelle σ l'ensemble des règles sauf B. La σ-forme normale d'un terme est le terme résultant de l'application de la clôture transitive de σ. On sait qu'une telle forme normale existe puisque σ est fortement normalisant (voir [1]). On notera $\sigma(t)$ la σ-forme normale de t.

On commence par définir une notion de redexabilité des termes. L'intuition est de repérer les termes "méchants", c'est-à-dire ceux qui peuvent engendrer des B-redex.

Définition 7.2.7

On dira qu'un terme est potentiellement redexable (et on notera $PR(t)$) si il contient des noeuds application ou des noeuds λ.

On définit ensuite la relation $\preccurlyeq$ qui va nous garantir que si $u \preccurlyeq t$ alors u a la même redexabilité que t.

$$
\begin{array}{lcll}
\text{pour tout } n \text{ et } m & & n \preccurlyeq m & \\
t \preccurlyeq t' \text{ et } u \preccurlyeq u' & \Rightarrow & (t\ u) \preccurlyeq (t'\ u') & \\
t \preccurlyeq t' & \Rightarrow & \lambda t \preccurlyeq \lambda t' & \\
t \preccurlyeq t' & \Rightarrow & t \preccurlyeq t'[s] & \text{si } \neg PR(s) \\
t \preccurlyeq t' \text{ et } s \preccurlyeq s' & \Rightarrow & t[s] \preccurlyeq t'[s'] &
\end{array}
$$

$$
\begin{array}{lcll}
 & & \uparrow \preccurlyeq \uparrow & \\
 & & id \preccurlyeq id & \\
 & & id \preccurlyeq s & \text{si } \neg PR(s) \\
t \preccurlyeq t' \text{ et } s \preccurlyeq s' & \Rightarrow & t \cdot s \preccurlyeq t' \cdot s' & \\
s \preccurlyeq s' & \Rightarrow & s \preccurlyeq s' \circ s_1 & \text{si } \neg PR(s_1) \\
s \preccurlyeq s' & \Rightarrow & s \preccurlyeq s_1 \circ s' & \text{si } \neg PR(s_1) \\
s_1 \preccurlyeq s_1' \text{ et } s_2 \preccurlyeq s_2' & \Rightarrow & s_1 \circ s_2 \preccurlyeq s_1' \circ s_2' &
\end{array}
$$

On définit la relation $\lessdot$ de la façon suivante.

Définition 7.2.8

Pour tout t et u, $u \lessdot t \iff u \preccurlyeq t$ et $\sigma(t) = \sigma(u)$.

On remarque que l'on a toujours $t \lessdot t$. Voici quelques lemmes qui nous seront utiles pour le lemme d'initialisation. Le premier exprime le fait qu'on ne change pas la σ-forme normale en supprimant une substitution $[id]$.

Lemme 7.2.9

Pour tout t, on a $\sigma(t) = \sigma(t[id])$.

Preuve : Voir [1]. ■

Lemme 7.2.10

Pour tout t, on a $\sigma(\mathcal{U}_0^1(t)) = \mathcal{U}_0^1(\sigma(t))$.

Preuve : Comme l'application de $\mathcal{U}_0^1()$ ne fait qu'augmenter les indices des variables libres, elle est orthogonale à la réduction des substitutions, qui opèrent sur les variables liées. ■

Le lemme suivant exprime que la σ-forme normale d'un terme $t[\uparrow]$ est la même que celle de $\mathcal{U}_0^1(t)$.

Lemme 7.2.11

Pour tout t, on a $\sigma(\mathcal{U}_0^1(t)) = \sigma(t[\uparrow])$.

Preuve : En fait, on va prouver un résultat plus général. Soit $\Uparrow(s) = 1 \cdot (s \circ \uparrow)$, on démontre que pour tout t et i, on a $\sigma(\mathcal{U}_i^1(t)) = \sigma(t[\Uparrow^i(\uparrow)])$. Puisque $\sigma(t[\Uparrow^i(\uparrow)]) = \sigma(\sigma(t)[\Uparrow^i(\uparrow)])$, il suffit d'effectuer la preuve pour t en σ-forme normale. On procède par induction sur celle-ci.

- $t = u\ v$: alors $\sigma((u\ v)[\Uparrow^i(\uparrow)]) = \sigma((u[\Uparrow^i(\uparrow)])\ \sigma(v[\Uparrow^i(\uparrow)])$, et le résultat suit par hypothèse d'induction.
- $t = \lambda u$: alors $\sigma((\lambda u)[\Uparrow^i(\uparrow)]) = \lambda(\sigma(u[\Uparrow^{i+1}(\uparrow)]))$ et $\sigma(\mathcal{U}_i^1(\lambda u)) = \lambda(\sigma(\mathcal{U}_{i+1}^1(u)))$. On conclut par hypothèse d'induction.
- $t = 1$: deux cas sont possibles,
  - soit $i = 0$, auquel cas $\sigma(\mathcal{U}_0^1(1)) = \sigma(2)$ et $\sigma(1[\uparrow]) = \sigma(2)$.
  - soit $i > 0$, auquel cas $\sigma(\mathcal{U}_i^1(1)) = \sigma(1) = 1$ et $\sigma(1[\Uparrow^i(\uparrow)]) = \sigma(1[1 \cdot (\Uparrow^{i-1}(\uparrow) \circ \uparrow)]) =_{VarCons} \sigma(1) = 1$.
- $t = n > 1$: deux cas sont possibles,

– soit $i < n$, auquel cas $\sigma(\mathcal{U}_i^1(n)) = \sigma(n+1) = \sigma(1\underbrace{[\uparrow]...[\uparrow]}_{n}) =_{Clos} 1\underbrace{[\uparrow \circ ... \circ \uparrow]}_{n}$ et

$$\sigma(n[\Uparrow^i(\uparrow)]) = \sigma(1\underbrace{[\uparrow]...[\uparrow]}_{n-1}[\Uparrow^i(\uparrow)]) =_{Clos} \sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-1} \circ \Uparrow^i(\uparrow)])$$

On va prouver que ce dernier terme est égal à $1\underbrace{[\uparrow \circ ... \circ \uparrow]}_{n}$ par induction sur i :

* $i = 0$: $\sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-1} \circ \uparrow]) = 1\underbrace{[\uparrow \circ ... \circ \uparrow]}_{n}$
* $i > 0$:

$$\sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-1} \circ \Uparrow^i(\uparrow)]) = \sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-1} \circ (1 \cdot (\Uparrow^{i-1}(\uparrow) \circ \uparrow))]) = \sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow) \circ \uparrow])$$

Par la règle *Clos*, on a

$$\sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow) \circ \uparrow]) = \sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow)][\uparrow]) = \sigma(\sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow)])[\uparrow]).$$

Puisque $i < n$, alors $i - 1 < n - 1$ et on peut appliquer l'hypothèse d'induction sur i, ce qui nous donne

$$\sigma(\sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow)])[\uparrow]) = \sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-1}][\uparrow]) =_{Clos} 1\underbrace{[\uparrow \circ ... \circ \uparrow]}_{n}.$$

– soit $i \geq n$, auquel cas $\sigma(\mathcal{U}_i^1(n)) = \sigma(n) = \sigma(1\underbrace{[\uparrow]...[\uparrow]}_{n-1}) =_{Clos} 1\underbrace{[\uparrow \circ ... \circ \uparrow]}_{n-1}$ et

$$\sigma(n[\Uparrow^i(\uparrow)]) = \sigma(1\underbrace{[\uparrow]...[\uparrow]}_{n-1}[\Uparrow^i(\uparrow)]) =_{Clos} \sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-1} \circ \Uparrow^i(\uparrow)])$$

On va prouver que ce dernier terme est égal à $1\underbrace{[\uparrow \circ ... \circ \uparrow]}_{n-1}$ par induction sur i :

* $i = 0$: impossible car $i \geq n > 1$.
* $i = 1$: impossible car $i \geq n > 1$.
* $i = 2$: on a forcément $n = 2 = 1[\uparrow]$, ce qui nous donne

$$\begin{array}{lll} \sigma(1[\uparrow \circ \Uparrow^2(\uparrow)]) & = & \sigma(1[\uparrow \circ (1 \cdot ((1 \cdot (\uparrow \circ \uparrow)) \circ \uparrow))]) \\ & =_{ShiftCons} & \sigma(1[(1 \cdot (\uparrow \circ \uparrow)) \circ \uparrow]) \\ & =_{Map} & \sigma(1[1[\uparrow] \cdot (\uparrow \circ \uparrow \circ \uparrow)]) \\ & =_{VarCons} & 1[\uparrow] \end{array}$$

* $i > 2$:

$$\sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-1} \circ \Uparrow^i(\uparrow)]) = \sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-1} \circ (1 \cdot (\Uparrow^{i-1}(\uparrow) \circ \uparrow))]) = \sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow) \circ \uparrow])$$

Par la règle *Clos*, on a $\sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow) \circ \uparrow]) = \sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow)][\uparrow])$. Puisque $i \geq n$, alors $i - 1 \geq n - 1$ et on peut appliquer l'hypothèse d'induction sur i, ce qui nous donne $\sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-2} \circ \Uparrow^{i-1}(\uparrow)][\uparrow]) = \sigma(1\underbrace{[\uparrow \circ ... \circ \uparrow}_{n-2}][\uparrow]) =_{Clos} 1\underbrace{[\uparrow \circ ... \circ \uparrow]}_{n-1}$.

■

On va aussi avoir besoin d'un lemme pour égaliser des σ-formes normales.

Lemme 7.2.12

> Pour tout t, u, s et s' tous en σ-forme normale, si $\sigma(t[1\cdot(s\circ\uparrow)]) = \sigma(t'[1\cdot(s'\circ\uparrow)])$ alors $\sigma(t[u\cdot s]) = \sigma(t'[u\cdot s'])$.

Preuve : Pour se convaincre du résultat, il suffit de regarder les variables libres de t et t'. Dire que les σ-formes normales sont égales revient à dire que les termes sont égaux lorsqu'on remplace leur variables libres par les termes des substitutions. La première forme de substitution, $[1\cdot(s\circ\uparrow)]$, laisse la première variable libre inchangée et modifie les autres. La seconde substitution, $[u\cdot s]$, remplace la première variable libre par u dans les deux termes. ■

On peut maintenant passer au lemme initial de notre simulation.

Lemme 7.2.13 (Initialisation)

> Pour tout t, il existe u tel que $Ateb(t) \to_B^* u$ et $u \lessdot t$.

Preuve : On raisonne par induction sur le nombre d'étapes du calcul de $Ateb(t)$.
- Si $t = n$, alors $Ateb(t) = n$ et il suffit de prendre $u = n$.
- Si $t = (t_1\ t_2)$, alors $Ateb(t) = (Ateb(t_1)\ Ateb(t_2))$. Par hypothèse d'induction, il existe u_1 et u_2 tels que $Ateb(t_1) \to_B^* u_1$ et $u_1 \lessdot t_1$ et $Ateb(t_2) \to_B^* u_2$ et $u_2 \lessdot t_2$. On conclut en prenant $u = (u_1\ u_2)$.
- Si $t = \lambda t'$, alors $Ateb(t) = \lambda Ateb(t')$. Par hypothèse d'induction, il existe u' tel que $Ateb(t') \to_B^* u'$. On conclut en prenant $u = \lambda u'$.
- Si $t = t'[id]$, alors $Ateb(t) = Ateb(t')$. Par hypothèse d'induction, il existe u' tel que $Ateb(t') \to_B^* u'$. On prend $u = u'$ et on conclut à l'aide du lemme 7.2.9.
- Si $t = t'[\uparrow]$, alors $Ateb(t) = \mathcal{U}_0^1(Ateb(t'))$. Par hypothèse d'induction, il existe u' tel que $Ateb(t') \to_B^* u'$. On prend $u = \mathcal{U}_0^1(u')$ et on conclut à l'aide des lemmes 7.2.10 et 7.2.11.
- Si $t = t'[s\circ s']$, alors $Ateb(t) = Ateb(t'[s][s'])$. Par hypothèse d'induction, il existe u' tel que $Ateb(t'[s][s']) \to_B^* u'$. On a quatre cas possibles suivant les valeurs de $PR(s)$ et $PR(s')$, mais dans tous les cas, on peut conclure en prenant $u = u'$.
- Si $t = t_1[t_2\cdot s]$, alors $Ateb(t) = Ateb((\lambda t_1)[s])\ Ateb(t_2)$. Par hypothèse d'induction, il existe u_1 et u_2 tels que $Ateb((\lambda t_1)[s]) \to_B^* u_1$ et $u_1 \lessdot (\lambda t_1)[s]$ et $Ateb(t_2) \to_B^* u_2$ et $u_2 \lessdot t_2$. Deux cas sont possibles suivant la forme de u_1.
  - Si $u_1 = \lambda v_1$ (ce qui signifie que l'on a $\neg PR(s)$), alors on prend $u = v_1[u_2\cdot id]$. Il nous faut vérifier que $u \lessdot t$ et la difficulté est de montrer $\sigma(u) = \sigma(t)$. Par hypothèse, on a $\sigma(\lambda v_1) = \sigma((\lambda t_1)[s])$. Il est évident que $\sigma(\lambda v_1) = \sigma((\lambda v_1)[id]) = \lambda(\sigma(\sigma(v_1)[1\cdot(id\circ\uparrow)]))$. D'autre part on a $\sigma((\lambda t_1)[s]) = \lambda(\sigma(\sigma(t_1)[1\cdot(\sigma(s)\circ\uparrow)]))$, ce qui nous donne $\sigma(\sigma(v_1)[1\cdot(id\circ\uparrow)]) = \sigma(\sigma(t_1)[1\cdot(\sigma(s)\circ\uparrow)])$. On est dans de bonnes conditions pour appliquer le lemme 7.2.12 en insérant le terme $\sigma(u_2)$ qui est égal à $\sigma(t_2)$ par hypothèse, ce qui nous donne $\sigma(\sigma(v_1)[\sigma(u_2)\cdot id]) = \sigma(\sigma(t_1)[\sigma(t_2)\cdot s])$ et nous permet de conclure.
  - Si $u_1 = (\lambda v_1)[s_1]$, alors on prend $u = v_1[u_2\cdot s_1]$ et on conclut de la même façon que dans le point précédent avec l'aide du lemme 7.2.12.

■

7.2.3 Simulation

On peut à présent passer à la simulation proprement dite. On sépare l'ensemble des règles de réduction en deux : la règle B et les autres, qu'on regroupe sous l'appellation R_2. On a bien R_2 fortement normalisant (voir [1, 22, 67, 72]). On veut établir les diagrammes suivants :

$$\begin{array}{ccc} t & \to_B & t' \\ \forall & & \forall \\ u & \to^+_{\lambda\sigma} & u' \end{array} \qquad\qquad \begin{array}{ccc} t & \to_{R_2} & t' \\ \forall & & \forall \\ u & \to^*_{\lambda\sigma} & u' \end{array}$$

On commence par regarder la simulation de B, puis celle des autres règles. Les deux propriétés suivantes correspondent aux lemmes généraux 4.3.2 et 4.3.3.

Lemme 7.2.14

Pour tout $t \to_B t'$, pour tout $u \lessdot t$ il existe u' tel que $u \to_B u'$ et $u' \lessdot t'$.

$$\begin{array}{ccc} t & \to_B & t' \\ \forall & & \forall \\ u & \to_B & u' \end{array}$$

Preuve : $u \lessdot t$ nous donne $u \preccurlyeq t$ et $\sigma(u) = \sigma(t)$. De plus, la relation $\preccurlyeq$ définie à l'aide du prédicat PR nous assure que le redex réduit par B apparaît dans u. On a donc $u \to_B u'$ et on veut prouver $u' \lessdot t'$. S'il est évident que $u' \preccurlyeq t'$ vient directement du fait que $u \preccurlyeq t$, il n'en est pas de même pour l'égalité des σ-formes normales. On veut obtenir $\sigma(u') = \sigma(t')$ avec pour hypothèse $\sigma(u) = \sigma(t)$. On pose $t = C[(\lambda v)\ w]$, ce qui nous donne $t' = C[v[w \cdot id]]$ et $u = C'[(\lambda v')\ w']$. Deux cas sont possibles :

- le redex $(\lambda v)\ w$ n'apparaît pas dans $\sigma(t)$. C'est-à-dire que le calcul de $\sigma(t)$ peut être découpé de la façon suivante :

$$C[(\lambda v)\ w] \to^*_\sigma C_1[\uparrow \circ (C_2[(\lambda v)\ w] \cdot s)] \to_{ShiftCons} C_1[s] \to^*_\sigma \sigma(t)$$

  Puisque $\sigma(t) = \sigma(u)$, le même scénario a lieu pour u. De la même façon que pour le redex, le réduit va être effacé de t' et de u' et on obtient bien $\sigma(u') = \sigma(t')$.

- le redex $(\lambda v)\ w$ apparaît dans $\sigma(t)$. On écrira, pour tout t, $\underline{t}$ pour $\sigma(t)$, afin d'alléger les calculs. On a alors les égalités suivantes :

$$\begin{array}{rcl} \underline{t} & = & \sigma(C[(\lambda v)\ w]) \\ & = & C_1[\sigma(((\lambda v)\ w)[\underline{s}])] \\ & = & C_1[(\lambda\sigma(\underline{v}[1 \cdot (\underline{s} \circ \uparrow)]))\ \sigma(\underline{w}[\underline{s}])] \end{array}$$

  Et, de la même façon, $\underline{u} = C'_1[(\lambda\sigma(\underline{v'}[1 \cdot (\underline{s'} \circ \uparrow)]))\ \sigma(\underline{w'}[\underline{s'}])]$. De $\underline{t} = \underline{u}$ on déduit $C_1 = C'_1$, $\sigma(\underline{v}[1 \cdot (\underline{s} \circ \uparrow)]) = \sigma(\underline{v'}[1 \cdot (\underline{s'} \circ \uparrow)])$ et $\sigma(\underline{w}[\underline{s}]) = \sigma(\underline{w'}[\underline{s'}])$. On regarde à présent les termes $\underline{t'}$ et $\underline{u'}$:

$$\begin{array}{rcl} \underline{t'} = & \sigma(C[v[w \cdot id]]) \\ & = & C_1[\sigma(\underline{v}[\underline{w} \cdot id][\underline{s}])] \\ =_{Clos} & C_1[\sigma(\underline{v}[(\underline{w} \cdot id) \circ \underline{s}])] \\ =_{Map} & C_1[\sigma(\underline{v}[\underline{w}[\underline{s}] \cdot \underline{s}])] \\ = & C_1[\sigma(\underline{v}[\sigma(\underline{w}[\underline{s}]) \cdot \underline{s}])] \end{array}$$

  Et, de la même façon, $\underline{u'} = C'_1[\sigma(\underline{v'}[\sigma(\underline{w'}[\underline{s'}]) \cdot \underline{s'}])]$. Des égalités précédemment définies on obtient $\underline{u'} = C'_1[\sigma(\underline{v'}[\sigma(\underline{w}[\underline{s}]) \cdot \underline{s'}])]$, et on peut conclure à l'aide du lemme 7.2.12.

■

Lemme 7.2.15

Pour tout $t \to_{R_2} t'$, pour tout $u \lessdot t$ il existe u' tel que $u \to^*_{\lambda\sigma} u'$ et $u' \lessdot t'$.

$$\begin{array}{ccc} t & \to_{R_2} & t' \\ \forall & & \forall \\ u & \to^*_{\lambda\sigma} & u' \end{array}$$

Preuve : La preuve est facile car R_2 est exactement le calcul des substitutions σ. $u \lessdot t$ nous donne $u \preccurlyeq t$ et $\sigma(u) = \sigma(t)$, et, d'autre part, $t \to_{R_2} t'$ implique $\sigma(t) = \sigma(t')$. Deux cas sont possibles suivant que le redex apparaît aussi dans u ou non. S'il n'apparaît pas, on prend $u' = u$ et on conclut directement. Sinon, on réduit le redex avec la même règle et on conclut en calculant $\sigma(u') = \sigma(u) = \sigma(t')$. ■

Puisque la fonction $Ateb(t)$ nous renvoie un terme t' qui se réduit vers $u \lessdot t$ (par le lemme 7.2.13), on peut utiliser le théorème 4.4.2. On obtient que PSN implique SN pour le $\lambda\sigma$-calcul, mais comme celui-ci n'a pas la propriété PSN, on ne peut pas en tirer de conclusion de normalisation forte.

Chapitre 8

Application à $\lambda\sigma_n$

Voici une version avec noms du calcul $\lambda\sigma$ [1]. Les mêmes remarques que celles du chapitre précédent s'appliquent à ce calcul.

8.1 Présentation du calcul

Les termes du $\lambda\sigma_n$-calcul sont donnés par la grammaire suivante :

$$t ::= x \mid (t\ t) \mid \lambda x.t \mid t[s]$$
$$s ::= id \mid (t/x) \cdot s \mid s \circ s$$

Voici l'ensemble des règles de réduction :

$(\lambda x.t)u$	$\rightarrow_B$	$t[(u/x) \cdot id]$	
$(t\ u)[s]$	$\rightarrow_{App}$	$(t[s])\ (u[s])$	
$(\lambda x.t)[s]$	$\rightarrow_{Lambda}$	$\lambda y.(t[(y/x) \cdot s])$	avec y variable fraîche
$x[id]$	$\rightarrow_{VarId}$	x	
$x[(t/x) \cdot s]$	$\rightarrow_{VarCons1}$	t	
$x[(t/y) \cdot s]$	$\rightarrow_{VarCons2}$	$x[s]$	$(x \neq y)$
$t[s][s']$	$\rightarrow_{Clos}$	$t[s \circ s']$	
$id \circ s$	$\rightarrow_{IdL}$	s	
$((t/x) \cdot s) \circ s'$	$\rightarrow_{Map}$	$(t[s']/x) \cdot (s \circ s')$	
$(s_1 \circ s_2) \circ s_3$	$\rightarrow_{Ass}$	$s_1 \circ (s_2 \circ s_3)$	

Regardons les règles de typage (voir la remarque 6.1.1 en ce qui concerne le symbole $\triangleright$) :

$$\frac{}{\Delta, x : A, \Gamma \vdash x : A} \qquad \frac{\Gamma \vdash s \triangleright \Gamma' \quad \Gamma' \vdash t : A}{\Gamma \vdash t[s] : A}$$

$$\frac{\Gamma \vdash t : B \rightarrow A \quad \Gamma \vdash u : B}{\Gamma \vdash (t\ u) : A} \qquad \frac{x : B, \Gamma \vdash t : A}{\Gamma \vdash \lambda x.t : B \rightarrow A}$$

$$\frac{}{\Gamma \vdash id \triangleright \Gamma}$$

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash s \triangleright \Gamma'}{\Gamma \vdash (t/x) \cdot s \triangleright x : A, \Gamma'} \qquad \frac{\Gamma \vdash s' \triangleright \Gamma'' \quad \Gamma'' \vdash s \triangleright \Gamma'}{\Gamma \vdash s \circ s' \triangleright \Gamma'}$$

8.2 PSN implique SN

On définit la fonction $Ateb$ de la façon suivante :

$$\begin{array}{lcl} Ateb(x) & = & x \\ Ateb(t\ u) & = & Ateb(t)\ Ateb(u) \\ Ateb(\lambda x.t) & = & \lambda x.Ateb(t) \\ Ateb(t[id]) & = & Ateb(t) \\ Ateb(t[s \circ s']) & = & Ateb(t[s][s']) \\ Ateb(t[(t'/x) \cdot s]) & = & Ateb((\lambda x.t)[s])\ Ateb(t') \end{array}$$

Il est évident que pour tout t, $Ateb(t)$ ne contient pas de substitutions. Il nous faut vérifier que le terme obtenu est typable.

Lemme 8.2.1

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash Ateb(t) : A$$

Preuve : On procède par induction sur le nombre d'étapes du calcul de $Ateb(t)$.

- $t = x$ et

$$\frac{}{x : A, \Delta \vdash x : A}$$

  On a alors $Ateb(t) = x$ et la dérivation de typage est la même.

- $t = (u\ v)$ et

$$\frac{\Gamma \vdash u : B \to A \quad \Gamma \vdash v : B}{\Gamma \vdash (u\ v) : A}$$

  Par hypothèse d'induction, on a $\Gamma \vdash Ateb(u) : B \to A$ et $\Gamma \vdash Ateb(v) : B$. On peut typer $Ateb(t) = Ateb(u)\ Ateb(v)$ de la façon suivante

$$\frac{\Gamma \vdash Ateb(u) : B \to A \quad \Gamma \vdash Ateb(v) : B}{\Gamma \vdash (Ateb(u)\ Ateb(v)) : A}$$

- $t = \lambda x.u$ et

$$\frac{x : B, \Gamma \vdash u : A}{\Gamma \vdash \lambda x.u : B \to A}$$

  Par hypothèse d'induction, on a $\Gamma, x : B \vdash Ateb(u) : A$. On peut typer $Ateb(t) = \lambda x.Ateb(u)$ de la façon suivante

$$\frac{x : B, \Gamma \vdash Ateb(u) : A}{\Gamma \vdash \lambda x.Ateb(u) : B \to A}$$

- $t = u[id]$ et

$$\frac{\Gamma \vdash id \triangleright \Gamma \quad \Gamma \vdash u : A}{\Gamma \vdash u[id] : A}$$

  On conclut directement par hypothèse d'induction.

- $t = u[(v/x) \cdot s]$ et

$$\dfrac{\dfrac{\Gamma \vdash s \triangleright \Gamma' \quad \Gamma \vdash v : B}{\Gamma \vdash (v/x) \cdot s \triangleright x : B, \Gamma'} \quad x : B, \Gamma' \vdash u : A}{\Gamma \vdash u[(v/x) \cdot s] : A}$$

Par hypothèse d'induction, on a $\Gamma \vdash Ateb((\lambda x.u)[s]) : B \to A$ et $\Gamma \vdash Ateb(v) : B$. On conclut en typant

$$\dfrac{\Gamma \vdash Ateb((\lambda x.u)[s]) : B \to A \quad \Gamma \vdash Ateb(v) : B}{\Gamma \vdash Ateb((\lambda x.u)[s])\ Ateb(v) : A}$$

- $t = u[s \circ s']$, alors on conclut directement par hypothèse d'induction.

■

8.2.1 Définition de la relation $\lessdot$

Nous allons procéder de la même façon que dans le chapitre précédent pour $\lambda\sigma$, mais plus simplement, puisque nous n'avons pas de $\uparrow$. On utilise la même notion de redexabilité (voir définition 7.2.7) et la relation $\preccurlyeq$ est définie de manière identique (sans les $\uparrow$). On définit la relation $\lessdot$ de la façon suivante.

Définition 8.2.2

Pour tout t et u, $u \lessdot t \iff u \preccurlyeq t$ et $\sigma(t) = \sigma(u)$.

On aura besoin du lemme 7.2.9 et il nous faut reformuler le lemme 7.2.12.

Lemme 8.2.3

Pour tout t, u, s et s' tous en σ-forme normale, si $\sigma(t[(y/x)\cdot s]) = \sigma(t'[(y/x)\cdot s'])$ alors $\sigma(t[(u/x)\cdot s]) = \sigma(t'[(u/x) \cdot s'])$.

Preuve : Pour se convaincre du résultat, il suffit de regarder les variables libres de t et t'. Dire que les σ-formes normales sont égales revient à dire que les termes sont égaux lorsqu'on remplace leur variables libres par les termes des substitutions. Comme on ne fait que changer le terme substitué, l'égalité est préservée. ■

Voici le lemme initial de notre simulation.

Lemme 8.2.4 (Initialisation)

Pour tout t, il existe u tel que $Ateb(t) \to_B^* u$ et $u \lessdot t$.

Preuve : On raisonne par induction sur le nombre d'étapes du calcul de $Ateb(t)$.
- Si $t = x$, alors $Ateb(t) = x$ et il suffit de prendre $u = x$.
- Si $t = (t_1\ t_2)$, alors $Ateb(t) = (Ateb(t_1)\ Ateb(t_2))$. Par hypothèse d'induction, il existe u_1 et u_2 tels que $Ateb(t_1) \to_B^* u_1$ et $u_1 \lessdot t_1$ et $Ateb(t_2) \to_B^* u_2$ et $u_2 \lessdot t_2$. On conclut en prenant $u = (u_1\ u_2)$.
- Si $t = \lambda x.t'$, alors $Ateb(t) = \lambda x.Ateb(t')$. Par hypothèse d'induction, il existe u' tel que $Ateb(t') \to_B^* u'$. On conclut en prenant $u = \lambda x.u'$.
- Si $t = t'[id]$, alors $Ateb(t) = Ateb(t')$. Par hypothèse d'induction, il existe u' tel que $Ateb(t') \to_B^* u'$. On prend $u = u'$ et on conclut à l'aide du lemme 7.2.9.
- Si $t = t'[s \circ s']$, alors $Ateb(t) = Ateb(t'[s][s'])$. Par hypothèse d'induction, il existe u' tel que $Ateb(t'[s][s']) \to_B^* u'$. On a quatre cas possibles suivant les valeurs de $PR(s)$ et $PR(s')$, mais dans tous les cas, on peut conclure en prenant $u = u'$.

– Si $t = t_1[(t_2/x)\cdot s]$, alors $Ateb(t) = Ateb((\lambda x.t_1)[s])\ Ateb(t_2)$. Par hypothèse d'induction, il existe u_1 et u_2 tels que $Ateb((\lambda x.t_1)[s]) \to_B^* u_1$ et $u_1 \lessdot (\lambda x.t_1)[s]$ et $Ateb(t_2) \to_B^* u_2$ et $u_2 \lessdot t_2$. Deux cas sont possibles suivant la forme de u_1.

- Si $u_1 = \lambda x.v_1$ (ce qui signifie que l'on a $\neg PR(s)$), alors on prend $u = v_1[(u_2/x) \cdot id]$. Il nous faut vérifier que $u \lessdot t$ et la difficulté est de montrer $\sigma(u) = \sigma(t)$. Par hypothèse, on a $\sigma(\lambda x.v_1) = \sigma((\lambda x.t_1)[s])$. Il est évident que $\sigma(\lambda x.v_1) = \sigma((\lambda x.v_1)[id]) = \lambda y.(\sigma(\sigma(v_1)[(y/x)\cdot id]))$. D'autre part on a $\sigma((\lambda x.t_1)[s]) = \lambda y.(\sigma(\sigma(t_1)[(y/x) \cdot \sigma(s)]))$, ce qui nous donne $\sigma(\sigma(v_1)[(y/x)\cdot id]) = \sigma(\sigma(t_1)[(y/x)\cdot\sigma(s)])$. On est dans de bonnes conditions pour appliquer le lemme 8.2.3 en insérant le terme $\sigma(u_2)$ qui est égal à $\sigma(t_2)$ par hypothèse, ce qui nous donne $\sigma(\sigma(v_1)[(\sigma(u_2)/x) \cdot id]) = \sigma(\sigma(t_1)[(\sigma(t_2)/x) \cdot s])$ et nous permet de conclure.
- Si $u_1 = (\lambda x.v_1)[s_1]$, alors on prend $u = v_1[(u_2/x) \cdot s_1]$ et on conclut de la même façon que dans le point précédent avec l'aide du lemme 8.2.3.

■

8.2.2 Simulation

On peut à présent passer à la simulation proprement dite, qui va être effectuée de la même façon que pour $\lambda\sigma$. On sépare l'ensemble des règles de réduction en deux : la règle B et les autres, qu'on regroupe sous l'appellation R_2. On a bien R_2 fortement normalisant. On veut établir les diagrammes suivants :

$$\begin{array}{ccc} t & \to_B & t' \\ \forall & & \forall \\ u & \to^+_{\lambda\sigma_n} & u' \end{array} \qquad\qquad \begin{array}{ccc} t & \to_{R_2} & t' \\ \forall & & \forall \\ u & \to^*_{\lambda\sigma_n} & u' \end{array}$$

On commence par regarder la simulation de B, puis celle des autres règles. Les deux propriétés suivantes correspondent aux lemmes généraux 4.3.2 et 4.3.3.

Lemme 8.2.5

Pour tout $t \to_B t'$, pour tout $u \lessdot t$ il existe u' tel que $u \to_B u'$ et $u' \lessdot t'$.

$$\begin{array}{ccc} t & \to_B & t' \\ \forall & & \forall \\ u & \to_B & u' \end{array}$$

Preuve : $u \lessdot t$ nous donne $u \preccurlyeq t$ et $\sigma(u) = \sigma(t)$. De plus, la relation $\preccurlyeq$ définie à l'aide du prédicat PR nous assure que le redex réduit par B apparaît dans u. On a donc $u \to_B u'$ et on veut prouver $u' \lessdot t'$. S'il est évident que $u' \preccurlyeq t'$ vient directement du fait que $u \preccurlyeq t$, il n'en est pas de même pour l'égalité des σ-formes normales. On veut obtenir $\sigma(u') = \sigma(t')$ avec pour hypothèse $\sigma(u) = \sigma(t)$. On pose $t = C[(\lambda x.v)\ w]$, ce qui nous donne $t' = C[v[(w/x) \cdot id]]$ et $u = C'[(\lambda x.v')\ w']$. Deux cas sont possibles :

- le redex $(\lambda x.v)\ w$ n'apparaît pas dans $\sigma(t)$. C'est-à-dire que le calcul de $\sigma(t)$ peut être découpé de la façon suivante :

$$C[(\lambda x.v)\ w] \to^*_\sigma C_1[y[((C_2[(\lambda x.v)\ w])/x) \cdot s]] \to_{VarCons2} C_1[y[s]] \to^*_\sigma \sigma(t)$$

  Puisque $\sigma(t) = \sigma(u)$, le même scénario a lieu pour u. De la même façon que pour le redex, le réduit va être effacé de t' et de u' et on obtient bien $\sigma(u') = \sigma(t')$.
- le redex $(\lambda x.v)\ w$ apparaît dans $\sigma(t)$. Afin d'alléger les notations dans les calculs, on écrira, pour tout t, $\underline{t}$ pour $\sigma(t)$. On a les égalités suivantes :

$$\begin{aligned}\underline{\mathrm{t}} &= \sigma(C[(\lambda x.v)\ w]) \\ &= C_1[\sigma(((\lambda x.v)\ w)[\underline{\mathrm{s}}])] \\ &= C_1[(\lambda y.\sigma(\underline{\mathrm{v}}[(y/x)\cdot\underline{\mathrm{s}}]))\ \sigma(\underline{\mathrm{w}}[\underline{\mathrm{s}}])]\end{aligned}$$

Et, de la même façon, $\underline{\mathrm{u}} = C_1'[(\lambda y.\sigma(\underline{\mathrm{v'}}[(y/x)\cdot\underline{\mathrm{s'}}]))\ \sigma(\underline{\mathrm{w'}}[\underline{\mathrm{s'}}])]$. De $\underline{\mathrm{t}} = \underline{\mathrm{u}}$ on déduit $C_1 = C_1'$, $\sigma(\underline{\mathrm{v}}[(y/x)\cdot\underline{\mathrm{s}}]) = \sigma(\underline{\mathrm{v'}}[(y/x)\cdot\underline{\mathrm{s'}}])$ et $\sigma(\underline{\mathrm{w}}[\underline{\mathrm{s}}]) = \sigma(\underline{\mathrm{w'}}[\underline{\mathrm{s'}}])$. On regarde à présent les termes $\underline{\mathrm{t'}}$ et $\underline{\mathrm{u'}}$:

$$\begin{aligned}\underline{\mathrm{t'}} &= \sigma(C[v[w\cdot id]]) \\ &= C_1[\sigma(\underline{\mathrm{v}}[\underline{\mathrm{w}}\cdot id][\underline{\mathrm{s}}])] \\ =_{Clos}\ & C_1[\sigma(\underline{\mathrm{v}}[(\underline{\mathrm{w}}\cdot id)\circ\underline{\mathrm{s}}])] \\ =_{Map}\ & C_1[\sigma(\underline{\mathrm{v}}[\underline{\mathrm{w}}[\underline{\mathrm{s}}]\cdot\underline{\mathrm{s}}])] \\ &= C_1[\sigma(\underline{\mathrm{v}}[\sigma(\underline{\mathrm{w}}[\underline{\mathrm{s}}])\cdot\underline{\mathrm{s}}])]\end{aligned}$$

Et, de la même façon, $\underline{\mathrm{u'}} = C_1'[\sigma(\underline{\mathrm{v'}}[\sigma(\underline{\mathrm{w'}}[\underline{\mathrm{s'}}])\cdot\underline{\mathrm{s'}}])]$. Des égalités précédemment définies on obtient $\underline{\mathrm{u'}} = C_1'[\sigma(\underline{\mathrm{v'}}[\sigma(\underline{\mathrm{w}}[\underline{\mathrm{s}}])\cdot\underline{\mathrm{s'}}])]$, et on peut conclure à l'aide du lemme 8.2.3.

■

Lemme 8.2.6

Pour tout $t \to_{R_2} t'$, pour tout $u \lessdot t$ il existe u' tel que $u \to^*_{\lambda\sigma_n} u'$ et $u' \lessdot t'$.

$$\begin{array}{ccc} t & \to_{R_2} & t' \\ \diagdown\!\!\diagup & & \diagdown\!\!\diagup \\ u & \to^*_{\lambda\sigma_n} & u' \end{array}$$

Preuve : La preuve est facile car R_2 est exactement le calcul des substitutions σ. $u \lessdot t$ nous donne $u \preccurlyeq t$ et $\sigma(u) = \sigma(t)$, et, d'autre part, $t \to_{R_2} t'$ implique $\sigma(t) = \sigma(t')$. Deux cas sont possibles suivant que le redex apparaît aussi dans u ou non. S'il n'apparaît pas, on prend $u' = u$ et on conclut directement. Sinon, on réduit le redex avec la même règle et on conclut en calculant $\sigma(u') = \sigma(u) = \sigma(t')$. ■

Puisque la fonction $Ateb(t)$ nous renvoie un terme t' qui se réduit vers $u \lessdot t$ (par le lemme 8.2.4), on peut utiliser le théorème 4.4.2. On obtient que PSN implique SN pour le $\lambda\sigma_n$-calcul, mais comme celui-ci n'a pas la propriété PSN, on ne peut pas en tirer de conclusion de normalisation forte.

Troisième partie

Normalisation forte du $\overline{\lambda}\mu\tilde{\mu}$-calcul avec substitutions explicites

Le $\overline{\lambda}\mu\tilde{\mu}$-calcul [23] est une variante du $\lambda\mu$-calcul [63] qui propose une notation de termes pour le calcul des séquents de la logique classique. Il met en évidence des symétries terme/contexte et appel-par-nom/appel-par-valeur. Ses deux règles de réduction principales forment une paire critique, ce qui rend le calcul non-déterministe (non-confluent) et soulève des difficultés dans les preuves de normalisation forte : une preuve par réductibilité utilisant une définition naïve des candidats de réductibilité tomberait dans une boucle symétrique d'induction mutuelle.

Une preuve de normalisation forte par réductibilité se fait en plusieurs étapes, comme nous l'avons vu dans le chapitre 3. La définition des candidats de réductibilité doit être choisie avec le plus grand soin afin de faciliter les preuves ultérieures. Pour illustrer les difficultés dues à la symétrie, regardons ce qui se passe lors de la preuve du lemme d'adéquation pour le λ-calcul simplement typé. Dans ce lemme, on part d'un terme typé et on montre, par induction structurelle, qu'il est un candidat de réductibilité. Si on nomme ECR l'ensemble des candidats de réductibilité, sa formulation, dans le cas de l'abstraction, est la suivante :

$$\lambda x.t \text{ typable} \quad \Rightarrow \quad \lambda x.t \in \text{ECR}$$

La définition des candidats nous dit :

$$\lambda x.t \in \text{ECR} \quad \Longleftrightarrow \quad \forall u \in \text{ECR}\ (\lambda x.t)\ u \in \mathcal{SN}$$

Pour montrer que $\lambda x.t$ est un bon candidat, il faut le confronter à tous les candidats possibles et vérifier que l'application des deux est un terme fortement normalisant. Pour prouver l'adéquation, on prend alors un candidat u quelconque, on construit l'application $(\lambda x.t)\ u$ et on regarde si ce terme est fortement normalisant : on le réduit vers $t\{x \leftarrow u\}$ et on utilise l'hypothèse d'induction qui nous dit que t est un candidat. Comme on a prouvé auparavant que les candidats sont fortement normalisants, on peut conclure positivement pour le lemme d'adéquation.

Le λ-calcul symétrique [6] est un exemple de calcul symétrique. Sans entrer dans les détails, on peut dire que la principale différence avec le λ-calcul est que l'application est symétrique, c'est-à-dire qu'il n'y a plus de gauche et de droite. Le terme $(\lambda x.t)\ (\lambda y.u)$, par exemple, peut se réduire de deux façons, suivant le choix qu'on fait pour la fonction et son argument : si on prend $(\lambda x.t)$ comme fonction, le terme se réduit en $t\{x \leftarrow (\lambda y.u)\}$, dans l'autre cas, on obtient $u\{y \leftarrow (\lambda x.t)\}$. Comme nous l'avons dit, un tel calcul n'est pas confluent, mais cette paire critique soulève aussi des problèmes dans la preuve de normalisation. Supposons qu'on ait naïvement défini les candidats en disant :

$$\lambda x.t \in \text{ECR} \quad \Longleftrightarrow \quad \forall u \in \text{ECR}\ (\lambda x.t)\ u \in \mathcal{SN} \text{ et } u\ (\lambda x.t) \in \mathcal{SN}$$

Alors quand on voudra prouver le lemme d'adéquation :

$$\lambda x.t \text{ typable} \quad \Rightarrow \quad \lambda x.t \in \text{ECR}$$

On sera obligé de prendre un candidat quelconque, en particulier une abstraction $\lambda y.u$ et de regarder si le terme $(\lambda x.t)\ (\lambda y.u)$ est fortement normalisant. Dans le λ-calcul, on avait une seule réduction possible de ce terme, ce qui nous permettait de conclure facilement. Mais là, on peut réduire $(\lambda x.t)\ (\lambda y.u)$ en $u\{y \leftarrow (\lambda x.t)\}$, et on ne peut plus rien dire de ce terme. En effet, on savait que $t \in ECR$ mais, puisqu'on est en train de prouver que $\lambda x.t \in ECR$, on ne peut pas utiliser ce fait, et la définition de $\lambda y.u \in ECR$ en aurait besoin pour conclure.

C'est pour cette raison que, dans [6], la définition des candidats de réductibilité est modifiée et s'exprime plutôt de la façon suivante :

$$\lambda x.t \in \text{ECR} \quad \Longleftrightarrow \quad \forall u \in \text{ECR}\ t\{x \leftarrow u\} \in \mathcal{SN}$$

Avant de tester la normalisation forte, on commence par effectuer une étape de β-réduction, de façon à interdire la réduction qui posait problème ci-dessus. Grâce à cette astuce, nous allons pouvoir établir,

dans le chapitre 11, la normalisation forte du $\overline{\lambda}\mu\tilde{\mu}$-calcul typé. Ensuite, on introduira des substitutions explicites dans ce calcul.

L'ajout de substitutions explicites pose déjà des problèmes dans les calculs non symétriques. Il semble que dans un calcul symétrique, cela rende très difficile l'utilisation de la technique de preuve par réductibilité. Dans [45] est proposée une preuve de normalisation forte d'un fragment du $\overline{\lambda}\mu\tilde{\mu}$-calcul auquel sont ajoutées des substitutions explicites. Pour pouvoir appliquer la technique de réductibilité, la notion de candidat est étendue pour englober, d'une certaine façon, l'environnement de typage des termes. Dans les preuves, il arrive un moment où l'on est confronté à une substitution qui traverse un lieur. On a vu dans le chapitre 3 que l'environnement de typage de la substitution doit alors être enrichi de la variable liée. Mais comme la notion de candidat contient cet environnement, cela signifie que le terme doit changer de candidat.

La solution apportée dans [45] à ce problème échoue à cause de la gestion des variable liées et libres qui composent les termes. Pour prouver la normalisation forte du $\overline{\lambda}\mu\tilde{\mu}\mathtt{x}$-calcul, nous prendrons une autre approche : nous commencerons par utiliser la technique de perpétuité formalisée dans [13] pour prouver que ce calcul possède la propriété PSN, puis nous appliquerons la technique "PSN implique SN" présentée dans la deuxième partie de cette thèse pour obtenir le résultat.

On se base initialement sur le travail de [6]. On va procéder en trois étapes, en partant du fragment de $\overline{\lambda}\mu\tilde{\mu}$ qui correspond à peu près au λ-calcul symétrique, puis en ajoutant successivement les constructions propres à $\overline{\lambda}\mu\tilde{\mu}$ et enfin les substitutions explicites. De ce fait, la première étape (chapitre 10) contient une preuve similaire à celle de [6] avec les notations de [45]. La deuxième étape (chapitre 11) présente la preuve de normalisation forte du $\overline{\lambda}\mu\tilde{\mu}$-calcul (sans substitutions explicites), essentielle à la troisième étape (chapitre 12) qui contient la preuve de normalisation forte du $\overline{\lambda}\mu\tilde{\mu}\mathtt{x}$-calcul. On commence par présenter le $\overline{\lambda}\mu\tilde{\mu}$-calcul dans le chapitre 9.

Ce travail à été publié dans [64].

Chapitre 9

Présentation du $\overline{\lambda}\mu\tilde{\mu}$-calcul

Ce chapitre donne une courte présentation du $\overline{\lambda}\mu\tilde{\mu}$-calcul. On commence par donner quelques motivations à la définition de ce calcul, puis on présente les termes et les règles de typage. Enfin on explique comment la symétrie a été complétée avec l'ajout d'un opérateur logique et de deux formes de termes supplémentaires. Cette présentation reprend l'introduction de [23].

9.1 Motivations

L'étude des langages de programmation a conduit à l'observation qu'il existe des symétries, comme par exemple les entrées/sorties ou encore celle, programme/contexte, entre un programme et son contexte d'exécution. Il est aussi apparu récemment qu'une symétrie existe entre les deux principales stratégies d'évaluation du λ-calcul : l'*appel par nom* et l'*appel par valeur*.

9.1.1 Appel par nom, appel par valeur

Dans ces deux notions on retrouve le mot *appel* car ces deux stratégies parlent de la façon dont on traite l'appel d'une fonction pour un argument, c'est-à-dire l'application. En effet, regardons un terme quelconque de la forme $t\ u$, il peut être β-réduit de plusieurs manières :

- on peut réduire un redex de t (s'il en existe un) : $t \to t'$, ce qui nous donne la réduction : $t\ u \to t'\ u$,
- on peut réduire un redex de u (s'il en existe un) : $u \to u'$, ce qui nous donne la réduction : $t\ u \to t\ u'$,
- et si $t = \lambda x.v$, alors on peut aussi réduire le β-redex : $(\lambda x.v)\ u \to v\{x \leftarrow u\}$.

Lorsque nous avons présenté le λ-calcul (chapitre 2), nous n'avons rien dit sur les différentes stratégies d'évaluation qui choisissent, parmi ces trois possibilités ci-dessus, dans quel ordre effectuer les β-réductions.

Pour le terme $t\ u$, la stratégie appelée *appel par nom* consiste à évaluer d'abord le terme t jusqu'à obtenir un λ, c'est-à-dire jusqu'à obtenir un terme $t' = \lambda x.v$. On s'interdit de réduire le terme u tant qu'on a pas obtenu ce terme $\lambda x.v$. Ce dernier une fois obtenu, on réduit le terme $(\lambda x.v)\ u$ vers $v\{x \leftarrow u\}$ et on poursuit l'évaluation de notre λ-terme. Si la variable x n'apparaît pas dans v, alors le terme u sera effacé. L'intérêt de cette stratégie est donc qu'elle retarde l'évaluation de u jusqu'à être sûr que ce terme sera effectivement utilisé dans la fonction.

La stratégie appelée *appel par valeur* consiste à ne réduire un redex $(\lambda x.v)\ u$ que si u est une valeur. En première intuition, on peut considérer qu'une valeur est un terme qui ne peut pas être réduit davantage, c'est-à-dire un terme en forme normale. Ainsi, si u est une valeur, on peut réduire $(\lambda x.v)\ u$ vers $v\{x \leftarrow u\}$ et on poursuit l'évaluation de notre λ-terme. L'avantage de cette stratégie est qu'elle factorise les réductions de u : si la variable x apparaît n fois dans le terme v, la stratégie précédente nécessitera n fois l'évaluation du terme u, contre une fois seulement pour cette stratégie.

Évaluer d'abord la fonction ou d'abord l'argument semble symétrique, le $\overline{\lambda}\mu\tilde{\mu}$-calcul rendra compte de cette symétrie à l'intérieur même de ses termes et de ses règles de réduction.

9.1.2 Calcul des séquents *vs.* déduction naturelle

Comme nous allons le voir, les règles de typage du $\overline{\lambda}\mu\tilde{\mu}$-calcul sont données dans le formalisme du calcul des séquents. En effet, bien que ce formalisme soit moins naturel que la déduction naturelle, il possède d'autres avantages : il vérifie la propriété de la sous-formule, ce qui est un atout précieux dans la recherche automatique de preuves, et la coupure y est bien mieux caractérisée que dans la déduction naturelle (voir section 3.1.2). En outre, la symétrie des règles d'introduction à gauche et à droite en fait une bonne base de départ pour la conception d'un calcul exhibant la symétrie appel-par-nom/appel-par-valeur.

Par rapport au calcul des séquent présenté dans le chapitre 3, une différence notable sera la présence de *plusieurs* formules à droite des séquents. Sans entrer dans les détails, cela corresond au fait qu'on travaille ici dans la logique classique, et non plus seulement dans le fragment intuitionniste.

Pour donner les règles de typage du calcul, nous aurons besoin d'isoler dans un séquent la *formule active*. Pour cela, nous noterons parfois les séquents de la façon suivante :

$$\overline{\Gamma \vdash A|\Delta} \qquad \text{ou encore} \qquad \overline{\Gamma|A \vdash \Delta}$$

D'un point de vue logique, le symbole | ne sert à rien, il est juste une autre notation de la virgule. Il nous permettra d'identifer une formule sur laquelle la règle va opérer.

9.2 Définitions

Avec ces intuitions de symétrie, nous allons pouvoir regarder la définition des termes et des règles de réduction et de typage du $\overline{\lambda}\mu\tilde{\mu}$-calcul.

Pour pouvoir retranscrire l'intuition que l'on a donnée de la symétrie programme/contexte, il nous faut tout d'abord séparer ces deux objets. Au lieu de réaliser l'*application* d'une fonction à son argument, on va mettre cette fonction dans un contexte qui contient son argument. Cette notion est moins naturelle que l'application, mais elle est plus proche de ce qui se passe dans l'exécution des programmes. Nous aurons donc trois catégoriés d'objets : les termes, les contextes, et les commandes qui seront constituées d'un couple (terme,contexte). Les termes correspondent aux programmes, les contextes à leur environnement, et les commandes à la rencontre d'un programme et d'un contexte. La symétrie terme/contexte apparaîtra dans le typage : les termes seront typés à droite du séquent tandis que les contextes seront typés à gauche, les commandes seront alors naturellement typées par la règle de coupure.

objets et réductions, première version

Commençons par regarder les termes. En partant du λ-calcul, on peut tout de suite donner deux termes : les variables x, et les fonctions $\lambda x.v$, où v est lui-aussi un terme. Nous n'avons plus l'application puisqu'elle sera représentée par une commande. Regardons tout de suite les règles de typage de ces termes. Mis à part le symbole |, elle nous sont familières :

$$\overline{\Gamma, x : A \vdash x : A|\Delta} \qquad \frac{\Gamma, x : A \vdash v : B|\Delta}{\Gamma \vdash \lambda x.v : A \to B|\Delta}$$

Si nous voulons maintenant construire des contextes, nous pouvons nous aider de la logique. La commande composée d'un terme de la forme $\lambda x.v$ et d'un contexte e quelconque sera typé par une coupure. Le terme $\lambda x.v$ correspondant à l'introduction de la flèche à droite, on doit donc avoir une forme de contexte qui corresponde à l'introduction de la flèche à gauche. Voici la règle de typage,

dans laquelle les ? représentent les "trous" qu'il faut combler par des objets (termes, contextes ou commandes) :

$$\frac{\Gamma \vdash ? : A | \Delta \quad \Gamma | ? : B \vdash \Delta}{\Gamma | ? : A \Rightarrow B \vdash \Delta}$$

Pour combler les trous, il suffit de regarder les deux séquents en hypothèse. Le premier, $\Gamma \vdash ? : A | \Delta$ correspond au typage d'un terme, notons le v, le deuxième, $\Gamma | ? : B \vdash \Delta$ correspond au typage d'un contexte, notons le e. On est donc en présence de deux objets, un peu comme dans la règle qui type l'application dans le λ-calcul. Le terme dans le séquent conclusion doit être une sorte de composition entre v et e. Regardons plus loin. Si on recommence à typer le contexte e avec la même règle, cela va nous donner un autre terme v', et un autre contexte e' qui pourrait lui-aussi être composé d'un terme v'', etc... Le typage d'un contexte avec cette règle correspond au typage d'une suite de termes, cela correspond à la notion d'environnement d'un programme, c'est-à-dire à une pile. On note l'empilement d'un terme v sur un contexte e avec la syntaxe suivante : $v \cdot e$. La règle de typage devient :

$$\frac{\Gamma \vdash v : A | \Delta \quad \Gamma | e : B \vdash \Delta}{\Gamma | v \cdot e : A \Rightarrow B \vdash \Delta}$$

Il faut pouvoir terminer le typage d'un contexte, c'est-à-dire qu'il nous faut un contexte dont le typage ne fasse pas appel au typage d'un contexte plus petit. Autrement dit, il nous faut une fin de pile. Le plus simple consiste à donner un contexte dont le typage utilise la règle axiome, ce qui nous garantit que le typage s'arrêtera là. Par symétrie, on est satisfait de cette possibilité qui correspondra au typage d'une variable de contexte. Les variables de termes étant notées x, y, z, etc. nous noterons les variables de contexte α, β, etc. Voici la règle de typage :

$$\frac{}{\Gamma | \alpha : A \vdash \alpha : A, \Delta}$$

Elle est identique à celle du typage des variables de termes, sauf pour la position du symbole $|$, qui se trouve ici à gauche puisqu'il s'agit d'un contexte.

Comme nous l'avons dit, une commande est composée d'un terme v et d'un contexte e, nous noterons cela $\langle v | e \rangle$. Le typage d'une commande correspond à la règle de coupure du calcul des séquents :

$$\frac{\Gamma \vdash v : A | \Delta \quad \Gamma | e : A \vdash \Delta}{\langle v | e \rangle : (\Gamma \vdash \Delta)}$$

Nous avons placé le type de $\langle v | e \rangle$ entre parenthèse et à part de la commande elle-même car une commande n'est ni un terme, ni un contexte, et ne peut donc apparaître ni à droite, ni à gauche du séquent.

Préoccupons nous un peu de l'évaluation de nos objets. Prenons la commande $\langle \lambda x.v | v' \cdot e \rangle$, elle correspond à l'application de la fonction $\lambda x.v$ au paramètre v'. Nous pourrions être tentés de la réduire vers $\langle v[v'/x] | e \rangle$ pour mimer la β-réduction, mais nous souhaitons ici avoir un contrôle plus fin sur l'exécution dans le but de choisir l'appel par nom ou l'appel par valeur. Notre commande va se réduire vers une autre commande qui laissera encore le choix de la stratégie de réduction. Cette commande dira que l'évaluation peut se poursuivre dans le paramètre v', avec comme contexte le souvenir que le résultat de v' devra remplacer x dans le terme v. Il faut introduire un nouveau contexte correspondant à cette mémorisation, il sera noté $\tilde{\mu}x.c$ où c est une commande, en l'occurence la commande $\langle v | e \rangle$ puisque la mémorisation doit aussi prendre en compte le contexte e.

Voici une première version de notre système. Les termes sont donnés par la grammaire suivante :

$$\begin{array}{lcl} v & ::= & x \mid \lambda x.v \\ e & ::= & \alpha \mid v \cdot e \mid \tilde{\mu}x.c \\ c & ::= & \langle v | e \rangle \end{array}$$

Et nous pouvons donner deux règles de réduction :

$$\begin{array}{lllll} (\beta) & \langle \lambda x.v | v' \cdot e \rangle & \rightarrow & \langle v' | \tilde{\mu}x.\langle v | e \rangle \rangle & \text{si } x \notin FV(e) \\ (\tilde{\mu}) & \langle v | \tilde{\mu}x.c \rangle & \rightarrow & c[v/x] & \end{array}$$

Compléter la symétrie

On peut faire deux observations sur le système ci-dessus. Premièrement, il ne contient pas l'opérateur μ de [63], dont nous avons pourtant dit que ce calcul s'inspirait. Deuxièmement, il y a une dissymétrie entre les termes et les contextes, alors qu'on souhaitait une symétrie. La symétrie entre termes et contextes va être obtenue en ajoutant l'opérateur μ, symétrique du $\tilde{\mu}$, et en ajoutant deux autres opérateurs que nous allons présenter maintenant.

Dans notre présentation ci-dessus, les termes ont un lieur $\lambda x.v$ tandis que les contextes ont un constructeur de piles de termes $v \cdot e$. Il semble naturel de vouloir compléter notre système en donnant un lieur pour les contextes, par exemple $\alpha\lambda.e$, et un constructeur de piles de contextes, que l'on notera $e \cdot v$ (le symbole $\cdot$ est surchargé par cette nouvelle définition, mais le type des lettres – v ou e – nous permettra toujours de lever l'ambiguïté). Du point de vue logique, il faut ajouter un nouveau connecteur pour pouvoir typer ces nouveaux objets : ce sera le connecteur "différence", noté $-$, et il sera le dual de l'implication. Voici la définition des termes du $\overline{\lambda}\mu\tilde{\mu}$-calcul :

$$\begin{array}{lcl} v & ::= & x \mid \lambda x.v \mid e \cdot v \mid \mu\alpha.c \\ e & ::= & \alpha \mid \alpha\lambda.e \mid v \cdot e \mid \tilde{\mu}x.c \\ c & ::= & \langle v|e\rangle \end{array}$$

Voici les règles de typage des deux nouvelles constructions syntaxiques :

$$\frac{\Gamma|e : B \vdash \alpha : A, \Delta}{\Gamma|\alpha\lambda.e : B - A \vdash \Delta} \qquad \frac{\Gamma \vdash v : B|\Delta \quad \Gamma|e : A \vdash \Delta}{\Gamma \vdash e \cdot v : B - A|\Delta}$$

Nous ajoutons une règle pour le λ des contextes et une pour le μ, en effectuant la symétrie avec (β) et $(\tilde{\mu})$, ce qui nous donne les quatre règles suivantes :

$$\begin{array}{llcll} (\beta) & \langle \lambda x.v | v' \cdot e\rangle & \to & \langle v' | \tilde{\mu}x.\langle v|e\rangle\rangle & \text{si } x \notin FV(e) \\ (\tilde{\beta}) & \langle e'' \cdot v | \alpha\lambda.e\rangle & \to & \langle \mu\alpha.\langle v|e\rangle | e'\rangle & \text{si } \alpha \notin FV(v) \\ (\mu) & \langle \mu\alpha.c | e\rangle & \to & c[e/\alpha] & \\ (\tilde{\mu}) & \langle v | \tilde{\mu}x.c\rangle & \to & c[v/x] & \end{array}$$

On s'aperçoit que les règles (μ) et $(\tilde{\mu})$ forment une paire critique : on peut réduire le terme $\langle \mu\alpha.c | \tilde{\mu}x.c'\rangle$ de deux façons différentes.

$$\begin{array}{ccc} & \langle \mu\alpha.c | \tilde{\mu}x.c'\rangle & \\ (\mu) \swarrow & & \searrow (\tilde{\mu}) \\ c[\tilde{\mu}x.c'/\alpha] & & c'[\mu\alpha.c/x] \end{array}$$

C'est ici que se trouve le choix entre appel par nom et appel par valeur. Si on choisit de toujours réduire le μ-redex en priorité, on suit la stratégie d'appel par valeur, tandis que si on choisit de toujours réduire le $\tilde{\mu}$-redex en priorité, on suit la stratégie d'appel par nom. Cette paire critique ne peut pas être jointe, cela rend le système non-confluent et c'est cela aussi qui soulève des difficultés dans les preuves de normalisation forte.

Exemple 9.2.1

- Pour illustrer le choix entre appel par nom et appel par valeur, regardons un exemple de réduction. Le terme $(\lambda x.x\ x)\ ((\lambda x.x)\ x)$ se réduit, dans le λ-calcul, de deux manières différentes

suivant la stratégie que l'on emploie :

$$\begin{array}{ccc} & (\lambda x.x\ x)\ ((\lambda x.x)\ x) & \\ \text{appel par valeur} \swarrow & & \searrow \text{appel par nom} \\ (\lambda x.x\ x)\ x & & ((\lambda x.x)\ x)\ ((\lambda x.x)\ x) \\ \downarrow & & \downarrow \\ x\ x & & x\ ((\lambda x.x)\ x) \\ & & \downarrow \\ & & x\ x \end{array}$$

La traduction de $(\lambda x.x\ x)\ ((\lambda x.x)\ x)$ dans le $\overline{\lambda}\mu\tilde{\mu}$-calcul donne le terme

$$\mu\alpha.\langle\mu\beta.\langle x|\tilde{\mu}y.\langle\lambda x.x|y\cdot\beta\rangle\rangle|\tilde{\mu}z.\langle\lambda x.\mu\gamma.\langle x|\tilde{\mu}z'.\langle x|z'\cdot\gamma\rangle\rangle|z\cdot\alpha\rangle\rangle$$

En isolant les deux sous-termes principaux, on constate qu'ils forment la paire critique qui nous laisse le choix entre appel par nom (règle $\tilde{\mu}$) et appel par valeur (règle μ) :

$$\mu\alpha.\langle\quad\mu\beta.\langle x|\tilde{\mu}y.\langle\lambda x.x|y\cdot\beta\rangle\rangle\quad|\quad\tilde{\mu}z.\langle\lambda x.\mu\gamma.\langle x|\tilde{\mu}z'.\langle x|z'\cdot\gamma\rangle\rangle|z\cdot\alpha\rangle\quad\rangle$$

Pour mieux voir les réductions suivants ces deux stratégies, on commence par réduire les deux sous-termes :

$$\begin{array}{c} \mu\alpha.\langle\mu\beta.\langle x|\tilde{\mu}y.\langle\lambda x.x|y\cdot\beta\rangle\rangle|\tilde{\mu}z.\langle\lambda x.\mu\gamma.\langle x|\tilde{\mu}z'.\langle x|z'\cdot\gamma\rangle\rangle|z\cdot\alpha\rangle\rangle \\ \downarrow\tilde{\mu} \\ \mu\alpha.\langle\mu\beta.\langle\lambda x.x|x\cdot\beta\rangle|\tilde{\mu}z.\langle\lambda x.\mu\gamma.\langle x|\tilde{\mu}z'.\langle x|z'\cdot\gamma\rangle\rangle|z\cdot\alpha\rangle\rangle \\ \downarrow\tilde{\mu} \\ \mu\alpha.\langle\mu\beta.\langle\lambda x.x|x\cdot\beta\rangle|\tilde{\mu}z.\langle\lambda x.\mu\gamma.\langle x|x\cdot\gamma\rangle|z\cdot\alpha\rangle\rangle \\ \downarrow\beta \\ \mu\alpha.\langle\mu\beta.\langle\lambda x.x|x\cdot\beta\rangle|\tilde{\mu}z.\langle z|\tilde{\mu}x.\langle\mu\gamma.\langle x|x\cdot\gamma\rangle|\alpha\rangle\rangle\rangle \\ \downarrow\mu \\ \mu\alpha.\langle\mu\beta.\langle\lambda x.x|x\cdot\beta\rangle|\tilde{\mu}z.\langle z|\tilde{\mu}x.\langle x|x\cdot\alpha\rangle\rangle\rangle \\ \downarrow\tilde{\mu} \\ \mu\alpha.\langle\mu\beta.\langle\lambda x.x|x\cdot\beta\rangle|\tilde{\mu}z.\langle z|z\cdot\alpha\rangle\rangle \end{array}$$

Voici enfin les deux réductions de ce dernier terme :

$$\begin{array}{ccc} & \mu\alpha.\langle\ \mu\beta.\langle\lambda x.x|x\cdot\beta\rangle\ |\ \tilde{\mu}z.\langle z|z\cdot\alpha\rangle\ \rangle & \\ \text{appel par valeur } (\mu) \swarrow & & \searrow \text{appel par nom } (\tilde{\mu}) \\ \mu\alpha.\langle\lambda x.x|x\cdot\tilde{\mu}z.\langle z|z\cdot\alpha\rangle\rangle & & \mu\alpha.\langle\mu\beta.\langle\lambda x.x|x\cdot\beta\rangle|\tilde{\mu}z.\langle z|z\cdot\alpha\rangle\rangle \\ \beta\downarrow & & \downarrow\mu \\ \mu\alpha.\langle x|\tilde{\mu}x.\langle x|\tilde{\mu}z.\langle z|z\cdot\alpha\rangle\rangle\rangle & & \mu\alpha.\langle\lambda x.x|x\cdot\mu\beta.\langle\lambda x.x|x\cdot\beta\rangle\cdot\alpha\rangle \\ \tilde{\mu}\downarrow & & \downarrow\beta \\ \mu\alpha.\langle x|\tilde{\mu}z.\langle z|z\cdot\alpha\rangle\rangle & & \mu\alpha.\langle x|\tilde{\mu}x.\langle x|\mu\beta.\langle\lambda x.x|x\cdot\beta\rangle\cdot\alpha\rangle\rangle \\ \tilde{\mu}\downarrow & & \downarrow\tilde{\mu} \\ \mu\alpha.\langle x|x\cdot\alpha\rangle & & \mu\alpha.\langle x|\mu\beta.\langle\lambda x.x|x\cdot\beta\rangle\cdot\alpha\rangle \\ & & \downarrow\beta \\ & & \mu\alpha.\langle x|\mu\beta.\langle x|\tilde{\mu}x.\langle x|\beta\rangle\rangle\cdot\alpha\rangle \\ & & \downarrow\tilde{\mu} \\ & & \mu\alpha.\langle x|\mu\beta.\langle x|\beta\rangle\cdot\alpha\rangle \\ & & \downarrow sv\ ^1 \\ & & \mu\alpha.\langle x|x\cdot\alpha\rangle \end{array}$$

- Un exemple sans doute encore plus explicite met en jeu le terme Ω qui, rappelons-le, n'a

pas de forme normale, c'est-à-dire qu'il peut être réduit indéfiniment (voir chapitre 2). Dans le λ-calcul, le terme $(\lambda x.y)\ \Omega$ se réduit, en appel par nom, vers y, alors que sa réduction ne s'arrête jamais en appel par valeur :

$$\begin{array}{ccc} & (\lambda x.y)\ \Omega & \\ \text{appel par valeur} \swarrow & & \searrow \text{appel par nom} \\ (\lambda x.y)\ \Omega & & y\{x \leftarrow \Omega\} \\ \downarrow & & = \\ (\lambda x.y)\ \Omega & & y \\ \downarrow & & \\ \vdots & & \end{array}$$

Supposons que la traduction de Ω dans le $\overline{\lambda}\mu\tilde{\mu}$-calcul soit $\mu\beta.\overline{\Omega}$, alors la traduction de $(\lambda x.y)\ \Omega$ est $\mu\alpha.\langle\mu\beta.\overline{\Omega}|\tilde{\mu}z.\langle\lambda x.y|z\cdot\alpha\rangle\rangle$. La non-terminaison de Ω est préservée par la traduction, c'est-à-dire que pour tout contexte e, $\overline{\Omega}\{\beta \leftarrow e\}$ admet une dérivation infinie en stratégie d'appel par valeur. On a les réductions suivantes :

$$\begin{array}{ccc} & \mu\alpha.\langle\mu\beta.\overline{\Omega}|\tilde{\mu}z.\langle\lambda x.y|z\cdot\alpha\rangle\rangle & \\ & \downarrow \beta & \\ & \downarrow se\ ^1 & \\ & \mu\alpha.\langle\mu\beta.\overline{\Omega}|\tilde{\mu}x.\langle y|\alpha\rangle\rangle & \\ \text{appel par valeur } (\mu) \swarrow & & \searrow \text{appel par nom } (\tilde{\mu}) \\ \mu\alpha.\overline{\Omega}\{\beta \leftarrow \tilde{\mu}x.\langle y|\alpha\rangle\} & & \mu\alpha.\langle y|\alpha\rangle\{x \leftarrow \mu\beta.\overline{\Omega}\} \\ \downarrow & & = \\ \mu\alpha.\overline{\Omega}\{\beta \leftarrow \tilde{\mu}x.\langle y|\alpha\rangle\} & & \mu\alpha.\langle y|\alpha\rangle \\ \downarrow & & \downarrow sv\ ^1 \\ \vdots & & y \end{array}$$

[1] Le terme $\mu\alpha.\langle v|\alpha\rangle$ dans lequel α n'est pas libre dans v, et le terme v sont sémantiquement équivalents. La règle se, et son symétrique sv, sont des règles de simplification (voir leur définition dans chapitre suivant).

Chapitre 10

Normalisation forte du $\mu\tilde{\mu}$-calcul

Cette section suit fidèlement le travail de Barbanera et Berardi [6]. On se contente d'adapter la preuve en distinguant les parties droite et gauche de la coupure et en prenant les notations du $\overline{\lambda}\mu\tilde{\mu}$-calcul.

10.1 Définition du calcul

Il y a trois catégories syntaxiques : les termes, les contextes et les commandes ; notées respectivement v, e, c. On donne deux ensembles de variables : Var est l'ensemble des variables de termes, ce sont des valeurs notées x, y, z etc. ; $Var^{\perp}$ est l'ensemble des variables de contextes, ce sont des contextes notés α, β, γ etc. La syntaxe du $\mu\tilde{\mu}$-calcul est :

$$\begin{array}{rcl} c & ::= & \langle v|e\rangle \\ v & ::= & x \mid \mu\alpha.c \\ e & ::= & \alpha \mid \tilde{\mu}x.c \end{array}$$

Les règles de réduction sont données ci-dessous. Les deux règles μ et $\tilde{\mu}$ forment une paire critique non joignable, ce qui rend le calcul non-déterministe :

$$\begin{array}{llcll} (\mu) & \langle \mu\alpha.c|e\rangle & \to & c[e/\alpha] & \\ (\tilde{\mu}) & \langle v|\tilde{\mu}x.c\rangle & \to & c[v/x] & \\ & & & & \\ (sv) & \mu\alpha.\langle v|\alpha\rangle & \to & v & \text{si } \alpha \notin FV(v) \\ (se) & \tilde{\mu}x.\langle x|e\rangle & \to & e & \text{si } x \notin FV(e) \end{array}$$

Notation 10.1.1
Lorsque cela sera nécessaire, on notera ce système de réduction $\to_{\mu\tilde{\mu}}$.

Trois formes de séquents typent les catégories syntaxiques : les commandes sont typées par $(\Gamma \vdash \Delta)$, les termes par $\Gamma \vdash A|\Delta$, et les contextes par $\Gamma|A \vdash \Delta$. Voici les règles de typage :

$$\frac{\Gamma \vdash v : A|\Delta \quad \Gamma|e : A \vdash \Delta}{\langle v|e\rangle : (\Gamma \vdash \Delta)}$$

$$\frac{}{\Gamma|\alpha : A \vdash \Delta, \alpha : A} \qquad \frac{}{\Gamma, x : A \vdash \Delta|x : A}$$

$$\frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma|\tilde{\mu}x.c : A \vdash \Delta} \qquad \frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A|\Delta}$$

Remarque 10.1.2

On note qu'il n'y a pas de structure de types. Le calcul est en fait un simple calcul de manipulation de la coupure.

10.2 Définition des ECR

Définition 10.2.1

On définit simultanément :

– les opérateurs :

$$Mu(X) \quad =_{Def} \quad \{\mu\alpha.c \mid \forall e \in X \;\; c[e/\alpha] \in [\![\vdash]\!]\}$$

$$\widetilde{Mu}(X) \quad =_{Def} \quad \{\tilde{\mu}x.c \mid \forall v \in X \;\; c[v/x] \in [\![\vdash]\!]\}$$

Puis,

$$Neg_{[\![\vdash \; -]\!]}(X) = Var \cup Mu(X)$$

$$Neg_{[\![- \; \vdash]\!]}(X) = Var^{\perp} \cup \widetilde{Mu}(X)$$

On a besoin, pour chaque type, d'inclure *toutes* les variables dans les ensembles de réductibilité, afin d'être sûr de toujours pouvoir trouver une variable libre pour réaliser la preuve du lemme 10.3.1. Les deux Neg sont des opérateurs décroissants, donc $Neg_{[\![\vdash \; -]\!]} \circ Neg_{[\![- \; \vdash]\!]}$ est un opérateur croissant. D'après Tarski, il existe alors un point fixe X_0.

Remarque 10.2.2

On peut prendre le plus petit point fixe si on le souhaite, mais, puisqu'on ne se sert de ce point fixe que pour prouver la propriété 10.2.4, n'importe quel point fixe fait l'affaire.

– Les ECR :

$$[\![\vdash]\!] = \mathcal{SN}_{\mu\tilde{\mu}}$$

Puis

$$[\![\vdash \; -]\!] = X_0 \quad \text{et} \quad [\![- \; \vdash]\!] = Neg_{[\![- \; \vdash]\!]}(X_0)$$

Remarque 10.2.3

On sait que $Neg_{[\![- \; \vdash]\!]}(X_0)$ est un point fixe car $Neg_{[\![- \; \vdash]\!]} \circ Neg_{[\![\vdash \; -]\!]}$ est aussi un opérateur croissant et car, puisque X_0 est un point fixe, on a :

$$\begin{array}{c} Neg_{[\![\vdash \; -]\!]} \circ Neg_{[\![- \; \vdash]\!]}(X_0) = X_0 \\ \Downarrow \\ Neg_{[\![- \; \vdash]\!]}((Neg_{[\![\vdash \; -]\!]} \circ Neg_{[\![- \; \vdash]\!]})(X_0)) = Neg_{[\![- \; \vdash]\!]}(X_0) \\ \Longleftrightarrow \\ (Neg_{[\![- \; \vdash]\!]} \circ Neg_{[\![\vdash \; -]\!]})(Neg_{[\![- \; \vdash]\!]}(X_0)) = Neg_{[\![- \; \vdash]\!]}(X_0) \end{array}$$

Propriété 10.2.4 (Bonne définition)

Les ECR définis ci-dessus vérifient

(i) $Var \subset [\![\vdash \; -]\!]$

(ii) $Var^{\perp} \subset [\![- \; \vdash]\!]$

(iii) $\mu\alpha.c \in [\![\vdash \; -]\!] \iff \forall e \in [\![- \; \vdash]\!] \;\; c[e/\alpha] \in [\![\vdash]\!]$

(iv) $\tilde{\mu}x.c \in [\![- \; \vdash]\!] \iff \forall v \in [\![\vdash \; -]\!] \;\; c[v/x] \in [\![\vdash]\!]$

Preuve : Par la définition 10.2.1 on a $[\![\vdash]\!] = \mathcal{SN}_{\mu\tilde{\mu}}$ et les points (i) et (ii). On prouve à présent

(iii) et (iv). En fait, on se contente de prouver (iii), car (iv) est symétriquement identique.

$$\begin{array}{ccc} \mu\alpha.c \in [\![\vdash\ -]\!] & \overset{?}{\iff} & \forall e \in [\![-\ \vdash]\!]\ c[e/\alpha] \in [\![\vdash]\!] \\ \iff & & \iff \\ \mu\alpha.c \in Neg_{[\![\vdash\ -]\!]} \circ Neg_{[\![-\ \vdash]\!]}([\![\vdash\ -]\!]) & & \mu\alpha.c \in Mu([\![-\ \vdash]\!]) \\ \iff & & \iff \\ \mu\alpha.c \in Var \cup Mu(Neg_{[\![-\ \vdash]\!]}([\![\vdash\ -]\!])) & \iff & \mu\alpha.c \in Var \cup Mu([\![-\ \vdash]\!]) \end{array}$$

■

10.3 Propriétés des ECR

Voici les lemmes habituels des preuves par réductibilité : normalisation forte des ECR puis fermeture par réduction.

Lemme 10.3.1 (Normalisation forte des ECR)

On a

1. $[\![\vdash\ -]\!] \subset \mathcal{SN}_{\mu\tilde{\mu}}$
2. $[\![-\ \vdash]\!] \subset \mathcal{SN}_{\mu\tilde{\mu}}$
3. $[\![\vdash]\!] \subset \mathcal{SN}_{\mu\tilde{\mu}}$

Preuve :

1. On regarde les différentes formes de $v \in [\![\vdash\ -]\!]$:
   - $v = x$: $x \in \mathcal{SN}_{\mu\tilde{\mu}}$.
   - $v = \mu\alpha.c$: par le point (ii) de la proposition 10.2.4, $\alpha \in [\![-\ \vdash]\!]$, et, par le point (iii) de la proposition 10.2.4, $c[\alpha/\alpha] = c \in [\![\vdash]\!] (= \mathcal{SN}_{\mu\tilde{\mu}})$. Ce qui nous donne $\mu\alpha.c \in \mathcal{SN}_{\mu\tilde{\mu}}$.
2. Le cas pour e est traité de façon similaire au cas ci-dessus en considérant la symétrie.
3. Par définition $[\![\vdash]\!] = \mathcal{SN}_{\mu\tilde{\mu}}$.

■

Lemme 10.3.2 (Fermeture par réduction)

Pour tout v, e et c, on a

1. $v \in [\![\vdash\ -]\!],\ v \to v' \implies v' \in [\![\vdash\ -]\!]$
2. $e \in [\![-\ \vdash]\!],\ e \to e' \implies e' \in [\![-\ \vdash]\!]$
3. $c \in [\![\vdash]\!],\ c \to c' \implies c' \in [\![\vdash]\!]$

Preuve : En considérant les différentes formes de v, e, et c.

1.1. $v = x$: alors il n'y a pas de réduction possible.

1.2. $v = \mu\alpha.c$: deux cas sont possibles.
   - La réduction est $c \to c'$. On sait par hypothèse que $\mu\alpha.c \in [\![\vdash\ -]\!]$ donc, par le point (iii) de la proposition 10.2.4, $\forall e \in [\![-\ \vdash]\!]\ c[e/\alpha] \in \mathcal{SN}_{\mu\tilde{\mu}}$. On a alors $c'[e/\alpha] \in \mathcal{SN}_{\mu\tilde{\mu}}$ (toujours pour tout $e \in [\![-\ \vdash]\!]$) et on conclut à l'aide du point (iii) de la proposition 10.2.4.
   - La réduction est $\mu\alpha.\langle v|\alpha\rangle \to v$ avec $\alpha \notin FV(v)$. On sait par hypothèse que $\mu\alpha.\langle v|\alpha\rangle \in [\![\vdash\ -]\!]$ donc, par le point (iii) de la proposition 10.2.4, $\forall e \in [\![-\ \vdash]\!]\ \langle v|\alpha\rangle[e/\alpha] \in \mathcal{SN}_{\mu\tilde{\mu}}$, c'est-à-dire $\langle v|e\rangle \in \mathcal{SN}_{\mu\tilde{\mu}}$. Si v est une variable, alors la conclusion est immédiate. Sinon, on a $v = \mu\beta.c$ et $\langle \mu\beta.c|e\rangle \in \mathcal{SN}_{\mu\tilde{\mu}}$ implique $c[e/\beta] \in \mathcal{SN}_{\mu\tilde{\mu}}$, ce qui nous donne $\mu\beta.c \in [\![\vdash\ -]\!]$ par le point (iii) de la proposition 10.2.4.

2.1. $e = \alpha$: alors il n'y a pas de réduction possible.

2.2. $e = \tilde{\mu}x.c$: la preuve est identique à celle du cas $v = \mu\alpha.c$ en prenant la symétrie.

3. $c \in [\![\vdash]\!]$: alors $c \in \mathcal{SN}_{\mu\tilde{\mu}}$ et $c \to c'$ implique que $c' \in \mathcal{SN}_{\mu\tilde{\mu}} = [\![\vdash]\!]$.

■

10.4 Preuve de normalisation forte

Voici un lemme qui nous sera utile pour construire inductivement l'appartenance d'un terme à un ECR.

Lemme 10.4.1

Pour tout v et e, on a

$$v \in [\![\vdash\ -]\!],\ e \in [\![-\ \vdash]\!] \implies \langle v|e\rangle \in [\![\vdash]\!]$$

Preuve : Par définition de $[\![\vdash]\!]$, il faut et il suffit de montrer que $\langle v|e\rangle \in \mathcal{SN}_{\mu\tilde{\mu}}$. On va procéder en regardant les différents couples v, e possibles. Puisque par le lemme 10.3.1, on a $v \in \mathcal{SN}_{\mu\tilde{\mu}}$ et $e \in \mathcal{SN}_{\mu\tilde{\mu}}$, on raisonnera aussi par induction sur la forte normalisation de v et e. Si $\langle v|e\rangle \to \langle v'|e\rangle$ (resp. $\langle v|e\rangle \to \langle v|e'\rangle$) on conclut par hypothèse d'induction sur la forte normalisation de v (resp. e) et par le lemme 10.3.2. Sinon la réduction est soit μ soit $\tilde{\mu}$. Dans les deux cas, on conclut alors par définition de $\mu\alpha.c \in [\![\vdash\ -]\!]$ ou $\tilde{\mu}x.c \in [\![-\ \vdash]\!]$. ■

Le lemme qui suit est le dernier nécessaire, c'est le lemme d'adéquation.

Lemme 10.4.2 (Adéquation)

Soit t un terme du langage avec $FV(t) = X_1 \cup X_2$ ($X_1 \subset Var$ et $X_2 \subset Var^{\perp}$) et les variables $x_i \in X_1$ sont de type B_i et les variables $\alpha_j \in X_2$ sont de type C_j. Pour tout ensemble de termes v_i, e_j tels que $\forall i\ \ v_i \in [\![\vdash\ -]\!]$ et $\forall j\ \ e_j \in [\![-\ \vdash]\!]$, on a selon la forme de t

1. Soit $X_1 : B \vdash v : A | X_2 : C$ alors
$$v[v_1/x_1, ..., v_n/x_n, e_1/\alpha_1, ..., e_m/\alpha_m] \in [\![\vdash\ -]\!]$$
2. Soit $X_1 : B | e : A \vdash X_2 : C$ alors
$$e[v_1/x_1, ..., v_n/x_n, e_1/\alpha_1, ..., e_m/\alpha_m] \in [\![-\ \vdash]\!]$$
3. Soit $c : (X_1 : B \vdash X_2 : C)$ alors
$$c[v_1/x_1, ..., v_n/x_n, e_1/\alpha_1, ..., e_m/\alpha_m] \in [\![\vdash]\!]$$

Remarque 10.4.3

La notation $X_1 : B$ est un raccourci pour une énumération de $\{x_i : B_i | i \in [1, n]\}$ (idem pour $X_2 : C$).

Preuve : On notera $[//]$ la substitution $[v_1/x_1, ..., v_n/x_n, e_1/\alpha_1, ..., e_m/\alpha_m]$. On raisonne par induction sur la structure de t.

– $v = x$: alors, par hypothèse, $\exists i \in [1, n]\ \ A = B_i$. Ainsi
$$v[x_1/v_1, ..., x_n/v_n, \alpha_1/e_1, ..., \alpha_m/e_m] = v_i \in [\![\vdash\ -]\!] = [\![\vdash\ -]\!]$$

– $v = \mu\alpha.c$: puisqu'on peut renommer les variables liées, on peut supposer que $\alpha \notin \{\alpha_1, ..., \alpha_m\}$. Maintenant, par le point (iii) de la proposition 10.2.4, pour prouver que
$$\mu\alpha.c[v_1/x_1, ..., v_n/x_n, e_1/\alpha_1, ..., e_m/\alpha_m] = \mu\alpha.c[//] \in [\![\vdash\ -]\!]$$
il est suffisant de prouver que, pour tout $e \in [\![-\ \vdash]\!]$,
$$c[//, e/\alpha] \in [\![\vdash]\!]$$
ce qui est fait par hypothèse d'induction.

- $e = \alpha$: ce cas est le même que $v = x$ par symétrie
- $e = \widetilde{\mu} x.c$: ce cas est le même que $v = \mu\alpha.c$ par symétrie
- $c = \langle v|e\rangle$. Par hypothèse d'induction sur v et e et par le lemme 10.4.1 le résultat est immédiat.

■

On peut maintenant passer au théorème principal de ce chapitre.

Théorème 10.4.4

Le $\mu\tilde{\mu}$-calcul est fortement normalisant.

Preuve : Soit t un terme du $\mu\tilde{\mu}$-calcul typé par Γ et Δ, c'est-à-dire tel que la conclusion de sa dérivation de typage est soit $\Gamma \vdash t : A|\Delta$, soit $\Gamma|t : A \vdash \Delta$, soit $t : (\Gamma \vdash \Delta)$. Supposons que ses variables libres sont $\{\alpha_1, ..., \alpha_m, x_1, ..., x_n\}$, chacune typée $x_i : A_i$ et $\alpha_i : B_i$. Par les points (i) et (ii) de la proposition 10.2.4, on obtient que pour tout i et j, $x_i \in [\![\vdash\ -]\!]$ et $\alpha_i \in [\![-\ \vdash]\!]$. Alors, par le lemme 10.4.2, $t[x_1/x_1, ..., x_n/x_n, \alpha_1/\alpha_1, ..., \alpha_m/\alpha_m] = t$ est bien dans un ECR. Par le lemme 10.3.1, on conclut $t \in \mathcal{SN}_{\mu\tilde{\mu}}$. ■

Chapitre 11

Normalisation forte du $\overline{\lambda}\mu\tilde{\mu}$-calcul

On étend le calcul du chapitre précédent en ajoutant les deux λ-abstractions duales ainsi que les piles de termes et de contextes.

11.1 Définition du calcul

Il y a trois catégories syntaxiques : les termes, les contextes et les commandes ; notés respectivement v, e, c. On donne deux ensembles de variables : Var est l'ensemble des variables de termes, ce sont des valeurs notées x, y, z etc. ; $Var^{\perp}$ est l'ensemble des variables de contextes, ce sont des contextes notés α, β, γ etc. La syntaxe du $\overline{\lambda}\mu\tilde{\mu}$-calcul est :

$$\begin{array}{lcl} c & ::= & \langle v|e\rangle \\ v & ::= & x \mid \lambda x.v \mid e \cdot v \mid \mu\alpha.c \\ e & ::= & \alpha \mid \alpha\lambda.e \mid v \cdot e \mid \tilde{\mu}x.c \end{array}$$

Les règles de réduction sont données dans le tableau suivant. Les règles (μ) et $(\tilde{\mu})$ forment une paire critique :

$$\begin{array}{llcll} (\beta) & \langle \lambda x.v | v' \cdot e\rangle & \to & \langle v' | \tilde{\mu}x.\langle v|e\rangle\rangle & \text{si } x \notin FV(e) \\ (\tilde{\beta}) & \langle e' \cdot v | \alpha\lambda.e\rangle & \to & \langle \mu\alpha.\langle v|e\rangle | e'\rangle & \text{si } \alpha \notin FV(v) \\ (\mu) & \langle \mu\alpha.c | e\rangle & \to & c[e/\alpha] & \\ (\tilde{\mu}) & \langle v | \tilde{\mu}x.c\rangle & \to & c[v/x] & \\ & & & & \\ (sv) & \mu\alpha.\langle v|\alpha\rangle & \to & v & \text{si } \alpha \notin FV(v) \\ (se) & \tilde{\mu}x.\langle x|e\rangle & \to & e & \text{si } x \notin FV(e) \end{array}$$

Trois formes de séquents typent les catégories syntaxiques : les commandes sont typées par $(\Gamma \vdash \Delta)$, les termes par $\Gamma \vdash A|\Delta$ et les contextes par $\Gamma|A \vdash \Delta$. On regarde à présent les règles de typage :

$$\frac{\Gamma \vdash v : A|\Delta \quad \Gamma|e : A \vdash \Delta}{\langle v|e\rangle : (\Gamma \vdash \Delta)}$$

$$\frac{}{\Gamma|\alpha : A \vdash \Delta, \alpha : A} \quad \frac{}{\Gamma, x : A \vdash \Delta|x : A}$$

$$\frac{\Gamma|e : B \vdash \alpha : A, \Delta}{\Gamma|\alpha\lambda.e : A - B \vdash \Delta} \quad \frac{\Gamma, x : A \vdash v : B|\Delta}{\Gamma \vdash \lambda x.v : A \to B|\Delta}$$

$$\frac{\Gamma \vdash v : A|\Delta \quad \Gamma|e : B \vdash \Delta}{\Gamma|v \cdot e : A \to B \vdash \Delta} \quad \frac{\Gamma \vdash v : B|\Delta \quad \Gamma|e : A \vdash \Delta}{\Gamma \vdash e \cdot v : A - B|\Delta}$$

$$\frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma|\tilde{\mu}x.c : A \vdash \Delta} \quad \frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A|\Delta}$$

11.2 Définition des ECR

On définit simultanément et par induction sur la structure du type :
– les opérateurs :

$$\begin{array}{lll} \mathit{Lambda}(X_1, X_2) & =_{Def} & \{\lambda x.v \mid \forall v' \in X_1, e \in X_2 \ \langle v[v'/x]|e\rangle \in [\![\vdash]\!]\} \\ \mathit{Cons}(X_1, X_2) & =_{Def} & \{v \cdot e \mid v \in X_1 \text{ et } e \in X_2\} \\ & & \\ \widetilde{\mathit{Lambda}}(X_1, X_2) & =_{Def} & \{\alpha\lambda.e \mid \forall e' \in X_1, v \in X_2 \ \langle v|e[e'/\alpha]\rangle \in [\![\vdash]\!]\} \\ \widetilde{\mathit{Cons}}(X_1, X_2) & =_{Def} & \{e \cdot v \mid e \in X_1 \text{ et } v \in X_2\} \\ & & \\ \mathit{Mu}(X) & =_{Def} & \{\mu\alpha.c \mid \forall e \in X \ \ c[e/\alpha] \in [\![\vdash]\!]\} \\ \widetilde{\mathit{Mu}}(X) & =_{Def} & \{\tilde{\mu}x.c \mid \forall v \in X \ \ c[v/x] \in [\![\vdash]\!]\} \end{array}$$

Puis,
- si A est atomique

$$\mathit{Neg}_{[\![\vdash A]\!]}(Y) = \mathit{Var} \cup \mathit{Mu}(Y)$$

$$\mathit{Neg}_{[\![A\vdash]\!]}(X) = \mathit{Var}^{\perp} \cup \widetilde{\mathit{Mu}}(X)$$

- si $A = A_1 \to A_2$

$$\mathit{Neg}_{[\![\vdash A]\!]}(Y) = \mathit{Var} \cup \mathit{Mu}(Y) \cup \mathit{Lambda}([\![\vdash A_1]\!], [\![A_2 \vdash]\!])$$

$$\mathit{Neg}_{[\![A\vdash]\!]}(X) = \mathit{Var}^{\perp} \cup \widetilde{\mathit{Mu}}(X) \cup \mathit{Cons}([\![\vdash A_1]\!], [\![A_2 \vdash]\!])$$

- si $A = A_1 - A_2$

$$\mathit{Neg}_{[\![\vdash A]\!]}(Y) = \mathit{Var} \cup \mathit{Mu}(Y) \cup \widetilde{\mathit{Cons}}([\![A_1 \vdash]\!], [\![\vdash A_2]\!])$$

$$\mathit{Neg}_{[\![A\vdash]\!]}(X) = \mathit{Var}^{\perp} \cup \widetilde{\mathit{Mu}}(X) \cup \widetilde{\mathit{Lambda}}([\![A_1 \vdash]\!], [\![\vdash A_2]\!])$$

Neg est un opérateur décroissant, donc $\mathit{Neg}_{[\![\vdash A]\!]} \circ \mathit{Neg}_{[\![A\vdash]\!]}$ est un opérateur croissant. D'après Tarski, il existe alors un point fixe X_0^A.

– Les ensembles de réductibilité :

$$[\![\vdash]\!] = \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$$

Puis

$$[\![\vdash A]\!] = X_0^A \quad \text{et} \quad [\![A \vdash]\!] = \mathit{Neg}_{[\![A\vdash]\!]}(X_0^A)$$

Remarque 11.2.1

On sait que $Neg_{[\![A\vdash]\!]}(X_0^A)$ est un point fixe (voir la remarque 10.2.3).

Propriété 11.2.2 (Bonne définition)

Les ECR définis ci-dessus vérifient

$$
\begin{array}{lll}
\text{(i)} & Var \subset [\![\vdash A]\!] & \\
\text{(ii)} & Var^{\perp} \subset [\![A \vdash]\!] & \\
\text{(iii)} & v \in [\![\vdash A]\!] \iff & \text{soit } v = x \\
 & & \text{soit } v = e \cdot v' \text{ avec } A = A_1 - A_2, \\
 & & \quad e \in [\![A_1 \vdash]\!] \text{ et } v' \in [\![\vdash A_2]\!] \\
 & & \text{soit } v = \mu\alpha.c \text{ et} \\
 & & \quad \forall e \in [\![A \vdash]\!] \;\; c[e/\alpha] \in [\![\vdash]\!] \\
 & & \text{soit } v = \lambda x.v' \text{ avec } A = A_1 \to A_2 \text{ et} \\
 & & \quad \forall v'' \in [\![\vdash A_1]\!], e \in [\![A_2 \vdash]\!] \;\; \langle v'[v''/x] | e \rangle \in [\![\vdash]\!] \\
 & & \\
\text{(iv)} & e \in [\![A \vdash]\!] \iff & \text{soit } e = \alpha \\
 & & \text{soit } e = v \cdot e' \text{ avec } A = A_1 \to A_2, \\
 & & \quad v \in [\![\vdash A_1]\!] \text{ et } e' \in [\![A_2 \vdash]\!] \\
 & & \text{soit } e = \tilde{\mu}x.c \text{ et} \\
 & & \quad \forall v \in [\![\vdash A]\!] \;\; c[v/x] \in [\![\vdash]\!] \\
 & & \text{soit } e = \alpha\lambda.e' \text{ avec } A = A_1 - A_2 \text{ et} \\
 & & \quad \forall e'' \in [\![A_1 \vdash]\!], v \in [\![\vdash A_2]\!] \;\; \langle v | e'[e''/\alpha] \rangle \in [\![\vdash]\!]
\end{array}
$$

Preuve : Par définition des ensembles de réductibilité, on a $[\![\vdash]\!] = \mathcal{SN}^{-}_{\overline{\lambda\mu\tilde{\mu}}}$ et les points (i) et (ii). On prouve les points (iii) et (iv). En fait, on se contente de prouver (iii), car (iv) est symétriquement identique.

$$
\begin{array}{c}
v \in [\![\vdash A]\!] \\
\iff \\
v \in Neg_{[\![\vdash A]\!]} \circ Neg_{[\![A\vdash]\!]}([\![\vdash A]\!])
\end{array}
$$

Arrivés ici, il nous faut considérer les différentes formes de A.

- Si $A = A_1 \to A_2$

$$
\begin{array}{c}
v \in Neg_{[\![\vdash A]\!]} \circ Neg_{[\![A\vdash]\!]}([\![\vdash A]\!]) \\
\iff \\
v \in Var \cup Mu(Neg_{[\![A\vdash]\!]}([\![\vdash A]\!])) \cup Lambda_{\Gamma\vdash A|\Delta}([\![\vdash A_1]\!], [\![A_2 \vdash]\!]) \\
\iff \\
\text{soit } v = x \\
\text{soit } v = \mu\alpha.c \text{ et } \mu\alpha.c \in Mu(Neg_{[\![A\vdash]\!]}([\![\vdash A]\!])) \\
\text{soit } v = \lambda x.v' \text{ et } \lambda x.v' \in Lambda_{\Gamma\vdash A|\Delta}([\![\vdash A_1]\!], [\![A_2 \vdash]\!])
\end{array}
$$

avec

$$
\begin{array}{c}
\mu\alpha.c \in Mu(Neg_{[\![A\vdash]\!]}([\![\vdash A]\!])) \\
\iff \\
\forall e \in Neg_{[\![A\vdash]\!]}([\![\vdash A]\!]) \;\; c[e/\alpha] \in [\![\vdash]\!] \\
\iff \\
\forall e \in [\![A \vdash]\!] \;\; c[e/\alpha] \in [\![\vdash]\!]
\end{array}
$$

et

$$\lambda x.v' \in Lambda_{\Gamma\vdash A|\Delta}([\![\vdash A_1]\!],[\![A_2\vdash]\!])$$
$$\Longleftrightarrow$$
$$\forall v''\in[\![\vdash A_1]\!], e\in[\![A_2\vdash]\!]\ \ \langle v'[v''/x]|e\rangle\in[\![\vdash]\!]$$
$$\Longleftrightarrow$$
$$\forall v''\in[\![\vdash A_1]\!], e\in[\![A_2\vdash]\!]\ \ \langle v'[v''/x]|e\rangle\in[\![\vdash]\!]$$

- Si $A = A_1 - A_2$

$$v\in Neg_{[\![\vdash A]\!]}\circ Neg_{[\![A\vdash]\!]}([\![\vdash A]\!])$$
$$\Longleftrightarrow$$
$$v\in Var\cup Mu(Neg_{[\![A\vdash]\!]}([\![\vdash A]\!]))\cup\widetilde{Cons}([\![A_1\vdash]\!],[\![\vdash A_2]\!])$$
$$\Longleftrightarrow$$
$$\text{soit } v=x$$
$$\text{soit } v=\mu\alpha.c \text{ et } \mu\alpha.c\in Mu(Neg_{[\![A\vdash]\!]}([\![\vdash A]\!]))$$
$$\text{soit } v=e\cdot v' \text{ et } e\cdot v'\in\widetilde{Cons}([\![A_1\vdash]\!],[\![\vdash A_2]\!])$$

avec la même preuve pour $\mu\alpha.c$ et

$$e\cdot v'\in\widetilde{Cons}([\![A_1\vdash]\!],[\![\vdash A_2]\!])$$
$$\Longleftrightarrow$$
$$e\in[\![A_1\vdash]\!] \text{ et } v'\in[\![\vdash A_2]\!]$$

- Sinon, si A est atomique

$$v\in Neg_{[\![\vdash A]\!]}\circ Neg_{[\![A\vdash]\!]}([\![\vdash A]\!])$$
$$\Longleftrightarrow$$
$$v\in Var\cup Mu(Neg_{[\![A\vdash]\!]}([\![\vdash A]\!]))$$
$$\Longleftrightarrow$$
$$\text{soit } v=x$$
$$\text{soit } v=\mu\alpha.c \text{ et } \mu\alpha.c\in Mu(Neg_{[\![A\vdash]\!]}([\![\vdash A]\!]))$$

avec la même preuve pour $\mu\alpha.c$.

■

11.3 Propriétés des ECR

Voici les lemmes de normalisation forte des ECR et de fermeture par réduction.

Lemme 11.3.1 (Normalisation forte des ECR)

Soit A un type. Alors

1. $[\![\vdash A]\!]\subset\mathcal{SN}_{\overline{\lambda}\mu\tilde{\mu}}$
2. $[\![A\vdash]\!]\subset\mathcal{SN}_{\overline{\lambda}\mu\tilde{\mu}}$
3. $[\![\vdash]\!]\subset\mathcal{SN}_{\overline{\lambda}\mu\tilde{\mu}}$

Preuve : Par induction sur la structure de A

1. On considère les différentes formes de $v\in[\![\vdash A]\!]$:
   - $v = x$: alors $v\in\mathcal{SN}_{\overline{\lambda}\mu\tilde{\mu}}$
   - $v = e\cdot v'$: alors $A = A_1 - A_2$ et on conclut en appliquant deux fois l'hypothèse d'induction.
   - $v = \mu\alpha.c$: par le point (ii) de la proposition 11.2.2, $\alpha\in[\![A\vdash]\!]$, alors, par le point (iii) de la proposition 11.2.2, $c[\alpha/\alpha]\in[\![\vdash]\!]$, ce qui nous donne $c\in[\![\vdash]\!](=\mathcal{SN}_{\overline{\lambda}\mu\tilde{\mu}})$. On a donc $\mu\alpha.c\in\mathcal{SN}_{\overline{\lambda}\mu\tilde{\mu}}$.

- $v = \lambda x.v'$, alors $A = A_1 \to A_2$: pour que $v \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$, il suffit que $v' \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$. Par réductibilité de $\lambda x.v'$, on a $\forall v'' \in [\![\vdash A_1]\!], e \in [\![A_2 \vdash]\!]\ \langle v'[v''/x]|e\rangle \in [\![\vdash]\!] (= \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}})$. Par les points (i) et (ii) de la proposition 11.2.2, on peut prendre x pour v'' et α pour e, ce qui nous donne $\langle v'[x/x]|\alpha\rangle \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$. On en déduit $v' \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$ et on conclut.

2. Le cas pour e est traité de façon similaire au cas ci-dessus en considérant la symétrie
3. Par définition $[\![\vdash]\!] = \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$

■

Lemme 11.3.2 (Fermeture par réduction)

1. $v \in [\![\vdash A]\!],\ v \to v' \implies v' \in [\![\vdash A]\!]$
2. $e \in [\![A \vdash]\!],\ e \to e' \implies e' \in [\![A \vdash]\!]$
3. $c \in [\![\vdash]\!],\ c \to c' \implies c' \in [\![\vdash]\!]$

Preuve : Par induction sur A, en considérant les différentes formes de v, e, et c.

1.1. $v = x$: alors il n'y a pas de réduction possible.

1.2. $v = e_1 \cdot v_1$: deux réductions sont possibles $e_1 \cdot v_1 \to e_2 \cdot v_1$ ou $e_1 \cdot v_1 \to e_1 \cdot v_2$. Dans les deux cas, on conclut en appliquant l'hypothèse d'induction.

1.3. $v = \mu\alpha.c$: deux cas sont possibles.

- La réduction est $\mu\alpha.c \to \mu\alpha.c'$. Par définition de $\mu\alpha.c \in [\![\vdash A]\!]$ on a, $\forall e \in [\![A \vdash]\!]\ \ c[e/\alpha] \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$. On a alors $c'[e/\alpha] \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$ (toujours pour tout $e \in [\![A \vdash]\!]$) et on conclut à l'aide du point (iii) de la proposition 11.2.2.
- La réduction est $\mu\alpha.\langle v|\alpha\rangle \to v$ avec $\alpha \notin FV(v)$. On sait par hypothèse que $\mu\alpha.\langle v|\alpha\rangle \in [\![\vdash A]\!]$ donc, par le point (iii) de la proposition 10.2.4, $\forall e \in [\![A \vdash]\!]\ \ \langle v|\alpha\rangle[e/\alpha] \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$, c'est-à-dire $\langle v|e\rangle \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$. Si v est une variable, alors la conclusion est immédiate. Si $v = \mu\beta.c$, $\langle \mu\beta.c|e\rangle \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$ implique que $c[e/\beta] \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$, ce qui nous donne $\mu\beta.c \in [\![\vdash A]\!]$ par le point (iii) de la proposition 10.2.4. Si $v = \lambda x.v'$, $\langle \lambda x.v'|e\rangle \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$ nous donne, pour $e = v_1 \cdot e_1$, $\langle v_1|\tilde{\mu}x.\langle v'|e_1\rangle\rangle \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$ puis $\langle v'|e_1\rangle[v_1/x] \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$ et $\langle v'[v_1/x]|e_1[v_1/x]\rangle \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$ et enfin, puisque x n'est pas libre dans e_1, $\langle v'[v_1/x]|e_1\rangle \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$, ce qui est suffisant, grâce aux point (iv) et (iii) de la proposition 10.2.4, pour conclure.

1.4. $v = \lambda x.v'$: $A = A_1 \to A_2$ et la réduction est $\lambda x.v' \to \lambda x.v''$. Par le point (iv) de la proposition 11.2.2, on sait que $\forall v''' \in [\![\vdash A_1]\!], e \in [\![A_2 \vdash]\!]\ \ \langle v'[v'''/x]|e\rangle \in [\![\vdash]\!] = \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$, donc $\forall v''' \in [\![\vdash A_1]\!], e \in [\![A_2 \vdash]\!]\ \langle v''[v'''/x]|e\rangle \in [\![\vdash]\!] = \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$, ce qui nous permet de conclure.

2.x. Comme pour les cas 1.x. en respectant la symétrie (avec x allant de 1 à 4).

3. $c \in [\![\vdash]\!]$: alors $c \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$ et $c \to c'$ implique que $c' \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}} = [\![\vdash]\!]$

■

Voici à présent les lemmes qui nous permettent de "construire inductivement" l'appartenance des termes aux ECR.

Lemme 11.3.3

$$v \in [\![\vdash A]\!],\ e \in [\![A \vdash]\!] \implies \langle v|e\rangle \in [\![\vdash]\!]$$

Preuve : Montrer que $\langle v|e\rangle \in [\![\vdash]\!]$ revient à montrer que $\langle v|e\rangle \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$. On prend toutes les paires possibles pour v et e et on raisonne par induction sur la forte normalisation de v et e (obtenue grâce au lemme 11.3.1). On regarde les réductions possibles de $\langle v|e\rangle$. Si la réduction a lieu dans v ou e, on conclut par hypothèse d'induction et par le lemme 11.3.2. Sinon,

- si $v = \mu\alpha.c$ et si la réduction est $\langle \mu\alpha.c|e\rangle \to c[e/\alpha]$ et on conclut par définition de $\mu\alpha.c \in [\![\vdash A]\!]$,
- si $e = \tilde{\mu}x.c$, on conclut symétriquement au point précédent,

- si $v = \lambda x.v'$ et $e = v'' \cdot e'$ (avec $A = A_1 \to A_2$), et si la réduction est $\langle \lambda x.v | v'' \cdot e' \rangle \to \langle v'' | \tilde{\mu}x.\langle v'|e'\rangle\rangle$, on regarde les réductions possibles de $\langle v''|\tilde{\mu}x.\langle v'|e'\rangle\rangle$. Par réductibilité de v, on a $\langle v'[x/x]|e'\rangle \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$ et on a d'autre part $v'' \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$. Ainsi, les réductions ne pouvant pas avoir lieu infiniment dans ces termes, on finira par effectuer l'une des réductions suivantes (où $v'' \to^* v_1$, $\langle v'|e'\rangle \to^* \langle v_2|e_2\rangle$) :
  - $\langle v_1|\tilde{\mu}x.\langle x|e_2\rangle\rangle \to \langle v_1|e_2\rangle$: par hypothèse d'induction, on a $\langle v''|e'\rangle \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$ et $\langle v_1|e_2\rangle$ est l'un de ses réduits.
  - $\langle v_1|\tilde{\mu}x.\langle v_2|e_2\rangle\rangle \to \langle v_2[v_1/x]|e_2[v_1/x]\rangle$: ce terme est aussi un réduit de $\langle v'[v''/x]|e'[v''/x]\rangle$ qui est dans $\mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$ par réductibilité de v.
  - $\langle \mu\alpha.c_1|\tilde{\mu}x.\langle v_2|e_2\rangle\rangle \to c_1[\tilde{\mu}x.\langle v_2|e_2\rangle/\alpha]$ avec $v_1 = \mu\alpha.c_1$. Par réductibilité de e et par le lemme 11.3.2 on a $\mu\alpha.c_1 \in [\![\vdash A_1]\!]$, ce qui nous donne, par définition, que $c_1[\tilde{\mu}x.\langle v_2|e_2\rangle/\alpha]$ appartient à $[\![\vdash]\!]$ si et seulement si $\tilde{\mu}x.\langle v_2|e_2\rangle$ appartient à $[\![A_1 \vdash]\!]$. Or cette dernière condition est vérifiée, par définition, si et seulement si $\forall v_3 \in [\![\vdash A_1]\!]$ on a $\langle v_2[v_3/x]|e_2[v_3/x]\rangle \in [\![\vdash]\!]$, ce qui est une conséquence de la réductibilité de v.
- si $e = \alpha\lambda.e'$ et $v = e'' \cdot v'$, on conclut symétriquement au point précédent,
- dans tous les autres cas, aucune réduction n'est possible.

■

Lemme 11.3.4

- Si $v[v'/x] \in [\![\vdash B]\!]$ pour tout $v' \in [\![\vdash A]\!]$ alors $\lambda x.v \in [\![\vdash A \to B]\!]$.
- Si $e[e'/\alpha] \in [\![B \vdash]\!]$ pour tout $e' \in [\![A \vdash]\!]$ alors $\alpha\lambda.e \in [\![\vdash A - B]\!]$.

Preuve : Par symétrie, il nous suffit de montrer l'une des implications, prenons la première. Pour prouver que $\lambda x.v \in [\![\vdash A \to B]\!]$, il faut, par le point (iii) de la proposition 11.2.2, prouver que pour tout $v' \in [\![\vdash A]\!], e \in [\![B \vdash]\!]$, $\langle v[v'/x]|e\rangle \in [\![\vdash]\!]$. Par hypothèse, on a $v[v'/x] \in [\![\vdash B]\!]$. On conclut en utilisant le lemme 11.3.3. ■

11.4 Preuve de normalisation forte

Voici enfin le lemme d'adéquation.

Lemme 11.4.1 (Adéquation)

Soit A un type et t un terme du langage avec $FV(t) \subset X_1 \cup X_2$ ($X_1 \subset Var$ et $X_2 \subset Var^{\perp}$) et les variables $x_i \in X_1$ sont de type B_i et les variables $\alpha_j \in X_2$ sont de type C_j. Pour tout ensemble de termes v_i, e_j tels que $\forall i \;\; v_i \in [\![\vdash A_i]\!]$ et $\forall j \;\; e_j \in [\![B_j \vdash]\!]$, on a selon la forme de t

1. si $X_1 : B \vdash v : A | X_2 : C$ alors
$$v[v_1/x_1, ..., v_n/x_n, e_1/\alpha_1, ..., e_m/\alpha_m] \in [\![\vdash A]\!]$$
2. si $X_1 : B | e : A \vdash X_2 : C$ alors
$$e[v_1/x_1, ..., v_n/x_n, e_1/\alpha_1, ..., e_m/\alpha_m] \in [\![A \vdash]\!]$$
3. si $c : (X_1 : B \vdash X_2 : C)$ alors
$$c[v_1/x_1, ..., v_n/x_n, e_1/\alpha_1, ..., e_m/\alpha_m] \in [\![\vdash]\!]$$

On rappelle que la notation $X_1 : B$ est un raccourci pour une énumération de $\{x_i : B_i | i \in [1, n]\}$ (idem pour $X_2 : C$).

Preuve : On notera $[//]$ la substitution $[x_1/v_1, ..., x_n/v_n, \alpha_1/e_1, ..., \alpha_m/e_m]$. On raisonne par induction sur la structure de t

- $v = x$: alors, par hypothèse, $\exists i \in [1, n]\ \ A = B_i$. Ainsi

$$v[v_1/x_1, ..., v_n/x_n, e_1/\alpha_1, ..., e_m/\alpha_m] = v_i \in [\![\vdash B_i]\!] = [\![\vdash A]\!]$$

- $v = e \cdot v'$: par hypothèse d'induction sur e et v' et par le point (iii) de la proposition 11.2.2, le résultat est immédiat.
- $v = \lambda x.v'$: on a alors $A = A' \to A''$. Puisqu'on peut renommer les variables liées, on peut supposer que $x \notin \{x_1, ..., x_n\}$, ce qui nous donne $(\lambda x.v')[//] = \lambda x.(v'[//])$. Par hypothèse d'induction, pour tout $v'' \in [\![\vdash A']\!]$ on a $v'[v''/x, //] \in [\![\vdash A'']\!]$ et par le lemme 11.3.4, on peut conclure.
- $v = \mu\alpha.c$: puisqu'on peut renommer les variables liées, on peut supposer que $\alpha \notin \{\alpha_1, ..., \alpha_m\}$. Maintenant, par le point (iii) de la proposition 10.2.4, pour prouver que

$$(\mu\alpha.c)[//] = \mu\alpha.(c[//]) \in [\![\vdash A]\!]$$

  il est suffisant de prouver que, pour tout $e \in [\![A \vdash]\!]$,

$$c[e/\alpha, //] \in [\![\vdash]\!]$$

  ce qui est fait par hypothèse d'induction.
- $e = \alpha$: ce cas est le même que $v = x$ par symétrie.
- $e = v \cdot e'$: ce cas est le même que $v = e \cdot v'$ par symétrie.
- $e = \alpha\lambda.e'$: ce cas est le même que $v = \lambda x.v'$ par symétrie.
- $e = \widetilde{\mu}x.c$: ce cas est le même que $v = \mu\alpha.c$ par symétrie.
- $c = \langle v|e\rangle$. Par hypothèse d'induction sur v et e et par le lemme 11.3.3 le résultat est immédiat.

■

On peut maintenant passer au théorème principal de ce chapitre.

Théorème 11.4.2

Le $\overline{\lambda}\mu\tilde{\mu}$-calcul est fortement normalisant.

Preuve : La preuve est identique à celle du théorème 10.4.4. ■

Chapitre 12

Normalisation forte du $\overline{\lambda}\mu\tilde{\mu}$-calcul avec substitutions explicites

On ajoute à présent les substitutions explicites. On souhaite utiliser la technique PSN implique SN présentée dans la deuxième partie de la thèse. Pour cela, on procède en trois étapes. Après avoir introduit le $\overline{\lambda}\mu\tilde{\mu}\mathtt{x}$-calcul, on commence par donner quelques résultats sur le calcul des substitutions. On prouve ensuite que notre calcul préserve la normalisation forte (PSN) en utilisant la technique de perpétuité formalisé dans [13], laquelle s'appuie sur la preuve de PSN proposée dans [9]. Enfin, on applique la technique PSN implique SN.

12.1 Définition du calcul

Aux trois catégories syntaxiques présentées dans le chapitre précédent, on ajoute une quatrième concernant les substitutions explicites, notée τ. Dans la suite de notre travail, on notera $*$ une variable de terme ou de contexte dont il n'est pas nécessaire de connaître précisément l'ensemble d'origine, et on notera t un élément syntaxique indéterminé parmi v, e et c. La syntaxe du $\overline{\lambda}\mu\tilde{\mu}\mathtt{x}$-calcul est :

$$\begin{array}{rcl} c & ::= & \langle v|e\rangle \mid c\tau \\ v & ::= & x \mid \lambda x.v \mid e \cdot v \mid \mu\alpha.c \mid v\tau \\ e & ::= & \alpha \mid \alpha\lambda.e \mid v \cdot e \mid \tilde{\mu}x.c \mid e\tau \\ \tau & ::= & [x \leftarrow v] \mid [\alpha \leftarrow e] \end{array}$$

La source $Dom(\tau)$ de τ est x si $\tau = [x \leftarrow v]$ et α si $\tau = [\alpha \leftarrow e]$. Le corps $S(\tau)$ de τ est v dans le premier cas et e dans le second. On dira par la suite qu'une substitution appartient à $\mathcal{SN}_{\overline{\lambda}\mu\tilde{\mu}\mathtt{x}}$ si son corps appartient lui-même à cet ensemble.

Quatre formes de séquents typent les catégories syntaxique : les commandes sont typées par $(\Gamma \vdash \Delta)$, les termes par $\Gamma \vdash A|\Delta$, les contextes par $\Gamma|A \vdash \Delta$, une forme supplémentaire de séquent permet de typer une substitution τ par $(\Gamma \vdash \Delta) \Rightarrow (\Gamma' \vdash \Delta')$. On regarde à présent les règles de typage :

$$\frac{\Gamma \vdash v : A|\Delta \quad \Gamma|e : A \vdash \Delta}{\langle v|e\rangle : (\Gamma \vdash \Delta)}$$

$$\frac{}{\Gamma|\alpha : A \vdash \Delta, \alpha : A} \quad \frac{}{\Gamma, x : A \vdash \Delta | x : A}$$

$$\frac{\Gamma|e : B \vdash \alpha : A, \Delta}{\Gamma|\alpha\lambda.e : A - B \vdash \Delta} \quad \frac{\Gamma, x : A \vdash v : B|\Delta}{\Gamma \vdash \lambda x.v : A \to B|\Delta}$$

$$\frac{\Gamma \vdash v : A|\Delta \quad \Gamma|e : B \vdash \Delta}{\Gamma|v \cdot e : A \to B \vdash \Delta} \quad \frac{\Gamma \vdash v : B|\Delta \quad \Gamma|e : A \vdash \Delta}{\Gamma \vdash e \cdot v : A - B|\Delta}$$

$$\frac{c : (\Gamma, x : A \vdash \Delta)}{\Gamma|\tilde{\mu}x.c : A \vdash \Delta} \quad \frac{c : (\Gamma \vdash \alpha : A, \Delta)}{\Gamma \vdash \mu\alpha.c : A|\Delta}$$

Voici les règles pour typer les substitutions explicites :

$$\frac{\Gamma \vdash v : A|\Delta}{[x \leftarrow v] : (\Gamma, x : A \vdash \Delta) \Rightarrow (\Gamma \vdash \Delta)} \qquad \frac{\Gamma|e : A \vdash \Delta}{[\alpha \leftarrow e] : (\Gamma \vdash \Delta, \alpha : A) \Rightarrow (\Gamma \vdash \Delta)}$$

$$\frac{\Gamma|e : A \vdash \Delta \quad \tau : (\Gamma \vdash \Delta) \Rightarrow (\Gamma' \vdash \Delta')}{\Gamma'|e\tau : A \vdash \Delta'} \qquad \frac{\Gamma \vdash v : A|\Delta \quad \tau : (\Gamma \vdash \Delta) \Rightarrow (\Gamma' \vdash \Delta')}{\Gamma' \vdash v\tau : A|\Delta'}$$

$$\frac{c : (\Gamma \vdash \Delta) \quad \tau : (\Gamma \vdash \Delta) \Rightarrow (\Gamma' \vdash \Delta')}{c\tau : (\Gamma' \vdash \Delta')}$$

Les règles de réduction sont données dans le tableau suivant. Les nouvelles règles (mu) et $(\widetilde{mu})$ forment aussi une paire critique :

(β)	$\langle \lambda x.v \mid v' \cdot e \rangle$	$\to$	$\langle v' \mid \tilde{\mu}x.\langle v \mid e \rangle \rangle$	si $x \notin FV(e)$
$(\tilde{\beta})$	$\langle e' \cdot v \mid \alpha\lambda.e \rangle$	$\to$	$\langle \mu\alpha.\langle v \mid e \rangle \mid e' \rangle$	si $\alpha \notin FV(v)$
(mu)	$\langle \mu\alpha.c \mid e \rangle$	$\to$	$c[\alpha \leftarrow e]$	
$(\widetilde{mu})$	$\langle v \mid \tilde{\mu}x.c \rangle$	$\to$	$c[x \leftarrow v]$	
(sv)	$\mu\alpha.\langle v \mid \alpha \rangle$	$\to$	v	si $\alpha \notin FV(v)$
(se)	$\tilde{\mu}x.\langle x \mid e \rangle$	$\to$	e	si $x \notin FV(e)$
$(c\tau)$	$\langle v \mid e \rangle \tau$	$\to$	$\langle v\tau \mid e\tau \rangle$	
$(x\tau 1)$	$x\tau$	$\to$	$S(\tau)$	Si $x \in Dom(\tau)$
$(x\tau 2)$	$x\tau$	$\to$	x	Si $x \notin Dom(\tau)$
$(\alpha\tau 1)$	$\alpha\tau$	$\to$	$S(\tau)$	Si $\alpha \in Dom(\tau)$
$(\alpha\tau 2)$	$\alpha\tau$	$\to$	α	Si $\alpha \notin Dom(\tau)$
$(\cdot\tau)$	$(v \cdot e)\tau$	$\to$	$(v\tau) \cdot (e\tau)$	
$(\widetilde{\cdot}\tau)$	$(e \cdot v)\tau$	$\to$	$(e\tau) \cdot (v\tau)$	
$(\lambda\tau)$	$(\lambda x.v)\tau$	$\to$	$\lambda x.(v\tau)$	
$(\tilde{\lambda}\tau)$	$(\alpha\lambda.e)\tau$	$\to$	$\alpha\lambda.(e\tau)$	
$(\mu\tau)$	$(\mu\alpha.c)\tau$	$\to$	$\mu\alpha.(c\tau)$	
$(\tilde{\mu}\tau)$	$(\tilde{\mu}x.c)\tau$	$\to$	$\tilde{\mu}x.(c\tau)$	

Pour les règles $(\mu\tau)$ et $(\tilde{\lambda}\tau)$ (resp. $(\tilde{\mu}\tau)$ et $(\lambda\tau)$) on raisonne modulo α-conversion sur la variable liée α (resp. x).

12.2 Le calcul des substitutions

On notera :

- $\mathtt{x}$ l'ensemble des règles concernant la propagation des substitutions : $c\tau$, $x\tau 1$, $x\tau 2$, $\alpha\tau 1$, $\alpha\tau 2$, $\cdot\tau$, $\widetilde{\cdot}\tau$, $\lambda\tau$, $\widetilde{\lambda}\tau$, $\mu\tau$ et $\widetilde{\mu}\tau$,
- $\neg\mathtt{x}$ l'ensemble des règles qui ne sont pas dans $\mathtt{x}$, c'est-à-dire celle concernant les réductions du calcul originel : β, $\widetilde{\beta}$, mu, $\widetilde{mu}$, sv et se.

On présente ci-après quelques résultats connus sur les calculs des substitutions [11, 13].

Lemme 12.2.1 (Normalisation forte de x)

Le système $\mathtt{x}$ est fortement normalisant et ses formes normales sont des objets purs (sans substitutions).

Preuve : Soit la mesure h définie de la façon suivante :

$$\begin{array}{lcl} h(*) & = & 1 \\ h(\langle v|e\rangle) & = & h(v)+h(e)+1 \\ h(v\cdot e) & = & h(v)+h(e)+1 \\ h(e\cdot v) & = & h(v)+h(e)+1 \\ h(\lambda x.v) & = & h(v)+1 \\ h(\alpha\lambda.e) & = & h(e)+1 \\ h(\mu\alpha.c) & = & h(c)+1 \\ h(\widetilde{\mu}x.c) & = & h(c)+1 \\ h(t[*\leftarrow t']) & = & h(t)\times(h(t')+1) \end{array}$$

On vérifie facilement que chaque $\mathtt{x}$-réduction fait décroître strictement h. On prouve par l'absurde que les formes normales sont des objets purs : si il reste une substitution, on regarde l'objet auquel elle est appliquée et on trouve une réduction à effectuer. ■

On notera $\mathtt{x}(t)$ la $\mathtt{x}$-forme normale d'un objet t.

Lemme 12.2.2 (Confluence de $\mathtt{x}$)

Le système $\mathtt{x}$ est confluent.

Preuve : Il n'y a pas de paire critique dont les redex soient superposés, ce qui nous donne la confluence locale. En utilisant le lemme de Newman et le lemme 12.2.1 on obtient la confluence. ■

Lemme 12.2.3 (Substitution)

$\mathtt{x}(t[*\leftarrow t']) = \mathtt{x}(t)\{*\leftarrow \mathtt{x}(t')\}$

Preuve : On montre, par récurrence sur la somme des tailles de t et des t_i, que

$$\mathtt{x}(t[*_1\leftarrow t_1]...[*_n\leftarrow t_n]) = \mathtt{x}(t)\{*_1\leftarrow \mathtt{x}(t_1)\}...\{*_n\leftarrow \mathtt{x}(t_n)\}$$

– $t = *_i$: on a

$$\begin{array}{lcl} & & \mathtt{x}(*_i[*_1\leftarrow t_1]...[*_n\leftarrow t_n]) \\ & = & \mathtt{x}(*_i[*_i\leftarrow t_i]...[*_n\leftarrow t_n]) \\ & = & \mathtt{x}(t_i[*_{i+1}\leftarrow t_{i+1}]...[*_n\leftarrow t_n]) \\ & = & \mathtt{x}(t_i)\{*_{i+1}\leftarrow \mathtt{x}(t_{i+1})\}...\{*_n\leftarrow \mathtt{x}(t_n)\} \quad \text{par hyp. de réc.} \\ & = & *_i\{*_{i+1}\leftarrow \mathtt{x}(t_{i+1})\}...\{*_n\leftarrow \mathtt{x}(t_n)\} \\ & = & *_i\{*_1\leftarrow \mathtt{x}(t_1)\}...\{*_n\leftarrow \mathtt{x}(t_n)\} \\ & = & \mathtt{x}(*_i)\{*_1\leftarrow \mathtt{x}(t_1)\}...\{*_n\leftarrow \mathtt{x}(t_n)\} \end{array}$$

– $t = *$: le résultat est évident.
– $t = \langle v|e\rangle$: le résultat est évident en utilisant deux fois l'hypothèse de récurrence.
– $t = v\cdot e$ (ou $e\cdot v$ par symétrie) : le résultat est évident en utilisant deux fois l'hypothèse de récurrence.
– $t = \lambda x.v$ (ou $\alpha\lambda.e$ par symétrie) : le résultat est évident en utilisant l'hypothèse de récurrence.

- $t = \mu\alpha.c$ (ou $\tilde{\mu}x.c$ par symétrie) : le résultat est évident en utilisant l'hypothèse de récurrence.
- $t = t'[* \leftarrow t'']$: la somme des tailles de t et des t_i est plus grande que la somme des tailles de t', t'' et des t_i (on a perdu la substitution), on peut alors appliquer l'hypothèse de récurrence.

$$\begin{array}{lll} & \mathtt{x}(t'[* \leftarrow t''][*_1 \leftarrow t_1]...[*_n \leftarrow t_n]) & \\ = & \mathtt{x}(t')\{* \leftarrow \mathtt{x}(t'')\}\{*_1 \leftarrow \mathtt{x}(t_1)\}...\{*_n \leftarrow \mathtt{x}(t_n)\} & \text{par hyp. de réc.} \\ = & \mathtt{x}(t'[* \leftarrow t''])\{*_1 \leftarrow \mathtt{x}(t_1)\}...\{*_n \leftarrow \mathtt{x}(t_n)\} & \text{par hyp. de réc.} \end{array}$$

■

Lemme 12.2.4 (Simulation du $\overline{\lambda}\mu\tilde{\mu}$-calcul)

> Si $t \rightarrow_{\overline{\lambda}\mu\tilde{\mu}} u$ alors $t \rightarrow^*_{\overline{\lambda}\mu\tilde{\mu}\mathtt{x}} u$.

Preuve : Par induction sur la structure de t, les seuls cas intéressants sont ceux où la réduction à lieu à la racine.

- $\langle\mu\alpha.c|e\rangle \rightarrow_\mu c\{* \leftarrow e\}$: on a

$$\langle\mu\alpha.c|e\rangle \rightarrow_{mu} c[* \leftarrow e] \rightarrow_{\mathtt{x}} \mathtt{x}(c[* \leftarrow e]) \overset{lemme\ 12.2.3}{=} \mathtt{x}(c)\{x \leftarrow \mathtt{x}(e)\}$$

  Puisque $\langle\mu\alpha.c|e\rangle$ est un objet pur, c et e le sont aussi, ce qui signifie que $\mathtt{x}(c) = c$ et $\mathtt{x}(e) = e$. De ce fait, l'objet final est égal à $c\{* \leftarrow e\}$.
- $\langle v|\tilde{\mu}x.c\rangle \rightarrow_\mu c\{* \leftarrow v\}$: ce cas est le même que le précédent, par symétrie.
- Les règles β, $\widetilde{\beta}$, sv et se sont simulées en exactement une étape par leurs homophones dans $\overline{\lambda}\mu\tilde{\mu}\mathtt{x}$.

■

On dit qu'une réduction est *vide* si elle a lieu dans le corps d'une substitution $t[* \leftarrow t']$ telle que $* \notin \mathtt{x}(t)$. On la note $\overset{v}{\rightarrow}$. La substitution elle-même sera appelée *substitution vide*.

Lemme 12.2.5 (Projection)

> 1. Si $t \rightarrow_{\overline{\lambda}\mu\tilde{\mu}\mathtt{x}} u$ alors $\mathtt{x}(t) \rightarrow^*_{\overline{\lambda}\mu\tilde{\mu}} \mathtt{x}(u)$.
> 2. Si $t \rightarrow_{\neg\mathtt{x}} u$ n'est pas une réduction vide, alors $\mathtt{x}(t) \rightarrow^+_{\overline{\lambda}\mu\tilde{\mu}} \mathtt{x}(u)$.

Preuve : Trois cas sont possibles :

- la réduction est $t \rightarrow_{\mathtt{x}} u$. Alors $\mathtt{x}(t) = \mathtt{x}(u)$.
- la réduction est $t \overset{v}{\rightarrow}_{\neg\mathtt{x}} u$. Alors $\mathtt{x}(t) = \mathtt{x}(u)$.
- la réduction est $t \rightarrow_{\neg\mathtt{x}} u$. Le redex est présent dans $\mathtt{x}(t)$ et on peut le réduire, obtenant $\mathtt{x}(u)$. Par exemple, supposons que la règle utilisée soit mu, cela signifie qu'il existe un contexte C et un objet $\langle\mu\alpha.c|e\rangle$ tels que $t = C[\langle\mu\alpha.c|e\rangle]$ et $u = C[c[\alpha \leftarrow e]]$. Comme la réduction n'a pas lieu dans une substitution vide, il existe C', $c' = \mathtt{x}(c)$ et $e' = \mathtt{x}(e)$ tels que $\mathtt{x}(t) = C'[\langle\mu\alpha.c'|e'\rangle]$ et $\mathtt{x}(u) = C[c'[\alpha \leftarrow e']]$. On peut réduire $\mathtt{x}(t)$ vers $\mathtt{x}(u)$ en exactement une étape à l'aide de la règle mu. On procède de façon similaire pour les autres règles.

■

12.3 PSN : autour de la perpétuité

On utilise la technique de perpétuité, formalisée par Bonelli [13]. En fait, on utilise seulement la première partie de cette technique, qui est suffisante pour prouver la normalisation forte. On commence par donner quelques lemmes pour extraire une substitution vide avec une dérivation infinie dans son corps, ainsi que pour pister cette substitution en arrière.

Lemme 12.3.1

Soit la réduction infinie

$$t_0 \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} t_1 \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} t_2 \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} t_3 \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} \cdots$$

Si $\mathtt{x}(t_0) \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$, alors il existe un entier k tel que pour tout $i > k$, on a $t_i \overset{v}{\to}_{\overline{\lambda\mu\tilde{\mu}}} t_{i+1}$.

Preuve : Puisque $\mathtt{x}$ est fortement normalisant, la réduction doit être de la forme

$$t_0 \to^*_{\mathtt{x}} t_1 \to_{\neg\mathtt{x}} t_2 \to^*_{\mathtt{x}} t_3 \to_{\neg\mathtt{x}} t_4 \ldots$$

Par le lemme 12.2.5, on a

$$\mathtt{x}(t_0) \to^*_{\overline{\lambda\mu\tilde{\mu}}} \mathtt{x}(t_1) \to^*_{\overline{\lambda\mu\tilde{\mu}}} \mathtt{x}(t_2) \to^*_{\overline{\lambda\mu\tilde{\mu}}} \mathtt{x}(t_3) \to^*_{\overline{\lambda\mu\tilde{\mu}}} \mathtt{x}(t_4) \ldots$$

avec, de plus, que pour tout i pair, si $t_{i+1} \to_{\neg\mathtt{x}} t_{i+2}$ n'est pas une réduction vide, alors $x(t_i) \to^+_{\overline{\lambda\mu\tilde{\mu}}} \mathtt{x}(t_{i+2})$. Du fait que $\mathtt{x}(t_0) \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$ on déduit qu'il existe k tel que pour tout i pair supérieur à k on a $t_{i+1} \overset{v}{\to}_{\neg\mathtt{x}} t_{i+2}$. Il nous reste à montrer qu'à partir d'un certain point, non seulement les réductions $\neg\mathtt{x}$, mais aussi les $\mathtt{x}$-réductions sont vides. Pour cela, on définit la mesure suivante :

$$\begin{array}{lcl}
h(*) & = & 1 \\
h(\langle v|e\rangle) & = & h(v) + h(e) + 1 \\
h(\mu\alpha.c) & = & h(c) + 1 \\
h(\tilde{\mu}x.c) & = & h(c) + 1 \\
h(\lambda x.v) & = & h(v) + 1 \\
h(\alpha\lambda.e) & = & h(e) + 1 \\
h(v \cdot e) & = & h(v) + h(e) + 1 \\
h(e \cdot v) & = & h(v) + h(e) + 1 \\
h(t[* \leftarrow t']) & = & \begin{cases} h(t) * (h(t') + 1) & \text{si } * \in FV(\mathtt{x}(t)) \\ h(t) * 2 & \text{sinon} \end{cases}
\end{array}$$

La dernière clause nous garantit que les réductions vides laissent la mesure inchangée. On vérifie facilement que toutes les autres réductions font décroître strictement cette mesure, et on conclut. ■

La notion suivante est utile pour isoler une substitution vide.

Définition 12.3.2 (Squelette)

On utilise un symbole $\bullet$ qui peut prendre la place d'un terme ou d'un contexte. Le squelette d'un objet, noté $SK(t)$ est défini inductivement de la façon suivante

$$\begin{array}{lcl}
SK(*) & = & * \\
SK(\langle v|e\rangle) & = & \langle SK(v)|SK(e)\rangle \\
SK(\mu\alpha.c) & = & \mu\alpha.SK(c) \\
SK(\tilde{\mu}x.c) & = & \tilde{\mu}x.SK(c) \\
SK(\lambda x.v) & = & \lambda x.SK(v) \\
SK(\alpha\lambda.e) & = & \alpha\lambda.SK(e) \\
SK(v \cdot e) & = & SK(v) \cdot SK(e) \\
SK(e \cdot v) & = & SK(e) \cdot SK(v) \\
SK(t[* \leftarrow u]) & = & SK(t)[* \leftarrow \bullet]
\end{array}$$

On remarque que si $t \overset{v}{\to} u$, alors $SK(t) = SK(u)$. En effet, la réduction vide ayant lieu dans le corps d'une substitution τ, il existe dans t une substitution τ' dont τ est un sous-terme et qui n'est pas elle-même un sous-terme strict d'une autre substitution. Dans le squelette de t, le corps de τ' est remplacé par un $\bullet$, ce qui rend invisible la réduction qui y a lieu.

Le lemme suivant établit que si il y a une dérivation infinie d'un terme, il existe une substitution vide dans laquelle on peut extraire une dérivation infinie.

Lemme 12.3.3

Soit la réduction infinie

$$t_0 \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} t_1 \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} t_2 \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} t_3 \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} \cdots$$

Si $\mathtt{x}(t_0) \in \mathcal{SN}_{\overline{\lambda\mu\tilde{\mu}}}$, alors il existe un entier k, un objet t, une variable $*$, un contexte C et une suite d'objets u_i tels que

$$\begin{array}{rcl} t_0 \to^*_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} t_k & = & C[t[* \leftarrow u_k]] \\ & \to^v_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} & C[t[* \leftarrow u_{k+1}]] \\ & \to^v_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} & C[t[* \leftarrow u_{k+2}]] \\ & \to^v_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} & C[t[* \leftarrow u_{k+3}]] \\ & & \vdots \end{array}$$

avec $u_k \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} u_{k+1} \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} u_{k+2} \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} u_{k+3} \cdots$

Preuve : Par le lemme 12.3.1, il existe k tel que pour tout $i > k$, $t_i \to^v_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} t_{i+1}$. On a alors $SK(t_k) = SK(t_i)$ pour tout $i \geq k$, comme on le faisait remarquer dans la définition des squelettes. $SK(t_k)$ est de la forme

$$\Diamond[* \leftarrow \bullet]\Diamond[* \leftarrow \bullet]\Diamond[* \leftarrow \bullet]\Diamond ... \Diamond[* \leftarrow \bullet]\Diamond$$

Où les symboles $\Diamond$ correspondent au contexte et ne contiennent pas de substitutions. L'arbre des dérivations de t_k étant infini (mais à branchement fini), par le principe des tiroirs, puisque ces dérivations infinies n'ont lieu que dans le corps des substitutions vides, on peut extraire une dérivation infinie qui n'a lieu que dans une seule d'entre elles. Cela nous permet de conclure. ■

Lemme 12.3.4 (Traçabilité des substitutions - 1 pas)

Soient t et u deux objets tels que $t \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} u$ et $u = C[u_1[* \leftarrow u_2]]$. Alors

1. soit $t = C'[u'_1[* \leftarrow u_2]]$
2. soit $t = C'[u'_1[* \leftarrow u'_2]]$ avec $u_2 \to u'_2$
3. soit u_1 est une commande et
   - $* = \alpha$ et $t = C[\langle \mu\alpha.u_1 | u_2 \rangle]$, ou bien
   - $* = x$ et $t = C[\langle u_2 | \tilde{\mu}x.u_1 \rangle]$

Preuve : On raisonne par induction sur t et on considère les deux cas suivants :

- La réduction a lieu à la racine. Notons tout d'abord que si $u_1[* \leftarrow u_2]$ apparaît dans un sous-terme de u qui est aussi un sous-terme de t alors pour un contexte C' et $u'_1 = u_1$ le premier point est vérifié. Ceci s'applique aussi lorsque la règle appliquée à la racine est l'une des $x\tau$ ou $\alpha\tau$. Sinon on est dans l'un des cas suivants :
  - $t = \langle \mu\alpha.u_1 | u_2 \rangle$ ou $t = \langle u_2 | \tilde{\mu}x.u_1 \rangle$ et $t \to_{mu} u_1[* \leftarrow u_2] = u$ avec le contexte C vide. Alors le troisième point est vérifié.
  - $t = \langle v|e \rangle[* \leftarrow u_2] \to_{c\tau} \langle v[* \leftarrow u_2] | e[* \leftarrow u_2] \rangle = u$ où u_1 est v ou e. Alors on prend $u'_1 = \langle v|e \rangle$, le contexte vide, et le premier point est vérifié.
  - On procède comme dans le point précédent pour les cas $t = (v \cdot e)[* \leftarrow u_2]$ et $t = (e \cdot v)[* \leftarrow u_2]$.
  - $t = (\mu\alpha.c)[* \leftarrow u_2] \to_{\mu\tau} \mu\alpha.(c[* \leftarrow u_2]) = u$. Alors on prend le contexte vide, $u'_1 = \mu\alpha.c$ et le premier point est vérifié.
  - On procède comme dans le point précédent pour les cas $t = (\tilde{\mu}x.c)[* \leftarrow u_2]$, $t = (\lambda x.v)[* \leftarrow u_2]$ et $t = (\alpha\lambda.e)[* \leftarrow u_2]$.
- La réduction est interne.

- $t = *$. Le résultat est vérifié trivialement.
- $t = \langle v|e\rangle$ avec soit $v \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} v'$ soit $e \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} e'$. On considère le premier cas, le second étant similaire. On a $u = \langle v'|e\rangle$ et il faut regarder deux cas :
  - ⋆ le sous-terme $u_1[* \leftarrow u_2]$ apparaît dans v'. Alors on utilise l'hypothèse d'induction.
  - ⋆ le sous-terme $u_1[* \leftarrow u_2]$ apparaît dans e. Alors le premier point est vérifié.
- $t = v \cdot e$ ou $t = e \cdot v$ avec soit $v \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} v'$ soit $e \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} e'$. On conclut comme dans le point précédent.
- $t = \mu\alpha.c$ ou $t = \tilde{\mu}x.c$ avec $c \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} c'$. On utilise l'hypothèse d'induction.
- $t = \lambda x.v$ avec $v \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} v'$, ou $t = \alpha\lambda.e$ avec $e \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} e'$. On utilise l'hypothèse d'induction.
- $t = t_1[* \leftarrow t_2]$ avec
  - ⋆ soit, $t_1 \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} t_1'$ et $u = t_1'[* \leftarrow t_2]$. Alors si $u_1[* \leftarrow u_2]$ apparaît dans t_1' on utilise l'hypothèse d'induction. Si il apparaît dans t_2 le premier point est vérifié trivialement. Enfin, si $u = u_1[* \leftarrow u_2]$ alors on prend pour C' le contexte vide, $u_1' = t_1$ et le premier point est vérifié.
  - ⋆ soit, $t_2 \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} t_2'$ et $u = t_1[* \leftarrow t_2']$. Alors si $u_1[* \leftarrow u_2]$ apparaît dans t_1 le premier point est vérifié trivialement. Si il apparaît dans t_2' on utilise l'hypothèse d'induction. Enfin, si $u = u_1[* \leftarrow u_2]$ alors on prend pour C' le contexte vide, $u_1' = t_1$ et $u_2' = t_2$ et le deuxième point est vérifié.

■

Ce résultat s'étend naturellement aux dérivations en plusieurs pas.

Lemme 12.3.5 (Traçabilité des substitutions)

Soient $t_1, ..., t_n$ des objets tels que, pour tout i, $t_i \to_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} t_{i+1}$ et $t_n = C[u_1[* \leftarrow u_2]]$. Alors

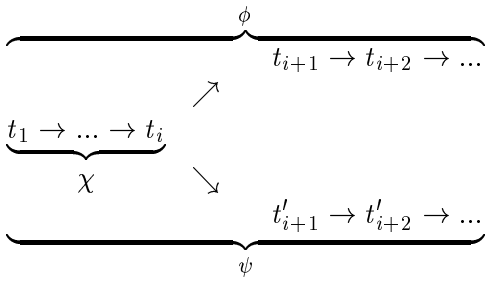
1. soit $* = \alpha$ et il existe i tel que $t_i = C'[\langle\mu\alpha.u_1'|u_2'\rangle]$ avec $u_2 \to^*_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} u_2'$.
2. soit $* = x$ et il existe i tel que $t_i = C'[\langle u_2'|\tilde{\mu}x.u_1'\rangle]$ avec $u_2 \to^*_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} u_2'$.
3. soit $t_1 = C'[u_1'[* \leftarrow u_2']]$ avec $u_2 \to^*_{\overline{\lambda\mu\tilde{\mu}}\mathtt{x}} u_2'$.

Preuve : Par récurrence sur le nombre de pas de réécriture, en utilisant le lemme 12.3.4. ■

Pour la preuve de PSN, on a besoin d'ordonner les dérivations de la façon suivante.

Définition 12.3.6 (Ordre perpétuel)

Soient ϕ et ψ deux dérivations infinies partant du même objet t_1. Soit χ le préfixe commun (éventuellement vide) à ces deux dérivations, et t_i le dernier objet de ce préfixe. Nos dérivations s'écrivent alors :



On dit que ϕ est *plus petite* que ψ dans *l'ordre perpétuel* si le redex de t_i réduit par ϕ est un sous-terme strict du redex réduit par ψ dans ce même terme.

Exemple 12.3.7

Soit les deux dérivations infinies suivantes :

$$\langle u|\tilde{\mu}y.\langle\mu\alpha.c|e\rangle\rangle \to (\langle\mu\alpha.c|e\rangle)[y \leftarrow u] \to \langle(\mu\alpha.c)[y \leftarrow u]|e[y \leftarrow u]\rangle \to \langle\mu\alpha.(c[y \leftarrow u])|e[y \leftarrow u]\rangle \to ...$$

et

$$\langle u|\tilde{\mu}y.\langle\mu\alpha.c|e\rangle\rangle \to (\langle\mu\alpha.c|e\rangle)[y \leftarrow u] \to (\langle\mu\alpha.c|e\rangle)[y \leftarrow u'] \to (\langle\mu\alpha.c|e\rangle)[y \leftarrow u''] \to ...$$

avec $u \to u' \to u'' \to ....$ Le préfixe commun est

$$\langle u|\tilde{\mu}y.\langle\mu\alpha.c|e\rangle\rangle \to (\langle\mu\alpha.c|e\rangle)[y \leftarrow u]$$

et le dernier terme du préfixe commun est $(\langle\mu\alpha.c|e\rangle)[y \leftarrow u]$. Dans ce terme, la première dérivation réduit le redex de tête, tandis que la deuxième dérivation réduit un redex dans u, qui est un sous-terme strict du redex de tête. La deuxième réduction est donc plus petite que la première dans l'ordre perpétuel.

Propriété 12.3.8 (Dérivation minimale)

Soit t un terme admettant une dérivation infinie. On peut construire une dérivation infinie minimale de t, selon l'ordre perpétuel, de la façon suivante. À chaque étape de réduction, en commençant par t, si plusieurs dérivations infinies sont possibles, alors on a un ensemble de redex R menant chacun à l'une de ces dérivations. Pour obtenir une dérivation minimale, il suffit de choisir un redex de R pour lequel aucun autre redex de R n'en est un sous-terme strict.

Voici le théorème qui énonce la propriété de PSN.

Théorème 12.3.9 (PSN : préservation de la normalisation forte)

Tout objet fortement normalisant du $\overline{\lambda}\mu\tilde{\mu}$-calcul est aussi fortement normalisant dans le $\overline{\lambda}\mu\tilde{\mu}$-calcul avec substitutions explicites. Autrement dit, pour tout objet t, on a

$$t \in \mathcal{SN}_{\overline{\lambda}\mu\tilde{\mu}} \quad \Rightarrow \quad t \in \mathcal{SN}_{\overline{\lambda}\mu\tilde{\mu}\mathtt{x}}$$

Preuve : Preuve par l'absurde. Supposons qu'il existe un terme pur t qui admette une dérivation infinie dans le $\overline{\lambda}\mu\tilde{\mu}$-calcul avec substitutions explicites. Par la propriété 12.3.8, on construit une dérivation infinie minimale de t :

$$t \to t_1 \to t_2 \to t_3 \to ... \to t_i \to ...$$

Par le lemme 12.3.1, cette dérivation infinie n'est composée que de réduction vides à partir d'un terme t_j :

$$t \to t_1 \to t_2 \to t_3 \to ... \to t_j \xrightarrow{v} t_{j+1} \xrightarrow{v} ... \xrightarrow{v} t_n \xrightarrow{v} ...$$

Par le lemme 12.3.3, on peut trouver une substitution dans le corps de laquelle prend place une infinité d'étapes de réductions (pas nécessairement consécutives) [1] :

$$t \to t_1 \to t_2 \to t_3 \to ... \to t_j = C[t'[\alpha \leftarrow u_j]] \xrightarrow{v} t_{j+1} = C[t'[\alpha \leftarrow u_{j+1}]] \xrightarrow{v} ... \xrightarrow{v} t_n \xrightarrow{v} ...$$

Par le lemme 12.3.5, on peut trouver le point de création de cette substitution (avec $t'' \to^* t'$ et $u' \to^* u_j$) :

$$t \to t_1 \to t_2 \to t_3 \to ... \to C'[\langle\mu\alpha.t''|u'\rangle] \to ... \to t_j = C[t'[\alpha \leftarrow u_j]] \xrightarrow{v} ...$$

Le redex qui constitue ce point de création a été choisi par notre dérivation minimale. Or c'est un sur-terme strict d'un redex de la dérivation infinie qui prend place dans le corps de la future substitution ($u' \to ... \to u_j \to ...$). Ceci contredit la minimalité de notre dérivation infinie. ■

[1] Dans l'exemple, on a supposé que la substitution était de la forme $[\alpha \leftarrow u_j]$, mais l'autre cas est similaire.

12.4 Preuve de normalisation forte

On applique la technique de preuve présentée dans la deuxième partie de la thèse. Pour cela, on commence par définir la fonction $Ateb$, de la façon suivante :

$$\begin{array}{lcll}
Ateb(x) & = & x & \\
Ateb(\alpha) & = & \alpha & \\
Ateb(\langle v|e\rangle) & = & \langle Ateb(v)|Ateb(e)\rangle & \\
Ateb(\lambda x.v) & = & \lambda x.Ateb(v) & \\
Ateb(\alpha\lambda.e) & = & \alpha\lambda.Ateb(e) & \\
Ateb(\mu\alpha.c) & = & \mu\alpha.Ateb(c) & \\
Ateb(\tilde{\mu}x.c) & = & \tilde{\mu}x.Ateb(c) & \\
Ateb(e \cdot v) & = & Ateb(e) \cdot Ateb(v) & \\
Ateb(v \cdot e) & = & Ateb(v) \cdot Ateb(e) & \\
 & & & \\
Ateb(c[x \leftarrow v]) & = & \langle Ateb(v)|\tilde{\mu}x.Ateb(c)\rangle & \\
Ateb(c[\alpha \leftarrow e]) & = & \langle \mu\alpha.Ateb(c)|Ateb(e)\rangle & \\
Ateb(v[x \leftarrow v']) & = & \mu\alpha.\langle \lambda x.Ateb(v)|Ateb(v') \cdot \alpha\rangle & \text{Avec } \alpha \text{ variable fraîche} \\
Ateb(v[\alpha \leftarrow e]) & = & \mu\beta.\langle \mu\alpha.\langle Ateb(v)|\beta\rangle|Ateb(e)\rangle & \text{Avec } \beta \text{ variable fraîche} \\
Ateb(e[x \leftarrow v]) & = & \tilde{\mu}y.\langle Ateb(v)|\tilde{\mu}x.\langle y|Ateb(e)\rangle\rangle & \text{Avec } y \text{ variable fraîche} \\
Ateb(e[\alpha \leftarrow e']) & = & \tilde{\mu}x.\langle Ateb(e') \cdot x|\alpha\lambda.Ateb(e)\rangle & \text{Avec } x \text{ variable fraîche}
\end{array}$$

Il est évident que pour tout t, $Ateb(t)$ ne contient pas de substitutions. Il nous faut vérifier, d'une part, que le terme obtenu est typable et, d'autre part, qu'il se réduit vers le terme d'origine.

Lemme 12.4.1

$$\Gamma \vdash t : A \quad \Rightarrow \quad \Gamma \vdash Ateb(t) : A$$

Preuve : On procède par induction sur la dérivation de typage de t. Les seuls cas non immédiats sont ceux des substitutions.

– On type $c[x \leftarrow v]$

$$\frac{c : (\Gamma, x : A \vdash \Delta) \quad \dfrac{\Gamma \vdash v : A|\Delta}{[x \leftarrow v] : (\Gamma, x : A \vdash \Delta) \Rightarrow (\Gamma \vdash \Delta)}}{c[x \leftarrow v] : (\Gamma \vdash \Delta)}$$

Par hypothèse d'induction, on a $Ateb(c) : (\Gamma, x : A \vdash \Delta)$ et $\Gamma \vdash Ateb(v) : A|\Delta$. On peut typer $Ateb(c[x \leftarrow v]) = \langle Ateb(v)|\tilde{\mu}x.Ateb(c)\rangle$ de la façon suivante

$$\frac{\Gamma \vdash Ateb(v) : A|\Delta \quad \dfrac{Ateb(c) : (\Gamma, x : A \vdash \Delta)}{\Gamma|\tilde{\mu}x.Ateb(c) : A \vdash \Delta}}{\langle Ateb(v)|\tilde{\mu}x.Ateb(c)\rangle : (\Gamma \vdash \Delta)}$$

– Le cas $c[\alpha \leftarrow e]$ est identique à celui ci-dessus par symétrie.

– On type $v[x \leftarrow v']$

$$\frac{\Gamma, x : B \vdash v : A|\Delta \quad \dfrac{\Gamma \vdash v' : B|\Delta}{[x \leftarrow v'] : (\Gamma, x : B \vdash \Delta) \Rightarrow (\Gamma \vdash \Delta)}}{\Gamma \vdash v[x \leftarrow v'] : A|\Delta}$$

Par hypothèse d'induction, on a $\Gamma, x : B \vdash Ateb(v) : A|\Delta$ et $\Gamma \vdash Ateb(v') : B|\Delta$. On peut typer $Ateb(v[x \leftarrow v']) = \mu\alpha.\langle \lambda x.Ateb(v)|Ateb(v') \cdot \alpha\rangle$ de la façon suivante

$$\dfrac{\dfrac{\dfrac{\Gamma, x : B \vdash Ateb(v) : A|\Delta}{\Gamma, x : B \vdash Ateb(v) : A|\Delta, \alpha : A}}{\Gamma \vdash \lambda x.Ateb(v) : B \to A|\Delta, \alpha : A} \quad \dfrac{\dfrac{\Gamma \vdash Ateb(v') : B|\Delta}{\Gamma \vdash Ateb(v') : B|\Delta, \alpha : A} \quad \Gamma|\alpha : A \vdash \Delta, \alpha : A}{\Gamma|Ateb(v') \cdot \alpha : B \to A \vdash \Delta, \alpha : A}}{\dfrac{\langle \lambda x.Ateb(v)|Ateb(v') \cdot \alpha\rangle : (\Gamma \vdash \Delta, \alpha : A)}{\Gamma \vdash \mu\alpha.\langle \lambda x.Ateb(v)|Ateb(v') \cdot \alpha\rangle : A|\Delta}}$$

– On type $v[\alpha \leftarrow e]$

$$\dfrac{\Gamma \vdash v : A|\Delta, \alpha : B \quad \dfrac{\Gamma \vdash e : B|\Delta}{[\alpha \leftarrow e] : (\Gamma \vdash \Delta, \alpha : B) \Rightarrow (\Gamma \vdash \Delta)}}{\Gamma \vdash v[\alpha \leftarrow e] : A|\Delta}$$

Par hypothèse d'induction, on a $\Gamma \vdash Ateb(v) : A|\Delta, \alpha : B$ et $\Gamma \vdash Ateb(e) : B|\Delta$. On peut typer $Ateb(v[\alpha \leftarrow e]) = \mu\beta.\langle \mu\alpha.\langle Ateb(v)|\beta\rangle|Ateb(e)\rangle$ de la façon suivante

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash Ateb(v) : A|\Delta, \alpha : B}{\Gamma \vdash Ateb(v) : A|\Delta, \beta : A, \alpha : B} \quad \Gamma|\beta : A \vdash \Delta, \beta : A, \alpha : B}{\langle Ateb(v)|\beta\rangle : (\Gamma \vdash \Delta, \beta : A, \alpha : B)}}{\Gamma \vdash \mu\alpha.\langle Ateb(v)|\beta\rangle : B|\Delta, \beta : A} \quad \dfrac{\Gamma \vdash Ateb(e) : B|\Delta}{\Gamma \vdash Ateb(e) : B|\Delta, \beta : A}}{\langle \mu\alpha.\langle Ateb(v)|\beta\rangle|Ateb(e)\rangle : (\Gamma \vdash \Delta, \beta : A)}}{\Gamma \vdash \mu\beta.\langle \mu\alpha.\langle Ateb(v)|\beta\rangle|Ateb(e)\rangle : A|\Delta}$$

– Les cas pour $e[* \leftarrow t]$ sont identiques à ceux ci-dessus par symétrie.

■

Lemme 12.4.2

$$Ateb(t) \to^* t$$

Preuve : On procède par induction sur t. Les seuls cas non immédiats sont ceux des substitutions.
– On a $Ateb(c[x \leftarrow v]) = \langle Ateb(v)|\tilde{\mu}x.Ateb(c)\rangle$ et

$$\langle Ateb(v)|\tilde{\mu}x.Ateb(c)\rangle \to_\mu Ateb(c)[x \leftarrow Ateb(v)]$$

On conclut par hypothèse d'induction.
– Le cas $c[\alpha \leftarrow e]$ est identique à celui ci-dessus par symétrie.
– On a $Ateb(v[x \leftarrow v']) = \mu\alpha.\langle \lambda x.Ateb(v)|Ateb(v') \cdot \alpha\rangle$ et

$$\begin{array}{c} \mu\alpha.\langle \lambda x.Ateb(v)|Ateb(v') \cdot \alpha\rangle \\ \downarrow \beta \\ \mu\alpha.\langle Ateb(v')|\tilde{\mu}x.\langle Ateb(v)|\alpha\rangle\rangle \\ \downarrow \tilde{\mu} \\ \mu\alpha.(\langle Ateb(v)|\alpha\rangle[x \leftarrow Ateb(v')]) \\ \downarrow c\tau \\ \mu\alpha.\langle Ateb(v)[x \leftarrow Ateb(v')]|\alpha[x \leftarrow Ateb(v')]\rangle \\ \downarrow \alpha\tau 2 \\ \mu\alpha.\langle Ateb(v)[x \leftarrow Ateb(v')]|\alpha\rangle \\ \downarrow sv \\ Ateb(v)[x \leftarrow Ateb(v')] \end{array}$$

On conclut par hypothèse d'induction.

– On a $Ateb(v[\alpha \leftarrow e]) = \mu\beta.\langle\mu\alpha.\langle Ateb(v)|\beta\rangle|Ateb(e)\rangle$ et

$$\begin{array}{c} \mu\beta.\langle\mu\alpha.\langle Ateb(v)|\beta\rangle|Ateb(e)\rangle \\ \downarrow \mu \\ \mu\beta.(\langle Ateb(v)|\beta\rangle[\alpha \leftarrow Ateb(e)]) \\ \downarrow c\tau \\ \mu\beta.\langle Ateb(v)[\alpha \leftarrow Ateb(e)]|\beta[\alpha \leftarrow Ateb(e)]\rangle \\ \downarrow \alpha\tau 2 \\ \mu\beta.\langle Ateb(v)[\alpha \leftarrow Ateb(e)]|\beta\rangle \\ \downarrow sv \\ Ateb(v)[\alpha \leftarrow Ateb(e)] \end{array}$$

On conclut par hypothèse d'induction.

– Les cas pour $e[* \leftarrow t]$ sont identiques à ceux ci-dessus par symétrie.

■

On peut à présent collecter nos résultats afin de prouver le théorème principal de cette partie.

Théorème 12.4.3

Le $\overline{\lambda}\mu\tilde{\mu}\mathbf{x}$-calcul est fortement normalisant.

Preuve : Par le théorème 11.4.2 (SN du calcul pur), le théorème 12.3.9 (PSN) et le théorème 4.4.1 (PSN implique SN). ■

Quatrième partie

Le λ_{ws}-calcul avec noms

Comme on l'a vu dans le chapitre 2, le λ_{ws}-calcul [26, 41], introduit une notion d'étiquette en plus des substitutions explicites. Ces étiquettes sont des affaiblissements explicites, c'est-à-dire que dans un cadre typé, elles correspondent à l'application d'une règle d'affaiblissement de la logique sous-jacente. Grâce à cette notion, le λ_{ws}-calcul vérifie toutes les propriétés souhaitées.

Dans [18], nous avons prouvé la normalisation forte du λ_{ws}-calcul simplement typé en traduisant ses termes dans les réseaux de preuves. Lors de cet étude, nous nous sommes aperçus que les réseaux de preuves se passaient d'un certain nombre de détails de λ_{ws}. En particulier, le λ_{ws}-calcul est un calcul avec indices de De Bruijn, or les réseaux de preuves ne se soucient ni de la notion de variable, ni même d'un ordre dans l'environnement de typage des termes (voir chapitres 2 et 3). Nous avons donc proposé un calcul avec noms de variable qui se rapprochait davantage des réseaux de preuves tout en conservant la notion d'étiquette. La figure 12.1 rappelle brièvement la grammaire des termes et les règles de réduction.

Les termes :

$$t ::= x \mid (t\ t) \mid \lambda x.t \mid \Gamma t \mid t[x, t, \Gamma, \Gamma]$$

Les règles de réduction :

$$
\begin{array}{lrcll}
(b_1) & (\lambda x.t)u & \to & t[x, u, \emptyset, \emptyset] & \\
(b_2) & (\Delta(\lambda x.t))u & \to & t[x, u, \emptyset, \Delta] & \\
 & & & & \\
(f) & (\lambda y.t)[x, u, \Gamma, \Delta] & \to & \lambda y.t[x, u, \Gamma \cup \{y\}, \Delta] & \\
(a) & (t\ u)[x, v, \Gamma, \Delta] & \to & (t[x, v, \Gamma, \Delta]\ u[x, v, \Gamma, \Delta]) & \\
(e_1) & (\Lambda t)[x, u, \Gamma, \Delta] & \to & (\Delta \cup (\Lambda \setminus \{x\}))t & x \in \Lambda \\
(e_2) & (\Lambda t)[x, u, \Gamma, \Delta] & \to & (\Gamma \cap \Lambda)t[x, u, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] & x \notin \Lambda \\
(n_1) & y[x, t, \Gamma, \Delta] & \to & \Delta y & x \neq y \\
(n_2) & x[x, t, \Gamma, \Delta] & \to & \Gamma t & \\
(c_1) & t[y, u, \Lambda, \Phi][x, v, \Gamma, \Delta] & \to & & x \in \Phi \setminus \Lambda \\
 & & & t[y, u[x, v, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})] & \\
(c_2) & t[y, u, \Lambda, \Phi][x, v, \Gamma, \Delta] & \to & t[x, v, (\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)] & \\
 & & & [y, u[x, v, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi] & x \notin \Phi \cup \Lambda \\
 & & & & \\
(d) & \Gamma\Delta t & \to & (\Gamma \cup \Delta)t &
\end{array}
$$

FIG. 12.1 – Définition du λ_{wsn}-calcul de [18]

Lorsque nous avons introduit ce calcul, notre souci était de conserver une grande proximité vis-à-vis du λ_{ws}-calcul. Cependant, il nous est vite apparu que ce nouveau calcul possède une souplesse qui va nous permettre de nous rapprocher encore plus des réseaux de preuves, et peut-être même de se passer de l'α-équivalence qui est en général de rigueur dans les calculs nommés. Au moment de la rédaction de cette thèse, une partie du travail concernant ce calcul est inachevé. Les preuves de normalisation forte et de confluence du calcul des substitutions, de confluence sur les termes ouverts, ainsi que celle de préservation de la normalisation forte sont encore au stade de développement.

Pour commencer, nous regardons ce que les différentes définitions du λ_{wsn}-calcul nous apportent comme possibilités nouvelles (chapitre 13). Les différences entre ces définitions résident dans les règles de réduction ; la syntaxe des termes reste inchangée. Ensuite, puisque nous avons des affaiblissements explicites dans nos termes, nous devons, comme pour λ_{ws} [26, 41], nous baser sur un calcul pur qui en contient aussi. Pour cela, nous définissons le λ_{wn}-calcul dans le chapitre 14. Le typage de notre calcul est présenté chapitre 15 et on y donne la preuve de préservation du typage par réduction. Pour pouvoir établir la normalisation forte du calcul simplement typé, nous avons besoin d'enrichir l'élimination des

coupures de la logique linéaire avec une règle supplémentaire : le chapitre 16 prouve la terminaison de cette nouvelle notion de réécriture. Enfin, le chapitre 17 donne la preuve de normalisation forte par traduction dans les réseaux de preuves enrichis et le chapitre 18 explore des possibilités d'extension futures pour ce calcul.

Chapitre 13

Définition du λ_{wsn}-calcul

Ce chapitre est consacré à la définition du calcul. On commence par en donner les termes. Puis on s'intéresse à de nouvelles règles de réduction et équivalences, et on obtient deux formulations de notre système. On regarde ensuite ce qui se passe du côté de l'α-équivalence, pour se rendre compte qu'on pourrait peut-être s'en passer, ce qui nous conduit à proposer deux futures formulations. Enfin, on effectue une simulation de l'une de nos formulations dans l'autre, afin de fixer notre choix sur l'une d'elles.

13.1 Définition des termes

Soit Var un ensemble infini de variables, les termes sont construits partir de la grammaire suivante :

$$t \ ::= \ x \ | \ (t \ t) \ | \ \lambda x.t \ | \ t[x, t, \Gamma, \Gamma] \ | \ \Gamma t$$

où $x \in Var$, et $\Gamma \subset Var$. On notera Λ_{wsn} l'ensemble des termes du λ_{wsn}-calcul.

Remarque 13.1.1

- Dans le terme Γt, le Γ correspond à une étiquette du λ_{ws}-calcul. C'est un *affaiblissement explicite* qui "protège" t, indiquant que les variables présentes dans Γ n'apparaissent pas dans le terme (modulo α-conversion, pour l'instant). Par exemple :

  $$\{x, z\}t$$

  est un terme pour lequel on sait que ni x, ni z ne sont libres dans t. On comprend bien l'intérêt d'une telle information. Si le terme ci-dessus vient à rencontrer une substitution sur la variable x ou z, on peut éliminer directement la substitution. L'étiquette est un "délieur", il indique la fin de liaison d'une variable liée par un λ (voir section 13.3).
- Dans le terme $t[x, u, \Gamma, \Delta]$, Γ et Δ sont aussi des affaiblissements. Le premier, Γ, protège le terme substituant u, et le deuxième, Δ, protège le terme substitué t. Par exemple, dans le terme

  $$t[x, u, \{y, z\}, \{z, w\}]$$

  on sait que la variable y n'est pas libre dans u, que w n'est pas libre dans t et que z n'est libre ni dans u ni dans t.

Plus formellement, on définit les variables libres d'un terme de la façon suivante :

$$\begin{array}{lcl} FV(x) & = & x \\ FV(t\ u) & = & FV(t) \cup FV(u) \\ FV(\lambda x.t) & = & FV(t) \setminus \{x\} \\ FV(\Gamma t) & = & FV(t) \setminus \Gamma \\ FV(t[x, u, \Gamma, \Delta] & = & ((FV(t) \setminus \{x\}) \setminus \Delta) \cup (FV(u) \setminus \Gamma) \end{array}$$

13.2 Les nouvelles règles

La plus grande liberté d'expression des termes de λ_{wsn} nous permet de modifier les règles d'une part pour faire une règle générique d'introduction de substitution, et d'autre part pour permettre plus d'interaction entre les substitutions lors de la composition. Cet assouplissement des règles permet d'une part des rendre le calcul plus concis et plus complet, et d'autre part de se rapprocher encore plus de l'élimination des coupures des réseaux de preuves.

13.2.1 L'unique règle b

Dans [18] on présentait, pour la création des substitutions, les deux règles suivantes :

$$\begin{array}{lrcl} (b_1) & (\lambda x.t)\ u & \to & t[x,u,\emptyset,\emptyset] \\ (b_2) & (\Delta(\lambda x.t))\ u & \to & t[x,u,\emptyset,\Delta] \end{array}$$

en faisant judicieusement remarquer que la règle b_1 est en fait un cas particulier de la règle b_2 pour $\Delta = \emptyset$. Il était dès lors naturel de vouloir faire une règle encore plus générale :

$$(b) \quad (\Delta(\lambda x.t))\ (\Gamma u) \quad \to \quad t[x,u,\Gamma,\Delta]$$

De cette règle on peut déduire les règles b_1, b_2 et la règle b_3 qu'on aurait pu ajouter : $(\lambda x.t)\ (\Gamma u) \to t[x,u,\Gamma,\emptyset]$. Le lecteur attentif aura remarqué que, pour l'instant, il n'est même pas possible de réduire le terme $(\lambda x.x)\ y$ avec la nouvelle règle : en effet, dans ce terme n'apparaît ni Γ, ni Δ. Pour que cette règle soit utilisable, nous avons besoin d'une équivalence qui ajoute des étiquettes quand il le faut. On appelle $\sim_\emptyset$ cette équivalence, et voici sa définition :

$$\forall t \in \Lambda_{wsn} \quad t \quad \sim_\emptyset \quad \emptyset t$$

Cependant, un problème apparaît si on introduit cette équivalence. Il devient possible d'effectuer des dérivations infinies en utilisant, par exemple, les règles e_2 et d :

$$\begin{array}{lllllll} t[x,u,\Gamma,\Delta] & \sim_\emptyset & (\emptyset t)[x,u,\Gamma,\Delta] & \to_{e_2} & \emptyset(t[x,u,\Gamma,\Delta]) & \sim_\emptyset & t[x,u,\Gamma,\Delta] \\ \Gamma t & \sim_\emptyset & \emptyset\Gamma t & \to_d & \Gamma t & & \end{array}$$

Deux choix s'offrent alors, soit conserver la règle b nouvellement créée ainsi que l'équivalence $\sim_\emptyset$, mais il faut alors transformer e_2 et d en équivalences, soit conserver les quatre règles b_1, b_2, b_3 et b_4 (qui s'énonce comme b). Nous discuterons de ce choix plus loin.

13.2.2 Les trois règles et l'équivalence pour la composition

Pour mimer les réductions de λ_{ws}, on proposait les deux règles de composition suivantes :

$$\begin{array}{lrcll} (c_1) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & & x \in \Phi \\ & & & t[y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})] & \\ (c_2) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[x,v,(\Gamma \setminus \Phi) \cup y, \Delta \cup (\Phi \setminus \Gamma)] & \\ & & & [y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi] & x \notin \Phi \cup \Lambda \end{array}$$

En outre, la définition des termes nous imposait, pour mimer le λ_{ws}-calcul, que $\Lambda \cap \Phi = \emptyset$ et $\Gamma \cap \Phi = \emptyset$.

Pour comprendre ces règles de composition, il faut se demander ce que peut devenir le terme

$$t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta].$$

On souhaite pouvoir continuer à propager la substitution $[x,v,\Gamma,\Delta]$ sans être obligé, comme dans le $\lambda\mathbf{x}$-calcul, de propager d'abord $[y,u,\Lambda,\Phi]$ dans t. Dans notre ancienne version de λ_{wsn}, comme dans λ_{ws}, deux cas étaient distingués :

- soit $x \in \Phi$, auquel cas on sait que la substitution n'a rien à faire dans t, on la propage à l'intérieur de l'autre pour la faire atteindre u (on sait que $x \notin \Lambda$ grâce à notre restriction), c'est la règle c_1,

- soit $x \notin \Phi$ et $x \notin \Lambda$, auquel cas la substitution doit être propagée à la fois dans t et dans u, c'est la règle c_2.

Puisqu'on souhaite donner plus de souplesse à notre calcul, on relâche les contraintes $\Lambda \cap \Phi = \emptyset$ et $\Gamma \cap \Phi = \emptyset$. Il y a alors quatre cas pour la composition. Les deux premiers sont identiques à ceux ci-dessus, pourvu que l'on ajoute la condition $x \notin \Lambda$ pour la règle c_1. Il nous reste deux cas à voir.

- Tout d'abord, supposons que x apparaisse à la fois dans Φ et dans Λ. La substitution n'a alors le droit d'aller nulle part, on peut l'effacer.

$$(c_3) \quad t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] \quad \to \quad t[y,u,(\Lambda \setminus \{x\}) \cup \Delta, (\Phi \setminus \{x\}) \cup \Delta] \quad x \in \Phi \cap \Lambda$$

Cela nous donne les trois règles de composition de notre calcul :

$$\begin{array}{lll}
(c_1) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] \quad \to & x \in \Phi \setminus \Lambda \\
& \quad t[y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})] & \\
(c_2) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] \quad \to \quad t[x,v,(\Gamma \setminus \Phi) \cup y, \Delta \cup (\Phi \setminus \Gamma)] & \\
& \qquad\qquad [y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi] & x \notin \Phi \cup \Lambda \\
(c_3) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] \quad \to \quad t[y,u,(\Lambda \setminus \{x\}) \cup \Delta, (\Phi \setminus \{x\}) \cup \Delta] & x \in \Phi \cap \Lambda
\end{array}$$

- Le dernier cas est un peu à part. En effet, supposons que x soit dans Λ et pas dans Φ, on pourrait être tenté d'écrire une règle comme :

$$\begin{array}{ll}
(c_4) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] \quad \to \qquad\qquad x \in \Lambda \setminus \Phi \\
& \quad t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y,u,\Delta \cup (\Lambda \setminus \{x\}), \Gamma \cap \Phi]
\end{array}$$

Mais comme on raisonne modulo α-conversion (pour l'instant), y n'est pas dans $\Delta \cup (\Phi \setminus \Gamma)$, et les conditions $y \in (\Gamma \setminus \Phi) \cup \{y\}$ et $y \notin \Delta \cup (\Phi \setminus \Gamma)$ sont vérifiées. On peut alors appliquer de nouveau la règle de la façon suivante :

$$\begin{array}{l}
t_1 = t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y,u,\Delta \cup (\Lambda \setminus \{x\}), \Gamma \cap \Phi] \quad \to \\
t[y,u,(\Delta \cup (\Lambda \setminus \{x\})) \setminus (\Delta \cup (\Phi \setminus \Gamma)) \cup \{x\}, (\Gamma \cap \Phi) \cup ((\Delta \cup (\Phi \setminus \Gamma)) \setminus (\Delta \cup (\Lambda \setminus \{x\})))] \\
\qquad [x,v,(\Gamma \cap \Phi) \cup (((\Gamma \setminus \Phi) \cup \{y\}) \setminus \{y\}), (\Delta \cup (\Lambda \setminus \{x\})) \cap (\Delta \cup (\Phi \setminus \Gamma))] = t_2
\end{array}$$

On pose $\Lambda' = \Lambda \setminus \{x\}$ et on calcule les égalités suivantes [1] :

1. $(\Delta \cup \Lambda') \setminus (\Delta \cup (\Phi \setminus \Gamma)) \cup \{x\}$
   $= (\Delta \setminus (\Delta \cup (\Phi \setminus \Gamma))) \cup (\Lambda' \setminus (\Delta \cup (\Phi \setminus \Gamma))) \cup \{x\}$ $(a.)$
   $= (\Lambda' \setminus (\Delta \cup (\Phi \setminus \Gamma))) \cup \{x\}$ $(b.)$
   $= ((\Lambda \setminus \{x\}) \setminus (\Phi \setminus \Gamma)) \cup \{x\}$ $\Delta \cap \Lambda = \emptyset$ et $(e.)$
   $= ((\Lambda \setminus \{x\}) \cup \{x\}) \setminus (\Phi \setminus \Gamma)$ $x \notin \Phi$ et $(f.)$
   $= \Lambda \setminus (\Phi \setminus \Gamma)$ $(h.)$

2. $(\Gamma \cap \Phi) \cup ((\Delta \cup (\Phi \setminus \Gamma)) \setminus (\Delta \cup \Lambda'))$
   $= (\Gamma \cap \Phi) \cup (\Delta \setminus (\Delta \cup \Lambda')) \cup ((\Phi \setminus \Gamma) \setminus (\Delta \cup \Lambda'))$ $(a.)$
   $= (\Gamma \cap \Phi) \cup ((\Phi \setminus \Gamma) \setminus (\Delta \cup \Lambda'))$ $(b.)$
   $= (\Gamma \cap \Phi) \cup ((\Phi \setminus \Gamma) \setminus \Lambda')$ $\Delta \cap \Phi = \emptyset$ et $(e.)$
   $= \Phi \setminus (\Lambda' \setminus \Gamma)$ $(i.)$

3. $(\Gamma \cap \Phi) \cup (((\Gamma \setminus \Phi) \cup \{y\}) \setminus \{y\})$ $y \notin \Gamma$ et $(j.)$
   $= (\Gamma \cap \Phi) \cup (\Gamma \setminus \Phi)$ $(k.)$
   $= \Gamma$

4. $(\Delta \cup \Lambda') \cap (\Delta \cup (\Phi \setminus \Gamma))$
   $= (\Delta \cap (\Delta \cup (\Phi \setminus \Gamma))) \cup (\Lambda' \cap (\Delta \cup (\Phi \setminus \Gamma)))$
   $= \Delta \cup (\Lambda' \cap (\Delta \cup (\Phi \setminus \Gamma)))$
   $= \Delta \cup ((\Lambda' \cap \Delta) \cup (\Lambda' \cap (\Phi \setminus \Gamma)))$
   $= \Delta \cup (\Lambda' \cap (\Phi \setminus \Gamma))$ $\Delta \cap \Lambda = \emptyset$
   $= \Delta \cup ((\Phi \cap \Lambda') \setminus \Gamma)$ $(l.)$

On a alors :

$$t_2 = t[y, u, \Lambda \setminus (\Phi \setminus \Gamma), \Phi \setminus (\Lambda' \setminus \Gamma)][x, v, \Gamma, \Delta \cup ((\Phi \cap \Lambda') \setminus \Gamma)]$$

De $x \in \Lambda \setminus \Phi$ on déduit $x \in \Lambda \setminus (\Phi \setminus \Gamma)$ et $x \notin \Phi \setminus (\Lambda' \setminus \Gamma)$. On peut encore appliquer la même règle :

$$t_2 \to t_3 = t[x, v, (\Gamma \setminus (\Phi \setminus (\Lambda' \setminus \Gamma))) \cup \{y\}, \Delta \cup ((\Phi \cap \Lambda') \setminus \Gamma) \cup ((\Phi \setminus (\Lambda' \setminus \Gamma)) \setminus \Gamma)]$$
$$[y, u, \Delta \cup ((\Phi \cap \Lambda') \setminus \Gamma) \cup ((\Lambda \setminus (\Phi \setminus \Gamma)) \setminus \{x\}), \Gamma \cap (\Phi \setminus (\Lambda' \setminus \Gamma))]$$

On calcule les égalités suivantes :

1. $(\Gamma \setminus (\Phi \setminus (\Lambda' \setminus \Gamma))) \cup \{y\}$
   $= (\Gamma \setminus \Phi) \cup \{y\}$ $(m.)$

2. $\Delta \cup ((\Phi \cap \Lambda') \setminus \Gamma) \cup ((\Phi \setminus (\Lambda' \setminus \Gamma)) \setminus \Gamma)$
   $= \Delta \cup ((\Phi \cap \Lambda') \setminus \Gamma) \cup ((\Phi \setminus \Lambda') \setminus \Gamma)$ $(o.)$
   $= \Delta \cup ((\Phi \cap \Lambda') \cup (\Phi \setminus \Lambda'))) \setminus \Gamma$ $(a.)$
   $= \Delta \cup (\Phi \setminus \Gamma)$ $(k.)$

3. $\Delta \cup ((\Phi \cap \Lambda') \setminus \Gamma) \cup ((\Lambda \setminus (\Phi \setminus \Gamma)) \setminus \{x\})$
   $= \Delta \cup ((\Phi \cap \Lambda') \setminus \Gamma) \cup ((\Lambda \setminus \{x\}) \setminus (\Phi \setminus \Gamma))$ $(p.)$
   $= \Delta \cup ((\Phi \cap \Lambda') \setminus \Gamma) \cup (\Lambda' \setminus (\Phi \setminus \Gamma))$
   $= \Delta \cup \Lambda'$ $(q.)$

4. $\Gamma \cap (\Phi \setminus (\Lambda' \setminus \Gamma))$
   $= \Gamma \cap \Phi$ $(r.)$

[1] Dans ces calculs, ainsi que dans d'autres par la suite, les lettres entre parenthèses font référence à des petites propriétés du calcul des ensembles qui sont rappelées en annexe A. L'annotation indique que la propriété correspondante a été utilisée pour passer de la ligne précédente du calcul à la ligne sur laquelle elle figure.

On a obtenu $t_3 = t_1$, ce qui nous permet de poursuivre la réduction de façon infinie : la règle c_4 rend le système non terminant. Pour pouvoir l'utiliser tout de même, nous devons introduire cette réécriture, non pas comme une règle, mais comme une équivalence. Elle devient l'équivalence $\sim_{c_4}$ qui s'énonce comme suit :

$$\begin{array}{ll} t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \sim_{c_4} \qquad\qquad\qquad\qquad\qquad\qquad x \in \Lambda \setminus \Phi \\ t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y,u,\Delta \cup (\Lambda \setminus \{x\}), \Gamma \cap \Phi] & \end{array}$$

13.2.3 Le choix entre équivalences et règles

Le problème évoqué ci-dessus de l'introduction de l'équivalence $\sim_\emptyset$ nous amène à choisir entre deux calculs : l'un avec beaucoup de règles, et l'autre avec beaucoup d'équivalences. Le choix n'est pas évident car les deux options ont leurs avantages et leurs inconvénients respectifs.

- Si l'on choisit un calcul avec beaucoup d'équivalence, cela nous donne le système présenté figure 13.1. Celui-ci est très proche des réseaux de preuves dans la mesure où les équivalences correspondent à des égalités pour l'élimination des coupures (voir chapitre 17 et cinquième partie).
- Le système présenté figure 13.2 est plus satisfaisant du point de vue du calcul car il indique clairement comment arriver aux formes normales. En contrepartie, nous sommes obligés de donner les quatre règles b_1, b_2, b_3 et b_4.

Comme on le verra plus tard, c'est le système avec équivalences qu'on choisira d'étudier.

13.3 α-équivalence ?

L'α-équivalence sert à empêcher la capture intempestive de variables. L'exemple le plus fréquemment montré est le suivant. Lors de l'application de la règle *Lambda* (du $\lambda\mathtt{x}$-calcul) :

$$(\lambda x.t)[y \leftarrow u] \qquad \rightarrow \qquad \lambda x.(t[y \leftarrow u])$$

les variables x de u sont capturées par le λ traversé. Pour se prémunir contre cela, on dit qu'on travaille modulo α-équivalence, ce qui permet de transformer le terme $\lambda x.t$ en $\lambda z.t\{x \leftarrow z\}$, avec z n'apparaissant ni dans t, ni dans u. Dans notre système, la règle f est la suivante :

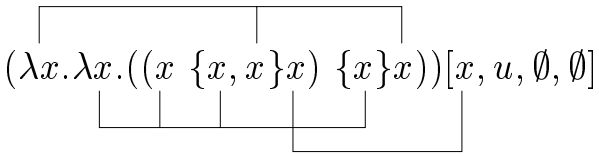
$$(\lambda x.t)[y,u,\Gamma,\Delta] \qquad \rightarrow \qquad \lambda x.(t[y,u,\Gamma \cup \{x\},\Delta])$$

L'étiquette étant un délieur, le problème évoqué ci-dessus n'apparaît pas puisqu'on indique explicitement que x n'est pas libre dans u. On peut alors se demander si l'α-équivalence est encore nécessaire. Voici quelques éléments de réponse, qui ne nous permettront cependant pas de conclure.

Sans α-équivalence, les variables peuvent apparaître plusieurs fois dans les étiquettes. Nos étiquettes vont donc être des multi-ensembles. Regardons le terme suivant

$$(\lambda x.\lambda x.((x\ \{x,x\}x)\ \{x\}x))[x,u,\emptyset,\emptyset]$$

Voici le dessin indiquant la liaison entre les différentes variables et leurs lieurs :



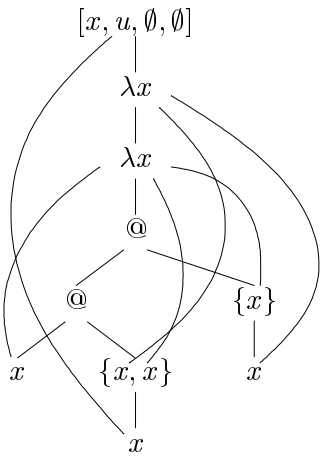
C'est peut-être plus clair si on regarde le terme sous forme d'arbre :

$$
\begin{array}{lrcll}
(b) & (\Delta(\lambda x.t))(\Gamma u) & \to & t[x,u,\Gamma,\Delta] & \\
\\
(a) & (t\ u)[x,v,\Gamma,\Delta] & \to & (t[x,v,\Gamma,\Delta]\ u[x,v,\Gamma,\Delta]) & \\
(e_1) & (\Lambda t)[x,u,\Gamma,\Delta] & \to & (\Delta \cup (\Lambda \setminus \{x\}))t & x \in \Lambda \\
(n_1) & y[x,t,\Gamma,\Delta] & \to & \Delta y & x \neq y \\
(n_2) & x[x,t,\Gamma,\Delta] & \to & \Gamma t & \\
(c_1) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & & x \in \Phi \setminus \Lambda \\
 & & & t[y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})] & \\
(c_2) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)] & \\
 & & & [y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi] & x \notin \Phi \cup \Lambda \\
(c_3) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[y,u,(\Lambda \setminus \{x\}) \cup \Delta, (\Phi \setminus \{x\}) \cup \Delta] & x \in \Phi \cap \Lambda \\
\\
(f) & (\lambda y.t)[x,u,\Gamma,\Delta] & \sim & \lambda y.t[x,u,\Gamma \cup \{y\}, \Delta] & y \notin \Delta \\
(e_2) & (\Lambda t)[x,u,\Gamma,\Delta] & \sim & (\Gamma \cap \Lambda)t[x,u,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] & x \notin \Lambda \\
(d) & \Gamma \Delta t & \sim & (\Gamma \cup \Delta)t & \\
(\emptyset) & \emptyset t & \sim & t & \\
(c_4) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \sim & & x \in \Lambda \setminus \Phi \\
 & t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y,u,\Delta \cup (\Lambda \setminus \{x\}), \Gamma \cap \Phi] & & &
\end{array}
$$

FIG. 13.1 – Système avec équivalences

$$
\begin{array}{lrcll}
(b_1) & (\lambda x.t)\ u & \to & t[x,u,\emptyset,\emptyset] & \\
(b_2) & (\Delta(\lambda x.t))\ u & \to & t[x,u,\emptyset,\Delta] & \\
(b_3) & (\lambda x.t)\ (\Gamma u) & \to & t[x,u,\Gamma,\emptyset] & \\
(b_4) & (\Delta(\lambda x.t))\ (\Gamma u) & \to & t[x,u,\Gamma,\Delta] & \\
\\
(f) & (\lambda y.t)[x,u,\Gamma,\Delta] & \to & \lambda y.t[x,u,\Gamma \cup \{y\}, \Delta] & \\
(a) & (t\ u)[x,v,\Gamma,\Delta] & \to & (t[x,v,\Gamma,\Delta]\ u[x,v,\Gamma,\Delta]) & \\
(e_1) & (\Lambda t)[x,u,\Gamma,\Delta] & \to & (\Delta \cup (\Lambda \setminus \{x\}))t & x \in \Lambda \\
(e_2) & (\Lambda t)[x,u,\Gamma,\Delta] & \to & (\Gamma \cap \Lambda)t[x,u,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] & x \notin \Lambda \\
(n_1) & y[x,t,\Gamma,\Delta] & \to & \Delta y & x \neq y \\
(n_2) & x[x,t,\Gamma,\Delta] & \to & \Gamma t & \\
(c_1) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & & x \in \Phi \setminus \Lambda \\
 & & & t[y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})] & \\
(c_2) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)] & \\
 & & & [y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi] & x \notin \Phi \cup \Lambda \\
(c_3) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[y,u,(\Lambda \setminus \{x\}) \cup \Delta, (\Phi \setminus \{x\}) \cup \Delta] & x \in \Phi \cap \Lambda \\
(d) & \Gamma \Delta t & \to & (\Gamma \cup \Delta)t & \\
(\emptyset) & \emptyset t & \to & t & \\
\\
(c_4) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \sim & & x \in \Lambda \setminus \Phi \\
 & t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y,u,\Delta \cup (\Lambda \setminus \{x\}), \Gamma \cap \Phi] & & &
\end{array}
$$

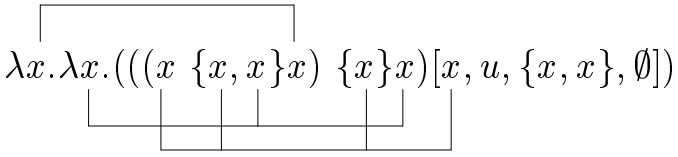
FIG. 13.2 – Système avec règles



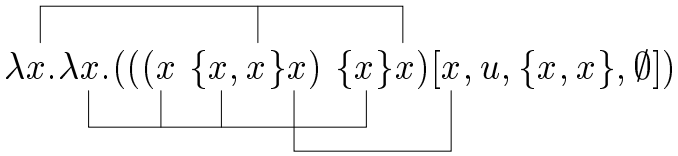
Il se réduit de la façon suivante :

$$\begin{array}{c}(\lambda x.\lambda x.((x\ \{x,x\}x)\ \{x\}x))[x,u,\emptyset,\emptyset]\\ \downarrow f\\ \lambda x.((\lambda x.((x\ \{x,x\}x)\ \{x\}x))[x,u,\{x\},\emptyset])\\ \downarrow f\\ \lambda x.\lambda x.(((x\ \{x,x\}x)\ \{x\}x)[x,u,\{x,x\},\emptyset])\end{array}$$

Si on regarde naïvement la liaison entre les différentes variables et leurs lieurs, on voit le dessin suivant :



Cette constatation suffirait à montrer qu'un tel système est incorrect. Cependant, on peut retrouver la "bonne" liaison en utilisant l'information stockée dans la substitution. Puisqu'on travaille avec des multi-ensembles, le nombre d'occurrence d'une variable dans l'étiquette protégeant le substituant nous indique le nombre de lieurs de cette variable qu'on a traversé. Comme pour les indices de De Bruijn, cette information va nous permettre de relier les variables correctement. Il suffira de considérer qu'une substitution $t[x, u, \Gamma, \Delta]$ lie les occurrences de x dans t qui sont déliées autant de fois que le nombre d'occurrences de x dans Γ. Cela nous donne le résultat attendu :



Pour obtenir un système correct, il suffit de prendre en compte cette information dans la définition des règles. Il nous faut alors convertir les opérations sur les ensembles par des opérations sur les multi-ensembles. Les figures 13.3 et 13.4 présentent une ébauche de ce que pourrait être de tels systèmes. Dans ceux-ci, nous utilisons les opérateurs suivants sur les multi-ensembles :

- Nous avons deux opérateurs pour l'union. Le premier est l'union multi-ensembliste : $\{x, x, y, z\} \sqcup \{u, v, x, z\} = \{x, x, x, u, v, y, z, z\}$ et l'autre est l'union habituelle, mais sur des multi-ensembles, c'est-à-dire qu'elle prend, pour chaque variable, le maximum du nombre d'apparition de celle-ci dans les deux ensembles : $\{x, x, y, z\} \cup \{u, v, x, z\} = \{x, x, u, v, y, z\}$.
- L'intersection multi-ensembliste : $\{x, x, x, y, z\} \sqcap \{u, v, x, x, z\} = \{x, x, z\}$.
- La différence symétrique multi-ensembliste : $\{x, x, x, y, z\} \setminus\!\setminus \{u, v, x, z\} = \{x, x, y\}$.

$$
\begin{array}{lrcll}
(b) & (\Delta(\lambda x.t))(\Gamma u) & \to & t[x,u,\Gamma,\Delta] & \\
 & & & & \\
(a) & (t\ u)[x,v,\Gamma,\Delta] & \to & (t[x,v,\Gamma,\Delta]\ u[x,v,\Gamma,\Delta]) & \\
(e_1) & (\Lambda t)[x,u,\Gamma,\Delta] & \to & (\Delta \sqcup (\Lambda \setminus\setminus \{x\}))t & x \in \Lambda \setminus\setminus \Gamma \\
(n_1) & y[x,t,\Gamma,\Delta] & \to & \Delta y & x \neq y \text{ or } y \in \Gamma \\
(n_2) & x[x,t,\Gamma,\Delta] & \to & \Gamma t & \\
(c_1) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & & x \in \Phi \setminus\setminus \Gamma \text{ et } x \notin \Lambda \setminus\setminus \Gamma \\
 & \multicolumn{3}{r}{t[y,u[x,v,\Gamma \setminus\setminus \Lambda, \Delta \sqcup (\Lambda \setminus\setminus \Gamma)], \Lambda \sqcap \Gamma, \Delta \sqcup (\Phi \setminus\setminus \{x\})]} & \\
(c_2) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[x,v,(\Gamma \setminus\setminus \Phi) \sqcup \{y\}, \Delta \sqcup (\Phi \setminus\setminus \Gamma)] & \\
 & & & [y,u[x,v,\Gamma \setminus\setminus \Lambda, \Delta \sqcup (\Lambda \setminus\setminus \Gamma)], \Lambda \sqcap \Gamma, \Gamma \sqcap \Phi] & x \notin \Phi \setminus\setminus \Gamma \text{ et } x \notin \Lambda \setminus\setminus \Gamma \\
(c_3) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[y,u,(\Lambda \setminus\setminus \{x\}) \sqcup \Delta, (\Phi \setminus\setminus \{x\}) \sqcup \Delta] & x \in \Phi \setminus\setminus \Gamma \text{ et } x \in \Lambda \setminus\setminus \Gamma \\
 & & & & \\
(f) & (\lambda y.t)[x,u,\Gamma,\Delta] & \sim & \lambda y.t[x,u,\Gamma \sqcup \{y\}, \Delta] & \\
(e_2) & (\Lambda t)[x,u,\Gamma,\Delta] & \sim & (\Gamma \sqcap \Lambda)t[x,u,\Gamma \setminus\setminus \Lambda, \Delta \sqcup (\Lambda \setminus\setminus \Gamma)] & x \notin \Lambda \setminus\setminus \Gamma \\
(d) & \Gamma \Delta t & \sim & (\Gamma \sqcup \Delta)t & \\
(\emptyset) & \emptyset t & \sim & t & \\
(c_4) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \sim & & x \in \Lambda \setminus\setminus \Gamma \text{ et } x \notin \Phi \setminus\setminus \Gamma \\
 & \multicolumn{3}{r}{t[x,v,(\Gamma \setminus\setminus \Phi) \sqcup \{y\}, \Delta \sqcup (\Phi \setminus\setminus \Gamma)][y,u,\Delta \sqcup (\Lambda \setminus\setminus \{x\}), \Gamma \sqcap \Phi]} &
\end{array}
$$

FIG. 13.3 – Système avec équivalences, sans α

$$
\begin{array}{lrcll}
(b_1) & (\lambda x.t)\ u & \to & t[x,u,\emptyset,\emptyset] & \\
(b_2) & (\Delta(\lambda x.t))\ u & \to & t[x,u,\emptyset,\Delta] & \\
(b_3) & (\lambda x.t)\ (\Gamma u) & \to & t[x,u,\Gamma,\emptyset] & \\
(b_4) & (\Delta(\lambda x.t))\ (\Gamma u) & \to & t[x,u,\Gamma,\Delta] & \\
 & & & & \\
(f) & (\lambda y.t)[x,u,\Gamma,\Delta] & \to & \lambda y.t[x,u,\Gamma \sqcup \{y\}, \Delta] & \\
(a) & (t\ u)[x,v,\Gamma,\Delta] & \to & (t[x,v,\Gamma,\Delta]\ u[x,v,\Gamma,\Delta]) & \\
(e_1) & (\Lambda t)[x,u,\Gamma,\Delta] & \to & (\Delta \sqcup (\Lambda \setminus\setminus \{x\}))t & x \in \Lambda \setminus\setminus \Gamma \\
(e_2) & (\Lambda t)[x,u,\Gamma,\Delta] & \to & (\Gamma \sqcap \Lambda)t[x,u,\Gamma \setminus\setminus \Lambda, \Delta \sqcup (\Lambda \setminus\setminus \Gamma)] & x \notin \Lambda \setminus\setminus \Gamma \\
(n_1) & y[x,t,\Gamma,\Delta] & \to & \Delta y & x \neq y \text{ or } y \in \Gamma \\
(n_2) & x[x,t,\Gamma,\Delta] & \to & \Gamma t & \\
(c_1) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & & x \in \Phi \setminus\setminus \Gamma \text{ et } x \notin \Lambda \setminus\setminus \Gamma \\
 & \multicolumn{3}{r}{t[y,u[x,v,\Gamma \setminus\setminus \Lambda, \Delta \sqcup (\Lambda \setminus\setminus \Gamma)], \Lambda \sqcap \Gamma, \Delta \sqcup (\Phi \setminus\setminus \{x\})]} & \\
(c_2) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[x,v,(\Gamma \setminus\setminus \Phi) \sqcup \{y\}, \Delta \sqcup (\Phi \setminus\setminus \Gamma)] & \\
 & & & [y,u[x,v,\Gamma \setminus\setminus \Lambda, \Delta \sqcup (\Lambda \setminus\setminus \Gamma)], \Lambda \sqcap \Gamma, \Gamma \sqcap \Phi] & x \notin \Phi \setminus\setminus \Gamma \text{ et } x \notin \Lambda \setminus\setminus \Gamma \\
(c_3) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[y,u,(\Lambda \setminus\setminus \{x\}) \sqcup \Delta, (\Phi \setminus\setminus \{x\}) \sqcup \Delta] & x \in \Phi \setminus\setminus \Gamma \text{ et } x \in \Lambda \setminus\setminus \Gamma \\
(d) & \Gamma \Delta t & \to & (\Gamma \sqcup \Delta)t & \\
(\emptyset) & \emptyset t & \to & t & \\
 & & & & \\
(c_4) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \sim & & x \in \Lambda \setminus\setminus \Gamma \text{ et } x \notin \Phi \setminus\setminus \Gamma \\
 & \multicolumn{3}{r}{t[x,v,(\Gamma \setminus\setminus \Phi) \sqcup \{y\}, \Delta \sqcup (\Phi \setminus\setminus \Gamma)][y,u,\Delta \sqcup (\Lambda \setminus\setminus \{x\}), \Gamma \sqcap \Phi]} &
\end{array}
$$

FIG. 13.4 – Système avec règles, sans α

La notion de variable libre doit être adaptée pour ces calculs. On la définit inductivement de la façon suivante :

$$\begin{array}{lcl} FV(x) & = & x \\ FV(t\ u) & = & FV(t) \cup FV(u) \\ FV(\lambda x.t) & = & FV(t) \setminus\setminus \{x\} \\ FV(\Gamma t) & = & FV(t) \sqcup \Gamma \\ FV(t[x,u,\Gamma,\Delta] & = & ((FV(t) \setminus\setminus \{x\}) \sqcup \Delta) \cup (FV(u) \sqcup \Gamma) \end{array}$$

Il est intéressant de remarquer que, d'un point de vue logique et typage, le $\sqcup$ correspond à un affaiblissement tandis que le $\cup$ correspond à une contraction. On peut continuer de réduire le terme de notre exemple jusqu'à obtenir sa forme normale, qui correspond bien à ce qu'on attendait :

$$\begin{array}{lll} \lambda x.\lambda x.(((x\ \{x,x\}x)\ \{x\}x)[x,u,\{x,x\},\emptyset]) & & \\ \downarrow a & & \\ \downarrow a & & \\ \lambda x.\lambda x.((x[x,u,\{x,x\},\emptyset]\ (\{x,x\}x)[x,u,\{x,x\},\emptyset])\ (\{x\}x)[x,u,\{x,x\},\emptyset])) & & \\ \downarrow n_1 & & x \in \{x,x\} \\ \lambda x.\lambda x.((\emptyset x\ (\{x,x\}x)[x,u,\{x,x\},\emptyset])\ (\{x\}x)[x,u,\{x,x\},\emptyset])) & & \\ \downarrow e_2 & & x \notin \{x\} \setminus\setminus \{x,x\} \\ \lambda x.\lambda x.((\emptyset x\ (\{x,x\}x)[x,u,\{x,x\},\emptyset])\ \{x\}(x[x,u,\{x\},\emptyset])) & & \\ \downarrow n_1 & & x \in \{x\} \\ \lambda x.\lambda x.((\emptyset x\ (\{x,x\}x)[x,u,\{x,x\},\emptyset])\ \{x\}(\emptyset x)) & & \\ \downarrow e_2 & & x \notin \{x,x\} \setminus\setminus \{x,x\} \\ \lambda x.\lambda x.((\emptyset x\ (\{x,x\}(x[x,u,\emptyset,\emptyset]))\ \{x\}(\emptyset x)) & & \\ \downarrow n_1 & & x \notin \emptyset \\ \lambda x.\lambda x.((\emptyset x\ (\{x,x\}(\emptyset u))\ \{x\}(\emptyset x)) & & \\ \downarrow \emptyset & & \\ \downarrow \emptyset & & \\ \downarrow \emptyset & & \\ \lambda x.\lambda x.((x\ \{x,x\}u)\ \{x\}x) & & \end{array}$$

Cependant, un autre problème apparaît lors du croisement des lieurs, comme l'illustre l'exemple suivant. Le terme $t[y,u,\emptyset,\emptyset][x,v,\emptyset,\{y\}]$ peut se réduire de deux façons :

- $y[y,u,\emptyset,\emptyset][x,v,\emptyset,\{y\}] \quad \to_{n_1} \quad u[x,v,\emptyset,\{y\}]$
- $y[y,u,\emptyset,\emptyset][x,v,\emptyset,\{y\}] \quad \to_{c_2} \quad y[x,v,\{y\},\{y\}][y,u[x,v,\emptyset,\{y\}],\emptyset,\emptyset] \quad \to_{c_3} \quad y[x,u,\emptyset,\emptyset] \quad \to_{n_2} \quad y$

Il se passe ici que le délieur de y dans la substitution sur x devient, après composition des substitutions, un délieur pour la substitution sur y, ce qui n'est pas correct. Malgré tout, il semble que l'on ne soit plus très loin d'une solution acceptable pour se passer d'α-conversion. Dans la suite de la thèse, nous continuons d'utiliser le système modulo α-équivalence ; nous reportons l'étude de cette possibilité pour un travail futur.

Puisqu'on travaille modulo α-équivalence, il faut définir l'opération de renommage.

Définition 13.3.1 (Renommage de λ_{wsn})

On suppose qu'on dispose d'un ensemble infini de variables fraîches $\mathcal{F}$. L'opération de renommage

est définie inductivement sur la structure des termes par les égalités suivantes :

$$
\begin{array}{rcll}
x\{x \leftarrow y\} & = & y & \\
z\{x \leftarrow y\} & = & z & \text{si } z \neq x \\
(t\ u)\{x \leftarrow y\} & = & (t\{x \leftarrow y\}\ u\{x \leftarrow y\}) & \\
(\Gamma t)\{x \leftarrow y\} & = & \Gamma(t\{x \leftarrow y\}) & \text{si } x \notin \Gamma \\
(\Gamma t)\{x \leftarrow y\} & = & ((\Gamma \setminus \{x\}) \cup \{y\})t & \text{si } x \in \Gamma \\
(\lambda x.t)\{x \leftarrow y\} & = & \lambda z.(t\{x \leftarrow z\}\{x \leftarrow y\}) & z \in \mathcal{F} \\
(\lambda z.t)\{x \leftarrow y\} & = & \lambda z.(t\{x \leftarrow y\}) & \text{si } x \neq z \\
(t[z,u,\Gamma,\Delta])\{x \leftarrow y\} & = & t[z,u,(\Gamma \setminus \{x\}) \cup \{y\},(\Delta \setminus \{x\}) \cup \{y\}] & \text{si } x \in \Gamma \cap \Delta \\
(t[z,u,\Gamma,\Delta])\{x \leftarrow y\} & = & t[z,u\{x \leftarrow y\},\Gamma,(\Delta \setminus \{x\}) \cup \{y\}] & \text{si } x \in \Delta \setminus \Gamma \\
(t[z,u,\Gamma,\Delta])\{x \leftarrow y\} & = & (t\{x \leftarrow y\})[z,u,(\Gamma \setminus \{x\}) \cup \{y\},\Delta] & \text{si } x \in \Gamma \setminus \Delta \text{ et } x \neq z \\
(t[z,u,\Gamma,\Delta])\{x \leftarrow y\} & = & (t\{x \leftarrow y\})[z,u\{x \leftarrow y\},\Gamma,\Delta] & \text{si } x \notin \Gamma \cup \Delta \text{ et } x \neq z \\
(t[x,u,\Gamma,\Delta])\{x \leftarrow y\} & = & (t\{x \leftarrow z\}\{x \leftarrow y\})[z,u,(\Gamma \setminus \{x\}) \cup \{y\},\Delta] & \text{si } x \in \Gamma \setminus \Delta,\ z \in \mathcal{F} \\
(t[x,u,\Gamma,\Delta])\{x \leftarrow y\} & = & (t\{x \leftarrow z\}\{x \leftarrow y\})[z,u\{x \leftarrow y\},\Gamma,\Delta] & \text{si } x \notin \Gamma \cup \Delta,\ z \in \mathcal{F}
\end{array}
$$

13.4 Le λ_{wsn}-calcul

Il faut à présent choisir entre les deux systèmes, celui avec règles ou celui avec équivalences. D'un point de vue théorique, il est plus intéressant d'étudier celui avec équivalences, qui nous rapproche un peu plus de la logique. D'un point de vue pratique, il est plus intéressant d'étudier celui avec règles, qui nous rapproche un peu plus de l'implantation (de la programmation). On va choisir le système avec équivalences et on va voir ci-dessous qu'on peut tirer de son étude des conséquences pour le calcul avec règles. C'est donc le système avec équivalences qui sera noté λ_{wsn} (il est présenté figure 13.1) ; le système avec règles, quand à lui, sera noté $\lambda^{\rightarrow}_{wsn}$. Dans la suite, on appelle *Req* l'ensemble des règles de réduction f, e_2, d et $\emptyset$ et on note $\neg Req$ l'ensemble des autres.

Pour approfondir les relations entre les deux systèmes, nous allons établir une simulation du système avec règles par le système avec équivalences. Une telle simulation est toujours possible puisqu'il suffit, à chaque fois qu'on applique une règle de *Req*, d'utiliser dans l'autre système l'équivalence correspondante. Par exemple, la réduction de gauche dans le système avec règles sera simulée par la réduction de droite dans le système avec équivalences :

$$
\begin{array}{c}
(\{x\}y)[z,u,\{x\},\emptyset][y,v,\emptyset,\emptyset] \\
\rightarrow_{e_2} \\
(\{x\}(y[z,u,\emptyset,\emptyset]))[y,v,\emptyset,\emptyset] \\
\rightarrow_{e_2} \\
y[z,u,\emptyset,\emptyset][y,v,\emptyset,\{x\}] \\
\rightarrow_{n_1} \\
(\emptyset y)[y,v,\emptyset,\{x\}] \\
\rightarrow_{\emptyset} \\
y[y,v,\emptyset,\{x\}] \\
\rightarrow_{n_1} \\
\emptyset v \\
\rightarrow_{\emptyset} \\
v
\end{array}
\qquad\qquad
\begin{array}{c}
(\{x\}y)[z,u,\{x\},\emptyset][y,v,\emptyset,\emptyset] \\
\sim_{e_2} \\
(\{x\}(y[z,u,\emptyset,\emptyset]))[y,v,\emptyset,\emptyset] \\
\sim_{e_2} \\
y[z,u,\emptyset,\emptyset][y,v,\emptyset,\{x\}] \\
\rightarrow_{n_1} \\
(\emptyset y)[y,v,\emptyset,\{x\}] \\
\sim_{\emptyset} \\
y[y,v,\emptyset,\{x\}] \\
\rightarrow_{n_1} \\
\emptyset v \\
\sim_{\emptyset} \\
v
\end{array}
$$

Les deux systèmes se comportent de la même façon pour la préservation du typage. Le point intéressant de cette simulation, c'est qu'on peut déduire la normalisation forte du système avec règles de celle du système avec équivalences, à la condition suivante : il faut établir qu'aucune réduction infinie n'est possible, dans le système avec règles, en n'utilisant que les règles de *Req*. C'est l'objet du lemme suivant :

Lemme 13.4.1 (Normalisation forte de *Req*)

> Le système de réécriture dont les termes sont ceux de λ_{wsn} et les règles sont celles contenues dans *Req* est fortement normalisant.

Preuve : On se donne comme poids pour les termes, le couple composé du nombre d'étiquettes suivi (dans l'ordre lexicographique) de la mesure h définie inductivement de la façon suivante :

$$\begin{array}{lcl} h(x) & = & x \\ h(t\ u) & = & h(t) + h(u) + 1 \\ h(\lambda x.t) & = & h(t) + 1 \\ h(\Gamma t) & = & h(t) + 1 \\ h(t[x, u, \Gamma, \Delta]) & = & h(t) \times (h(u) + 1) \end{array}$$

L'application des règles d et $\emptyset$ fait décroître strictement la première mesure. Les règles f et e_2 laissent le nombre d'étiquettes inchangé, et voici ce qui se passe pour la deuxième mesure.

- Si $(\lambda y.t)[x, u, \Gamma, \Delta] \to \lambda y.(t[x, u, \Gamma \cup \{y\}, \Delta]$ alors on a $h((\lambda y.t)[x, u, \Gamma, \Delta]) = (h(t)+1) \times (h(u)+1) = h(t) \times (h(u) + 1) + h(u) + 1$ et $h(\lambda y.(t[x, u, \Gamma \cup \{y\}, \Delta]) = h(t) \times (h(u) + 1) + 1$. Puisque la mesure de tout terme est toujours supérieure ou égale à 1, on obtient que la règle f a fait décroître h.
- Si $(\Lambda t)[x, u, \Gamma, \Delta] \to (\Gamma \cap \Lambda)t[x, u, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)]$ alors on a $h((\Lambda t)[x, u, \Gamma, \Delta]) = (h(t) + 1) \times (h(u)+1) = h(t) \times (h(u)+1) + h(u) + 1$ et $h((\Gamma \cap \Lambda)t[x, u, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)]) = h(t) \times (h(u)+1) + 1$. De la même façon que dans le point précédent, on obtient que la règle e_2 à fait décroître h.

De plus, la mesure h étant monotone, lorsqu'un sous-terme voit sa valeur décroître, alors il en est de même pour le terme tout entier. ■

Voici le lemme qui établit la relation de normalisation forte entre les deux systèmes.

Lemme 13.4.2

> Si le λ_{wsn}-calcul est fortement normalisant, alors le $\lambda^{\to}_{wsn}$-calcul l'est aussi.

Preuve : On raisonne par l'absurde. Supposons qu'un terme t admette une dérivation infinie dans $\lambda^{\to}_{wsn}$. Comme le λ_{wsn}-calcul est fortement normalisant (par hypothèse), l'ensemble des règles de $\neg Req$, qui en est un sous-système, est lui aussi fortement normalisant. Par le lemme 13.4.1, l'ensemble des règles de *Req* est aussi fortement normalisant. De ce fait, la dérivation infinie de t doit forcément être une alternance de réduction de règles de *Req* et de règles de $\neg Req$:

$$t \to_{Req} t_1 \to_{\neg Req} t_2 \to_{Req} t_3 \to_{\neg Req} t_4 \to_{Req} t_5 \to_{\neg Req} \ldots$$

On peut alors construire une dérivation infinie dans λ_{wsn} :

$$t \sim_{Req} t_1 \to_{\neg Req} t_2 \sim_{Req} t_3 \to_{\neg Req} t_4 \sim_{Req} t_5 \to_{\neg Req} \ldots$$

Ce qui contredit l'hypothèse de normalisation forte de λ_{wsn}. ■

Remarque 13.4.3

> Dans la preuve précédente, pour construire la dérivation infinie dans λ_{wsn}, on a utilisé une simulation de $\lambda^{\to}_{wsn}$ par λ_{wsn}. Celle-ci a été brièvement introduite dans l'explication au début de la section. Chaque règle de *Req* (dans $\lambda^{\to}_{wsn}$) est simulée exactement par l'étape d'équivalence qui porte le même nom dans λ_{wsn}. Les règles b_1, b_2, b_3 et b_4 sont simulées en utilisant la règle b et l'équivalence $\emptyset$. Les autres règles sont simulées par les mêmes dans λ_{wsn}.

Chapitre 14

Le λ-calcul avec affaiblissement explicite

On introduit une version du λ-calcul avec affaiblissement explicite. Il correspond à la version nommée du λ_w-calcul introduit dans [26, 41], à l'exception des équivalences que l'on a définies. Il semble évident qu'il est connecté au $\curlywedge$-calcul de [44], mais l'étude des connexions entre eux fait partie des travaux futurs. L'introduction des étiquettes dans les λ-termes est intéressante à plusieurs points de vue. En plus de l'efficacité accrue des réductions, le gain majeur est une gestion plus explicite des variables, ce qui devrait permettre de se passer d'α-équivalence (comme cela est fait, parallèlement, dans le $\curlywedge$-calcul).

14.1 Définition du λ_{wn}-calcul

La grammaire des termes est celle de λ_{wsn} sans les substitutions explicites. Pour ce qui est des règles de réduction, on a de nouveau le choix de donner quatre règles pour la β-réduction, ou bien une seule en ajoutant l'équivalence $\emptyset$. C'est cette deuxième option qu'on choisit. Par commodité, on ajoute aussi l'équivalence d afin de ne pas avoir à gérer le nombre d'étiquettes consécutives dans les termes. Cela nous donne la définition suivante.

Définition 14.1.1 (Le λ_{wn}-calcul)

Les termes du λ_{wn}-calcul sont donnés par ma grammaire suivante :

$$t ::= x \mid (t\ t) \mid \lambda x.t \mid \Gamma t$$

Voici l'ensemble des règles de réduction :

$$\begin{array}{lrcl} (\beta) & (\Delta(\lambda x.t))\ (\Gamma u) & \to & t\{x \leftarrow u, \Gamma, \Delta\} \\ \\ (\emptyset) & \emptyset t & \sim & t \\ (d) & \Gamma\Delta t & \sim & (\Gamma \cup \Delta)t \end{array}$$

Puisque nous travaillons modulo α-équivalence, nous avons besoin d'une opération de renommage. Alors que dans le λ-calcul on pouvait utiliser à cette fin la substitution implicite, ici cela n'est pas possible car le renommage doit aussi avoir lieu dans les étiquettes, ce qui n'est pas possible si on utilise la substitution implicite de λ_{wn}. Voici la définition de la fonction de renommage, qui est un fragment de la fonction de renommage de λ_{wsn}.

Définition 14.1.2 (Renommage de λ_{wn})

On suppose qu'on dispose d'un ensemble infini de variables fraîches $\mathcal{F}$. L'opération de renommage

est définie inductivement sur la structure des termes par les égalités suivantes :

$$\begin{array}{rcll} x\{x \leftarrow y\} & = & y & \\ z\{x \leftarrow y\} & = & z & \text{si } z \neq x \\ (t\ u)\{x \leftarrow y\} & = & (t\{x \leftarrow y\}\ u\{x \leftarrow y\}) & \\ (\Gamma t)\{x \leftarrow y\} & = & \Gamma(t\{x \leftarrow y\}) & \text{si } x \notin \Gamma \\ (\Gamma t)\{x \leftarrow y\} & = & ((\Gamma \setminus \{x\}) \cup \{y\})t & \text{si } x \in \Gamma \\ (\lambda x.t)\{x \leftarrow y\} & = & \lambda z.(t\{x \leftarrow z\}\{x \leftarrow y\}) & z \in \mathcal{F} \\ (\lambda z.t)\{x \leftarrow y\} & = & \lambda z.(t\{x \leftarrow y\}) & \text{si } x \neq z \end{array}$$

La substitution implicite est plus sophistiquée en raison de la présence des étiquettes. Cela permettra de garder à jour les informations qui y sont stockées. La définition de la substitution implicite correspond exactement à la définition des règles de λ_{wsn} ; c'est normal puisque ces deux objets sont censés faire le même travail. On notera que les règles de composition ont disparues : elle sont inutiles pour prouver le lemme de substitution.

Définition 14.1.3 (Substitution implicite de λ_{wn})

On suppose qu'on dispose d'un ensemble infini de variables fraîches $\mathcal{F}$. La substitution implicite est définie inductivement sur la structure des termes par les égalités suivantes :

$$\begin{array}{rcll} (t\ u)\{x \leftarrow v, \Gamma, \Delta\} & = & (t\{x \leftarrow v, \Gamma, \Delta\}\ u\{x \leftarrow v, \Gamma, \Delta\}) & \\ (\lambda y.t)\{x \leftarrow v, \Gamma, \Delta\} & = & \lambda z.t\{y \leftarrow z\}\{x \leftarrow v, \Gamma \cup \{y\}, \Delta\} & z \in \mathcal{F} \\ (\Lambda t)\{x \leftarrow v, \Gamma, \Delta\} & = & (\Delta \cup (\Lambda \setminus \{x\}))t & x \in \Lambda \\ (\Lambda t)\{x \leftarrow v, \Gamma, \Delta\} & = & (\Gamma \cap \Lambda)t\{x \leftarrow u, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)\} & x \notin \Lambda \\ y\{x \leftarrow v, \Gamma, \Delta\} & = & \Delta y & x \neq y \\ x\{x \leftarrow v, \Gamma, \Delta\} & = & \Gamma t & \end{array}$$

14.2 Lien entre le λ_{wn}-calcul et le λ-calcul

On souhaite établir que les deux notions calculs possèdent les mêmes propriétés. Pour cela, on donne deux traductions et deux résultats de simulation réciproques de l'un dans l'autre. Cela nous donne automatiquement la confluence et la préservation de la normalisation forte. Tout terme du λ-calcul est aussi un terme du λ_{wn}-calcul ; la traduction est directe. Voyons comment passer d'un terme étiqueté à un terme sans étiquette.

Définition 14.2.1 (Traduction de λ_{wn} dans λ)

Soit t un terme du λ_{wn}-calcul. Il est toujours possible de trouver un terme t' qui lui soit α-équivalent et qui vérifie la convention suivante (appelée "convention de Barendregt") :

- les variables liées sont toutes différentes
- les variables libres sont toutes différentes des variables liées.

Dans le terme t', l'étiquette qui fait office de délieur n'a plus de raison d'être, car les variables dont il parle ne peuvent pas apparaître dans le sous-terme qu'il protège. Si c'était le cas, cela voudrait dire que l'un des deux points ci-dessus n'a pas été respecté. La traduction $M(t)$ d'un terme t consiste alors à effacer les étiquettes :

$$\begin{array}{rcl} M(x) & = & x \\ M(t\ u) & = & M(t)\ M(u) \\ M(\lambda x.t) & = & \lambda x.M(t) \\ M(\Gamma t) & = & M(t) \end{array}$$

Le lemme suivant dit que la traduction préserve la bonne liaison des variables.

Lemme 14.2.2

Si t vérifie la convention ci-dessus, alors quelle que soit la substitution $\{x \leftarrow u, \Gamma, \Delta\}$, on a

$$M(t\{x \leftarrow u, \Gamma, \Delta\}) \sim_\alpha M(t)\{x \leftarrow M(u)\}.$$

Preuve : Par induction sur la structure de t.

- Si $t = x$, alors
$$M(t\{x \leftarrow u, \Gamma, \Delta\}) = M(\Gamma u) = M(u)$$
et
$$M(t)\{x \leftarrow M(u)\} = x\{x \leftarrow M(u)\} = M(u).$$
- Si $t = y$ (avec $y \neq x$), alors
$$M(t\{x \leftarrow u, \Gamma, \Delta\}) = M(\Delta y) = M(y) = y$$
et
$$M(t)\{x \leftarrow M(u)\} = y\{x \leftarrow M(u)\} = y.$$
- Si $t = (v\ w)$ ou $t = \lambda y.v$, alors on conclut par hypothèse d'induction.
- Si $t = \Lambda v$ avec $x \in \Lambda$, alors
$$M(t\{x \leftarrow u, \Gamma, \Delta\}) = M(\Lambda' v) = M(v)$$
et
$$M(t)\{x \leftarrow M(u)\} = M(v)\{x \leftarrow M(u)\} = M(v)$$
car x ne peut pas être libre dans v.
- Enfin, si $t = \Lambda v$ avec $x \notin \Lambda$, alors on conclut par hypothèse d'induction.

■

Les preuves des deux lemmes de simulation suivants sont immédiates en utilisant le lemme ci-dessus et l'α-équivalence.

Lemme 14.2.3 (Simulation de λ_{wn} dans λ)

Pour tout λ_{wn}-termes t et u, si $t \to_{\lambda_{wn}} u$ alors $M(t) \to_\lambda M(u)$.

Lemme 14.2.4 (Simulation de λ dans λ_{wn})

Pour tout λ_{wn}-terme t et tout λ-terme u, si $M(t) \to_\lambda u$ alors il existe un λ_{wn}-terme t' tel que $t \to_{\lambda_{wn}} t'$ et $M(t') = u$.

Chapitre 15

Typage du λ_{wsn}-calcul

Ce chapitre présente les règles de typage du λ_{wsn}-calcul. Celles-ci permettent de mieux comprendre la raison d'être et le fonctionnement des étiquettes : ce sont des affaiblissements explicites. Ensuite, nous montrons que ce calcul possède la propriété de préservation du typage par réduction.

15.1 Règles de typage

Les règles de typage sont données dans la Figure 15.1.

$$\frac{}{\Gamma, x : A \vdash x : A}\ Ax \qquad \frac{\Gamma \vdash t : A \quad \Gamma \cap \Delta = \emptyset}{\Gamma, \Delta \vdash \Delta t : A}\ Weak$$

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

$$\frac{\Pi \setminus \Gamma \vdash u : A \quad \Pi \setminus \Delta, x : A \vdash t : B \quad (\Gamma \cup \Delta) \subset \Pi}{\Pi \vdash t[x, u, \Gamma, \Delta] : B}\ Sub$$

FIG. 15.1 – Règles de typage du λ_{wsn}-calcul

Puisqu'on travaille modulo α-conversion, on considère que, dans la règle $Weak$, l'ensemble Δ ne contient pas de variables liées dans t. Pour la rendre plus proche de la formulation de la règle Sub, on peut l'écrire de la façon suivante :

$$\frac{\Gamma \setminus \Delta \vdash t : A \quad \Delta \subset \Gamma}{\Gamma \vdash \Delta t : A}\ Weak$$

De manière symétrique, on peut aussi formuler la règle Sub de façon "additive", pour se rapprocher de la première formulation de la règle $Weak$. Pour cela, on appelle

- $\Pi_1 = \Pi \setminus (\Gamma \cup \Delta)$ la partie de l'environnement de typage qui sert à typer les deux sous-termes t et u,
- $\Pi_2 = \Gamma \setminus \Delta$ la partie qui sert à typer uniquement t,
- $\Pi_3 = \Delta \setminus \Gamma$ la partie qui sert à typer uniquement u, et
- $\Pi_4 = \Delta \cap \Gamma$ la partie qui ne type ni l'un, ni l'autre.

On formule alors la règle Sub de la façon suivante :

$$\frac{\Pi_1, \Pi_3 \vdash u : A \quad \Pi_1, \Pi_2, x : A \vdash t : B \quad \forall i, j \in \{1, 2, 3, 4\}\ \Pi_i \cap \Pi_j = \emptyset}{\Pi_1, \Pi_2, \Pi_3, \Pi_4 \vdash t[x, u, \Pi_2 \cup \Pi_4, \Pi_3 \cup \Pi_4] : B}\ Sub$$

Le découpage que l'on vient d'effectuer correspond exactement aux questions que l'on s'est posées lorsqu'on a établi les quatre règles de composition. Dans la suite de la thèse, on prendra l'une ou l'autre des formulations précédentes des règles $Weak$ et Sub, en choisissant à chaque fois celle qui s'utilisera de la façon la plus simple.

15.2 Préservation du typage

La préservation du typage par réduction est une propriété importante, particulièrement pour nous, car elle nous permettra d'effectuer la traduction de nos termes dans les réseaux de preuve. Sans elle, un terme pourrait ne plus être typable ou bien changer de type, ce qui ferait perdre au calcul son intérêt en sémantique. Pour nous, cela conduirait aussi au fait que les réduction de λ_{wsn} ne pourraient plus être simulées par l'élimination des coupures des réseaux de preuve (voir chapitre 17). Voici le théorème qui établit cette propriété.

Théorème 15.2.1 (Préservation du typage)

> Si t est typable et se réduit en une étape vers t', alors t' est typable dans le même environnement et possède le même type que t. Plus formellement :
>
> $$\Psi \vdash t : C \quad \text{et} \quad t \to t' \quad \Rightarrow \quad \Psi \vdash t' : C$$

Preuve : On prouve cela par induction sur la structure du terme t. Si la réduction a lieu dans un sous-terme de t, il est facile de voir qu'on obtient le résultat attendu en appliquant l'hypothèse d'induction sur le sous-terme réduit.

Sinon, la réduction a lieu à la racine de t, et on doit considérer tous les cas possibles. Le schéma de preuve est simple : à partir de la forme de t et du fait que $\Psi \vdash t : C$, on détermine les dernières règles appliquées dans la dérivation de typage, et on isole les sous-dérivations $\pi_1,\dots\pi_n$ grâce auxquelles il est facile de reconstruire une dérivation de typage pour $\Psi \vdash t' : C$.

On rappelle que, dans un environnement de typage, la virgule symbolise l'union ensembliste.

- Règle $b : t = (\Delta(\lambda x.u))\ (\Gamma v)$ se réduit vers $u[x, v, \Gamma, \Delta] = t'$. Puisque $t = (\Delta(\lambda x.u))\ (\Gamma v)$ sa dérivation de typage est de la forme suivante

$$\dfrac{\dfrac{\dfrac{\dfrac{\pi_1}{\Psi \setminus \Delta, x : B \vdash u : C}}{\Psi \setminus \Delta \vdash (\lambda x.u) : B \to C} \quad \dfrac{\dfrac{\pi_2}{\Psi \setminus \Gamma \vdash v : B}}{\Psi \vdash \Gamma v : B}}{\Psi \vdash \Delta(\lambda x.u) : B \to C}}{\Psi \vdash (\Delta(\lambda x.u))(\Gamma v) : C}$$

  On peut reconstruire aisément une dérivation de typage valide pour t' de la façon suivante

$$\dfrac{\dfrac{\pi_2}{\Psi \setminus \Gamma \vdash v : B} \quad \dfrac{\dfrac{\pi_1}{\Psi \setminus \Delta, x : B \vdash u : C}}{\Psi \vdash u[x, v, \Gamma, \Delta] : C}}{}$$

- Équivalence $f : t = (\lambda y.u)[x, v, \Gamma, \Delta]$ est équivalent à $\lambda y.(u[x, v, (\Gamma \cup \{y\}), \Delta]) = t'$. En raison de la forme de t, sa dérivation de typage est de la forme suivante, avec $C = A \to D$

$$\dfrac{\dfrac{\pi_1}{\Psi \setminus \Gamma \vdash v : B} \quad \dfrac{\dfrac{\pi_2}{\Psi \setminus \Delta, x : B, y : A \vdash u : D}}{\Psi \setminus \Delta, x : B \vdash (\lambda y.u) : C = A \to D}}{\Psi \vdash (\lambda y.u)[x, v, \Gamma, \Delta] : C}$$

  On peut construire la dérivation suivante,

$$\dfrac{\dfrac{\dfrac{\pi_1}{(\Psi, y : A) \setminus (\Gamma \cup \{y\}) = \Psi \setminus \Gamma \vdash v : B} \quad \dfrac{\pi_2}{(\Psi, y : A) \setminus \Delta = \Psi \setminus \Delta, y : A, x : B \vdash u : D}}{\Psi, y : A \vdash u[x, v, (\Gamma \cup \{y\}), \Delta] : D}}{\Psi \vdash \lambda y.(u[x, v, (\Gamma \cup \{y\}), \Delta]) : C = A \to D}$$

  Et réciproquement si on utilise l'équivalence dans l'autre sens.

- Règle $a : t = (u\ v)[x, w, \Gamma, \Delta]$ se réduit vers $(u[x, w, \Gamma, \Delta]\ v[x, w, \Gamma, \Delta]) = t'$ et la dérivation de typage de t est de la forme

$$\dfrac{\dfrac{\pi_1}{\Psi \setminus \Gamma \vdash w : A} \quad \dfrac{\dfrac{\pi_2}{\Psi \setminus \Delta, x : A \vdash u : B \to C} \quad \dfrac{\pi_3}{\Psi \setminus \Delta, x : A \vdash v : B}}{\Psi \setminus \Delta, x : A \vdash u\ v : C}}{\Psi \vdash (u\ v)[x, w, \Gamma, \Delta] : C}$$

  On peut construire la dérivation suivante

$$\dfrac{\dfrac{\dfrac{\pi_1}{\Psi \setminus \Gamma \vdash w : A} \quad \dfrac{\pi_2}{\Psi \setminus \Delta, x : A \vdash u : B \to C}}{\Psi \vdash u[x, w, \Gamma, \Delta] : B \to C} \quad \dfrac{\dfrac{\pi_1}{\Psi \setminus \Gamma \vdash w : A} \quad \dfrac{\pi_3}{\Psi \setminus \Delta, x : A \vdash v : B}}{\Psi \vdash v[x, w, \Gamma, \Delta] : B}}{\Psi \vdash (u[x, w, \Gamma, \Delta]\ v[x, w, \Gamma, \Delta]) : C}$$

- Règle $e_1 : t = (\Lambda u)[x, v, \Gamma, \Delta]$ se réduit vers $(\Delta \cup (\Lambda \setminus \{x\}))u = t'$, avec $x \in \Lambda$. La dérivation de typage de t est de la forme

$$\dfrac{\dfrac{\pi_1}{\Psi \setminus \Gamma \vdash v : A} \quad \dfrac{\dfrac{\pi_2}{(\Psi \setminus \Delta, x : A) \setminus \Lambda = (\Psi \setminus \Delta) \setminus (\Lambda \setminus \{x\}) \vdash u : C}}{\Psi \setminus \Delta, x : A \vdash \Lambda u : C}}{\Psi \vdash (\Lambda u)[x, v, \Gamma, \Delta] : C}$$

  On peut construire la dérivation de typage suivante pour t'

$$\dfrac{\dfrac{\pi_2}{\Psi \setminus (\Delta \cup (\Lambda \setminus \{x\})) \vdash u : C}}{\Psi \vdash (\Delta \cup (\Lambda \setminus \{x\}))u : C}$$

  Pour π_2, il faut vérifier que

$$(\Psi \setminus \Delta) \setminus (\Lambda \setminus \{x\}) \quad = \quad \Psi \setminus (\Delta \cup (\Lambda \setminus \{x\}))$$

  ce qui est immédiat en utilisant la propriété $(n.)$ [1]

- Équivalence $e_2 : t = (\Lambda u)[x, v, \Gamma, \Delta]$ est équivalent à $(\Gamma \cap \Lambda)u[x, v, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] = t'$ avec $x \notin \Lambda$. La dérivation de typage de t est de la forme

$$\dfrac{\dfrac{\pi_1}{\Psi \setminus \Gamma \vdash v : A} \quad \dfrac{\dfrac{\pi_2}{(\Psi \setminus \Delta, x : A) \setminus \Lambda = (\Psi \setminus \Delta) \setminus \Lambda, x : A \vdash u : C}}{\Psi \setminus \Delta, x : A \vdash \Lambda u : C}}{\Psi \vdash (\Lambda u)[x, v, \Gamma, \Delta] : C}$$

  On peut construire la dérivation de typage suivante pour t'

$$\dfrac{\dfrac{\dfrac{\pi_1}{(\Psi \setminus (\Gamma \cap \Lambda)) \setminus (\Gamma \setminus \Lambda) \vdash v : A} \quad \dfrac{\pi_2}{(\Psi \setminus (\Gamma \cap \Lambda)) \setminus (\Delta \cup (\Lambda \setminus \Gamma)), x : A \vdash u : C}}{\Psi \setminus (\Gamma \cap \Lambda) \vdash u[x, v, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] : C}}{\Psi \vdash (\Gamma \cap \Lambda)u[x, v, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] : C}$$

  Il faut vérifier :

  1. pour π_1, que

$$(\Psi \setminus (\Gamma \cap \Lambda)) \setminus (\Gamma \setminus \Lambda) = \Psi \setminus \Gamma$$

[1] Voir annexe A.

2. et pour π_2, que

$$(\Psi \setminus (\Gamma \cap \Lambda)) \setminus (\Delta \cup (\Lambda \setminus \Gamma)) = (\Psi \setminus \Delta) \setminus \Lambda$$

On calcule :

$$\begin{array}{lll} 1. & (\Psi \setminus (\Gamma \cap \Lambda)) \setminus (\Gamma \setminus \Lambda) & \\ = & \Psi \setminus ((\Gamma \cap \Lambda) \cup (\Gamma \setminus \Lambda)) & (n.) \\ = & \Psi \setminus \Gamma & (k.) \end{array}$$

$$\begin{array}{lll} 2. & (\Psi \setminus (\Gamma \cap \Lambda)) \setminus (\Delta \cup (\Lambda \setminus \Gamma)) & \\ = & ((\Psi \setminus (\Gamma \cap \Lambda)) \setminus (\Lambda \setminus \Gamma)) \setminus \Delta & (n.) \\ = & (\Psi \setminus ((\Gamma \cap \Lambda) \cup (\Lambda \setminus \Gamma))) \setminus \Delta & (n.) \\ = & (\Psi \setminus \Lambda) \setminus \Delta & (k.) \\ = & (\Psi \setminus \Delta) \setminus \Lambda & (p.) \end{array}$$

On procède de la même façon si on utilise l'équivalence dans l'autre sens.

- Règle $n_1 : t = y[x, u, \Gamma, \Delta]$ se réduit vers $\Delta y = t'$. La dérivation de typage de t est de la forme

$$\dfrac{\dfrac{\pi_1}{\Psi \setminus \Gamma \vdash u : A} \quad \overline{\Psi \setminus \Delta, x : A \vdash y : C}}{\Psi \vdash y[x, u, \Gamma, \Delta] : C}$$

On peut aisément construire la dérivation de typage de t'

$$\dfrac{\overline{\Psi \setminus \Delta \vdash y : C}}{\Psi \vdash \Delta y : C}$$

Car si $y \in \Psi \setminus \Delta$ alors $y \in \Psi$.

- Règle $n_2 : t = x[x, u, \Gamma, \Delta]$ se réduit vers $\Gamma u = t'$. On a la dérivation suivante pour t

$$\dfrac{\dfrac{\pi_1}{\Psi \setminus \Gamma \vdash u : C} \quad \overline{\Psi \setminus \Delta, x : C \vdash x : C}}{\Psi \vdash x[x, u, \Gamma, \Delta] : C}$$

On a directement la dérivation de typage de t'

$$\dfrac{\dfrac{\pi_1}{\Psi \setminus \Gamma \vdash u : C}}{\Psi \vdash \Gamma u : C}$$

- Règle $c_1 : t = u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta]$ se réduit vers $u[y, v[x, w, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})] = t'$ avec $x \in \Phi \setminus \Lambda$. La dérivation de typage de t est de la forme

$$\dfrac{\dfrac{\pi_1}{\Psi \setminus \Gamma \vdash w : A} \quad \dfrac{\dfrac{\pi_2}{(\Psi \setminus \Delta, x : A) \setminus \Lambda \vdash v : B} \quad \dfrac{\pi_3}{(\Psi \setminus \Delta, x : A) \setminus \Phi, y : B \vdash u : C}}{\Psi \setminus \Delta, x : A \vdash u[y, v, \Lambda, \Phi] : C}}{\Psi \vdash u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta] : C}$$

On construit la dérivation de typage de t'

$$\dfrac{\pi \qquad \dfrac{\pi_3}{\Psi \setminus (\Delta \cup (\Phi \setminus \{x\})), y : B \vdash u : C}}{\Psi \vdash u[y, v[x, w, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})] : C}$$

Où π est la dérivation suivante

$$\dfrac{\dfrac{\pi_1}{(\Psi \setminus (\Lambda \cap \Gamma)) \setminus (\Gamma \setminus \Lambda) \vdash w : A} \quad \dfrac{\pi_2}{(\Psi \setminus (\Lambda \cap \Gamma)) \setminus (\Delta \cup (\Lambda \setminus \Gamma)), x : A \vdash v : B}}{\Psi \setminus (\Lambda \cap \Gamma) \vdash v[x, w, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] : B}$$

Il faut vérifier :

1. pour π_1, que
$$(\Psi \setminus (\Lambda \cap \Gamma)) \setminus (\Gamma \setminus \Lambda) = \Psi \setminus \Gamma$$
2. pour π_2, que
$$(\Psi \setminus (\Lambda \cap \Gamma)) \setminus (\Delta \cup (\Lambda \setminus \Gamma)) = (\Psi \setminus \Delta) \setminus \Lambda$$
car $(\Psi \setminus \Delta, x : A) \setminus \Lambda = (\Psi \setminus \Delta) \setminus \Lambda, x : A$ en raison du fait que $x \notin \Lambda$
3. et pour π_3, que
$$(\Psi \setminus \Delta, x : A) \setminus \Phi = \Psi \setminus (\Delta \cup (\Phi \setminus \{x\}))$$

Les points 1. et 2. sont vérifiés par les calculs du cas de la règle e_2. On calcule :

$$\begin{array}{lll} & (\Psi \setminus \Delta, x : A) \setminus \Phi & \\ = & ((\Psi \setminus \Delta) \setminus \Phi) \cup \emptyset & (a.) \\ = & (\Psi \setminus \Delta) \setminus \Phi & \end{array}$$

$$\begin{array}{lll} & \Psi \setminus (\Delta \cup (\Phi \setminus \{x\})) & \\ = & (\Psi \setminus (\Phi \setminus \{x\})) \setminus \Delta & (n.) \\ = & (\Psi \setminus \Phi) \setminus \Delta & (s.) \text{ car, modulo } \alpha\text{-conversion, } x \notin \Psi \\ = & (\Psi \setminus \Delta) \setminus \Phi & (p.) \end{array}$$

- Règle $c_2 : t = u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta]$ se réduit vers $u[x, w, (\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y, v[x, w, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi] = t'$ avec $x \notin \Phi \cup \Lambda$. La dérivation de typage de t est de la forme

$$\dfrac{\dfrac{\pi_1}{\Psi \setminus \Gamma \vdash w : A} \quad \dfrac{\dfrac{\pi_2}{(\Psi \setminus \Delta, x : A) \setminus \Lambda \vdash v : B} \quad \dfrac{\pi_3}{(\Psi \setminus \Delta, x : A) \setminus \Phi, y : B \vdash u : C}}{\Psi \setminus \Delta, x : A \vdash u[y, v, \Lambda, \Phi] : C}}{\Psi \vdash u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta] : C}$$

On construit la dérivation de typage de t'

$$\dfrac{\dfrac{\pi \quad \pi'}{\Psi \setminus (\Lambda \cap \Gamma) \vdash v[x, w, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] : B} \quad \dfrac{\pi'' \quad \pi'''}{\Psi \setminus (\Gamma \cap \Phi), y : B \vdash u[x, w, (\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)] : C}}{\Psi \vdash u[x, w, (\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y, v[x, w, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi] : C}$$

Où π est la dérivation suivante

$$\dfrac{\pi_1}{(\Psi \setminus (\Lambda \cap \Gamma)) \setminus (\Gamma \setminus \Lambda) \vdash w : A}$$

π' est la dérivation

$$\dfrac{\pi_2}{(\Psi \setminus (\Lambda \cap \Gamma)) \setminus (\Delta \cup (\Lambda \setminus \Gamma)), x : A \vdash v : B}$$

π'' est la dérivation

$$\dfrac{\pi_1}{(\Psi \setminus (\Gamma \cap \Phi), y : B) \setminus ((\Gamma \setminus \Phi) \cup \{y\}) \vdash w : A}$$

et enfin π''' est la dérivation

$$\dfrac{\pi_3}{(\Psi \setminus (\Gamma \cap \Phi), y : B) \setminus (\Delta \cup (\Phi \setminus \Gamma)), x : A \vdash u : C}$$

Il faut vérifier :

1. pour π_1, que
$$(\Psi \setminus (\Lambda \cap \Gamma)) \setminus (\Gamma \setminus \Lambda) = \Psi \setminus \Gamma$$
et
$$(\Psi \setminus (\Gamma \cap \Phi), y : B) \setminus ((\Gamma \setminus \Phi) \cup \{y\}) = \Psi \setminus \Gamma$$

2. pour π_2, que
$$(\Psi \setminus (\Lambda \cap \Gamma)) \setminus (\Delta \cup (\Lambda \setminus \Gamma)) = (\Psi \setminus \Delta) \setminus \Lambda$$
car $(\Psi \setminus \Delta, x : A) \setminus \Lambda = (\Psi \setminus \Delta) \setminus \Lambda, x : A$ en raison du fait que $x \notin \Lambda$
3. et pour π_3, que
$$(\Psi \setminus (\Gamma \cap \Phi), y : B) \setminus (\Delta \cup (\Phi \setminus \Gamma)), x : A = (\Psi \setminus \Delta) \setminus \Phi, x : A, y : B$$
car $(\Psi \setminus \Delta, x : A) \setminus \Phi, y : B = (\Psi \setminus \Delta) \setminus \Phi, x : A, y : B$ en raison du fait que $x \notin \Phi$; on peut simplifier en
$$(\Psi \setminus (\Gamma \cap \Phi), y : B) \setminus (\Delta \cup (\Phi \setminus \Gamma)) = (\Psi \setminus \Delta) \setminus \Phi, y : B$$

La première égalité du point 1. et le point 2. sont vérifiés comme dans le cas de la règle c_1. On calcule :

1.	$(\Psi \setminus (\Gamma \cap \Phi), y : B) \setminus ((\Gamma \setminus \Phi) \cup \{y\})$	
=	$(\Psi \setminus (\Gamma \cap \Phi)) \setminus ((\Gamma \setminus \Phi) \cup \{y\})$ $\cup (y : B \setminus ((\Gamma \setminus \Phi) \cup \{y\}))$	$(a.)$
=	$(\Psi \setminus (\Gamma \cap \Phi)) \setminus ((\Gamma \setminus \Phi) \cup \{y\})$	
=	$((\Psi \setminus (\Gamma \cap \Phi)) \setminus (\Gamma \setminus \Phi)) \setminus \{y\}$	$(n.)$
=	$(\Psi \setminus ((\Gamma \cap \Phi) \cup (\Gamma \setminus \Phi))) \setminus \{y\}$	$(n.)$
=	$(\Psi \setminus \Gamma) \setminus \{y\}$	$(k.)$
=	$\Psi \setminus \Gamma$	$y \notin \Psi$

3.	$(\Psi \setminus (\Gamma \cap \Phi), y : B) \setminus (\Delta \cup (\Phi \setminus \Gamma))$	
=	$(\Psi \setminus (\Gamma \cap \Phi)) \setminus (\Delta \cup (\Phi \setminus \Gamma)), y : B$	$(a.)$ et, modulo α-conversion, $y \notin \Delta \cup (\Phi \setminus \Gamma)$
=	$((\Psi \setminus (\Gamma \cap \Phi)) \setminus (\Phi \setminus \Gamma)) \setminus \Delta, y : B$	$(n.)$
=	$(\Psi \setminus ((\Gamma \cap \Phi) \cup (\Phi \setminus \Gamma))) \setminus \Delta, y : B$	$(n.)$
=	$(\Psi \setminus \Phi) \setminus \Delta, y : B$	$(k.)$
=	$(\Psi \setminus \Delta) \setminus \Phi, y : B$	$(p.)$

- Règle c_3 : $t = u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta]$ se réduit vers $u[y, v, (\Lambda \setminus \{x\}) \cup \Delta, (\Phi \setminus \{x\}) \cup \Delta] = t'$ avec $x \in \Phi \cap \Lambda$. La dérivation de typage de t est de la forme
$$\dfrac{\dfrac{\pi_1}{\Psi \setminus \Gamma \vdash w : A} \quad \dfrac{\dfrac{\pi_2}{(\Psi \setminus \Delta, x : A) \setminus \Lambda \vdash v : B} \quad \dfrac{\pi_3}{(\Psi \setminus \Delta, x : A) \setminus \Phi, y : B \vdash u : C}}{\Psi \setminus \Delta, x : A \vdash u[y, v, \Lambda, \Phi] : C}}{\Psi \vdash u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta] : C}$$
On peut construire la dérivation de typage de t'
$$\dfrac{\dfrac{\pi_2}{\Psi \setminus ((\Lambda \setminus \{x\}) \cup \Delta) \vdash v : B} \quad \dfrac{\pi_3}{\Psi \setminus ((\Phi \setminus \{x\}) \cup \Delta), y : B \vdash u : C}}{\Psi \vdash u[y, v, (\Lambda \setminus \{x\}) \cup \Delta, (\Phi \setminus \{x\}) \cup \Delta] : C}$$
Il faut vérifier :
1. pour π_2, que
$$\Psi \setminus ((\Phi \setminus \{x\}) \cup \Delta) = (\Psi \setminus \Delta, x : A) \setminus \Phi$$
2. pour π_3, que
$$\Psi \setminus ((\Lambda \setminus \{x\}) \cup \Delta) = (\Psi \setminus \Delta, x : A) \setminus \Lambda$$

Les deux calculs sont identiques à celui du point 3. du cas de la règle c_1.

- Équivalence $c_4 : t = u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta]$ est équivalent à $u[x, w, (\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y, v, \Delta \cup (\Lambda \setminus \{x\}), \Gamma \cap \Phi] = t'$ avec $x \in \Lambda \setminus \Phi$. La dérivation de typage de t est de la forme

$$\dfrac{\dfrac{\pi_1}{\Psi \setminus \Gamma \vdash w : A} \quad \dfrac{\dfrac{\pi_2}{(\Psi \setminus \Delta, x : A) \setminus \Lambda \vdash v : B} \quad \dfrac{\pi_3}{(\Psi \setminus \Delta, x : A) \setminus \Phi, y : B \vdash u : C}}{\Psi \setminus \Delta, x : A \vdash u[y, v, \Lambda, \Phi] : C}}{\Psi \vdash u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta] : C}$$

  On peut construire la dérivation de typage de t'

$$\dfrac{\dfrac{\pi_2}{\Psi \setminus (\Delta \cup (\Lambda \setminus \{x\})) \vdash v : B} \quad \dfrac{\dfrac{\pi_1}{(\Psi \setminus (\Gamma \cap \Phi), y : B) \setminus ((\Gamma \setminus \Phi) \cup \{y\}) \vdash w : A} \quad \pi}{\Psi \setminus (\Gamma \cap \Phi), y : B \vdash u[x, w, (\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)] : C}}{\Psi \vdash u[x, w, (\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y, v, \Delta \cup (\Lambda \setminus \{x\}), \Gamma \cap \Phi] : C}$$

  Où π est la dérivation suivante

$$\dfrac{\pi_3}{(\Psi \setminus (\Gamma \cap \Phi), y : B) \setminus (\Delta \cup (\Phi \setminus \Gamma)), x : A \vdash u : C}$$

  Il faut vérifier :

  1. pour π_1, que
  $$(\Psi \setminus (\Gamma \cap \Phi), y : B) \setminus ((\Gamma \setminus \Phi) \cup \{y\}) = \Psi \setminus \Gamma$$
  2. pour π_2, que
  $$\Psi \setminus (\Delta \cup (\Lambda \setminus \{x\})) = (\Psi \setminus \Delta, x : A) \setminus \Lambda$$
  3. et pour π_3, que
  $$(\Psi \setminus (\Gamma \cap \Phi), y : B) \setminus (\Delta \cup (\Phi \setminus \Gamma)), x : A = (\Psi \setminus \Delta, x : A) \setminus \Phi, y : B$$

  Le premier point est identique à celui de la deuxième égalité du premier point du cas de la règle c_2. Le deuxième point est comme le deuxième point du cas de la règle c_3. Le troisième point est identique au troisième point du cas de la règle c_2 puisqu'on a aussi $x \notin \Phi$.
- Équivalence $d : t = \Gamma \Delta u$ est équivalent à $(\Gamma \cup \Delta)u = t'$. La dérivation de typage de t est de la forme

$$\dfrac{\dfrac{\dfrac{\pi}{(\Psi \setminus \Gamma) \setminus \Delta \vdash u : C}}{\Psi \setminus \Gamma \vdash \Delta u : C}}{\Psi \vdash \Gamma \Delta u : C}$$

  On construit la dérivation de typage de t'

$$\dfrac{\dfrac{\pi}{\Psi \setminus (\Gamma \cup \Delta) \vdash u : C}}{\Psi \vdash (\Gamma \cup \Delta)u : C}$$

  Il faut vérifier que

$$(\Psi \setminus \Gamma) \setminus \Delta = \Psi \setminus (\Gamma \cup \Delta)$$

  ce qui est immédiat en utilisant la propriété $(n.)$ On procède de la même façon si on utilise l'équivalence dans l'autre sens.
- Équivalence $\emptyset : t = \emptyset u$ est équivalent à $u = t'$. La dérivation de typage de t est de la forme

$$\dfrac{\dfrac{\pi}{\Psi \vdash u : C}}{\Psi \vdash \emptyset u : C}$$

  On a déjà la dérivation de typage de t'

$$\dfrac{\pi}{\Psi \vdash u : C}$$

  On procède de la même façon si on utilise l'équivalence dans l'autre sens.

■

Chapitre 16

Normalisation forte de $\mathcal{R}_E$ avec wc et wb

Pour effectuer la simulation des règles de λ_{wsn} par l'élimination des coupures dans les réseaux de preuve, nous avons besoin d'enrichir le système présenté dans [18] d'une nouvelle règle, wb, qui permet de sortir un nœud weakening d'une boîte. Dans ce chapitre, nous démontrons la normalisation forte du système de réécriture $\mathcal{R}_E$ enrichi de cette règle supplémentaire.

Après avoir présentée la règle wb et énoncé quelques propriétés, nous passerons à la preuve de normalisation forte de cette nouvelle réduction $\mathcal{R}_E$.

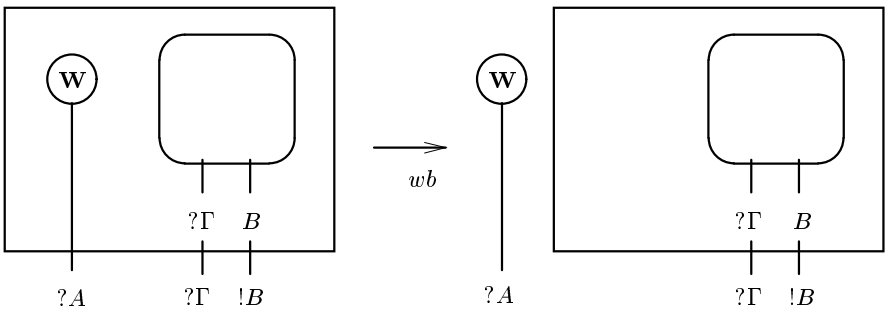
16.1 La règle wb

Définition 16.1.1 (Règle wb)

Dans la logique linéaire, les deux preuves ci-dessous sont équivalentes à l'ordre d'application des règles près, elle prouvent exactement le même séquent :

$$\dfrac{\dfrac{\vdash \Gamma, B}{\vdash ?A, \Gamma, B}\ Weakening}{\vdash ?A, \Gamma, !B}\ Box \qquad \dfrac{\dfrac{\vdash \Gamma, B}{\vdash \Gamma, !B}\ Box}{\vdash ?A, \Gamma, !B}\ Weakening$$

On en déduit que cet ordre n'est pas significatif et qu'on peut proposer une réduction pour laquelle les formes normales des réseaux n'auraient plus de nœuds weakening directement branchés sur des sorties de boîtes : ceux-ci seraient tous passés à l'extérieur de ces boîtes. On définit la réduction wb de la façon suivante :



Remarque 16.1.2

On a définit wb comme une règle et non comme une équivalence, contrairement à A et B. Ce choix n'a pas été fait à cause d'un problème de normalisation, comme c'est le cas pour la règle wc, mais à cause de la bonne formation des réseaux de preuve. En effet, même si faire entrer un weakening

dans la boîte dont il vient de sortir ne pose pas de problème, de façon générale, faire entrer un weakening quelconque dans une boîte quelconque donne un graphe qui n'est plus un réseau de preuve correct.

Nous pouvons tout de suite donner une proposition fort simple.

Proposition 16.1.3 (wb est fortement normalisant)

Le système de réécriture composé de la règle wb est fortement normalisant.

Preuve : Il suffit de considérer un poids sur les réseaux égal à la somme des profondeurs des nœuds weakening dans les boîtes. Ce poids décroît strictement à chaque application de la règle wb. ■

On rappelle au passage un résultat similaire pour wc.

Proposition 16.1.4 (wc est fortement normalisant)

Le système de réécriture composé de la règle wc est fortement normalisant.

Preuve : Voir [18]. ■

Grâce à ces propositions, il va nous être possible de démontrer la normalisation forte de $\mathcal{R}_E$ qui contient désormais wb.

16.2 Normalisation forte de $\mathcal{R}_E$

On commence par rappeler le résultat de remontée de wc par rapport aux autres règles de $\mathcal{PN}_E$.

Lemme 16.2.1

Si on a une réduction $t \to_{wc} \to_{\mathcal{PN}_E} t'$ alors il existe une réduction $t \to^+_{\mathcal{PN}_E} \to^*_{wc} t'$.

Preuve : Voir [18]. ■

On souhaite maintenant établir la normalisation forte de $\mathcal{W}_E$, ce système comprenant les règles wc et wb modulo l'équivalence E. Pour cela, on montre quelques lemmes de remontées. Les deux premiers expriment que l'on peut toujours faire remonter une étape d'équivalence devant l'application d'une règle wb ou wc.

Lemme 16.2.2

Si on a une réduction $t \to_{wb} \sim_E t'$ alors il existe une réduction $t \sim_E \to_{wb} t'$.

Preuve : Le résultat est évident puisqu'il n'y a pas de paires critiques entre la règle wb et l'équivalence $\sim_E$. ■

Lemme 16.2.3

Si on a une réduction $t \to_{wc} \sim_E t'$ alors il existe une réduction $t \sim_E \to_{wc} t'$.

Preuve : La preuve est un fragment de celle du lemme 16.2.1. ■

On peut à présent établir la normalisation forte de $\mathcal{W}_E$.

Lemme 16.2.4

La réduction $\mathcal{W}_E$ est fortement normalisante.

Preuve : On mesure les réseaux avec comme poids l'ordre lexicographique du nombre de nœuds de celui-ci suivi de la somme des profondeurs des nœuds weakening dans les boîtes. La règle wc fait décroître strictement la première partie, tandis que la règle wb la laisse inchangée mais fait décroître strictement la seconde partie. L'équivalence $\sim_E$, quant à elle, laisse le poids inchangé. ■

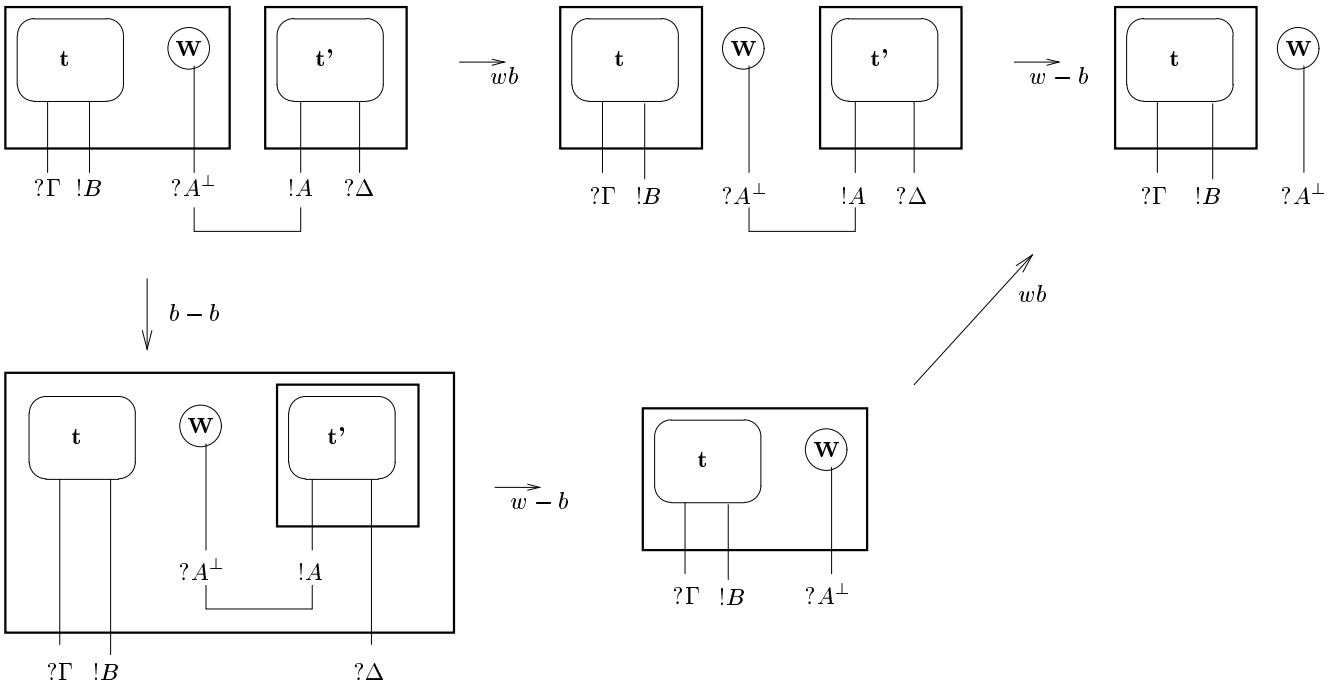
Il nous faut à présent un lemme de remontée de $\mathcal{PN}_E$ par rapport à $\mathcal{W}_E$. Pour cela, on établit deux lemmes de remontée de $\mathcal{PN}$ par rapport à wb et wc.

Lemme 16.2.5

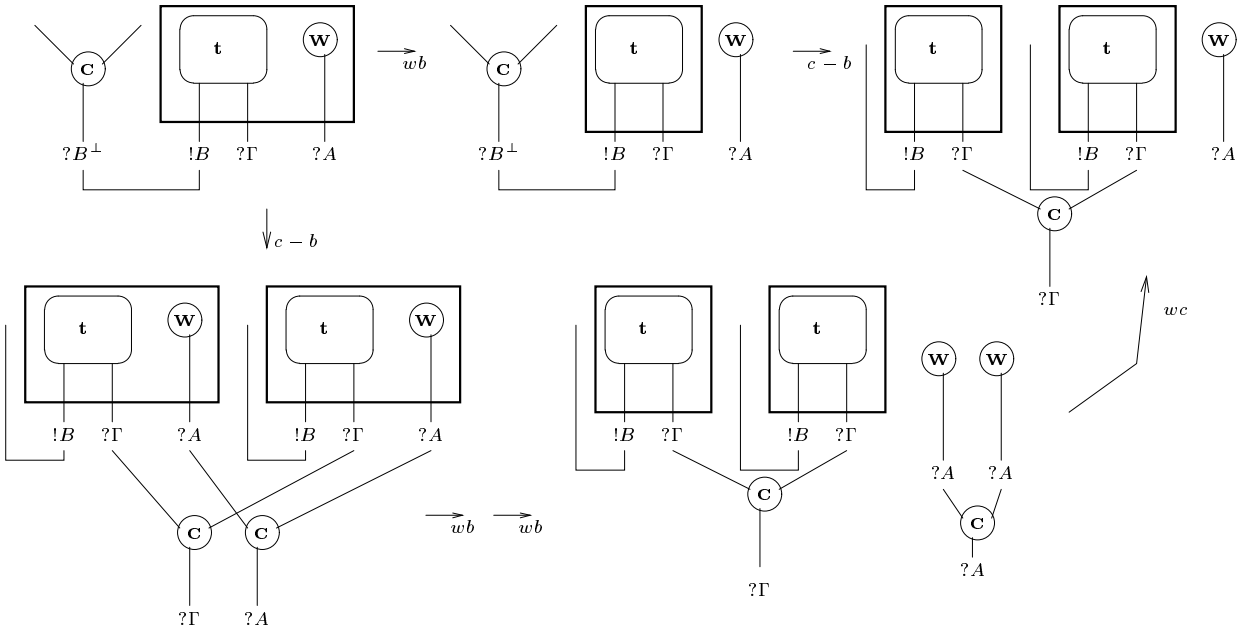
Si on a une réduction $t \to_{wb} \to_{\mathcal{PN}} t'$ alors il existe une réduction $t \to^+_{\mathcal{PN}} \to^*_{\mathcal{W}_E} t'$.

Preuve : On procède par cas sur la règle de $\mathcal{PN}$. On ne regarde que les paires critiques dont la résolution n'est pas évidente.

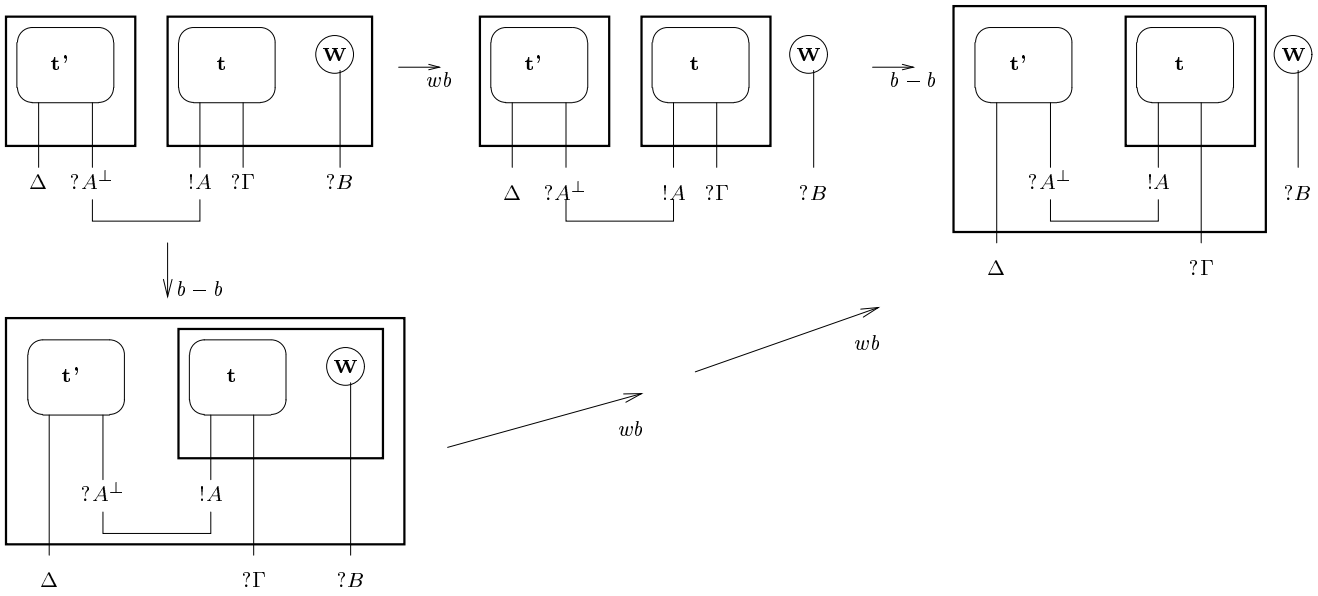
1. La règle $w - b$:



2. La règle $c - b$:



3. La règle $b - b$:



Dans les autres cas, on peut appliquer la règle de $\mathcal{PN}$, puis appliquer 0, 1 ou 2 fois la règle wb suivant que le redex est effacé, inchangé ou dupliqué. ■

Proposition 16.2.6

Si on a une réduction $t \to_{wc} \to_{\mathcal{PN}} t'$ alors il existe une réduction $t \to^+_{\mathcal{PN}} \to^*_{\mathcal{W}_E} t'$.

Preuve : La preuve est un fragment de celle du lemme 16.2.1. ■

Voici le lemme de remontée de $\mathcal{PN}_E$ par rapport à $\mathcal{W}_E$.

Lemme 16.2.7

Si on a une réduction $t \to_{\mathcal{W}_E} \to_{\mathcal{PN}_E} t'$ alors il existe une réduction $t \to_{\mathcal{PN}_E} \to^*_{\mathcal{R}_E} t'$.

Preuve : On peut décomposer la réduction

$$t \to_{\mathcal{W}_E} \to_{\mathcal{PN}_E} t'$$

en

$$t \sim_E \to_{\mathcal{W}} \sim_E \sim_E \to_{\mathcal{PN}} \sim_E t'$$

qui est équivalent à

$$t \sim_E \to_{\mathcal{W}} \sim_E \to_{\mathcal{PN}} \sim_E t'$$

En appliquant le lemme 16.2.2 ou 16.2.3, on peut construire la réduction $t \sim_E \to_{\mathcal{W}} \to_{\mathcal{PN}} \sim_E t'$. En appliquant le lemme 16.2.5 ou 16.2.6 on obtient la réduction recherchée. ■

Le lemme suivant permet d'extraire une application d'une règle de $\mathcal{PN}_E$ dans une séquence infinie de réductions de $\mathcal{R}_E$.

Lemme 16.2.8

Si on a une réduction infinie $t \to^+_{\mathcal{R}_E}$, on peut construire une réduction infinie $t \to_{\mathcal{PN}_E} \to^+_{\mathcal{R}_E}$.

Preuve : Si la réduction infinie commence déjà par une règle de $\mathcal{PN}_E$, il n'y a rien a montrer. Sinon, grâce aux lemmes 3.2.5 et 16.2.4, on sait qu'on ne peut pas avoir de séquence de réduction infinie composée uniquement de $\mathcal{PN}_E$ ou de $\mathcal{W}_E$. La séquence est donc de la forme

$$t \to^+_{\mathcal{W}_E} \to^+_{\mathcal{PN}_E} \to^+_{\mathcal{W}_E} \to^+_{\mathcal{PN}_E} \cdots$$

Il nous suffit alors d'appliquer plusieurs fois le lemme 16.2.7 pour obtenir le résultat attendu. ■

Le dernier lemme est le cœur de la preuve de normalisation forte. Il construit une suite arbitrairement grande de $\mathcal{PN}_E$ à partir d'une suite infinie de $\mathcal{R}_E$.

Lemme 16.2.9

> Soit une réduction infinie $t \to^+_{\mathcal{R}_E}$, pour tout n on peut construire une réduction infinie $t \to^n_{\mathcal{PN}_E} \to^+_{\mathcal{R}_E}$.

Preuve : Il suffit d'appliquer n fois le lemme 16.2.8 ■

On peut enfin énoncer le théorème.

Théorème 16.2.10 (Normalisation forte de $\mathcal{R}_E$)

> Le système de réduction $\mathcal{R}_E$ est fortement normalisant.

Preuve : Raisonnons par l'absurde. Supposons qu'il existe une suite de réductions infinie de $\mathcal{R}_E$, on peut alors, grâce au lemme 16.2.9, construire une suite infinie de $\mathcal{PN}_E$, ce qui contredit le théorème 3.2.5. ■

Chapitre 17

Normalisation forte du λ_{wsn}-calcul typé

La preuve de normalisation forte que nous allons effectuer présente, en plus du résultat qu'elle établit, un intérêt pour l'approfondissement des liens entre la logique et les substitutions explicites. Cette dernière méthode est inspirée et quasiment identique à celle de [18]. On commence par donner la traduction des termes de λ_{wsn} dans les réseaux de preuves de $\mathcal{PN}$, puis on passe à la preuve de normalisation forte du λ_{wsn}-calcul typé.

17.1 Traduction du λ_{wsn}-calcul dans $\mathcal{PN}$

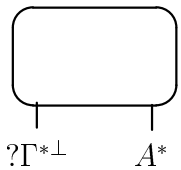
On utilise la traduction standard de la logique intuitionniste dans la logique linéaire. Voici la traduction des types :

$$\begin{array}{lcll} A^* & = & A & \text{si } A \text{ est un type atomique} \\ (A \to B)^* & = & ?((A^*)^\perp) \parr B^* & (\text{c'est-à-dire } !A^* \multimap B^*) \text{ sinon} \end{array}$$

Puisque les réseaux de preuves sont une représentation graphique et que leurs fils sont commutatifs, on se permettra de les changer de place pour faciliter la lisibilité des réseaux correspondant à la traduction de nos termes. Si t est un terme du λ_{wsn}-calcul typé, dont la dérivation de typage se termine par :

$$\overline{\Gamma \vdash t : A}$$

Le réseau que nous obtiendrons comme traduction sera de la forme :

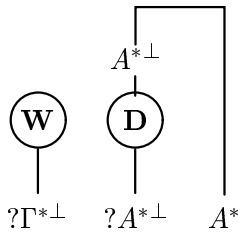


Voici à présent la traduction T des termes de λ_{wsn} dans $\mathcal{PN}$.

- Si le terme est une variable dont la dérivation de typage se termine par

$$\overline{\Gamma, x : A \vdash x : A}\ Ax$$

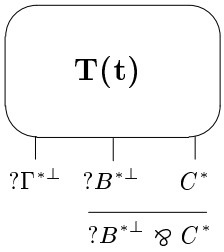
alors sa traduction est le réseau suivant



- Si le terme est une λ-abstraction don la dérivation de typage se termine par

$$\frac{x : B, \Gamma \vdash t : C}{\Gamma \vdash \lambda x.t : B \to C}\ Lambda$$

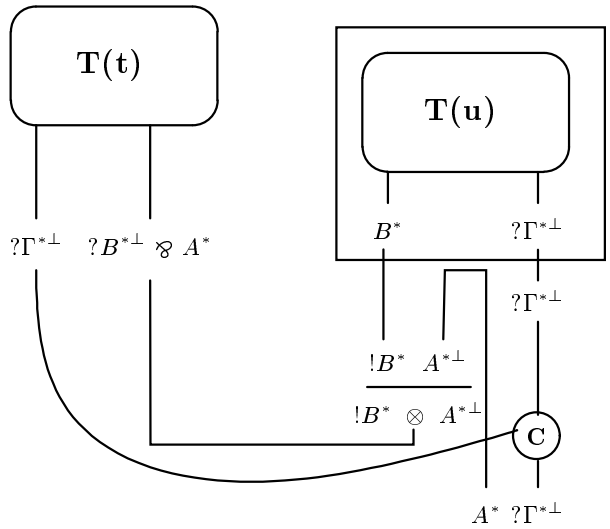
alors sa traduction est le réseau suivant



- Si le terme est une application dont la dérivation de typage se termine par

$$\frac{\Gamma \vdash t : B \to A \quad \Gamma \vdash u : B}{\Gamma \vdash (t\ u) : A}\ App$$

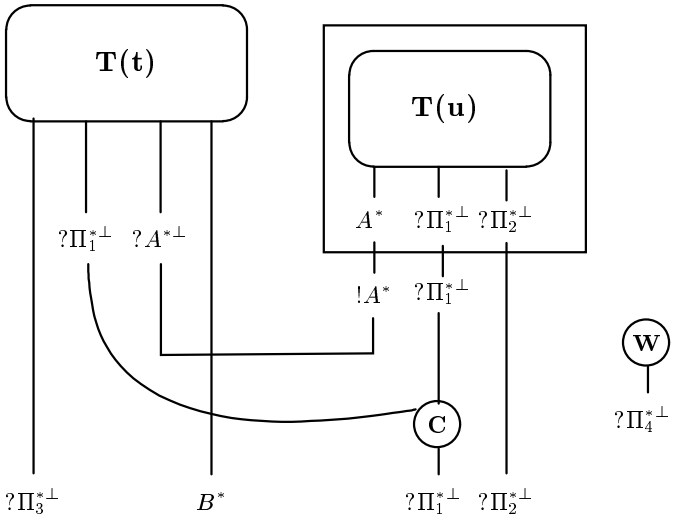
alors sa traduction est le réseau suivant



- Si le terme est une substitution dont la dérivation de typage se termine par

$$\frac{\Pi_1, \Pi_3 \vdash u : A \quad \Pi_1, \Pi_2, x : A \vdash t : B}{\Pi_1, \Pi_2, \Pi_3, \Pi_4 \vdash t[x, u, \Pi_2 \cup \Pi_4, \Pi_3 \cup \Pi_4] : B}\ Sub$$

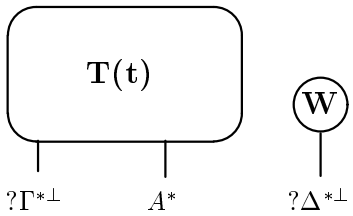
alors sa traduction est le réseau suivant



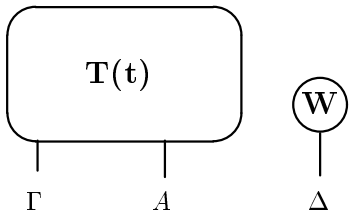
- Si le terme est une étiquette dont la dérivation de typage se termine par

$$\frac{\Gamma \vdash t : A}{\Gamma, \Delta \vdash \Delta t : A}\ Weak$$

alors sa traduction est le réseau suivant



Dans la section suivante, pour alléger les notations et ainsi faciliter la lecture des réseaux, nous ne marquerons plus l'étoile de traduction (*) sur aucune formule, et nous ne marquerons plus le point d'interrogation ni le $^\perp$ autour des lettres grecques majuscules de l'environnement de typage. Ainsi, le réseau présenté ci-dessus sera noté comme le réseau suivant



17.2 Simulation du λ_{wsn}-calcul par $\mathcal{R}_E$

Le choix que nous avons fait du système avec équivalence va rendre plus claire la simulation du λ_{wsn}-calcul par l'élimination des coupures enrichie $\mathcal{R}_E$. Par rapport à la simulation établie dans [18], il y a deux différences principales. La première est justement que la présence des équivalences ne nécessite plus de cas à part dans la simulation (en fait, ce traitement a été reporté aux lemmes 13.4.1 et 13.4.2). La deuxième concerne le fait que dans une substitution $[x, u, \Gamma, \Delta]$ on peut avoir $\Gamma \cap \Delta \neq \emptyset$. Cela va poser des problèmes car on va voir apparaître dans les traductions des membres droits de certaines règles (a et c_3) des redex wc, alors qu'ils ne sont pas dans les traductions des membres gauches de ces règles.

Nous ferons donc des réductions supplémentaires avant de comparer les traductions des termes. Il faudra éliminer les redex wc, mais aussi, auparavant, éliminer les redex wb, afin de sortir les weakening des boîtes pour réduire certains redex wc. On appelle $\mathcal{W}$-forme normale la forme normale d'un réseau pour les règles wc et wb (c'est-à-dire le système $\mathcal{W}$). On notera $w(R)$ la $\mathcal{W}$-forme normale d'un réseau R. On reviendra là-dessus dans le chapitre 18.

Voici le lemme de simulation.

Lemme 17.2.1 (Simulation de λ_{wsn} par $\mathcal{R}_E$)

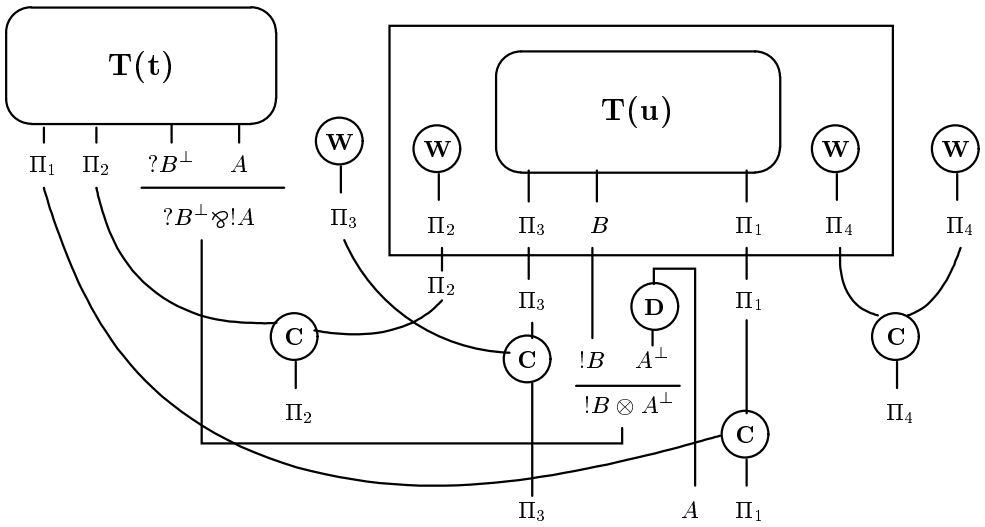
- Si $t \to_{\neg Req} t'$ alors $T(t) \to^+_{\mathcal{R}_E} T(t')$.
- Si $t \sim_{Req} t'$ alors $T(t) \sim T(t')$.
- Si $t \sim_{c_4} t'$ alors $T(t) \sim T(t')$.

Preuve : On va raisonner par cas sur la règle de réduction utilisée. Comme les règles de réduction de λ_{wsn} et de $\mathcal{R}_E$ passent au contexte, il nous suffit de regarder les réductions de tête. Dans chaque cas, on présente les réseaux obtenus par traduction des membres gauches et droits des règles (avant d'en prendre la $\mathcal{W}$-forme normale), puis on donne les réductions nécessaires pour passer de la $\mathcal{W}$-forme normale de la traduction du membre gauche à la $\mathcal{W}$-forme normale de la traduction du membre droit.

- **règle** b **:** $(\Delta(\lambda x.t))\ (\Gamma u) \to t[x,u,\Gamma,\Delta]$. La dérivation de typage de $(\Delta(\lambda x.t))\ (\Gamma u)$ se termine par

$$\dfrac{\dfrac{\dfrac{\Pi_1,\Pi_2,x:B\vdash t:A}{\Pi_1,\Pi_2\vdash\lambda x.t:B\to A}}{\Pi_1,\Pi_2,\Pi_3,\Pi_4\vdash\Delta(\lambda x.t):B\to A}\quad\dfrac{\Pi_1,\Pi_3\vdash u:B}{\Pi_1,\Pi_2,\Pi_3,\Pi_4\vdash\Gamma u:B}}{\Pi_1,\Pi_2,\Pi_3,\Pi_4\vdash(\Delta(\lambda x.t))\ (\Gamma u):A}$$

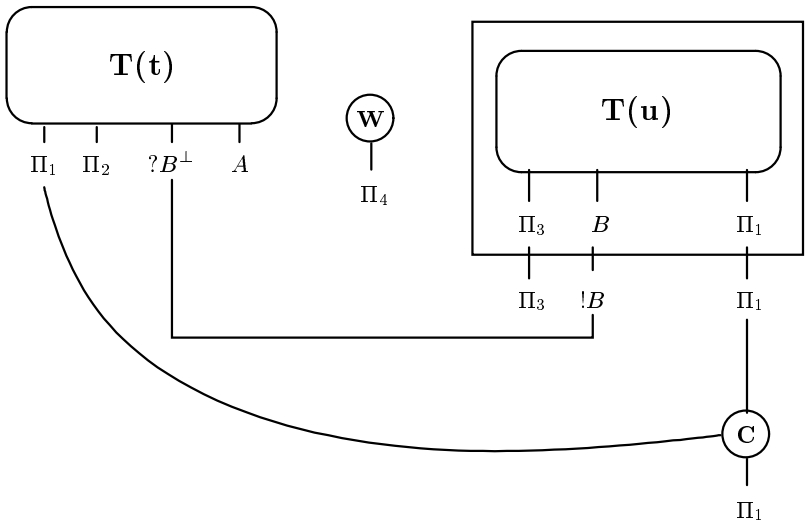
où $\Gamma=\Pi_2\cup\Pi_4$ et $\Delta=\Pi_3\cup\Pi_4$. Sa traduction est le réseau suivant



La dérivation de typage de $t[x,u,\Gamma,\Delta]$ se termine par

$$\dfrac{\Pi_1,\Pi_3\vdash u:B\quad\Pi_1,\Pi_2,x:B\vdash t:A}{\Pi_1,\Pi_2,\Pi_3,\Pi_4\vdash t[x,u,\Pi_2\cup\Pi_4,\Pi_3\cup\Pi_4]:A}$$

et sa traduction est le réseau suivant

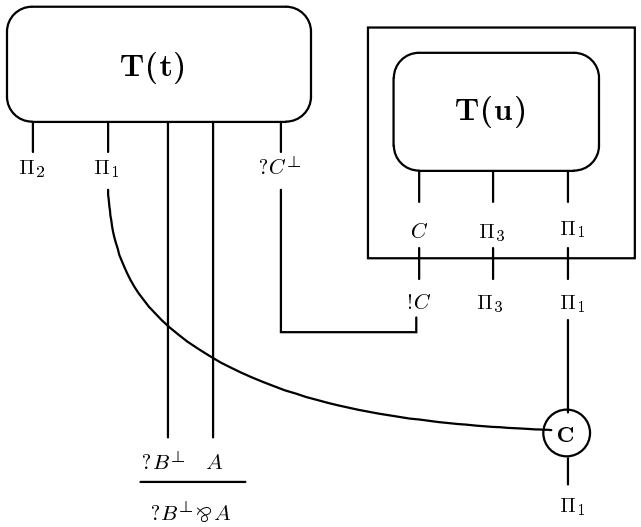


En partant de réseaux en $\mathcal{W}$-forme normale, pour réduire le premier vers le second, nous devons éliminer la coupure $⅋-\otimes$, puis la coupure $Ax-cut$ engendrée.

- **équivalence** f **:** $(\lambda y.t)[x,u,\Gamma,\Delta]\sim\lambda y.t[x,u,\Gamma\cup\{y\},\Delta]$. La dérivation de typage de $(\lambda y.t)[x,u,\Gamma,\Delta]$ se termine par

$$\dfrac{\Pi_1,\Pi_3\vdash u:C\quad\dfrac{\Pi_1,\Pi_2,y:B\vdash t:A}{\Pi_1,\Pi_2\vdash\lambda y.t:B\to A}}{\Pi_1,\Pi_2,\Pi_3,\Pi_4\vdash(\lambda y.t)[x,u,\Gamma,\Delta]:B\to A}$$

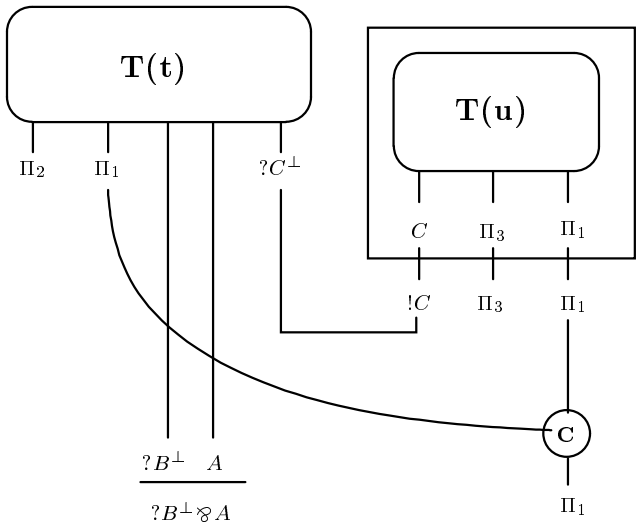
où $\Gamma=\Pi_2\cup\Pi_4$ et $\Delta=\Pi_3\cup\Pi_4$. Sa traduction est le réseau suivant



La dérivation de typage de $\lambda y.t[x, u, \Gamma \cup \{y\}, \Delta]$ se termine par

$$\dfrac{\dfrac{\Pi_1, \Pi_3 \vdash u : C \quad \Pi_1, \Pi_2, y : B \vdash t : A}{\Pi_1, \Pi_2, \Pi_3, \Pi_4, y : B \vdash t[x, u, \Gamma \cup \{y\}, \Delta] : A}}{\Pi_1, \Pi_2, \Pi_3, \Pi_4 \vdash \lambda y.t[x, u, \Gamma \cup \{y\}, \Delta] : B \to A}$$

et sa traduction est le réseau suivant

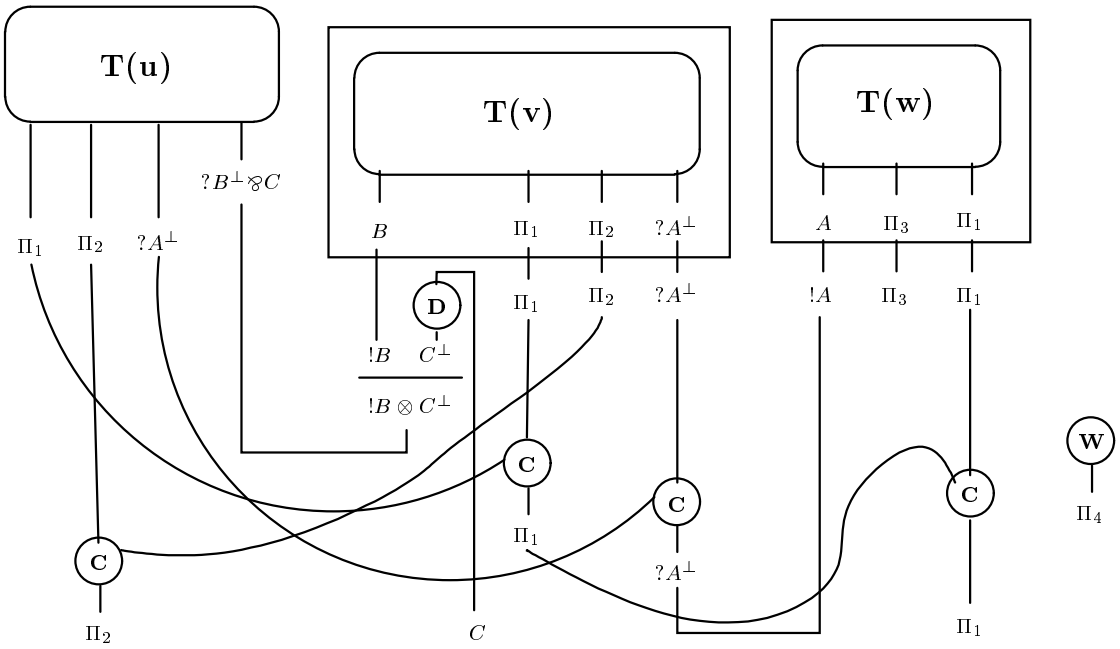


Les deux réseaux sont bien identiques.

- **règle** a **:** $(u\ v)[x, w, \Gamma, \Delta] \to (u[x, w, \Gamma, \Delta]\ v[x, w, \Gamma, \Delta])$. La dérivation de typage de $(u\ v)[x, w, \Gamma, \Delta]$ se termine par

$$\dfrac{\Pi_1, \Pi_3 \vdash w : A \qquad \dfrac{\Pi_1, \Pi_2, x : A \vdash u : B \to C \quad \Pi_1, \Pi_2, x : A \vdash v : B}{\Pi_1, \Pi_2, x : A \vdash u\ v : C}}{\Pi_1, \Pi_2, \Pi_3, \Pi_4 \vdash (u\ v)[x, w, \Gamma, \Delta] : C}$$

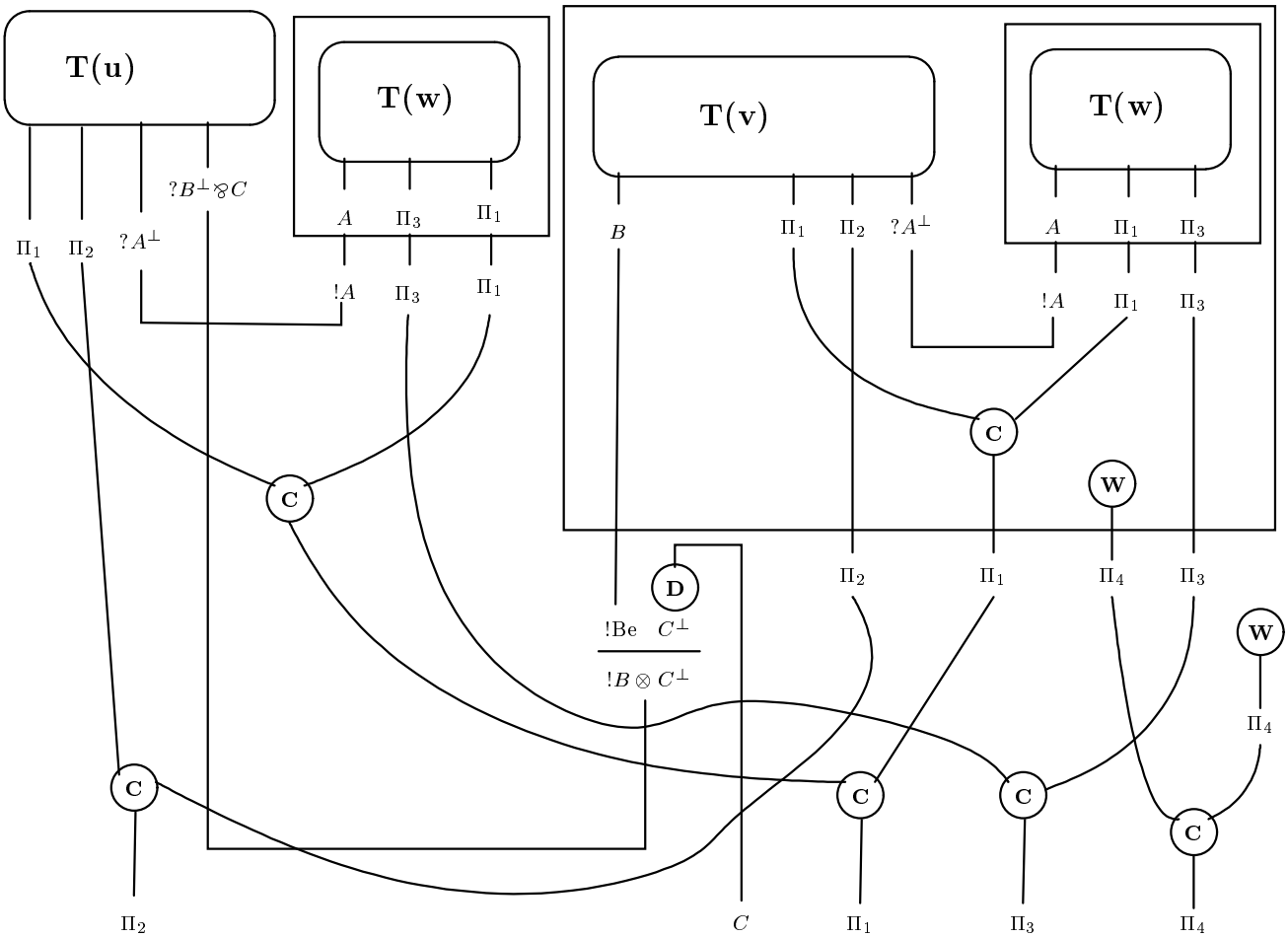
où $\Gamma = \Pi_2 \cup \Pi_4$ et $\Delta = \Pi_3 \cup \Pi_4$. Sa traduction est le réseau suivant



La dérivation de typage de $(u[x, w, \Gamma, \Delta]\ v[x, w, \Gamma, \Delta])$ se termine par

$$\dfrac{\dfrac{\Pi_1, \Pi_3 \vdash w : A \quad \Pi_1, \Pi_2, x : A \vdash u : B \to C}{\Pi_1, \Pi_2, \Pi_3, \Pi_4 \vdash u[x, w, \Gamma, \Delta] : B \to C} \quad \dfrac{\Pi_1, \Pi_3 \vdash w : A \quad \Pi_1, \Pi_2, x : A \vdash v : B}{\Pi_1, \Pi_2, \Pi_3, \Pi_4 \vdash v[x, w, \Gamma, \Delta] : B}}{\Pi_1, \Pi_2, \Pi_3, \Pi_4 \vdash (u[x, w, \Gamma, \Delta]\ v[x, w, \Gamma, \Delta]) : C}$$

Sa traduction est le réseau suivant

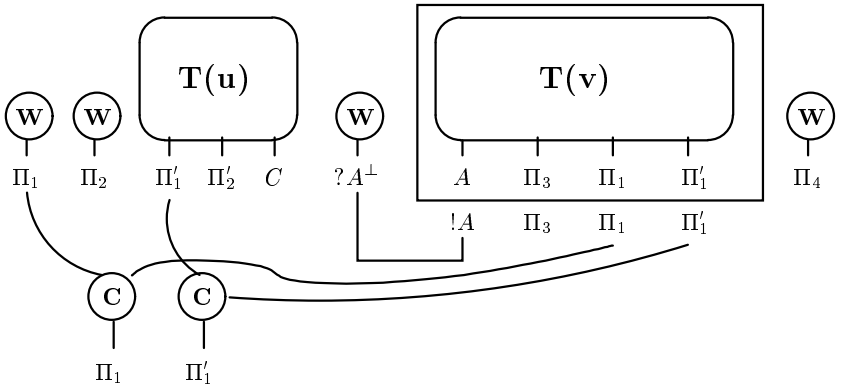


Il suffit, pour réduire la $\mathcal{W}$-forme normale du premier réseau vers la $\mathcal{W}$-forme normale du second, d'éliminer la coupure $c - b$, puis la coupure $b - b$, et enfin d'appliquer les équivalences A et B.

- **règle** e_1 **:** $(\Lambda u)[x, v, \Gamma, \Delta] \rightarrow (\Delta \cup (\Lambda \setminus \{x\}))u$, avec $x \in \Lambda$. La dérivation de typage de $(\Lambda u)[x, v, \Gamma, \Delta]$ se termine par :

$$\dfrac{\dfrac{}{\Pi_1, \Pi_1' \vdash v : A} \quad \dfrac{\dfrac{}{\Pi_1', \Pi_2' \vdash u : C}}{\Pi_1, \Pi_1', \Pi_2, \Pi_2', x : A \vdash \Lambda u : C}}{\Pi_1, \Pi_1', \Pi_2, \Pi_2', \Pi_3, \Pi_4 \vdash (\Lambda u)[x, v, \Gamma, \Delta] : C}$$

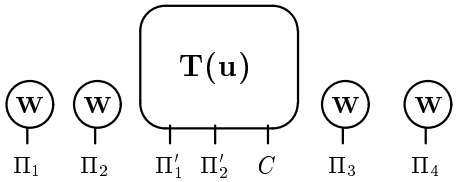
où $\Gamma = \Pi_2 \cup \Pi_2' \cup \Pi_4$, $\Delta = \Pi_3 \cup \Pi_4$ et $\Lambda = \Pi_1 \cup \Pi_2 \cup \{x\}$. Sa traduction est le réseau suivant



La dérivation de typage de $(\Delta \cup (\Lambda \setminus \{x\}))u$ se termine par

$$\dfrac{\dfrac{}{\Pi_1', \Pi_2' \vdash u : C}}{\Pi_1, \Pi_1', \Pi_2, \Pi_2', \Pi_3, \Pi_4 \vdash (\Delta \cup (\Lambda \setminus \{x\}))u : C}$$

Sa traduction est le réseau suivant

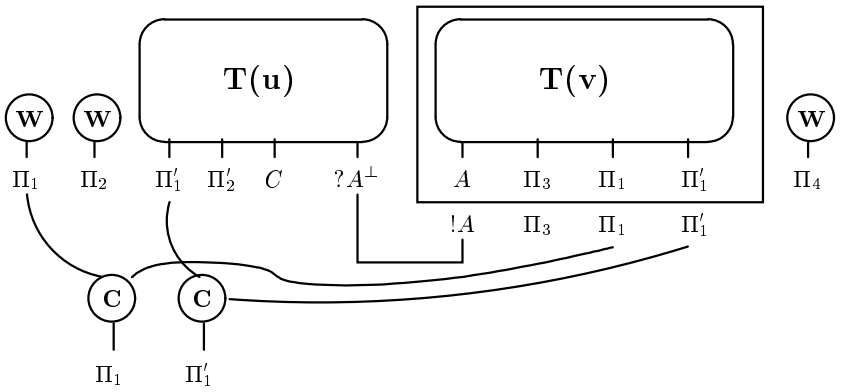


En partant de la $\mathcal{W}$-forme normale du premier réseau, on élimine la coupure $w - b$, puis on applique une fois la règle wc pour arriver au second réseau.

- **équivalence** e_2 **:** $(\Lambda u)[x, v, \Gamma, \Delta] \sim (\Gamma \cap \Lambda)u[x, v, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)]$ avec $x \notin \Lambda$. La dérivation de typage de $(\Lambda u)[x, v, \Gamma, \Delta]$ se termine par

$$\dfrac{\dfrac{}{\Pi_1, \Pi_1' \vdash v : A} \quad \dfrac{\dfrac{}{\Pi_1', \Pi_2', x : A \vdash u : C}}{\Pi_1, \Pi_1', \Pi_2, \Pi_2', x : A \vdash \Lambda u : C}}{\Pi_1, \Pi_1', \Pi_2, \Pi_2', \Pi_3, \Pi_4 \vdash (\Lambda u)[x, v, \Gamma, \Delta] : C}$$

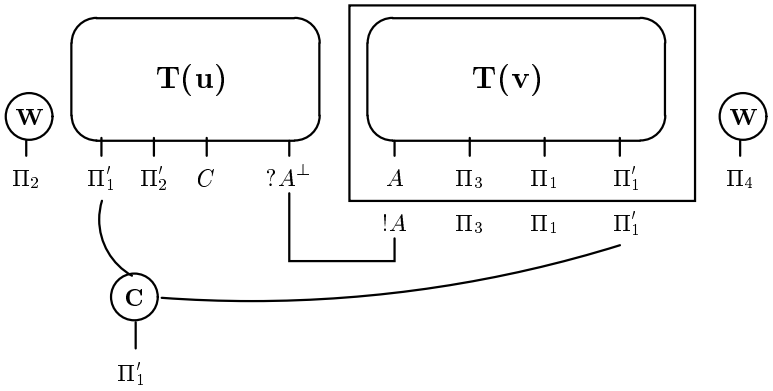
où $\Gamma = \Pi_2 \cup \Pi_2' \cup \Pi_4$, $\Delta = \Pi_3 \cup \Pi_4$ et $\Lambda = \Pi_1 \cup \Pi_2$. Sa traduction est le réseau suivant



La dérivation de typage de $(\Gamma \cap \Lambda)u[x, v, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)]$ se termine par

$$\dfrac{\dfrac{\Pi_1, \Pi'_1, \Pi_3 \vdash v : A \quad \Pi'_1, \Pi'_2, x : A \vdash u : C}{\Pi_1, \Pi'_1, \Pi'_2, \Pi_3, \Pi_4 \vdash u[x, v, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] : C}}{\Pi_1, \Pi'_1, \Pi_2, \Pi'_2, \Pi_3, \Pi_4 \vdash (\Gamma \cap \Lambda)u[x, v, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] : C}$$

Sa traduction est le réseau suivant

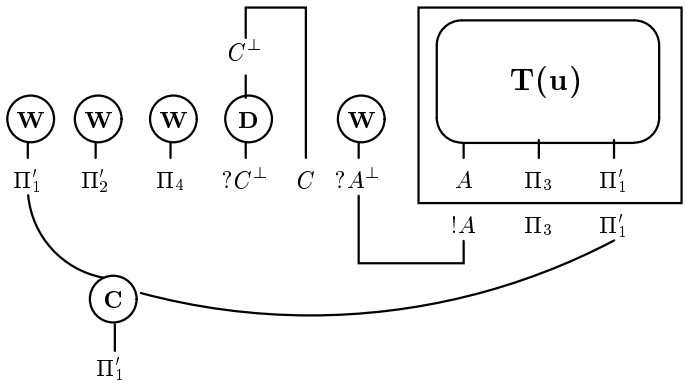


Les deux $\mathcal{W}$-formes normales sont identiques.

- **règle** n_1 **:** $y[x, u, \Gamma, \Delta] \to \Delta y$. La dérivation de typage de $y[x, u, \Gamma, \Delta]$ se termine par

$$\dfrac{\Pi_1, \Pi_3 \vdash u : A \quad \overline{\Pi_1, \Pi_2, x : A \vdash y : C}}{\Pi_1, \Pi_2, \Pi_3, \Pi_4 \vdash y[x, u, \Gamma, \Delta] : C}$$

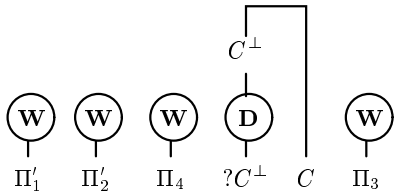
où $\Gamma = \Pi_2 \cup \Pi_4$ et $\Delta = \Pi_3 \cup \Pi_4$. Sa traduction est le réseau suivant (où $\Pi'_1 \cup \Pi'_2 = (\Pi_1 \cup \Pi_2) \setminus \{y\}$)



La dérivation de typage de Δy se termine par

$$\dfrac{\overline{\Pi_1, \Pi_2 \vdash y : C}}{\Pi_1, \Pi_2, \Pi_3, \Pi_4 \vdash \Delta y : C}$$

Sa traduction est le réseau suivant

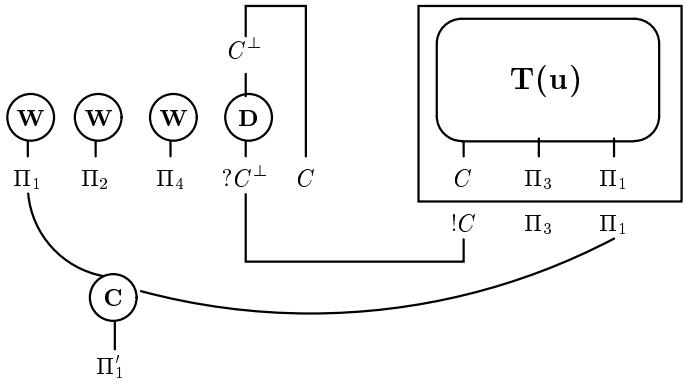


Pour passer de la $\mathcal{W}$-forme normale du premier réseau au second, il suffit d'éliminer la coupure $w - b$.

- **règle** n_2 **:** $x[x, u, \Gamma, \Delta] \to \Gamma u$. La dérivation de typage de $x[x, u, \Gamma, \Delta]$ se termine par

$$\frac{\Pi_1, \Pi_3 \vdash u : C \quad \overline{\Pi_1, \Pi_2, x : C \vdash x : C}}{\Pi_1, \Pi_2, \Pi_3, \Pi_4 \vdash x[x, u, \Gamma, \Delta] : C}$$

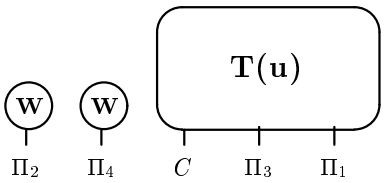
où $\Gamma = \Pi_2 \cup \Pi_4$ et $\Delta = \Pi_3 \cup \Pi_4$. Sa traduction est le réseau suivant



La dérivation de typage de Γu se termine par

$$\frac{\Pi_1, \Pi_3 \vdash u : C}{\Pi_1, \Pi_2, \Pi_3, \Pi_4 \vdash \Gamma u : C}$$

Sa traduction est le réseau suivant

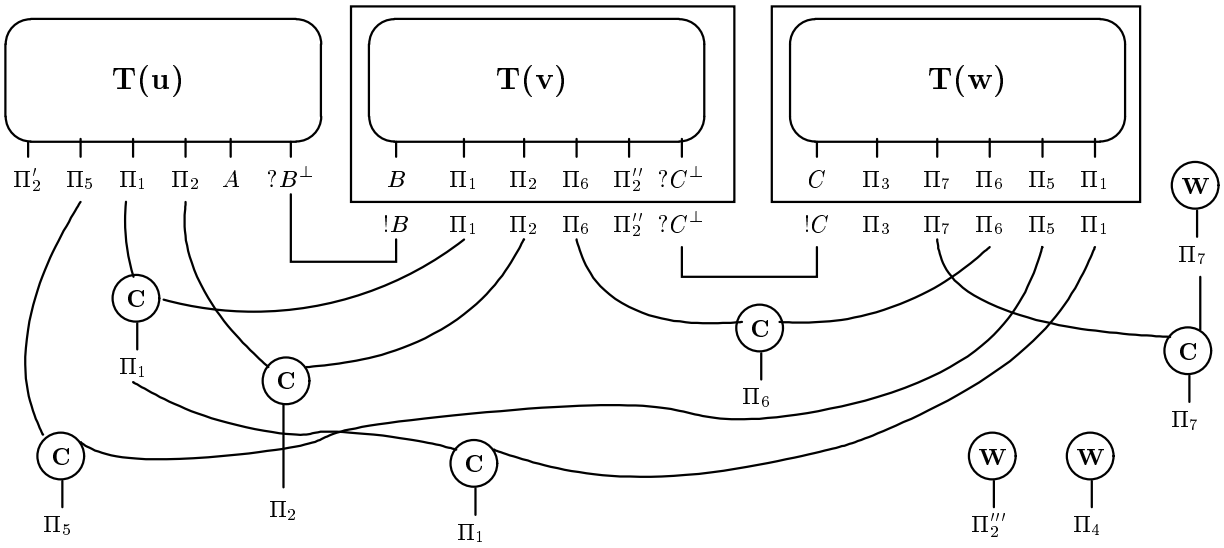


En partant de la $\mathcal{W}$-forme normale du premier réseau, on élimine la coupure $d - b$, puis la coupure $Ax - cut$ engendrée.

- **règle** c_1 **:** $u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta] \to u[y, v[x, w, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})]$ avec $x \in \Phi \setminus \Lambda$. La dérivation de typage de $u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta]$ se termine par

$$\frac{\Pi_1, \Pi_3, \Pi_5, \Pi_6, \Pi_7 \vdash w : C \qquad \dfrac{\Pi_1, \Pi_2, \Pi''_2, \Pi_6, x : C \vdash v : B \quad \Pi_1, \Pi_2, \Pi'_2, \Pi_5, y : B \vdash u : A}{\Pi_1, \Pi_2, \Pi'_2, \Pi''_2, \Pi'''_2, \Pi_5, \Pi_6, \Pi_7, x : C \vdash u[y, v, \Lambda, \Phi] : A}}{\Pi_1, \Pi_2, \Pi'_2, \Pi''_2, \Pi'''_2, \Pi_3, \Pi_4, \Pi_5, \Pi_6, \Pi_7 \vdash u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta] : A}$$

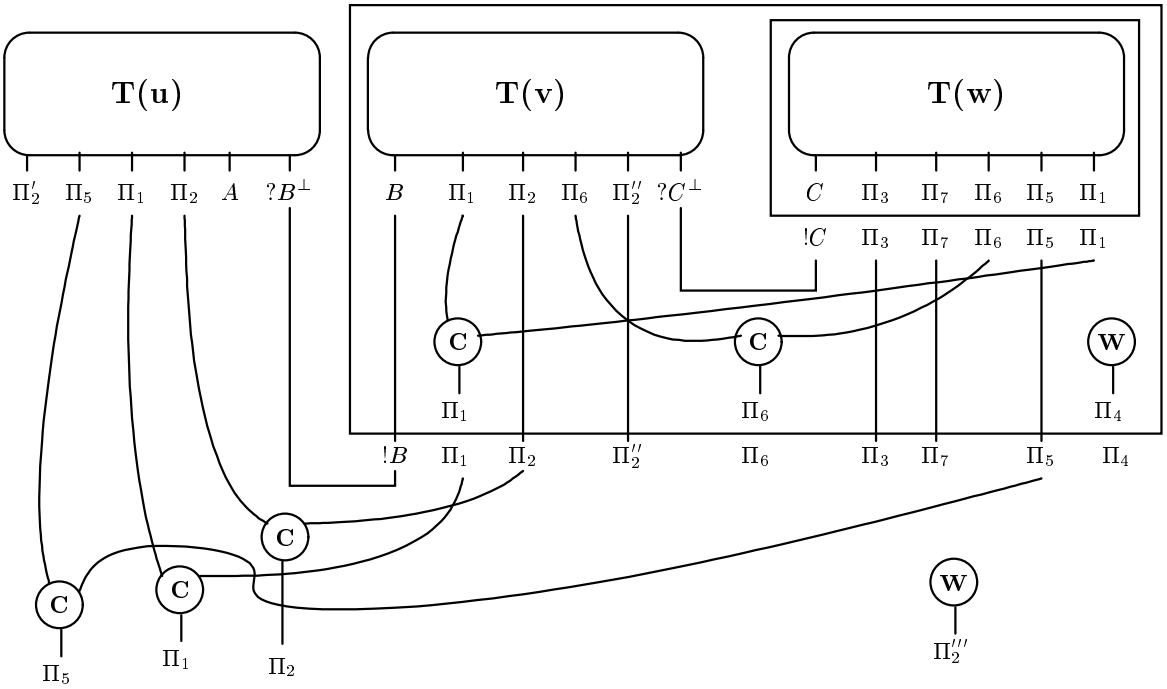
où $\Gamma = \Pi_2 \cup \Pi'_2 \cup \Pi''_2 \cup \Pi'''_2 \cup \Pi_4$, $\Delta = \Pi_3 \cup \Pi_4$, $\Lambda = \Pi'_2 \cup \Pi'''_2 \cup \Pi_5 \cup \Pi_7$ et $\Phi = \Pi''_2 \cup \Pi'''_2 \cup \Pi_6 \cup \Pi_7 \cup \{x\}$. Sa traduction est le réseau suivant



La dérivation de typage de $u[y, v[x, w, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})]$ se termine par

$$\dfrac{\dfrac{\Pi_1, \Pi_3, \Pi_5, \Pi_6, \Pi_7 \vdash w : C \quad \Pi_1, \Pi_2, \Pi''_2, \Pi_6, x : C \vdash v : B}{\Pi_1, \Pi_2, \Pi''_2, \Pi_3, \Pi_4, \Pi_5, \Pi_6, \Pi_7 \vdash v[x, w, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] : B} \quad \Pi_1, \Pi_2, \Pi'_2, \Pi_5, y : B \vdash u : A}{\Pi_1, \Pi_2, \Pi'_2, \Pi''_2, \Pi'''_2, \Pi_3, \Pi_4, \Pi_5, \Pi_6, \Pi_7 \vdash u[y, v[x, w, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})] : A}$$

Sa traduction est le réseau suivant

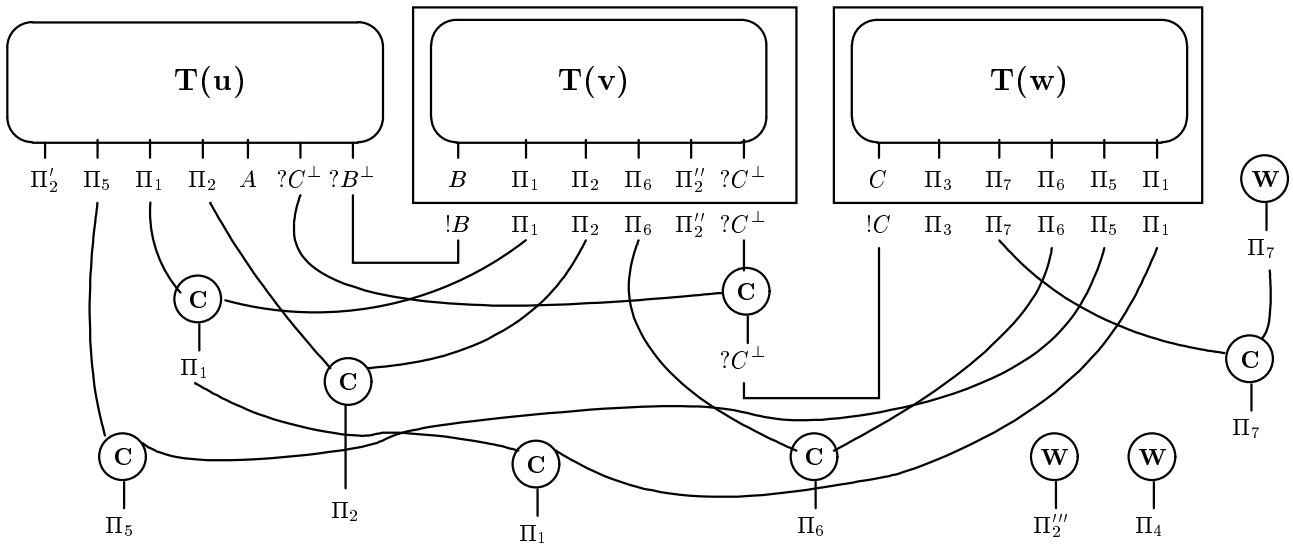


Puisqu'on travaille sur les $\mathcal{W}$-formes normales de ces réseaux, il suffit, pour passer du premier au second, d'éliminer la coupure $b - b$, puis d'utiliser les relations d'équivalence A et B.

- **règle** c_2 **:** $u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta] \to u[x, w, (\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y, v[x, w, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi]$ avec $x \notin \Phi \cup \Lambda$. La dérivation de typage de $u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta]$ se termine par

$$\dfrac{\Pi_1, \Pi_3, \Pi_5, \Pi_6, \Pi_7 \vdash w : C \qquad \dfrac{\Pi_1, \Pi_2, \Pi''_2, \Pi_6, x : C \vdash v : B \quad \Pi_1, \Pi_2, \Pi'_2, \Pi_5, y : B, x : C \vdash u : A}{\Pi_1, \Pi_2, \Pi'_2, \Pi''_2, \Pi'''_2, \Pi_5, \Pi_6, \Pi_7, x : C \vdash u[y, v, \Lambda, \Phi] : A}}{\Pi_1, \Pi_2, \Pi'_2, \Pi''_2, \Pi'''_2, \Pi_3, \Pi_4, \Pi_5, \Pi_6, \Pi_7 \vdash u[y, v, \Lambda, \Phi][x, w, \Gamma, \Delta] : A}$$

où $\Gamma = \Pi_2 \cup \Pi_2' \cup \Pi_2'' \cup \Pi_2''' \cup \Pi_4$, $\Delta = \Pi_3 \cup \Pi_4$, $\Lambda = \Pi_2' \cup \Pi_2''' \cup \Pi_5 \cup \Pi_7$ et $\Phi = \Pi_2'' \cup \Pi_2''' \cup \Pi_6 \cup \Pi_7$. Sa traduction est le réseau suivant



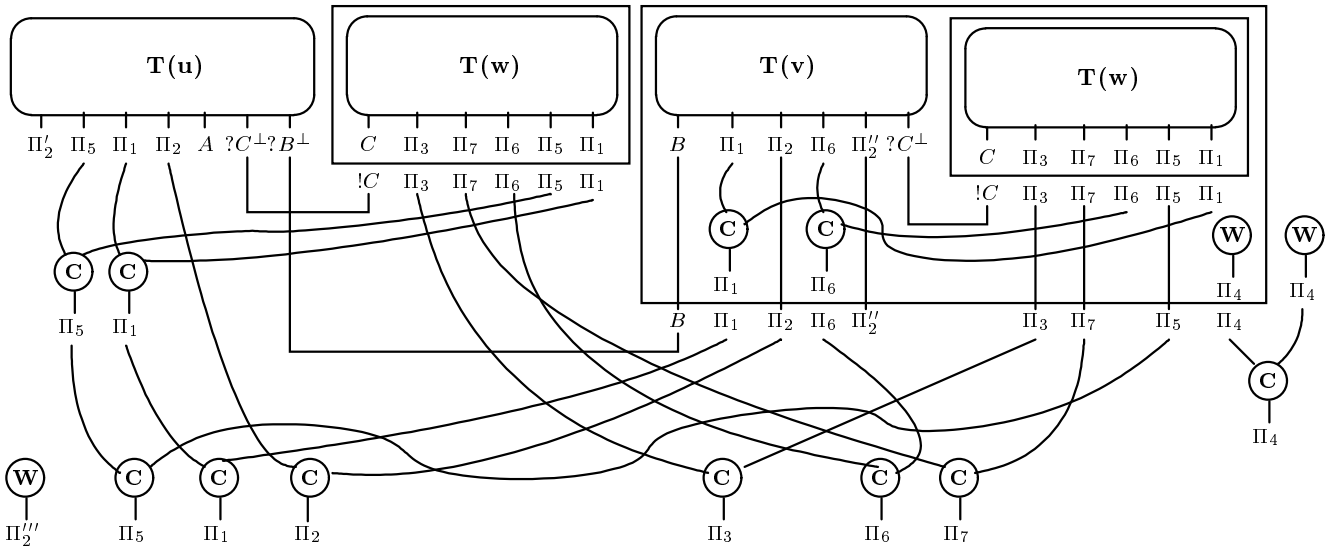
La dérivation de typage de $u[x, w, (\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y, v[x, w, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi]$ (qu'on appelle t) se termine par

$$\dfrac{\dfrac{\Pi_1, \Pi_3, \Pi_5, \Pi_6, \Pi_7 \vdash w : C \quad \Pi_1, \Pi_2, \Pi_2'', \Pi_6, x : C \vdash v : B}{\Pi_1, \Pi_2, \Pi_2'', \Pi_3, \Pi_4, \Pi_5, \Pi_6, \Pi_7 \vdash v[x, w, \Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)] : B} \quad \Pi_1, \Pi_2, \Pi_2', \Pi_3, \Pi_4, \Pi_5, \Pi_6, \Pi_7, y : B \vdash t' : A}{\Pi_1, \Pi_2, \Pi_2', \Pi_2'', \Pi_2''', \Pi_3, \Pi_4, \Pi_5, \Pi_6, \Pi_7 \vdash t : A}$$

où $t' = u[x, w, (\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)]$, dont la dérivation de typage se termine par

$$\dfrac{\Pi_1, \Pi_3, \Pi_5, \Pi_6, \Pi_7 \vdash w : C \quad \Pi_1, \Pi_2, \Pi_2', \Pi_5, y : B, x : C \vdash u : A}{\Pi_1, \Pi_2, \Pi_2', \Pi_3, \Pi_4, \Pi_5, \Pi_6, \Pi_7, y : B \vdash t' : A}$$

Sa traduction est le réseau suivant

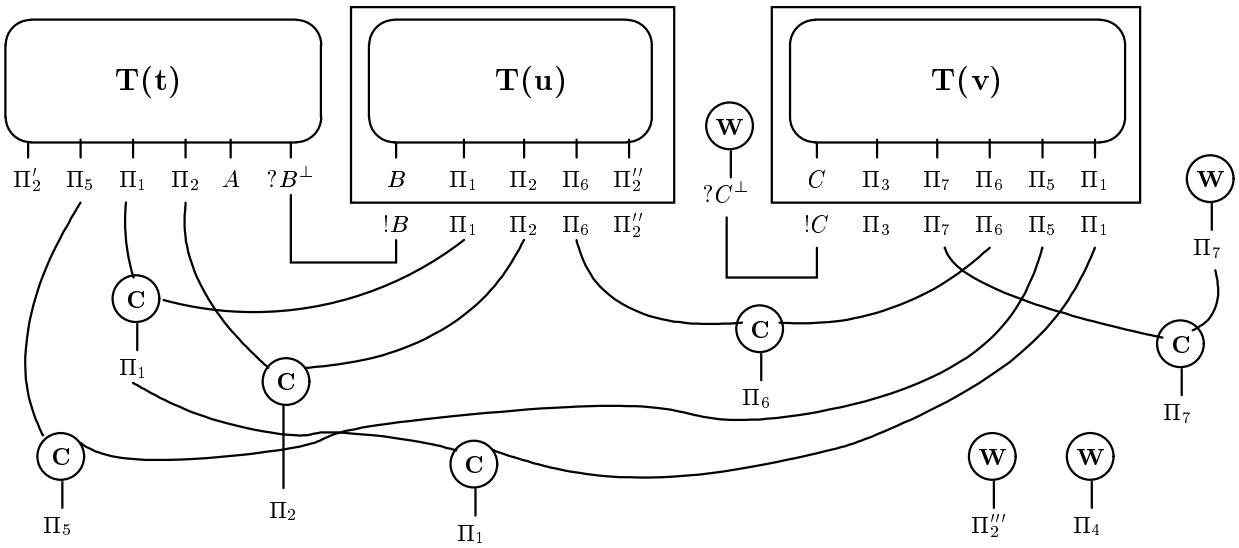


Puisqu'on travaille sur les $\mathcal{W}$-formes normales de ces réseaux, il suffit, pour passer du premier au second, d'éliminer la coupure $c - b$, puis la coupure $b - b$, et d'appliquer les règles d'équivalence A et B.

- **règle** c_3 **:** $t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] \to t[y,u,(\Lambda\setminus\{x\})\cup\Delta,(\Phi\setminus\{x\})\cup\Delta]$ avec $x\in\Lambda\cap\Phi$. La dérivation de typage de $t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta]$ se termine par

$$\dfrac{\Pi_1,\Pi_3,\Pi_5,\Pi_6,\Pi_7\vdash v:C \qquad \dfrac{\Pi_1,\Pi_2,\Pi_2'',\Pi_6\vdash u:B \quad \Pi_1,\Pi_2,\Pi_2',\Pi_5,y:B\vdash t:A}{\Pi_1,\Pi_2,\Pi_2',\Pi_2'',\Pi_2''',\Pi_5,\Pi_6,\Pi_7,x:C\vdash t[y,u,\Lambda,\Phi]:A}}{\Pi_1,\Pi_2,\Pi_2',\Pi_2'',\Pi_2''',\Pi_3,\Pi_4,\Pi_5,\Pi_6,\Pi_7\vdash t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta]:A}$$

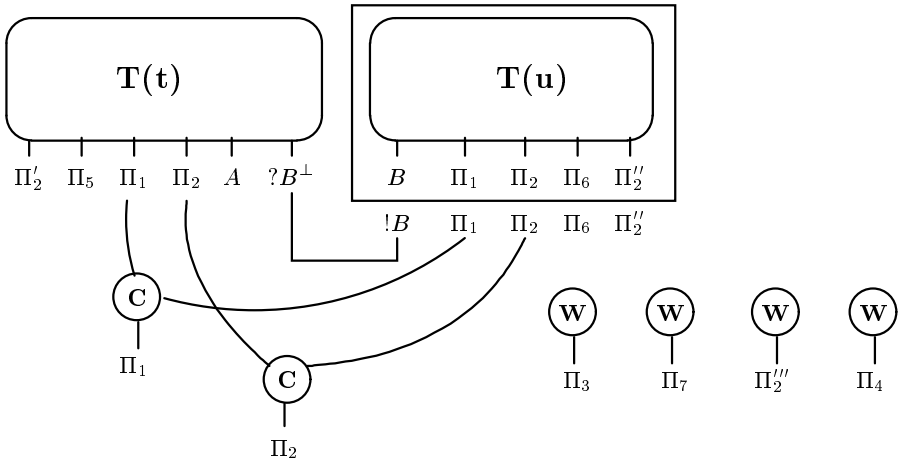
où $\Gamma=\Pi_2\cup\Pi_2'\cup\Pi_2''\cup\Pi_2''\cup\Pi_4$, $\Delta=\Pi_3\cup\Pi_4$, $\Lambda=\Pi_2'\cup\Pi_2'''\cup\Pi_5\cup\Pi_7\cup\{x\}$ et $\Phi=\Pi_2''\cup\Pi_2'''\cup\Pi_6\cup\Pi_7\cup\{x\}$. Sa traduction est le réseau suivant



La dérivation de typage de $t[y,u,(\Lambda\setminus\{x\})\cup\Delta,(\Phi\setminus\{x\})\cup\Delta]$ se termine par

$$\dfrac{\Pi_1,\Pi_2,\Pi_2'',\Pi_6\vdash u:B \quad \Pi_1,\Pi_2,\Pi_2',\Pi_5,y:B\vdash t:A}{\Pi_1,\Pi_2,\Pi_2',\Pi_2'',\Pi_2''',\Pi_3,\Pi_4,\Pi_5,\Pi_6,\Pi_7\vdash t[y,u,(\Lambda\setminus\{x\})\cup\Delta,(\Phi\setminus\{x\})\cup\Delta]:A}$$

Sa traduction est le réseau suivant

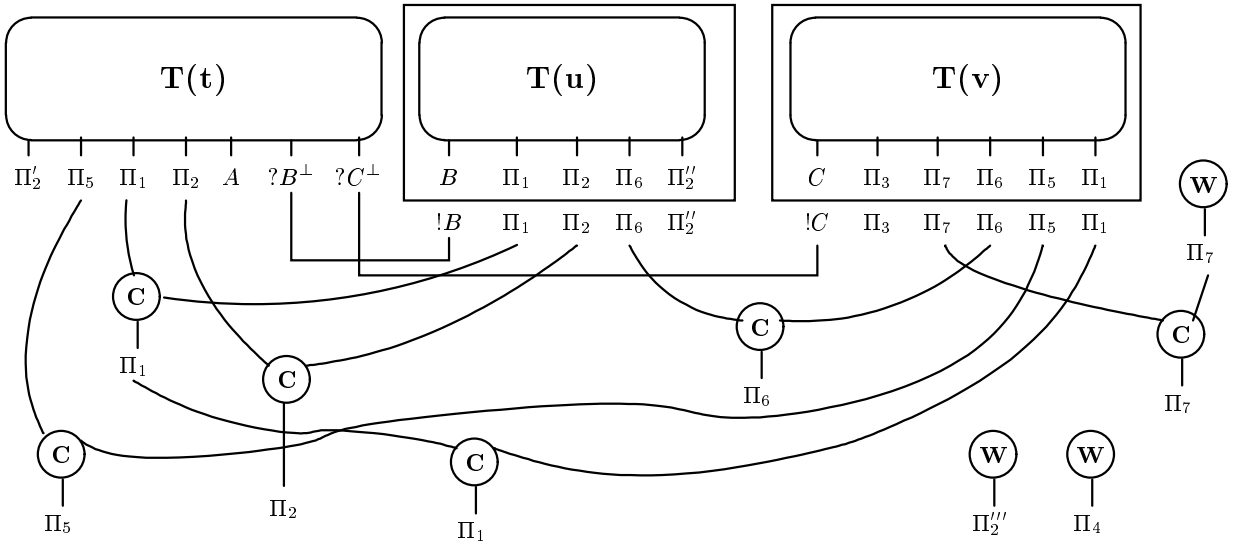


Pour réduire la $\mathcal{W}$-forme normale du premier réseau vers celle du second, nous devons d'abord éliminer la coupure $w-b$, puis il suffit, pour terminer, d'appliquer quatre fois la règle wc.

- **équivalence** c_4 **:** $t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta]\sim t[x,v,(\Gamma\setminus\Phi)\cup\{y\},\Delta\cup(\Phi\setminus\Gamma)][y,u,(\Lambda\setminus\{x\})\cup\Delta,\Phi\cap\Gamma]$ avec $x\in\Lambda\setminus\Phi$. La dérivation de typage de $t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta]$ se termine par

$$\dfrac{\Pi_1,\Pi_3,\Pi_5,\Pi_6,\Pi_7\vdash v:C \qquad \dfrac{\Pi_1,\Pi_2,\Pi_2'',\Pi_6\vdash u:B \quad \Pi_1,\Pi_2,\Pi_2',\Pi_5,y:B,x:C\vdash t:A}{\Pi_1,\Pi_2,\Pi_2',\Pi_2'',\Pi_2''',\Pi_5,\Pi_6,\Pi_7,x:C\vdash t[y,u,\Lambda,\Phi]:A}}{\Pi_1,\Pi_2,\Pi_2',\Pi_2'',\Pi_2''',\Pi_3,\Pi_4,\Pi_5,\Pi_6,\Pi_7\vdash t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta]:A}$$

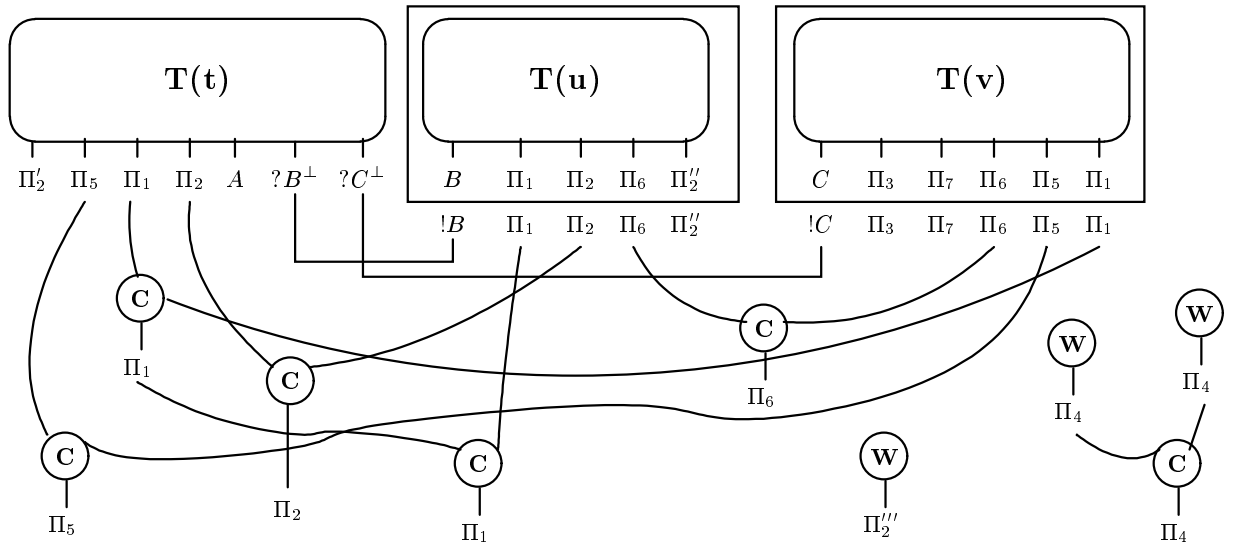
où $\Gamma = \Pi_2 \cup \Pi_2' \cup \Pi_2'' \cup \Pi_2''' \cup \Pi_4$, $\Delta = \Pi_3 \cup \Pi_4$, $\Lambda = \Pi_2' \cup \Pi_2''' \cup \Pi_5 \cup \Pi_7 \cup \{x\}$ et $\Phi = \Pi_2'' \cup \Pi_2''' \cup \Pi_6 \cup \Pi_7$. Sa traduction est le réseau suivant



La dérivation de typage de $t[x, v, (\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y, u, (\Lambda \setminus \{x\}) \cup \Delta, \Phi \cap \Gamma]$ se termine par

$$\dfrac{\dfrac{\Pi_1, \Pi_2, \Pi_2'', \Pi_6 \vdash u : B \quad \dfrac{\Pi_1, \Pi_3, \Pi_5, \Pi_6, \Pi_7 \vdash v : C \quad \Pi_1, \Pi_2, \Pi_2', \Pi_5, y : B, x : C \vdash t : A}{\Pi_1, \Pi_2, \Pi_2', \Pi_3, \Pi_4, \Pi_5, \Pi_6, \Pi_7, y : B \vdash t[x, v, (\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)] : A}}{\Pi_1, \Pi_2, \Pi_2', \Pi_2'', \Pi_2''', \Pi_3, \Pi_4, \Pi_5, \Pi_6, \Pi_7 \vdash t[x, v, (\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y, u, (\Lambda \setminus \{x\}) \cup \Delta, \Phi \cap \Gamma] : A}$$

Sa traduction est le réseau suivant

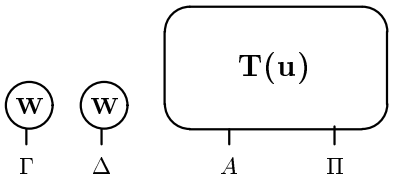


Et les deux $\mathcal{W}$-formes normales des réseaux sont équivalentes modulo l'équivalence A.

- **équivalence** d **:** $\Gamma\Delta u \sim (\Gamma \cup \Delta)u$. La dérivation de typage de $\Gamma\Delta u$ se termine par

$$\dfrac{\dfrac{\Pi \vdash u : C}{\Pi, \Delta \vdash \Delta u : C}}{\Pi, \Gamma, \Delta \vdash \Gamma\Delta u : C}$$

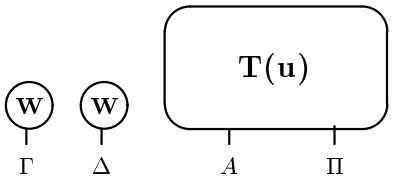
Sa traduction est le réseau suivant



La dérivation de typage de $(\Gamma \cup \Delta)u$ se termine par

$$\frac{\Pi \vdash u : C}{\Pi, \Gamma, \Delta \vdash (\Gamma \cup \Delta)u : C}$$

Sa traduction est le réseau suivant

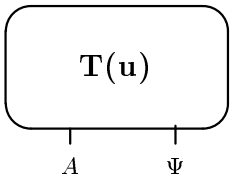


Et les deux réseaux sont identiques.

- **règle $\emptyset$:** $\emptyset u \sim u$. La dérivation de typage de $\emptyset u$ se termine par

$$\frac{\Psi \vdash u : C}{\Psi \vdash \emptyset u : C}$$

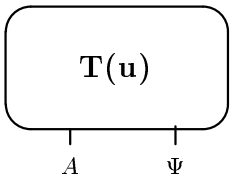
Sa traduction est le réseau suivant



La dérivation de typage de u se termine par

$$\Psi \vdash u : C$$

Sa traduction est le réseau suivant



Et les deux réseaux sont identiques.

■

Voici enfin le théorème principal de ce chapitre.

Théorème 17.2.2 (Normalisation forte de λ_{wsn})

Le λ_{wsn}-calcul typé est fortement normalisant.

Preuve : On prouve par l'absurde. Supposons qu'il existe une dérivation infinie à partir d'un terme t de λ_{wsn}. En utilisant le lemme 17.2.1, on peut construire à partir de $T(t)$ une dérivation infinie dans les réseaux de preuves $\mathcal{R}_E$, ce qui contredit le théorème 16.2.10. ■

Chapitre 18

Pour aller plus loin

Dans ce chapitre, nous introduisons deux autres versions du λ_{wsn}-calcul, basées sur un calcul de remontée des affaiblissements explicites. Ici encore, nous nous contentons de modifier les règles de réduction et la notion de calcul sans toucher à la syntaxe des termes. Après avoir donné les motivations et la définition du calcul de remontée, nous présentons une première version modifiée du λ_{wsn}-calcul : le $\lambda_{wsn\uparrow}$-calcul. Pour pouvoir réellement aller plus loin, nous modifions la notion de réduction du $\lambda_{wsn\uparrow}$-calcul en proposant la définition du $\lambda_{wsn\to c}$-calcul, puis nous étudions les propriétés de ce dernier.

18.1 La remontée des affaiblissements explicites

18.1.1 Motivations

Simulation

Nous avons déjà mentionné, dans le chapitre précédent, la première motivation à la remontée des substitutions : c'est l'obligation que nous avons eue de travailler sur les $\mathcal{W}$-formes normales dans la simulation. Si on observe bien les endroits où cela est nécessaire, on constate que son origine réside dans le fait que dans une substitution $[x, u, \Gamma, \Delta]$ l'intersection de Γ et Δ peut ne pas être vide. Cette intersection apporte la présence de nœuds *Weakening* qui ne se comportent pas correctement dans les réseaux : ils apparaissent soudainement au fil des réductions.

Regardons ce que signifie le fait d'avoir $\Gamma \cap \Delta \neq \emptyset$. Le principal moment où cela nous sert est la règle c_3 :

$$t[y, u, \Lambda, \Phi][x, v, \Gamma, \Delta] \to t[y, u, (\Lambda \setminus \{x\}) \cup \Delta, (\Phi \setminus \{x\}) \cup \Delta]$$

Pour utiliser cette règle, la condition de bord $x \in \Phi \cap \Lambda$ fait explicitement appel à cette intersection. La règle nous permet d'effacer une substitution lorsque sa variable apparaît à la fois dans Λ et Φ, les protecteurs respectifs de t et de u. Comme nous l'avons dit dans l'introduction du λ_{wsn}-calcul, cela correspond au fait que la substitution sur x ne doit aller ni dans t, ni dans u.

Cela ressemble fort à une construction de nos termes déjà existante : l'affaiblissement explicite. En effet, la règle e_1 effectue le même travail que la règle c_3, regardons-les ensembles :

$$\begin{array}{lcll} t[y, u, \Lambda, \Phi][x, v, \Gamma, \Delta] & \to & t[y, u, (\Lambda \setminus \{x\}) \cup \Delta, (\Phi \setminus \{x\}) \cup \Delta] & x \in \Phi \cap \Lambda \\ (\Lambda t)[x, u, \Gamma, \Delta] & \to & (\Delta \cup (\Lambda \setminus \{x\}))t & x \in \Lambda \end{array}$$

Certaines similitudes nous frappent, comme l'expression $(\Delta \cup (\Lambda \setminus \{x\}))$, dans le membre droit de la règle e_1, qui apparaît aussi dans le membre droit de la règle c_3. En fait, la règle c_3 est une règle e_1 déguisée. Dans un terme $t[x, u, \Gamma, \Delta]$ l'intersection de Γ et Δ contient des variables qui ne sont libres ni dans t, ni dans u, ce qui correspond à un affaiblissement explicite sur t et u. Soit $\Pi = \Gamma \cap \Delta$. Le terme $t[x, u, \Gamma, \Delta]$ est équivalent à $\Pi(t[x, u, \Gamma \setminus \Pi, \Delta \setminus \Pi])$. On a regroupé les variables de l'intersection, puis on les a mises dans un affaiblissement explicite à l'extérieur du terme. Reprenons le membre

gauche de la règle c_3 et appliquons ce principe pour la substitution intérieure ; cela nous donne le terme suivant, dans lequel $\Pi = \Lambda \cap \Phi$:

$$(\Pi(t[y, u, \Lambda \setminus \Pi, \Phi \setminus \Pi]))[x, v, \Gamma, \Delta]$$

Dans ce terme, x est dans Π puisqu'il est dans $\Lambda \cap \Phi$. On peut donc réduire ce terme avec la règle e_1, ce qui nous donne :

$$(\Pi(t[y, u, \Lambda \setminus \Pi, \Phi \setminus \Pi]))[x, v, \Gamma, \Delta] \to (\Delta \cup (\Pi \setminus \{x\}))(t[y, u, \Lambda \setminus \Pi, \Phi \setminus \Pi])$$

Ce dernier terme correspond exactement à la transformation du membre droit de la règle c_3.

Il y a une autre règle qui utilise la $\mathcal{W}$-forme normale pour la simulation, c'est la règle a :

$$(\Delta(\lambda x.t))\ (\Gamma u) \to t[x, u, \Gamma, \Delta]$$

Cela ne nous surprend pas, puisque là encore on est en présence d'une intersection non vide possible entre Γ et Δ. On fait la même remarque que précédemment : l'intersection de Γ et Δ correspond à une étiquette qu'on pourrait placer au dessus de l'application. Ainsi, un terme $(\Gamma t)\ (\Delta u)$ avec $\Pi = \Gamma \cap \Delta$ a le même sens que le terme $\Pi((\Gamma \setminus \Pi)t\ (\Delta \setminus \Pi)u)$.

Élimination précoce des substitutions

On peut donner des règles de réécriture qui formalisent les transformations ci-dessus :

$$\begin{array}{lrcl} (\uparrow\text{-}sub\text{-}e) & t[x, u, \Gamma, \Delta] & \to & (\Gamma \cap \Delta)(t[x, u, \Gamma \setminus \Delta, \Delta \setminus \Gamma]) \\ (\uparrow\text{-}app) & (\Gamma t)\ (\Delta u) & \to & (\Gamma \cap \Delta)((\Gamma \setminus \Delta)t\ (\Delta \setminus \Gamma)u) \end{array}$$

Regardons ce qui se passe si une substitution vient rencontrer une application. Prenons le terme $((\Gamma t)\ (\Delta u))[x, v, \lambda, \Phi]$ en supposant que x soit à la fois dans Γ et dans Δ : la substitution va être effacée. Dans λ_{wsn}, on a la réduction suivante :

$$\begin{array}{c} ((\Gamma t)\ (\Delta u))[x, v, \Lambda, \Phi] \\ \downarrow a \\ (\Gamma t)[x, v, \Lambda, \Phi]\ (\Delta u)[x, v, \Lambda, \Phi] \\ \downarrow e_1 \\ (\Phi \cup (\Gamma \setminus \{x\}))t\ (\Delta u)[x, v, \Lambda, \Phi] \\ \downarrow e_1 \\ (\Phi \cup (\Gamma \setminus \{x\}))t\ (\Phi \cup (\Delta \setminus \{x\}))u \end{array}$$

Si on utilise la transformation du terme $(\Gamma t)\ (\Delta u)$ par la règle $\uparrow$-app, on obtient la réduction suivante :

$$\begin{array}{c} ((\Gamma \cap \Delta)((\Gamma \setminus \Delta)t\ (\Delta \setminus \Gamma)u))[x, v, \lambda, \Phi] \\ \downarrow a \\ (\Phi \cup ((\Gamma \cap \Delta) \setminus \{x\}))((\Gamma \setminus \Delta)t\ (\Delta \setminus \Gamma)u) \end{array}$$

Et ce dernier terme correspond à la transformation du dernier terme de la réduction précédente. Le fait marquant est que nous avons gagné deux étapes de réduction. Le fait d'avoir remonté l'information sur l'affaiblissement de x nous a permis d'éliminer plus tôt la substitution. On en déduit que plus on fait remonter les affaiblissements, plus on pourra gagner d'étapes de réduction. La formalisation complète de la remontée des affablissements explicites, associée aux règles du λ_{wsn}-calcul, nous mène à la définition du $\lambda_{wsn\uparrow}$-calcul.

18.1.2 Définition et propriétés du $\lambda_{wsn\uparrow}$-calcul

La remontée des étiquettes est possible dans les circonstances suivantes.

- Si l'étiquette est sous une application, nous avons déjà donné la règle qui en permet la remontée :

$$(\uparrow\text{-}app)\quad (\Gamma t)\ (\Delta u)\quad \to\quad (\Gamma \cap \Delta)((\Gamma \setminus \Delta)t\ (\Delta \setminus \Gamma)u)$$

- Si l'étiquette est dans la partie droite d'une substitution, la règle, déjà mentionnée, est :

$$(\uparrow\text{-}sub\text{-}e)\quad t[x, u, \Gamma, \Delta]\quad \to\quad (\Gamma \cap \Delta)(t[x, u, \Gamma \setminus \Delta, \Delta \setminus \Gamma])$$

- Si l'étiquette est dans le terme substituant, on peut la faire remonter dans la partie de la substitution qui "protège" celui-ci :

$$(\uparrow\text{-}sub\text{-}d)\quad t[x, \Phi u, \Gamma, \Delta]\quad \to\quad t[x, u, \Phi \cup \Gamma, \Delta]$$

- De la même façon, si l'étiquette est dans le terme auquel est appliquée la substitution, on peut la faire remonter, sauf l'affaiblissement sur x s'il y en a un :

$$(\uparrow\text{-}sub\text{-}g)\quad (\Phi t)[x, u, \Gamma, \Delta]\quad \to\quad ((\Phi \cap \{x\})t)[x, u, \Gamma, (\Phi \setminus \{x\}) \cup \Delta]$$

- Si l'étiquette est sous un λ, on peut la faire remonter, excepté l'affaiblissement sur x s'il y en a un :

$$(\uparrow\text{-}\lambda)\quad \lambda x.(\Phi t)\quad \to\quad (\Phi \setminus \{x\})(\lambda x.((\Phi \cap \{x\})t))$$

- Enfin, si l'étiquette est sous une autre étiquette, il suffit d'employer l'équivalence d.

On peut regrouper les règles $\uparrow$-sub-e, $\uparrow$-sub-g et $\uparrow$-sub-d pour faire l'unique règle suivante, dont les précédentes sont dérivables en utilisant l'équivalence $\emptyset$:

$$(\uparrow\text{-}sub)\quad (\Phi t)[x, \Psi u, \Gamma, \Delta]\quad \to$$
$$((\Psi \cup \Gamma) \cap ((\Phi \setminus \{x\}) \cup \Delta))(((\Phi \cap \{x\})t)[x, u, (\Psi \cup \Gamma) \setminus ((\Phi \setminus \{x\}) \cup \Delta), ((\Phi \setminus \{x\}) \cup \Delta) \setminus (\Psi \cup \Gamma)])$$

Pour toutes ces règles, on donne comme condition que les ensembles considérés ne doivent pas être vides, ou ne pas avoir une intersection vide. Sinon, cela nous premettrait de faire une réduction infinie avec la seule règle $\uparrow$-app :

$$(\emptyset t\ \emptyset u)\quad \to_{\uparrow\text{-}app}\quad \emptyset(\emptyset t\ \emptyset u)$$

La figure 18.1 donne la définition des règles de réduction du $\lambda_{wsn\uparrow}$-calcul. On constate la disparition de l'équivalence e_2 qui est un sous-cas de la règle $\uparrow$-sub. Voici quelques propriétés, qui ne sont encore que des conjectures mais pour lesquelles nous donnons des pistes pour les prouver. Le calcul composé des règles $\uparrow$-app, $\uparrow$-λ et $\uparrow$-sub, modulo les équivalences, est appelé $\uparrow$-calcul, l'ensemble du reste des règles sera noté $\neg\uparrow$.

Conjecture 18.1.1 (Normalisation forte du $\uparrow$-calcul)

Le $\uparrow$-calcul est fortement normalisant.

Preuve : Il faut considérer la distance entre les étiquettes et les lieurs correspondants (ou le sommet du terme pour les variables libres). Cette distance devrait décroître strictement à chaque étape. ■

Conjecture 18.1.2 (Préservation du typage du $\uparrow$-calcul)

Le $\uparrow$-calcul préserve le type des termes.

Preuve : La preuve devrait être immédiate par cas. ■

$$
\begin{array}{lrcll}
(b) & (\Delta(\lambda x.t))\ (\Gamma u) & \to & t[x,u,\Gamma,\Delta] & \\
\\
(a) & (t\ u)[x,v,\Gamma,\Delta] & \to & (t[x,v,\Gamma,\Delta]\ u[x,v,\Gamma,\Delta]) & \\
(e_1) & (\Lambda t)[x,u,\Gamma,\Delta] & \to & (\Delta \cup (\Lambda \setminus \{x\}))t & x \in \Lambda \\
(n_1) & y[x,t,\Gamma,\Delta] & \to & \Delta y & x \neq y \\
(n_2) & x[x,t,\Gamma,\Delta] & \to & \Gamma t & \\
(c_1) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & & x \in \Phi \setminus \Lambda \\
 & & & t[y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})] & \\
(c_2) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)] & \\
 & & & [y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi] & x \notin \Phi \cup \Lambda \\
\\
(\uparrow\text{-}app) & (\Gamma t)\ (\Delta u) & \to & (\Gamma \cap \Delta)((\Gamma \setminus \Delta)t\ (\Delta \setminus \Gamma)u) & \Gamma \cap \Delta \neq \emptyset \\
(\uparrow\text{-}\lambda) & \lambda x.(\Phi t) & \to & (\Phi \setminus \{x\})(\lambda x.((\Phi \cap \{x\})t)) & \Phi \neq \{x\} \text{ et } \Phi \neq \emptyset \\
(\uparrow\text{-}sub) & (\Phi t)[x,\Psi u,\Gamma,\Delta] & \to & & \Phi \neq \{x\} \text{ et } \Phi \neq \emptyset \\
 & & & ((\Psi \cup \Gamma) \cap ((\Phi \setminus \{x\}) \cup \Delta)) & \text{ou } \Psi \neq \emptyset \text{ ou} \\
 & & & (((\Phi \cap \{x\})t) & \Gamma \cap \Delta \neq \emptyset \\
 & & & [x,u,(\Psi \cup \Gamma) \setminus ((\Phi \setminus \{x\}) \cup \Delta), ((\Phi \setminus \{x\}) \cup \Delta) \setminus (\Psi \cup \Gamma)]) & \\
\\
(f) & (\lambda y.t)[x,u,\Gamma,\Delta] & \sim & \lambda y.t[x,u,\Gamma \cup \{y\},\Delta] & \\
(d) & \Gamma\Delta t & \sim & (\Gamma \cup \Delta)t & \\
(\emptyset) & \emptyset t & \sim & t & \\
\\
(c_4) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \sim & & x \in \Lambda \setminus \Phi \\
 & t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y,u,\Delta \cup (\Lambda \setminus \{x\}), \Gamma \cap \Phi] & & &
\end{array}
$$

FIG. 18.1 – Règles de réduction du $\lambda_{wsn\uparrow}$-calcul

Conjecture 18.1.3 (Normalisation forte du $\lambda_{wsn\uparrow}$-calcul)

Le $\lambda_{wsn\uparrow}$-calcul typé est fortement normalisant.

Preuve : La simulation utilisée dans le chapitre précédent devrait s'appliquer directement. ■

18.2 Vers la contraction explicite

Stratégie de réduction

Le système que nous avons présenté ne nous satisfait pas encore. En effet, même si nous avons ajouté un calcul pour éliminer plus rapidement les substitutions sur des affaiblissements explicites, nous n'avons pas pour autant résolu le problème qui était notre première motivation. Le fait d'ajouter les règles du $\uparrow$-calcul n'empêche pas que l'intersection des étiquettes d'une substitution ne soit pas vide. Pour cela, il faut *imposer* l'utilisation de $\uparrow$ à chaque étape du calcul.

Nous allons donc nous fixer une stratégie de réduction qui modifie la notion de calcul que nous avions auparavant ; voici sa définition.

Définition 18.2.1 (Stratégie $\uparrow$)

On dit que $t \to t'$ s'il existe t_1 et t_2 tels que :

$$t \ \sim \ t_1 \ \to_{\neg\uparrow} \ t_2 \ \to_{\uparrow}^{*} \ \uparrow(t_3)$$

Où $\uparrow(t)$ dénote la $\uparrow$-forme normale de t.

Après l'application d'une règle de réduction, nous prenons la ↑-forme normale du terme obtenu. Cette stratégie peut aussi s'écrire en rendant le système ↑ implicite dans le calcul. Nous pouvons déjà proposer une première propriété.

Conjecture 18.2.2 (Séparation des étiquettes)

> Les ↑-formes normales sont données par la grammaire suivante :
>
> $$\begin{array}{lll} u & ::= & x \mid \lambda x.u \mid \lambda x.(\{x\}u) \mid (u\ u) \mid (\Gamma u\ u) \mid (u\ \Gamma u) \mid (\Gamma u\ \Delta u) \mid u[x,u,\Gamma,\Delta] \mid (\{x\}u)[x,u,\Gamma,\Delta] \\ t & ::= & \Gamma u \end{array}$$
>
> Avec $\Gamma \cap \Delta = \emptyset$ et, dans les termes $\lambda x.(\{x\}u)$ et $(\{x\}u)[x,u,\Gamma,\Delta]$, le fait que la variable x dans l'affaiblissement est celle qui est liée par l'abstraction ou par la substitution.

Preuve : On devrait prouver cela facilement par l'absurde en considérant chaque terme qui n'est pas dans cette forme et en montrant qu'il n'est alors pas en ↑-forme normale. ■

Ce calcul possède maintenant la propriété attendue : l'intersection des étiquettes des applications ou des substitutions est toujours vide. Nous n'aurons plus besoin de considérer les $\mathcal{W}$-formes normales pour effectuer la preuve de normalisation forte par traduction et simulation dans les réseaux de preuve. Cependant, on peut encore aller un peu plus loin.

Si on regarde bien la grammaire des ↑-formes normales, on constate que les affaiblissements explicites apparaissent à certains endroits précis : sous une application, sous une substitution, et sous une abstraction. Dans ce dernier cas, comme dans le cas de l'étiquette $\{x\}$ sous la substitution, cela correspond à la déliaison de la variable x. Ce sont les autres cas qui nous intéressent à présent.

Prenons le terme $(\Gamma t\ \Delta u)$: la partie des affaiblissements explicites qui était commune aux deux sous-termes a déjà été remontée, et il reste seulement les parties qui portent sur des variables différentes. Cette partie commune et cette partie différente correspondent à une information particulière dans le typage des termes : c'est la règle de contraction.

Étudions de plus près cette apparition de la contraction dans la forme des termes. Dans le terme $(\Gamma t\ \Delta u)[x,v,\Phi,\Psi]$, puisque $\Gamma \cap \Delta = \emptyset$ nous avons trois possibilités :

- Soit x est dans Γ, auquel cas la substitution ne doit aller que dans u.
- Soit x est dans Δ, auquel cas la substitution ne doit aller que dans t.
- Soit x n'est pas dans $\Gamma \cup \Delta$, auquel cas la substitution doit aller à la fois dans t et dans u.

Ces trois cas ressemblent très fortement aux trois cas de la composition. En donnant trois règles pour gérer la rencontre d'une substitution avec une application, on rend explicite, dans les règles, la contraction qui a eu lieu dans le typage. Pour que cela fonctionne complètement, il faut essayer d'avoir le plus d'affaiblissements possibles dans les termes. Dans ce but, nous modifions les règles de typage afin d'obliger l'apparition de tout les affaiblissements explicites, ce qui nous demande de modifier la règle Ax. Cela nous donne le système de typage suivant.

$$\frac{}{x : A \vdash x : A}\ Ax \qquad \frac{\Gamma \vdash t : A \quad \Gamma \cap \Delta = \emptyset}{\Gamma, \Delta \vdash \Delta t : A}\ Weak$$

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

$$\frac{\Pi \setminus \Gamma \vdash u : A \quad \Pi \setminus \Delta, x : A \vdash t : B \quad (\Gamma \cup \Delta) \subset \Pi}{\Pi \vdash t[x,u,\Gamma,\Delta] : B}\ Sub$$

Le fait d'avoir contraint le typage de la variable à n'avoir lieu que dans un environnement supplémentaire vide nous oblige à utiliser la règle $Weak$, ce qui fait apparaître les étiquettes. Ainsi, si

on typait x avec l'ancienne règle (à gauche), on doit maintenant typer cette variable avec deux règles (à droite) :

$$\frac{}{\Gamma, x : A \vdash x : A}\ Ax \qquad\qquad \frac{\dfrac{}{x : A \vdash x : A}\ Ax}{\Gamma, x : A \vdash \Gamma x : A}\ Weak$$

L'ensemble des règles de réduction du $\lambda_{wsn\to c}$-calcul est donné figure 18.2. On constate la disparition de la règle n_1, devenue inutile puisqu'on oblige les variables à être précédées d'affaiblissements sur toutes les autres variables : les substitutions doivent rencontrer des affaiblissements ou bien concerner la variable atteinte (conjecture 18.2.5).

$$\begin{array}{lrcll}
(b) & (\Delta(\lambda x.t))\ (\Gamma u) & \to & t[x,u,\Gamma,\Delta] & \\
(a_1) & (\Phi t\ \Psi u)[x,v,\Gamma,\Delta] & \to & & x \in \Phi \\
 & (((\Phi \setminus \{x\}) \cup \Delta)t\ (\Gamma \cap \Psi)(u[x,v,\Gamma \setminus \Psi, \Delta \cup (\Psi \setminus \Gamma)])) & & & \\
(a_2) & (\Phi t\ \Psi u)[x,v,\Gamma,\Delta] & \to & & x \notin \Phi \cup \Psi \\
 & ((\Gamma \cap \Phi)(t[x,v,\Gamma \setminus \Phi, \Delta \cup (\Phi \setminus \Gamma)])\ (\Gamma \cap \Psi)(u[x,v,\Gamma \setminus \Psi, \Delta \cup (\Psi \setminus \Gamma)])) & & & \\
(a_4) & (\Phi t\ \Psi u)[x,v,\Gamma,\Delta] & \to & & x \in \Psi \\
 & ((\Gamma \cap \Phi)(t[x,v,\Gamma \setminus \Phi, \Delta \cup (\Phi \setminus \Gamma)])\ ((\Psi \setminus \{x\}) \cup \Delta)u) & & & \\
(n) & x[x,t,\Gamma,\Delta] & \to & \Gamma t & \\
(c_1) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & & x \in \Phi \setminus \Lambda \\
 & t[y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Delta \cup (\Phi \setminus \{x\})] & & & \\
(c_2) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \to & t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)] & \\
 & & & [y,u[x,v,\Gamma \setminus \Lambda, \Delta \cup (\Lambda \setminus \Gamma)], \Lambda \cap \Gamma, \Gamma \cap \Phi] & x \notin \Phi \cup \Lambda \\
(\uparrow\text{-}app) & (\Gamma t)\ (\Delta u) & \to & (\Gamma \cap \Delta)((\Gamma \setminus \Delta)t\ (\Delta \setminus \Gamma)u) & \Gamma \cap \Delta \neq \emptyset \\
(\uparrow\text{-}\lambda) & \lambda x.(\Phi t) & \to & (\Phi \setminus \{x\})(\lambda x.((\Phi \cap \{x\})t)) & \Phi \neq \{x\} \text{ et } \Phi \neq \emptyset \\
(\uparrow\text{-}sub) & (\Phi t)[x,\Psi u,\Gamma,\Delta] & \to & & \Phi \neq \{x\} \text{ et } \Phi \neq \emptyset \\
 & ((\Psi \cup \Gamma) \cap ((\Phi \setminus \{x\}) \cup \Delta)) & & & \text{ou } \Psi \neq \emptyset \text{ ou} \\
 & (((\Phi \cap \{x\})t) & & & \Gamma \cap \Delta \neq \emptyset \\
 & [x,u,(\Psi \cup \Gamma) \setminus ((\Phi \setminus \{x\}) \cup \Delta), ((\Phi \setminus \{x\}) \cup \Delta) \setminus (\Psi \cup \Gamma)]) & & & \\
(f_1) & (\lambda y.t)[x,u,\Gamma,\Delta] & \sim & \lambda y.t[x,u,\Gamma \cup \{y\}, \Delta] & \\
(f_2) & (\lambda y.\{y\}t)[x,u,\Gamma,\Delta] & \sim & \lambda y.\{y\}(t[x,u,\Gamma \cup \{y\}, \Delta]) & \\
(d) & \Gamma\Delta t & \sim & (\Gamma \cup \Delta)t & \\
(\emptyset) & \emptyset t & \sim & t & \\
(c_4) & t[y,u,\Lambda,\Phi][x,v,\Gamma,\Delta] & \sim & & x \in \Lambda \setminus \Phi \\
 & t[x,v,(\Gamma \setminus \Phi) \cup \{y\}, \Delta \cup (\Phi \setminus \Gamma)][y,u,\Delta \cup (\Lambda \setminus \{x\}), \Gamma \cap \Phi] & & &
\end{array}$$

FIG. 18.2 – Règles de réduction du $\lambda_{wsn\to c}$-calcul

Propriétés

Là encore, nous ne pouvons donner que des conjectures. La principale conjecture concerne la normalisation forte.

Conjecture 18.2.3 (Normalisation forte du $\lambda_{wsn\to c}$-calcul)

Le $\lambda_{wsn\to c}$-calcul typé est fortement normalisant.

Preuve : La simulation utilisée dans le chapitre précédent devrait s'appliquer directement, sans avoir besoin de travailler sur les $\mathcal{W}$-formes normales. ■

Il serait sans doute possible de se passer de notre notion de réduction avec stratégie si les réductions du calcul sans celles de $\uparrow$ préserve le fait que les termes soient en $\uparrow$-forme normale. C'est l'objet de la conjecture ci-dessous.

Conjecture 18.2.4 (Préservation de la $\uparrow$-normalité)

> Le $\lambda_{wsn\to c}$-calcul typé, sans les règles $\uparrow$-app, $\uparrow$-sub et $\uparrow$-λ, appliqué à des termes en $\uparrow$-forme normale produit des termes eux-aussi en $\uparrow$-forme normale.

Preuve : Par cas. ■

Si cette conjecture est vraie, alors on peut éliminer les règles $\uparrow$-app, $\uparrow$-sub (pour peu qu'on remette la règle e_1) et $\uparrow$-λ du calcul, et considérer qu'on travaille sur des termes qui sont, à l'origine, en $\uparrow$-forme normale; cela nous donnerait un nouveau calcul qui serait sans doute très proche du λlxr-calcul [53].

La dernière conjecture permet de s'assurer qu'on n'a plus besoin de la règle n_1.

Conjecture 18.2.5 (Dispense de n_1)

> Dans le $\lambda_{wsn\to c}$-calcul typé, travaillant sur des $\uparrow$-formes normales, la règle n_1 ne peut jamais s'appliquer.

L'ensemble des conjectures de ce chapitre, ainsi que l'étude plus approfondie de cette dernière possibilité de calcul, constitue l'un des travaux futurs mentionnés dans la conclusion de cette thèse.

Cinquième partie

Du λ-calcul aux réseaux de preuve, un voyage en suivant l'explicite

L'élimination des coupures étant un mécanisme interne à la logique, il a toujours semblé intéressant d'établir des liens entre celui-ci et les notions de réduction que l'on peut avoir dans les calculs utilisés en sémantique, et principalement la β-réduction.

D'un côté, la logique linéaire affine la logique intuitionniste. Elle propose en plus une représentation des preuves sous forme de graphes et son processus d'élimination des coupures est alors un système de réécriture de graphe. De l'autre, les calculs avec opérateurs explicites permettent de décomposer la β-réduction en étapes plus atomiques de calcul.

L'étude des liens entre les calculs avec opérateurs explicites et l'élimination des coupures dans les réseaux de preuve, guide la définition de nouveaux calculs, comme λ_{wsn} et λlxr, qui décomposent encore plus la β-réduction. Pour arriver à ces nouveaux calculs, le processus d'explicitation a procédé en plusieurs étapes : d'abord les substitutions explicites, puis les affaiblissements explicites, et enfin la contraction explicite.

Dans cette courte partie, nous nous proposons d'étudier l'atomicité de ces calculs avec, comme référence, l'élimination des coupures dans les réseaux de preuve $\mathcal{R}_E$. Avant cela, nous allons regarder la définition d'un calcul qui effectue la démarche d'explicitation en sens inverse : nous partons des réseaux de preuve pour nous approcher des calculs avec opérateurs explicites. L'idée de ce calcul vient de la constatation que, dans la traduction de λ_{wsn} dans les réseaux, la substitution, ainsi que le membre droit d'une application, apparaissent dans une boîte. On s'est alors demandé s'il ne serait pas possible de trouver une syntaxe qui confonde ces deux objets en un seul.

Chapitre 19

Un calcul chimique pour les réseaux de preuve

Dans ce chapitre, nous nous intéressons à la définition d'un calcul directement issu des réseaux de preuve. Puisque l'élimination des coupures $\mathcal{R}_E$ est une réécriture de graphe, nous sommes obligés d'avoir un calcul qui correspond à cette notion de réécriture. Les définitions et propriétés présentées dans le chapitre 1 ne sont plus valables pour la plupart.

L'intérêt de ce calcul est de proposer une notation simple pour écrire les réseaux de preuve et leurs réductions. Cela permet, entre autres, de programmer une implantation simple de l'élimination des coupures, et même, comme nous le verrons ci-dessous, d'en tirer naturellement une implantation distribuée[1]. Nous donnons la définition du calcul, puis quelques propositions sous forme de conjectures.

19.1 Définition du λ_{PN}-calcul

Lorsqu'on doit découper un réseau de preuve pour en donner un notation linéaire, il faut trouver un moyen de conserver l'information que l'on va perdre, à savoir la connexion des différents éléments entre eux. Pour cela, nos termes seront décorés de formules de la logique linéaire qui permettront de repérer à quels autres termes ils sont connectés.

Les termes de λ_{PN} sont directement inspirés des réseaux de preuve :

- le W correspond à l'affaiblissement,
- le C à la contraction,
- le D à la déréliction, c'est-à-dire aux variables (voir la traduction de λ_{wsn}),
- le $\parr$ à l'abstraction fonctionnelle, et
- la boîte contient un réseau plus petit à l'intérieur.

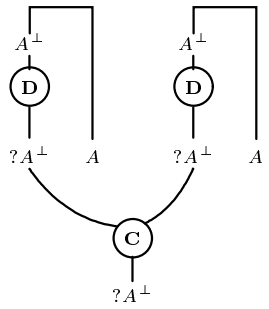
Les systèmes de réécriture de graphes sont peut-être moins intuitifs que les systèmes de réécriture de termes, car ils n'ont pas de structure pour écrire leurs objets. À la place d'un terme qui contient des redex, on y trouve un ensemble de termes qu'il faut parcourir pour trouver ceux qui peuvent se réduire. Dans la récriture de graphes, il y a deux notions de redex. Un redex peut être un terme isolé, auquel cas il suffit de le trouver pour le réduire. Un redex peut aussi mettre en jeu plusieurs termes, auquel cas il faut s'assurer que tous sont présents dans l'ensemble avant d'effectuer la réduction.

Souvent, on donne le principe d'une solution chimique comme intuition pour ces systèmes. En effet, de la même façon que les molécules présentes dans la solution peuvent interagir pour en produire de nouvelles, les termes d'un système de réécriture de graphes peuvent se combiner pour en produire de nouveaux. Dans notre cas, nous allons aussi faire un parallèle qui semble intéressant entre les termes du λ_{PN}-calcul et les processus évoluant dans un système d'exploitation.

[1] Il existe actuellement une implantation séquentielle de ce calcul, l'implantation distribuée étant en cours de développement.

Les termes

Prenons le réseau suivant :



Pour l'écrire sous forme de solution, on propose de noter D la combinaison du lien *Axiome* et du lien *dereliction* et C le lien *contraction*. Pour ne pas perdre l'information de connexion, on indique les formules qui sont entre les liens : les deux liens *dereliction* sont notés ${}_{?A^\perp}D_A$, et le lien *contraction* est noté ${}^{?A^\perp,?A^\perp}_{?A^\perp}C$, ce qui nous donne le terme-solution suivant :

$$ {}_{?A^\perp}D_A \mid {}_{?A^\perp}D_A \mid {}^{?A^\perp,?A^\perp}_{?A^\perp}C $$

On a noté la juxtaposition des liens avec une barre verticale. La présence des formules ne permet cependant pas de retrouver la connexion exacte, puisque la formule $?A^\perp$ revient deux fois. Pour lever ce genre d'ambiguïtés, on commence par numéroter les formules, ce qui nous donne :

$$ {}_{?A_1^\perp}D_{A_1} \mid {}_{?A_2^\perp}D_{A_2} \mid {}^{?A_1^\perp,?A_2^\perp}_{?A_3^\perp}C $$

Le tableau ci-dessous présente la grammaire des termes-solutions notés S et des termes-molécules notés t :

$$ t \quad ::= \quad {}_{?A_1^\perp}D_{A_2} \mid {}^{?A_1^\perp,?A_2^\perp}_{?A_3^\perp}C \mid {}^{?A_1^\perp}_{?A_2^\perp}L \mid L^{A_1}_{A_2} \mid {}_{?A^\perp}W \mid \invamp_{?A^\perp \invamp B} \mid \Gamma[S]_{\otimes !A} \mid \Gamma[S]_{!A} $$

$$ S \quad ::= \quad t \mid t|S $$

Les différents symboles de liens sont décorés de formules à différents endroits. En bas à gauche, on trouve les formules de l'environnement de typage ajoutées dans le séquent conclusion de la règle correspondante. En bas à droite, on trouve la formule (s'il y en a une) principale dans le séquent conclusion. En haut à gauche, on trouve les formules de l'environnement de typage du séquent hypothèse qui sont utilisées par la règle correspondante.

La boîte est symbolisée par une paire de crochets à gauche de laquelle on trouve, en bas, l'ensemble des formules de l'environnement de typage du séquent. En bas à droite, on y trouve deux types de formule : celle commençant par un !, auquel cas le terme-molécule correspond à une substitution, et celle commençant par un $\otimes$, auquel cas le terme-molécule correspond au membre droit d'une application. Les termes L servent à conserver la connexion des termes au fil des réductions. Ils peuvent être vus comme des opérateurs explicites d'α-conversion.

On peut ajouter une notion de type à nos termes-solutions, cela rend plus clair le sens des termes-molécules. Voici les règles de typage :

$$\frac{}{A_1 \vdash ({}_{?A_1^\perp} D_{A_2}) : A_2}\ Ax \qquad \frac{\Gamma \vdash (S) : A}{B, \Gamma \vdash ({}_{?B^\perp} W \mid S) : A}\ Weak$$

$$\frac{A_1, A_2, \Gamma \vdash (S) : A}{A_3, \Gamma \vdash ({}^{?A_1^\perp, ?A_2^\perp}_{?A_3^\perp} C \mid S) : A}\ Contr \qquad \frac{A, \Gamma \vdash (S) : B}{\Gamma \vdash (S \mid \invamp_{?A^\perp \otimes B}) : A \to B}\ Par$$

$$\frac{A_1, \Gamma \vdash (S) : A}{A_2, \Gamma \vdash ({}^{?A_1^\perp}_{?A_2^\perp} L \mid S) : A}\ Lg \qquad \frac{\Gamma \vdash (S) : A_1}{\Gamma \vdash (L^{A_1}_{A_2} \mid S) : A_2}\ Ld$$

$$\frac{\Gamma_1 \vdash (S_1) : A \to B \quad \Gamma_2 \vdash (S_2) : A}{\Gamma_1, \Gamma_2 \vdash (S_1 \mid {}_{\Gamma_2}[S_2]_{\otimes !A}) : B}\ Times$$

$$\frac{A, \Gamma_1 \vdash (S_1) : B \quad \Gamma_2 \vdash (S_2) : A}{\Gamma_1, \Gamma_2 \vdash (S_1 \mid {}_{\Gamma_2}[S_2]_{!A}) : B}\ Box$$

Les règles de réduction

Les règles de réduction reprennent exactement l'élimination des coupures de la logique linéaire enrichie $\mathcal{R}_E$. Les règles dont le nom comporte un L sont des règles supplémentaires, chargées d'éliminer les termes-molécules L de la solution. La figure 19.1 présente l'ensemble des règles de réduction ainsi qu'une règle d'équivalence.

On appelle l-calcul le calcul constitué des règles *lw*, *lc*, *lb*, *ld*, *llg*, *lld*, *bl*, *dl*.

L'observation des règles nous conduit à établir un parallèle entre ce calcul et un système de gestion distribuée des processus. Dans ce système, on a plusieurs processus qui s'exécutent sur plusieurs machines. Chaque boîte correspond à un processus s'exécutant sur une machine. Les termes-molécules à l'extérieur des boîtes sont les organisateurs des processus. Les processus corespondent entre eux à l'aide de signaux, et peuvent changer la localisation d'un processus. On considère que les formules correspondent à des canaux de communication. Voici un sens que l'on peut donner aux différents termes, ainsi que la façon dont ils s'utilisent :

- Les termes ${}^{?A_2^\perp}_{?A_1^\perp} L$ et $L^{A_2}_{A_1}$ sont des relais de communication. Ils indiquent que le canal de communication a changé.
- Le terme ${}_{?A^\perp} W$ est un signal de destruction (`kill`) envoyé au processus qui écoute sur le canal A. Ce signal est ensuite propagé vers tous les processus qui communiquaient avec celui-ci : c'est la règle *wb*.
- Le terme ${}^{?A_1^\perp, ?A_2^\perp}_{?A_3^\perp} C$ est un signal de duplication (`fork`) envoyé au processus sur le canal A_3, lui indiquant que l'un des deux devra communiquer sur A_1 et l'autre sur A_2. Ce signal est ensuite propagé vers tous les processus qui communiquaient avec celui-ci : c'est la règle *cb*.
- Le terme ${}_{?A_1^\perp} D_{A_2}$ est une primitive de recouvrement (`exec`). Elle récupère le processus qui s'exécutait ailleurs afin de poursuivre son exécution : c'est la règle *db*.
- L'annotation $\otimes$ au bord d'une boîte est une indication de masquage des signaux. Tant qu'elle est présente, les signaux W, C et D sont ignorés : ils ne peuvent pas atteindre le processus qui est à l'intérieur.
- Le terme $\invamp_{?A^\perp \otimes B}$ permet de supprimer le masque du processus sur A, permettant la délivrance des signaux : c'est la règle $\invamp - \otimes$.
- Les processus peuvent être déplacés, sans recouvrement, sur la machine d'un processus qui aura besoin de son résultat, c'est la règle *bb*.

Ce parallèle définit une notion de calcul distribué. Il serait intéressant de la comparer avec les calculs de processus déjà existants, comme le π-calcul ou le join-calcul (voir [62, 10, 36]).

19.2 Conjectures

Voici quelques conjectures. On donne des indications pour les prouver.

Conjecture 19.2.1

Le l-calcul termine.

Preuve : En prenant comme poids de la solution : le nombre de termes-molécules présents plus la somme des écarts entre la profondeur maximum et la profondeur de chaque molécule L ; ce poids décroit strictement à chaque application de règle du $l - calcul$ et est borné par zéro. ■

Conjecture 19.2.2

Le typage est préservé par les λ_{PN}-réductions.

Preuve : Par cas. ■

On peut traduire une solution en réseau de preuve par récurrence sur la dérivation de typage de celle-ci, en oubliant les indices des formules et les termes-molécules L. On obtient un réseau correct. Les règles wb, wc, $wout$, db, cb, bb, $\parr - \otimes$ ainsi que l'équivalence $c - i/o$ sont simulables en exactement une étape par $\mathcal{R}_E$. Comme le $l - calcul$ termine, on en déduit la conjecture suivante.

Conjecture 19.2.3

Le λ_{PN}-calcul termine.

Comme on a une traduction dans les deux sens, entre λ_{PN} et les réseaux de preuve, il devrait être possible d'obtenir d'autres résultats.

$$
\begin{array}{lrcl}
(wc) & {}_{?A^\perp}W \mid {}^{?A^\perp,?A_1^\perp}_{?A_2^\perp} C & \to & {}^{?A_1^\perp}_{?A_2^\perp} L \\
(wb) & {}_{?B,\ldots,?Z}[S]_{!A} \mid {}_{?A^\perp}W & \to & {}_{?B^\perp}W \mid \ldots \mid {}_{?Z^\perp}W \\
(cb) & {}_{?B,\ldots,?Z}[S]_{!A_3} \mid {}^{?A_1^\perp,?A_2^\perp}_{?A_3^\perp} C & \to & {}_{?B_1,\ldots,?Z_1}[S \mid L^{A_3}_{A_1}]_{!A_1} \mid {}_{?B_2,\ldots,?Z_2}[S \mid L^{A_3}_{A_2}]_{!A_2} \\
 & & & \mid {}^{?B_1^\perp,?B_2^\perp}_{?B^\perp} C \mid \ldots \mid {}^{?Z_1^\perp,?Z_2^\perp}_{?Z^\perp} C \\
(wout) & {}_{\Gamma\cup ?A}[{}_{?A^\perp}W \mid S]_\alpha & \to & {}_{?A^\perp}W \mid {}_\Gamma[S]_\alpha \\
(db) & {}_\Gamma[S]_{!A_1} \mid {}_{?A_1^\perp}D_{A_2} & \to & S \mid L^{A_1}_{A_2} \\
(bb) & {}_{\Gamma_1}[S_1]_{!A} \mid {}_{\Gamma_2\cup ?A}[S_2]_\alpha & \to & {}_{\Gamma_1\cup\Gamma_2}[{}_{\Gamma_1}[S_1]_{!A} \mid S_2]_\alpha \\
(\parr - \otimes) & \parr_{?A^\perp \parr B} \mid {}_\Gamma[S]_{\otimes !A} & \to & {}_\Gamma[S]_{!A} \\
(lw) & {}^{?A_2^\perp}_{?A_1^\perp} L \mid {}_{?A_2^\perp}W & \to & {}_{?A_1^\perp}W \\
(lc) & {}^{?A_2^\perp}_{?A_1^\perp} L \mid {}^{?A_3^\perp,?A_4^\perp}_{?A_2^\perp} C & \to & {}^{?A_3^\perp,?A_4^\perp}_{?A_1^\perp} C \\
(lb) & {}^{?A_2^\perp}_{?A_1^\perp} L \mid {}_{\Gamma\cup ?A_2}[S]_\alpha & \to & {}_{\Gamma\cup ?A_1}[{}^{?A_2^\perp}_{?A_1^\perp} L \mid S]_\alpha \\
(ld) & {}^{?A_2^\perp}_{?A_1^\perp} L \mid {}_{?A_2^\perp}D_A & \to & {}_{?A_1^\perp}D_A \\
(llg) & {}^{?A_2^\perp}_{?A_1^\perp} L \mid {}^{?A_3^\perp}_{?A_2^\perp} L & \to & {}^{?A_3^\perp}_{?A_1^\perp} L \\
(lld) & L^{A_2}_{A_1} \mid L^{A_3}_{A_2} & \to & L^{A_3}_{A_1} \\
(bl) & L^{A_2}_{A_1} \mid {}_\Gamma[S]_{!A_2} & \to & {}_\Gamma[L^{A_2}_{A_1} \mid S]_{!A_1} \\
(dl) & L^{A_2}_{A_1} \mid {}_{?A^\perp}D_{A_2} & \to & {}_{?A^\perp}D_{A_1} \\
(c - i/o) & {}_{\Gamma\cup ?A}[{}^{?A_1^\perp,?A_2^\perp}_{?A^\perp} C \mid S]_\alpha & \sim & {}^{?A_1^\perp,?A_2^\perp}_{?A^\perp} C \mid {}_{\Gamma\cup ?A_1\cup ?A_2}[S]_\alpha
\end{array}
$$

FIG. 19.1 – Règles de réduction du λ_{PN}-calcul

Chapitre 20

Un aller et retour

Dans ce chapitre, nous proposons un tableau comparatif de différents calculs connectés aux réseaux de preuve. Nous regardons essentiellement les règles de réduction, en prenant comme référence l'élimination des coupures $\mathcal{R}_E$. Les calculs comparés sont, dans l'ordre d'apparition dans le tableau : le λ-calcul et le λ_{ws}-calcul[1] (présentés dans le chapitre 2), le λ_{wsn}-calcul et le $\lambda_{wsn\to c}$-calcul (présenté dans la partie IV), le λlxr-calcul (présenté chapitre 2), le λ_{PN}-calcul (presenté dans le chapitre précédent), et enfin l'élimination des coupures $\mathcal{R}_E$ dans les réseaux de preuve (présentée dans le chapitre 3 et enrichie dans le chapitre 16). Le tableau est présenté figure 20.1.

Chaque colonne correspond à un calcul. On y trouve le nom des règles de réduction de celui-ci, regroupés par rapport aux règles de $\mathcal{R}_E$: pour chaque règle de $\mathcal{R}_E$ on indique quelles règles du calcul considéré y fait appel lors de sa simulation. Dans l'avant-dernière ligne, on recense toutes les équivalences de ces calculs, et, dans la dernière ligne, on donne les règles dont la simulation ne nécessite aucune étape de $\mathcal{R}_E$ car les réseaux sont équivalents.

Ce tableau nous permet de voir comment le fait de rendre explicites les mécanismes internes de la β-réduction permet de se rapprocher de $\mathcal{R}_E$. On peut comparer les calculs de la façon suivante.

Pour la partie haute du tableau :

- Plus il y a de cases dans une colonne, plus le calcul est fin, c'est-à-dire plus il faut utiliser de règles pour simuler la β-reduction.
- Moins il y a de règles par case, moins le calcul est redondant par rapport aux réseaux de preuve.

Pour la partie basse qui concerne les règles et équivalences supplémentaires :

- Il est plutôt intéressant de minimiser leur nombre puisqu'elles correspondent à des réseaux équivalents. Pour la même raison, des équivalences sont préférables aux règles, même si ce choix est discutable (voir la discussion section 13.2.3).
- Il faut tout de même en garder suffisament pour permettre à nos systèmes de réécriture de termes de se comporter comme le système de réécriture de graphes qu'est $\mathcal{R}_E$.

Les cases marquées d'une étoile dans la colonne du $\lambda_{wsn\to c}$-calcul signifient qu'on ne sait pas encore les remplir. En effet, comme nous l'avons dit dans le chapitre 18, ce calcul est en cours d'élaboration, et la simulation n'a pas encore été établie formellement. De plus, il se pourrait que l'ensemble des règles de remontée des affaiblissements explicites devienne inutiles si l'on définit finement les autres règles du calcul.

[1] On aurait aussi pu prendre l'ancienne version du λ_{wsn}-calcul, c'est-à-dire le λ_{ws}-calcul avec noms de [19].

λ	λ_{ws}	λ_{wsn}	$\lambda_{wsn\to c}$	λlxr	λ_{PN}	$\mathcal{R}_E$
β	n_1	n_1	n	Var	db	$d-b$
	$b_1,\ b_2$	b	b	B	$⅋-\otimes$	$Ax-cut$
						$⅋-\otimes$
	$\dfrac{a,\ c_2}{c_1}$	$\dfrac{a,\ c_2}{c_1}$	$\dfrac{a_2,\ c_2}{c_1}$	$Cont1$	cb	$c-b$
				$App2,\ Comp$	bb	$b-b$
	$n_2,\ n_3,\ e_1$	$n_2,\ e_1,\ c_3$	e_1	$Weak1$	wb	$w-b$
	$b_2,\ e_1,\ n_1$ $n_2,\ n_3$	$b,\ a,\ e_1,\ e_2,$ $n_1,\ n_2,\ c_1,\ c_2,$ $c_3,\ c_4$	$*$	$Merge$	wc	wc
		$b,\ a,\ c_1,\ c_2$	$*$	$WApp2$	$wout$	wb
		$f,\ e_2,\ d$ $\emptyset,\ c_4$	$*$	$A,\ C1_c,\ C2_c,$ $C_w,\ S$	$c-i/o$	$\sim$
	$f,\ e_2,\ d$		$*$	$Abs,\ App1,\ Weak2,$ $Cont2,\ WAbs,$ $WApp1,\ Cross,$ $CAbs,\ CApp1,$ $CApp2$	$lw,\ lc,\ lb,\ ld,$ $llg,\ lld,\ bl,\ dl$	$\sim$

FIG. 20.1 – Tableau comparatif des règles de réduction de différents calculs

Conclusion

Nous avons pu constater, à travers ces travaux, que la normalisation forte des calculs avec substitutions explicites n'est pas une propriété évidente à établir. Afin d'en faciliter la démonstration, nous avons formalisé une technique de preuve qui permet de déduire la normalisation forte à partir de la préservation de la normalisation forte (partie II). Nous nous sommes ensuite intéressé au $\overline{\lambda}\mu\tilde{\mu}$-calcul, un calcul symétrique non-déterministe issu de la logique classique étendant l'isomorphisme de Curry-Howard. Nous avons démontré sa normalisation forte ainsi que la normalisation forte d'une version avec substitutions explicites (partie III). Nous avons étudié une nouvelle version du λ_{ws}-calcul avec nom, le λ_{wsn}-calcul, qui apporte une grande souplesse dans les calculs avec substitutions explicites. En prouvant sa normalisation forte, nous avons constaté qu'il permet d'approfondir les liens entre les substitutions explicites et les réseaux de preuve de la logique linéaire (partie IV). Enfin, nous avons observé comment, en allant du λ-calcul jusqu'aux réseaux de preuve, l'introduction successive de constructions explicites dans les calculs permet, tout en donnant à ceux-ci des propriétés intéressantes, de tisser une trame presque continue entres les différentes notions de réduction (partie V).

* * *

Après avoir formalisé la technique "PSN implique SN", nous l'avons appliquée à quelques calculs avec substitutions explicites représentant les différents types de définitions les plus classiques. Nous avons remarqué que son application à des calculs nommés est en général facile et directe (sauf pour $\lambda\sigma_n$), tandis que dans le cas de calculs avec indices de de Bruijn, elle nécessite une simulation qui peut être difficile à établir. En particulier, nous n'avons pas réussi à l'établir pour le λ_{ws}-calcul, à cause de la présence des affaiblissements explicites (appelés aussi étiquettes). On peut donner comme directions de recherches futures :

- L'application au λ_{ws}-calcul, qui devrait permettre de mieux comprendre les mécanismes qui se cachent derrière la transformation de l'α-équivalence en indices de de Bruijn.
- L'application à d'autres calculs qui auraient PSN mais pour lesquels il n'y aurait pas encore de preuve de SN. Cela permettrait sans doute d'établir ces preuves relativement facilement.
- Enfin l'application de cette technique en utilisant des systèmes de typage plus évolués que les types simples, afin de voir quelles difficultés apparaissent et quels résultats on peut obtenir.

L'étude du $\overline{\lambda}\mu\tilde{\mu}$-calcul avec et sans substitutions explicites nous a permis de remarquer que la technique de preuve par réductibilité semble difficile à mettre en place dans le cas avec substitutions explicites. Alors que cette technique fonctionne pour le cas du calcul symétrique sans substitutions explicites, ainsi que dans le cas de calculs non-symétriques avec substitutions explicites, la tentative que nous avons faite pour le $\overline{\lambda}\mu\tilde{\mu}\mathbf{x}$-calcul s'est avérée un échec. Nous nous sommes tournés sur la technique de perpétuité pour prouver PSN, puis nous avons utilisé la technique PSN implique SN. Deux perspectives de recherches s'ouvrent après ce travail :

- Il serait très intéressant de trouver un moyen d'effectuer une preuve par réductibilité pour le $\overline{\lambda}\mu\tilde{\mu}\mathbf{x}$-calcul ou tout autre calcul symétrique (non-déterministe on non) avec substitutions explicites, comme par exemple le λ-calcul symétrique.
- On pourrait aussi observer ce que pourrait donner une version du $\overline{\lambda}\mu\tilde{\mu}$-calcul avec substitutions et affaiblissement explicites, comme λ_{wsn}.

La nouvelle définition du λ_{wsn}-calcul nous a conduit à plusieurs études. Comme calcul de référence, à la place du λ-calcul, nous avons formalisé le λ_{wn}-calcul, un calcul nommé avec affaiblissement explicite. Pour prouver la normalisation forte de λ_{wsn} par traduction dans les réseaux de preuve de la logique linéaire, nous avons dû enrichir son procédé d'élimination des coupures avec une nouvelle règle *wb* et prouver la terminaison de ce nouveau système. Enfin, la définition du λ_{wsn}-calcul que nous

avons choisie comporte un grand nombre d'équivalences, ce qui nous a permis de nous rapprocher encore plus des réseaux de preuve. Nous avons aussi proposé une ébauche de calcul, le $\lambda_{wsn\to c}$-calcul, qui devrait permettre de simplifier la simulation précédente, ainsi que de se rapprocher d'une forme explicite de contraction. Voici quelques pistes de recherche que nous avons déjà commencé d'explorer :

- Il reste plusieurs propriétés à prouver pour le λ_{wsn}-calcul. La première, presque terminée au moment de la rédaction de cette thèse, est la normalisation forte du calcul des substitutions. Comme dans λ_{ws}, cette propriété est relativement difficile à établir. La technique par simulation utilisée pour le λ_{ws}-calcul est longue et fastidieuse à mettre en place, ce qui nous a incité à essayer la technique, bien plus élégante, utilisée pour le λlxr-calcul. Il apparaît cependant que celle-ci profite des propriétés particulières que vérifient les termes de ce calcul et, de ce fait, ne serait pas utilisable pour λ_{wsn}.

  Après la normalisation forte du calcul des substitutions, on poura s'intéresser à la confluence sur les termes ouverts et à la préservation de la normalisation forte. Vu la souplesse du calcul, d'une part, et sa proximité avec le λ_{ws}-calcul et le λlxr-calcul, d'autre part, on ne doute pas du fait que ces propriétés puissent être aisément établies. Une bonne méthode pourrait consister à traduire le λ_{wsn}-calcul dans le λlxr-calcul.
- Concernant l'α-équivalence, il est prévu à court terme d'étudier le λ_{wn}-calcul et ses liens avec le λ-calcul, ainsi que la version du λ_{wsn}-calcul avec multi-ensembles qu'on a présentée rapidement. En effet, celle-ci semble proche de pouvoir se passer d'α-équivalence. Par ailleurs, puisque la traduction dans les réseaux de preuve fait perdre l'information sur les variables, un calcul sans α-conversion en serait encore plus proche.
- L'étude du $\lambda_{wsn\to c}$-calcul nous permettra de vérifier ce que nous avons avancé, à savoir que ce calcul est plus facilement simulé dans les réseaux de preuve. Il sera sans aucun doute fructueux de comparer ce système avec le λlxr-calcul, celui-ci intégrant la contraction directement dans sa syntaxe. Cette comparaison devrait permettre d'enrichir le $\lambda_{wsn\to c}$-calcul.

L'observation des différents calculs intermédiaires entre le λ-calcul et les réseaux de preuves nous a montré comment on pouvait rapprocher ces derniers en intégrant dans ces calculs de plus en plus d'opérateurs explicites. Nous avons présenté le λ_{PN}-calcul, un calcul chimique qui correspond presque exactement aux réseaux de preuves.

Cette dernière partie de la thèse a été rédigée dans le but de donner quelques intuitions sur les travaux touchant aux substitutions explicites en lien avec les réseaux de preuve ; notre observation gagnerait à devenir une étude plus formelle des ces calculs et des liens entre eux. Nous avons déjà mentionné, dans le paragraphe précédent, les connexions qu'il faudrait établir entre $\lambda_{wsn\to c}$ et λlxr, nous pouvons ajouter, comme directions de recherche (en partie déjà en cours) :

- L'étude plus approfondie du λ_{PN}-calcul et en particulier de ses liens avec les réseaux de preuve et avec le λlxr-calcul. Dans ce dernier cas, on peut imaginer des modification du λlxr-calcul afin de le rendre encore plus proche de $\mathcal{R}_E$.
- La comparaison (ou les liens) entre le λ_{PN}-calcul et les travaux réalisés dans le domaine de la géométrie de l'interaction et des réductions optimales.
- Les liens entre le λ_{PN}-calcul et les calculs concurrents comme le π-calcul.

Annexes

Annexe A

Calculs sur les ensembles

Voici quelques propriétés du calcul ensembliste. Chacune est associée à une lettre, ce qui permet d'y faire référence dans un calcul plus complexe. On pense que cela offre une plus grande lisibilité, et, de ce fait, qu'il est plus facile de vérifier que les calculs plus complexes sont corrects.

Propriété A.0.4

$(a.)\ (A \cup B) \setminus C = (A \setminus C) \cup (B \setminus C)$
$(b.)\ A \setminus (A \cup B) = \emptyset$
$(c.)\ A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$
$(d.)\ A \cap (A \setminus B) = A \setminus B$
$(e.)\ A \cap B = \emptyset \ \Rightarrow\ A \setminus (B \cup C) = A \setminus C$
$(f.)\ B \cap C = \emptyset \ \Rightarrow\ (A \setminus B) \cup C = (A \cup C) \setminus B$
$(g.)\ (A \setminus B) \cup B = A \cup B$
$(h.)\ B \subset A \ \Rightarrow\ (A \setminus B) \cup B = A$
$(i.)\ (A \cap B) \cup ((B \setminus A) \setminus C) = B \setminus (C \setminus A)$
$(j.)\ A \cap B = \emptyset \ \Rightarrow\ (A \cup B) \setminus B = A$
$(k.)\ (A \cap B) \cup (A \setminus B) = A$
$(l.)\ A \cap (B \setminus C) = (A \cap B) \setminus C$
$(m.)\ A \setminus (B \setminus (C \setminus A)) = A \setminus B$
$(n.)\ (A \setminus B) \setminus C = A \setminus (B \cup C)$
$(o.)\ (A \setminus (B \setminus C)) \setminus C = (A \setminus B) \setminus C$
$(p.)\ (A \setminus B) \setminus C = (A \setminus C) \setminus B$
$(q.)\ ((A \cap B) \setminus C) \cup (B \setminus (A \setminus C)) = B$
$(r.)\ A \cap (B \setminus (C \setminus A)) = A \cap B$
$(s.)\ A \cap C = \emptyset \ \Rightarrow\ A \setminus (B \setminus C) = A \setminus B$
$(t.)\ A \cap C = \emptyset \ \Rightarrow\ A \cap B = A \cap (B \cup C)$
$(u.)\ (A \cap B) \setminus C = (A \setminus C) \cap (B \setminus C)$
$(v.)\ B \cap C = \emptyset \ \Rightarrow\ (A \setminus B) \cap C = A \cap C$
$(w.)\ A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C)$
$(x.)\ (A \cap (B \setminus C)) \cup (B \cap C) = (A \cup C) \cap B$

Bibliographie

[1] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *Journal of Functional Programming*, 1991.

[2] M. Abadi, G. Gonthier, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Proceedings of POPL'92*, pages 15–26, 1992.

[3] S. Abramsky and R. Jagadeesan. New foundations for the geometry of interaction. In *Proceedings of LICS'92*, pages 211–222, 1992.

[4] P.B. Andrews. *An Introduction to Mathematical Logic and Type Theory : To Truth Through Proof.* Academic Press, 1986.

[5] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[6] F. Barbanera and S. Berardi. A symmetric lambda-calculus for classical program extraction. In *Proceedings of TACS'94*, number 789 in LNCS, pages 495–515. Springer-Verlag, 1994.

[7] H. P. Barendregt. *The Lambda Calculus : its Syntax and Semantics.* Number 103 in Studies in Logic and the Foundations of Mathematics. 1981.

[8] H. P. Barendregt. *Lambda Calculi with Types.* Handbook of Logic in Computer Science. Oxford University Press, 1992.

[9] 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*, 1996.

[10] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96 :217–248, 1992.

[11] R. Bloo. *Preservation of Termination for Explicit Substitution.* PhD thesis, Eindhoven University, 1997.

[12] R. Bloo and H. Geuvers. Explicit substitution : on the edge of strong normalisation. *Theoretical Computer Science*, 211 :375–395, 1999.

[13] E. Bonelli. *Substitutions explicites et réécriture de termes.* Thèse de doctorat, Université Paris XI Orsay, 2001.

[14] A. Church. A set of postulates for the foundation of logic (1). *Annals of Mathematics*, 33 :346–366, 1932.

[15] A. Church. A formulation of the simple theory of types. *The journal of Symbolic Logic*, 5 :56–68, 1940.

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

[17] R. Di Cosmo and S. Guerrini. Strong normalization of proof nets modulo structural congruences. In *Proceedings of RTA'99*, number 1631 in LNCS, pages 75–89. Springer-Verlag, 1999.

[18] R. Di Cosmo, D. Kesner, and E. Polonovski. Proof nets and explicit substitutions. In *Proceedings of FOSSACS'00*, number 1784 in LNCS. Springer-Verlag, 2000. Version longue et révisée dans [19].

[19] R. Di Cosmo, D. Kesner, and E. Polonovski. Proof nets and explicit substitutions. *Mathematical Structures in Computer Science*, 13(3) :409–450, 2003.

[20] P.-L. Curien. *Categorical combinators, sequential algorithms and functional programming.* Pitman, 1986.

[21] P.-L. Curien, T. Hardin, and J.-J. Levy. Confluence properties of weak and strong calculi of explicit substitutions. Technical Report 1617, INRIA Rocquencourt, 1992.

[22] P.-L. Curien, T. Hardin, and A. Ríos. Strong normalisation of substitutions. *MFCS*, LNCS 629 :209–218, 1992.

[23] P.-L. Curien and H. Herbelin. The duality of computation. In *Proceedings of ICFP'00*, pages 233–243. ACM Press, 2000.

[24] P.-L. Curien and A. Ríos. Un résultat de complétude pour les substitutions explicites. *Comptes rendus de l'académie des sciences de Paris*, t. 312, Série I :471–476, 1991.

[25] V. Danos. *La logique linéaire appliquée à l'étude de divers processus de normalisation (et principalement du λ-calcul).* Thèse de doctorat, Université Paris VII, 1990.

[26] R. David and B. Guillaume. The λ_l-calculus. In *Proceedings of WESTAPP*, 1999.

[27] R. David and B. Guillaume. A λ-calculus with explicit weakening and explicit substitutions. *Mathematical Structure in Computer Science*, 11(1), 2001.

[28] R. David and B. Guillaume. Strong normalisation of the typed λ_{ws}-calculus. In *Proceedings of CSL'03*, volume 2803 of *LNCS*. Springer, 2003.

[29] N.G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagationes Math.*, 5(35) :381–392, 1972.

[30] N.G. de Bruijn. A namefree lambda calculus with facilities for internal definition of expressions and segments. Technical Report 78-WSK-03, Department of Mathematics, Eindhoven University of Technology, 1978.

[31] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. *Handbook of Theoretical Computer Science*, B :234–309, 1990.

[32] D. Dougherty, S. Lengrand, and P. Lescanne. An improved system of intersection types for explicit substitutions. In *Proceedings of TCS'02*, pages 511–523, 2002.

[33] G. Dowek. *La part du calcul.* Mémoire d'habilitation, Université de Paris 7, 1999.

[34] G. Dowek. Théorie des types. Notes de cours du DEA Programmation, 2002.

[35] M.-C.F. Ferreira, D. Kesner, and L. Puel. λ-calculi with explicit substitutions preserving strong normalization. *Applicable Algebra in Engineering, Communication and Computing*, 1999.

[36] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join calculus. In *Proceedings of POPL'96*, 1996.

[37] G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens.* Nebert, Halle, 1879.

[38] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50(1) :1–101, 1987.

[39] J.-Y. Girard. Geometry of interaction I : interpretation of system F. *Logic colloquium 1988*, pages 221–260, 1989.

[40] J.-Y. Girard, Y. Lafont, and P. Taylor. *Proofs and types.* Cambridge University Press, 1989.

[41] B. Guillaume. *Un calcul de substitution avec étiquettes.* Thèse de doctorat, Université de Savoie, 1999.

[42] T. Hardin. Confluence results for the pure strong categorical combinatory logic CCL : lambda-calculi as subsystems of CCL. *Theoretical Computer Science*, 65 :291–342, 1989.

[43] T. Hardin and A. Laville. Proof of termination of the rewriting system subst on ccl. *Theoretical Computer Science*, 46 :305–312, 1986.

[44] D. Hendriks and V. van Oostrom. Adbmal. In *Proceedings of the 19th Conference on Automated Deduction (CADE 19)*, number 2741 in LNAI, pages 136–150, 2003.

[45] H. Herbelin. Explicit substitutions and reducibility. *Journal of Logic and Computation*, 11 :429–449, 2001.

[46] D. Hilbert and W. Ackermann. *Grundzüge der Theoretischen Logik*. Springer-Verlag, 1928.

[47] W.A. Howard. The formulæ-as-type notion of construction. In *To H.B. Curry : Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 579–490. Academic Press, 1969.

[48] G. Huet. Confluent reductions : Abstract properties and application to term rewriting systems. *Journal of the Association for Computing Machinery*, 27(4) :797–821, 1980.

[49] F. Kamareddine and A. Ríos. Extending a λ-calculus with explicit substitutions which preserves strong normalization into a confluent calculus on open terms. *Journal of Functionnal Programming*, 7(4), 1997.

[50] F. Kamareddine and A. Ríos. Bridging de Bruijn indices and variable names in explicit substitutions calculi. *Logic Journal of the Interest Group of Pure and Applied Logic*, 6(6) :843–874, 1998.

[51] F. Kamareddine and A. Ríos. Relating the $\lambda\sigma$- and λs-styles of explicit substitutions. *Journal of Logic and Computation*, 10(3) :349–380, 2000.

[52] B. Kernighan and D. Ritchie. *The C Programming Language*. Prentice-Hall, 2 edition, 1988.

[53] D. Kesner and S. Lengrand. Broadening the horizon of the explicit substitution paradigm via a logical model. Submitted.

[54] J.W. Klop. Term rewriting systems. *Handbook of Logic in Computer Science*, 2 :1–116, 1992.

[55] J.-L. Krivine. *Lambda-calcul, types et modèles*. Masson, 1990.

[56] J. Lamping. An algorithm for optimal lambda calculus reduction. In *Proceedings of POPL'90*, pages 16–30, 1990.

[57] O. Laurent. *Étude de la polarisation en logique*. Thèse de doctorat, Université Aix-Marseille II, 2002.

[58] X. Leroy and P. Weis. *Le langage Caml*. Dunod, 2nd edition, 1999.

[59] P. Lescanne. From lambda-sigma to lambda-upsilon : a journey through calculi of explicit substitutions. In *Proceedings of the 21st ACM Symposium on Principles of Programming Languages (POPL)*, pages 60–69, 1994.

[60] P. Martin-Löf. *Intuitionnistic type theory*. Bibliopolis, 1984.

[61] P.-A. Mellies. Typed λ-calculi with explicit substitutions may not terminate. In *Proceedings of TLCA'95*, number 902 in LNCS, pages 328–334. Springer, 1995.

[62] R. Milner. A calculus of communicating systems. In Springer, editor, *LNCS*, volume 92, 1980.

[63] M. Parigot. $\lambda\mu$-calculus : An algorithmic interpretation of classical natural deduction. In *Proceedings of LICS'93*, pages 39–46. Computer Society Press, 1993.

[64] E. Polonovski. Strong normalization of $\overline{\lambda}\mu\tilde{\mu}$-calculus with explicit substitutions. In *Proceedings of FOSSACS'04*, number 2987 in LNCS, pages 423–437. Springer-Verlag, 2004.

[65] E. Polonovski. PSN implies SN. In *Proceedings of HOR'04*, Technical report of the Computer Science Departement of RWTH Aachen, 2004. Accepté pour publication.

[66] F.P. Ramsey. The foundations of mathematics. In *Proceedings of the London Mathematical Society*, volume 25 of *2nd*, pages 338–384, 1926.

[67] A. Ríos. *Contribution à l'étude des λ-calculs avec substitutions explicites*. Thèse de doctorat, Université Paris VII, 1993.

[68] K.H. Rose. Explicit cyclic substitutions. In *Proceedings of CTRS*, number 656 in LNCS. Springer-Verlag, 1992.

[69] B. Russel and A.N. Whitehead. *Principia Mathematica*, volume I. Cambridge University Press, 1910-1912.

[70] J. van Heijenoort. *From Frege to Gödel, a source book in mathematical logic, 1879–1931.* Harvard University Press, 1967.

[71] N. Wirth. The design of a PASCAL compiler. *Softw., Pract. Exper.*, 1(4) :309–333, 1971.

[72] H. Zantema. Termination of term rewriting by semantic labelling. *Fundamenta Informaticae*, 24, 1995.

Index

Calculs

Symboles

A

C

S

T

V

Résumé :

Les substitutions explicites ont été introduites comme un raffinement du λ-calcul, celui-ci étant le formalisme utilisé pour étudier la sémantique des langages de programmation. L'objet de cette thèse est l'étude de leurs propriétés de normalisation forte et de préservation de la normalisation forte. Ce manuscrit rend compte de plusieurs travaux autour de ces propriétés de normalisation, regroupés en trois volets.

Le premier d'entre eux formalise une technique générale de preuve de normalisation forte utilisant la préservation de la normalisation forte. On applique cette technique à un spectre assez large de calculs avec substitutions explicites afin de mesurer les limites de son utilisation. Grâce à cette technique, on prouve un résultat nouveau : la normalisation forte du $\lambda\upsilon$-calcul simplement typé.

Le deuxième travail est l'étude de la normalisation d'un calcul symétrique non-déterministe issu de la logique classique formulée dans le calcul des séquents, auquel est ajouté des substitutions explicites. La conjonction des problèmes posés par les calculs symétriques et ceux posés par les substitutions explicites semble vouer à l'échec l'utilisation de preuves par réductibilité. On utilise alors la technique formalisée dans le premier travail, ce qui nous demande de prouver tout d'abord la préservation de la normalisation forte. À cette fin, on utilise un fragment de la théorie de la perpétuité dans les systèmes de réécriture.

La définition d'une nouvelle version du λ_{ws}-calcul avec nom, le λ_{wsn}-calcul, constitue le troisième volet de la thèse. Pour prouver sa normalisation forte par traduction et simulation dans les réseaux de preuve, on enrichit l'élimination des coupures de ceux-ci avec une nouvelle règle, ce qui nous oblige à prouver que cette nouvelle notion de réduction est fortement normalisante.

Mots-Clés : Lambda-calcul – Substitutions explicites – Réécriture – Calcul symétrique – Normalisation – Logique classique – Logique linéaire – Réseaux de preuve.

Title : Explicit substitutions, logic and normalization

Abstract :

Explicit substitutions have been introduced as a refinment of the λ-calculus – the usual formalism used to study the semantic of programming languages. In this thesis, we study their properties of strong normalization and preservation of strong normalization. Several works around those normalization properties are presented here in three parts.

The first one formalises a general proof technique of strong normalization using preservation of strong normalization. We apply this technique to several calculi with explicit substitutions to measure its usefulness. A benefit of this work is a new result : the strong normalization of simply typed $\lambda\upsilon$-calculus.

The second one is the study of the normalization of a symmetric non-deterministic calculus providing a term notation for classical sequent calculus, extended with explicit substitutions. Due to the conjunction of symmetry and explicit substitutions, usual strong normalization proofs by reducibility technique seems to fail. We then use the technique formalised above, which requires to establish the preservation of strong normalization of the calculus. For this purpose, we use a fragment of the theory of perpetuality in rewriting systems.

The definition and the study of a new version of the λ_{ws}-calculus with names, the λ_{wsn}-calculus, is the third part of the thesis. To prove its strong normalization by translation and simulation in linear logic proof nets, we enrich their cut elimination procedure with a new rule. It therefore constrains us to establish strong normalization of this new notion of reduction on proof nets.

Keywords : Lambda-calculus – Explicit Substitutions – Rewriting – Symmetric calculus – Normalization – Classical Logic – Linear Logic – Proof Nets.

Discipline : Informatique

Laboratoire : Preuves, Programmes et Systèmes
Université Paris 7 – Case 7014 – 2 Place Jussieu – 75251 PARIS Cedex 05