Stratégies pour la réduction forte

Antoine Lanco

▶ **To cite this version:**

Antoine Lanco. Stratégies pour la réduction forte. Informatique et langage [cs.CL]. Université Paris-Saclay, 2023. Français. NNT : 2023UPASG097 . tel-04512443

HAL Id: tel-04512443
https://theses.hal.science/tel-04512443

Submitted on 20 Mar 2024

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

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

université
PARIS-SACLAY

Stratégies pour la réduction forte
Strategies for strong reduction

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580, Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique
Graduate School : Informatique et sciences du numérique Référent : Faculté des sciences d'Orsay

Thèse préparée dans l'unité de recherche **Laboratoire Méthodes Formelles** (Université Paris-Saclay, CNRS, ENS Paris-Saclay),
sous la direction de **Guillaume Melquiond**, directeur de recherche,
et la co-encadrement de **Thibaut Balabonski**, maître de conférences

Thèse soutenue à Paris-Saclay, le 15 Décembre 2023, par

Antoine LANCO

Composition du jury

Membres du jury avec voix délibérative

Johanne COHEN Directrice de recherche, CNRS, Université Paris Saclay	Présidente
Antonio BUCCIARELLI Maître de conférences (HDR), Université Paris Cité	Rapporteur & Examinateur
Giulio MANZONETTO Professeur des universités, Université Paris Cité	Rapporteur & Examinateur
Delia KESNER Professeure des universités, Université Paris Cité	Examinatrice

THESE DE DOCTORAT

NNT : 2023UPASG097

ÉCOLE DOCTORALE

universite PARIS-SACLAY | Sciences et technologies de l'information et de la communication (STIC)

Titre : Stratégies pour la réduction forte

Mots clés : Lambda-calcul, Évaluation partielle, Normalisation

Résumé : La sémantique d'un langage de programmation, et d'un langage fonctionnel en particulier, laisse généralement une certaine liberté quant à l'ordre dans lequel sont effectuées les différentes opérations. Les différentes stratégies qui peuvent être adoptées, comme l'appel par valeur ou l'évaluation paresseuse, bénéficient déjà à la fois d'un large corpus théorique et de nombreuses implémentations efficaces. Ce corpus cependant est majoritairement tourné vers un objectif d'évaluation des programmes, c'est-à-dire de production d'une valeur. Le cadre associé est celui de l'évaluation faible, dans lequel aucune évaluation n'est effectuée à l'intérieur d'une fonction qui ne serait pas totalement appliquée. En effet, dans un langage fonctionnel la fermeture représentant une telle fonction est déjà en elle-même considérée comme une valeur.

Cependant plusieurs situations justifient d'aller audelà de ce cadre vers un objectif de normalisation, où la réduction est poussée jusqu'au bout, y compris à l'intérieur de ce qui est usuellement considéré comme une valeur. On parle alors de réduction forte. L'évaluation forte peut notamment intervenir comme un outil d'optimisation à la compilation, pour évaluer partiellement le corps d'une fonction en fonction de valeurs qui seraient déjà connues pour certains de ses arguments. Dans un autre contexte, cela concerne également l'implémentation des assistants à la preuve, avec par exemple dans Coq les différentes machines de conversion, en stratégie paresseuse, et de réduction, en appel par valeur.

Malgré l'importance des applications, le corpus théorique fondant l'extension des stratégies usuelles de l'évaluation faible au cadre de la réduction forte reste très en-deçà du corpus traditionnel de l'évaluation faible, tant du point de vue du volume que de la maturité. Il se concentre sur quelques extensions particulières ou sur des cadres intermédiaires comme l'évaluation ouverte. Ces travaux, pour l'essentiel très récents, n'apportent cependant pas encore de réponses satisfaisantes à tous les nouveaux comportements observés en réduction forte, par exemple relatifs à l'explosion de la taille de certains termes. Par conséquent, les implémentations déjà réalisées utilisent souvent des stratégies ad hoc pour limiter l'impact de ces problèmes.

Cette thèse définit un calcul en appel par nécessité avec réduction forte, c'est-à-dire des stratégies de normalisation qui étendent les stratégies usuelles d'évaluation en appel par nécessité. Le calcul utilise des substitutions explicites afin de supprimer certains effets d'explosion de la taille des termes. Par ailleurs, les stratégies détectent de manière précoce certains radicaux nécessaires, réduisant ainsi le nombre de calculs dupliqués.

De plus, notre calcul bénéficie de propriétés de correction (les résultats obtenus correspondent bien à des formes normales du lambda-calcul) et de complétude (pour tout terme normalisable, nos stratégies atteignent bien la forme normale). Nous avons formalisé ce calcul et prouvé sa correction avec l'assistant de preuve Abella.

Parmi les stratégies autorisées par notre calcul, nous en avons isolé une qui produit systématiquement les séquences de réduction les plus courtes, et défini et implémenté une machine abstraite qui réalise cette stratégie. L'implémentation a permis de réaliser un grand nombre de tests qui confirment les gains attendus par rapport aux stratégies d'évaluation usuelles.

Title : Strategies for strong reduction
Keywords : Lambda-calculs, Partial evaluation, Normalization

Abstract : Semantics of a programming language, especially functional ones, usually leaves underspecified the order in which the various operations are performed. Some usual strategies, such as call by value or lazy evaluation, already benefit from a large theoretical corpus as well as numerous efficient implementations. This corpus, however, mainly focuses on program evaluation, that is, producing a value. This is the framework of weak evaluation, in which no evaluation is performed inside a function that is not fully applied. Indeed, in a functional language, the closure that represents such a function is already considered as a value.

Yet, several situations necessitate to go further and to look at normalization, which means that reduction is performed as far as possible, including in objects that are already values. That is called strong reduction. Among other things, strong evaluation can be used as an optimization pass during compilation, in order to partly evaluate the body of a function depending on the already known values of some of its arguments. Strong evaluation is also used inside proof assistants, for example Coq's machines of conversion (lazy strategy) and of reduction (call by value).

Despite the importance of these applications, the theoretical corpus that extends the usual strategies from weak evaluation to strong reduction is way less comprehensive than the corpus for weak evaluation. It either focuses on some specific extensions or on some intermediate frameworks such as open evaluation. These works, very recent for the most part, do not yet bring any satisfying explanation to the way strong reduction behaves, for example the size explosion of some terms. As a consequence, the existing implementations often rely on some adhoc strategies to alleviate the impact of these issues.

This thesis defines a call-by-need calculus with strong reduction, meaning a normalization strategies that extend the usual call-by-need evaluation strategies. The computation uses explicit substitutions to eliminate certain term size explosion effects, Furthermore, the strategies offer early detection of certain necessary redexes, thereby reducing the number of duplicated computations.

Furthermore, our computation benefits from correctness properties (the obtained results correspond to normal forms of the lambda calculus), and completeness (for any normalizable term, our strategies reach the normal form). We have formalized our computation and prove its correctness with the Abella proof assistant.

In this space, we have isolated a strategy that consistently produces the shortest reduction sequences and defined and implemented an abstract machine that executes this strategy. The implementation allowed us to conduct numerous tests which confirm the expected gains over conventional évaluation strategies.

Table des matières

1 Introduction **11**

2 Lambda-calcul et appel par nécessité **15**
2.1 Lambda-calcul . . . 15
2.1.1 Définition . . . 15
2.1.2 Stratégie d'évaluation . . . 22
2.2 Normalisation . . . 25
2.2.1 Définition . . . 25
2.2.2 Stratégie de réduction normalisante . . . 28
2.3 Position nécessaire . . . 30
2.3.1 Notion de nécessité . . . 30
2.3.2 Variable gelée / Position isolée . . . 34
2.4 Appel par nécessité en lambda-calcul pur . . . 36
2.4.1 Position et contexte . . . 37
2.4.2 Stratégie d'évaluation . . . 38
2.4.3 Approximation des positions isolées et des variables gelées . . . 39
2.4.4 Contexte d'évaluation pour l'appel par nécessité . . . 41
2.4.5 Substitution restreinte et problème des règles de commutation . . . 44
2.5 Substitution explicite . . . 44
2.5.1 Définition . . . 45
2.5.2 Calcul avec substitution explicite . . . 48
2.6 Conclusion . . . 50

3 Calcul λ_{sn} : Appel par nécessité en réduction forte **53**
3.1 Calcul . . . 53
3.2 Correction . . . 57
3.2.1 Simulation . . . 58
3.2.2 Caractérisation des formes normales . . . 63
3.2.3 Dépliage des formes normales . . . 73
3.3 Complétude . . . 75
3.3.1 Le calcul hôte λ_c . . . 76
3.3.2 Complétude . . . 79
3.3.3 Nécessité . . . 83
3.4 Conclusion . . . 84

4 **Calcul amélioré $\lambda_{\mathsf{sn}+}$** **87**
4.1 Calcul . . . 87
4.2 Forme normale locale . . . 89
4.3 Propriétés . . . 92
4.3.1 Correction des formes normales locales . . . 92
4.3.2 Propriété du diamant . . . 98
4.3.3 Optimalité relative . . . 101
4.4 Conclusion . . . 101

5 **Formalisation** **105**
5.1 Présentation d'Abella . . . 105
5.1.1 Règle d'inférence . . . 106
5.1.2 Mécanisme de preuve de base . . . 107
5.1.3 Gestion du contexte . . . 109
5.1.4 Induction structurelle . . . 110
5.1.5 Relations et fonctions . . . 112
5.1.6 Induction sur les dérivations . . . 113
5.2 Formalisation des calculs . . . 115
5.2.1 Formalisation du lambda-calcul pur . . . 115
5.2.2 Formalisation du calcul amélioré $\lambda_{\mathsf{sn}+}$ . . . 116
5.2.3 Fichier sig et mod . . . 120
5.3 Preuve de correction . . . 124
5.3.1 Correspondance des formes normales . . . 124
5.3.2 Simulation . . . 124
5.3.3 Adéquation de la caractérisation des formes normales locales . . . 125
5.3.4 Adéquation de la caractérisations des formes normales . . . 127
5.4 Conclusion . . . 127

6 **Machine abstraite** **129**
6.1 Configuration de la machine . . . 129
6.2 Règles de réductions . . . 130
6.3 Détails d'implémentation . . . 135
6.4 Conclusion . . . 139

7 **Conclusion** **143**
7.1 Contribution . . . 143
7.2 Pour la suite . . . 144
7.3 À étudier . . . 144

Table des figures

2.1 Variables libres d'un terme . . . 17
2.2 Règles de substitution . . . 17
2.3 Définition de la bêta-réduction . . . 18
2.4 Illustration de l'alpha-conversion d'un terme . . . 18
2.5 Graphique complet de réduction du terme $((\lambda x.\lambda y.xx)\ ((\lambda z.z)\ a))\ \Omega$ . . . 20
2.6 Combinateur de point fixe de Turing . . . 21
2.7 Confluence . . . 21
2.8 Graphique de réduction du terme $(\lambda x.xx)\ (\lambda z.(\lambda w.w)\ z)$ . . . 23
2.9 Graphique complet de réduction du terme $(\lambda x.xx)\ ((\lambda z.z)\ a)$ . . . 23
2.10 Graphique complet de réduction du terme $(\lambda x.y)\ ((\lambda y.y)z)$ . . . 23
2.11 Graphique complet de réduction du terme $(\lambda x.y)\ \Omega$ . . . 24
2.12 Stratégie d'appel par nécessité appliquée au terme $(\lambda x.xx)\ ((\lambda z.z)\ a)$ . . . 25
2.13 Caractérisation des formes normales du lambda-calcul . . . 26
2.14 Graphique complet de réduction du terme $(\lambda x.\lambda y.xy)\ ((\lambda a.a)z)\ ((\lambda b.b)z)$ . . . 27
2.15 Graphique complet de réduction du terme $(\lambda x.\lambda y.xw)\ ((\lambda a.a)z)\ \Omega$ . . . 28
2.16 Tableau des stratégies pour la forme normale du terme $(\lambda x.\lambda z.z(xx))\ (\lambda y.I\ y)$ . . . 29
2.17 Étapes de réduction du terme $(\lambda x.\lambda y.x)\ (I\ z)\ (I\ z)$ avec la fonction identité représentée par I. . . . 30
2.18 Réduction du terme $(\lambda x.x)\ t$ . . . 31
2.19 Identification des positions nécessaires dans le terme $(\lambda x.\lambda y.x)\ (I\ a)\ (I\ b)\ (I\ c)$ . . . 32
2.20 Étapes de réduction du terme $(\lambda x.\lambda y.x)\ (I\ a)\ (I\ b)\ (I\ c)$ . . . 32
2.21 Étapes de réduction du terme $(\lambda x.\lambda y.x)\ g\ (I\ b)\ h$ . . . 32
2.22 Graphique de réduction du terme $(\lambda x.\lambda y.x)\ a\ (I\ b)\ c$. . . . 33
2.23 Réduction du terme $(\lambda x.xx)\ ((\lambda x.x)\ b)$ avec la stratégie leftmost-outermost qui duplique du calcul . 33
2.24 Réduction du terme $\lambda x.x\ (I\ a)$ avec en bleu les variables gelées et en rouge la réduction du terme $I\ a$. 35
2.25 Graphique de réduction du terme $\lambda x.x\ (I\ a)$ . . . 35
2.26 Réduction de la structure $\lambda x.x\ (I\ a)\ (I\ b)$ . . . 36
2.27 Structures avec φ l'ensemble des variables gelées . . . 36
2.28 Graphique de réduction du terme $\lambda y.(\lambda x.x)\ (y\ (Ia)\ b)$ . . . 37
2.29 Ensemble $\mathcal{E}$ des contextes d'évaluation pour la stratégie d'évaluation en appel par nom . . . 38
2.30 Valeur ouverte et bêta-réduction pour la stratégie d'évaluation en appel par valeur . . . 39
2.31 Ensemble $\mathcal{E}$ des contextes d'évaluation pour la stratégie d'évaluation en appel par valeur . . . 39
2.32 Définition des modes qui approximent les positions isolées . . . 40
2.33 Exemple de calcul du mode du contexte $\lambda y.(y\ \lambda x.\bullet)$ . . . 40
2.34 Exemple de calcul du mode du contexte $(\lambda x.y)\ \bullet$ . . . 41
2.35 Exemple de calcul du mode du contexte $(\lambda x.x)\ (\lambda y.y\ \bullet)$ . . . 41

2.36 Évolution de la structure xab dans le terme $(\lambda y.y)(\lambda x.xab)$ . . . 42
2.37 Contexte d'évaluation paramétrique . . . 43
2.38 Vérification du fait que le contexte $\lambda x. \bullet\ (I\ y))$ est bien un contexte d'évaluation . . . 43
2.39 Règles de réduction pour le lambda-calcul avec substitution explicite . . . 45
2.40 Variables libres des termes avec substitution explicite . . . 45
2.41 Règles de substitution des termes avec substitution explicite . . . 46
2.42 Application de la règle dB sur le terme $(\lambda x.\lambda y.y\ x)\ a\ (\lambda z.z)$ . . . 46
2.43 Application des règles lsv et dB sur le terme $(y\ x)[y \setminus \lambda z.z][x \setminus a]$ . . . 47
2.44 Application de la règle lsv sur le terme $(x\ x)[x \setminus (\lambda a.a\ y)[y \setminus \lambda z.z]]$ . . . 47
2.45 Application de la règle gc sur le terme $z[z \setminus x][y \setminus \lambda z.z][x \setminus a]$ . . . 48
2.46 Fonction de dépliage . . . 48
2.47 Exemple de dépliage qui fait grossir exponentiellement le terme . . . 49
2.48 Définition des structures avec les substitutions explicites . . . 49
2.49 Contexte d'évaluation paramétrique . . . 50

3.1 Réduction définie inductivement pour la stratégie d'appel par nom . . . 54
3.2 Réduction définie inductivement pour la stratégie d'appel par valeur . . . 54
3.3 Première partie des règles de réduction du calcul λ_{sn} . . . 55
3.4 Deuxième partie des règles de réduction de notre calcul λ_{sn} . . . 55
3.5 Règles de réduction auxiliaire pour λ_{sn} . . . 56
3.6 Légende pour les preuves graphiques . . . 58
3.7 Graphique pour la preuve du cas DB-σ pour la simulation . . . 60
3.8 Graphique pour la preuve du cas LSV-BASE pour la simulation avec v une valeur . . . 61
3.9 Graphique pour la preuve du cas LSV-σ et LSV-σ-φ pour la simulation . . . 62
3.10 Caractérisation des formes normales avec les substitutions explicites . . . 63
3.11 Variable vivante . . . 65
3.12 Problème de confluence pour le terme $x[x \setminus \lambda y.Iz]$ . . . 65
3.13 Réponse . . . 65
3.14 Règles de typage pour λ_{c} . . . 76
3.15 Typage de $\lambda xy.x\ (x\ x)$ . . . 77
3.16 Typage de $(\lambda xy.x(xx))\ (\lambda z.z)\ \Omega$ . . . 77
3.17 Définition des occurrences positives et négatives des types . . . 78
3.18 Système annoté de types avec intersection non-idempotent pour λ_{sn} . . . 80
3.19 Exemple de non-confluence . . . 85

4.1 Nouvelle règle LSV-BASE pour $\lambda_{\mathsf{sn}+}$ . . . 89
4.2 Forme normale locale $\top$ . . . 90
4.3 Forme normale locale $\bot$ . . . 90
4.4 Forme normale locale . . . 91

5.1 Fichier de signature . . . 121
5.2 Fichier de spécification partie 1 . . . 122
5.3 Fichier de spécification partie 2 . . . 123

6.1 Réduction de $(\lambda x.x)(\lambda y.y\ t)$ . . . 134
6.2 Réduction de y qui est associée à $x\ t_1$ avec une pile $t_2\ t_3$ . . . 135

6.3 Sémantique à grand pas pour la machine abstraite 140
6.4 Caractérisation des structures pour la machine abstraite 140
6.5 Squelette de la fonction d'évaluation 141
6.6 Fonctionnement du suffixe et du lifting sur l'exemple $\lambda y.((\lambda z.\lambda w.z)\ (\lambda x.x))(\lambda k.y)$ 142

Remerciments

Dans un premier temps, je tiens à remercier les membres de mon jury, Antonio Bucciarelli et Giulio Manzonetto, pour leurs commentaires très pertinents qui m'ont permis d'améliorer mon manuscrit, Johanne Cohen pour avoir accepté de présider mon jury, et Delia Kesner d'avoir accepté de faire partie du jury.

Je tiens également à remercier tout particulièrement mes deux encadrants de thèse, Guillaume Melquiond et Thibaut Balabonski, sans qui tout cela n'aurait jamais été possible. Un très grand merci à vous pour votre patience et votre pédagogie. Merci d'avoir pris autant de temps pour m'apprendre autant de choses. Merci à toi, Thibaut, de m'avoir donné le goût pour les langages formels. Merci à toi, Guillaume, de m'avoir permis d'être plus rigoureux dans mon travail.

Je remercie aussi tous mes collègues de l'ancienne équipe VALS qui ont fait de mon lieu de travail un endroit chaleureux dans lequel j'aimais être. Grâce à toutes les discussions intéressantes ou drôles que j'ai pu avoir et grâce aux bons moments que j'ai pu passer. Je tiens à remercier particulièrement Sylvain, avec qui j'ai eu beaucoup de discussions sur de nombreux sujets qui m'ont beaucoup appris et qui m'ont bien fait rire.

Pour finir, je remercie mes amies Alexandrina et Rebecca avec qui j'ai passé des super moments. Avec qui j'ai fait des missions Costco pour transformer le bureau en magasin. Qui m'ont soutenu pour venir à bout de cette épreuve qu'est la thèse.

Pour conclure, merci à toutes ces personnes qui m'ont permis de venir au bout de ma thèse et qui m'ont tant appris.

Chapitre 1

Introduction

La sémantique d'un langage de programmation, et d'un langage fonctionnel en particulier, laisse généralement une certaine liberté quant à l'ordre dans lequel sont effectuées les différentes opérations. Les différentes stratégies qui peuvent être adoptées, comme l'appel par valeur ou l'évaluation paresseuse, bénéficient déjà à la fois d'un large corpus théorique et de nombreuses implémentations efficaces. Ce corpus cependant est majoritairement tourné vers un objectif d'évaluation des programmes, c'est-à-dire de production d'une valeur. Le cadre associé est celui de l'évaluation faible, dans lequel aucune évaluation n'est effectuée à l'intérieur d'une fonction qui ne serait pas totalement appliquée. En effet, dans un langage fonctionnel la fermeture représentant une telle fonction est déjà en elle-même considérée comme une valeur.

Cependant plusieurs situations justifient d'aller au-delà de ce cadre vers un objectif de normalisation, où la réduction est poussée jusqu'au bout, y compris à l'intérieur de ce qui est usuellement considéré comme une valeur. On parle alors de réduction forte. L'évaluation forte peut notamment intervenir comme un outil d'optimisation à la compilation, pour évaluer partiellement le corps d'une fonction en fonction de valeurs qui seraient déjà connues pour certains de ses arguments. Dans un autre contexte, cela concerne également l'implémentation des assistants à la preuve, avec par exemple dans Coq les différentes machines de conversion, en stratégie paresseuse, et de réduction, en appel par valeur.

Malgré l'importance des applications, le corpus théorique fondant l'extension des stratégies usuelles de l'évaluation faible au cadre de la réduction forte reste très en-deçà du corpus traditionnel de l'évaluation faible, tant du point de vue du volume que de la maturité. Il se concentre sur quelques extensions particulières ou sur des cadres intermédiaires comme l'évaluation ouverte. Ces travaux, pour l'essentiel très récents, n'apportent cependant pas encore de réponses satisfaisantes à tous les nouveaux comportements observés en réduction forte, par exemple relatifs à l'explosion de la taille de certains termes. Par conséquent, les implémentations déjà réalisées utilisent souvent des stratégies ad hoc pour limiter l'impact de ces problèmes.

Cette thèse définit un calcul en appel par nécessité avec réduction forte, c'est-à-dire des stratégies de normalisation qui étendent les stratégies usuelles d'évaluation en appel par nécessité. Le calcul utilise des substitutions explicites afin de supprimer certains effets d'explosion de la taille des termes. Par ailleurs, les stratégies détectent

de manière précoce certains radicaux nécessaires, réduisant ainsi le nombre de calculs dupliqués.

Dans le chapitre 2, nous allons présenter le lambda-calcul en détaillant sa syntaxe, ses réductions et les différentes définitions qui lui sont propres. Une fois ce cadre rappelé, nous allons parler des stratégies d'évaluation [29], c'est-à-dire de l'ordre dans lequel les bêta-réductions sont faites jusqu'à l'obtention d'une valeur. Les stratégies d'évaluation les plus célèbres sont call-by-value, qui évalue immédiatement les arguments d'une fonction avant de résoudre l'appel de la fonction, call-by-name, où les arguments d'une fonction sont évalués lorsqu'ils sont nécessaires, et call-by-need [31, 5], qui étend call-by-name avec un mécanisme de mémoïsation ou de partage pour se souvenir de la valeur d'un argument qui a déjà été évalué. Le lambda-calcul est considéré comme le modèle standard dans les langages de programmation fonctionnels, une fois équipé d'une stratégie d'évaluation. Ensuite, nous allons aborder les substitutions et la manière de gérer les variables dans les lambda-termes. Après avoir parlé des stratégies d'évaluation, nous allons nous intéresser aux stratégies de réduction normalisante, c'est-à-dire aux stratégies qui ne s'arrêtent pas lorsqu'elles ont atteint à une valeur, mais qui réduisent aussi le terme sous les abstractions.

Mais la réduction sous les abstractions pose toute une série de problèmes, notamment celui de réduire des termes qui ne sont pas utiles pour obtenir la forme normale. C'est pourquoi nous allons étudier comment identifier les termes nécessaires. Mais comme c'est un problème indécidable, nous allons définir une approximation des termes nécessaires grâce à un système d'annotation des variables que nous appelons «variable gelée». Avec ce système d'annotation il y a deux types de position dans un terme : les positions nécessaires et celles pour lesquelles nous n'avons pas encore suffisamment d'informations pour nous prononcer. Nous allons utiliser cette approximation des positions nécessaires, un système de contexte pour parler des positions et une stratégie d'appel par nécessité pour définir notre premier calcul pour le réduction normalisante.

Cependant, l'accumulation d'outils techniques va rendre la syntaxe trop compliquée à lire et va même introduire des blocages liés à la commutation des termes. Comme nous avons l'objectif de prouver notre calcul, nous allons utiliser une solution, qui s'adaptera bien à notre problème : le lambda-calcul avec substitution explicite.

Dans le chapitre 3, nous allons donc redéfinir notre calcul avec ce nouveau formalisme, ce qui va le rendre beaucoup plus lisible et nous permettre de ne plus utiliser les contextes, ce qui est un atout majeur pour les preuves à venir. Une fois le calcul λ_{sn} défini nous allons en prouver la correction par rapport au lambda-calcul traditionnel. Autrement dit, si on prend un terme du lambda-calcul et qu'on le réduit avec notre calcul et avec le lambda-calcul, les deux résultats doivent correspondre.

La preuve de correction est découpée en plusieurs étapes. Tout d'abord, nous allons établir le lemme de simulation qui met en relation les réductions de notre calcul et celles du lambda-calcul. Ensuite, nous allons caractériser les formes normales de notre calcul, afin de prouver que nos formes normales correspondent bien aux formes normales du lambda-calcul.

Cependant la condition pour substituer les termes est trop permissive. Elle permet de dupliquer des calculs et fait perdre la confluence en raison des radicaux qui restent

dans les substitutions explicites qui ne sont plus accessibles.

Dans le chapitre 4, nous allons définir un nouveau calcul $\lambda_{\mathsf{sn}+}$ qui utilise une autre condition pour substituer les termes dans les substitutions explicites, ce que nous allons appeler la forme normale locale. Elle caractérise parfaitement les termes qui ne peuvent plus être réduits par notre calcul $\lambda_{\mathsf{sn}+}$ dans les substitutions explicites.

Grâce à ces modifications du calcul et à cette nouvelle restriction, nous allons pouvoir prouver la confluence et même la propriété du diamant. On obtient également un résultat de minimalité, ce qui signifie que, quel que soit le choix de réduction effectué par le calcul, toutes les séquences de réduction ont la même longueur.

Dans le chapitre 5, nous allons formaliser le calcul $\lambda_{\mathsf{sn}+}$ et les preuves. Pour ce faire, nous utilisons l'assistant de preuve Abella [6]. Avec cet assistant nous allons formaliser la preuve de correction par rapport au lambda-calcul.

Dans le chapitre 6, nous allons implémenter une machine abstraite pour pouvoir tester notre calcul $\lambda_{\mathsf{sn}+}$. Nous allons également implémenter des machines abstraites pour des stratégies plus classiques telles que l'appel par nécessité faible itéré afin de pouvoir comparer notre machine. Grâce à ces tests, nous allons pouvoir constater dans quel cas notre machine est plus efficace et de combien.

La première machine abstraite connue pour trouver la forme normale d'un terme est due à Crégut [18] et implémente la réduction en « ordre normal ». Récemment, plusieurs travaux ont transposé les stratégies d'évaluation connues vers des stratégies de réduction forte ou des machines abstraites : appel par valeur [21, 11, 3], appel par nom [1] et appel par nécessité [9, 12]. Des extensions fortes de l'appel par nom ou de l'appel par nécessité peuvent également être trouvées dans les moteurs internes de certains assistants de preuves, notamment Coq, sans avoir toujours fait l'objet de publications. Les stratégies et la machine présentées dans cette thèse anticipent certaines réductions d'une manière que ne permet aucune de ces machines précédentes, de sorte à réduire le nombre total d'étapes de réduction nécessaire à l'obtention d'une forme normale.

Chapitre 2

Lambda-calcul et appel par nécessité

Ce chapitre présente le lambda-calcul dans sa forme traditionnelle et dans une forme étendue. Il introduit également des notions telles que les formes normales et les positions nécessaires.

La section 2.1 présente le lambda-calcul, composé de lambda-termes définis syntaxiquement et de règles de bêta-réduction qui mettent en relation deux lambda-termes. Elle explique le fonctionnement de la règle de bêta-réduction et ses difficultés, ainsi que certaines stratégies permettant de réduire les lambda-termes.

Dans la section 2.2, nous nous intéressons au fait qu'un terme ait ou non une séquence de réduction qui le mène à une forme irréductible, appelée forme normale. Nous étudions également la capacité des stratégies à mener à la forme normale d'un terme.

Dans la section 2.3, nous cherchons à augmenter nos chances d'obtenir la forme normale des termes qui en ont, en identifiant les réductions qui ne peuvent pas être évitées, que nous appelons nécessaires.

Dans la section 2.4, nous introduisons une stratégie qui utilise une notion de position nécessaire dans le lambda-calcul pur. Cependant, nous rencontrons des problèmes de blocage liés à des radicaux dont la réduction est mise en attente par la stratégie.

Afin de résoudre ces problèmes, nous utilisons une extension du lambda-calcul avec des substitutions explicites dans la section 2.5.

2.1 Lambda-calcul

La section 2.1.1 présente les différentes définitions qui sont liées au lambda-calcul, telles que les lambda-termes définis syntaxiquement, la réduction appelée bêta-réduction et quelques propriétés classiques. Le lambda-calcul est accompagné de stratégies de réduction qui donnent des règles pour déterminer l'ordre dans lequel les termes doivent être réduits. Celles-ci sont présentées dans la section 2.1.2.

2.1.1 Définition

Définition 1 (Lambda-calcul)**.** *Le lambda-calcul est un modèle dont les termes sont générés par trois constructions : les variables, les applications et les abstractions. Le lambda-calcul vient avec une notion de réduction qui décrit l'effet de l'application d'une fonction.*

Terme Les lambda-termes peuvent être de trois formes : une variable est représentée par un lettre (x), une application d'un lambda-terme t à un lambda-terme u est représentée par ces deux lambda-termes l'un à côté de l'autre ($t\ u$) et une fonction, aussi appelé lambda-abstraction, est représentée par la lettre grecque λ suivie d'une variable et d'un terme ($\lambda x.t$). Dans une lambda-abstraction $\lambda x.t$ la variable x représente le paramètre de la fonction et le terme t le corps de la fonction.

Définition 2 (Valeur)**.** *Nous définissons comme valeurs les lambda-termes de la forme $\lambda x.t$. On note Val l'ensemble des valeurs.*

La fonction identité peut être représentée par le terme $\lambda x.x$ c'est-à-dire une fonction avec un paramètre x dont le résultat est x lui même. L'application de la fonction identité à l'argument y est représentée par le lambda-terme $(\lambda x.x)\ y$. Les lambda-abstractions n'ont qu'un argument. Pour représenter une fonction qui a deux arguments, par exemple, il faut une lambda-abstraction qui a comme corps une autre lambda-abstraction. Par exemple, une fonction qui prend x et y en arguments et qui applique x à y est représentée par $\lambda x.\lambda y.xy$.

La lambda-abstraction introduit un nouveau nom de variable qui n'existe que dans les limites de cette lambda-abstraction. Chaque occurrence de variable dans un terme peut ou non être liée à une lambda-abstraction.

Définition 3 (Variable libre)**.** *Une variable qui n'est pas liée à une lambda-abstraction est une variable libre. On note* $\mathrm{FV}(t)$ *l'ensemble des variables libres dans le terme t, caractérisé par les équations données dans la figure 2.1.*

Un terme qui ne possède pas de variable libre est un terme clos.

Dans le terme $\lambda x.\lambda y.xz$, la variable z est libre et la variable x est liée à la lambda-abstraction λx. Dans le terme $(\lambda x.x)\ x$, la première occurrence de x est liée à la lambda-abstraction λx, alors que la deuxième est libre. Lorsqu'une variable est sous deux lambda-abstractions qui ont le même nom de variable, par convention on considère que la variable est liée au lieur le plus proche. Par exemple, dans le lambda-terme $\lambda x.\lambda x.x$, la variable x est liée à la lambda-abstraction qui se trouve directement au-dessus.

Dans les termes, on distingue plusieurs types de positions. Par exemple, les deux termes qui composent une application sont dits des sous-termes.

Définition 4 (Top-level)**.** *Quand un terme n'est pas un sous-terme c'est un terme en position racine, et on dit qu'il est en position* top-level*, car c'est le terme le plus à l'extérieur (le plus en haut) du terme.*

$$\begin{array}{rcl}\mathrm{FV}(x) & = & \{x\} \\ \mathrm{FV}(\lambda x.t) & = & \mathrm{FV}(t) \setminus x \\ \mathrm{FV}(t\ u) & = & \mathrm{FV}(t) \cup \mathrm{FV}(u)\end{array}$$

FIGURE 2.1 – Variables libres d'un terme

$$\begin{array}{rrcl} Var & z\{x \setminus t\} & = & z \text{ si } x \neq z \\ Subst & x\{x \setminus t\} & = & t \\ App & (t\ u)\{x \setminus w\} & = & (t\{x \setminus w\})\ (u\{x \setminus w\}) \\ Abs & (\lambda x.t)\{y \setminus u\} & = & \lambda x.(t\{y \setminus u\}) \text{ si } x \notin \mathrm{FV}(u) \text{ et } x \neq y \end{array}$$

FIGURE 2.2 – Règles de substitution

Définition 5 (Substitution)**.** *La substitution est le mécanisme de base de la bêta-réduction. Elle permet de remplacer, dans un terme, une variable par un terme. La substitution dans un terme t d'une variable x par un terme u est notée $t\{x \setminus u\}$. La figure 2.2 définit les règles de la substitution.*

On note que la quatrième règle est soumise à une condition, ce qui fait que la définition n'est pas complète. Nous reviendrons dessus bientôt.

Réduction Le lambda-calcul est composé de lambda-termes, mais aussi d'une réduction, la bêta-réduction, qui permet d'appliquer les lambda-abstractions à leurs arguments. Cette réduction est représentée par une flèche indicée par un bêta $\rightarrow_\beta$. L'application d'une abstraction à un argument est appelé un radical.

Définition 6 (Bêta-réduction)**.** *La bêta-réduction remplace toutes les occurrences de la variable de la lambda-abstraction par son argument dans le corps de la lambda-abstraction, à l'aide de l'opération de substitution (Fig. 2.3).*

De plus, on note que cette bêta-réduction est applicable à n'importe quel radical du terme.

Par exemple, dans la réduction $(\lambda x.xx)\ y \rightarrow_\beta yy$, toutes les occurrences de la variable x de l'abstraction sont remplacées par l'argument y, donc dans xx les x sont remplacés par des y.

On utilise la notation $\rightarrow_\beta^*$ pour parler de la fermeture réflexive transitive de la bêta-réduction.

Définition 7 (Capture)**.** *Si lors d'une substitution, une variable se retrouve liée à une abstraction à laquelle elle n'était pas liée à la base, on dit que la variable a été capturée.*

$$\frac{t \to_\beta t'}{t\ u \to_\beta t'\ u} \qquad \frac{u \to_\beta u'}{t\ u \to_\beta t\ u'} \qquad \frac{t \to_\beta t'}{\lambda x.t \to_\beta \lambda x.t'} \qquad \frac{}{(\lambda x.t)\ u \to_\beta t\{x \setminus u\}}$$

FIGURE 2.3 – Définition de la bêta-réduction

$$\begin{array}{rcl}
(\lambda f.f\ (f\ (\lambda a.a)))\ (\lambda z.\lambda y.\lambda x.z\ (y\ x)) & \to_\beta & (\lambda z.\lambda y.\lambda x.z\ (y\ x))\ ((\lambda z.\lambda y.\lambda x.z\ (y\ x))\ (\lambda a.a)) \\
(\lambda f.f\ (f\ (\lambda a.a)))\ (\lambda z.\lambda y.\lambda x.z\ (y\ x)) & \to_\beta & (\lambda z.\lambda y.\lambda x.z\ (y\ x))\ ((\lambda z.\lambda y.\lambda x.z\ (y\ x))\ (\lambda a.a)) \\
 & =_\alpha & (\lambda z.\lambda y.\lambda x.z\ (y\ x))\ ((\lambda z_1.\lambda y_1.\lambda x_1.z_1\ (y_1\ x_1))\ (\lambda a.a))
\end{array}$$

FIGURE 2.4 – Illustration de l'alpha-conversion d'un terme

Les variables peuvent avoir deux états : libre ou liée. Il y a donc des variables libres qui se font capturer, et des variables liées. Quand une variable liée change de lieur après une bêta-réduction, on dit également qu'elle est capturée.

Notre définition de la substitution ne comprend aucune règle qui déclenche une capture. Dans le cas où la substitution aurait pu déclencher une capture elle n'est pas définie. Nous allons voir par la suite comment faire en sorte que notre substitution ne traite que des cas pour lesquels son comportement est défini.

Renommage et convention de Barendregt La manière la plus classique de représenter les variables est d'utiliser des noms, comme présenté jusqu'à présent. Cette méthode présente l'avantage d'être lisible. Toutefois, le problème est que notre substitution n'est pas complètement définie. Comme vu dans le paragraphique précédent sur la capture, la réduction $(\lambda y.\lambda x.y)\ x \to_\beta ...$, n'est pas définie.

Une première manière de faire pour que la substitution fonctionne est de renommer les lieurs et leurs variables avec des noms de variables qui n'existent pas dans le terme. Ce renommage s'appelle l'alpha-conversion, noté aussi α-conversion. Le terme $(\lambda y.\lambda x.y)\ x$ peut être renommé $(\lambda y.\lambda z.y)\ x$ sans que cela ne change la signification du terme. Cela permet d'effectuer la réduction $(\lambda y.\lambda z.y)\ x \to_\beta \lambda z.x$.

De la même manière, dans la figure 2.4, on voit qu'il n'y a pas de variable libre dans le terme de base et qu'il n'y a pas deux fois le même nom pour un lieur. Toutefois, le problème se pose quand même. Il est donc nécessaire de vérifier si l'on doit procéder à un renommage après chaque bêta-réduction. Par exemple, comme le montre la figure 2.4, après la première réduction, un renommage est nécessaire car les abstractions ont été dupliquées.

La solution du renommage est utilisée lors de l'implémentation de cette représentation des lambda-termes. Le fait de renommer correctement les termes après chaque substitution, lorsque c'est nécessaire, nous garantit que nous traitons des termes qui n'ont pas subi de capture. La convention de Barendregt [22] nous dit à quel moment il faut renommer un terme.

Dans le reste du manuscrit, pour plus de facilité, on considère les termes modulo alpha-conversion, de sorte que $\lambda x.x = \lambda y.y$. Par conséquent, on peut supposer que tous les termes manipulés respectent la convention de Barendregt.

Représentation alternative : indice de de Bruijn Une autre manière de représenter les variables est d'utiliser les indices de de Bruijn : les variables sont représentées par des indices qui correspondent au nombre de lieurs qui les séparent de leur lieur. Dans le terme $(\lambda.((\lambda.1)\ 0))\ (\lambda.0)$, le 0 bleu et le 0 rouge font référence aux abstractions qui sont directement au-dessus d'eux alors que le 1 bleu fait référence à l'abstraction un niveau au-dessus. L'avantage est qu'il n'y a plus d'ambiguïté car chaque variable fait directement référence à son lieur. En revanche, la substitution est plus compliquée car il faut modifier les indices pour qu'ils correspondent bien au même lieur, ce qu'on appelle le lifting. Dans le cas où le terme substitué ne contient pas de variable liée à une abstraction extérieure au terme, il n'y a pas de décalage d'indice à effectuer. Par exemple, dans la réduction $(\lambda.0)\ (\lambda.0) \to_\beta \lambda.0$, la variable 0 qui est dans l'argument est liée à l'abstraction qui se trouve également dans l'argument, donc il n'est pas nécessaire de décaler l'indice de la variable. En revanche, dans la réduction $\lambda.((\lambda.\lambda.1)\ 0) \to_\beta \lambda.(\lambda.1)$, la variable 0 est liée à une abstraction qui se trouve à l'extérieur du terme qui va être substitué, donc il faut décaler son indice du nombre d'abstractions qui la sépare de son lieur, ici un.

Un autre avantage des indices de de Bruijn est qu'il n'y a qu'une seule manière de représenter un terme, alors qu'avec l'alpha-renommage dans le nommage classique, il y en a une infinité. Cela est plus pratique pour comparer les termes entre eux. Les termes $\lambda a.a$, $\lambda b.b$, $\ldots$, $\lambda z.z$ sont tous α-équivalents et correspondent au même terme avec les indices de de Bruijn $\lambda.0$. Une implémentation d'un calcul qui utilise les indices de de Bruijn est présentée plus tard dans le chapitre 6.

Une autre manière de gérer les variables liées est présentée plus tard dans le chapitre 5, et elle utilise un nouveau quantificateur appelé nabla, qui introduit de nouvelles variables.

Calcul Un calcul est la combinaison d'une syntaxe et d'une ou plusieurs règles de réduction. Le lambda-calcul pur ne met pas de restriction sur les endroits où la bêta-réduction peut être appliquée. Les stratégies choisissent parmi toutes les possibilités que leur propose un calcul.

On peut représenter l'ensemble des réductions possibles par un graphique (Fig. 2.5). Certains chemins dans le graphique peuvent boucler, d'autres peuvent grandir à l'infini, d'autres peuvent finir en différents nombres d'étapes de réduction. Pour plus de lisibilité, le lambda-terme $(\lambda x.xx)\ (\lambda x.xx)$, qui est un lambda-terme qui s'évalue forcément en lui-même, sera représenté par Ω.

On constate que la forme la plus réduite du terme $((\lambda x.\lambda y.xx)\ ((\lambda z.z)\ a))\ \Omega$ est aa. On voit aussi qu'il y a plusieurs chemins pour y arriver. Le graphique contient un cycle, ce qui veut dire que certains chemins bouclent infiniment, notamment tant qu'il reste Ω dans le terme. Mais il peut aussi y avoir des termes qui bouclent sur plusieurs étapes. Dans notre exemple, on voit que les chemins pour arriver à la forme la plus réduite peuvent avoir des tailles différentes. On constate que ce graphique particulier

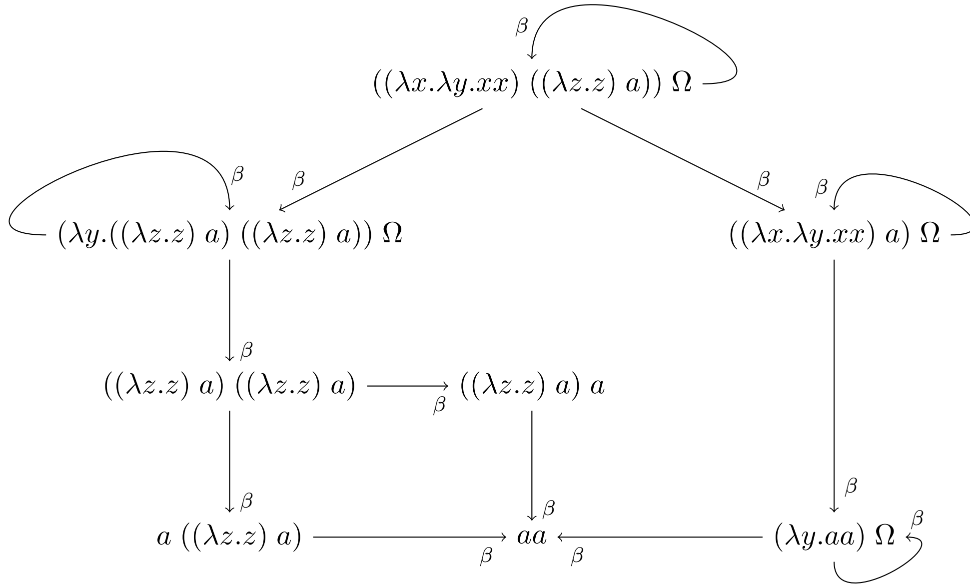

FIGURE 2.5 – Graphique complet de réduction du terme $((\lambda x.\lambda y.xx)\ ((\lambda z.z)\ a))\ \Omega$

est fini, ce n'est pas forcément le cas. Il y a aussi des termes qui grandissent infiniment. L'un des termes les plus connus qui grandit infiniment est le combinateur de point fixe de Turing Θ présenté dans la figure 2.6.

Confluence Le lambda-calcul possède la propriété de confluence, ce qui signifie que lorsqu'un terme t peut être réduit en deux termes différents v et w, alors il existe des séquences de réduction les menant au même terme u, comme illustré dans la figure 2.7.

Type Dans le lambda-calcul typé, les règles de typage définissent comment les types sont associés aux lambda-termes et comment les opérations sont typées. Par exemple, une règle pourrait spécifier que l'application d'une fonction à un argument est valide si le type de l'argument correspond au type attendu par la fonction.

Il existe différents systèmes de typage pour le lambda-calcul, tels que le système de types simples, le système de types polymorphes et le système des types dépendants. Chaque système offre différents niveaux d'expressivité et de vérification statique.

Ici, nous voulons traiter tous les termes sans restriction, c'est pourquoi nous utilisons le lambda-calcul non typé, qui nous permet de traiter tous les termes que le lambda-calcul peut créer.

On utilise également un certain nombre de lemmes simples que l'on appelle ici propositions.

$$
\begin{array}{rcl}
\Theta & = & \Delta\Delta \\
\Delta & = & (\lambda x.\lambda y.(y(xxy))) \\
\Delta\Delta & \rightarrow_\beta & \lambda y.(y(\Delta\Delta y)) \\
 & \rightarrow_\beta & \lambda y.(y(\lambda y.(y(\Delta\Delta y))y)) \\
 & \rightarrow_\beta & ...
\end{array}
$$

FIGURE 2.6 – Combinateur de point fixe de Turing

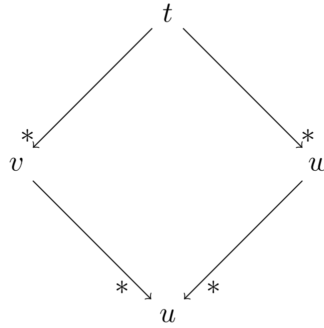

FIGURE 2.7 – Confluence

Proposition 1 (Compatibilité de la substitution avec la réduction). *Si* $t \rightarrow_\beta t'$ alors $t\{x \setminus u\} \rightarrow_\beta t'\{x \setminus u\}$

Proposition 2 (Arguments sans variable substituée). *Si* $x \notin \mathrm{FV}(u)$ *alors* $t\{x \setminus w\}\ u = (t\ u)\{x \setminus w\}$

Proposition 3 (Emboîtement des substitutions). *Si* $y \notin \mathrm{FV}(t)$ *alors* $t\{x \setminus u\{y \setminus w\}\} = t\{x \setminus u\}\{y \setminus w\}$

Proposition 4 (Substitution sans cible). *Si une variable n'est pas libre dans un terme, alors la substitution de cette variable dans ce terme n'aura pas d'impact.*
$x \notin \mathrm{FV}(t) \implies t\{x \setminus u\} = t$

Proposition 5 (Variable libre et substitution). *Les variables libres d'une substitution, dans le cas où la variable substituée est présente dans le terme, sont les variables libres des deux sous-termes privés de la variable en question.*
$x \in \mathrm{FV}(t) \implies \mathrm{FV}(t\{x \setminus u\}) = (\mathrm{FV}(t) \setminus \{x\}) \cup \mathrm{FV}(u)$

Proposition 6 (Valeur et substitution). *Un terme qui n'est pas une valeur et qui est substitué par un autre terme qui n'est pas une valeur n'est pas une valeur.*
$(\forall t, u, x, \quad t \notin Val \land u \notin Val \implies t\{x \setminus u\} \notin Val)$

Proposition 7 (Substitution d'une forme normale)**.** *Si tous les sous-termes d'une substitution implicite sont en forme normale et que la substitution est appliquée sur un terme qui n'est pas une abstraction, alors la substitution implicite est en forme normale.*
$(\forall t, u, x, \quad t \in \mathscr{N}_\lambda \wedge u \notin \mathit{Val} \wedge u \in \mathscr{N}_\lambda \implies t\{x \setminus u\} \in \mathscr{N}_\lambda)$.

Démonstration. Par induction sur la dérivation de t.

- VAR :
  - Si $t = x$ avec $u \in \mathscr{N}_\lambda$ alors $x\{x \setminus u\} = u$ et $u \in \mathscr{N}_\lambda$.
  - Si $t = y$ alors $y\{x \setminus u\} = y$ est $y \in \mathscr{N}_\lambda$ (règle VAR).
- APP $t = v\ w$: Par définition de $(v\ w) \in \mathscr{N}_\lambda$ on a $v \in \mathscr{N}_\lambda$, $v \notin \mathit{Val}$ et $w \in \mathscr{N}_\lambda$. Par définition $(v\ w)\{x \setminus u\} = v\{x \setminus u\}\ w\{x \setminus u\}$. Par hypothèse d'induction sur v avec u on a $v\{x \setminus u\} \in \mathscr{N}_\lambda$ et par hypothèse d'induction sur w avec u on a $w\{x \setminus u\} \in \mathscr{N}_\lambda$. Et par la proposition 6 sur v, u, x on a $v\{x \setminus u\} \notin \mathit{Val}$. Donc par la règle APP on a $(v\ w)\{x \setminus u\} \in \mathscr{N}_\lambda$
- ABS $t = \lambda y.v$: Par définition de $(\lambda y.v) \in \mathscr{N}_\lambda$ on a $v \in \mathscr{N}_\lambda$. Par définition on a $(\lambda y.v)\{x \setminus u\} = \lambda y.v\{x \setminus u\}$. Puis par hypothèse d'induction sur v on a $v\{x \setminus u\} \in \mathscr{N}_\lambda$ donc $(\lambda y.v)\{x \setminus u\} \in \mathscr{N}_\lambda$. □

2.1.2 Stratégie d'évaluation

Le lambda-calcul permet d'appliquer la bêta-réduction sur n'importe quel radical. Une stratégie de réduction est une manière de choisir le prochain radical à réduire.

Il y a plusieurs types de stratégies de réduction, ici, nous allons en étudier deux : les stratégies de normalisation, qui cherchent à réduire un terme au maximum, que nous verrons plus en détail dans la section 2.2.2, et les stratégies d'évaluation, qui cherchent à réduire un terme jusqu'à l'obtention d'une valeur. Traditionnellement, les valeurs du lambda-calcul sont les abstractions.

Les stratégies d'évaluation s'arrêtent dès qu'elles ont trouvé une valeur. Elle ne font que des étapes de réduction faible qui ne vont pas évaluer sous les lambda-abstractions car ce sont déjà des valeurs. Dans la figure 2.8, les étapes de réduction faible sont en bleu. Elles permettent d'atteindre une valeur, mais pas d'aller jusqu'à la forme normale. On ne peut prendre que le chemin bleu qui s'arrête avant la forme la plus réduite. Même si des flèches subsistent dans le graphique, on ne peut pas continuer. On observe également qu'une flèche bleue se trouve seule à droite du graphique, mais elle n'est pas accessible depuis un chemin bleu.

Appel par valeur La majorité des langages de programmation utilisent l'appel par valeur, qui consiste à évaluer les arguments d'une fonction avant de résoudre l'appel lui-même. Cette stratégie évalue exactement une fois chaque argument, ce qui peut s'avérer avantageux lorsque un argument est utilisé plusieurs fois dans le corps de la fonction. De plus, elle peut évaluer le membre gauche ou droit d'une application au choix sans que cela change l'issue de l'évaluation. Par exemple, dans la figure 2.9, on peut voir que le chemin qui correspond à l'appel par valeur (le chemin bleu) est le plus court, car il n'évalue l'argument qu'une seule fois.

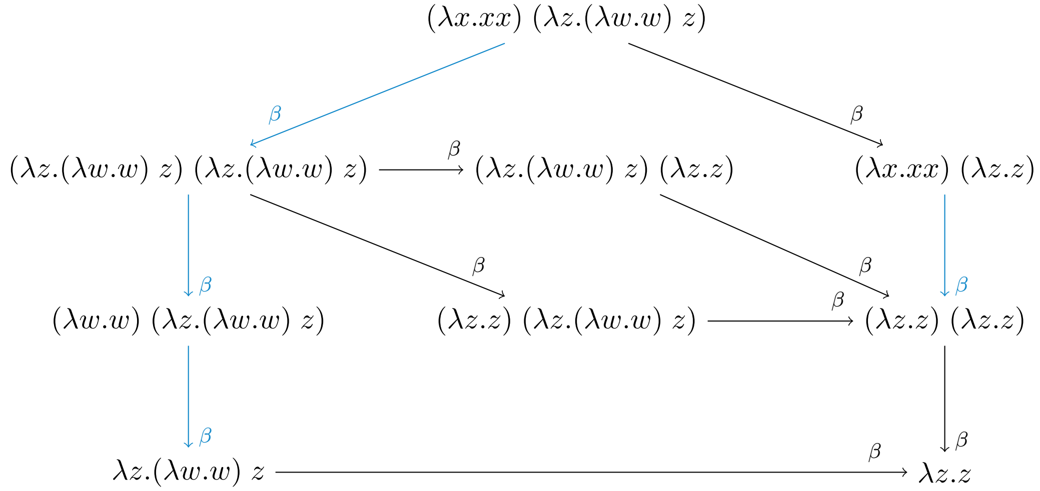

FIGURE 2.8 – Graphique de réduction du terme $(\lambda x.xx)\ (\lambda z.(\lambda w.w)\ z)$. Les flèches bleues représentent les réductions faibles.

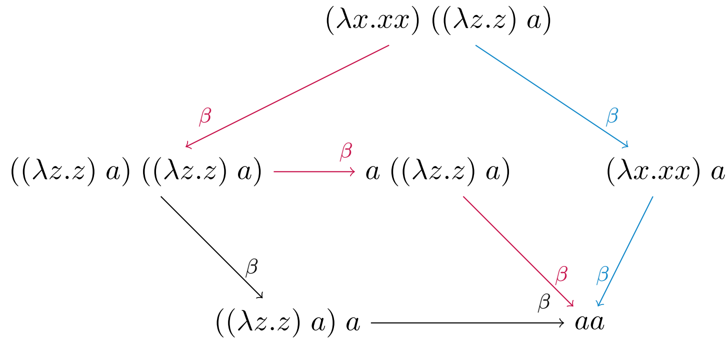

FIGURE 2.9 – Graphique complet de réduction du terme $(\lambda x.xx)\ ((\lambda z.z)\ a)$. Le chemin représenté par les flèches bleues correspond à l'appel par valeur et le chemin représenté par les flèches rouges correspond à l'appel par nom.

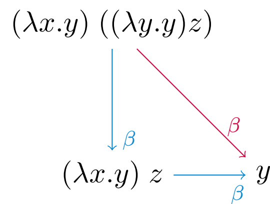

FIGURE 2.10 – Graphique complet de réduction du terme $(\lambda x.y)\ ((\lambda y.y)z)$. Le chemin représenté par les flèches bleues correspond à l'appel par valeur et le chemin représenté par les flèches rouges correspond à l'appel par nom.

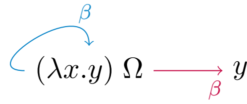

FIGURE 2.11 – Graphique complet de réduction du terme $(\lambda x.y)\ \Omega$. Le chemin représenté par les flèches bleues correspond à l'appel par valeur et le chemin représenté par les flèches rouges correspond à l'appel par nom.

Dans le cas où l'argument n'est pas utilisé, l'appel par valeur va conduire à des calculs inutiles. Comme on peut le voir dans la figure 2.10, le chemin bleu n'est pas le plus court. On constate que le chemin rouge obtient le résultat plus rapidement.

Le cas où l'appel par valeur effectue des calculs superflus pour obtenir le résultat n'est pas le pire cas. Si l'argument diverge c'est-à-dire que son évaluation n'aboutit jamais à un résultat, tenter de l'évaluer fera diverger l'évaluation du terme alors que le terme a un résultat. Par exemple, dans la figure 2.11, le chemin bleu diverge.

La stratégie d'appel par valeur présente l'avantage de ne pas dupliquer les calculs, mais elle comporte le risque de provoquer des calculs inutiles qui pourraient, dans le pire des cas, entraîner la divergence de l'évaluation d'un terme qui pourrait être évalué.

Appel par nom La stratégie d'appel par nom, utilisée entre autres pour les macros, par exemple dans LaTeX, n'évalue pas les arguments d'une abstraction. Elle commence par appliquer l'abstraction, les arguments seront évalués au dernier moment, lorsqu'ils seront rencontrés.

Les arguments sont évalués exactement le nombre de fois qu'ils sont utilisés par le corps de la fonction, ce qui a pour avantage de ne pas effectuer de calculs inutiles. Par exemple, dans la figure 2.10, l'appel par nom (le chemin rouge) ne calcule pas l'argument inutilement. Et dans certains cas, le fait de ne pas procéder à des réductions inutiles nous permet d'éviter la divergence. Ainsi dans la figure 2.11, le chemin rouge trouve le résultat.

En revanche, si l'argument est présent plusieurs fois dans le corps de l'abstraction, l'appel par nom va l'évaluer autant de fois que rencontré. Par exemple, dans la figure 2.9, on voir que le chemin rouge évalue deux fois l'argument.

On voit également dans la figure 2.9 qu'il y a un chemin qui n'est pas colorié. C'est un chemin qui n'appartient à aucune des deux stratégies précédemment présentées. Dans ces stratégies, lorsqu'il y a une application entre deux termes et que le terme de gauche n'est pas une abstraction, on va systématiquement réduire le terme de gauche. Une fois que le terme de gauche a été réduit à une valeur, on va alors décider de la réduction à effectuer en fonction de la stratégie d'évaluation choisie, en particulier dans l'appel par nom où les arguments ne sont pas évalués avant l'application de l'abstraction.

La stratégie d'appel par nom est donc moins efficace dans le cas où les arguments sont présents plus d'une fois dans le corps de l'abstraction. Mais elle ne fait pas de calcul inutile, ce qui est très intéressant car cela nous donne la garantie que s'il y a un chemin vers le résultat, alors la stratégie le trouvera.

$$
\begin{aligned}
(\lambda x.xx)\ ((\lambda z.z)\ a) &\rightarrow_\beta xx \text{ avec } x = (\lambda z.z)\ a \\
&\rightarrow_\beta xx \text{ avec } x = a \\
&\rightarrow_\beta aa
\end{aligned}
$$

FIGURE 2.12 – Stratégie d'appel par nécessité appliquée au terme $(\lambda x.xx)\ ((\lambda z.z)\ a)$. Une manière de représenter le «avec» est présentée dans la section 2.5.

Appel par nécessité L'appel par nécessité [5] est une extension de l'appel par nom qui, grâce à un système de partage, permet de conserver en mémoire les valeurs des arguments évalués. Ainsi, il présente les avantages des deux stratégies précédentes : un argument n'est pas évalué s'il n'est pas nécessaire au sein du corps de la fonction et on a la garantie de trouver le résultat s'il existe. En revanche, si l'argument est présent dans le corps de la fonction, il n'est évalué qu'une seule fois.

L'appel par nécessité va directement évaluer le corps de l'abstraction en gardant une référence vers les arguments. La première fois que cette référence est rencontrée lors de l'évaluation, l'argument sera évalué une fois pour toutes. Les autres fois où la référence est rencontrée, le terme aura déjà été évalué.

Dans les figures 2.10 et 2.11, cela correspond au chemin rouge, car l'argument n'est pas présent dans le corps de l'abstraction.

Une illustration de la stratégie d'appel par nécessité est présentée dans la figure 2.12.

2.2 Normalisation

Dans cette section, nous étudions la normalisation, qui consiste à réduire un terme jusqu'à une forme irréductible. Dans la section 2.2.1, nous commençons par définir les formes normales et les différentes catégories de termes. Certains termes n'ont pas de forme normale, alors que d'autres en ont qui sont plus ou moins accessibles. Après avoir présenté ces différents termes, nous nous intéressons, dans la section 2.2.2, à la manière d'obtenir ces formes normales via deux types de stratégies de réduction.

2.2.1 Définition

Généralement, les lambda-termes peuvent être réduits.

Définition 8 (Formes normales)**.** *Lorsqu'un terme ne peut plus être réduit, on dit qu'il est en forme normale.*

Grâce à la propriété de confluence énoncée dans la section 2.1.1, chaque lambda-terme a au plus une forme normale. Il y a donc des termes sans forme normale et des termes qui en ont une, que l'on appelle termes normalisables. Un cas particulier des termes normalisables sont les termes fortement normalisants, lorsque tous les chemins mènent à la forme normale.

$$\frac{\text{VAR}}{x \in \mathcal{N}_\lambda} \qquad \frac{\text{APP} \quad t \notin Val \quad t \in \mathcal{N}_\lambda \quad u \in \mathcal{N}_\lambda}{t\ u \in \mathcal{N}_\lambda} \qquad \frac{\text{ABS} \quad t \in \mathcal{N}_\lambda}{\lambda x.t \in \mathcal{N}_\lambda}$$

FIGURE 2.13 – Caractérisation des formes normales du lambda-calcul

Définition 9 (Caractérisation des formes normales)**.** *On donne une caractérisation syntaxique des formes normales dans la figure 2.13.*

Terme sans forme normale Il y a des termes qui n'ont pas de forme normale, cela peut se manifester de deux manières différentes : les termes qui grandissent infiniment, par exemple Θ, présenté dans la figure 2.6, et les termes qui bouclent, c'est-à-dire qui se réduisent vers eux-mêmes après une ou plusieurs étapes de réduction, le plus connu étant Ω.

Le graphique complet qui représente les réductions d'un terme qui boucle est un graphique qui contient un cycle. Par ailleurs, le graphique qui représente les réductions d'un terme qui grandit de manière infinie n'est pas un graphique fini.

Définition 10 (Terme fortement normalisant)**.** *On dit qu'un terme est fortement normalisant si tous les chemins de réduction mènent à la forme normale. Grâce à la confluence, si un terme n'a pas de chemin de réduction qui diverge, alors tous les chemins vont à la forme normale. Les graphiques correspondant à un terme fortement normalisant sont finis et acycliques.*

On voit dans la figure 2.14 que le graphique est fini et acyclique donc le terme présenté est fortement normalisant.

Définition 11 (Terme normalisable)**.** *Il y a des termes qui ont une forme normale mais qui ont aussi des chemins qui divergent. Toutefois, si un terme est normalisable, il le reste même après un nombre arbitraire de réductions. Autrement dit, il est impossible de s'enfermer dans un cycle : il est toujours possible de trouver un chemin qui nous mène à la forme normale. Cette propriété découle de la propriété de confluence : si un terme a deux réductions possibles, un qui le mène à la forme normale et un qui boucle, alors la propriété de confluence implique que le chemin qui boucle peut aussi trouver la forme normale.*

La figure 2.15 présente le graphique de réduction d'un terme normalisable. On constate dans un premier temps que le graphique contient un cycle, ce qui veut dire qu'il est possible de prendre un chemin qui ne mène pas à la forme normale. Cependant, quelque soit l'endroit où l'on se trouve, il est toujours possible de trouver un chemin pour aller à la forme normale.

Il y a donc des stratégies de réduction qui ont pour objectif de trouver la forme normale d'un terme lorsque celle-ci existe.

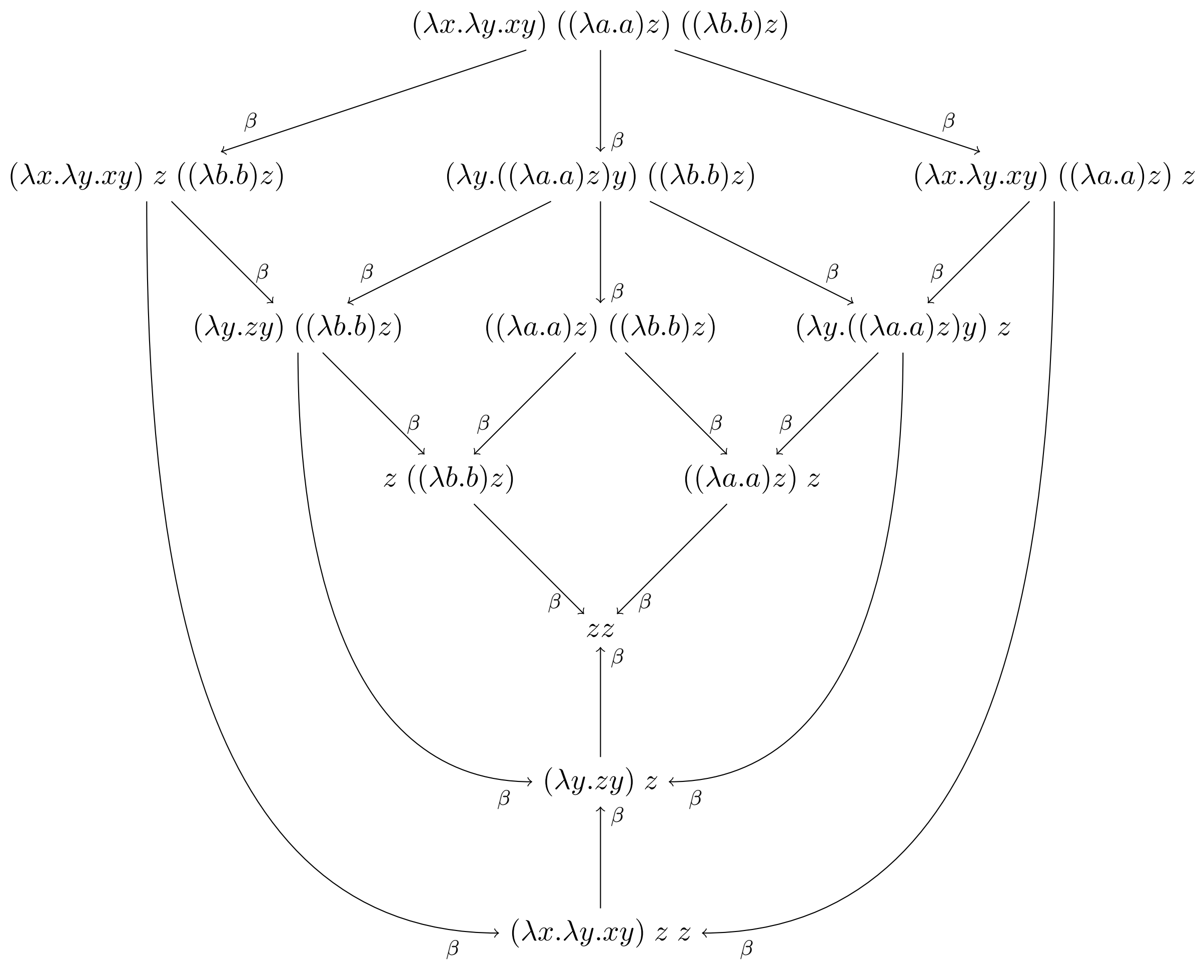

FIGURE 2.14 – Graphique complet de réduction du terme $(\lambda x.\lambda y.xy)\ ((\lambda a.a)z)\ ((\lambda b.b)z)$

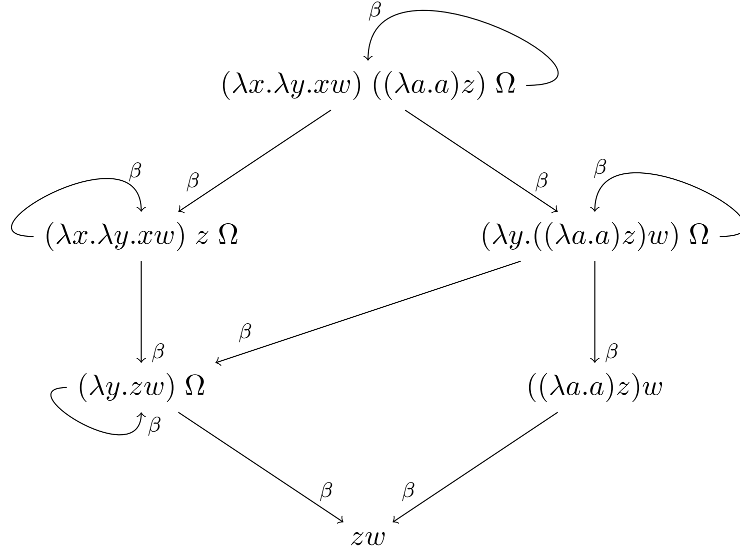

Figure 2.15 – Graphique complet de réduction du terme $(\lambda x.\lambda y.xw)\ ((\lambda a.a)z)\ \Omega$

2.2.2 Stratégie de réduction normalisante

Les stratégies d'évaluation évaluent les termes jusqu'à obtenir une valeur, c'est-à-dire une abstraction. Cependant, elles ne vont pas réduire sous les abstractions, ce qui les fait s'arrêter avant d'avoir atteint la forme normale. Par exemple, le terme $\lambda x.((\lambda y.y)x)$ est considéré comme une valeur, donc les stratégies d'évaluation ne vont pas aller plus loin alors que le terme n'est pas en forme normale.

On appelle «stratégie de réduction faible» les stratégies de réduction qui ne vont pas réduire sous les abstractions. Toutes les stratégies d'évaluation sont des stratégies de réduction faible. C'est ce qui est utilisé dans les langages de programmation.

Définition 12 (Stratégie de réduction normalisante)**.** *Les stratégies de réduction normalisante réduisent les termes jusqu'à leur forme normale.*

Nous nous intéressons ici à trouver une stratégie de réduction qui normalise les termes. Ce genre de stratégie est très utile pour les systèmes d'évaluation partielle ou pour les assistants de preuve, car elle permet de déterminer l'équivalence entre deux termes.

Actuellement, les systèmes qui souhaitent obtenir une forme normale utilisent des stratégies de réduction faible qu'ils appliquent sur les différents sous-termes jusqu'à obtenir une forme normale [21, 9].

Ces stratégies fortes sont généralement conservatrices par rapport à leur stratégie sous-jacente faible et elles procèdent souvent par application itérative d'une stratégie

Appel par valeur	Appel par nécessité	Appel par nécessité fort
$(\lambda x.\lambda z.z(xx))\ (\lambda y.I\ y)$	$(\lambda x.\lambda z.z(xx))\ (\lambda y.I\ y)$	$(\lambda x.\lambda z.z(xx))\ (\lambda y.I\ y)$
$\lambda z.z\ ((\lambda y.I\ y)\ (\lambda y.I\ y))$	$\lambda z.z(xx)$ avec $x = (\lambda y.I\ y)$	$\lambda z.z(xx)$ avec $x = (\lambda y.I\ y)$
$\lambda z.z\ ((\lambda y.I\ y)\ (\lambda y.I\ y))$	$\lambda z.z(xx)$ avec $x = (\lambda y.I\ y)$	$\lambda z.z(xx)$ avec $x = (\lambda y.y)$
$\lambda z.z\ (I\ (\lambda y.I\ y))$	$\lambda z.z\ ((\lambda y.I\ y)\ x)$ avec $x = (\lambda y.I\ y)$	$\lambda z.z\ ((\lambda y.y)\ x)$ avec $x = (\lambda y.y)$
$\lambda z.z\ (\lambda y.I\ y)$	$\lambda z.z\ (I\ x)$ avec $x = (\lambda y.I\ y)$	$\lambda z.zx$ avec $x = (\lambda y.y)$
$\lambda z.z\ (\lambda y.I\ y)$	$\lambda z.zx$ avec $x = (\lambda y.I\ y)$	$\lambda z.z\ (\lambda y.y)$
$\lambda z.z\ (\lambda y.y)$	$\lambda z.z\ (\lambda y.I\ y)$	
	$\lambda z.z\ (\lambda y.I\ y)$	
	$\lambda z.z\ (\lambda y.y)$	

FIGURE 2.16 – Tableau qui présente trois stratégies qui permettent de trouver la forme normale du terme $(\lambda x.\lambda z.z(xx))\ (\lambda y.I\ y)$. Pour plus de lisibilité la fonction identité est représentée par I. Les barres horizontales représentent le fait qu'on relance l'évaluation sous l'abstraction. La partie du terme qui n'est plus dans l'évaluation est représentée en gris. Quand la fonction identité va être réduite elle est en rouge.

faible sur des termes ouverts. En d'autres termes, elles utilisent une forme restreinte de réduction forte pour permettre la réduction à la forme normale, mais elles n'essaient pas de profiter de la réduction forte pour obtenir des séquences de réduction plus courtes. Depuis que l'appel par nécessité a été utilisé pour capturer une réduction faible optimale [8], on sait que l'utilisation délibérée de la réduction forte [23] est la seule façon d'autoriser des séquences de réduction plus courtes.

La figure 2.16 présente trois manières d'obtenir la forme normale d'un terme. Dans ces trois stratégies, on examine le nombre de fois où l'application de la fonction identité I est réduite. Avec les stratégies faibles itérées, lorsque l'évaluation est relancée, les termes sont séparés dans le tableau par une barre et les parties du terme qui ne sont plus en jeu dans l'évaluation sont affichées en gris.

Dans les stratégies faibles itérées, lorsque l'on relance l'évaluation sous une abstraction, la variable associée à cette abstraction devient libre. Alors que, dans les stratégies fortes, lorsque l'on autorise la réduction sous une abstraction, la variable associée reste liée, et devra être gérée d'une manière particulière.

Dans la première colonne, on utilise l'appel par valeur itéré. L'appel par valeur a l'avantage de ne pas dupliquer le calcul quand un argument est présent plusieurs fois dans le corps de l'abstraction. Pourtant, on voit dans le tableau que le radical $I\ y$ dans l'argument $\lambda y.I\ y$ est réduit deux fois. Cela vient du fait que l'argument est une valeur. Donc, la stratégie d'appel par valeur ne va pas plus le réduire. On voit aussi que l'évaluation est appelée une deuxième fois à la fin pour aller réduire dans le corps de l'abstraction afin d'obtenir la forme normale.

La deuxième colonne présente l'appel par nécessité itéré qui a le même problème que l'appel par valeur : le terme partagé (l'argument) est une valeur, donc il n'est pas plus réduit. Cela signifie que certaines parties de l'argument sont également réduites deux fois.

Nous nous intéressons aux stratégies de réduction forte, qui évaluent une expres-

$$
\begin{aligned}
(\lambda x.\lambda y.x)\ (I\ z)\ (I\ z) &\rightarrow_\beta (\lambda y.(I\ z))\ (I\ z) \\
&\rightarrow_\beta (I\ z) \\
&\rightarrow_\beta z
\end{aligned}
$$

FIGURE 2.17 – Étapes de réduction du terme $(\lambda x.\lambda y.x)\ (I\ z)\ (I\ z)$ avec la fonction identité représentée par I.

sion sous des abstractions pour obtenir un terme en forme normale. Dans la troisième colonne, nous utilisons l'appel par nécessité de manière forte, ce qui signifie que l'expression est réduite sous des abstractions. De la même manière que dans l'appel par nécessité itéré, l'argument est partagé. La différence par rapport aux stratégies faibles itérées est que l'argument est normalisé avant d'être substitué, ce qui signifie qu'il n'est réduit qu'une fois.

2.3 Position nécessaire

Définition 13 (Position nécessaire)**.** *On considère comme nécessaires tous les radicaux d'un terme qui contribuent à l'obtention de la forme normale du terme.*

On dit aussi qu'un terme est à une position nécessaire si le terme qui est à cet endroit est nécessaire. Ainsi, nous cherchons à trouver tous les radicaux nécessaires dans un terme pour obtenir sa forme normale.

Dans la section 2.3.1, nous présentons la notion de nécessité d'une position dans un terme. Dans la section 2.3.2, nous présentons les variables gelées et les positions isolées, qui sont les deux premières notions qui nous permettent de caractériser les positions nécessaires.

2.3.1 Notion de nécessité

On parle de nécessité d'une position dans un terme normalisable ou d'une nécessité d'un terme ou d'une réduction. Le fait qu'un radical soit nécessaire dépend uniquement du contexte dans lequel il se place.

Par exemple, si le même radical apparaît à deux positions différentes dans un terme, il peut être nécessaire à une position mais pas à l'autre. La figure 2.17 représente les étapes de réduction du terme $(\lambda x.\lambda y.x)\ (I\ z)\ (I\ z)$ dans lequel il y a deux fois le même radical $I\ z$. On constate qu'il n'y a que le premier qui est nécessaire.

De la même manière, si une position est nécessaire, quelque soit le terme qui s'y trouve il est nécessaire. Par exemple, dans la figure 2.18, quelque soit le sous-terme t dans le terme $(\lambda x.x)\ t$, il est nécessaire.

On utilise deux définitions différentes pour déterminer si un terme est en position nécessaire ou non.

$$
\begin{aligned}
(\lambda x.x)\ (I\ a) &\rightarrow_\beta\ I\ a \\
&\rightarrow_\beta\ a
\end{aligned}
$$

$$
\begin{aligned}
(\lambda x.x)\ (I\ b) &\rightarrow_\beta\ I\ b \\
&\rightarrow_\beta\ b
\end{aligned}
$$

FIGURE 2.18 – Réduction du terme $(\lambda x.x)\ t$ avec la fonction identité représentée par I et deux sous-terme différents pour t.

Dans un terme t normalisable, si on remplace un sous-terme s par un autre s' et que cela modifie la forme normale du terme t, cela signifie que le sous-terme s est à une position nécessaire.

Cela ne veut pas dire que, si la forme normale ne change pas, le sous-terme n'est pas (nécessairement) à une position nécessaire. Il est possible de remplacer le sous-terme par un autre qui a les mêmes propriétés, par exemple $\lambda x.x$ par $\lambda x.(\lambda y.y)\ x$.

À l'inverse, pour montrer qu'un radical r dans un terme t normalisable n'est pas nécessaire, on peut construire le graphique de réduction de t et, si l'on arrive à trouver un chemin de réduction sans réduire r, cela signifie que ce radical r est à une position non nécessaire (on peut également dire que ce radical n'est pas nécessaire). Alternativement, on peut remplacer r par Ω, et voir si le terme t' obtenu est toujours normalisable.

En utilisant ces deux critères, nous pourrons déterminer quels termes sont nécessaires et lesquels ne le sont pas dans le terme $(\lambda x.\lambda y.x)\ (I\ a)\ (I\ b)\ (I\ c)$. La figure 2.19 présente toutes les positions nécessaires dans ce terme se réduit en ac avec les étapes de réduction présentées dans la figure 2.20.

Dans ce terme, on voit que $I\ a$ et $I\ c$ sont tous les deux à une position nécessaire car si on modifie ces deux sous-termes par une variable libre, la forme normale change. Par exemple, dans la figure 2.21, $(\lambda x.\lambda y.x)\ g\ (I\ b)\ h$ se réduit en gh.

En revanche, on peut constater sur le graphique 2.22 que le sous-terme $I\ b$ est à une position qui n'est pas nécessaire, car il existe un chemin de réduction qui ne le réduit pas.

On voudrait donc pouvoir écrire des règles qui trouvent toutes les positions nécessaires d'un terme. Malheureusement, il a été prouvé par Barendregt, Bergstra, Klop et Volken que c'était indécidable [22, 10]. C'est pourquoi nous devons nous contenter d'identifier un sous-ensemble de positions nécessaires que l'on espère suffisant pour obtenir une forme normale. Mais si l'ensemble ne décrit pas assez de positions, il est possible que nous ne puissions pas atteindre le résultat souhaité.

Par exemple, si on prend comme sous-ensemble uniquement les termes à gauche des applications, il est possible de se retrouver bloqué. Avec cette règle, le terme $a\ ((\lambda x.x)\ b)$ est bloqué, car il n'est pas possible de réduire le radical $(\lambda x.x)\ b$, alors qu'il n'est pas en forme normale.

En fonction du type de stratégie utilisée, le résultat recherché n'est pas le même.

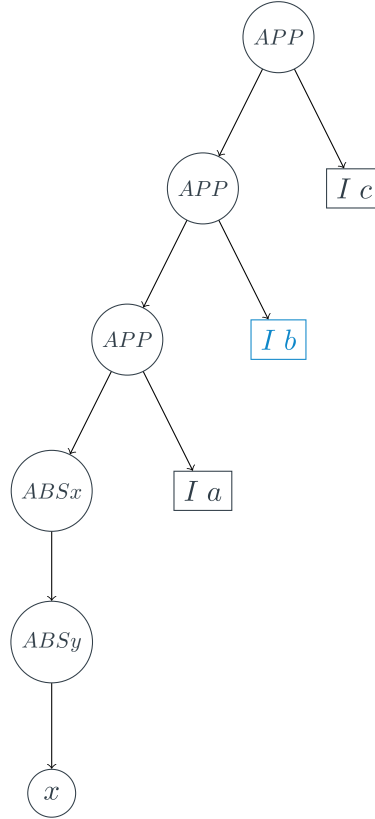

FIGURE 2.19 – Identification des positions nécessaires dans le terme $(\lambda x.\lambda y.x)\ (I\ a)\ (I\ b)\ (I\ c)$. Le vert exprime une position nécessaire. Les rectangles sont des termes.

$$\begin{aligned}(\lambda x.\lambda y.x)\ (I\ a)\ (I\ b)\ (I\ c) &\to_\beta (\lambda y.(I\ a))\ (I\ b)\ (I\ c)\\ &\to_\beta (I\ a)\ (I\ c)\\ &\to_\beta a\ (I\ c)\\ &\to_\beta ac\end{aligned}$$

FIGURE 2.20 – Étapes de réduction du terme $(\lambda x.\lambda y.x)\ (I\ a)\ (I\ b)\ (I\ c)$

$$\begin{aligned}(\lambda x.\lambda y.x)\ g\ (I\ b)\ h &\to_\beta (\lambda y.g)\ (I\ b)\ h\\ &\to_\beta gh\end{aligned}$$

FIGURE 2.21 – Étapes de réduction du terme $(\lambda x.\lambda y.x)\ g\ (I\ b)\ h$

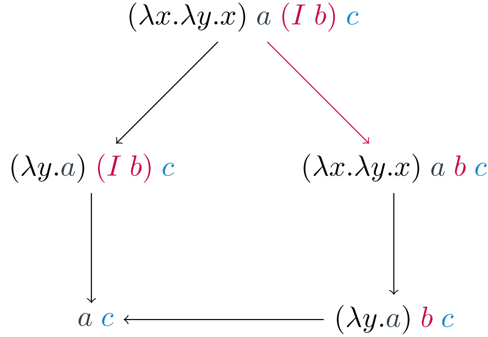

FIGURE 2.22 – Graphique de réduction du terme $(\lambda x.\lambda y.x)\ a\ (I\ b)\ c$.

$$\begin{aligned}
(\lambda x.xx)\ ((\lambda x.x)\ b) &\rightarrow_\beta ((\lambda x.x)\ b)\ ((\lambda x.x)\ b)\\
&\rightarrow_\beta b\ ((\lambda x.x)\ b)\\
&\rightarrow_\beta bb
\end{aligned}$$

FIGURE 2.23 – Réduction du terme $(\lambda x.xx)\ ((\lambda x.x)\ b)$ avec la stratégie leftmost-outermost qui duplique du calcul

Si l'on parle de stratégie d'évaluation, le but est de produire une valeur. Dans le cas d'une stratégie de réduction normalisante, le but est de produire une forme normale.

Dans les deux cas, le sous-ensemble le plus simple que nous pouvons trouver est le sous-ensemble utilisé par la stratégie leftmost-outermost en ne réduisant pas sous les abstractions pour les stratégies d'évaluation et en y réduisant pour les stratégies de réduction qui normalisent. La stratégie leftmost-outermost n'autorise que la réduction du radical le plus à gauche et le plus à l'extérieur du terme. Dans le terme $\lambda x.((\lambda y.y)\ x)\ ((\lambda w.w)\ z)$ le radical le plus à gauche est le terme en rouge.

Leftmost-outermost est une bonne base car, premièrement, elle cible obligatoirement un radical. Cela signifie que, tant qu'il y a des calculs à effectuer, le sous-ensemble nous permettra de réduire le terme. Et si la stratégie ne cible plus rien, c'est que le terme est en forme normale. De plus, le terme ciblé est toujours nécessaire. En effet, la stratégie leftmost-outermost cible un radical qui est le plus à gauche mais aussi le plus à l'extérieur, ce qui veut dire qu'il n'a pas de radical au-dessus de lui, c'est-à-dire pas d'abstraction appliquée. Cela nous garantit que le radical ne va pas être appliqué à un argument.

Mais nous voulons définir un sous-ensemble plus grand pour avoir plus de liberté dans l'ordre dans lequel nous voulons réduire les termes afin d'obtenir un calcul plus performant, car la stratégie leftmost-outermost peut entraîner la duplication du calcul. Dans la figure 2.23, on voit que le sous-terme b n'apparaît qu'une fois dans le terme de base, alors qu'il est présent deux fois dans le terme réduit.

Pour définir notre sous-ensemble, nous partons du constat suivant : si un terme

ne possède pas de radical au-dessus de lui, c'est-à-dire si aucune abstraction n'est appliquée, alors aucune variable à son niveau ne peut être remplacée et donc faire disparaître ses arguments.

Donc, nous avons besoin de savoir quelles variables ne pourront plus jamais être remplacées et quelles variables pourront encore l'être. Nous avons donc défini la notion de variable gelée qui nous dit si une variable ne sera plus remplacée. Et nous avons défini la notion de position isolée pour savoir quand un terme ne pourra plus disparaître.

2.3.2 Variable gelée / Position isolée

On définit la notion de variable gelée pour savoir quelle variable ne sera pas remplacée. Puis, on utilise cette notion de variable gelée pour trouver les positions isolées (qui elles-mêmes permettent d'identifier plus de variables gelées). Par exemple, un terme qui est à droite d'une variable gelée est dans une position isolée. On définit la notion de structure pour gérer cet aspect récursif (les structures seront bientôt introduites plus proprement).

Variable gelée L'objectif est de caractériser les variables qui ne seront jamais substituées. Premièrement, toutes les variables libres sont gelées car elles ne seront jamais substituées. Ensuite, pour les variables liées. La seule possibilité pour qu'une variable liée ne soit pas substituée est qu'elle soit liée à une abstraction qui n'est pas appliquée, et ne le deviendra jamais.

En revanche, si une variable est liée à une abstraction qui est appliquée, cette variable peut être substituée et donc disparaître. Par exemple, dans la réduction $(\lambda x.xy)\ a \to_\beta ay$, la variable x qui est liée à une abstraction qui est appliquée disparaît.

On gèle donc toutes les variables qui sont libres et toutes les variables liées à des abstractions qui ne sont pas, et ne seront jamais appliquées. Une fois qu'une variable est gelée, elle le reste jusqu'à la fin. Une fois qu'on sait qu'une variable ne va plus être substituée, on peut s'en servir pour caractériser les termes qui ne vont jamais disparaître.

Position isolée La seule manière pour qu'un terme disparaisse est qu'il soit l'argument d'une abstraction qui va le faire disparaître, par exemple $(\lambda x.y)z \to_\beta y$. De manière plus générale, on sait que si un terme n'est pas un argument d'une abstraction et ne le deviendra jamais, il ne va pas disparaître.

Donc, un terme à top-level est à une position isolée car aucune abstraction ne peut s'appliquer à lui. Les termes à droite d'une variable libre sont également à une position isolée car il n'y a pas de bêta-réduction possible.

Par exemple, dans le terme $x\ (I\ z)$, la variable x est une variable libre, donc elle ne va pas être substituée. Par conséquent, $(I\ z)$ est à une position isolée et peut être réduit sans souci. Mais pour les termes appliqués à une variable liée, on va utiliser la notion de variable gelée : Si un terme est à droite d'une variable gelée, alors le terme est dans une position isolée.

$$(\lambda x.x\ (I\ a)) \xrightarrow[\beta]{} (\lambda x.xa)$$

Figure 2.24 – Réduction du terme $\lambda x.x\ (I\ a)$ avec en bleu les variables gelées et en rouge la réduction du terme $I\ a$.

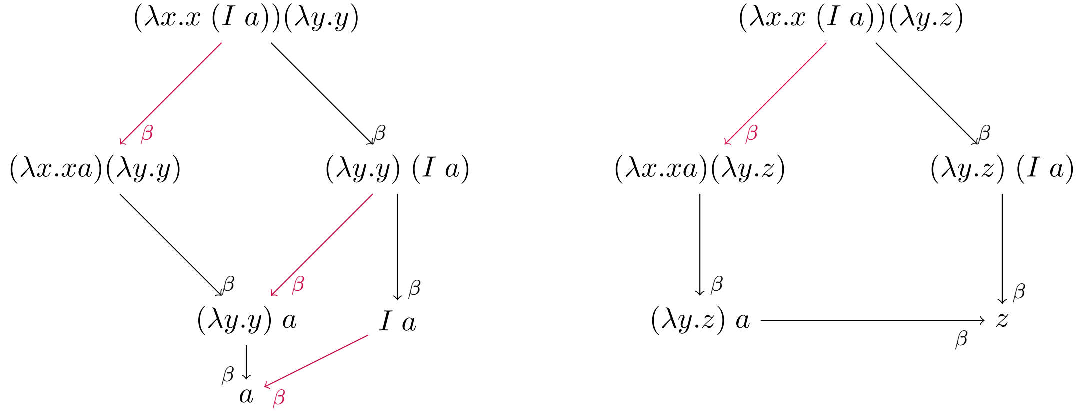

Figure 2.25 – Graphique de réduction du terme $\lambda x.x\ (I\ a)$ appliqué à un terme qui fait ou non disparaître son argument, avec en rouge la réduction du terme $I\ a$.

Dans la figure 2.24, l'abstraction n'est pas appliquée, donc la variable x est gelée (en bleu), et nous pouvons réduire le radical $(I\ a)$ à droite. Cependant, dans la figure 2.25, les deux termes sont appliqués, donc la variable x n'est pas gelée car nous ne savons pas quel terme pourrait la remplacer.

On voit qu'il y a deux cas ici, dans le graphique de gauche l'argument $(\lambda y.y)$ ne fait pas disparaître $I\ a$ et on voit que ce terme est nécessaire car il est dans tous les chemins de réduction. Mais le graphique de droite le fait disparaître et on voit qu'il n'est pas nécessaire car il existe un chemin de réduction qui ne le réduit pas.

On en déduit qu'il est possible de réduire à droite d'une variable gelée sans risquer de faire des réductions en trop. Toutefois, lorsque la variable n'est pas gelée, on ne sait pas à quoi s'en tenir, et on ne doit donc pas procéder à cette réduction.

De la même manière que pour les variables gelées, quand un terme est dans une position isolée, il le reste jusqu'à la fin. Quand une variable gelée a plusieurs arguments, ils sont tous dans une position isolée et on appelle cela une structure.

Définition 14 (Structure)**.** *On appelle «structure» une variable gelée et ses arguments. Les structures sont définies dans la figure 2.27.*

Par exemple, $a\ r_1\ r_2$ est une structure car a est une variable libre, donc gelée, qui a deux arguments r_1 et r_2.

Les seules réductions possibles dans une structure se trouvent dans les arguments,

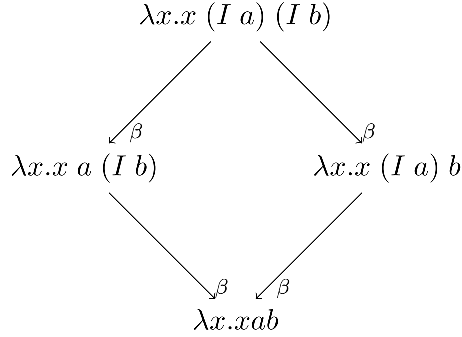

FIGURE 2.26 – Réduction de la structure $\lambda x.x\ (I\ a)\ (I\ b)$

$$\frac{x \in \varphi}{x \in S_\varphi} \qquad \frac{t \in S_\varphi}{t\ u \in S_\varphi}$$

FIGURE 2.27 – Structures avec φ l'ensemble des variables gelées

il n'y a pas de réduction au niveau de la structure, donc l'ordre dans lequel les arguments sont réduits n'a pas d'importance.

Dans la figure 2.26, on voit que le chemin emprunté ne modifie pas le nombre de réductions ni la forme normale, mais seulement l'ordre dans lequel les arguments sont réduits.

On hérite du fait qu'une variable gelée le restera jusqu'à la fin pour dire qu'une structure restera toujours une structure avec au moins le même nombre d'arguments et la même variable de tête. Les arguments peuvent évoluer, mais la forme de la structure ne changera pas. Un exemple de structure qui se déplace dans le terme, mais qui ne change pas de forme, est donné dans la figure 2.28.

2.4 Appel par nécessité en lambda-calcul pur

Dans cette section, nous allons commencer à définir notre propre calcul, en appliquant plus formellement les principes énoncés précédemment. Dans la section 2.4.1, nous allons parler de la position d'un terme et de la manière de le représenter. Dans la section 2.4.2, nous allons utiliser les contextes de la section 2.4.1 pour discuter des stratégies d'évaluation par nom et par valeur. Ensuite, dans la section 2.4.3, nous allons présenter une manière de caractériser les positions isolées et les variables gelées. Puis, dans la section 2.4.4, nous allons utiliser les contextes de la section 2.4.1 pour définir les positions nécessaires à notre calcul. Pour terminer, la section 2.4.5 présentera les problèmes rencontrés avec la définition de ce calcul.

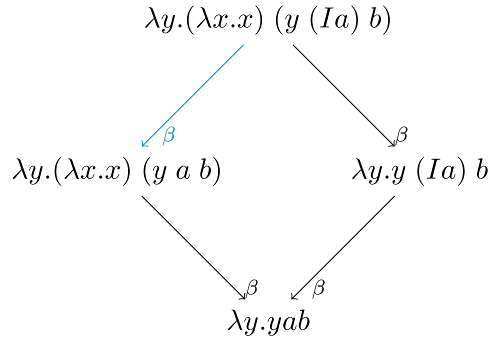

FIGURE 2.28 – Graphique de réduction du terme $\lambda y.(\lambda x.x)\ (y\ (Ia)\ b)$ pour illustrer le fait que la structure $y\ (Ia)\ b$ ne change pas de forme.

2.4.1 Position et contexte

Nous avons vu dans la section 2.3 qu'il était difficile de parler de position dans un terme. Cette section présente les contextes qui sont une manière plus formelle de parler de position dans un terme.

Contexte Pour parler des positions dans un terme, nous allons utiliser les contextes. Les contextes sont des termes qui contiennent un trou, noté $\bullet$, représentant la position que nous voulons exprimer. On utilise généralement la lettre $\mathcal{C}$ pour parler des contextes. Il est donc possible de parler d'un contexte $\mathcal{C}$ qui vaut un terme avec un trou, par exemple $(\lambda x.\bullet)\ y$. Un terme peut donc être exprimé par un contexte dans lequel on a mis un terme dans le trou. On note $\mathcal{C}(t)$ le contexte $\mathcal{C}$ dans lequel on a mis t dans le trou. Il est aussi possible d'imbriquer les contextes, par exemple $\mathcal{C}_1(\mathcal{C}_2)$ est le contexte $\mathcal{C}_1$ dans lequel on a mis le contexte $\mathcal{C}_2$. Les contextes sont pratiques pour parler d'une position dans un terme, mais ils sont aussi très pratiques pour exprimer le fait qu'un terme possède un sous-terme sans préciser l'endroit où il se trouve. Par exemple, pour parler d'un terme t qui possède une variable x, il est possible d'écrire $\exists\ \mathcal{C}$ tel que $t = \mathcal{C}(x)$. Cependant, il faut faire attention car la nature de la variable qui est dans le trou, ici x, n'est pas précisée. La variable peut très bien être liée et la position où se trouve le x peut être sous un certain nombre de lieurs.

Remplacement Il est possible de remplacer un terme qui se trouve dans un contexte. Notamment, si on a un terme t qui se réduit vers un terme u alors même si ce terme est entouré par un contexte, il est possible de le réduire : $t \to_\beta u \implies \mathcal{C}(t) \to_\beta \mathcal{C}(u)$.

On peut également utiliser cette notation pour écrire le remplacement d'une occurrence libre de x dans un terme $\mathcal{C}(x)$ par un terme t, ce qui donne $\mathcal{C}(t)$, mais nous n'avons aucune garantie sur le fait que les variables qui se trouvent dans t ne soient pas capturées lors du remplacement.

$$\frac{}{\bullet \in \mathcal{E}}\text{ BOX} \qquad \frac{\mathcal{C} \in \mathcal{E}}{\mathcal{C}\ t \in \mathcal{E}}\text{ APP-LEFT} \qquad \frac{\mathcal{C} \in \mathcal{E} \qquad t \in S}{t\ \mathcal{C} \in \mathcal{E}}\text{ APP-RIGHT}$$

FIGURE 2.29 – Ensemble $\mathcal{E}$ des contextes d'évaluation pour la stratégie d'évaluation en appel par nom

Contexte d'évaluation Pour définir les règles de notre calcul, nous avons besoin de parler des positions dans le terme. C'est pourquoi nous allons utiliser les contextes pour définir la notion de contexte d'évaluation [5, 14]. Les contextes d'évaluation représentent les positions où les réductions sont autorisées par la stratégie d'évaluation.

2.4.2 Stratégie d'évaluation

Dans les deux stratégies présentées ci-dessous, on utilise des structures pour gérer les variables libres, car nous sommes dans le cadre des stratégies faibles itérées qui relancent l'évaluation sur des termes ouverts. Cependant, dans tous les cas, il est toujours possible de réduire à gauche une application avec la règle APP-LEFT figure 2.31 et figure 2.29.

Appel par nom La stratégie d'évaluation en appel par nom peut toujours aller réduire à gauche d'une application car elle va aller chercher une valeur, c'est-à-dire soit une abstraction pour effectuer la bêta-réduction, soit une variable. La stratégie n'évalue pas le membre droit si le membre gauche est une abstraction, mais elle évalue le membre droit si le membre gauche est une variable. Ainsi, elle permet d'évaluer les arguments d'une structure. La figure 2.29 présente l'ensemble des contextes d'évaluation pour la stratégie d'évaluation en appel par nom.

Appel par valeur La stratégie d'évaluation en appel par valeur peut quant à elle réduire où elle veut : à gauche d'une application comme l'appel par nom, mais elle peut également réduire les membres droits car l'appel par valeur réduit les arguments avant d'appliquer la bêta-réduction. Pour gérer le fait que la bêta-réduction ne s'applique que lorsque les arguments sont évalués, il est nécessaire d'utiliser une nouvelle définition. Pour définir la nouvelle bêta-réduction pour l'appel par valeur, nous devons également définir une nouvelle notion de valeur, car si l'on se réfère à la définition usuelle des valeurs qui n'inclut pas les structures, alors il ne sera pas possible de réduire les radicaux qui ont pour argument une structure.

Définition 15 (Valeur ouverte)**.** *On définit donc la notion de valeur ouverte (*VO*) pour définir la bêta-réduction pour la stratégie d'appel par valeur et la notion de structure évaluée (*SV*) à la figure 2.30.*

Les valeurs ouvertes n'ont pas de radicaux, sauf sous les abstractions. La figure 2.31 présente l'ensemble des contextes d'évaluation pour la stratégie d'évaluation

$$\frac{}{\lambda x.t \in \text{VO}} \qquad \frac{t \in \text{SV}}{t \in \text{VO}} \qquad \frac{}{x \in \text{SV}} \qquad \frac{t \in \text{SV} \qquad u \in \text{VO}}{t\ u \in \text{SV}}$$

$$(\lambda x.t)\ u \rightarrow_{\beta v} t\{x \setminus u\} \text{ avec } u \in \text{VO}$$

FIGURE 2.30 – Valeur ouverte et bêta-réduction pour la stratégie d'évaluation en appel par valeur

$$\text{BOX}\ \frac{}{\bullet \in \mathcal{E}} \qquad \text{APP-LEFT}\ \frac{\mathcal{C} \in \mathcal{E}}{\mathcal{C}\ t \in \mathcal{E}} \qquad \text{APP-RIGHT}\ \frac{\mathcal{C} \in \mathcal{E}}{t\ \mathcal{C} \in \mathcal{E}}$$

FIGURE 2.31 – Ensemble $\mathcal{E}$ des contextes d'évaluation pour la stratégie d'évaluation en appel par valeur

par appel par valeur.

2.4.3 Approximation des positions isolées et des variables gelées

Comme nous l'avons mentionné précédemment, il n'est pas possible de trouver toutes les positions isolées et toutes les variables gelées d'un terme. Dans cette section, nous présentons une approximation des positions isolées et des variables gelées. Nous commençons par distinguer deux types de position : les positions isolées et les positions incertaines en utilisant ce que nous avons appelé le mode. Cependant, notre approximation est complète pour les termes en forme normale.

Mode & Variables gelées

Définition 16 (Mode)**.** *Le mode est une approximation des positions isolées présenté dans la figure 2.32*

Il se propage grâce à des règles, mais ne s'applique qu'à la position donnée dans le terme. On a le mode top ($\top$) qui représente les positions isolées et le mode bot ($\bot$) qui représente les positions dont on ne sait pas encore si elles vont se retrouver dans une position isolée ou non, appelées également des positions incertaines.

On utilise un ensemble noté φ pour stocker les variables gelées.

On part du top-level qui est toujours en mode *top* car jamais appliqué et de l'ensemble de toutes les variables libres du terme pour les variables gelées. Puis, on parcourt le terme en suivant les règles définies dans la figure 2.32 pour aller jusqu'à un trou pour définir son mode.

Il y a deux règles pour l'abstraction : ABS-$\bot$, qui dit que si l'abstraction est en mode $\bot$, alors son corps l'est aussi, et ABS-$\top$, qui propage également le mode

$$\text{BOX}\ \frac{}{\bullet \vdash \tau, \varphi} \qquad \text{ABS-}\top\ \frac{\mathcal{C} \vdash \top, \varphi}{\mathcal{C}(\lambda x.\bullet) \vdash \top, \varphi \cup \{x\}} \qquad \text{ABS-}\bot\ \frac{\mathcal{C} \vdash \bot, \varphi}{\mathcal{C}(\lambda x.\bullet) \vdash \bot, \varphi}$$

$$\text{APP-LEFT}\ \frac{\mathcal{C} \vdash \tau, \varphi}{\mathcal{C}(\bullet\ t) \vdash \bot, \varphi} \qquad \text{APP-RIGHT}\ \frac{\mathcal{C} \vdash \tau, \varphi}{\mathcal{C}(t\ \bullet) \vdash \bot, \varphi} \qquad \text{APP-RIGHT-S}\ \frac{\mathcal{C} \vdash \tau, \varphi \quad t \in S_\varphi}{\mathcal{C}(t\ \bullet) \vdash \top, \varphi}$$

FIGURE 2.32 – Définition des modes qui approximent les positions isolées

$$\text{ABS-}\top\ \dfrac{\text{APP-RIGHT-S}\ \dfrac{\text{ABS-}\top\ \dfrac{\text{BOX}\ \dfrac{}{\bullet \vdash \top, \emptyset}}{\lambda y.\bullet \vdash \top, \{y\}} \qquad y \in S_{\{y\}}}{\lambda y.(y\ \bullet) \vdash \top, \{y\}}}{\lambda y.(y\ \lambda x.\bullet) \vdash \top, \{y, x\}}$$

FIGURE 2.33 – Exemple de calcul du mode du contexte $\lambda y.(y\ \lambda x.\bullet)$

mais qui gèle la variable liée en même temps en l'ajoutant dans l'ensemble φ. Pour les applications, il y a trois règles : APP-LEFT, qui dit que la partie gauche d'une application est en mode $\bot$, car elle est appliquée ; APP-RIGHT, qui est le cas général quand on va à droite et qui est aussi en mode $\bot$, car on n'a pas d'information sur la nature de t qui peut très bien être une abstraction et qui pourrait placer notre trou dans une position appliquée ; et la règle APP-RIGHT-S, qui est la même que APP-RIGHT mais avec l'information supplémentaire que t est une structure, donc que notre position est un argument de cette structure, donc en position isolée, donc en mode $\top$.

Dans la figure 2.33, on voit que le contexte cible une position en mode $\top$ et qu'à cette position les variables x et y sont gelées, ce qui est bien le comportement voulu car la position représentée par le contexte est isolée. Dans la figure 2.34, on voit que la position ciblée est en mode $\bot$, ce qui est bien car elle va être appliquée et même disparaître. Cependant, notre approximation a ses limites car, dans la figure 2.35, on aurait voulu que la position ciblée soit considérée comme $\top$, mais les règles ne permettent pas de l'affirmer.

Monotonie Dans cette section, nous discutons de la monotonie par rapport à la réduction, c'est-à-dire que, lorsqu'une propriété est vraie pour un terme, elle reste vraie après une réduction. Les fonctions qui calculent les ensembles des variables gelées ou des positions isolées sont monotones par rapport aux réductions effectuées. Nous déduisons de cela que nos ensembles qui approximent les variables gelées et les positions isolées ne font que croître.

$$\text{APP-RIGHT}\ \dfrac{\text{BOX}\ \dfrac{}{\bullet \vdash \top, \emptyset}}{(\lambda x.y)\ \bullet \vdash \bot, \emptyset}$$

FIGURE 2.34 – Exemple de calcul du mode du contexte $(\lambda x.y)\ \bullet$

$$\text{APP-RIGHT}\ \dfrac{\text{ABS-}\bot\ \dfrac{\text{APP-RIGHT}\ \dfrac{\text{BOX}\ \dfrac{}{\bullet \vdash \top, \emptyset}}{(\lambda x.x)\ \bullet \vdash \bot, \emptyset}}{(\lambda x.x)\ (\lambda y.\bullet) \vdash \bot, \emptyset}}{(\lambda x.x)\ (\lambda y.y\ \bullet) \vdash \bot, \emptyset}$$

FIGURE 2.35 – Exemple de calcul du mode du contexte $(\lambda x.x)\ (\lambda y.y\ \bullet)$

L'utilité des variables gelées, des positions isolées et des structures est directement liée à leurs propriétés de monotonie. En effet, on utilise ces notions pour garantir que le terme ne disparaîtra pas. Une variable gelée le restera jusqu'à la fin, de même qu'une position isolée ou une structure. Par contre, une variable qui n'est pas gelée peut le devenir. Par exemple, dans la figure 2.36, la variable x n'était pas gelée au début, mais elle le devient. Une fois gelée, elle le reste jusqu'à la fin. La structure dont la variable de tête est x évolue de la même manière : elle n'était pas une structure au début, puis elle le devient lorsque x devient gelée et elle le reste jusqu'à la fin.

Approximation et forme normale Notre approximation des variables gelées et des positions isolées ne couvre qu'un sous-ensemble des variables gelées et des positions isolées.

Avec la propriété de complétude des formes normales par rapport au lambda-calcul, on sait qu'un terme en forme normale dans notre calcul n'a plus de radical. La seule manière d'avoir une variable qui n'est pas gelée est d'avoir un radical. Car qui dit variable non gelée dit abstraction appliquée et donc radical.

Donc, quand un terme est en forme normale, notre approximation devient complète, c'est-à-dire qu'elle trouve toutes les variables gelées et toutes les positions isolées. Notre approximation avait pour objectif de trouver suffisamment de positions isolées pour avancer dans le terme, donc le fait qu'elle trouve toutes ces positions une fois le terme en forme normale incite à penser que cela fonctionne correctement.

2.4.4 Contexte d'évaluation pour l'appel par nécessité

Une fois que toutes les notions que nous souhaitons employer pour définir notre calcul ont été définies de manière plus formelle, nous pouvons commencer à proposer

$$
\begin{aligned}
(\lambda y.y)(\lambda x.xab) &\to_\beta \lambda x.xab \\
&\to_\beta \lambda x.xa'b \\
&\to_\beta \lambda x.xa'b'
\end{aligned}
$$

FIGURE 2.36 – Évolution de la structure qui a pour variable de tête x dans le terme $(\lambda y.y)(\lambda x.xab)$ avec a qui se réduit en a' et b en b'

une version de notre contexte d'évaluation pour l'appel par nécessité dans le lambda-calcul pur.

Il convient de noter que les règles proposées pour le calcul ne sont pas très intuitives. C'est pourquoi, dans le chapitre 5, nous les formaliserons à l'aide d'un assistant de preuve.

Contexte d'évaluation Notre contexte d'évaluation est paramétré par φ, l'ensemble des variables gelées initialisé avec les variables libres du terme, et τ, le mode initialisé à $\top$. Quand un contexte est un contexte d'évaluation, on le note $\mathcal{C} \in \mathcal{E}_{\varphi,\tau}$.

Les contextes d'évaluation sont définis par plusieurs règles figure 2.37. Premièrement, il y a deux règles pour traiter les abstractions. Sous les abstractions en mode $\bot$, on ne gèle pas les variables et on propage le mode $\bot$ (règle ABS-$\bot$). Sous les abstractions en mode $\top$, on gèle la variable liée et on propage le mode $\top$ (règle ABS-$\top$).

Ensuite, il y a aussi deux règles pour les applications. Premièrement, quelque soit le mode dans lequel se trouve l'application, il est toujours possible d'aller à gauche en mode $\bot$, car on est dans une position appliquée (règle APP-LEFT). Deuxièmement, il est possible d'aller à droite d'une application, et on le fait alors en mode $\top$ uniquement quand le membre gauche est une structure (règle APP-RIGHT-S).

Puis, la dernière règle est liée à l'aspect nécessaire de la stratégie. On veut pouvoir réduire les arguments d'abstraction en avance, pour ne pas avoir à refaire la réduction plusieurs fois. Mais cela n'est possible que si c'est nécessaire. Pour cela, on va utiliser les contextes d'évaluation pour savoir si au moins une occurrence de la variable liée à l'abstraction du radical est nécessaire (règle ARG). Le terme $\mathcal{C}'(x)$ est composé du contexte d'évaluation $\mathcal{C}'$ dans lequel on a mis la variable x. Cela ne vaut que si x n'est pas capturée par $\mathcal{C}$. La conclusion de la règle ARG est effectivement un contexte car elle ne contient qu'un seul trou.

On constate que les règles correspondent aux modes (Fig. 2.32) auxquels on a enlevé la règle APP-RIGHT et ajouté la règle ARG. Comme expliqué précédemment, nous avons ajouté la règle ARG pour réduire, à l'avance, les arguments d'une abstraction afin de réduire le nombre de calculs dupliqués. De plus, nous avons enlevé la règle APP-RIGHT car elle nous donnait un accès sans condition au membre droit d'une application, alors que cette position peut potentiellement être supprimée par l'abstraction.

Conjecture de complétude On fait ici la conjecture que notre calcul est complet, c'est-à-dire qu'il trouve bien la forme normale d'un terme si elle existe.

$$\frac{}{\bullet \in \mathcal{E}_{\varphi,\tau}}\ \text{BOX} \qquad \frac{\mathcal{C} \in \mathcal{E}_{\varphi\cup\{x\},\top}}{(\lambda x.\mathcal{C}) \in \mathcal{E}_{\varphi,\top}}\ \text{ABS-}\top \qquad \frac{\mathcal{C} \in \mathcal{E}_{\varphi,\bot}}{(\lambda x.\mathcal{C}) \in \mathcal{E}_{\varphi,\bot}}\ \text{ABS-}\bot$$

$$\frac{\mathcal{C} \in \mathcal{E}_{\varphi,\bot}}{\mathcal{C}\ t \in \mathcal{E}_{\varphi,\tau}}\ \text{APP-LEFT} \qquad \frac{\mathcal{C} \in \mathcal{E}_{\varphi,\top} \quad t \in S_\varphi}{t\ \mathcal{C} \in \mathcal{E}_{\varphi,\tau}}\ \text{APP-RIGHT-S} \qquad \frac{\mathcal{C}' \in \mathcal{E}_{\varphi,\tau} \quad \mathcal{C} \in \mathcal{E}_{\varphi,\bot}}{(\lambda x.\mathcal{C}'(x))\ \mathcal{C} \in \mathcal{E}_{\varphi,\tau}}\ \text{ARG}$$

FIGURE 2.37 – Contexte d'évaluation paramétrique

$$\text{APP-LEFT}\ \cfrac{\text{ABS-}\top\ \cfrac{\text{BOX}\ \cfrac{}{\bullet \in \mathcal{E}_{\emptyset,\top}}}{\lambda x.\bullet \in \mathcal{E}_{\{x\},\top}}}{\lambda x.\bullet\ \ (I\ y) \in \mathcal{E}_{\{x\},\bot}}$$

FIGURE 2.38 – Vérification du fait que le contexte $\lambda x.\bullet\ \ (I\ y))$ est bien un contexte d'évaluation

L'ensemble de nos contextes d'évaluation est un sous-ensemble suffisamment grand des positions nécessaires pour permettre une normalisation, car il contient toujours le radical ciblé par la stratégie leftmost-outermost, qui est une stratégie complète. De plus, il n'y a pas de restriction sur les réductions, à partir du moment où elles sont effectuées dans un contexte d'évaluation. Par exemple, dans le terme $\lambda x.((I(I\ z))\ (I\ y))$ avec I qui représente la fonction identité, la stratégie leftmost-outermost va construire l'ensemble des radicaux les plus externes. Dans notre cas, l'ensemble contient deux radicaux, $I(I\ z)$ et $I\ y$, et elle va prendre le plus à gauche, c'est-à-dire le radical $I(I\ z)$. Ce radical est bien à une position qui se trouve dans l'ensemble de nos contextes d'évaluation, $\lambda x.\bullet\ \ (I\ y) \in \mathcal{E}_{\{x\},\bot}$, comme illustré dans la figure 2.38.

Il est assez facile de se convaincre que nos contextes ciblent bien le radical leftmost, mais la partie la plus délicate vient de la partie outermost. L'aspect leftmost est géré par les deux règles APP-LEFT et APP-RIGHT-S qui nous permettent d'aller le plus à gauche possible et, si le terme de gauche est inerte (structure), d'aller à droite de l'application. L'aspect outermost est géré par la règle ABS-$\top$ et le fait que le top-level est toujours un contexte d'évaluation par la règle BOX. La règle ABS-$\top$ nous permet d'aller sous l'abstraction dans le cas où elle est en mode $\top$, c'est-à-dire qu'elle n'est pas appliquée donc qu'il n'y a pas de radical. De manière plus générale, nous faisons la conjecture que le radical ciblé par leftmost-outermost est dans nos contextes d'évaluation.

On voit que les quatre règles de gauche de notre calcul (BOX, APP-LEFT, APP-RIGHT-S et ABS-$\top$) suffisent à avoir une stratégie complète et que les deux règles de droite (ABS-$\bot$ et ARG) nous permettent de réduire plus de termes pour réduire le nombre de calculs dupliqués.

2.4.5 Substitution restreinte et problème des règles de commutation

Actuellement, nous permettons la bêta-réduction sur tous les radicaux. Cependant, dans la stratégie d'appel par nécessité, on ne souhaite pas substituer toutes les formes de terme et nous voulons donc imposer une restriction sur la bêta-réduction. Mais, dans le lambda-calcul pur, si l'on interdit certaines réductions, il peut y avoir des blocages liés à l'empilement de radicaux qui ne peuvent pas être réduits. Pour traiter ce problème, nous pourrions écrire des règles de commutation pour avoir accès aux termes bloqués. Cependant, notre calcul deviendrait moins lisible et il existe déjà une solution pour gérer ce problème, à savoir les substitutions explicites.

Règles de réduction Pour limiter les calculs dupliqués, nous ne voulons substituer que les termes déjà évalués, c'est-à-dire des valeurs, comme c'est fait en appel par valeur et en appel par nécessité.

$$(\lambda x.t)\ u \to_{\beta} t\{x \setminus u\} \text{ avec } u \in \mathit{Val}$$

Blocage Le problème dans notre calcul avec les contextes d'évaluation avec cette restriction est qu'on pourrait être confronté à des blocages. Dans le cas où un radical ne peut pas être réduit car son argument n'est pas une valeur, mais qu'il y a un radical qui pourrait être fait une fois le premier réduit.

Dans l'exemple qui suit, la variable a n'est pas une valeur et donc ne va pas pouvoir être réduite. Le problème est que cela bloque le calcul : $((\lambda x.\lambda y.yx)a)(\lambda z.z)$.

Dans certains cas, il est même possible d'être bloqué sur des termes plus simples. Par exemple, le terme $(\lambda x.x)a$ n'est pas en forme normale, mais il est bloqué car a n'est pas une valeur.

Une solution serait de faire remonter les valeurs, comme dans l'approche de Ariola et al. [5]. Cependant, avec ces règles en plus, notre calcul deviendrait moins lisible en introduisant des réductions qui ne sont pas pertinentes pour le calcul. Heureusement, il existe déjà une solution bien connue pour gérer ce problème, à savoir les substitutions explicites.

2.5 Substitution explicite

Dans cette section, nous présentons les substitutions explicites. Nous définitions la syntaxe qui est issue de de l'approche de Accatoli et Kesner [5] dans la section 2.5.1. Quant à la règle qui gère les substitutions explicites, plutôt que d'utiliser les règles de propagation traditionnelles [24], elle s'appuie sur le principe du Linear Substitution Calculus [28, 4, 2], qui est similaire aux constructions let-in couramment utilisées pour définir l'appel par nécessité. Enfin, nous définissons notre calcul avec ce nouvel outil dans la section 2.5.2.

$$
\begin{array}{rcll}
(\lambda x.t)\mathcal{L}\ u & \to_{dB} & t[x \setminus u]\mathcal{L} & \\
\mathcal{C}(x)[x \setminus v\mathcal{L}] & \to_{lsv} & \mathcal{C}(v)[x \setminus v]\mathcal{L} & \text{avec } v \text{ une valeur et } \mathcal{C} \in \mathcal{E}_{\varphi,\tau} \\
t[x \setminus y] & \to_{gc} & t & \text{si } x \notin \mathrm{FV}(t)
\end{array}
$$

FIGURE 2.39 – Règles de réduction pour le lambda-calcul avec substitution explicite

$$
\begin{array}{rcl}
\mathrm{FV}(x) & = & \{x\} \\
\mathrm{FV}(\lambda x.t) & = & \mathrm{FV}(t) \setminus \{x\} \\
\mathrm{FV}(t\ u) & = & \mathrm{FV}(t) \cup \mathrm{FV}(u) \\
\mathrm{FV}(t[x \setminus u]) & = & (\mathrm{FV}(t) \setminus \{x\}) \cup \mathrm{FV}(u)
\end{array}
$$

FIGURE 2.40 – Variables libres des termes avec substitution explicite

2.5.1 Définition

Le lambda-calcul avec substitutions explicites est une extension du lambda-calcul dans laquelle une nouvelle construction, appelé substitution explicite, est ajoutée et où la bêta-réduction est divisée en trois règles : dB, qui transforme un radical en une substitution explicite ; lsv, qui substitue une occurrence de la variable de la substitution explicite ; et gc, qui supprime les substitutions explicites qui ne sont plus utiles. Cela nous permet de gérer les réductions à distance plus facilement qu'en introduisant des règles de commutation et de gérer également le partage.

L'avantage de cette solution par rapport au fait d'ajouter des règles de commutation est qu'elle ne comporte pas de réductions purement techniques dans les séquences de réduction, ce qui permet de conserver une certaine lisibilité et de faciliter le lien avec le lambda-calcul pur.

Nous présentons également la nouvelle définition des variables libres et des substitutions implicites pour les termes avec substitution explicite dans les figures 2.40 et 2.41.

Termes Les termes du lambda-calcul sont les mêmes qu'auparavant, à savoir les variables, les applications et les abstractions, auxquelles est ajoutée la substitution explicite. La substitution implicite $t\{x \setminus u\}$ est une fonction qui remplace toutes les occurrences de x par u dans t. La substitution explicite, notée $t[x \setminus u]$, est un élément syntaxique qui représente le fait que dans t, x vaut u. La syntaxe du lambda-calcul s'écrit alors :

$$\Lambda := x \mid t\ u \mid \lambda x.t \mid t[x \setminus u]$$

La bêta-réduction va désormais transformer le radical en une substitution explicite, plutôt qu'une substitution implicite. Cela va nous permettre de gérer les substitutions plus finement, c'est-à-dire que, une fois qu'un radical a été transformé en une substitution explicite, on peut restreindre la règle qui applique la substitution explicite. Cela nous permet également de gérer le partage. Ceci nous permet, d'une

$$
\begin{array}{rrcl}
Var & z\{x \setminus t\} & = & z \text{ si } x \neq z \\
Subst & x\{x \setminus t\} & = & t \\
App & (t\ u)\{x \setminus w\} & = & t\{x \setminus w\}\ u\{x \setminus w\} \\
Abs & (\lambda x.t)\{y \setminus u\} & = & \lambda x.(t\{y \setminus u\}) \text{ si } x \notin \text{FV}(u) \text{ et } x \neq y \\
ES & t[x \setminus v]\{y \setminus u\} & = & t\{y \setminus u\}[x \setminus v\{y \setminus u\}] \text{ si } x \notin \text{FV}(u) \text{ et } x \neq y
\end{array}
$$

FIGURE 2.41 – Règles de substitution des termes avec substitution explicite

$$
\begin{array}{rcl}
((\lambda x.\lambda y.yx)a)(\lambda z.z) & \rightarrow_{dB} & (\lambda y.yx)[x \setminus a](\lambda z.z) \\
& \rightarrow_{dB} & (y\ x)[y \setminus \lambda z.z][x \setminus a]
\end{array}
$$

FIGURE 2.42 – Application de la règle dB sur le terme $(\lambda x.\lambda y.y\ x)\ a\ (\lambda z.z)$

part, de ne réduire le terme qu'une fois pour toutes les occurrences d'une variable et, d'autre part, de ne réduire le terme que s'il est associé à une variable qui est dans une position nécessaire.

Les substitutions explicites vont également pouvoir effectuer des réductions à distance afin de résoudre le problème présenté à la fin de la section précédente. Nous utilisons la lettre $\mathcal{L}$ pour représenter une séquence de substitution explicite $[x_1 \setminus u_1] \cdots [x_n \setminus u_n]$. Nous notons $(\lambda x.x)\mathcal{L}\ a$ pour dire qu'il y a potentiellement des substitutions explicites entre l'abstraction et le terme a.

Règle dB La règle dB (distance bêta) de Ariola et al. [5], est l'une des deux règles du lambda-calcul avec substitution explicite. Elle ressemble à la bêta-réduction classique, mais elle ne substitue pas directement : elle transforme le radical en une substitution explicite. Cette règle s'applique «à distance», c'est-à-dire qu'il est possible qu'il y ait des substitutions explicites entre l'abstraction et l'argument du radical, et la règle dB peut quand même s'appliquer. Elle est représentée par la première ligne de la figure 2.39.

Le fait de pouvoir faire cette bêta à distance nous débloque de la situation de la section précédente. Par exemple, le terme $(\lambda x.\lambda y.y\ x)\ a\ (\lambda z.z)$ qui était bloqué car a n'est pas une valeur peut maintenant se réduire comme présenté dans la figure 2.42.

La substitution explicite créée par la règle dB est mise au début des substitutions explicites, qui séparaient l'abstraction et son argument. Cela ne pose pas de problème car, grâce à la convention de Barendregt, les substitutions explicites se trouvant au-dessus de l'argument ne peuvent pas faire référence à des éléments de ce dernier et cela préserve l'ordre des lieurs.

Règle lsv La règle lsv (substitution linéaire de valeur) de Ariola et al. [5], est la deuxième partie de la bêta-réduction, qui permet de substituer une occurrence d'une variable par l'argument de la substitution explicite. En sortant les substitutions explicites du corps de la substitution explicite, l'occurrence de la variable qui est

$$\begin{array}{lll} (y\ x)[y \setminus \lambda z.z][x \setminus a] & \rightarrow_{lsv} & ((\lambda z.z)\ x)[y \setminus \lambda z.z][x \setminus a] \\ & \rightarrow_{dB} & z[z \setminus x][y \setminus \lambda z.z][x \setminus a] \end{array}$$

FIGURE 2.43 – Application des règles lsv et dB sur le terme $(y\ x)[y \setminus \lambda z.z][x \setminus a]$

$$\begin{array}{lll} (x\ x)[x \setminus (\lambda a.a\ y)[y \setminus \lambda z.z]] & \rightarrow_{lsv} & ((\lambda a.a\ y)x)[x \setminus \lambda a.a\ y][y \setminus \lambda z.z] \\ & \rightarrow_{dB} & a\ y[a \setminus x][x \setminus \lambda a.a\ y][y \setminus \lambda z.z] \end{array}$$

FIGURE 2.44 – Application de la règle lsv sur le terme $(x\ x)[x \setminus (\lambda a.a\ y)[y \setminus \lambda z.z]]$ pour illustrer le fait qu'il est nécessaire de sortir le $\mathcal{L}$

substituée doit se trouver dans un contexte d'évaluation paramétré par les bonnes valeurs de φ et de τ. Le terme sur lequel s'applique la règle lsv est dans un contexte qui a ses propres paramètres φ et τ. Ce sont ces mêmes paramètres que l'on prend pour le contexte d'évaluation de la variable dans la règle lsv. La règle lsv est représentée par la deuxième ligne dans la figure 2.39.

Pour continuer l'exemple du paragraphique précédent, une fois que l'on a transformé les deux radicaux en substitution explicite, nous allons pouvoir appliquer la règle lsv dans la figure 2.43, qui s'applique aussi à distance.

Un autre mécanisme de la règle lsv est de sortir le $\mathcal{L}$ du corps de la substitution explicite, ce qui est indispensable car, quand on substitue une variable par un terme dans la partie gauche de la substitution explicite, il faut que la liste de substitution explicite qui y était accrochée soit accessible à la fois par la variable qui vient d'être substituée, mais aussi par le corps de la substitution explicite qui n'a pas bougé. Il ne serait pas possible de le mettre aux deux endroits car, dans ce cas, nous casserions le partage. Notons que la variable substituée ne doit pas être capturée par $\mathcal{C}(\bullet)$ (la convention de Barendregt le dit implicitement)

La figure 2.44 présente un exemple avec le terme $(x\ x)[x \setminus (\lambda a.a\ y)[y \setminus \lambda z.z]]$. On observe que, une fois la réduction lsv effectuée, la variable y se trouve à deux niveaux de profondeur différents ; par conséquent, la substitution explicite doit être appliquée afin de se placer au-dessus des deux occurrences de y.

Règle gc Pour finir, il y a la dernière règle gc (garbage collector) qui permet de supprimer les substitutions explicites lorsque leurs variables ne sont plus présentes dans le terme. La règle dB introduit une substitution explicite, la règle lsv s'en sert et la règle gc la supprime. La règle gc est représentée par la troisième ligne dans la figure 2.39.

On voit que la règle s'applique quand la variable de la substitution explicite ne fait plus partie des variables libres du terme. La variable ne peut pas faire partie des variables liées car, avec la convention de Barendregt, la variable ne peut pas être liée à autre chose qu'à la substitution explicite en question.

$$z[z \setminus x][y \setminus \lambda z.z][x \setminus a] \quad \rightarrow_{gc} \quad z[z \setminus x][x \setminus a]$$

FIGURE 2.45 – Application de la règle *gc* sur le terme $z[z \setminus x][y \setminus \lambda z.z][x \setminus a]$

$$\begin{array}{rcl} x^* & = & x \\ (t\ u)^* & = & t^*\ u^* \\ (\lambda x.t)^* & = & \lambda x.t^* \\ (t[x \setminus u])^* & = & t^*\{x \setminus u^*\} \end{array}$$

FIGURE 2.46 – Fonction de dépliage

Pour finir l'exemple des deux paragraphes précédents, une fois que l'on a transformé les deux radicaux en substitution explicite, puis que nous avons appliqué la règle *lsv* pour substituer toutes les variables, il ne reste plus qu'à supprimer les substitutions explicites qui ne servent plus avec la règle *gc* dans la figure 2.45.

Définition 17 (Dépliage)**.** *Pour pouvoir faire le lien avec le lambda-calcul pur, nous utilisons une fonction de dépliage qui transforme les termes avec substitutions explicites en termes purs. La fonction est définie dans la figure 2.46.*

Dans le cas des variables, des applications, et de l'abstraction, la fonction de dépliage se contente de se propager dans les sous-termes. La fonction va transformer les substitutions explicites du terme en substitutions implicites, c'est-à-dire appliquer toutes les substitutions.

En revanche, en supprimant les substitutions explicites, la fonction de dépliage supprime le partage qu'elles avaient créé. La fonction de dépliage peut faire grossir le terme exponentiellement, car en supprimant le partage, on duplique des termes qui peuvent eux-mêmes contenir d'autres termes dupliqués. La figure 2.47 donne un exemple.

La fonction de dépliage englobe la règle de *gc* mais va plus loin. En effet, elle va appliquer toutes les substitutions explicites, y compris celles qui font référence à des variables dans le terme. Ainsi, elle va non seulement éliminer les substitutions explicites qui font référence à des variables qui ne sont pas présentes dans le terme, comme le ferait le *gc*, mais elle va aussi appliquer les substitutions explicites utiles qui vont substituer des variables.

2.5.2 Calcul avec substitution explicite

Dans cette section, nous adaptons notre calcul au lambda-calcul avec substitutions explicites. Cela permet de résoudre les problèmes de termes bloqués et de gérer plus facilement le partage.

$$
\begin{aligned}
(xx[x \setminus yy][y \setminus zz])^* &= (xx)\{x \setminus yy\}\{y \setminus zz\} \\
&= (yy)(yy)\{y \setminus zz\} \\
&= ((zz)(zz))((zz)(zz))
\end{aligned}
$$

FIGURE 2.47 – Exemple de dépliage qui fait grossir exponentiellement le terme

$$
\text{S-VAR}\ \frac{x \in \varphi}{x \in S_\varphi} \qquad \text{S-APP}\ \frac{t \in S_\varphi}{t\ u \in S_\varphi} \qquad \text{S-ES}\ \frac{t \in S_\varphi}{t[x \setminus u] \in S_\varphi} \qquad \text{S-ES-S}\ \frac{t \in S_{\varphi \cup \{x\}} \qquad u \in S_\varphi}{t[x \setminus u] \in S_\varphi}
$$

FIGURE 2.48 – Définition des structures avec les substitutions explicites

Il est important de noter que toutes les règles que nous avons vues s'appliquent dans un contexte d'évaluation, ce qui signifie que les radicaux sur lesquels s'appliquent les réductions doivent être dans ce contexte.

Structure Les structures pour le lambda-calcul avec substitution explicite ont deux règles de plus par rapport aux règles du lambda-calcul pur. La première règle, dite substitution explicite (S-ES), figure 2.48, traite le cas où un terme est une structure avec une substitution explicite qui ne parle pas de sa variable de tête. Grâce à la convention de Barendregt, si on ne précise pas que x est dans l'ensemble φ, alors il n'y est pas. Il y a une deuxième règle, dite S-ES-S (Fig. 2.48), qui traite le cas où la variable de tête est la variable de la substitution explicite, auquel cas le terme est une structure si le terme qui est dans la substitution explicite est une structure.

Contexte d'évaluation La règle ARG des contextes d'évaluation définie dans la figure 2.37 a été remplacée par la règle ES-RIGHT dans la figure 2.49.

On découpe en deux les règles ES. Il y a d'une part les règles ES-LEFT et ES-LEFT-FROZEN qui décrivent comment aller à gauche d'une substitution explicite et d'autre part ES-RIGHT pour aller à droite.

Dans un premier temps, la règle ES-LEFT indique que la partie gauche de la substitution explicite est toujours un contexte d'évaluation en propageant le mode. Ensuite, la règle ES-LEFT-FROZEN effectue la même opération, mais si le terme de la substitution explicite est une structure, alors la variable de la substitution explicite est gelée. Cela vient du fait que si le terme de la substitution explicite est une structure, on sait que la substitution ne créera pas de radical.

Puis, dans la dernière règle ES-RIGHT, si la variable de la substitution explicite apparaît dans un contexte d'évaluation, il est possible d'aller évaluer dans le corps de la substitution explicite en mode $\perp$. Ce mode provient du fait que l'on ne connaît pas la position de la variable qui donne accès au terme de la substitution explicite.

Avant, on ne gelait que les variables qui ne devaient pas être substituées avec la règle ABS-$\top$; avec ces nouvelles règles, on gèle également les variables dont on

$$\text{BOX}\ \frac{}{\bullet \in \mathcal{E}_{\varphi,\tau}} \qquad \text{ABS-}\top\ \frac{\mathcal{C} \in \mathcal{E}_{\varphi\cup\{x\},\top}}{(\lambda x.\mathcal{C}) \in \mathcal{E}_{\varphi,\top}} \qquad \text{ABS-}\bot\ \frac{\mathcal{C} \in \mathcal{E}_{\varphi,\bot}}{(\lambda x.\mathcal{C}) \in \mathcal{E}_{\varphi,\bot}}$$

$$\text{APP-LEFT}\ \frac{\mathcal{C} \in \mathcal{E}_{\varphi,\bot}}{\mathcal{C}\ t \in \mathcal{E}_{\varphi,\tau}} \qquad \text{APP-RIGHT-S}\ \frac{\mathcal{C} \in \mathcal{E}_{\varphi,\top} \quad t \in S_\varphi}{t\ \mathcal{C} \in \mathcal{E}_{\varphi,\tau}}$$

$$\text{ES-LEFT}\ \frac{\mathcal{C} \in \mathcal{E}_{\varphi,\tau}}{\mathcal{C}[x \setminus t] \in \mathcal{E}_{\varphi,\tau}} \qquad \text{ES-LEFT-FROZEN}\ \frac{\mathcal{C} \in \mathcal{E}_{\varphi\cup\{x\},\tau} \quad t \in S_\varphi}{\mathcal{C}[x \setminus t] \in \mathcal{E}_{\varphi,\tau}} \qquad \text{ES-RIGHT}\ \frac{\mathcal{C}' \in \mathcal{E}_{\varphi,\tau} \quad \mathcal{C} \in \mathcal{E}_{\varphi,\bot}}{\mathcal{C}'(x)[x \setminus \mathcal{C}] \in \mathcal{E}_{\varphi,\tau}}$$

FIGURE 2.49 – Contexte d'évaluation paramétrique

sait qu'elles ne peuvent être substituées que par un terme qui ne crée pas de radical, comme avec la règle ES-LEFT-FROZEN.

Toutes les règles des contextes d'évaluation sont présentées dans la figure 2.49, nous précisons que dans la règle ES-RIGHT, la variable x n'est pas capturée par $\mathcal{C}(\bullet)$ grâce à la convention de Barendregt.

2.6 Conclusion

Dans ce chapitre, nous avons présenté le lambda-calcul et ses termes, ainsi que ses réductions. Nous avons également abordé la notion de stratégie de réduction et les stratégies les plus connues. Nous avons ensuite discuté des formes normales et des stratégies normalisantes. Nous avons parlé des stratégies faibles itérées, qui nous permettent de trouver une forme normale en relançant l'évaluation des sous-termes. Enfin, nous avons abordé les stratégies d'évaluation forte, qui autorisent la réduction sous les abstractions.

Une fois le cadre établi, nous nous sommes intéressés à la notion de nécessité et plus précisément aux positions nécessaires d'un terme. Le problème est que formaliser cette définition est très difficile. Trouver toutes les positions nécessaires d'un terme est un problème qui n'est pas calculable. C'est pourquoi nous avons défini une approximation des positions nécessaires grâce à une approximation des variables qui ne seront jamais substituées, appelées variables gelées.

Cette approximation doit trouver uniquement des positions nécessaires, suffisamment de positions dans un terme qui n'est pas en forme normale (au moins une) et plus de positions nécessaires que les critères qui existent déjà dans la littérature, afin d'avoir un intérêt.

Avec les contextes qui représentent les positions et l'approximation des positions nécessaires, nous avons défini un calcul d'appel par nécessité en réduction forte dans la syntaxe du lambda-calcul pur. Bien que le calcul ne puisse pas trouver toutes les positions nécessaires, il est capable d'identifier un sous-ensemble suffisamment grand

pour ne jamais être bloqué là où il reste des réductions, et donc ne jamais s'arrêter avant d'avoir atteint la forme normale.

Notre calcul partage les termes en ne substituant que les valeurs afin de réduire le nombre de réductions dupliquées. Le problème avec ce calcul défini avec la syntaxe du lambda-calcul pur et avec la bêta-réduction restreinte aux valeurs est que l'on peut s'y retrouver bloqué, comme illustré dans la section 2.4.5.

Pour résoudre le problème, une des solutions aurait été d'introduire des règles de commutation pour avoir accès aux radicaux inaccessibles. Cependant, cette solution impliquait l'introduction de règles de réduction qui n'avaient aucun intérêt comparé à la réduction du lambda-calcul pur, ce ne sont que des règles de réduction destinées à traiter un problème technique. De ce fait, cela rend les séquences de réduction moins lisibles.

C'est pourquoi nous avons choisi d'utiliser les substitutions explicites qui sont un outil bien connu pour représenter le partage et de remplacer la bêta-réduction par trois règles. La règle dB permet de transformer un radical en une substitution explicite à distance. La règle lsv permet de substituer une occurrence d'une variable liée à une substitution explicite. Et la règle gc supprime les substitutions explicites dont la variable n'apparaît pas dans le terme.

Contribution Nous avons présenté dans ce chapitre deux contributions. Premièrement, l'approximation des formes normales, et deuxièmement, un calcul d'appel par nécessité en réduction forte qui utilise cette approximation.

Notre approximation est basée sur le fait qu'il est possible de séparer les positions des sous-termes d'un terme donné en deux catégories. Les sous-termes se trouvant dans une position qui est susceptible de les faire disparaître après bêta-réduction, et ceux qui sont dans une position qui ne peut plus les faire disparaître.

La seule manière pour un terme de disparaître est qu'il soit en argument d'une abstraction. Nous avons défini la notion de position isolée qui est une position qui ne sera jamais appliquée. Les termes à «top-level» sont isolés car ils ne seront jamais appliqués ni donnés en argument à une fonction.

Les termes à droite d'une variable libre sont à une position isolée car il n'y a pas de bêta-réduction possible et la variable ne peut pas être substituée. En revanche, dans le cas des variables liées, on distingue deux cas. Si la variable est liée à une abstraction qui n'est pas, et ne sera jamais, appliquée, on se retrouve dans la même situation que pour une variable libre. Mais si la variable est liée à une abstraction appliquée, c'est autre chose : l'argument de l'abstraction peut faire disparaître les «arguments» de la variable liée.

On utilise la notion de variable gelée pour savoir si une variable est liée à une abstraction qui est dans une position appliquée ou non. Les variables gelées sont celles liées à une abstraction qui ne disparaîtra jamais. On a la garantie que les termes passés en argument d'une variable gelée sont à des positions qui ne disparaîtront jamais. Une variable gelée et tous ses arguments forment ce que l'on appelle une structure.

Nous utilisons les notions de position isolée et de variable gelée pour caractériser notre approximation des positions nécessaires. On commence par trouver les abstractions qui sont en position isolée. Ensuite, nous trouvons les variables gelées, c'est-à-dire

celles liées aux abstractions qui sont dans une position isolée. Enfin, nous utilisons les variables gelées pour trouver les positions nécessaires et d'autres positions isolées qui permettent d'itérer le processus plus profondément dans le terme. Les arguments d'une variable gelée qui se trouve dans une position nécessaire sont également dans une position nécessaire.

Grâce à cette approximation des positions nécessaires, nous avons pu définir un calcul d'appel par nécessité en réduction forte dans le lambda-calcul avec substitutions explicites. L'approximation nous permet de n'effectuer des réductions que sur des termes nécessaires. De plus, puisque cette approximation cible toujours un radical, nous sommes certains de ne jamais rester bloqués et d'atteindre une forme normale. En effet, au fil des réductions, l'approximation va trouver de nouvelles positions nécessaires et nous permettre de réduire de nouveau les termes jusqu'à la forme normale.

Le premier calcul et l'approximation des positions nécessaires présentée dans le lambda-calcul pur est complet grâce au fait qu'il trouve tous les radicaux ciblés par «leftmost-outermost».

Ce calcul est défini avec des contextes et nous permet de gérer le partage grâce aux substitutions explicites. De plus, dans certaines conditions, il autorise la réduction des arguments d'une fonction, ce qui permet de réduire le nombre de calculs dupliqués. Ce calcul est une version plus avancée du calcul déjà existant de Balabonski et al. [9]. Notre calcul permet de réduire sous les abstractions qui sont appliquées, ce qui nous permet de faire une réduction forte par rapport au calcul déjà existant.

Pour la suite Dans ce chapitre, nous avons défini un calcul avec substitution explicite. En essayant de prouver des lemmes, nous nous sommes aperçus que la notation avec les contextes ne se prête pas très bien aux preuves. Cela s'est avéré très fastidieux, en particulier lorsque nous avons voulu prouver la correction et la confluence. Nous ne donnons pas ici le détail des preuves car nous les donnerons dans la nouvelle version du calcul du chapitre 3. Dans le prochain chapitre, nous allons définir le même calcul d'appel par nécessité en réduction forte, avec un nouveau formalisme qui se prête mieux aux démonstrations rigoureuses et qui est plus adapté à l'implémentation que nous présenterons dans le chapitre 6.

Chapitre 3

Calcul λ_{sn} : Appel par nécessité en réduction forte

Dans ce chapitre nous nous sommes écartés du style de présentation plus traditionnel de la réduction faible appelée par nécessité, basée sur des contextes d'évaluation, et nous nous sommes tournés vers les règles de réduction de style SOS [30]. Nous présentons le calcul λ_{sn} dans la section 3.1 avec ce nouveau style.

Ensuite, nous allons présenter les différentes propriétés que nous avons prouvées avec cette nouvelle formalisation. Les deux propriétés les plus importantes sont la correction abordée dans la section 3.2 et la complétude vue dans la section 3.3. Nous expliquerons également pourquoi notre calcul λ_{sn} n'est pas confluent.

Toutes les preuves présentées dans ce chapitre seront abordées avec un certain niveau de détail, grâce à des preuves formelles dans le chapitre 5.

3.1 Calcul

L'objectif est de définir un calcul en appel par nécessité avec réduction forte sans utiliser de contexte car comme nous l'avons vu cela rend les preuves plus difficiles, mais plutôt en utilisant des règles de réduction définies inductivement à l'aide de règles d'inférences qui elles sont bien plus adéquates.

La règle APP-LEFT des contextes (Fig. 2.37) selon laquelle $\mathcal{C}\ t$ est un contexte de réduction lorsque $\mathcal{C}$ est un contexte de réduction se transforme en la règle suivante : si $t \to_\beta t'$, alors $t\ u \to_\beta t'\ u$. Dans les deux cas, l'idée est que l'on peut réduire le sous-terme gauche d'une application.

Pour illustrer cela, prenons comme exemple les stratégies d'appel par nom et d'appel par valeur. On voit que, que ce soit dans la traduction des contextes (Fig. 2.29) en règles inductives (Fig. 3.1) de la stratégie d'appel par nom ou bien des contextes (Fig 2.31) en règles inductives (Fig. 3.2) de la stratégie d'appel par valeur, les deux règles de propagation sont conservées. On remarque également que la règle BOX des contextes a disparu car elle n'avait pas d'équivalent dans les règles inductives. Et on voit que dans les règles inductives une nouvelle règle a fait son apparition, la règle BETA. Les contextes nous disaient où la bêta-réduction était applicable, tandis

$$
\frac{t \to_\beta t'}{t\ u \to_\beta t'\ u}\ \text{APP-LEFT} \qquad \frac{t \in S \qquad u \to_\beta u'}{t\ u \to_\beta t\ u'}\ \text{APP-RIGHT} \qquad \frac{}{\lambda x.t\ u \to_\beta t\{x \setminus u\}}\ \text{BETA}
$$

FIGURE 3.1 – Réduction définie inductivement pour la stratégie d'appel par nom

$$
\frac{t \to_\beta t'}{t\ u \to_\beta t'\ u}\ \text{APP-LEFT} \qquad \frac{u \to_\beta u'}{t\ u \to_\beta t\ u'}\ \text{APP-RIGHT} \qquad \frac{u \in \text{VO}}{\lambda x.t\ u \to_\beta t\{x \setminus u\}}\ \text{BETA}
$$

FIGURE 3.2 – Réduction définie inductivement pour la stratégie d'appel par valeur

que maintenant, tout est défini dans les règles inductives et notamment la règle de bêta-réduction.

Maintenant, nous voulons traduire notre calcul λ_{sn} dans cette nouvelle formalisation, mais ce n'est pas si simple car il y a plus de notions à formaliser. Dans un premier temps, nous formalisons les règles les plus simples dans la figure 3.3.

On a la flèche de réduction du calcul $\xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}}$ qui est paramétrée par trois arguments. Les paramètres φ et μ sont déjà connus, ce sont l'ensemble des variables gelées et le mode. Mais on voit apparaître un nouveau paramètre ρ. Ce paramètre fait référence à la règle de réduction que l'on utilise. En effet, cela permet de factoriser les deux règles de réduction dB et lsv car elles ont exactement le même comportement dans toutes les règles de propagation (Fig. 3.3).

On voit que dans la règle ES-RIGHT la réduction dans le premier prémisse, id_x, n'est ni dB ni lsv. Cela vient du fait que, dans les règles avec les contextes, nous utilisons $\mathcal{C}(x)$ pour parler de la présence d'une variable dans un terme. En effet si un terme est défini comme dans un contexte d'évaluation dans lequel il y a une variable x alors cette variable x est atteignable dans ce terme. Or dans cette nouvelle formalisation nous n'avons plus les contextes donc nous avons besoin d'une nouvelle règle qui exprime le fait qu'une variable soit atteignable par nos règles de réduction.

La règle ID (Fig. 3.4) nous permet d'identifier le fait que la variable en question soit bien dans le terme à une position atteignable. La règle SUB permet de substituer une occurrence atteignable d'une variable. Quant aux règles dB et lsv, elles permettent de faire le lien avec les flèches $\to_{\mathsf{db}}$ et $\dot{\to}_{\mathsf{lsv}}$ définies dans la figure 3.5.

Dans la figure 3.5, nous trouvons deux règles de base : dB-BASE qui permet d'appliquer la règle dB et LSV-BASE qui permet d'appliquer la règle lsv. Les trois autres règles permettent de traverser les substitutions explicites ce qui correspond au L dans les règles dB et lsv précédentes (Fig. 2.39). La règle dB-σ permet d'effectuer l'application de la règle dB à distance. Les règles LSV-σ et LSV-σ-φ permettent de sortir les substitutions explicites imbriquées, comme le faisaient les règles de réduction dans le calcul avec les contextes (Fig. 2.39).

APP-LEFT
$$\frac{t \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} t'}{t\ u \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t'\ u}$$

APP-RIGHT
$$\frac{t \in \mathcal{S}_\varphi \qquad u \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'}{t\ u \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t\ u'}$$

ABS-$\top$
$$\frac{t \xrightarrow{\rho,\varphi\cup\{x\},\top}_{\mathsf{sn}} t'}{\lambda x.t \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} \lambda x.t'}$$

ABS-$\bot$
$$\frac{t \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} t'}{\lambda x.t \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} \lambda x.t'}$$

ES-LEFT
$$\frac{t \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t'}{t[x \setminus u] \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t'[x \setminus u]}$$

ES-LEFT-FROZEN
$$\frac{t \xrightarrow{\rho,\varphi\cup\{x\},\mu}_{\mathsf{sn}} t' \qquad u \in \mathcal{S}_\varphi}{t[x \setminus u] \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t'[x \setminus u]}$$

ES-RIGHT
$$\frac{t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn}} t \qquad u \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u'}{t[x \setminus u] \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t[x \setminus u']}$$

FIGURE 3.3 – Première partie des règles de réduction du calcul λ_{sn}

ID
$$\frac{}{x \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn}} x}$$

SUB
$$\frac{}{x \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\mu}_{\mathsf{sn}} v}$$

dB
$$\frac{t \to_{\mathsf{db}} t'}{t \xrightarrow{\mathsf{dB},\varphi,\mu}_{\mathsf{sn}} t'}$$

lsv
$$\frac{t \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'}{t \xrightarrow{\mathsf{lsv},\varphi,\mu}_{\mathsf{sn}} t'}$$

FIGURE 3.4 – Deuxième partie des règles de réduction de notre calcul λ_{sn}

$$\text{DB-BASE}\quad \frac{}{(\lambda x.t)\ u \to_{\mathsf{db}} t[x \setminus u]}$$

$$\text{DB-}\sigma\quad \frac{t\ u \to_{\mathsf{db}} v}{t[x \setminus w]\ u \to_{\mathsf{db}} v[x \setminus w]}$$

$$\text{LSV-BASE}\quad \frac{t \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\mu}_{\mathsf{sn}} t' \qquad v \in \mathit{Val}}{t[x \setminus v] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'[x \setminus v]}$$

$$\text{LSV-}\sigma\quad \frac{t[x \setminus u] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'}{t[x \setminus u[y \setminus w]] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'[y \setminus w]}$$

$$\text{LSV-}\sigma\text{-}\varphi\quad \frac{t[x \setminus u] \xrightarrow{\varphi\cup\{y\},\mu}_{\mathsf{lsv}} t' \qquad w \in \mathcal{S}_\varphi}{t[x \setminus u[y \setminus w]] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'[y \setminus w]}$$

FIGURE 3.5 – Règles de réduction auxiliaire pour λ_{sn}

La flèche $\xrightarrow{\varphi,\mu}{}^{*}_{\mathsf{sn}}$ représente une séquence de flèches $\xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}}$ pour une ρ quelconque.

Propriétés du calcul

Nous présentons ici quelques lemmes qui sont des propriétés de base de notre calcul.

Lemme 1 (Monotonie de la réduction par rapport au mode)**.** *Si un terme est réductible en mode $\bot$, il est réductible en mode $\top$.*

— $\forall t, \varphi, \rho, \quad t \xrightarrow{\rho,\varphi,\bot}_{sn} t' \implies t \xrightarrow{\rho,\varphi,\top}_{sn} t'$
— $\forall t, \varphi, \quad t \xrightarrow{\varphi,\bot}_{lsv} t' \implies t \xrightarrow{\varphi,\top}_{lsv} t'$

Démonstration. Par induction mutuelle sur la dérivation des réductions $t \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} t'$ ou $t \xrightarrow{\varphi,\bot}_{\mathsf{lsv}} t'$.

Dans les règles APP-LEFT, APP-RIGHT et *dB*, les prémisses ne dépendent pas du mode, donc sur ces trois règles nous avons la monotonie en question. Les règles ID et SUB ne dépendent pas du mode non plus. La règle ABS-$\top$ fixe le mode à $\top$, donc elle n'est pas traitée.

— ES-LEFT : On a $t = u[x\backslash v]$ avec $u[x\backslash v] \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u'[x\backslash v]$ et avec $u \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u'$ comme prémisse. Par hypothèse d'induction sur $u \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u'$ on a $u \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'$ et donc par la règle ES-LEFT on a $u[x \setminus v] \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'[x \setminus v]$.
— ES-LEFT-FROZEN : On a $t = u[x \setminus v]$ avec $u[x \setminus v] \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u'[x \setminus v]$ et avec $u \xrightarrow{\rho,\varphi\cup\{x\},\bot}_{\mathsf{sn}} u'$ et $v \in S_\varphi$ comme prémisses. Par hypothèse d'induction on a $u \xrightarrow{\rho,\varphi\cup\{x\},\top}_{\mathsf{sn}} u'$ et donc par la règle ES-LEFT-FROZEN on a $u[x \setminus v] \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'[x \setminus v]$.
— ES-RIGHT : On $t = u[x\backslash v]$ avec $u[x\backslash v] \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u[x\backslash v']$ et avec $u \xrightarrow{\mathsf{id}_x,\varphi,\bot}_{\mathsf{sn}} u$ et $v \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} v'$ comme prémisses. Par hypothèse d'induction sur $u \xrightarrow{\mathsf{id}_x,\varphi,\bot}_{\mathsf{sn}} u$

on a $u \xrightarrow{\mathsf{id}_x,\varphi,\top}_{\mathsf{sn}} u$ et donc par la règle ES-RIGHT on a $u[x \setminus v] \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u[x \setminus v']$.

- LSV-BASE : On $t = u[x\setminus v]$ avec $u[x\setminus v] \xrightarrow{\varphi,\bot}_{\mathsf{lsv}} u'[x\setminus v]$ et avec $u \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\bot}_{\mathsf{sn}} u'$ et $v \in \mathit{Val}$ comme prémisses. Par hypothèse d'induction sur $u \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\bot}_{\mathsf{sn}} u'$ on a $u \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\top}_{\mathsf{sn}} u'$ et donc par la règle LSV-BASE on a $u[x \setminus v] \xrightarrow{\varphi,\top}_{\mathsf{lsv}} u'[x \setminus v]$
- LSV-σ : On $t = u[x \setminus v[y \setminus w]]$ avec $u[x \setminus v[y \setminus w]] \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u'[y \setminus w]$ et avec $u[x \setminus v] \xrightarrow{\varphi,\bot}_{\mathsf{lsv}} u'$ comme prémisse. Par hypothèse d'induction sur $u[x \setminus v] \xrightarrow{\varphi,\bot}_{\mathsf{lsv}} u'$ on a $u[x \setminus v] \xrightarrow{\varphi,\top}_{\mathsf{lsv}} u'$ et donc par la règle LSV-σ on a $u[x \setminus v[y \setminus w]] \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'[y \setminus w]$
- LSV-σ-φ : On $t = u[x \setminus v[y \setminus w]]$ avec $u[x \setminus v[y \setminus w]] \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u'[y \setminus w]$ et avec $u[x \setminus v] \xrightarrow{\varphi,\bot}_{\mathsf{lsv}} u'$ et $w \in S_\varphi$ comme prémisses. Par hypothèse d'induction sur $u[x \setminus v] \xrightarrow{\varphi,\bot}_{\mathsf{lsv}} u'$ on a $u[x \setminus v] \xrightarrow{\varphi,\top}_{\mathsf{lsv}} u'$ et donc par la règle LSV-σ-φ on a $u[x \setminus v[y \setminus w]] \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'[y \setminus w]$ □

Lemme 2 (Monotonie de la structure par rapport aux variable gelées)**.** *Si un terme est une structure (Fig. 2.48) pour un certain ensemble de variables gelées, alors le terme est aussi une structure si l'on ajoute une variable à cet ensemble.*
$(\forall t, x, \varphi, \quad t \in S_\varphi \implies t \in S_{\varphi \cup \{x\}})$.

Démonstration. La preuve de ce lemme est triviale car il n'y a pas de prémisse négative dans la définition de la structure (par exemple $x \notin \varphi$) donc ajouter des variables dans l'ensemble φ ne peut pas enlever d'élément de l'ensemble S_φ. □

Lemme 3 (Monotonie de la réduction par rapport aux variables gelées)**.** *Si un terme est réductible avec un ensemble de variables gelées φ, alors il l'est aussi avec tout ensemble de variables gelées contenant φ.*

- $\forall t, t', x, \varphi, \rho, \mu, \quad t \xrightarrow{\rho,\varphi,\mu}_{sn} t' \implies t \xrightarrow{\rho,\varphi\cup\{x\},\mu}_{sn} t'$
- $\forall t, t', x, \varphi, \rho, \mu, \quad t \xrightarrow{\varphi,\mu}_{lsv} t' \implies t \xrightarrow{\rho,\varphi\cup\{x\},\mu}_{sn} t'$

Démonstration. Par induction mutuelle sur la dérivation des réductions $t \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t'$ ou $t \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'$. □

3.2 Correction

Quand on parle de correction, on parle de correction de notre calcul λ_{sn} par rapport au lambda-calcul traditionnel. Pour que notre calcul λ_{sn} soit correct par rapport au lambda-calcul, il faut que le résultat de notre calcul λ_{sn} corresponde au résultat du lambda-calcul. Autrement dit, si notre calcul λ_{sn} produit un résultat, alors il correspond à celui du lambda-calcul.

Dans un premier temps, il nous faut une fonction pour pouvoir comparer nos termes et les termes du lambda-calcul traditionnel. Cette fonction est la fonction de dépliage présentée dans la figure 2.46.

— Relation d'ordre 1 (terme $\to$ terme)
$A \Longleftrightarrow A$: Égalité

$A \longrightarrow B$: Réduction

$A \dashrightarrow A^\star$: Dépliage

— Relation d'ordre 2 (relation $\to$ relation)
$R_1 \rightsquigarrow R_2$: Déduction

— Annotation sur les relations
(H) : Hypothèse

(HI) : Hypothèse d'induction

(C) : Conclusion

FIGURE 3.6 – Légende pour les preuves graphiques

Le théorème qui suit (Th. 1) sera intégralement prouvé avec Abella, donc nous nous autorisons ici un style elliptique.

Théorème 1 (Correction). *Si un terme t se réduit en une forme irréductible u dans notre calcul λ_{sn}, alors le terme déplié $t^\star$ se réduit dans le lambda-calcul en la version dépliée $u^\star$ de la forme irréductible u de notre calcul λ_{sn}.*
($t \xrightarrow{\mathrm{FV}(t),\top}{}^{*}_{sn} u \wedge u$ irréductible $\implies t^\star \to^*_\beta u^\star \wedge u^\star \in \mathscr{N}_\lambda$).

La preuve de correction est découpée en plusieurs étapes. Premièrement, nous allons définir le lemme de simulation qui met en relation les réductions de notre calcul λ_{sn} et celles du lambda-calcul dans la section 3.2.1. Ensuite, nous caractérisons les formes normales de notre calcul λ_{sn} dans la section 3.2.2, afin de prouver que nos formes normales correspondent bien aux formes normales du lambda-calcul dans la section 3.2.3.

3.2.1 Simulation

Ce chapitre présente des preuves et une partie de celles-ci sont graphiques. La figure 3.6 présente les différentes flèches et annotations que l'on peut retrouver dans ces différents graphiques.

Le lemme de simulation stipule que si un terme t de notre calcul se réduit en un terme t' par notre calcul λ_{sn}, alors il existe une séquence de réduction dans le lambda-calcul qui réduit le terme déplié $t^\star$ en le terme déplié $t'^\star$.

Lemme 4 (Simulation). *Si $t \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t'$ avec $\rho \in \{dB, lsv\}$ alors $t^\star \to^*_\beta t'^\star$*

Démonstration. Par induction sur la dérivation de la réduction $t \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t'$.

On voit que les règles qui sont dans la figure 3.3 ne font que propager la flèche sn aux sous-termes ce qui est compatible avec la bête-réduction. Et les règles de la figure 3.4 sont les flèches qui correspondent aux réductions. Notre lemme ne concernant que les deux réductions dB et lsv. Donc il faut prouver que la règle dB correspond bien à zéro, une ou plusieurs bêta-réductions, ce qui est fait par le lemme 5. Et il faut prouver la même chose pour la règle lsv, ce qui est fait par le lemme 7. □

La réduction dB correspond à au moins une bêta-réduction. Là où la bêta-réduction transforme un radical en une substitution, la réduction dB transforme le radical en une substitution explicite qui, modulo dépliage, va aussi se transformer en une substitution classique. Dans le cas où le radical est déjà dans une substitution explicite, une étape de réduction dB peut même se transformer en plusieurs bêta-réductions.

Lemme 5 (Simulation dB)**.** *Si* $t \to_{\mathsf{db}} t'$ *alors* $t^\star \to^*_\beta t'^\star$.

Démonstration. Par induction sur la dérivation de la réduction $t \to_{\mathsf{db}} t'$.

Comme on peut le voir dans la figure 3.5, la règle dB comporte deux cas : DB-BASE et DB-σ. La règle DB-BASE est le cas de base, c'est-à-dire le cas où la bêta n'est pas effectuée à distance, car il n'y a pas de substitution explicite entre l'abstraction et son argument, ce qui a un équivalent assez simple à voir dans le lambda-calcul. La flèche dB passe d'une abstraction appliquée à une substitution explicite et lors du dépliage, la substitution explicite est transformée en substitution classique, ce qui correspond précisément à l'étape de la bêta-réduction.

Ensuite, il y a la règle DB-σ qui permet d'appliquer la règle à distance. Elle permet de traverser les substitutions explicites pour aller jusqu'à l'argument de l'abstraction. On a comme hypothèse d'induction que si $tu \to_{\mathsf{db}} v$, alors $(t\ u)^\star \to_{\mathsf{db}} v^\star$. Avec le dépliage, on sait que $(t\ u)^\star = t^\star\ u^\star$. La preuve utilise aussi la proposition 2 qui nous permet d'extraire la substitution de l'application, à condition que la variable de la substitution n'apparaisse pas dans l'argument du radical. La convention de Barendregt nous apporte cette garantie. Puis, pour finir, on utilise la proposition 1 qui nous permet d'appliquer l'hypothèse d'induction sous la substitution. Une preuve graphique est présentée dans la figure 3.7. □

Lemme 6 (Adéquation)**.** $(t\{x \setminus u\})^\star = t^\star\{x \setminus u^\star\}$

Démonstration. Par induction sur le terme t. □

Lemme 7 (Simulation lsv)**.** *Si* $t \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'$ *alors* $t^\star = t'^\star$.

Démonstration. Par induction sur la dérivation de la réduction $t \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'$.

Dans la figure 3.5, la règle lsv comporte trois cas : LSV-BASE, LSV-σ et LSV-σ-φ. De manière similaire à ce qui est fait pour les règles dB, il y a d'abord une règle pour le cas de base où il n'y a pas de substitution explicite, puis un cas pour permettre la sortie de ce type de substitution.

Dans la règle LSV-BASE, on part de l'hypothèse que $t \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\mu}_{\mathsf{sn}} t'$ par définition, et grâce au lemme 8, on peut prouver que $t^\star\{x \setminus v^\star\} = t'^\star\{x \setminus v^\star\}$., ce qui

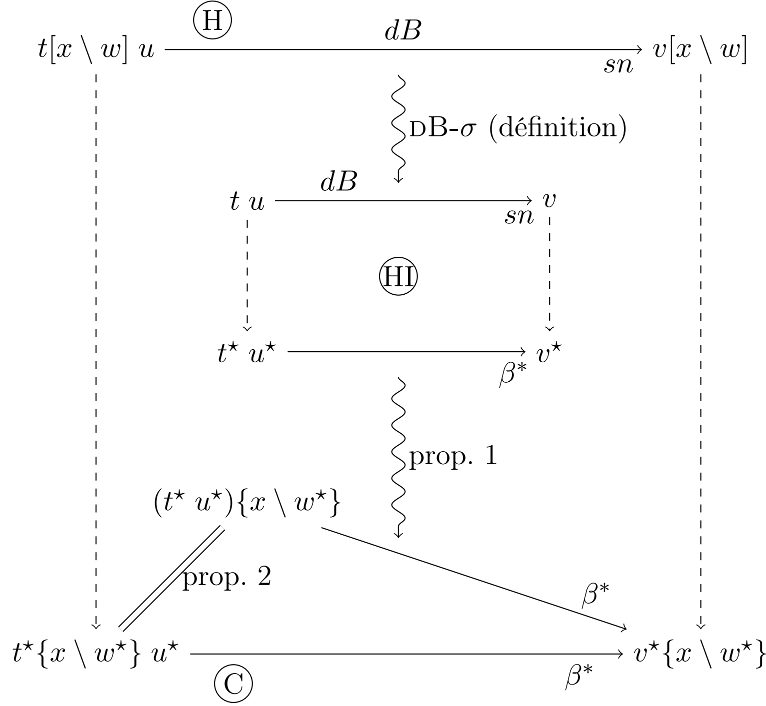

FIGURE 3.7 – Graphique pour la preuve du cas DB-σ pour la simulation

nous permet de prouver le cas de base. Une preuve graphique de cette assertion est présentée dans la figure 3.8, où v est une valeur.

Ensuite, pour les règles LSV-σ et LSV-σ-φ, on utilise le lemme 3. La condition nécessaire pour l'utiliser est toujours remplie par la convention de Barendregt. Et pour terminer, on utilise l'hypothèse d'induction. Une preuve graphique est présentée dans la figure 3.9. Le raisonnement par cas sur l'hypothèse donne bien deux cas, mais nous utilisons la prémisse commune pour faire la preuve dans les deux cas, donc nous les traitons avec la même preuve graphique. □

On voit dans les deux lemmes que l'on vient d'énoncer (Lem. 5 et 7) que la règle dB correspond bien au moins à une bêta-réduction dans le lambda-calcul modulo dépliage, alors que la règle lsv correspond au dépliage. La règle lsv est une règle qui substitue une occurrence d'une variable, alors que la bêta-réduction substitue toutes les occurrences en une seule réduction. La fonction de dépliage prend un terme et transforme toutes les substitutions explicites en substitutions classiques, ce qui a pour effet de substituer toutes les occurrences de la variable en question. Pour résumer, la flèche lsv substitue une occurrence, ce qui veut dire que la seule différence entre la partie gauche et la partie droite de la flèche est qu'une variable a été substituée. Mais le dépliage appliqué au membre droit ou gauche va finir de substituer toutes les occurrences. Donc, modulo dépliage, la flèche lsv ne fait rien dans le lambda-calcul.

Lemme 8 (Simulation sub)**.** *Si* $t \xrightarrow{sub_{x \setminus v}, \varphi, \mu}_{sn} t'$ *alors* $t^\star\{x \setminus v\} = t'^\star\{x \setminus v\}$.

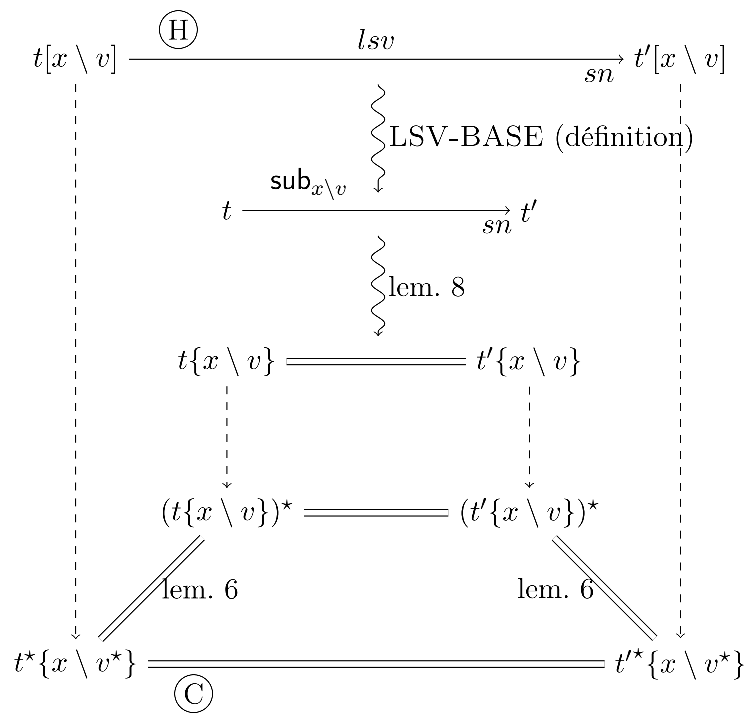

FIGURE 3.8 – Graphique pour la preuve du cas LSV-BASE pour la simulation avec v une valeur

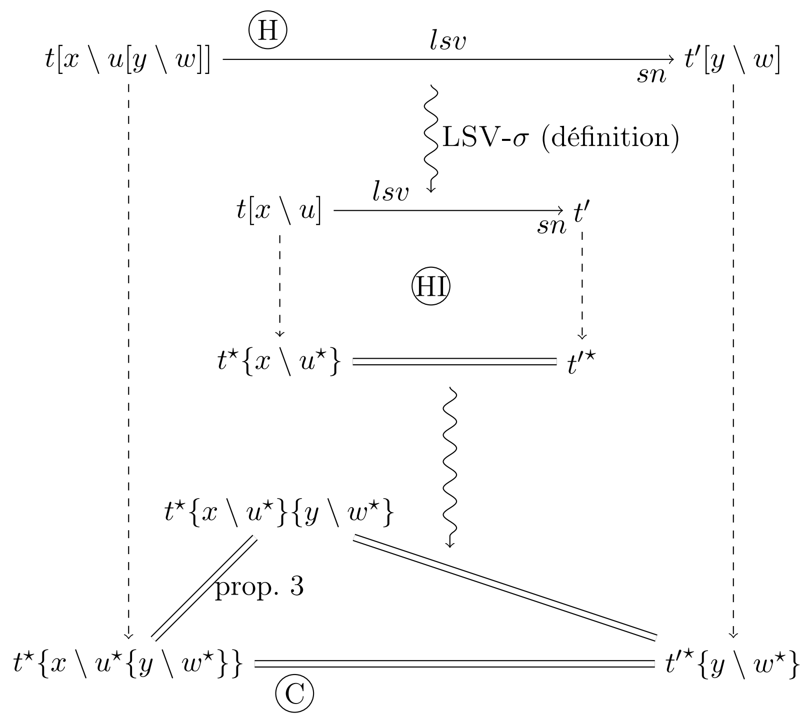

Figure 3.9 – Graphique pour la preuve du cas LSV-σ et LSV-σ-φ pour la simulation

$$\frac{}{x \in \mathscr{N}}\ \text{NF-VAR} \qquad \frac{t \in \mathscr{N} \quad t \in S_\varphi \quad u \in \mathscr{N}}{t\,u \in \mathscr{N}}\ \text{NF-APP} \qquad \frac{t \in \mathscr{N}}{\lambda x.t \in \mathscr{N}}\ \text{NF-ABS}$$

$$\frac{t \notin \mathscr{V}_x \quad t \in \mathscr{N}}{t[x \setminus u] \in \mathscr{N}}\ \text{NF-ES} \qquad \frac{t \in \mathscr{N} \quad u \in \mathscr{N} \quad u \in S_\varphi}{t[x \setminus u] \in \mathscr{N}}\ \text{NF-ES-S}$$

FIGURE 3.10 – Caractérisation des formes normales avec les substitutions explicites

Démonstration. Par induction sur la dérivation de $t \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\mu}_{\mathsf{sn}} t'$. La règle SUB substitue une occurrence d'une variable. Donc de la même manière que pour la règle *lsv*, la différence entre la partie gauche et la partie droite de la flèche est qu'une occurrence a été substituée. Donc, si l'on substitue toutes les occurrences de la variable dans le terme de gauche et dans le terme de droite, peu importe le nombre d'occurrences de la variable qui ont déjà été substituées, elles le seront toutes et donc les deux termes sont les mêmes. □

Avec ce dernier lemme, nous avons complété tous les lemmes utilisés dans le lemme de simulation 4.

3.2.2 Caractérisation des formes normales

Il faut démontrer que les termes irréductibles pour notre calcul λ_{sn} correspondent à des formes normales du lambda-calcul pur.

Pour cela, nous utilisons une caractérisation des formes normales pour notre calcul λ_{sn}. Puis, il faut prouver que cette caractérisation est correcte, c'est-à-dire qu'un terme qui est en forme normale selon notre caractérisation ne peut plus être réduit, et qu'un terme qui ne peut plus être réduit est bien considéré comme étant en forme normale par notre caractérisation.

Maintenant que nous utilisons les substitutions explicites, les structures, les contextes d'évaluation et les règles de réduction ont changé, donc la définition des formes normales change aussi. La nouvelle caractérisation des formes normales est présentée dans la figure 3.10.

Premièrement, toutes les variables sont en forme normale (règle NF-VAR). Ensuite, une application peut se réduire de trois manières différentes : si le sous-terme de gauche se réduit, si le sous-terme de droite se réduit ou si l'application elle-même est un radical. Donc, pour qu'une application soit en forme normale (règle NF-APP), il y a trois conditions : les deux sous-termes doivent être en forme normale et le sous-terme de gauche doit être une structure. Le fait que le sous-terme de gauche soit une structure garantit qu'il ne peut pas être une abstraction.

Dans les règles NF-APP et NF-ES-S,nous utilisons la définition classique des structures qui prend en paramètre un ensemble de variables gelées. Mais nous n'utilisons

la notion de structure que pour exprimer le fait qu'un terme n'est pas une réponse. Ainsi, la variable de tête de la structure n'importe pas. Par conséquent, l'ensemble passé en paramètre de la structure n'a pas d'importance, à condition que la variable de tête soit bien présente dans cet ensemble. Il suffit donc qu'il existe un ensemble.

Une abstraction ne peut être réduite que si son corps peut l'être. Donc, pour que l'abstraction soit en forme normale (règle NF-ABS), il suffit que son corps le soit et comme φ n'importe pas, le mode n'a pas d'importance non plus.

Pour la dernière forme de terme, à savoir la substitution explicite, il y a deux cas. Dans le premier cas, au moins une occurrence de la variable de la substitution explicite se trouve dans le sous-terme gauche, et dans le second cas elle n'y est pas. Pour distinguer ces deux cas, on utilise la notion de vivacité des variables. Une variable est considérée comme étant «en vie» si elle n'est pas dans une substitution explicite qui va être supprimée par la règle *gc*.

Définition 18 (Variables vivantes)**.** *La caractérisation des variables vivantes que l'on note $t \in \mathscr{V}_x$ pour dire que la variable x est vivante dans le terme t est donnée dans le figure 3.11.*

Dans le cadre des formes normales, la notion de variable vivante suffit pour nous garantir qu'une variable ne va pas disparaître car, dans un terme en forme normale, il n'y a plus de radical et donc plus de réduction qui pourrait entraîner la disparition d'une variable. À ce stade, la seule chose qui peut faire disparaître une variable est la règle *gc*.

Nous avons deux nouvelles règles : NF-ES et NF-ES-S. De la même manière que pour les règles des structures, NF-ES fait référence au cas où la variable de la substitution explicite n'est pas vivante dans le terme. Et la règle NF-ES-S fait référence au cas où la variable est vivante dans le terme. Dans ce cas le terme doit évidemment être en forme normale, mais le corps de la substitution explicite l'est aussi et, en plus, le corps de la substitution explicite doit être une structure pour être sûr que sa substitution ne créera pas de radical.

Donc, dans le cas de la substitution explicite, si la variable n'est pas vivante dans le sous-terme de gauche, la seule condition pour que le terme soit en forme normale est que le sous-terme de gauche soit lui-même en forme normale. Cela sous-entend que la substitution explicite va disparaître par *gc* et donc qu'il n'y a pas de restriction sur le sous-terme de droite. En revanche, si la variable de la substitution peut être dans le sous-terme de gauche, il faut aussi que ce même sous-terme soit en forme normale, mais ce n'est pas tout. Comme la variable donne accès au sous-terme de droite, il y a aussi des restrictions sur ce sous-terme. Il doit naturellement être en forme normale pour qu'aucune réduction ne puisse avoir lieu dans le sous-terme. Mais il doit aussi être une structure. Encore une fois, on utilise le fait que si un terme est une structure, il ne peut pas être une réponse (Lem. 9). Car, si le sous-terme de droite est une réponse, on pourrait appliquer une réduction *lsv*.

Sans la règle *gc*, le fait d'autoriser les substitutions dès que le corps de la substitution explicite est une réponse retire l'unicité des formes normales, car les réponses ne sont pas forcément en forme normale. Dans le cas où on substitue la dernière

$$\frac{}{x \in \mathscr{V}_x}\ \text{V-VAR} \qquad \frac{t \in \mathscr{V}_x}{t\ u \in \mathscr{V}_x}\ \text{V-APP-LEFT} \qquad \frac{u \in \mathscr{V}_x}{t\ u \in \mathscr{V}_x}\ \text{V-APP-RIGHT} \qquad \frac{t \in \mathscr{V}_x}{\lambda y.t \in \mathscr{V}_x}\ \text{V-ABS}$$

$$\frac{t \in \mathscr{V}_x}{t[y \setminus u] \in \mathscr{V}_x}\ \text{V-ES-LEFT} \qquad \frac{t \in \mathscr{V}_y \quad u \in \mathscr{V}_x}{t[y \setminus u] \in \mathscr{V}_x}\ \text{V-ES-RIGHT}$$

FIGURE 3.11 – Variable vivante

$$\begin{array}{ccc}
x[x \setminus \lambda y.(\lambda a.a)\ z] & \xrightarrow{lsv} & (\lambda y.(\lambda a.a)\ z)[x \setminus \lambda y.(\lambda a.a)\ z] \\
\downarrow dB & & \downarrow dB \\
x[x \setminus \lambda y.a[a \setminus z]] & & (\lambda y.z)[a \setminus z][x \setminus \lambda y.(\lambda a.a)\ z] \\
\downarrow lsv^* & & \downarrow gc^* \\
(\lambda y.z)[x \setminus \lambda y.z][a \setminus z] & \xrightarrow{gc^*} & \lambda y.z
\end{array}$$

FIGURE 3.12 – Problème de confluence pour le terme $x[x \setminus \lambda y.Iz]$

occurrence d'une variable par une réponse à deux endroit différant dans l'arbre de réduction, du fait qu'il n'est pas possible d'aller réduire dans le corps de la substitution explicite si il n'y a plus d'occurrences de la variable dans le terme, on obtient deux formes normales différentes, on donne un exemple dans la figure 3.12.

Définition 19 (Réponse)**.** *Les réponses présentées dans la figure 3.13 représente les abstractions sous un nombre arbitraire de substitutions explicites. Les réponses vont remplacer les valeurs.*

$$\frac{}{\lambda x.t \in \mathscr{R}}\ \text{R-ABS} \qquad \frac{t \in \mathscr{R}}{t[x \setminus u] \in \mathscr{R}}\ \text{R-ES}$$

FIGURE 3.13 – Réponse

Correction de la caractérisation

Dans cette section, nous démontrons le théorème selon lequel notre caractérisation des formes normales est cohérente avec l'absence de réduction dans un terme. Le lemme principal est le lemme 25 qui affirme qu'un terme est en forme normale si et seulement si il n'existe pas de réduction possible.

Le lemme 21 prouve qu'un terme ne peut pas être à la fois en forme normale et réductible. Le lemme 24 prouve qu'un terme est soit en forme normale, soit réductible. Le lemme 21 délègue les preuves sur les cas particuliers dB au lemme 19 et lsv au lemme 20. Ces lemmes utilisent à leur tour plusieurs lemmes annexes.

Lemme 9 (Les réponses ne sont pas des structures)**.** *Si un terme est une structure (Fig. 2.48), alors ce n'est pas une réponse (Fig. 3.13).*
$(\forall t, \varphi, \quad t \in S_\varphi \implies t \notin \mathscr{R})$

Démonstration. Par induction sur la dérivation de $t \in S_\varphi$.

- S-VAR $t = x$: Une variable n'est pas une réponse.
- S-APP $t = u\ v$ avec $u \in S_\varphi$: Une application n'est pas une réponse.
- S-ES $t = u[x \setminus v]$ avec $u \in S_\varphi$: Par hypothèse d'induction, on a $u \notin \mathscr{R}$. Or, la règle R-ES nous dit que pour que $u[x \setminus v] \in \mathscr{R}$, on doit avoir $u \in \mathscr{R}$. Donc $u[x \setminus v] \notin \mathscr{R}$
- S-ES-S $t = u[x \setminus v]$ avec $u \in S_{\varphi \cup \{x\}}$: Même raisonnement que pour S-ES. □

Lemme 10 (Réduction id_x sur une variable vivante dans un terme en forme normale)**.** *Si un terme est en forme normale (Fig. 3.10) et a une variable vivante (Fig. 3.11), alors on peut appliquer une réduction id à ce terme sur la variable vivante.*
$(\forall t, x, \quad t \in \mathscr{N} \wedge t \in \mathscr{V}_x \implies \exists \varphi, \mu, \quad t \xrightarrow{\mathsf{id}_x, \varphi, \mu}_{\mathsf{sn}} t)$

Démonstration. Par induction sur la dérivation de $t \in \mathscr{V}_x$.

- V-VAR $t = x$: Il est toujours possible d'appliquer la réduction id_x sur une variable x par la règle ID.
- V-APP-LEFT $t = u\ v$ avec $u \in \mathscr{V}_x$: On a $u\ v \in \mathscr{N}$, par définition on a $u \in \mathscr{N}$. Comme $u \in \mathscr{V}_x$ et $u \in \mathscr{N}$, par hypothèse d'induction on a $u \xrightarrow{\mathsf{id}_x, \varphi, \perp}_{\mathsf{sn}} u$. Donc par APP-LEFT, $u\ v \xrightarrow{\mathsf{id}_x, \varphi, \mu}_{\mathsf{sn}} u\ v$.
- V-APP-RIGHT $t = u\ v$ avec $v \in \mathscr{V}_x$: On a $u\ v \in \mathscr{N}$, par définition on a $v \in \mathscr{N}$ et $u \in S_\varphi$. Comme $v \in \mathscr{V}_x$ et $v \in \mathscr{N}$, par hypothèse d'induction on a $v \xrightarrow{\mathsf{id}_x, \varphi, \perp}_{\mathsf{sn}} v$. Donc par APP-RIGHT, $u\ v \xrightarrow{\mathsf{id}_x, \varphi, \mu}_{\mathsf{sn}} u\ v$.
- V-ABS $t = \lambda y.u$ avec $u \in \mathscr{V}_x$ et $x \neq y$: On a $\lambda y.u \in \mathscr{N}$, par définition on a $u \in \mathscr{N}$. Comme $u \in \mathscr{V}_x$ et $u \in \mathscr{N}$, par hypothèse d'induction on a $u \xrightarrow{\mathsf{id}_x, \varphi, \top}_{\mathsf{sn}} u$. Par le lemme 3 on a $u \xrightarrow{\mathsf{id}_x, \varphi \cup \{y\}, \top}_{\mathsf{sn}} u$ et donc par ABS-$\top$, $\lambda y.u \xrightarrow{\mathsf{id}_x, \varphi, \top}_{\mathsf{sn}} \lambda y.u$.
- V-ES-LEFT $t = u[y \setminus v]$ avec $u \in \mathscr{V}_x$: On a $u[y \setminus v] \in \mathscr{N}$, par définition on a $u \in \mathscr{N}$. Comme $u \in \mathscr{V}_x$ et $u \in \mathscr{N}$, par hypothèse d'induction on a $u \xrightarrow{\mathsf{id}_x, \varphi, \mu}_{\mathsf{sn}} u$. Donc par ES-LEFT, $u[y \setminus v] \xrightarrow{\mathsf{id}_x, \varphi, \mu}_{\mathsf{sn}} u[y \setminus v]$.
- V-ES-RIGHT $t = u[y \setminus v]$ avec $u \in \mathscr{V}_y$ et $v \in \mathscr{V}_x$: On a $u[y \setminus v] \in \mathscr{N}$, par définition on a $u \in \mathscr{N}$ et $v \in \mathscr{N}$. Comme $u \in \mathscr{V}_y$ et $u \in \mathscr{N}$, par hypothèse

d'induction on a $u \xrightarrow{\mathsf{id}_y,\varphi,\mu}_{\mathsf{sn}} u$. Comme $v \in \mathscr{V}_x$ et $v \in \mathscr{N}$, par hypothèse d'induction on a $v \xrightarrow{\mathsf{id}_x,\varphi,\perp}_{\mathsf{sn}} v$. Donc par ES-RIGHT, $u[y \setminus v] \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn}} u[y \setminus v]$. □

Lemme 11 (Réduction $\mathsf{sub}_{x\setminus v}$ sur une variable vivante dans un terme en forme normale)**.** *Si un terme est en forme normale et a une variable vivante, alors on peut appliquer une réduction $\mathsf{sub}_{x\setminus v}$ sur la variable vivante à ce terme.*
($\forall t, x, v, \quad t \in \mathscr{N} \wedge t \in \mathscr{V}_x \implies \exists t', \varphi, \mu, \quad t \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\mu}_{\mathsf{sn}} t'$)

Démonstration. La réduction ID et la réduction SUB partagent les mêmes prémisses et s'appliquent au même terme, donc si on peut effectuer une réduction ID, on peut effectuer une réduction SUB. Ainsi, la preuve est similaire à celle du lemme 10. □

Lemme 12 (Les variables ciblées dans les réductions id_x sont vivantes)**.** *Toutes les variables ciblées par une réduction id_x dans un terme t sont vivantes dans ce même terme.*
($\forall t, x, \varphi, \mu, \quad t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn}} t \implies t \in \mathscr{V}_x$).

Démonstration. Par induction sur la dérivation de la réduction $t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn}} t$.

Pour toutes les réductions de la figure 3.3, la preuve est résolue trivialement par hypothèse d'induction, à l'exception de la règle ES-RIGHT, qui utilise la réduction ID.

Cas ES-RIGHT : $t = u[z \setminus w]$ avec $u \xrightarrow{\mathsf{id}_z,\varphi,\mu}_{\mathsf{sn}} u$ et $w \xrightarrow{\mathsf{id}_x,\varphi,\perp}_{\mathsf{sn}} w$. Par hypothèse d'induction on a $u \in \mathscr{V}_z$ et $w \in \mathscr{V}_x$. Puis, par la règle V-ES-RIGHT on a $u[z\setminus w] \in \mathscr{V}_x$.

Le seul cas où notre lemme s'applique dans la figure 3.4 est le cas ID, qui est le cas de base, car il suffit de prouver que $x \in \mathscr{V}_x$, ce qui est le cas de base de la définition de $\mathscr{V}_x$. □

Lemme 13 (Les variables ciblées dans les réductions $\mathsf{sub}_{x\setminus v}$ sont vivantes)**.** *Toutes les variables ciblées par une réduction $\mathsf{sub}_{x\setminus v}$ dans un terme t sont vivantes dans ce même terme.*
($\forall t, t', x, v, \varphi, \mu, \quad t \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\mu}_{\mathsf{sn}} t' \implies t \in \mathscr{V}_x$).

Démonstration. Par induction sur la dérivation de la réduction $t \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\mu}_{\mathsf{sn}} t'$.

Pour toutes les réductions de la figure 3.3, la preuve est résolue trivialement par hypothèse d'induction, à l'exception de la règle ES-RIGHT, qui utilise la réduction ID.

Cas ES-RIGHT : $t = u[z \setminus w]$ avec $u \xrightarrow{\mathsf{id}_z,\varphi,\mu}_{\mathsf{sn}} u$ et $w \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\perp}_{\mathsf{sn}} w$. Avec le lemme 12 sur $u \xrightarrow{\mathsf{id}_z,\varphi,\mu}_{\mathsf{sn}} u$ on a $u \in \mathscr{V}_z$. Par hypothèse d'induction on a $w \in \mathscr{V}_x$. Puis, par la règle V-ES-RIGHT on a $u[z \setminus w] \in \mathscr{V}_x$.

Le seul cas où notre lemme s'applique dans la figure 3.4 est le cas SUB, qui est le cas de base : il suffit de prouver que $x \in \mathscr{V}_x$, ce qui est le cas de base de la définition de $\mathscr{V}_x$. □

Lemme 14 (Réductible $\mathsf{sub}_{x\setminus v}$ implique réductible id_x)**.** *Si l'on peut appliquer une réduction $\mathsf{sub}_{x\setminus v}$ a un terme alors on peut également lui appliquer une réduction id_x.*
($\forall t, x, v, \varphi, \mu, \quad t \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\mu}_{\mathsf{sn}} t' \implies \forall v, \exists t', \quad t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn}} t'$).

Démonstration. Triviale par induction sur la dérivation de $t \xrightarrow{\mathsf{sub}_{x\backslash v},\varphi,\mu}_{\mathsf{sn}} t'$. □

Lemme 15 (Réductible id_x implique réductible $\mathsf{sub}_{x\backslash v}$)**.** *Si l'on peut appliquer une réduction* id_x *a un terme alors on peut également lui appliquer une réduction* $\mathsf{sub}_{x\backslash v}$*.*
$(\forall t, x, \varphi, \mu, t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn}} t \implies \forall v, t \xrightarrow{\mathsf{sub}_{x\backslash v},\varphi,\mu}_{\mathsf{sn}} t')$.

Démonstration. Triviale par induction sur la dérivation de $t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn}} t$. □

Notons que le lemme 12 peut se déduire des deux lemmes 13 et 15.

Lemme 16 (Réductible $\top$ implique réductible $\bot$ ou $\mathscr{R}$)**.** *Si un terme est réductible en mode $\top$, alors soit il est aussi réductible en mode $\bot$, soit c'est une réponse.*
$(\forall t, t', \rho, \varphi, \quad t \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} t' \implies t \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} t' \vee t \in \mathscr{R})$.

Démonstration. Par induction sur la dérivation de la réduction $t \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} t'$.

- APP-LEFT : $t = u\ v$ avec $u\ v \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'\ v$ et avec $u \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u'$. Donc, par la règle APP-LEFT on a $u\ v \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u'\ v$.
- APP-RIGHT : Exactement le même raisonnement que pour APP-LEFT.
- ABS-$\top$: $t = \lambda x.u$: Le terme est une réponse.
- ES-LEFT : $t = u[x \setminus v]$ avec $u[x \setminus v] \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'[x \setminus v]$ et avec $u \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'$. On applique l'hypothèse d'induction sur $u \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'$:
  - Si $u \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u'$ alors, on a $u[x\setminus v] \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u'[x\setminus v]$ par la règle ES-LEFT.
  - Si $u \in \mathscr{R}$ alors par la règle R-ES (Fig. 3.13), $u[x \setminus v] \in \mathscr{R}$.
- ES-LEFT-FROZEN : Exactement le même raisonnement que pour ES-LEFT.
- ES-RIGHT : $t = u[x \setminus v]$ avec $u[x \setminus v] \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u[x \setminus v']$ et avec $u \xrightarrow{\mathsf{id}_x,\varphi,\top}_{\mathsf{sn}} u$ et $v \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} v'$.
  On applique l'hypothèse d'induction sur $u \xrightarrow{\mathsf{id}_x,\varphi,\top}_{\mathsf{sn}} u$:
  - Si $u \xrightarrow{\mathsf{id}_x,\varphi,\bot}_{\mathsf{sn}} u$, alors on a $u[x \setminus v] \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u[x \setminus v']$ par la règle ES-RIGHT.
  - Si $u \in \mathscr{R}$ alors par la règle R-ES (Fig. 3.13), $u[x \setminus v] \in \mathscr{R}$. □

Lemme 17 (Structure et variable vivante)**.** *Si un terme est une structure pour un ensemble, il est possible de retirer de l'ensemble n'importe quelle variable qui n'est pas vivante dans le terme.*
$(\forall t, \varphi, x, \quad t \in S_\varphi \wedge t \notin \mathscr{V}_x \implies t \in S_{\varphi\setminus\{x\}})$

Démonstration. Par induction sur t et Par définition de $t \in S_\varphi$.

- S-VAR : t ne peut pas valoir x car $x \in \mathscr{V}_x$. Donc dans le cas où $t = y$, on a $y \in \varphi$ et aussi $y \in \varphi \setminus \{x\}$. Donc $y \in S_{\varphi\setminus\{x\}}$.
- S-APP : $t = u\ v$ avec $u \in S_\varphi$. Par contraposée de $u\ v \notin \mathscr{V}_x$ on a $u \notin \mathscr{V}_x$. Par hypothèse d'induction on a $u \in S_{\varphi\setminus\{x\}}$. Donc $u\ v \in S_{\varphi\setminus\{x\}}$.
- S-ES : $t = u[y \setminus v]$ avec $u \in S_\varphi$. Par contraposée de $u[y \setminus v] \notin \mathscr{V}_x$ on a dans les deux cas $u \notin \mathscr{V}_x$ et soit $u \notin \mathscr{V}_y$ soit $v \notin \mathscr{V}_x$. $(u \notin \mathscr{V}_x \wedge (u \notin \mathscr{V}_y \vee v \notin \mathscr{V}_x))$. Dans les deux cas, par hypothèse d'induction on a $u \in S_{\varphi\setminus\{x\}}$. Donc $u[y\setminus v] \in S_{\varphi\setminus\{x\}}$.

- S-ES-S $t = u[y \backslash v]$ avec $u \in S_{\varphi \cup \{y\}}$ et $v \in S_\varphi$. Par contraposée de $u[y \backslash v] \notin \mathscr{V}_x$ on a dans les deux cas $u \notin \mathscr{V}_x$ et soit $u \notin \mathscr{V}_y$ soit $v \notin \mathscr{V}_x$. $(u \notin \mathscr{V}_x \wedge (u \notin \mathscr{V}_y \vee v \notin \mathscr{V}_x))$ ce qu'on peut transformer en $(u \notin \mathscr{V}_x \wedge u \notin \mathscr{V}_y) \vee (u \notin \mathscr{V}_x \wedge v \notin \mathscr{V}_x)$ sur quoi on fait une disjonction de cas :
  - $u \notin \mathscr{V}_x \wedge u \notin \mathscr{V}_y$: Par hypothèse d'induction sur $u \in S_{\varphi \cup \{y\}}$ on a $u \in S_{\varphi \cup \{y\} \backslash \{x\}}$. Puis par hypothèse d'induction sur $u \in S_{\varphi \cup \{y\} \backslash \{x\}}$ on a $u \in S_{\varphi \backslash \{x\}}$ donc $u[y \setminus v] \in S_{\varphi \backslash \{x\}}$ par la règle S-ES.
  - $u \notin \mathscr{V}_x \wedge v \notin \mathscr{V}_x$: Par hypothèse d'induction sur $u \in S_{\varphi \cup \{y\}}$ on a $u \in S_{\varphi \cup \{y\} \backslash \{x\}}$. Puis par hypothèse d'induction sur $v \in S_\varphi$ on a $v \in S_{\varphi \backslash \{x\}}$ donc $u[y \setminus v] \in S_{\varphi \backslash \{x\}}$ par la règle S-ES-S. □

Lemme 18 (Forme d'un terme en forme normale)**.** *Si un terme est en forme normale, alors soit c'est une structure, soit c'est une réponse.*
$(\forall t, \quad t \in \mathscr{N} \implies (\exists \varphi, t \in S_\varphi) \vee t \in \mathscr{R})$

Démonstration. Par induction sur le terme t.
- Var $t = x$: Une variable est une structure avec l'ensemble φ qui contient cette variable, par la règle S-VAR (Fig. 2.48).
- App $t = u\ v$: On a $u\ v \in \mathscr{N}$, et par définition, on a $u \in S_\varphi$. Par conséquent, d'après S-APP (Fig. 2.48), $u\ v \in S_\varphi$.
- Abs $t = \lambda x.u$: $\lambda x.u \in \mathscr{R}$ par la règle R-ABS (Fig. 3.13).
- ES $t = u[x \backslash v]$: On a $u[x \backslash v] \in \mathscr{N}$, et par définition il y a deux règles (NF-ES et NF-ES-S) mais dans les deux cas : $u \in \mathscr{N}$. Par hypothèse d'induction sur $u \in \mathscr{N}$ on a :
  - Si $u \in \mathscr{R}$, alors par la règle R-ES (Fig. 3.13), $u[x \setminus v] \in \mathscr{R}$.
  - Si $u \in S_\varphi$ on va traiter les deux règle différemment :
    - Si $u[x \backslash v] \in \mathscr{N}$ par la règle NF-ES alors on a $u \notin \mathscr{V}_x$. Par le lemme 17 on a $u \in S_{\varphi \backslash \{x\}}$. Et donc $u[x \setminus v] \in S_{\varphi \backslash \{x\}}$ (règle S-ES).
    - Si $u[x \setminus v] \in \mathscr{N}$ par la règle NF-ES-S alors on a $v \in \mathscr{N}$ et $v \in S_\varphi$. Par le lemme 2 sur $u \in S_\varphi$ on a $u \in S_{\varphi \cup \{x\}}$. Et donc $u[x \setminus v] \in S_\varphi$ (règle S-ES-S). □

Lemme 19 (Forme normale dB)**.** *Un terme ne peut pas à la fois être réductible par une réduction dB et être en forme normale.*
$(\forall t, t', \quad t \rightarrow_{\mathsf{db}} t' \wedge t \in \mathscr{N} \implies \bot)$

Démonstration. Par induction sur la dérivation de la réduction $t \rightarrow_{\mathsf{db}} t'$ (Fig. 3.5).
- dB-BASE $t = (\lambda x.u)\ v$: Par définition de $(\lambda x.u)\ v \in \mathscr{N}$ on a $(\lambda x.u) \in S_\varphi$ (règle NF-APP Fig. 3.10). Cependant, le lemme 9 nous indique qu'un terme ne peut pas être à la fois une structure et une abstraction. Par conséquent, cela constitue une contradiction.
- dB-σ $t = u[x \backslash w]\ v$ avec $u[x \backslash w]\ v \rightarrow_{\mathsf{db}} z[x \backslash w]$ et $u\ v \rightarrow_{\mathsf{db}} z$ comme prémisses : Par définition de $u[x \setminus w]\ v \in \mathscr{N}$, on a $u[x \setminus w] \in \mathscr{N}$, $u[x \setminus w] \in S_\varphi$, $v \in \mathscr{N}$ (règle NF-APP Fig. 3.10) et Par définition de $u[x \setminus w] \in \mathscr{N}$ on a $u \in \mathscr{N}$ (règles NF-ES et NF-ES-S Fig. 3.10). Puis, Par définition de $u[x \setminus w] \in S_\varphi$, on a $u \in S_\varphi$ ou $u \in S_{\varphi \cup \{x\}}$ (règles S-ES et S-ES-S Fig. 2.48). Donc avec

$u \in \mathscr{N}$, $v \in \mathscr{N}$ et $u \in S_\varphi$ ou $u \in S_{\varphi\cup\{x\}}$, on a $u\ v \in \mathscr{N}$ (règle NF-APP Fig. 3.10). Et donc, par hypothèse d'induction, on a une contradiction. □

Lemme 20 (Forme normale lsv). *Un terme ne peut pas être réduit avec une réduction lsv et être en forme normale.*
$(\forall t, t', \varphi, \mu, \quad t \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t' \wedge t \in \mathscr{N} \implies \bot)$.

Démonstration. Par induction sur la dérivation de la réduction $t \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'$ (Fig. 3.5) .

- LSV-BASE $t = u[x \setminus v]$ avec $u[x \setminus v] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} u'[x \setminus v]$ et avec $u \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\mu}_{\mathsf{sn}} u'$ et $v \in \mathit{Val}$ comme prémisses : Par définition de $u[x \setminus v] \in \mathscr{N}$ on a : (Fig. 3.10)
  - NF-ES $u \notin \mathscr{V}_x$: Or, d'après le lemme 13, si une variable n'est pas en vie dans un terme, alors on ne peut pas appliquer la réduction SUB sur cette même variable. Donc, on a une contradiction.
  - NF-ES-S $v \in S_\varphi$: Or, d'après le lemme 9, si un terme est une structure, il ne peut pas être une réponse. Donc, on a une contradiction.
- LSV-σ et LSV-σ-φ $t = u[x \setminus v[y \setminus w]]$ avec $u[x \setminus v[y \setminus w]] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} u'[y \setminus w]$ et avec $u[x \setminus v] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} u'$ comme prémisse pour la règle LSV-σ et $u[x \setminus v] \xrightarrow{\varphi\cup\{x\},\mu}_{\mathsf{lsv}} u'$ comme prémisse pour la règle LSV-σ-φ. Le raisonnement est le même pour les deux règles donc nous ne ferons que la règle LSV-σ : Par définition de $u[x \setminus v[y \setminus w]] \in \mathscr{N}$ on a :
  - NF-ES $u \notin \mathscr{V}_x$ et $u \in \mathscr{N}$: Donc par la règle NF-ES $u[x \setminus v] \in \mathscr{N}$. Et donc, par hypothèse d'induction, on a une contradiction.
  - NF-ES-S $u \in \mathscr{N}$, $v[y \setminus w] \in \mathscr{N}$ et $v[y \setminus w] \in S_\varphi$: Puis Par définition de $v[y \setminus w] \in \mathscr{N}$ on a $v \in \mathscr{N}$ (règles NF-ES et NF-ES-S Fig. 3.10). Et Par définition de $v[y \setminus w] \in S_\varphi$ on a $v \in S_\varphi$ ou $v \in S_{\varphi\cup\{y\}}$ (S-ES et S-ES-S). Donc avec $u \in \mathscr{N}$, $v \in \mathscr{N}$ et $v \in S_\varphi$ ou $v \in S_{\varphi\cup\{y\}}$ on a $u[x \setminus v] \in \mathscr{N}$. Et donc, par hypothèse d'induction, on a une contradiction. □

Lemme 21 (Un terme ne peut pas être en forme normale et réductible). *Un terme ne peut pas être réduit avec une réduction lsv ou dB et être en forme normale.*
$(\forall t, t', \varphi, \rho \in \{dB, lsv\}, \quad t \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} t' \wedge t \in \mathscr{N} \implies \bot)$.

Démonstration. Par induction sur t. Pour chaque cas, nous allons devoir effectuer un raisonnement par cas sur les règles de réduction. Donc pour avoir plus de lisibilité nous traitons directement les règles de réduction. (pour les règles de réduction voir les figures 3.3, 3.4 et 3.5).

Les règles de la figure 3.4 qui sont les quatre règles de réduction possibles : ID, SUB, dB et lsv. Notre lemme ne traite pas les deux règles ID et SUB. Les règles dB et lsv correspondent respectivement aux réductions $\rightarrow_{\mathsf{db}}$ et $\xrightarrow{\varphi,\mu}_{\mathsf{lsv}}$, qui sont traitées dans les lemmes 19 et 20.

Les règles de la figure 3.3 sont des règles de propagation et sont donc traitées principalement par hypothèse d'induction. Le seul cas qui est un peu plus compliqué est celui de la règle ES-RIGHT.

- APP-LEFT $t = u\ v$ avec $u\ v \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'\ v$ et avec $u \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u'$ comme prémisse : Par le lemme 1, on a $u \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'$. Puis, Par définition de $u\ v \in \mathscr{N}$ on a $u \in \mathscr{N}$ (règle NF-APP Fig. 3.10). Et donc, par hypothèse d'induction, on a une contradiction.

— APP-RIGHT $t = u\ v$ avec $u\ v \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u\ v'$ et avec $v \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} v'$ comme prémisse : Par définition de $u\ v \in \mathscr{N}$ on a $v \in \mathscr{N}$ (règle NF-APP Fig. 3.10). Et donc, par hypothèse d'induction, on a une contradiction.

— ABS-$\top$ $t = \lambda x.u$ avec $\lambda x.u \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} \lambda x.u'$ et avec $u \xrightarrow{\rho,\varphi\cup\{x\},\top}_{\mathsf{sn}} u'$ comme prémisse : Par définition de $\lambda x.u \in \mathscr{N}$ on a $u \in \mathscr{N}$ (règle NF-ABS Fig. 3.10). Et donc, par hypothèse d'induction, on a une contradiction.

— ES-LEFT $t = u[x \setminus v]$ avec $u[x \setminus v] \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'[x \setminus v]$ et avec $u \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'$ comme prémisse : Par définition de $u[x \setminus v] \in \mathscr{N}$ on a $u \in \mathscr{N}$ (règles NF-ES et NF-ES-S Fig. 3.10). Et donc, par hypothèse d'induction, on a une contradiction.

— ES-LEFT-FROZEN $t = u[x \setminus v]$ avec $u[x \setminus v] \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'[x \setminus v]$ et avec $u \xrightarrow{\rho,\varphi\cup\{x\},\top}_{\mathsf{sn}} u'$ comme prémisse : Par définition de $u[x\setminus v] \in \mathscr{N}$ on a $u \in \mathscr{N}$ (règles NF-ES et NF-ES-S Fig. 3.10). Et donc, par hypothèse d'induction, on a une contradiction.

— ES-RIGHT $t = u[x \setminus v]$ avec $u[x \setminus v] \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u[x \setminus v']$ et avec $u \xrightarrow{\mathsf{id}_x,\varphi,\top}_{\mathsf{sn}} u$ et $v \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} v'$ comme prémisses : Par définition de $u[x \setminus v] \in \mathscr{N}$ on a :
  — NF-ES $u \notin \mathscr{V}_x$: Or, d'après le lemme 12, si une variable n'est pas en vie dans un terme, alors on ne peut pas appliquer la réduction ID sur cette même variable. Donc, on a une contradiction.
  — NF-ES-S $v \in \mathscr{N}$: Par le lemme 1, on a $v \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} v'$. Et donc, par hypothèse d'induction, on a une contradiction. □

Lemme 22 (Une réponse à gauche d'une application est réductible)**.** *Si le sous-terme gauche d'une application est une réponse, alors l'application est réductible avec la règle de réduction dB.*
$(\forall t, u, t \in \mathscr{R} \implies \exists t', t\ u \to_{db} t')$

Démonstration. Par induction sur la dérivation de $t \in \mathscr{R}$
— Cas R-ABS : on a $t = \lambda x.r$ alors par la règle DB-BASE, $(\lambda x.r)\ u \to_{\mathsf{db}} r[x \setminus u]$
— Cas R-ES : on a $t = r[x \setminus w]$ avec $r \in \mathscr{R}$: on applique l'hypothèse d'induction sur r. Donc $r\ u \to_{\mathsf{db}} r'$ et donc par la règle DB-σ $(r[x \setminus w])\ u \to_{\mathsf{db}} r'[x \setminus w]$ □

Lemme 23 (Réponse réductible *lsv*)**.** *Si un terme qui a une variable atteignable est sous une substitution explicite qui a pour corps un terme qui est en forme normale et qui est une réponse, alors le terme global est réductible avec la règle de réduction lsv.*
$(\forall t, u, x, \varphi, \mu, \quad t \xrightarrow{id_x,\varphi,\mu}_{sn} t \wedge u \in \mathscr{N} \wedge u \in \mathscr{R} \implies \exists t', \quad t[x \setminus u] \xrightarrow{\varphi,\mu}_{lsv} t')$

Démonstration. Par induction sur la dérivation de $u \in \mathscr{R}$: Dans les deux cas a traiter on a besoin de $\mathsf{sub}_{x\setminus v}$ à la place de id_x. On utilise le lemme 15.
— Cas R-ABS : on a $u = \lambda y.r$ alors par la règle LSV-BASE, $t[x \setminus \lambda y.r] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'[x \setminus \lambda y.r]$.
— Cas R-ES : on a $u = r[y \setminus w]$ avec $r \in \mathscr{R}$. Par définition de $r[y \setminus w] \in \mathscr{N}$ on a $r \in \mathscr{N}$. On applique l'hypothèse d'induction sur $r \in \mathscr{R}$. Donc $t[x \setminus r] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'$ et donc par la règle LSV-σ on a, $t[x \setminus r[y \setminus w]] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'[y \setminus w]$. □

Lemme 24 (Un terme est en forme normale ou réductible)**.** *Un terme est soit en forme normale soit réductible avec une réduction lsv ou dB.*
$(\forall t, \exists \varphi, \rho \in \{dB, lsv\}, t', \quad t \xrightarrow{\rho,\varphi,\top}_{sn} t' \vee t \in \mathscr{N})$

Démonstration. Par induction sur le terme t.

Pour ce lemme, le raisonnement est un peu particulier. Lorsque l'on utilise l'hypothèse d'induction, on obtient que le sous-terme est soit en forme normale, soit réductible. Il faut donc raisonner sur les deux cas. Ainsi, s'il y a un sous-terme, il y a au moins deux cas, mais s'il y en a deux, comme c'est le cas pour l'application par exemple, il y a au minimum quatre cas.

1. Var $t = x$: $x \in \mathscr{N}$ (règle VAR Fig. 3.10).
2. App $t = u\ v$: Par hypothèse d'induction sur u on a deux possibilités
   — $u \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'$: Par le lemme 16 on a deux possibilités
      — $u \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} u'$: Donc avec la règle APP-LEFT on a $u\ v \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'\ v$.
      — $u \in \mathscr{R}$: Par le lemme 22 on a $u\ v \xrightarrow{dB,\varphi,\mu}_{\mathsf{sn}} w$
   — $u \in \mathscr{N}$: Par le lemme 18 on a deux possibilités
      — $u \in \mathscr{R}$: Par le lemme 22 on a $u\ v \xrightarrow{dB,\varphi,\mu}_{\mathsf{sn}} w$
      — $u \in S_\varphi$: Par hypothèse d'induction sur v on a deux possibilités
         — $v \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} v'$: Alors , $u\ v \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} u\ v'$ (règle APP-RIGHT Fig. 3.3)
         — $v \in \mathscr{N}$: Alors, $u\ v \in \mathscr{N}$ (règle NF-APP Fig. 3.10)
3. Abs $t = \lambda x.u$: Par hypothèse d'induction sur u on a deux possibilités
   — $u \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'$: Par le lemme 3 on a $u \xrightarrow{\rho,\varphi\cup\{x\},\top}_{\mathsf{sn}} u'$. Donc on a $\lambda x.u \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} \lambda x.u'$ (règle ABS-$\top$ Fig. 3.3).
   — $u \in \mathscr{N}$: Alors, $\lambda x.u \in \mathscr{N}$ (règle ABS Fig. 3.10).
4. ES $t = u[x \setminus v]$: Par hypothèse d'induction sur u on a deux possibilités
   — $u \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'$: Alors, par la règle ES-LEFT, on a $u[x \setminus v] \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'[x \setminus v]$.
   — $u \in \mathscr{N}$: Alors, nous faisons une disjonction de cas sur $u \in \mathscr{V}_x$:
      — $u \notin \mathscr{V}_x$: Alors, $u[x \setminus v] \in \mathscr{N}$ (règle NF-ES Fig. 3.10).
      — $u \in \mathscr{V}_x$: Par hypothèse d'induction sur v on a deux possibilités
         — $v \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} v'$: D'après le lemme 16 on a deux possibilités
            — $v \xrightarrow{\rho,\varphi,\bot}_{\mathsf{sn}} v'$: Avec, le lemme 10 on a $u \xrightarrow{\mathsf{id}_x,\varphi,\top}_{\mathsf{sn}} u$. Donc, par la règle ES-RIGHT, on a $u[x \setminus v] \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u[x \setminus v']$.
            — $v \in \mathscr{R}$: Avec, le lemme 10 on a $u \xrightarrow{\mathsf{id}_x,\varphi,\top}_{\mathsf{sn}} u$. Alors avec le lemme 23 on a $u[x \setminus v] \xrightarrow{\varphi,\top}_{\mathsf{lsv}} u'[x \setminus v]$. Et donc, on a $u[x \setminus v] \xrightarrow{lsv,\varphi,\top}_{\mathsf{sn}} u'[x \setminus v]$
         — $v \in \mathscr{N}$: D'après le lemme 18 on a deux possibilités
            — $v \in S_\varphi$: Alors, $u[x \setminus v] \in \mathscr{N}$ (règle NF-ES-S Fig. 3.10).
            — $v \in \mathscr{R}$: Cas déjà traité. □

Lemme 25 (Forme normale)**.** *Un terme est réductible si et seulement si il n'est pas en forme normale.* $(\forall t, t \notin \mathscr{N} \iff \exists \varphi, \rho \in \{dB, lsv\}, t', t \xrightarrow{\rho,\varphi,\top}_{sn} t')$

Démonstration. Pour prouver qu'un terme est en forme normale seulement s'il n'est pas réductible, on commence par prouver, dans le lemme 21, qu'un terme ne peut pas être en forme normale et réductible. Puis on prouve, dans le lemme 24, qu'un terme est en forme normale ou réductible. □

3.2.3 Dépliage des formes normales

Lemme 26 (Une variable qui n'est pas vivante n'est pas libre)**.** *Si une variable n'est pas vivante dans un terme, alors elle n'est pas non plus libre.*
$(t \notin \mathscr{V}_x \implies x \notin \mathrm{FV}(t^\star))$

Démonstration. On prouve la contraposée : $x \in \mathrm{FV}(t^\star) \implies t \in \mathscr{V}_x$ Par induction sur t.

- $t = x$: On a $x \in \mathscr{V}_x$ (V-VAR).
- $t = u\ v$: On a $\mathrm{FV}((u\ v)^\star) = \mathrm{FV}(u^\star\ v^\star) = x \in \mathrm{FV}(u^\star) \cup \mathrm{FV}(v^\star)$. Donc $x \in \mathrm{FV}(u^\star) \vee \mathrm{FV}(v^\star)$. Par hypothèse d'induction sur u et v on a :
  - Si $x \in \mathrm{FV}(u^\star)$ alors $u \in \mathscr{V}_x$: On a $(u\ v) \in \mathscr{V}_x$ par la règle V-APP-LEFT.
  - Si $x \in \mathrm{FV}(v^\star)$ alors $v \in \mathscr{V}_x$: On a $(u\ v) \in \mathscr{V}_x$ par la règle V-APP-RIGHT.
- $t = \lambda y.u$: On a $\mathrm{FV}((\lambda y.u)^\star) = \mathrm{FV}(u^\star) \backslash \{y\}$. Donc $x \in \mathrm{FV}(u^\star)$. Par hypothèse d'induction sur u on a $u \in \mathscr{V}_x$ Et donc on a $\lambda y.u \in \mathscr{V}_x$ par la règle V-ABS.
- $t = u[y \setminus v]$: On a $\mathrm{FV}((u[y \setminus v])^\star) = \mathrm{FV}(u^\star\{y \setminus v^\star\})$ On fait une disjonction de cas sur le fait que $y \in \mathrm{FV}(u^\star)$:
  - Si $y \in \mathrm{FV}(u^\star)$: Par la proposition 5, on a $\mathrm{FV}(u^\star\{y \setminus v^\star\}) = \mathrm{FV}(u^\star) \setminus \{y\} \cup \mathrm{FV}(v^\star)$. Donc $x \in \mathrm{FV}(u^\star) \cup \mathrm{FV}(v^\star)$.
    - Si $x \in \mathrm{FV}(u^\star)$ alors par hypothèse d'induction sur u avec x on a $v \in \mathscr{V}_x$. Donc $u[y \setminus v] \in \mathscr{V}_x$ par la règle V-ES-LEFT.
    - Si $x \in \mathrm{FV}(v^\star)$ alors par hypothèse d'induction sur v avec x et sur u avec y on a $v \in \mathscr{V}_x$ et $y \in \mathscr{V}_y$. Donc $u[y \setminus v] \in \mathscr{V}_x$ par la règle V-ES-RIGHT.
  - Si $y \notin \mathrm{FV}(u^\star)$: Par la proposition 4, on a $u^\star\{y \setminus v^\star\} = u^\star$. Donc $x \in \mathrm{FV}(u^\star)$. Par hypothèse d'induction on a $u \in \mathscr{V}_x$. Donc $u[y \setminus v] \in \mathscr{V}_x$ par la règle V-ES-LEFT. □

Lemme 27 (Structure et substitution)**.** *Si t est une structure et que u aussi alors $t\{x \setminus u\}$ est une structure.*
$\forall \varphi, t, x, u, \quad t \in S_\varphi \wedge u \in S_\varphi \implies t\{x \setminus u\} \in S_\varphi.$

Démonstration. Par induction sur la dérivation de $t \in S_\varphi$.

- S-VAR $t = y$ avec $x \neq y$: On a $y\{x \setminus u\} = y$ et $y \in S_\varphi$.
- S-VAR $t = x$: On a $x\{x \setminus u\} = u$ et $u \in S_\varphi$.
- S-APP $t = v\ w$ avec $v \in S_\varphi$: On a $(v\ w)\{x \setminus u\} = v\{x \setminus u\}\ w\{x \setminus u\}$. Par hypothèse d'induction sur $v \in S_\varphi$ on a $v\{x \setminus u\} \in S_\varphi$. Donc par la règle S-APP, $v\{x \setminus u\}\ w\{x \setminus u\} \in S_\varphi$ et donc $(v\ w)\{x \setminus u\} \in S_\varphi$.
- S-ES $t = v[y \setminus w]$ avec $v \in S_\varphi$: On a $(v[y \setminus w])\{x \setminus u\} = v\{x \setminus u\}[y \setminus w\{x \setminus u\}]$. Par hypothèse d'induction sur $v \in S_\varphi$ on a $v\{x \setminus u\} \in S_\varphi$. Donc par la règle S-ES, $v\{x \setminus u\}[y \setminus w\{x \setminus u\}] \in S_\varphi$ et donc $(v[y \setminus w])\{x \setminus u\} \in S_\varphi$.

— S-ES-S $t = v[y \setminus w]$ avec $v \in S_{\varphi\cup\{y\}}$ et $w \in S_\varphi$: On a $(v[y \setminus w])\{x \setminus u\} = v\{x\setminus u\}[y\setminus w\{x\setminus u\}]$. Par hypothèse d'induction sur $v \in S_{\varphi\cup\{y\}}$ on a $v\{x\setminus u\} \in S_{\varphi\cup\{y\}}$. Par hypothèse d'induction sur $w \in S_\varphi$ on a $w\{x \setminus u\} \in S_\varphi$. Donc par la règle S-ES-S, $v\{x \setminus u\}[y \setminus w\{x \setminus u\}] \in S_\varphi$ et donc $(v[y \setminus w])\{x \setminus u\} \in S_\varphi$. □

Lemme 28 (Substitution sur une structure sans cible)**.** *Si on substitue, dans une structure, une variable qui n'est pas dans l'ensemble des variables gelées, elle reste une structure.* $(\forall\varphi, x, t, u, \quad t \in S_{\varphi\setminus\{x\}} \implies t\{x \setminus u\} \in S_{\varphi\setminus\{x\}})$

Démonstration. Par induction sur la dérivation de $t \in S_\varphi$.

— S-VAR $t = y$: $y \in S_{\varphi\setminus\{x\}}$. Si $x = y$ il y a une contradiction et si $x \neq y$ alors $y\{x \setminus u\} = y$ donc $y\{x \setminus u\} \in S_{\varphi\setminus\{x\}}$.

— S-APP $t = v\ w$: $(v\ w) \in S_{\varphi\setminus\{x\}}$ avec $v \in S_{\varphi\setminus\{x\}}$. Par hypothèse d'induction sur $v \in S_{\varphi\setminus\{x\}}$ on a $v\{x \setminus u\} \in S_{\varphi\setminus\{x\}}$. Et par définition on a $(v\ w)\{x \setminus u\} = v\{x \setminus u\}\ w\{x \setminus u\}$. Donc par la règle S-APP on a $(v\ w)\{x \setminus u\} \in S_{\varphi\setminus\{x\}}$.

— S-ES $t = v[y\setminus w]$: $v[y\setminus w] \in S_{\varphi\setminus\{x\}}$ avec $v \in S_{\varphi\setminus\{x\}}$. Par hypothèse d'induction sur $v \in S_{\varphi\setminus\{x\}}$ on a $v\{y\setminus u\} \in S_{\varphi\setminus\{x\}}$. Et par définition on a $(v[y\setminus w])\{x\setminus u\} = v\{x \setminus u\}[y \setminus w\{x \setminus u\}]$ Donc par la règle S-ES on a $(v[y \setminus w])\{x \setminus u\} \in S_{\varphi\setminus\{x\}}$.

— S-ES-S $t = v[y \setminus w]$: $v[y \setminus w] \in S_{\varphi\setminus\{x\}}$ avec $v \in S_{\varphi\cup\{y\}\setminus\{x\}}$ et $w \in S_{\varphi\setminus\{x\}}$. Par hypothèse d'induction sur $v \in S_{\varphi\cup\{y\}\setminus\{x\}}$ on a $v\{x \setminus u\} \in S_{\varphi\cup\{y\}\setminus\{x\}}$ et par hypothèse d'induction sur $w \in S_{\varphi\setminus\{x\}}$ on a $w\{x \setminus u\} \in S_{\varphi\setminus\{x\}}$. Et par définition on a $(v[y \setminus w])\{x \setminus u\} = v\{x \setminus u\}[y \setminus w\{x \setminus u\}]$. Donc par la règle S-ES-S on a $(v[y \setminus w])\{x \setminus u\} \in S_{\varphi\setminus\{x\}}$. □

Lemme 29 (Structure et substitution bis)**.**
$\forall\varphi, t, x, u, \quad t \in S_{\varphi\cup\{x\}} \wedge u \in S_\varphi \implies t\{x \setminus u\} \in S_\varphi$

Démonstration. Par induction sur la dérivation de $t \in S_{\varphi\cup\{x\}}$.

— S-VAR $t = y$ avec $x \neq y$: On a $y\{x \setminus u\} = y$ et $y \in S_\varphi$.

— S-VAR $t = x$: On a $x\{x \setminus u\} = u$ et $u \in S_\varphi$

— S-APP $t = v\ w$ avec $v \in S_{\varphi\cup\{x\}}$: On a $(v\ w)\{x \setminus u\} = v\{x \setminus u\}\ w\{x \setminus u\}$. Par hypothèse d'induction sur $v \in S_{\varphi\cup\{x\}}$ on a $v\{x \setminus u\} \in S_\varphi$. Donc par la règle S-APP, $v\{x \setminus u\}\ w\{x \setminus u\} \in S_\varphi$ et donc $(v\ w)\{x \setminus u\} \in S_\varphi$.

— S-ES $t = v[y\setminus w]$ avec $v \in S_{\varphi\cup\{x\}}$: On a $(v[y\setminus w])\{x\setminus u\} = v\{x\setminus u\}[y\setminus w\{x\setminus u\}]$. Par hypothèse d'induction sur $v \in S_{\varphi\cup\{x\}}$ on a $v\{x \setminus u\} \in S_\varphi$. Donc par la règle S-ES, $v\{x \setminus u\}[y \setminus w\{x \setminus u\}] \in S_\varphi$ et donc $(v[y \setminus w])\{x \setminus u\} \in S_\varphi$.

— S-ES-S $t = v[y \setminus w]$ avec $v \in \mathcal{S}_{\varphi\cup\{x,y\}}$ et $w \in S_{\varphi\cup\{x\}}$: On a $(v[y \setminus w])\{x \setminus u\} = v\{x \setminus u\}[y \setminus w\{x \setminus u\}]$. Par hypothèse d'induction sur $v \in \mathcal{S}_{\varphi\cup\{x,y\}}$ on a $v\{x \setminus u\} \in S_{\varphi\cup\{y\}}$. Par hypothèse d'induction sur $w \in S_{\varphi\cup\{x\}}$ on a $w\{x \setminus u\} \in S_\varphi$. Donc par la règle S-ES-S, $v\{x \setminus u\}[y \setminus w\{x \setminus u\}] \in S_\varphi$ et donc $(v[y \setminus w])\{x \setminus u\} \in S_\varphi$. □

Lemme 30 (Persistance des structures après dépliage)**.** *Une structure reste une structure après dépliage.*
$(\forall t, \varphi, \quad t \in S_\varphi \implies t^\star \in S_\varphi)$

Démonstration. Par induction sur la dérivation de $t \in S_\varphi$.

- S-VAR : $t = x$ avec $x \in \varphi$. Donc $x^\star \in S_\varphi$
- S-APP : $t = u\ v$ avec $u \in S_\varphi$. Et $(u\ v)^\star = u^\star\ v^\star$. Par hypothèse d'induction sur $u \in S_\varphi$ on a $u^\star \in S_\varphi$. Donc par la règle S-APP on a $(u\ v)^\star \in S_\varphi$
- S-ES : $t = u[x \setminus v]$ avec $u \in S_\varphi$. Et $(u[x \setminus v])^\star = u^\star\{x \setminus v^\star\}$. Et par hypothèse d'induction sur $u \in S_\varphi$, on a $u^\star \in S_\varphi$. Par le lemme 28 sur $u^\star \in S_\varphi$ on a $(u^\star\{x \setminus v^\star\}) \in S_\varphi$. Donc on a $(u[x \setminus v])^\star \in S_\varphi$.
- S-ES-S : $t = u[x \setminus v]$ avec $u \in S_{\varphi\cup\{x\}}$ et $v \in S_\varphi$. Et $(u[x \setminus v])^\star = u^\star\{x \setminus v^\star\}$. Et par hypothèse d'induction sur $u \in S_{\varphi\cup\{x\}}$ et sur $v \in S_\varphi$, on a $u^\star \in S_{\varphi\cup\{x\}}$ et $v^\star \in S_\varphi$. Par le lemme 29 sur $u^\star \in S_{\varphi\cup\{x\}}$ et $v^\star \in S_\varphi$ on a $u^\star\{x \setminus v^\star\} \in S_\varphi$ Donc on a $(u[x \setminus v])^\star \in S_\varphi$. □

Lemme 31 (Dépliage des formes normales)**.** *Si un terme est en forme normale alors sa forme dépliée est également une forme normale dans le lambda-calcul.*
$(t \in \mathscr{N} \implies t^\star \in \mathscr{N}_\lambda)$

Démonstration. Par induction sur la dérivation de $t \in \mathscr{N}$

- NF-VAR $t = x$: une variable dépliée reste une variable. Donc par la règle VAR $x \in \mathscr{N}_\lambda$.
- NF-APP $t = u\ v$: $u\ v \in \mathscr{N}$ avec $u \in \mathscr{N}$, $v \in \mathscr{N}$ et $u \in S_\varphi$. Par hypothèse d'induction sur u et sur v on a $u^\star \in \mathscr{N}_\lambda$ et $v^\star \in \mathscr{N}_\lambda$. Et par le lemme 30 sur $u \in S_\varphi$ on a $u^\star \in S_\varphi$. Par définition des structures, $u \notin Val$. Par la règle APP de la figure 2.13, $u^\star\ v^\star \in \mathscr{N}_\lambda$. Puis par définition du dépliage $(u\ v)^\star = u^\star\ v^\star$. Donc $(u\ v)^\star \in \mathscr{N}_\lambda$.
- NF-ABS $t = \lambda x.u$: $\lambda x.u \in \mathscr{N}$ avec $u \in \mathscr{N}$. Par hypothèse d'induction sur u on a $u^\star \in \mathscr{N}_\lambda$. Par le règle ABS de la figure 2.13, $\lambda x.u^\star \in \mathscr{N}_\lambda$. Puis par définition du dépliage $(\lambda x.u)^\star = \lambda x.u^\star$. Donc $(\lambda x.u)^\star \in \mathscr{N}_\lambda$.
- NF-ES $t = u[x \setminus v]$: $u[x \setminus v] \in \mathscr{N}$ avec $u \in \mathscr{N}$ et $u \notin \mathscr{V}_x$. Le lemme 26 nous dit que $x \notin \mathrm{FV}(u^\star)$. Et la proposition 4 nous dit que $u^\star\{x \setminus v^\star\} = u^\star$. Par définition du dépliage, $(u[x \setminus v])^\star = u^\star\{x \setminus v^\star\}$. Par hypothèse d'induction sur u on a $u^\star \in \mathscr{N}_\lambda$. Donc $(u[x \setminus v])^\star \in \mathscr{N}_\lambda$.
- NF-ES-S $t = u[x \setminus v]$: $u[x \setminus v] \in \mathscr{N}$ avec $u \in \mathscr{N}$ et $v \in \mathscr{N}$ et $v \in S_\varphi$. Le lemme 30 sur $v \in S_\varphi$ nous dit que $v^\star \in S_\varphi$. Par définition des structures $v^\star \notin Val$. Par hypothèse d'induction sur $u \in \mathscr{N}$ et sur $v \in \mathscr{N}$ on a $u^\star \in \mathscr{N}_\lambda$ et sur $v^\star \in \mathscr{N}_\lambda$. Et avec la proposition 7 on a $u^\star\{x \setminus v^\star\} \in \mathscr{N}_\lambda$. Par définition du dépliage $u^\star\{x \setminus v^\star\} = (u[x \setminus v])^\star$. Donc $(u[x \setminus v])^\star \in \mathscr{N}_\lambda$. □

3.3 Complétude

Cette section présente, dans un premier temps, le calcul hôte section 3.3.1 et les différents résultats qui ont déjà été prouvés à son sujet. Puis, dans un deuxième temps, nous nous servons de ces résultats pour prouver la complétude de notre calcul λ_{sn} dans la section 3.3.2.

$$\dfrac{\text{TY-VAR}}{x : \{\!\{\sigma\}\!\} \vdash x : \sigma} \qquad \dfrac{\Gamma \vdash t : \mathcal{M} \to \tau \qquad (\Delta_\sigma \vdash u : \sigma)_{\sigma \in \mathcal{M}}}{\Gamma + \sum_{\sigma \in \mathcal{M}} \Delta_\sigma \vdash t\, u : \tau}\ \text{TY-APP}$$

$$\dfrac{\Gamma, x : \mathcal{M} \vdash t : \tau}{\Gamma \vdash \lambda x.t : \mathcal{M} \to \tau}\ \text{TY-ABS} \qquad \dfrac{\Gamma, x : \mathcal{M} \vdash t : \tau \qquad (\Delta_\sigma \vdash u : \sigma)_{\sigma \in \mathcal{M}}}{\Gamma + \sum_{\sigma \in \mathcal{M}} \Delta_\sigma \vdash t[x \setminus u] : \tau}\ \text{TY-ES}$$

FIGURE 3.14 – Règles de typage pour λ_{c}

3.3.1 Le calcul hôte λ_{c}

Notre calcul λ_{sn} est inclus dans un calcul déjà connu, λ_{c}, qui sert d'outil technique dans l'article de Balabonski et al. [9] et que nous nommons calcul hôte. Ce calcul donne la forme générale des règles de réduction permettant la mémoïsation et s'accompagne d'un système de types avec intersection non-idempotent. L'utilisation d'un système de types avec intersection non-idempotent évite les traditionnels fastidieux lemmes de commutation syntaxiques, fournissant ainsi des preuves plus élégantes. Cela constitue une amélioration par rapport à la technique utilisée dans les travaux précédents [25, 9], rendue possible par notre présentation de style SOS. Sa réduction n'est cependant pas contrainte par une quelconque notion de nécessité. Le λ_{c}-calcul utilise la syntaxe des lambda-termes avec des substitutions explicites, qui est isomorphe à la syntaxe originelle du calcul d'appel par nécessité utilisant les let-bindings [5]. Cette syntaxe est exactement la même que celle de notre calcul λ_{sn}.

Notez que ce calcul ne permet que la substitution des abstractions et non des variables, comme on le voit parfois [27]. Ce comportement restreint est suffisant pour les principaux résultats de ce manuscrit, et permet une présentation plus compacte. Enfin, la règle $\to_{\mathsf{gc}}$ décrit la suppression d'une substitution explicite d'une variable qui ne vit plus. La réduction par l'une de ces règles dans n'importe quel contexte s'écrit $t \to_{\mathsf{c}} u$.

Le calcul λ_{c} vient avec un système de types avec intersection non-idempotent [17, 20], défini dans [26] en ajoutant des substitutions explicites au système original de [20]. Un type τ peut être un type variable α ou un type flèche $\mathcal{M} \to \tau$, où $\mathcal{M}$ est un multi-ensemble $\{\!\{\sigma_1, \ldots, \sigma_n\}\!\}$ de types.

Un environnement de typage Γ associe à chaque variable de son domaine un multi-ensemble de types. Ce multi-ensemble contient un type pour chaque utilisation potentielle de la variable, et peut être vide si la variable n'est pas réellement utilisée. Dans ce dernier cas, la variable peut simplement être supprimée de l'environnement de typage, i.e., nous égalisons Γ avec $\Gamma, x : \{\!\{\}\!\}$.

Un jugement de typage $\Gamma \vdash t : \tau$ attribue exactement un type au terme t. Comme le montrent les règles de typage figurant sur la figure 3.14, un argument d'une application ou d'une substitution explicite peut être typé plusieurs fois dans une dérivation. Il convient de noter que, dans les règles, le descripteur $\sigma \in \mathcal{M}$ quantifie sur l'ensemble des instances des éléments du multi-ensemble $\mathcal{M}$.

Exemple Le terme $\Omega = (\lambda x.x\ x)\ (\lambda x.x\ x)$ n'est pas typable, Nous allons vérifier que $(\lambda xy.x\ (x\ x))\ (\lambda z.z)\ \Omega$ l'est Soit σ_0 un type. On note $\sigma_1 = \{\!\{\sigma_0\}\!\} \to \sigma_0$ et $\sigma_2 = \{\!\{\sigma_1\}\!\} \to \sigma_1$. Tout d'abord, nous dérivons le jugement $\vdash \lambda xy.x\ (x\ x) : \{\!\{\sigma_1, \sigma_2, \sigma_2\}\!\} \to \{\!\{\}\!\} \to \sigma_1$ de la manière suivante. La principale subtilité de la dérivation réside dans le fractionnement adéquat de l'environnement dans lequel la règle d'application est utilisée.

$$
\dfrac{\dfrac{\dfrac{}{x : \{\!\{\sigma_2\}\!\} \vdash x : \{\!\{\sigma_1\}\!\} \to \sigma_1}\ \textsc{Ty-VAR} \qquad \dfrac{\dfrac{}{x : \{\!\{\sigma_2\}\!\} \vdash x : \{\!\{\sigma_1\}\!\} \to \sigma_1}\ \textsc{Ty-VAR} \quad \dfrac{}{x : \{\!\{\sigma_1\}\!\} \vdash x : \sigma_1}\ \textsc{Ty-VAR}}{x : \{\!\{\sigma_1, \sigma_2\}\!\} \vdash x\ x : \sigma_1}\ \textsc{Ty-APP}}{\dfrac{x : \{\!\{\sigma_1, \sigma_2, \sigma_2\}\!\}, y : \{\!\{\}\!\} \vdash x\ (x\ x) : \sigma_1}{x : \{\!\{\sigma_1, \sigma_2, \sigma_2\}\!\} \vdash \lambda y.x\ (x\ x) : \{\!\{\}\!\} \to \sigma_1}\ \textsc{Ty-ABS}}\ \textsc{Ty-APP}}{\vdash \lambda xy.x\ (x\ x) : \{\!\{\sigma_1, \sigma_2, \sigma_2\}\!\} \to \{\!\{\}\!\} \to \sigma_1}\ \textsc{Ty-ABS}
$$

FIGURE 3.15 – Typage de $\lambda xy.x\ (x\ x)$

De ce premier jugement, nous déduisons une dérivation de typage pour $(\lambda xy.x\ (x\ x))\ (\lambda z.z)\ \Omega$. Cette dérivation contient trois dérivations de type pour le premier argument $\lambda z.z$ et pas de dérivation de type pour le second argument Ω. Les trois dérivations de type pour $\lambda z.z$ sont pour chacun des types dans le multi-ensemble $\{\!\{\sigma_1, \sigma_2, \sigma_2\}\!\}$. Deux d'entre elles sont identiques.

$$
\dfrac{\dfrac{\dots \qquad \dfrac{\dfrac{}{z : \{\!\{\sigma_0\}\!\} \vdash z : \sigma_0}\ \textsc{Ty-VAR}}{\vdash \lambda z.z : \sigma_1}\ \textsc{Ty-ABS} \quad \dfrac{\dfrac{}{z : \{\!\{\sigma_1\}\!\} \vdash z : \sigma_1}\ \textsc{Ty-VAR}}{\vdash \lambda z.z : \sigma_2}\ \textsc{Ty-ABS} \quad \dfrac{\dfrac{}{z : \{\!\{\sigma_1\}\!\} \vdash z : \sigma_1}\ \textsc{Ty-VAR}}{\vdash \lambda z.z : \sigma_2}\ \textsc{Ty-ABS}}{\vdash (\lambda xy.x(xx))\ (\lambda z.z) : \{\!\{\}\!\} \to \sigma_1}\ \textsc{Ty-APP}}{\vdash (\lambda xy.x(xx))\ (\lambda z.z)\ \Omega : \sigma_1}\ \textsc{Ty-APP}
$$

FIGURE 3.16 – Typage de $(\lambda xy.x(xx))\ (\lambda z.z)\ \Omega$

Ce système de type est connu pour caractériser les lambda-termes qui sont normalisables pour la bêta-réduction, si elle est associée à la condition supplémentaire que le multi-ensemble vide $\{\!\{\}\!\}$ n'apparaît pas à une position positive dans le jugement de typage $\Gamma \vdash t : \tau$. Les occurrences positives et négatives des types sont définies dans la figure 3.17.

Théorème 2 (Typabilité [19, 13]). *Si le lambda-terme pur t est normalisable pour la bêta-réduction, alors il existe un jugement de typage $\Gamma \vdash t : \tau$ dérivable, tel que $\{\!\{\}\!\} \notin \mathcal{T}_+(\Gamma \vdash t : \tau)$.*

$$
\begin{array}{rcl}
\mathcal{T}_+(\alpha) & = & \{\alpha\} \\
\mathcal{T}_+(\mathcal{M}) & = & \{\mathcal{M}\} \cup \bigcup_{\sigma\in\mathcal{M}} \mathcal{T}_+(\sigma) \\
\mathcal{T}_+(\mathcal{M}\to\sigma) & = & \{\mathcal{M}\to\sigma\} \cup \mathcal{T}_-(\mathcal{M}) \cup \mathcal{T}_+(\sigma) \\
\mathcal{T}_+(\Gamma\vdash t:\sigma) & = & \mathcal{T}_+(\sigma) \cup \bigcup_{x\in\mathsf{dom}(\Gamma)} \mathcal{T}_-(\Gamma(x))
\end{array}
\qquad
\begin{array}{rcl}
\mathcal{T}_-(\alpha) & = & \emptyset \\
\mathcal{T}_-(\mathcal{M}) & = & \bigcup_{\sigma\in\mathcal{M}} \mathcal{T}_-(\sigma) \\
\mathcal{T}_-(\mathcal{M}\to\sigma) & = & \mathcal{T}_+(\mathcal{M}) \cup \mathcal{T}_-(\sigma)
\end{array}
$$

FIGURE 3.17 – Définition des occurrences positives et négatives des types

Exemple Soient α, β, γ, et δ des variables de type. Les occurrences de type positives et négatives du type $\tau = \{\!\{\{\!\{\alpha,\beta\}\!\} \to \gamma\}\!\} \to \{\!\{\}\!\} \to \delta$ sont les suivantes :

$$
\begin{array}{rcl}
\mathcal{T}_+(\tau) & = & \{\{\!\{\{\!\{\alpha,\beta\}\!\} \to \gamma\}\!\} \to \{\!\{\}\!\} \to \delta, \{\!\{\alpha,\beta\}\!\}, \alpha, \beta, \{\!\{\}\!\} \to \delta, \delta\} \\
\mathcal{T}_-(\tau) & = & \{\{\!\{\{\!\{\alpha,\beta\}\!\} \to \gamma\}\!\}, \{\!\{\alpha,\beta\}\!\} \to \gamma, \gamma, \{\!\{\}\!\}\}
\end{array}
$$

Une dérivation de typage Φ pour un jugement de typage $\Gamma \vdash t : \tau$ (noté $\Phi \rhd \Gamma \vdash t : \tau$) définit dans t un ensemble de positions typées, qui sont les positions des sous-termes de t pour lesquels la dérivation Φ contient une sous-dérivation. Plus précisément :

— ε est une position typée pour toute dérivation ;
— si Φ se termine par la règle Ty-ABS, Ty-APP ou Ty-ES, alors $0p$ est une position typée de Φ si p est une position typée de la sous-dérivation Φ' par rapport à la première prémisse ;
— si Φ se termine par la règle Ty-APP ou Ty-ES, alors $1p$ est une position typée de Φ si p est une position typée de la sous-dérivation Φ' par rapport à l'une des instances de la seconde prémisse.

Notons que, dans ce dernier cas, il n'y a pas d'instance de la seconde prémisse et aucune position typée $1p$ lorsque le multi-ensemble $\mathcal{M}$ est vide. Au contraire, lorsque $\mathcal{M}$ a plusieurs éléments, nous obtenons l'union des positions typées apportées par chaque instance.

Exemple Dans la dérivation proposée ci-dessus pour le jugement
$\vdash (\lambda xy.x(xx))\ (\lambda z.z)\ \Omega : \sigma_1$, la position ε, et toutes les positions des termes commençant par 0 sont typées. Au contraire, aucune position commençant par 1 n'est typée. Autrement dit, toutes les positions sont typées sauf celles de Ω.

Ces positions typées ont une propriété importante ; elles satisfont une version pondérée des théorèmes de préservation de typage par réduction, qui assure une taille de dérivation décroissante, que nous utiliserons dans la section suivante. Nous appelons taille d'une dérivation Φ le nombre de nœuds de l'arbre de dérivation.

Théorème 3 (Préservation pondérée du typage par la réduction [9]). *Si $\Phi \rhd \Gamma \vdash t : \tau$ et $t \to_{\mathsf{c}} t'$ par réduction d'un radical à une position typée, alors il existe une dérivation $\Phi' \rhd \Gamma \vdash t' : \tau$ avec Φ', strictement plus petit que Φ.*

Notez également que le typage est préservé en inversant les substitutions explicites en bêta-radicaux.

Proposition 8 (Typage et substitution [9]). *$\Gamma \vdash (\lambda x.t)\ u : \tau$ si et seulement si $\Gamma \vdash t[x \setminus u] : \tau$.*

3.3.2 Complétude

Dans cette section, nous prouvons la complétude du calcul λ_{sn} (Th. 5) en nous appuyant sur le système de types avec intersection non-idempotent. Notre calcul est complet par rapport à la normalisation dans le lambda-calcul dans un sens fort : chaque fois qu'un lambda-terme t admet une forme normale dans le lambda-calcul pur, chaque chemin de réduction dans λ_{sn} atteint finalement une représentation de cette forme normale.

Premièrement, nous utilisons la typabilité (Th. 2) qui garantit que tout lambda-terme normalisable admet une dérivation de typage (sans occurrence positive de $\{\!\{\}\!\}$). Deuxièmement, nous prouvons que toute réduction dans λ_{sn} d'un terme typé t (sans occurrence positive de $\{\!\{\}\!\}$) est à une position typée de t (Th. 4). Troisièmement, nous utilisons que la compatibilité du typage avec la réduction pondérée (Th. 3) fournit une mesure décroissante pour la réduction dans λ_{sn}. Enfin, nous faisons le lien entre la forme normale obtenue et la forme normale du lambda-calcul pur en utilisant les lemmes 4, 25 et 31.

La preuve sur les réductions typées (Th. 4) utilise un raffinement du système de types avec intersection non-idempotent de λ_{c}, donné dans la figure 3.18. Les deux systèmes dérivent les mêmes jugements de typage avec les mêmes positions typées. Le système raffiné présente un jugement de typage annoté $\Gamma \vdash^{\mu}_{\varphi} t : \tau$, intégrant les mêmes informations de contexte qui sont définies par le jugement $\mathcal{C}(\bullet) \vdash \mu, \varphi$ et utilisées dans la relation de réduction inductive $\xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}}$, à savoir l'ensemble φ des variables gelées à la position considérée et un marqueur μ de positions de type top-level. Ces annotations sont des contreparties fidèles aux annotations correspondantes des règles de réduction λ_{sn} ; leurs informations circulent vers le haut dans les règles d'inférence selon les mêmes critères.

En particulier, la règle de typage d'une abstraction est divisée en deux versions, Ty-ABS-$\bot$ et Ty-ABS-$\top$, cette dernière s'appliquant aux positions $\top$. Elle gèle ainsi la variable liée par l'abstraction (dans les deux règles, par convention de fraîcheur, nous supposons $x \notin \varphi$). La règle de typage d'une application est également divisée en deux versions : Ty-APP-$\mathcal{S}$ est applicable lorsque la partie gauche de l'application est une structure et marque la partie droite comme une position $\top$, tandis que Ty-APP est applicable dans le cas contraire. Notez que cette deuxième règle permet à l'argument de l'application d'être typé même si sa position n'est pas (encore) réductible, mais que son typage est dans une position $\bot$. Enfin, la règle pour le typage d'une substitution explicite est également divisée en deux versions, selon que le contenu de la substitution est une structure ou non, et traite l'ensemble des variables gelées en conséquence. Dans les deux cas, le contenu de la substitution est typé dans une position $\bot$, puisque cette position n'est jamais de type top-level.

Comme c'était le cas pour les positions d'évaluation, la définition des positions typées dépend largement des notions de variable gelée et de position de type top-level. Cependant, les ensembles de positions typées et de positions de type top-level n'ont pas de relation particulière entre eux. Certaines positions typées ne sont pas de type top-level (e.g., à gauche d'une application typée), et certaines positions de type top-level ne sont pas typées (e.g., l'argument d'une structure non typée). Néanmoins, il existe une connexion réelle entre l'ensemble des positions typées et l'ensemble des

$$\textsc{Ty-VAR}\ \frac{}{x : \{\!\{\sigma\}\!\} \vdash^{\mu}_{\varphi} x : \sigma} \qquad \textsc{Ty-ABS-}\bot\ \frac{\Gamma, x : \mathcal{M} \vdash^{\bot}_{\varphi} t : \tau}{\Gamma \vdash^{\bot}_{\varphi} \lambda x.t : \mathcal{M} \to \tau} \qquad \textsc{Ty-ABS-}\top\ \frac{\Gamma, x : \mathcal{M} \vdash^{\top}_{\varphi \cup \{x\}} t : \tau}{\Gamma \vdash^{\top}_{\varphi} \lambda x.t : \mathcal{M} \to \tau}$$

$$\textsc{Ty-APP}\ \frac{\Gamma \vdash^{\bot}_{\varphi} t : \mathcal{M} \to \tau \qquad (\Delta_\sigma \vdash^{\bot}_{\varphi} u : \sigma)_{\sigma \in \mathcal{M}}}{\Gamma + \sum_{\sigma \in \mathcal{M}} \Delta_\sigma \vdash^{\mu}_{\varphi} t\ u : \tau} \qquad \textsc{Ty-APP-}\mathcal{S}\ \frac{\Gamma \vdash^{\bot}_{\varphi} t : \mathcal{M} \to \tau \quad t \in \mathcal{S}_\varphi \quad (\Delta_\sigma \vdash^{\top}_{\varphi} u : \sigma)_{\sigma \in \mathcal{M}}}{\Gamma + \sum_{\sigma \in \mathcal{M}} \Delta_\sigma \vdash^{\mu}_{\varphi} t\ u : \tau}$$

$$\textsc{Ty-ES}\ \frac{\Gamma, x : \mathcal{M} \vdash^{\mu}_{\varphi} t : \tau \quad (\Delta_\sigma \vdash^{\bot}_{\varphi} u : \sigma)_{\sigma \in \mathcal{M}}}{\Gamma + \sum_{\sigma \in \mathcal{M}} \Delta_\sigma \vdash^{\mu}_{\varphi} t[x \setminus u] : \tau} \qquad \textsc{Ty-ES-}\mathcal{S}\ \frac{\Gamma, x : \mathcal{M} \vdash^{\mu}_{\varphi \cup \{x\}} t : \tau \quad u \in \mathcal{S}_\varphi \quad (\Delta_\sigma \vdash^{\bot}_{\varphi} u : \sigma)_{\sigma \in \mathcal{M}}}{\Gamma + \sum_{\sigma \in \mathcal{M}} \Delta_\sigma \vdash^{\mu}_{\varphi} t[x \setminus u] : \tau}$$

FIGURE 3.18 – Système annoté de types avec intersection non-idempotent pour λ_{sn}

positions d'évaluation, donnée par le théorème 4.

Nous écrivons $\Phi \rhd \Gamma \vdash^{\mu}_{\varphi} t : \tau$ s'il existe une dérivation Φ du jugement de typage annoté $\Gamma \vdash^{\mu}_{\varphi} t : \tau$. Nous dénotons par $\mathsf{fzt}(\Phi)$ l'ensemble des types associés aux variables gelées dans les jugements de la dérivation Φ.

Lemme 32 (Annotation de dérivation de typage)**.** *S'il existe une dérivation $\Phi \rhd \Gamma \vdash t : \tau$, alors pour tout φ et μ il existe une dérivation $\Phi' \rhd \Gamma \vdash^{\mu}_{\varphi} t : \tau$ telle que les ensembles de positions typées dans Φ et Φ' sont égaux.*

Démonstration. Par induction sur Φ, puisque les annotations n'interfèrent pas avec le typage. □

La propriété inverse est également vraie, par effacement des annotations, mais elle n'est pas utilisée dans la preuve du résultat de complétude.

La partie la plus importante de la preuve du théorème 4 consiste à s'assurer que tout argument d'une structure typée est lui-même à une position typée. Ceci découle des trois lemmes suivants.

Lemme 33 (Structure typée)**.** *Si $\Gamma \vdash^{\mu}_{\varphi} t : \tau$ et $t \in \mathcal{S}_\varphi$, alors il existe un $x \in \varphi$ tel que $\tau \in \mathcal{T}_+(\Gamma(x))$.*

Démonstration. Par induction sur la structure de t. Le cas le plus intéressant est celui d'une substitution explicite $t_1[x \setminus t_2]$. L'hypothèse d'induction appliquée sur t_1 peut donner la variable x qui n'apparaît pas dans la conclusion, mais dans ce cas, t_2 est garanti comme étant une structure dont le type contient τ. Donnons quelques détails supplémentaires.

— Cas $t = x$. Par définition de $x \in \mathcal{S}_\varphi$ on déduit $x \in \varphi$. De plus, la seule règle applicable pour dériver $\Gamma \vdash^{\mu}_{\varphi} x : \tau$ est TY-VAR, qui fournit la conclusion.

— Cas $t = t_1\ t_2$. Par définition de $t_1\ t_2 \in \mathcal{S}_\varphi$ on déduit $t_1 \in \mathcal{S}_\varphi$. De plus, les seules règles applicables pour déduire $\Gamma \vdash^{\mu}_{\varphi} t_1\ t_2 : \tau$ sont TY-APP et TY-APP-$\mathcal{S}$. Toutes deux ont une prémisse $\Gamma' \vdash^{\bot}_{\varphi} t_1 : \mathcal{M} \to \tau$ avec $\Gamma' \subseteq \Gamma$, à

laquelle l'hypothèse d'induction s'applique, assurant $\mathcal{M} \to \tau \in \mathcal{T}_+(\Gamma'(x))$ et donc $\tau \in \mathcal{T}_+(\Gamma'(x))$ et $\tau \in \mathcal{T}_+(\Gamma(x))$.
- Cas $t = \lambda x.t_1$. Ce cas est exclu car une abstraction n'est pas une structure.
- Cas $t = t_1[x \setminus t_2]$. On raisonne par cas sur les dernières règles appliquées pour dériver $t_1[x \setminus t_2] \in \mathcal{S}_\varphi$ et $\Gamma \vdash^\mu_\varphi t_1[x \setminus t_2] : \tau$. Il existe deux règles possibles pour chacune d'elles.
  - Cas où $t_1[x\setminus t_2] \in \mathcal{S}_\varphi$ se déduit de $t_1 \in \mathcal{S}_\varphi$ (avec $x \notin \varphi$), et $\Gamma \vdash^\mu_\varphi t_1[x\setminus t_2] : \tau$ vient de la règle Ty-ES. Cette règle a notamment une prémisse $\Gamma' \vdash^\mu_\varphi t_1 : \tau$ pour un $\Gamma' = \Gamma'', x : \mathcal{M}$ tel que $\Gamma'' \subseteq \Gamma$. Nous avons donc par hypothèse d'induction sur t_1 que $\tau \in \mathcal{T}_+(\Gamma'(y))$ pour $y \in \varphi \cap \mathsf{dom}(\Gamma')$. Puisque $y \in \varphi$ et $x \notin \varphi$, nous avons $y \neq x$. Alors, $y \in \mathsf{dom}(\Gamma'')$, $y \in \mathsf{dom}(\Gamma)$, et $\Gamma(y) = \Gamma''(y)$.
  - Dans les trois autres cas, nous avons :
    1. une hypothèse $t_1 \in \mathcal{S}_\varphi$ ou $t_1 \in \mathcal{S}_{\varphi\cup\{x\}}$, d'où l'on déduit $t_1 \in \mathcal{S}_{\varphi\cup\{x\}}$,
    2. une hypothèse $\Gamma' \vdash^\mu_\varphi t_1 : \tau$ ou $\Gamma' \vdash^\mu_{\varphi\cup\{x\}} t_1 : \tau$ (pour un $\Gamma' = \Gamma'', x : \mathcal{M}$ tel que $\Gamma'' \subseteq \Gamma$), d'où l'on déduit : $\Gamma' \vdash^\mu_{\varphi\cup\{x\}} t_1 : \tau$, et
    3. une hypothèse $t_2 \in \mathcal{S}_\varphi$, provenant de la dérivation de $t_1[x \setminus t_2] \in S_\varphi$ ou de la dérivation de $\Gamma \vdash^\mu_\varphi t_1[x \setminus t_2] : \tau$ (ou les deux).

    Alors par hypothèse d'induction sur t_1, on a $\tau \in \mathcal{T}_+(\Gamma'(y))$ pour un certain $y \in \varphi \cup \{x\}$.
    - Si $y \neq x$, alors $y \in \varphi$ et $\Gamma(y) = \Gamma''(y)$, ce qui permet une conclusion directe.
    - Si $y = x$, alors $\tau \in \mathcal{T}_+(\Gamma'(x))$ implique $\mathcal{M} \neq \{\!\{\}\!\}$. Soit $\sigma \in \mathcal{M}$ avec $\tau \in \mathcal{T}_+(\sigma)$. L'instance de la règle Ty-ES ou Ty-ES-$\mathcal{S}$ que nous considérons a donc au moins une prémisse $\Delta \vdash^\bot_\varphi t_2 : \sigma$ avec $\Delta \subseteq \Gamma$. Puisque $t_2 \in \mathcal{S}_\varphi$, par hypothèse d'induction sur t_2, il existe $z \in \varphi \cap \mathsf{dom}(\Delta)$ tel que $\sigma \in \mathcal{T}_+(\Delta(z))$. Alors, $\tau \in \mathcal{T}_+(\Delta(z))$ et $\tau \in \Gamma$.

$\square$

Lemme 34 (Propriété de la sous-formule)**.**

1. *Si* $\Phi \rhd \Gamma \vdash^\top_\varphi t : \tau$ *alors* $\begin{cases} \mathcal{T}_+(\mathit{fzt}(\Phi)) \subseteq \bigcup_{x\in\varphi} \mathcal{T}_+(\Gamma(x)) \cup \mathcal{T}_-(\tau) \\ \mathcal{T}_-(\mathit{fzt}(\Phi)) \subseteq \bigcup_{x\in\varphi} \mathcal{T}_-(\Gamma(x)) \cup \mathcal{T}_+(\tau) \end{cases}$
2. *Si* $\Phi \rhd \Gamma \vdash^\bot_\varphi t : \tau$ *alors* $\begin{cases} \mathcal{T}_+(\mathit{fzt}(\Phi)) \subseteq \bigcup_{x\in\varphi} \mathcal{T}_+(\Gamma(x)) \\ \mathcal{T}_-(\mathit{fzt}(\Phi)) \subseteq \bigcup_{x\in\varphi} \mathcal{T}_-(\Gamma(x)) \end{cases}$

Démonstration. Par induction mutuelle sur les dérivations de typage. La plupart des cas sont assez simples. Le seul cas difficile vient de la règle Ty-APP-$\mathcal{S}$, dans laquelle il y a une prémisse $\Delta \vdash^\top_\varphi u : \sigma$ avec le mode $\top$ mais avec un type σ qui n'apparaît pas clairement dans la conclusion. Ici, nous avons besoin du lemme sur les structures typées (Lem. 33) pour conclure. Donnons quelques détails supplémentaires.
- Les deux propriétés sont immédiates dans le cas Ty-VAR, où $\mathsf{fzt}(\Phi) = \{\sigma\}$.
- Cas pour l'abstraction.
  - Si $\Phi \rhd \Gamma \vdash^\bot_\varphi \lambda x.t : \mathcal{M} \to \tau$ par la règle Ty-ABS-$\bot$ avec la prémisse $\Phi' \rhd \Gamma, x : \mathcal{M} \vdash^\bot_\varphi t : \tau$. Soit $\Gamma' = \Gamma, x : \mathcal{M}$. Par hypothèse d'induction,

nous avons $\mathcal{T}_+(\mathsf{fzt}(\Phi')) \subseteq \bigcup_{y\in\varphi} \mathcal{T}_+(\Gamma'(y))$. Puisque $x \notin \varphi$ par convention de renommage, nous déduisons que $\mathcal{T}_+(\mathsf{fzt}(\Phi')) \subseteq \bigcup_{y\in\varphi} \mathcal{T}_+(\Gamma(y))$ et $\mathcal{T}_+(\mathsf{fzt}(\Phi)) \subseteq \bigcup_{y\in\varphi} \mathcal{T}_+(\Gamma(y))$. Il en va de même pour les occurrences de type négatives ce qui conclut ce cas.

  — Si $\Phi \rhd \Gamma \vdash^{\top}_{\varphi} \lambda x.t : \mathcal{M} \to \tau$ par la règle TY-ABS-$\top$ avec la prémisse $\Phi' \rhd \Gamma, x : \mathcal{M} \vdash^{\top}_{\varphi\cup\{x\}} t : \tau$. Soit $\Gamma' = \Gamma, x : \mathcal{M}$. Par hypothèse d'induction, nous avons

$$\begin{array}{rcl} \mathcal{T}_+(\mathsf{fzt}(\Phi')) & \subseteq & \bigcup_{y\in(\varphi\cup\{x\})} \mathcal{T}_+(\Gamma'(y)) \cup \mathcal{T}_-(\tau) \\ & = & \bigcup_{y\in\varphi} \mathcal{T}_+(\Gamma(y)) \cup \mathcal{T}_+(\mathcal{M}) \cup \mathcal{T}_-(\tau) \\ & = & \bigcup_{y\in\varphi} \mathcal{T}_+(\Gamma(y)) \cup \mathcal{T}_-(\mathcal{M} \to \tau) \end{array}$$

Ainsi, $\mathcal{T}_+(\mathsf{fzt}(\Phi)) \subseteq \bigcup_{y\in\varphi} \mathcal{T}_+(\Gamma(y)) \cup \mathcal{T}_-(\mathcal{M} \to \tau)$. Il en va de même pour les occurrences négatives, ce qui conclut ce cas

— Cas de l'application.
  — Les cas pour TY-APP sont par application immédiate des hypothèses d'induction.
  — Si $\Phi \rhd \Gamma \vdash^{\mu}_{\varphi} t\ u : \tau$ par la règle TY-APP-$\mathcal{S}$, avec la prémisse $\Phi_t \rhd \Gamma_t \vdash^{\bot}_{\varphi} t : \mathcal{M} \to \tau$, $t \in \mathcal{S}_\varphi$ et $\Phi_\sigma \rhd \Delta_\sigma \vdash^{\top}_{\varphi} u : \sigma$ pour $\sigma \in \mathcal{M}$, avec $\Gamma_t \subseteq \Gamma$ et $\Gamma_\sigma \subseteq \Gamma$ pour tous $\sigma \in \mathcal{M}$. Indépendamment de la valeur de μ, nous montrons que $\mathcal{T}_+(\mathsf{fzt}(\Phi)) \subseteq \bigcup_{x\in\varphi} \mathcal{T}_+(\Gamma(x))$ et $\mathcal{T}_-(\mathsf{fzt}(\Phi)) \subseteq \bigcup_{x\in\varphi} \mathcal{T}_-(\Gamma(x))$ pour conclure des deux côtés de l'induction mutuelle.

Directement à partir de l'hypothèse d'induction, $\mathcal{T}_+(\mathsf{fzt}(\Phi_t)) \subseteq \bigcup_{x\in\varphi} \Gamma_t(x) \subseteq \mathcal{T}_+(\mathsf{fzt}(\Phi))$. Par hypothèse d'induction sur les autres prémisses, nous avons $\mathcal{T}_+(\mathsf{fzt}(\Phi_\sigma)) \subseteq \bigcup_{x\in\varphi} \Gamma_\sigma(x) \cup \mathcal{T}_-(\tau)$ pour $\sigma \in \mathcal{M}$. Nous avons immédiatement $\bigcup_{x\in\varphi} \Gamma_\sigma(x) \subseteq \bigcup_{x\in\varphi} \Gamma(x)$. Nous concluons en montrant que $\mathcal{T}_-(\sigma) \subseteq \mathcal{T}_+(\Gamma_t(x))$ pour un certain $x \in \varphi$. Puisque $t \in \mathcal{S}_\varphi$, par la première propriété des sous-formules et l'hypothèse de typage sur t, nous déduisons que il existe un $x \in \varphi$ tel que $\mathcal{M} \to \tau \in \mathcal{T}_+(\Gamma_t(x))$. Par proximité des ensembles d'occurrences de type $\mathcal{T}_+(\tau)$ cela signifie $\mathcal{T}_+(\mathcal{M} \to \tau) \subseteq \mathcal{T}_+(\Gamma_t(x))$. Par définition $\mathcal{T}_+(\mathcal{M} \to \tau) = \mathcal{T}_-(\mathcal{M}) \cup \mathcal{T}_+(\tau) = \bigcup_{\sigma\in\mathcal{M}} \mathcal{T}_-(\sigma) \cup \mathcal{T}_+(\tau)$, ce qui nous permet de conclure la preuve que $\bigcup_{x\in\varphi} \Gamma_\sigma(x) \cup \mathcal{T}_-(\tau) \subseteq \bigcup_{x\in\varphi} \Gamma(x)$.

Le même argument s'applique également aux positions négatives, et conclut ce cas.

— Les cas des substitutions explicites sont similaires aux précédents.

□

Lemme 35 (Argument de structure typée)**.** *Si $\Phi \rhd \Gamma \vdash^{\mu}_{\varphi} t : \tau$ avec $\{\!\!\{\}\!\!\} \notin \mathcal{T}_+(\Gamma \vdash t : \tau)$, alors chaque jugement de typage de la forme $\Gamma' \vdash^{\mu'}_{\varphi'} s : \mathcal{M} \to \sigma$ dans Φ avec $s \in \mathcal{S}_{\varphi'}$ satisfait $\mathcal{M} \neq \{\!\!\{\}\!\!\}$.*

Démonstration. Soit $\Gamma' \vdash^{\mu'}_{\varphi'} s : \mathcal{M} \to \sigma$ dans Φ avec $s \in \mathcal{S}_{\varphi'}$. Par le lemme 33, il existe $x \in \varphi'$ tel que $\mathcal{M} \to \sigma \in \mathcal{T}_+(\Gamma'(x))$. Alors $\mathcal{M} \in \mathcal{T}_-(\Gamma'(x))$ et $\mathcal{M} \in \mathcal{T}_-(\mathsf{fzt}(\Phi))$. Par le lemme 34, $\mathcal{M} \in \mathcal{T}_+(\Gamma \vdash^{\mu}_{\varphi} t : \tau)$, donc $\mathcal{M} \neq \{\!\!\{\}\!\!\}$. □

Théorème 4 (Réduction typée)**.** *Si $\Phi \rhd \Gamma \vdash^{\mu}_{\varphi} t : \tau$ avec $\{\!\!\{\}\!\!\} \notin \mathcal{T}_+(\Gamma \vdash t : \tau)$, alors chaque λ_{sn}-réduction $t \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t'$ est à une position typée.*

Démonstration. Nous prouvons par induction sur $t \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t'$ que, si $\Phi \triangleright \Gamma \vdash^{\mu}_{\varphi} t : \tau$ avec Φ tel que tout jugement de typage $\Gamma' \vdash^{\mu'}_{\varphi'} s : \mathcal{M} \to \sigma$ dans Φ avec $s \in \mathcal{S}_{\varphi'}$ satisfait $\mathcal{M} \neq \{\!\{\}\!\}$, alors $t \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t'$ se réduit à une position typée (la restriction sur Φ est permise par le lemme 35). Puisque tous les autres cas de réduction concernent des positions qui sont systématiquement typées, nous nous concentrons ici sur APP-RIGHT et ES-RIGHT.

— Cas APP-RIGHT : $t\ u \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t\ u'$ avec $t \in \mathcal{S}_{\varphi}$ et $u \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn}} u'$, en supposant que $\Phi \triangleright \Gamma \vdash^{\mu}_{\varphi} t\ u : \sigma$. Par définition de la dernière règle dans Φ, nous savons qu'il existe une sous-dérivation $\Phi' \triangleright \Gamma' \vdash^{\perp}_{\varphi} t : \mathcal{M} \to \sigma$ et par hypothèse $\mathcal{M} \neq \{\!\{\}\!\}$. Alors u est typé en Φ et on peut conclure par hypothèse d'induction.

— Cas ES-RIGHT : $t[x \setminus u] \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t[x \setminus u']$ avec $t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn}} t'$ et $u \xrightarrow{\rho,\varphi,\perp}_{\mathsf{sn}} u'$, en supposant que $\Phi \triangleright \Gamma \vdash^{\mu}_{\varphi} t[x \setminus u] : \tau$. Par définition de la dernière règle dans Φ, nous savons qu'il existe une sous-dérivation $\Phi' \triangleright \Gamma', x : \mathcal{M} \vdash^{\mu}_{\varphi} t : \tau$. Par hypothèse d'induction, nous savons que la réduction $t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn}} t'$ est à une position typée dans Φ', donc x est typée dans t et $\mathcal{M} \neq \{\!\{\}\!\}$. Alors u est typée dans Φ et on peut conclure par hypothèse induction sur $u \xrightarrow{\rho,\varphi,\perp}_{\mathsf{sn}} u'$.

□

Théorème 5 (Complétude)**.** *Si un lambda-terme t est normalisable dans le lambda-calcul, alors t est fortement normalisant dans λ_{sn}. De plus, si n_β est la forme normale de t dans le lambda-calcul, alors toute forme normale n_{sn} de t dans λ_{sn} est telle que $n^{\star}_{\mathsf{sn}} = n_\beta$.*

Démonstration. Soit t un lambda-terme pur qui admet une forme normale n_β pour la bêta-réduction. Par le théorème 2 il existe un jugement de typage dérivable $\Gamma \vdash t : \tau$ tel que $\{\!\{\}\!\} \notin \mathcal{T}_+(\Gamma \vdash t : \tau)$. Ainsi par les théorèmes 4 et 3, le terme t est fortement normalisant pour $\to_{\mathsf{sn}}$. Soit $t \to^{*}_{\mathsf{sn}} n_{\mathsf{sn}}$ une réduction maximale dans λ_{sn}. Par le lemme 25, $n_{\mathsf{sn}} \in \mathscr{N}$, et par le lemme 31, $n^{\star}_{\mathsf{sn}}$ est une forme normale dans le lambda-calcul. De plus, par simulation (Lem. 4), il existe une réduction $t^{\star} \to^{*}_{\beta} n^{\star}_{\mathsf{sn}}$. Par unicité de la forme normale dans le lambda-calcul, $n^{\star}_{\mathsf{sn}} = n_\beta$. □

Notez que, malgré le fait que λ_{sn} ne bénéficie pas de la propriété du diamant, nos théorèmes de correction (Th. 1) et de complétude (Th. 5) impliquent que, dans λ_{sn}, un terme est normalisable si et seulement s'il est fortement normalisant.

3.3.3 Nécessité

Ces résultats de terminaison et de complétude nous permettent également de formaliser notre affirmation selon laquelle λ_{sn} ne réduit que les radicaux «nécessaires».

On écrit $t^{\dagger}$ le lambda-terme pur obtenu en inversant toutes les substitutions explicites d'un terme t du λ_{c}, qui peut être défini par les équations suivantes :

$$\begin{array}{rclcrcl} x^{\dagger} & = & x & \quad & (t\ u)^{\dagger} & = & t^{\dagger}\ u^{\dagger} \\ (\lambda x.t)^{\dagger} & = & \lambda x.(t^{\dagger}) & & (t[x \setminus u])^{\dagger} & = & (\lambda x.t^{\dagger})\ u^{\dagger} \end{array}$$

Nous avons le théorème suivant.

Proposition 9 (Stabilité du typage Par définition des substitutions explicites). *Pour tout terme t, $\Gamma \vdash t : \tau$ si et seulement si $\Gamma \vdash t^\dagger : \tau$.*

Démonstration. Par induction sur t. □

Pour tout terme t, le terme inversé $t^\dagger$ peut être relié au terme déplié $t^\star$ par bêta-réduction des bêta-radicaux introduits par l'opération †. Ceci est prouvé par une simple induction sur t.

Proposition 10. *Pour tout terme t, on a $t^\dagger \to_\beta^* t^\star$.*

Donc $t^\dagger$ est normalisable, et donc t est typable, si et seulement si $t^\star$ est normalisable.

Théorème 6 (Nécessité). *Considérons un λ_{sn}-terme t tel que $t^\star$ est normalisable dans le lambda-calcul. Pour toute dB-réduction $t \xrightarrow{\mathsf{dB},\varphi,\mu}_{\mathsf{sn}} t'$ dans λ_{sn}, le sous-terme r de t qui est réduit est tel qu'au moins une occurrence du radical r' correspondant dans $t^\star$ soit nécessaire.*

Démonstration. Supposons qu'aucune occurrence du radical r' ne soit nécessaire dans $t^\star$. Alors par définition, il existe une séquence de bêta-réductions de $t^\star$ qui conduit à la forme normale de $t^\star$ sans réduire aucune copie de r' ni aucun de ses résidus (et la forme normale, étant normale, ne contient aucun résidu de r'). Soit p la position de r dans t, et $t(\Omega)_p$ le terme obtenu de t en remplaçant r par le terme divergent Ω. Alors le terme $(t(\Omega)_p)^\star$ est égal au terme obtenu à partir de $t^\star$ en remplaçant chaque copie de r' par Ω, et peut encore être normalisé par la même séquence de bêta réduction. Ainsi, ce terme est normalisable dans le lambda-calcul.

Puisque $(t(\Omega)_p)^\dagger$ se bêta-réduit en $(t(\Omega)_p)^\star$, nous déduisons que le terme pur $(t(\Omega)_p)^\dagger$ est aussi normalisable dans le lambda-calcul. Ainsi, par le théorème 2 il y a un jugement de typage $\Gamma \vdash (t(\Omega)_p)^\dagger : \tau$ tel que $\{\!\{\}\!\} \notin \mathcal{T}_+(\Gamma \vdash t : \tau)$, et par la proposition 9 nous avons également $\Gamma \vdash t(\Omega)_p : \tau$. Donc par les théorèmes 4 et 3, le terme $t(\Omega)_p$ est fortement normalisant dans λ_{sn}.

Cependant, si $\to_{\mathsf{sn}}$ permet de réduire r dans t, alors elle permet aussi de réduire Ω dans $t(\Omega)_p$, ainsi que n'importe quelle réduit de Ω à la même position, ce qui implique l'existence d'une séquence infinie de réduction $\to_{\mathsf{sn}}$ depuis t : contradiction. Ainsi, au moins une occurrence du radical r' est nécessaire dans $t^\star$. □

3.4 Conclusion

Dans ce chapitre, nous avons réécrit notre calcul λ_{sn} sans les contextes, avec des règles d'inférence. Cette nouvelle présentation des règles de réduction nous a permis d'écrire des preuves plus facilement. Nous avons, dans un premier temps, énoncé plusieurs propriétés propres au calcul, certaines que nous avons prouvées. Suite à cela, nous avons commencé à prouver la correction (Th. 1) par rapport au lambda-calcul i.e., les résultats de notre calcul λ_{sn} correspondent bien aux résultats du lambda-calcul. Pour ce faire, nous avons utilisé les différentes propriétés que nous avons énoncées plus tôt et trois lemmes principaux : un lemme de simulation (Lem. 4) qui nous dit

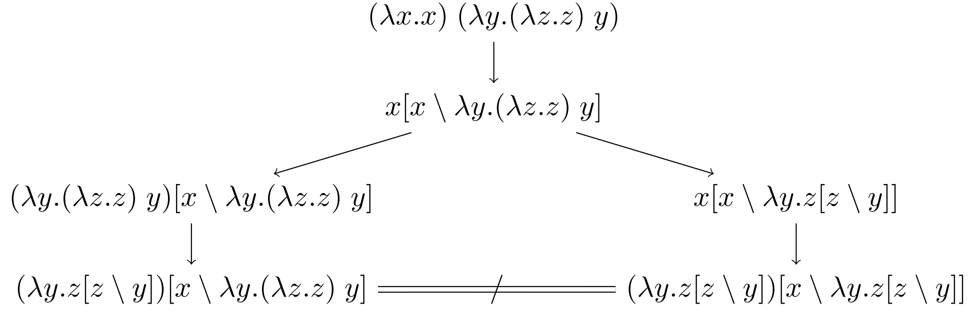

Figure 3.19 – Exemple de non-confluence

qu'une réduction dans notre calcul λ_{sn} correspond à une séquence de réduction dans le lambda-calcul classique ; un lemme de caractérisation des formes normales (Lem. 25) qui prouve que notre caractérisation des formes normales correspond bien à l'absence de radical ; et un lemme de dépliage des formes normales (Lem. 31) qui fait le lien entre nos formes normales et les formes normales du lambda-calcul classique. Une fois la correction prouvée, nous avons prouvé la complétude (Th. 5) par rapport au lambda-calcul i.e., tout lambda-terme qui a une forme normale dans le lambda-calcul classique est fortement normalisant dans notre calcul λ_{sn}. Pour ce faire, nous avons utilisé des résultats présents dans la littérature tels qu'un calcul hôte λ_{c} et système de types avec intersection non-idempotente. Ces résultats nous ont également permis de prouver une notion de nécessité (Th. 6) des réductions de notre calcul λ_{sn}.

Notre calcul n'est pas confluent, comme montré dans l'exemple de la figure 3.19. Toutefois, nous faisons la conjecture que le calcul est bien confluent, modulo la règle *gc*. Nous présentons dans le prochain chapitre une restriction du calcul qui limite le nombre de radicaux dupliqués et jouit d'une propriété de confluence forte.

Chapitre 4

Calcul amélioré $\lambda_{\mathsf{sn+}}$

Notre calcul λ_{sn} réduit les termes jusqu'à leurs formes normales et seuls les radicaux nécessaires sont réduits. Cependant, rien n'est garanti quant à la longueur de la séquence de réduction. En effet, un terme peut être substitué plusieurs fois avant d'être réduit, ce qui entraîne des calculs en double.

Dans le calcul λ_{sn}, pour appliquer la règle de substitution, la seule contrainte sur l'argument de la substitution est que le terme soit une valeur. Or, une valeur peut contenir des radicaux. Nous avons donc cherché à proposer un nouveau calcul qui restreigne cette règle afin d'éviter la duplication de calcul.

Dans la section 4.1, nous montrons que contraindre la substitution aux réponses en forme normale est trop restrictif, car cela peut bloquer la normalisation. Nous avons donc défini une version relâchée de la forme normale que nous avons nommée forme normale locale que nous présentons dans la section 4.2. Cependant, cette forme est suffisamment forte pour garantir qu'il n'y ait plus de radicaux accessibles dans le terme, ce qui nous permet d'éviter la duplication de calcul liée à ce phénomène. Comme $\lambda_{\mathsf{sn+}}$ est une restriction de λ_{sn}, il est inclus dans le premier calcul.

Dans la section 4.3, nous prouvons la correction de la caractérisation des formes normales locales, nous démontrons la propriété du diamant sur $\lambda_{\mathsf{sn+}}$, et nous expliquons en quoi $\lambda_{\mathsf{sn+}}$ sélectionne les meilleurs chemins de réduction de λ_{sn}.

4.1 Calcul

Notre proposition de calcul λ_{sn} garantit que, dans le processus de réduction d'un terme à sa forme normale, seuls les radicaux nécessaires sont réduits. Cependant, cela ne dit rien sur la longueur des séquences de réduction. En effet, un terme peut être substitué plusieurs fois avant d'être réduit, ce qui conduit à des calculs en double. Pour éviter cette duplication, nous avons d'abord voulu restreindre la substitution aux termes en forme normale, mais nous nous sommes aperçus que cela nous faisait perdre la propriété de normalisation forte, car il y avait certains termes qui ne pouvaient ni être réduits ni être substitués. Il n'était donc pas possible de normaliser un terme qui avait pourtant bien une forme normale. Nous allons expliquer ici ce phénomène.

Ce problème de blocage est principalement dû à une dissymétrie entre les variables non gelées à l'extérieur et à l'intérieur des substitutions. En effet, les variables

non gelées représentent les variables qui sont liées plus ou moins directement à une abstraction appliquée. Le paramètre qui gère cela est le mode : si une abstraction est vue en mode $\bot$, alors sa variable n'est pas gelée car l'abstraction est susceptible d'être appliquée. Le problème est que la règle de réduction ES-RIGHT (Fig. 3.3) qui nous permet d'aller évaluer le corps d'une substitution explicite ne le fait qu'en mode $\bot$. Le corps de la substitution est vu en mode $\bot$ car on ne sait pas où ce terme va être substitué et il peut potentiellement l'être dans une position appliquée, donc dans le doute on l'évalue en mode $\bot$. Les règles de réduction sont faites de sorte que si un terme est à droite d'une variable qui n'est pas gelée, il suffit de faire la bêta-réduction qui implique l'abstraction liée pour débloquer le terme de droite. Par exemple, dans le terme $(\lambda x.xR)\ R'$, le radical R n'est pas réductible car il est à droite de x qui n'est pas gelée. Pour pouvoir aller réduire R, il faut commencer par faire la bêta-réduction à top-level.

Dans λ_{sn}, les termes substitués peuvent être des valeurs arbitraires. En particulier, il peut s'agir d'abstractions dont le corps contient des radicaux. Puisque les variables substituées peuvent apparaître plusieurs fois, cela entraînerait que le radical soit réduit plusieurs fois si la valeur est substituée trop tôt. Dans l'exemple qui suit, I (comme dans les chapitres précédents) la séquence de réduction ne dépend ni de l'ensemble φ des variables gelées ni de la position μ. Nous ne les écrivons donc pas pour alléger un peu les notations. Les sous-termes qui sont sur le point d'être substitués ou réduits sont soulignés. Dans la figure qui suit $I = \lambda x.x$.

$$
\begin{array}{rcl}
\underline{(\lambda w.w\ w)\ (\lambda y.I\ y)} & \xrightarrow{\mathsf{db}}_{\mathsf{sn}} & (\underline{w}\ w)[w \setminus \lambda y.I\ y] \\
 & \xrightarrow{\mathsf{lsv}}_{\mathsf{sn}} & ((\lambda y.\underline{I\ y})\ w)[w \setminus \lambda y.I\ y] \\
 & \xrightarrow{\mathsf{db}}_{\mathsf{sn}} & \underline{((\lambda y.x[x \setminus y])\ w)}[w \setminus \lambda y.I\ y] \\
 & \xrightarrow{\mathsf{db}}_{\mathsf{sn}} & \underline{x}[x \setminus y][y \setminus w][w \setminus \lambda y.I\ y] \\
 & \xrightarrow{\mathsf{lsv}\,\times\,3}_{\mathsf{sn}} & (\lambda y.\underline{I\ y})[x \setminus \lambda y.I\ y][y \setminus \lambda y.I\ y][w \setminus \lambda y.I\ y] \\
 & \xrightarrow{\mathsf{db}}_{\mathsf{sn}} & (\lambda y.x[x \setminus y])[x \setminus \lambda y.I\ y][y \setminus \lambda y.I\ y][w \setminus \lambda y.I\ y]
\end{array}
$$

Remarquez comment $I\ y$ est réduit deux fois, ce qui n'aurait pas eu lieu si la seconde réduction avait porté sur le corps de l'abstraction dans la substitution. Cela suggère qu'une substitution devrait être autorisée uniquement si le terme substitué est en forme normale. Mais une telle exigence est incompatible avec notre calcul λ_{sn}, car elle empêcherait l'abstraction $\lambda y.y\ \Omega$ (avec Ω un terme divergent) d'être substituée dans l'exemple suivant, empêchant ainsi la normalisation (avec a un terme clos).

$$
\begin{array}{rcl}
(\underline{w}\ (\lambda x.a))[w \setminus \lambda y.y\ \Omega] & \xrightarrow{\mathsf{lsv}}_{\mathsf{sn}} & (\underline{(\lambda y.y\ \Omega)\ (\lambda x.a)})[w \setminus \lambda y.y\ \Omega] \\
 & \xrightarrow{\mathsf{db}}_{\mathsf{sn}} & (\underline{y}\ \Omega)[y \setminus \lambda x.a][w \setminus \lambda y.y\ \Omega] \\
 & \xrightarrow{\mathsf{lsv}}_{\mathsf{sn}} & (\underline{(\lambda x.a)\ \Omega})[y \setminus \lambda x.a][w \setminus \lambda y.y\ \Omega] \\
 & \xrightarrow{\mathsf{db}}_{\mathsf{sn}} & a[x \setminus \Omega][y \setminus \lambda x.a][w \setminus \lambda y.y\ \Omega]
\end{array}
$$

Remarquez comment la séquence de réductions a progressivement éliminé toutes les occurrences de Ω, jusqu'à ce que le seul terme restant à réduire soit le terme clos a.

En résumé, substituer n'importe quelle valeur est trop permissif et peut causer des calculs en double, tandis que substituer uniquement des formes normales est trop

$$\frac{t \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\mu}_{\mathsf{sn}} t' \qquad v \in \mathcal{N}_{\varphi,\emptyset,\bot} \qquad v \in \mathit{Val}}{t[x \setminus v] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'[x \setminus v]}\ \text{LSV-BASE}$$

FIGURE 4.1 – Nouvelle règle LSV-BASE pour $\lambda_{\mathsf{sn+}}$

restrictif car cela empêche la normalisation. Donc, nous avons besoin d'une notion relaxée de forme normale, que nous appelons forme normale locale. L'intuition est la suivante. Le terme $\lambda y.y\ \Omega$ n'est pas en forme normale. Mais dans une position $\bot$ la variable y n'est pas gelée, ce qui empêche toute nouvelle réduction de $y\ \Omega$.

Nous présentons les formes normales locales dans la section 4.2 qui est utilisée pour restreindre le critère *Val* dans la règle LSV-BASE. Ce calcul restreint, nommé $\lambda_{\mathsf{sn+}}$, a les mêmes règles que λ_{sn} (Fig. 3.3, 3.4 et 3.5), sauf que la règle LSV-BASE est remplacée par la règle de la figure 4.1.

Nous montrons ensuite que cette restriction est suffisamment forte pour garantir la propriété du diamant. Enfin, nous expliquons pourquoi notre calcul $\lambda_{\mathsf{sn+}}$ restreint ne produit que des séquences minimales parmi toutes les séquences de réduction autorisées par λ_{sn}, ce qui en fait une stratégie relativement optimale.

4.2 Forme normale locale

Nous allons ici caractériser un terme qui n'est pas une forme normale mais qui est irréductible en mode $\bot$. En effet le problème est que le corps de la substitution est vu en mode $\bot$, non pas parce qu'elle est appliquée, mais parce qu'elle pourrait être dans une position appliquée. C'est pourquoi nous avons besoin de faire la distinction parmi les variables non gelées, entre celles qui sont liées à l'extérieur (dont le calcul pourrait progresser) et celles qui le sont localement (qui sont bloquées en attendant la substitution).

Nous faisons la distinction entre les variables gelées (que nous stockons dans φ), les variables non gelées locales et néanmoins bloquées (que nous stockons dans ω) et les variables non gelées extérieures, (que nous ne traçons pas). En vérité, les seules variables qui sont non gelées sont celles qui sont liées à une abstraction qui n'est pas appliquée, mais qui est quand même vue en mode $\bot$. La seule situation dans laquelle ce phénomène se produit est lorsqu'une abstraction se trouve en tête de l'argument d'une substitution explicite, car elle est toujours vue en mode $\bot$. Ainsi, les seules variables que nous voulons vraiment mettre dans ω sont celles liées à une abstraction «de tête» et celles qui peuvent être substituées par une variable dans ω. En revanche, lorsque nous passons en mode «classique», c'est-à-dire lorsque les abstractions vues en mode $\bot$ sont vraiment appliquées, nous ne voulons plus mettre les variables qui y sont liées dans ω. C'est pourquoi nous avons commencé par définir deux formes normales locales. La première (Fig. 4.2) fonctionne en mode classique. La deuxième (Fig. 4.3) met les variables dans ω lorsque nous passons sous une abstraction en mode $\bot$, grâce à la règle LNF-ABS-$\bot$ et lorsque nous passons à gauche d'une substitution explicite

qui a pour sous-terme droit une structure dont la variable de tête est dans ω.

Nous constatons aussi que, dans la figure 4.3, nous avons la règle LNF-TOP qui permet de passer en mode «classique» à tout moment. Cela montre la monotonie du mode dans la forme normale locale. En revanche, lorsque nous sommes dans le mode $\top$, il n'est plus possible de retourner dans le mode de base, c'est-à-dire le mode $\bot$. Par la suite, nous fusionnons les définitions pour donner la caractérisation des formes normales locales (Fig. 4.4). Cette version fusionnée est plus facile à utiliser pour les preuves. De plus, nous constatons que les variables qui sont vraiment gelées sont toujours mises dans ω avec la règle LNF-ABS-$\bot$. Cela ne pose pas de problème car, dans les faits, il n'est pas possible qu'une abstraction appliquée soit en forme normale locale car les deux règles pour les applications veulent des structures à gauche, ce qui empêche tout radical, donc toute abstraction appliquée. L'aspect monotone du mode n'étant plus si évident, il sera prouvé par le lemme 45. Les règles d'inférence sont présentées dans la figure 4.4.

$$\frac{x \in \varphi \cup \omega}{x \in \mathcal{N}^{\top}_{\varphi,\omega}}\ \text{LNF-VAR} \qquad \frac{t \in \mathcal{N}^{\top}_{\varphi\cup\{x\},\omega}}{\lambda x.t \in \mathcal{N}^{\top}_{\varphi,\omega}}\ \text{LNF-ABS-}\top$$

$$\frac{t \in \mathcal{N}^{\top}_{\varphi,\omega} \quad t \in \mathcal{S}_{\varphi} \quad u \in \mathcal{N}^{\top}_{\varphi,\omega}}{t\ u \in \mathcal{N}^{\top}_{\varphi,\omega}}\ \text{LNF-APP-}\varphi \qquad \frac{t \in \mathcal{N}^{\top}_{\varphi,\omega} \quad t \in \mathcal{S}_{\omega}}{t\ u \in \mathcal{N}^{\top}_{\varphi,\omega}}\ \text{LNF-APP-}\omega \qquad \frac{t \in \mathcal{N}^{\top}_{\varphi,\omega}}{t[x \setminus u] \in \mathcal{N}^{\top}_{\varphi,\omega}}\ \text{LNF-ES}$$

$$\frac{t \in \mathcal{N}^{\top}_{\varphi\cup\{x\},\omega} \quad u \in \mathcal{N}^{\top}_{\varphi,\omega} \quad u \in \mathcal{S}_{\varphi}}{t[x \setminus u] \in \mathcal{N}^{\top}_{\varphi,\omega}}\ \text{LNF-ES-}\varphi \qquad \frac{t \in \mathcal{N}^{\top}_{\varphi,\omega\cup\{x\}} \quad u \in \mathcal{S}_{\omega}}{t[x \setminus u] \in \mathcal{N}^{\top}_{\varphi,\omega}}\ \text{LNF-ES-}\omega$$

FIGURE 4.2 – Forme normale locale $\top$

$$\frac{t \in \mathcal{N}^{\top}_{\varphi,\omega}}{t \in \mathcal{N}^{\bot}_{\varphi,\omega}}\ \text{LNF-TOP} \qquad \frac{t \in \mathcal{N}^{\bot}_{\varphi,\omega\cup\{x\}}}{\lambda x.t \in \mathcal{N}^{\bot}_{\varphi,\omega}}\ \text{LNF-ABS-}\bot$$

$$\frac{t \in \mathcal{N}^{\bot}_{\varphi,\omega}}{t[x \setminus u] \in \mathcal{N}^{\bot}_{\varphi,\omega}}\ \text{LNF-ES} \qquad \frac{t \in \mathcal{N}^{\bot}_{\varphi\cup\{x\},\omega} \quad u \in \mathcal{N}^{\top}_{\varphi,\omega} \quad u \in \mathcal{S}_{\varphi}}{t[x \setminus u] \in \mathcal{N}^{\bot}_{\varphi,\omega}}\ \text{LNF-ES-}\varphi$$

$$\frac{t \in \mathcal{N}^{\bot}_{\varphi,\omega\cup\{x\}} \quad u \in \mathcal{S}_{\omega}}{t[x \setminus u] \in \mathcal{N}^{\bot}_{\varphi,\omega}}\ \text{LNF-ES-}\omega$$

FIGURE 4.3 – Forme normale locale $\bot$

$$\dfrac{x \in \varphi \cup \omega}{x \in \mathcal{N}_{\varphi,\omega,\mu}}\ \text{LNF-VAR} \qquad \dfrac{t \in \mathcal{N}_{\varphi\cup\{x\},\omega,\top}}{\lambda x.t \in \mathcal{N}_{\varphi,\omega,\top}}\ \text{LNF-ABS-}\top \qquad \dfrac{t \in \mathcal{N}_{\varphi,\omega\cup\{x\},\bot}}{\lambda x.t \in \mathcal{N}_{\varphi,\omega,\bot}}\ \text{LNF-ABS-}\bot$$

$$\dfrac{t \in \mathcal{N}_{\varphi,\omega,\mu'} \qquad t \in \mathcal{S}_\varphi \qquad u \in \mathcal{N}_{\varphi,\omega,\top}}{t\ u \in \mathcal{N}_{\varphi,\omega,\mu}}\ \text{LNF-APP-}\varphi \qquad \dfrac{t \in \mathcal{N}_{\varphi,\omega,\mu'} \qquad t \in \mathcal{S}_\omega}{t\ u \in \mathcal{N}_{\varphi,\omega,\mu}}\ \text{LNF-APP-}\omega$$

$$\dfrac{t \in \mathcal{N}_{\varphi,\omega,\mu}}{t[x \setminus u] \in \mathcal{N}_{\varphi,\omega,\mu}}\ \text{LNF-ES} \qquad \dfrac{t \in \mathcal{N}_{\varphi\cup\{x\},\omega,\mu} \qquad u \in \mathcal{N}_{\varphi,\omega,\mu'} \qquad u \in \mathcal{S}_\varphi}{t[x \setminus u] \in \mathcal{N}_{\varphi,\omega,\mu}}\ \text{LNF-ES-}\varphi$$

$$\dfrac{t \in \mathcal{N}_{\varphi,\omega\cup\{x\},\mu} \qquad u \in \mathcal{S}_\omega}{t[x \setminus u] \in \mathcal{N}_{\varphi,\omega,\mu}}\ \text{LNF-ES-}\omega$$

FIGURE 4.4 – Forme normale locale

Si une abstraction se trouve en position $\top$, sa variable est ajoutée à l'ensemble φ des variables gelées, comme le montre la règle LNF-ABS-$\top$ de la figure 4.4. Mais si une abstraction est en position $\bot$, sa variable est ajoutée à l'ensemble ω, comme le montre la règle LNF-ABS-$\bot$. C'est ce qui se produit pour y dans $w[w \setminus \lambda y.y\ \Omega]$.

Pour une application, il faut toujours que la partie gauche soit une structure. Cependant, si la variable de tête de la structure n'est pas gelée (et donc dans ω), notre calcul λ_{sn} garantit qu'aucune réduction ne se produira dans la partie droite de l'application. Par conséquent, cette partie n'a pas besoin d'être contrainte de quelque manière que ce soit. C'est ce que montre la règle LNF-APP-ω de la figure 4.4. Cette règle s'applique à notre exemple, puisque $y\ \Omega$ est une structure dont la tête est $y \in \omega$. Les substitutions sont traitées de manière similaire, comme l'indique la règle LNF-ES-ω.

On notera que, dans les deux règles LNF-ES-φ et LNF-APP-φ, le mode de la forme normale locale du terme qui doit être une structure n'importe pas. Cela vient du fait que, quelque soit le mode, un terme qui est une structure regardera systématiquement les termes qui se trouvent à droite de la tête en mode $\top$, car les arguments d'une structure ne peuvent pas être appliqués.

La justification que l'argument de l'exemple $\lambda y.y\ \Omega$ est en forme normale locale dans une position $\bot$ est résumée par la dérivation suivante.

$$\dfrac{\dfrac{\text{LNF-VAR}\ \dfrac{y \in \{y\}}{y \in \mathcal{N}_{\emptyset,\{y\},\bot}} \qquad y \in S_{\{y\}}}{y\ \Omega \in \mathcal{N}_{\emptyset,\{y\},\bot}}\ \text{LNF-APP-}\omega}{\lambda y.y\ \Omega \in \mathcal{N}_{\emptyset,\emptyset,\bot}}\ \text{LNF-ABS-}\bot$$

La définition des formes normales locales (Fig. 4.4) est cohérente avec la définition

déjà donnée des formes normales (Fig. 3.10). En effet, les formes normales sont des formes normales locales à top-level.

Lemme 36 (Formes normales locales et structures)**.** *Si un terme est une structure et en forme normale locale alors il est en forme normale locale en mode* $\top$.
$\forall \varphi, \omega, \mu, t, \quad t \in S_\varphi \wedge t \in \mathcal{N}_{\varphi,\omega,\mu} \implies t \in \mathcal{N}_{\varphi,\omega,\top}$

Démonstration. Par induction sur la dérivation de $t \in \mathcal{N}_{\varphi,\omega,\mu}$. Soit les prémisses ne dépendent pas du mode, soit le terme est une abstraction, ce qui entre en contradiction avec le fait que ce soit une structure, soit le cas est résolu trivialement en utilisant l'hypothèse d'induction. □

Lemme 37 (Formes normales locales)**.** $t \in \mathscr{N}$ *si et seulement si* $t \in \mathcal{N}_{\mathrm{FV}(t),\emptyset,\top}$.

Démonstration. Chaque direction est obtenue par induction sur la dérivation de la normalité (locale). Traiter la règle ES-φ nécessite le lemme 36 affirmant que lorsque $u \in \mathcal{S}_\varphi$, nous avons $u \in \mathcal{N}_{\varphi,\omega,\top}$ si et seulement si $u \in \mathcal{N}_{\varphi,\omega,\perp}$. □

Cela implique que toutes les formes normales du calcul λ_{sn} sont incluses dans les formes normales locales du calcul $\lambda_{\mathsf{sn+}}$, et que le théorème de complétude énoncé pour λ_{sn} (Th. 5) reste valable pour le calcul restreint $\lambda_{\mathsf{sn+}}$.

Lemme 38 (Monotonie de la réduction $\lambda_{\mathsf{sn+}}$ par rapport au mode)**.** *Si un terme est réductible* $\lambda_{\mathsf{sn+}}$ *en mode* $\perp$*, il est réductible* $\lambda_{\mathsf{sn+}}$ *en mode* $\top$.
$\forall t, t', \varphi, \rho, \quad t \xrightarrow{\rho,\varphi,\perp}_{\mathsf{sn+}} t' \implies t \xrightarrow{\rho,\varphi,\top}_{\mathsf{sn+}} t'$.

Démonstration. Même preuve que le lemme 1 car la restriction dans LSV-BASE ne dépend pas du mode. □

4.3 Propriétés

Une fois le calcul et les formes normales locales définies, nous prouvons dans un premier temps un certain nombre de lemmes qui nous serviront par la suite à prouver la correction de la caractérisation des formes normales locales dans la section 4.3.1. Ensuite, nous prouvons la propriété du diamant dans la section 4.3.2. Pour finir, nous prouvons que les séquences de réduction de $\lambda_{\mathsf{sn+}}$ ne sont jamais plus longues que celles du calcul λ_{sn} dans la section 4.3.3.

4.3.1 Correction des formes normales locales

Nous avons tenté de démonter la correction de la caractérisation des formes normales locales (Th. 11), nous avons commencer par prouver un certain nombre de lemmes. Nous avons prouver un premier sous-théorèmes qui affirme qu'un terme ne peut pas être à la fois réductible et en forme normale locale (Th. 10 et Th. 7). Puis nous avons tenté de prouver un deuxième sous-théorèmes qui affirme qu'un terme est soit réductible, soit en forme normale locale (Th. 12 invalide), mais nous avons échoué.

Théorème 7 (Un terme ne peut pas être en forme normale locale et réductible id_x)**.** *Un terme ne peut pas être réductible avec une réduction id_x avec un x ni dans φ ni dans ω et en forme normale locale.*
$(\forall t, \varphi, \omega, \mu, x, \quad x \notin (\varphi \cup \omega) \wedge \varphi \cap \omega = \emptyset \wedge t \xrightarrow{\mathsf{id}_x, \varphi, \mu}_{\mathsf{sn+}} t \wedge t \in \mathcal{N}_{\varphi,\omega,\mu} \implies \bot)$.

Démonstration. Par induction sur t.
Pour chaque cas, nous allons devoir effectuer un raisonnement par cas sur les règles de réduction. Donc pour avoir plus de lisibilité nous traitons directement les règles de réduction. (pour les règles de réduction voir les figures 3.3). Notez que dans les cas où un lieur introduit une variable y, on a $y \neq x$ par fraîcheur donc $x \notin \varphi \cup \omega \cup \{y\}$ de plus, $y \notin \varphi$ donc $\varphi \cap (\omega \cup \{y\}) = \emptyset$.

— Cas APP-LEFT : $u\ v \xrightarrow{\mathsf{id}_x, \varphi, \mu}_{\mathsf{sn+}} u\ v$ avec $u \xrightarrow{\mathsf{id}_x, \varphi, \bot}_{\mathsf{sn+}} u$. Par définition de $u\ v \in \mathcal{N}_{\varphi,\omega,\mu}$ d'après les deux règles LNF-APP-φ et LNF-APP-ω, $u \in \mathcal{N}_{\varphi,\omega,\mu'}$. Par le lemme de monotonie sur les modes de réduction (Lem. 38) on a $u \xrightarrow{\mathsf{id}_x, \varphi, \top}_{\mathsf{sn+}} u$ donc $u \xrightarrow{\mathsf{id}_x, \varphi, \mu}_{\mathsf{sn+}} u$. Par hypothèse d'induction on prouve $\bot$.

— Cas APP-RIGHT : $u\ v \xrightarrow{\mathsf{id}_x, \varphi, \mu}_{\mathsf{sn+}} u\ v'$ avec $u \in \mathcal{S}_\varphi$ et $v \xrightarrow{\mathsf{id}_x, \varphi, \top}_{\mathsf{sn+}} v'$. Par définition de $u\ v \in \mathcal{N}_{\varphi,\omega,\mu}$ on a deux cas :
  — Cas LNF-APP-ω : avec l'hypothèse $\varphi \cap \omega = \emptyset$, u ne peut pas être dans $\mathcal{S}_\omega$ donc cette règle n'est pas possible.
  — Cas LNF-APP-φ : on a comme prémisse $v \in \mathcal{N}_{\varphi,\omega,\top}$. Par hypothèse d'induction on prouve $\bot$.

— Cas ABS-$\top$: $\lambda y.u \xrightarrow{\mathsf{id}_x, \varphi, \top}_{\mathsf{sn+}} \lambda y.u$ avec $u \xrightarrow{\mathsf{id}_y, \varphi \cup \{y\}, \top}_{\mathsf{sn+}} u$ Par définition de $\lambda y.u \in \mathcal{N}_{\varphi,\omega,\top}$ on a $u \in \mathcal{N}_{\varphi \cup \{y\},\omega,\top}$. Par hypothèse d'induction on prouve $\bot$.

— Cas ABS-$\bot$: $\lambda y.u \xrightarrow{\mathsf{id}_x, \varphi, \bot}_{\mathsf{sn+}} \lambda y.u$ avec $u \xrightarrow{\mathsf{id}_x, \varphi, \bot}_{\mathsf{sn+}} u$. Par définition de $\lambda y.u \in \mathcal{N}_{\varphi,\omega,\bot}$ on a $u \in \mathcal{N}_{\varphi,\omega \cup \{y\},\top}$. Par hypothèse d'induction avec $\omega \cup \{y\}$ on prouve $\bot$.

— Cas ES-LEFT : $u[y \setminus v] \xrightarrow{\mathsf{id}_x, \varphi, \mu}_{\mathsf{sn+}} u[y \setminus v]$ avec $u \xrightarrow{\mathsf{id}_x, \varphi, \mu}_{\mathsf{sn+}} u$. Par définition de $u[y \setminus v] \in \mathcal{N}_{\varphi,\omega,\mu}$ on a trois cas :
  — Cas LNF-ES : on a comme prémisse $u \in \mathcal{N}_{\varphi,\omega,\mu}$ Par hypothèse d'induction on prouve $\bot$.
  — Cas LNF-ES-φ : on a comme prémisse $u \in \mathcal{N}_{\varphi \cup \{y\},\omega,\mu}$. Avec le lemme d'affaiblissement des variables dans les formes normales locales (Lem. 44) on a $u \in \mathcal{N}_{\varphi,\omega \cup \{y\},\mu}$. Par hypothèse d'induction avec $\omega \cup \{y\}$ on prouve $\bot$.
  — Cas LNF-ES-ω : on a comme prémisse $u \in \mathcal{N}_{\varphi,\omega \cup \{y\},\mu}$. Par hypothèse d'induction avec $\omega \cup \{y\}$ on prouve $\bot$.

— Cas ES-LEFT-FROZEN : $u[y \setminus v] \xrightarrow{\mathsf{id}_x, \varphi, \mu}_{\mathsf{sn+}} u[y \setminus v]$ avec $u \xrightarrow{\mathsf{id}_x, \varphi \cup \{y\}, \mu}_{\mathsf{sn+}} u$ et $v \in \mathcal{S}_\varphi$. Par définition de $u[y \setminus v] \in \mathcal{N}_{\varphi,\omega,\mu}$ on a trois cas :
  — Cas LNF-ES : on a comme prémisse $u \in \mathcal{N}_{\varphi,\omega,\mu}$ et en outre $y \notin \omega$. Le lemme de monotonie des variables gelées sur les formes normales locales (Lem. 42) nous dit que $u \in \mathcal{N}_{\varphi \cup \{y\},\omega,\mu}$. Par hypothèse d'induction on prouve $\bot$.
  — Cas LNF-ES-φ : on a comme prémisse $u \in \mathcal{N}_{\varphi \cup \{y\},\omega,\mu}$. Par hypothèse d'induction on prouve $\bot$.

  — Cas LNF-ES-ω : on a comme prémisse $u \in \mathcal{N}_{\varphi,\omega\cup\{x\},\mu}$. Avec l'hypothèse $\varphi \cap \omega = \emptyset$, v ne peut pas être à la fois dans $\mathcal{S}_\varphi$ et dans $\mathcal{S}_\omega$ donc ce cas n'est pas possible.
— Cas ES-RIGHT : $u[y \backslash v] \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} u[y \backslash v]$ avec $u \xrightarrow{\mathsf{id}_y,\varphi,\mu}_{\mathsf{sn+}} u$ et $v \xrightarrow{\mathsf{id}_x,\varphi,\bot}_{\mathsf{sn+}} v$. Par définition de $u[y \setminus v] \in \mathcal{N}_{\varphi,\omega,\mu}$ on a trois cas :
  — Cas LNF-ES : on a comme prémisse $u \in \mathcal{N}_{\varphi,\omega,\mu}$ Par hypothèse d'induction on prouve $\bot$.
  — Cas LNF-ES-φ : on a comme prémisse $v \in \mathcal{N}_{\varphi,\omega,\mu'}$. Par le lemme de monotonie sur les mode de réduction (Lem. 1) on a $v \xrightarrow{\mathsf{id}_x,\varphi,\top}_{\mathsf{sn+}} v$ donc $v \xrightarrow{\mathsf{id}_x,\varphi,\mu'}_{\mathsf{sn+}} v$. Par hypothèse d'induction on prouve $\bot$.
  — Cas LNF-ES-ω : on a comme prémisse $v \in \mathcal{S}_\omega$. Le lemme sur les structures omega en forme normale (Lem. 43) nous dit que $v \in \mathcal{N}_{\varphi,\omega,\bot}$. Par hypothèse d'induction on prouve $\bot$. □

Théorème 8 (Un terme ne peut pas être en forme normale locale et réductible dB)**.** *Un terme ne peut pas être réductible avec une réduction dB et en forme normale.*
$\forall t, \varphi, \omega, \mu, t', \quad t \rightarrow_{db} t' \wedge t \in \mathcal{N}_{\varphi,\omega,\mu} \implies \bot.$

Démonstration. Les deux règles de réduction dB nous disent que t est une application $u\ v$. Et les deux règles des formes normales locales LNF-APP-φ et LNF-APP-ω ont comme prémisses que u est une structure. Or, le lemme sur les réductions dB et les réponses (Lem. 41) nous indique que u est une réponse. Et le lemme d'incompatibilité entre les structures et les réponses (Lem. 9) nous dit qu'un terme ne peut pas être à la fois une structure et une réponse. Par conséquent, il n'y a pas de règle des formes normales locales qui s'applique pour ces termes. □

Théorème 9 (Un terme ne peut pas être en forme normale locale et réductible lsv)**.** *Un terme ne peut pas être réductible avec une réduction lsv et en forme normale.*
$\forall t, t', \varphi, \omega, \mu, t, \quad \varphi \cap \omega = \emptyset \wedge t \xrightarrow{\varphi,\mu}_{lsv} t' \wedge t \in \mathcal{N}_{\varphi,\omega,\mu} \implies \bot.$

Démonstration. Par cas sur la réduction $t \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'$. Les trois règles de réduction qui s'appliquent sont LSV-BASE, LSV-σ et LSV-σ-φ qui sont uniquement sur des termes de la forme $t[x \setminus u]$ donc dans les trois cas on utilise le lemme 46 qui nous dit que $u \in \mathscr{R}$ et que $t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} t$. Pour qu'une substitution explicite soit une forme normale locale, il y a trois règles qui s'appliquent. Soit LNF-ES et le théorème 7 nous dit qu'un terme ne peut pas être réductible id_x et en forme normale locale. Soit par les règles LNF-ES-φ et LNF-ES-ω qui ont toutes les deux la prémisse que u soit une structure, ce qui entre en contradiction avec le fait que u soit une réponse par le lemme 9. □

Théorème 10 (Un terme ne peut pas être en forme normale locale et réductible)**.** *Un terme ne peut pas être réductible avec une réduction lsv ou dB et en forme normale locale.*
$\forall t, t', \varphi, \omega, \mu, \rho \in \{dB, lsv\} \quad \varphi \cap \omega = \emptyset \wedge t \xrightarrow{\rho,\varphi,\mu}_{sn+} t' \wedge t \in \mathcal{N}_{\varphi,\omega,\mu} \implies \bot.$

Démonstration. Par induction sur la dérivation de la réduction $t \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn+}} t'$.

Le théorème est découpé en 4 parties. Premièrement, on prouve qu'un terme ne peut pas être à la fois en forme normale locale et réductible id_x (Th. 7) avec un x ni dans φ ni dans ω. Ensuite, on prouve le théorème sur la réduction $\rightarrow_{\mathsf{db}}$ (Th. 8) qui est indépendant et le théorème $\xrightarrow{x,v}_{\mathsf{lsv}}$ (Th. 9) qui utilise le théorème avec id_x. Enfin, on prouve le théorème sur les réductions dB et lsv (Th. 10) en utilisant les 3 précédents.

Tous les cas sont identiques au théorème sur la réduction id_x (Th. 7) sauf le cas ES-RIGHT dans le cas où la forme normale locale vient de la règle LNF-ES-φ. Dans le théorème 7, l'hypothèse d'induction était bien sur la réduction id_x alors que la nôtre est sur un ρ. Donc, dans cette preuve, au lieu d'utiliser l'hypothèse d'induction, on utilise le théorème sur la réduction id_x (Th. 7). □

Théorème 11 (Forme normale locale)**.** *Un terme est réductible si et seulement si il n'est pas en forme normale locale.*
$(\forall t, \varphi, \omega, \mu, \quad t \notin \mathcal{N}_{\varphi,\omega,\mu} \iff (\exists \rho \in \{dB, lsv\}, t', \quad t \xrightarrow{\rho,\varphi,\mu}_{sn+} t' \vee \exists x \notin (\varphi \cup \omega), t', \quad t \xrightarrow{id_x,\varphi,\mu}_{sn+} t')$

Démonstration. De la même manière que pour la preuve de caractérisation des formes normales avec le lemme 25, on prouve qu'un terme est en forme normale locale ou réductible (Th. 12) puis on prouve qu'un terme ne peut pas être à la fois en forme normale locale et réductible (Th. 10). □

À la veille de rendre ce manuscrit, nous nous rendons compte que le théorème suivant, tel que formulé ici, est faux. Ceci n'entache pas notre confiance dans les résultats de la thèse puisque, comme cela sera présenté au chapitre suivant, un théorème équivalent bénéficie d'une formalisation dans un assistant de preuve et d'une preuve complète et vérifiée, dans une formulation toutefois subtilement différente (adaptée aux spécificités de l'outil de preuve). Nous maintenons le texte de cette preuve, avec seulement un commentaire expliquant le sous-cas problématique. À notre connaissance, tous les autres théorèmes et lemmes de la section restent valides (mis à part le théorème 11 qui hérite du défaut du théorème 12).

Théorème 12 (Forme normale locale ou réductible)**.** *Un terme est soit en forme normale locale soit réductible sois liée a une variable libre.*
$\forall t, \varphi, \omega, \mu, (\exists t', \rho \in \{dB, lsv\}, \quad t \xrightarrow{\rho,\varphi,\mu}_{sn+} t') \vee (\exists x \notin (\varphi \cup \omega), \quad t \xrightarrow{id_x,\varphi,\mu}_{sn+} t) \vee t \in \mathcal{N}_{\varphi,\omega,\mu}$

Ce théorème n'est pas prouvé !

Voici un contre-exemple : si $u = y$ et $v = \lambda a.b$ avec $b \in \omega$, alors
- il est impossible de réduire $u[y \setminus v]$ avec la règle lsv, car v n'est pas une forme normale locale avec omega vide,
- la seule réduction possible est par la règle id_b avec $b \in \omega$, ce qui contredit notre objectif d'avoir alors $b \notin \varphi \cup \omega$,
- $u[y \setminus v]$ n'est pas une forme normale locale, car aucune règle LNF ne permet d'avoir une lambda-abstraction dans une substitution utile.

Lemme 39 (Une réponse à gauche d'une application est réductible)**.** *Si le sous-terme gauche d'une application est une réponse, alors l'application est réductible avec la règle*

de réduction dB.
$(\forall t, u, \varphi, \mu, \quad t \in \mathscr{R} \implies \exists t', t\ u \xrightarrow{dB,\varphi,\mu}_{\mathsf{sn+}} t')$

Démonstration. Exactement la même preuve que le lemme 22, mais énoncée dans notre calcul $\lambda_{\mathsf{sn+}}$. □

Lemme 40 (Classification des termes en forme normale locale)**.** *Si un terme est en forme normale locale alors soit c'est une structure S_φ soit c'est une structure $\mathcal{S}_\omega$ soit c'est une réponse.*
$(\forall t, \varphi, \omega, \mu, \quad t \in \mathcal{N}_{\varphi,\omega,\mu} \implies t \in S_\varphi \vee t \in \mathcal{S}_\omega \vee t \in \mathscr{R})$

Démonstration. Par induction sur la dérivation de $t \in \mathcal{N}_{\varphi,\omega,\mu}$ (Fig. 4.4).

— Cas LNF-VAR : on a une variable $x \in \varphi \cup \omega$ donc soit x est dans φ auquel cas c'est une structure $\mathcal{S}_\varphi$ soit dans ω auquel cas c'est une structure $\mathcal{S}_\omega$ d'après la règle S-VAR.
— Cas LNF-APP-φ : on a une application $t = u\ v$ avec $u \in \mathcal{S}_\varphi$ donc d'après la règle S-APP, $u\ v \in \mathcal{S}_\varphi$.
— Cas LNF-APP-ω : on a une application $t = u\ v$ avec $u \in \mathcal{S}_\omega$ donc d'après la règle S-APP, $u\ v \in \mathcal{S}_\omega$.
— Cas LNF-ABS-$\perp$ et LNF-ABS-$\top$: on a une abstraction $t = \lambda x.u$ qui est une réponse par la règle R-ABS, donc $\lambda x.u \in \mathscr{R}$.
— Cas LNF-ES : on a une substitution explicite $t = u[x \setminus v]$ avec $u \in \mathcal{N}_{\varphi,\omega,\mu}$ donc on applique l'hypothèse d'induction :
  — Si $u \in \mathcal{S}_\varphi$ alors par la règle S-ES, $u[x \setminus v] \in \mathcal{S}_\varphi$.
  — Si $u \in \mathcal{S}_\omega$ alors par la règle S-ES, $u[x \setminus v] \in \mathcal{S}_\omega$.
  — Si $u \in \mathscr{R}$ alors par la règle R-ES, $u[x \setminus v] \in \mathscr{R}$.
— Cas LNF-ES-φ : on a une substitution explicite $t = u[x \setminus v]$ avec $u \in \mathcal{N}_{\varphi\cup\{x\},\omega,\mu}$ et $v \in S_\varphi$ donc on applique l'hypothèse d'induction :
  — Si $u \in \mathcal{S}_{\varphi\cup\{x\}}$ alors par la règle S-ES-S, $t \in \mathcal{S}_\varphi$.
  — Si $u \in \mathcal{S}_\omega$ alors par la règle S-ES, $t \in \mathcal{S}_\omega$.
  — Si $u \in \mathscr{R}$ alors par la règle R-ES, $t \in \mathscr{R}$.
— Cas LNF-ES-ω : on a une substitution explicite $t = u[x \setminus v]$ avec $u \in \mathcal{N}_{\varphi,\omega\cup\{x\},\mu}$ et $v \in \mathcal{S}_\omega$ donc on applique l'hypothèse d'induction :
  — Si $u \in \mathcal{S}_\varphi$ alors par la règle S-ES, $t \in \mathcal{S}_\varphi$.
  — Si $u \in \mathcal{S}_{\omega\cup\{x\}}$ alors par la règle S-ES-S, $t \in \mathcal{S}_\omega$.
  — Si $u \in \mathscr{R}$ alors par la règle R-ES, $t \in \mathscr{R}$. □

Lemme 41 (Règle dB et réponse)**.** *Si une application est réductible dB cela implique que le membre gauche de l'application est une réponse.*
$\forall t, u, v, \quad t\ u \to_{\mathsf{db}} v \implies t \in \mathscr{R}$

Démonstration. Par induction sur la dérivation de $t\ u \to_{\mathsf{db}} v$.

— Cas DB-BASE : On a $t = \lambda x.w$ donc la règle R-ABS nous dit que $t \in \mathscr{R}$.
— Cas DB-σ : On a $t = a[x \setminus w]$ avec $t = a\ z \to_{\mathsf{db}} v$. Par hypothèse d'induction on a $a \in \mathscr{R}$. Donc la règle R-ES nous dit que $a[x \setminus w] \in \mathscr{R}$. □

Lemme 42 (Monotonie des variables gelées dans les formes normales locales)**.** *Si un terme est en forme normale locale pour un certain φ, alors il l'est toujours pour un*

φ étendu.
$\forall t, \varphi, \omega, \mu, x, \quad x \notin \omega \wedge t \in \mathcal{N}_{\varphi,\omega,\mu} \implies t \in \mathcal{N}_{\varphi \cup \{x\},\omega,\mu}$

Démonstration. Preuve triviale car il n'y a pas de prémisse négative dans la définition des formes normales locales, donc faire grandir φ ne peut pas mettre en défaut une règle. □

Lemme 43 (Une structure omega est en forme normale locale)**.** *Si un terme est une structure $\mathcal{S}_\omega$, alors le terme est en forme normale locale pour ce même ω.*
$\forall t, \varphi, \omega, \mu, \quad t \in \mathcal{S}_\omega \implies t \in \mathcal{N}_{\varphi,\omega,\mu}$

Démonstration. Par induction sur la dérivation de $t \in \mathcal{S}_\omega$. Il existe une correspondance directe entre les règles des structures et les règles correspondant aux formes normales locales.

- S-VAR → LNF-VAR
- S-APP → LNF-APP-ω
- S-ES → LNF-ES
- S-ES-S → LNF-ES-ω □

Lemme 44 (Affaiblissement des variables dans les formes normales locales)**.** *Soit un terme en forme normale locale pour certains ensembles φ et ω. Si on déplace une variable de l'ensemble φ vers l'ensemble ω le terme reste en forme normale locale.*
$\forall t, \varphi, \omega, \mu, x, \quad t \in \mathcal{N}_{\varphi \cup \{x\},\omega,\mu} \implies t \in \mathcal{N}_{\varphi,\omega \cup \{x\},\mu}$

Démonstration. Par induction sur le terme t :

- Var : On a $t = y \in \mathcal{N}_{\varphi \cup \{x\},\omega,\mu}$ avec la prémisse $y \in \varphi \cup \{x\} \cup \omega = \varphi \cup \omega \cup \{x\}$. Donc $y \in \mathcal{N}_{\varphi,\omega \cup \{x\},\mu}$.
- Abs : On a $t = \lambda y.u \in \mathcal{N}_{\varphi \cup \{x\},\omega,\mu}$. Par définition :
  - Cas LNF-ABS-$\top$, on a comme prémisse $u \in \mathcal{N}_{\varphi \cup \{y,x\},\omega,\top}$. Par hypothèse d'induction sur u on a $u \in \mathcal{N}_{\varphi \cup \{y\},\omega \cup \{x\},\top}$. Donc par la règle LNF-ABS-$\top$ on a $\lambda y.u \in \mathcal{N}_{\varphi,\omega \cup \{x\},\mu}$.
  - Cas LNF-ABS-$\bot$, on a comme prémisse $u \in \mathcal{N}_{\varphi \cup \{x\},\omega \cup \{y\},\bot}$. Par hypothèse d'induction sur u on a $u \in \mathcal{N}_{\varphi,\omega \cup \{x,y\},\bot}$. Donc par la règle LNF-ABS-$\bot$ on a $\lambda y.u \in \mathcal{N}_{\varphi,\omega \cup \{x\},\mu}$.
- App : On a $u\ v \in \mathcal{N}_{\varphi \cup \{x\},\omega,\mu}$. Par définition :
  - Cas LNF-APP-φ, on a comme prémisses $u \in \mathcal{N}_{\varphi \cup \{x\},\omega,\mu'}$, $u \in \mathcal{S}_{\varphi \cup \{x\}}$ et $v \in \mathcal{N}_{\varphi \cup \{x\},\omega,\top}$. On a par hypothèse d'induction sur u, $u \in \mathcal{N}_{\varphi,\omega \cup \{x\},\mu'}$. Par disjonction de cas sur le variable de tête de $u \in \mathcal{S}_{\varphi \cup \{x\}}$:
    - Si $u \in \mathcal{S}_\varphi$ alors par hypothèse d'induction sur v on a $v \in \mathcal{N}_{\varphi,\omega \cup \{x\},\top}$. Donc par la règle LNF-APP-φ on a $u\ v \in \mathcal{N}_{\varphi,\omega \cup \{x\},\mu}$.
    - Si $u \in \mathcal{S}_{\{x\}}$ alors on a $u \in \mathcal{S}_{\omega \cup \{x\}}$ Donc par la règle LNF-APP-ω on a $u\ v \in \mathcal{N}_{\varphi,\omega \cup \{x\},\mu}$.
  - Cas LNF-APP-ω, on a comme prémisses $u \in \mathcal{N}_{\varphi \cup \{x\},\omega,\mu}$, $u \in \mathcal{S}_\omega$. Par le lemme de monotonie sur les structures (Lem. 2) on a $u \in \mathcal{S}_{\omega \cup \{x\}}$. Par hypothèse d'induction on a $u \in \mathcal{N}_{\varphi,\omega \cup \{x\},\mu}$. Donc par la règle LNF-APP-ω on a $u\ v \in \mathcal{N}_{\varphi,\omega \cup \{x\},\mu}$.
- ES : On a $u[y \setminus v] \in \mathcal{N}_{\varphi \cup \{x\},\omega,\mu}$. Par définition :

— Cas LNF-ES, on a comme prémisse $u \in \mathcal{N}_{\varphi\cup\{x\},\omega,\mu}$. Par hypothèse d'induction on a $u \in \mathcal{N}_{\varphi,\omega\cup\{x\},\mu}$. Donc par la règle LNF-ES on a $u[y \setminus v] \in \mathcal{N}_{\varphi,\omega\cup\{x\},\mu}$.
— Cas LNF-ES-φ, on a comme prémisses $u \in \mathcal{N}_{\varphi\cup\{x,y\},\omega,\mu}$, $v \in \mathcal{N}_{\varphi\cup\{x\},\omega,\mu'}$ et $v \in \mathcal{S}_{\varphi\cup\{x\}}$. On a par hypothèse d'induction sur u, $u \in \mathcal{N}_{\varphi\cup\{y\},\omega\cup\{x\},\mu}$. Par disjonction de cas sur le variable de tête de $v \in \mathcal{S}_{\varphi\cup\{x\}}$:
  — Si $v \in \mathcal{S}_\varphi$ alors par hypothèse d'induction sur v on a $v \in \mathcal{N}_{\varphi,\omega\cup\{x\},\mu'}$. Donc par la règle LNF-ES-φ on a $u[y \setminus v] \in \mathcal{N}_{\varphi,\omega\cup\{x\},\mu}$.
  — Si $v \in \mathcal{S}_{\{x\}}$ alors on a $v \in \mathcal{S}_{\omega\cup\{x\}}$. On applique l'hypothèse d'induction une deuxième fois sur u mais sur la variable y ce qui nous donne $u \in \mathcal{N}_{\varphi,\omega\cup\{y,x\},\mu}$. Donc par la règle LNF-ES-ω on a $u[y \setminus v] \in \mathcal{N}_{\varphi,\omega\cup\{x\},\mu}$.
— Cas LNF-ES-ω, on a comme prémisses $u \in \mathcal{N}_{\varphi\cup\{x\},\omega\cup\{y\},\mu}$, $u \in \mathcal{S}_\omega$. Par hypothèse d'induction on a $u \in \mathcal{N}_{\varphi,\omega\cup\{x,y\},\mu}$. Donc par la règle LNF-ES-ω on a $u[y \setminus v] \in \mathcal{N}_{\varphi,\omega\cup\{x\},\mu}$. □

Lemme 45 (Monotonie du mode dans les formes normales locales)**.** *Si un terme est en forme normale en mode $\top$ alors il est aussi en mode $\bot$.*
$\forall t, \varphi, \omega, \quad t \in \mathcal{N}_{\varphi,\omega,\top} \implies t \in \mathcal{N}_{\varphi,\omega,\bot}$

Démonstration. Par induction sur la dérivation de $t \in \mathcal{N}_{\varphi,\omega,\top}$. Le seul cas subtil est le cas LNF-ABS-$\top$, pour lequel il est nécessaire d'utiliser le lemme d'affaiblissement des variables (Lem. 44) avant d'appliquer LNF-ABS-$\bot$. □

Lemme 46 (Règle *lsv* et réponse)**.** *Si une substitution explicite est réductible lsv, alors le sous-terme de gauche est réductible ID, et le sous-terme de droite est une réponse. .*
$\forall t, t', u, v, x, \varphi, \mu, \quad t[x \setminus v] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t' \implies v \in \mathscr{R} \wedge t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} t$

Démonstration. Par induction sur la dérivation de $t[x \setminus v] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'$.
— Cas LSV-BASE : On a $t \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\mu}_{\mathsf{sn+}} t'$ et $v \in \mathit{Val}$. D'une part par le lemme 14 on a $t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} t$ et d'autre part une valeur est une réponse donc on a $v \in \mathscr{R}$.
— Cas LSV-σ : On a $v = u[y \setminus w]$ avec $t[x \setminus u] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'$. Par hypothèse d'induction sur $t[x \setminus u] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t'$ on a $u \in \mathscr{R}$ et $t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} t$. Et d'après la règle R-ES, $v \in \mathscr{R}$. Et par la règle ES-LEFT on a $t[x \setminus u[y \setminus w]] \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} t[x \setminus u[y \setminus w]]$.
— Cas LSV-σ-φ : Exactement le même raisonnement que sur le cas précédent mais avec la règle ES-LEFT-FROZEN. □

4.3.2 Propriété du diamant

Nous avons tenté de prouver la propriété du diamant sur notre premier calcul λ_{sn} et nous sommes tombés sur un contre-exemple. C'est à partir de ce contre-exemple que l'idée de la forme normale locale nous est venue.

Comme mentionné précédemment, dans λ_{sn}, les termes peuvent être substitués dès qu'il s'agit de valeur, ce qui peut entraîner des duplications de calculs et même rompre la confluence. Les choses sont cependant différentes dans notre calcul restreint $\lambda_{\mathsf{sn+}}$.

Considérons, par exemple, le contre-exemple de la confluence donné dans la figure 3.19. Étant donné le terme $x[x \setminus \lambda y.(\lambda z.z)\ y]$, le calcul λ_{sn} permet de substituer la réponse $\lambda y.(\lambda z.z)\ y$ ou de la réduire plus loin. Le calcul restreint $\lambda_{\mathsf{sn+}}$ empêche la substitution de $\lambda y.(\lambda z.z)\ y$ tant que cette réponse n'est pas en forme normale locale. Donc, $\lambda_{\mathsf{sn+}}$ ne permet qu'un seul des deux chemins qui sont possibles dans λ_{sn}, ce qui est suffisant pour restaurer la confluence. En fait, $\lambda_{\mathsf{sn+}}$ jouit même de la propriété du diamant, c'est-à-dire la confluence en une seule étape.

Théorème 13 (Diamant)**.** *Supposons que* $t \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_1$ *et* $t \xrightarrow{\rho_2,\varphi,\mu}_{\mathsf{sn+}} t_2$ *avec* $t_1 \neq t_2$*. Supposons que, si* ρ_1 *et* ρ_2 *sont* sub *ou* id*, alors ils s'appliquent à des variables distinctes et que, si* ρ_1 *ou* ρ_2 *est* sub*, alors il s'applique à une variable qui n'est pas dans* φ*. Alors il existe* t' *tel que* $t_1 \xrightarrow{\rho_2,\varphi,\mu}_{\mathsf{sn+}} t'$ *et* $t_2 \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t'$*.*

Démonstration. L'énoncé du théorème du diamant doit être d'abord généralisé de sorte que les étapes $t \to t_1$ et $t \to t_2$ puissent utiliser la réduction principale $\xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}}$ ou les réductions auxiliaires $\to_{\mathsf{db}}$ et $\xrightarrow{\varphi,\mu}_{\mathsf{lsv}}$. La preuve est alors une induction sur la taille de t, avec un raisonnement fastidieux par cas sur la forme de t et sur la dernière règle d'inférence appliquée sur chaque côté. La plupart des cas sont plutôt sans surprise. Nous présentons ici des sous-cas sélectionnés s'appliquant à la forme $t[x \setminus u]$, qui illustrent les principales subtilités rencontrées dans l'ensemble de la preuve.

— Cas ES-LEFT vs ES-RIGHT. Plus précisément, nous avons
  — $t[x \setminus u] \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_1[x \setminus u]$ par la règle ES-LEFT avec $t \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_1$, et
  — $t[x \setminus u] \xrightarrow{\rho_2,\varphi,\mu}_{\mathsf{sn+}} t[x \setminus u_2]$ par la règle ES-RIGHT avec $t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} t$ et $u \xrightarrow{\rho_2,\varphi,\top}_{\mathsf{sn+}} u_2$.

  Remarquez que, par la fraîcheur de x, la règle ρ_1 ne peut pas être une réduction sub ou id s'appliquant à la variable x. Alors nous pouvons appliquer notre hypothèse d'induction à $t \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_1$ et $t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} t$ pour en déduire que $t_1 \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} t_1$. Avec la règle ES-RIGHT nous déduisons alors $t_1[x \setminus u] \xrightarrow{\rho_2,\varphi,\mu}_{\mathsf{sn+}} t_1[x \setminus u_2]$. En outre, la règle ES-LEFT appliquée à $t \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_1$ donne immédiatement $t[x \setminus u_2] \xrightarrow{\rho_2,\varphi,\mu}_{\mathsf{sn+}} t_1[x \setminus u_2]$ et conclut le cas.

— Cas ES-LEFT-FROZEN vs ES-RIGHT. Ce cas est superficiellement similaire au précédent mais ajoute quelques subtilités, car nous avons maintenant
  — $t[x \setminus u] \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_1[x \setminus u]$ par la règle ES-LEFT-FROZEN avec $u \in \mathcal{S}_\varphi$ et $t \xrightarrow{\rho_1,\varphi\cup\{x\},\mu}_{\mathsf{sn+}} t_1$, et
  — $t[x \setminus u] \xrightarrow{\rho_2,\varphi,\mu}_{\mathsf{sn+}} t[x \setminus u_2]$ par la règle ES-RIGHT avec $t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} t$ et $u \xrightarrow{\rho_2,\varphi,\mu}_{\mathsf{sn+}} u_2$.

  La réduction $t \xrightarrow{\rho_1,\varphi\cup\{x\},\mu}_{\mathsf{sn+}} t_1$ utilise les variables gelées $\varphi \cup \{x\}$ plutôt que juste φ. Pour appliquer l'hypothèse d'induction, nous devons d'abord affaiblir $t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} t$ en $t \xrightarrow{\mathsf{id}_x,\varphi\cup\{x\},\mu}_{\mathsf{sn+}} t$ (Lem. 48 plus bas). L'hypothèse d'induction donne alors $t_1 \xrightarrow{\mathsf{id}_x,\varphi\cup\{x\},\mu}_{\mathsf{sn+}} t_1$, qui doit ensuite être renforcée en $t_1 \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} t_1$ (Lem. 49 plus bas) pour obtenir $t_1[x \setminus u] \xrightarrow{\rho_2,\varphi,\mu}_{\mathsf{sn+}} t_1[x \setminus u_2]$ avec la règle ES-RIGHT.

  Enfin, pour clore le cas avec la règle ES-LEFT-FROZEN nous devons également justifier que $u_2 \in \mathcal{S}_\varphi$, ce que nous faisons en utilisant une propriété de stabilité des structures (Lem. 47 plus bas).

— Cas ES-LEFT-FROZEN vs lsv. Ces cas ne sont pas compatibles. En effet, ES-LEFT-FROZEN s'applique à un terme $t[x \setminus u]$ où $u \in \mathcal{S}_\varphi$, alors que lsv et les règles auxiliaires associées s'appliquent à un terme $t[x \setminus u]$ où u est une réponse.

— Cas ES-LEFT vs LSV-BASE. Dans ce cas, notre terme de départ a nécessairement la forme $t[x \setminus v]$ avec $v \in \mathscr{R}$ et nous avons

  — $t[x \setminus v] \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_1[x \setminus v]$ par la règle ES-LEFT avec $t \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_1$, et

  — $t[x \setminus v] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t_2[x \setminus v]$ par la règle LSV-BASE avec $t \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\mu}_{\mathsf{sn}} t_2$.

  En raison de la fraîcheur, la variable x ne peut apparaître dans ρ_1 ni dans φ. Alors, par hypothèse d'induction, on obtient t_3 tel que $t_1 \xrightarrow{\mathsf{sub}_{x\setminus v},\varphi,\mu}_{\mathsf{sn}} t_3$ et $t_2 \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn}} t_3$. En appliquant la règle LSV-BASE à la première réduction, nous obtenons $t_1[x \setminus v] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t_3[x \setminus v]$, et en appliquant la règle ES-LEFT à la seconde réduction, nous obtenons $t_2[x \setminus v] \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn}} t_3[x \setminus v]$ et nous concluons.

— Cas ES-LEFT vs LSV-σ. Dans ce cas, notre terme de départ a nécessairement la forme $t[x \setminus u[y \setminus w]]$ et nous avons

  — $t[x \setminus u[y \setminus w]] \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_1[x \setminus u[y \setminus w]]$ par la règle ES-LEFT avec $t \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_1$, et

  — $t[x \setminus u[y \setminus w]] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t_2[y \setminus w]$ par la règle LSV-σ avec $t[x \setminus u] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t_2$.

  À partir de la prémisse $t \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_1$, par la règle ES-LEFT nous déduisons $t[x \setminus u] \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_1[x \setminus u]$. Puisque le terme $t[x \setminus u]$ est plus petit que $t[x \setminus u[y \setminus w]]$, nous appliquons l'hypothèse d'induction pour obtenir un terme t_3 tel que $t_1[x \setminus u] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t_3$ et $t_2 \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_3$.

  À partir de la première réduction, par la règle LSV-σ nous déduisons $t_1[x \setminus u[y \setminus w]] \xrightarrow{\varphi,\mu}_{\mathsf{lsv}} t_3[y \setminus w]$ et de la seconde réduction, par la règle ES-LEFT nous déduisons $t_2[y \setminus w] \xrightarrow{\rho_1,\varphi,\mu}_{\mathsf{sn+}} t_3[y \setminus w]$. Ces deux réductions finales ferment le diamant sur le terme $t_3[y \setminus w]$.

□

Les faits suivants ont été utilisés dans la preuve du théorème 13, mais ils ont un intérêt en soi, en particulier le lemme 47 de la stabilité des structures par réduction.

Lemme 47 (Stabilité des structures)**.** *Supposons que $t \in \mathcal{S}_\varphi$ et $t \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn+}} t'$. Supposons que, si ρ est sub, elle s'applique à une variable qui n'est pas dans φ. Alors $t' \in \mathcal{S}_\varphi$.*

Démonstration. Par induction sur la dérivation de $t \in \mathcal{S}_\varphi$, par cas sur la dernière règle d'inférence de la dérivation de $t \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn+}} t'$. □

La restriction lorsque ρ est de type sub a un objectif simple (mais crucial) : exclure une substitution directe de la variable de tête de la structure t, tels que $x\ u \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn+}} (\lambda y.y)\ u$ avec $\rho = \mathsf{sub}_{x \setminus \lambda y.y}$ et $\varphi = \{x\}$. En effet, toute structure doit doit avoir en tête une variable gelée, c'est-à-dire une variable dont on sait qu'elle ne peut être substituée. On peut vérifier qu'aucune réduction lsv valide ne peut être reliée à ce cas interdit de sub.

Lemme 48 (Affaiblissement de φ)**.** *Supposons que $t \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn+}} t'$ et $\varphi \subseteq \varphi'$. Supposons que, si ρ est sub, il s'applique à une variable qui n'est pas dans φ'. Alors $t \xrightarrow{\rho,\varphi',\mu}_{\mathsf{sn+}} t'$.*

Démonstration. Par induction sur la dérivation de $t \xrightarrow{\rho,\varphi',\mu}_{\mathsf{sn+}} t'$. □

Lemme 49 (Renforcement de φ). *Si $t \xrightarrow{\mathsf{id}_x,\varphi\cup\{x\},\mu}_{\mathsf{sn+}} t'$, alors $t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} t'$.*

Démonstration. Par induction sur la dérivation de $t \xrightarrow{\rho,\varphi',\mu}_{\mathsf{sn+}} t'$. Il y a deux cas subtils avec les règles APP-RIGHT et ES-LEFT-FROZEN qui reposent toutes deux sur la propriété suivante : Si $t \in \mathcal{S}_{\varphi\cup\{x\}}$ et $t \notin \mathcal{S}_{\varphi}$, alors $t \xrightarrow{\mathsf{id}_x,\varphi,\mu}_{\mathsf{sn+}} t$. Cette propriété est prouvée par induction sur $t \in \mathcal{S}_{\varphi\cup\{x\}}$. □

4.3.3 Optimalité relative

Le calcul $\lambda_{\mathsf{sn+}}$ est une restriction de λ_{sn} qui exige que les termes soient réduits à la forme normale locale avant de pouvoir être substitués (Fig. 4.1). Cette réduction anticipée n'est jamais inutile, car λ_{sn} (et a fortiori son sous-ensemble $\lambda_{\mathsf{sn+}}$) ne réduit que les radicaux nécessaires, c'est-à-dire ceux qui doivent être réduits dans toute réduction à la forme normale. En conséquence, les séquences de réductions de $\lambda_{\mathsf{sn+}}$ ne sont jamais plus longues que les séquences équivalentes dans λ_{sn}. Au contraire, en imposant certaines réductions avant que les termes ne soient substitués (i.e., potentiellement dupliqués), cette stratégie produit dans de nombreux cas des séquences de réduction qui sont strictement plus courtes que celles données par la stratégie d'appel par nécessité forte d'origine [9].

Théorème 14 (Minimalité). *Avec $t' \in \mathcal{N}_{\varphi}$, si $t \to_{\mathsf{sn}}^{n} t'$ et $t \to_{\mathsf{sn+}}^{m} t'$ alors $m \leqslant n$.*

Remarquez que ce résultat de minimalité est relatif à λ_{sn}. La réduction des séquences de $\lambda_{\mathsf{sn+}}$ ne sont pas nécessairement optimales par rapport au calcul λ_{c} ou au lambda-calcul, qui n'ont pas la contrainte de ne réduire que des radicaux nécessaires. Par exemple, ni $\lambda_{\mathsf{sn+}}$ ni λ_{sn} ne permettent de réduire r dans le terme $(\lambda x.x\ (x\ a))\ (\lambda y.y\ r)$ avant sa duplication.

4.4 Conclusion

Dans ce chapitre, nous avons présenté une restriction de notre calcul λ_{sn} appelée $\lambda_{\mathsf{sn+}}$. Ce nouveau calcul ressemble à une stratégie, mais nous ne voulons pas imposer un ordre des réductions qui ne joue pas un rôle «utile» relativement aux longueurs des séquences des séquences de réductions. De même, les stratégies les plus connues, telles que la stratégie d'appel par valeur, laissent souvent un certain degré de liberté dans leur ordre d'évaluation des arguments d'une abstraction.

La restriction de notre calcul $\lambda_{\mathsf{sn+}}$ porte sur le moment auquel nous avons le droit d'appliquer une substitution. Dans λ_{sn} on pouvait appliquer une substitution dès lors que le corps de la substitution était une réponse. Le problème est que cette restriction était trop faible et entraînait dans certains cas une duplication de calcul. Nous ne voulions substituer que des termes qui ne pouvaient plus être réduits dans notre calcul. La notion de forme normale paraissait être toute trouvée. Le problème est que le mode rendait la forme normale trop contraignante et ne nous permettait pas d'arriver à la forme normale du terme. C'est pourquoi nous avons défini une

nouvelle forme normale relâchée que nous avons appelée forme normale locale. Cette forme normale locale utilise de nouvelles règles et un ensemble de variables gelées supplémentaires. Une fois cette forme normale caractérisée, nous nous en sommes servis pour restreindre la règle LSV-BASE pour qu'il ne soit possible de substituer que les réponses en forme normale locale (Fig. 4.1). La notion de forme normale locale de $\lambda_{\mathsf{sn}+}$ généralise la notion de forme normale de λ_{sn}.

Nous avons prouvé la propriété du diamant (Th. 13) et par conséquent, la confluence. Enfin, nous avons évoqué théorème de minimalité (Th. 14) qui nous dit que $\lambda_{\mathsf{sn}+}$ met au plus autant de pas de réduction que λ_{sn} pour réduire un terme. Le calcul $\lambda_{\mathsf{sn}+}$ définit donc la stratégie optimale du calcul λ_{sn}. Nous avons également établi une forme de lien entre les formes normales et les formes normales locales (Lem. 37) en montrant que les formes normales et les formes normales locales en mode $\top$ sont équivalentes.

Nous avons tenté de prouver la correction de la caractérisation des formes normales locales (Th. 11) de la même manière que pour la caractérisation des formes normales. Mais au dernier moment de la finalisation du manuscrit, nous avons détecté un problème dans le théorème 12, dont le résultat est invalide. Pour mémoire, ce théorème énonçait que dans un contexte donné, tout terme était soit réductible, soit en forme normale locale. Avec lui, le théorème 11 énonçant l'équivalence entre la caractérisation des formes normales locales et l'irréductibilité est donc invalide également.

Voici un contre-exemple à ce théorème 12 : $a[a \setminus \lambda z.x[x \setminus \lambda y.z]]$.

Le contenu de la substitution est une abstraction et n'est pas réductible, et devrait donc pouvoir être substitué. Mais elle n'est pas capturée par LNF, car $x[x \setminus \lambda y.z]$ n'est pas LNF car $\lambda y.z$ n'est pas une structure. Et donc en passant, $x[x \setminus \lambda y.z]$ n'est pas réductible dans ce contexte car l'argument $\lambda y.z$ de la substitution dépend de z, et z n'est pas gelée.

À notre connaissance, tous les autres résultats du chapitre restent bien valides, et en particulier :

— Le fait que la caractérisation des formes normales locales ne capture bien que des termes irréductibles (théorème 10, qui forme l'autre sens de l'équivalence énoncée par le théorème 11).
— Les théorèmes 7, 8, 9 et leurs lemmes, sur lesquels repose la démonstration du théorème 10.
— La propriété du diamant (Th. 13).
— La propriété de minimalité (Th. 14).

Ainsi, notre caractérisation des termes irréductibles dans un certain contexte par la notion de forme normale locale est incomplète : elle ne capture que des termes irréductibles (Th. 10), mais en « oublie » certains (invalidité du Th. 12). En conséquence, la règle LSV-BASE du calcul $\lambda_{\mathsf{sn}+}$ (la seule règle qui fasse usage de la caractérisation des formes normales locales) autorise moins de réductions qu'attendu.

Ceci a une autre conséquence, qui n'invalide pas de théorème explicitement énoncé dans ce manuscrit, mais va à l'encontre de notre objectif pour ce chapitre : le calcul $\lambda_{\mathsf{sn}+}$, tel que défini dans ce chapitre, n'atteint pas toutes les formes normales voulues. Dans certains cas, la réduction sera bloquée avant cela par la dépendance de la règle LSV-BASE à la notion incomplète de forme normale locale. On pourrait imaginer une

correction très simple, qui consisterait à retirer la mention aux formes normales locales dans la règle LSV-BASE, et la remplacer par une indication explicite d'irréductibilité (c'est-à-dire par la propriété que nous souhaitions capturer avec les formes normales locales). Cette solution est cependant problématique puisqu'elle introduit une prémisse négative dans les règles d'inférence définissant la réduction. Notez cependant que cette solution respecterait l'esprit de la machine abstraite présentée au chapitre 6, dans laquelle une substitution est réalisée lorsque son argument a été réduit autant que possible.

Une correction plus satisfaisante serait de compléter les règles caractérisant les formes normales locales. Cette approche semble possible, mais un peu trop invasive pour être réalisée et consciencieusement vérifiée en quelques jours. Aussi cette investigation est remise à plus tard.

Chapitre 5

Formalisation

Nous avons utilisé l'assistant de preuve Coq pour nos premières tentatives de formalisation de nos résultats. Nous avons expérimenté à la fois avec l'approche locally nameless [15] et la syntaxe abstraite d'ordre supérieur paramétrique [16]. Bien que nous aurions pu réussir à utiliser l'approche locally nameless, le fait de devoir gérer manuellement les liens était beaucoup trop fastidieux. Ainsi, nous avons tourné notre attention vers un système formel dédié, Abella [6], dans l'espoir que cela rendrait les preuves syntaxiques plus simples. Ce chapitre décrit notre expérience avec cet outil.

Dans ce chapitre, nous commençons par présenter la formalisation des termes avec l'assistant de preuve Abella dans la section 5.1, qui nous permet également de présenter une partie de son fonctionnement. Puis, dans la section 5.2, nous présentons la formalisation des réductions tout en continuant la présentation de l'assistant de preuve. Et pour finir dans la section 5.3 nous prouvons la correction de notre calcul formalisé.

5.1 Présentation d'Abella

Abella est un assistant de preuve qui cible le raisonnement sur les langages et les règles d'inférence, mais avec un fonctionnement qui rappelle celui de Coq.

Dans un premier temps, nous avons essayé de formaliser notre calcul dans Coq, mais nous avons rencontré des difficultés avec la représentation et la gestion des variables. Comme nous l'avons vu dans le chapitre 2, il existe plusieurs façons de représenter les variables dans les lambda-termes et, elles ont toutes besoin d'une gestion particulière pour éviter toute capture. Nous nous sommes tournés vers un autre assistant de preuve, Abella. Abella introduit un nouveau quantificateur qui nous permet de parler plus facilement des variables fraîches, ce qui a réglé tous les problèmes que nous avions dans Coq.

Cet assistant de preuve fonctionne avec deux logiques. Premièrement, une logique de spécification minimale nous permet de définir les règles sur lesquelles nous allons raisonner, et deuxièmement, une logique de raisonnement va nous permettre de raisonner sur cette spécification.

5.1.1 Règle d'inférence

Le type `o` est le type des formules dans la logique de spécification et la syntaxe `kind ... type` permet de définir de nouveaux types atomiques. On commence par introduire un nouveau type atomique `trm` qui est le type des termes grâce auquel on définit le type de tous les éléments dont nous avons besoin.

```
kind trm  type.

type app  trm -> trm -> trm.
type abs  (trm -> trm) -> trm.
type es   (trm -> trm) -> trm -> trm.
```

Les termes sont construits par `app`, `abs` et `es`, de la même manière que dans leur version papier (Fig. 2.5.1). La seul différence vient du fait que les lieurs sont gérés par l'aspect ordre supérieur des termes. Par conséquent, il n'y a pas de constructeur pour les variables. Une application est un terme si ses deux sous-termes sont également des termes. Les abstractions sont considérées comme des termes si, lorsque l'on applique un terme au corps de l'abstraction, le tout reste un terme. La substitution explicite fonctionne comme une combinaison des deux constructeurs précédents. Le premier sous-terme doit satisfaire les mêmes conditions que le corps de l'abstraction, et le deuxième doit être un terme, comme pour l'application.

Pour mieux comprendre, prenons un exemple. Dans la version papier, on utilise les noms de variables et la convention de Barendregt pour gérer les liens, ce qui donne des termes comme celui-ci : $\lambda x.t\ x$. Dans ce terme, la variable x qui est dans le corps de l'abstraction est liée au x du lambda. Dans Abella, ce terme serait traduit par une fonction `abs x\ app T x` (implicitement parenthésé en `abs (x\(app T x))`), ce qui élimine les problèmes de nommage.

On définit le prédicat `pure` qui signifie qu'un terme est pur, c'est-à-dire qu'il ne contient aucune substitution explicite. Les termes purs sont ceux qui n'utilisent pas le constructeur `es`. Ces termes apparaissent au début du manuscrit (Sec. 2.1.1).

```
type pure  trm -> o.

pure (app U V) :- pure U, pure V.
pure (abs U) :- pi x\ pure x => pure (U x).
```

Les définitions des prédicats correspondent à des règles d'inférence. Le caractère :- représente la barre de règle. On note ici la première utilisation de `pi`, qui est le quantificateur universel utilisé par la logique de spécification. On traduit `pi x\ P` par $\forall x, P$. Donc, `abs U` est pur si et seulement si pour tout `x` qui est pur, `U x` est pur. Ici, `U` est considéré comme une fonction qui prend un terme et le met à tous les endroits correspondants.

Dans la logique de raisonnement et dans les preuves, les prédicats qui sont définis dans la logique de spécification sont entre accolades. Par exemple, le terme $(\lambda x.x)\ (\lambda x.x)$ est pur car il ne contient pas de substitution explicite.

```
{pure(app (abs x\ x)(abs x\ x))}.
```

Mais le terme $y[y \setminus x]$ ne l'est pas car il contient une substitution explicite.

```
forall x, {pure(es (y\y) x)} -> false.
```

On a un exemple de terme pur ci-dessous qui nous dit que si `x` est un terme pur, alors `app (abs y\ y) x` (c'est-à-dire $(\lambda y.y)\ x$) l'est aussi.

```
forall x, {pure x} -> {pure (app (abs y\y) x)}.
```

Le quantificateur `nabla` est un quantificateur qui introduit une variable nominale fraîche par rapport à tout ce qui a été introduit avant. Par exemple, si on introduit une variable x avec `nabla`, puis un terme y avec `exists` , alors y peut être égal à x car la quantification existentielle arrive après le `nabla`. En revanche, quelque soit le y qui a été introduit avant le `nabla`, il ne peut pas être égal à x.

```
nabla x, exists y, x = y.
```

```
forall y, nabla x, x = y -> false.
```

Donc, si l'on veut raisonner sur une variable fraîche, il nous suffit d'introduire cette variable avec `nabla` après avoir introduit les éléments par rapport auxquels on désire la fraîcheur. Par exemple, si on veut un `x` frais par rapport à un terme `T`, on écrira :

```
forall T, nabla x, ...
```

5.1.2 Mécanisme de preuve de base

Pour illustrer le fonctionnement d'Abella, nous utilisons un lemme de préservation de la pureté par la bêta-réduction classique. Cela signifie que si l'on a une abstraction t et un terme u, tous les deux purs, alors la substitution $t\{u\}$ reste pure.

```
Theorem test_pure_beta :
  forall t u, {pure (abs t)} ->
  {pure u} -> {pure (t u)}.
```

Après introduction des hypothèses, le contexte et de preuve est comme suit.

```
Variables: t u
H1 : {pure (abs t)}
H2 : {pure u}
============================
 {pure (t u)}
```

Pour faire la preuve, nous allons effectuer un raisonnement par cas sur le fait que `abs t` soit pur. Ce qui nous donne une nouvelle hypothèse `H3`.

```
test_pure_beta < case H1.

Variables: t u
H2 : {pure u}
H3 : {pure n1 |- pure (t n1)}
============================
 {pure (t u)}
```

En utilisant la définition de la pureté d'une abstraction, nous savons qu'une abstraction est pure si et seulement si le corps de l'abstraction appliqué à un terme pur est également pur. Autrement dit, comme le corps de l'abstraction est une fonction dans Abella, pour tester sa pureté nous devons le transformer en un terme ; pour ce faire, nous lui appliquons une variable fraîche. Ce qui est la signification de `t n1` dans `H3`.

Ensuite, nous instancions la variable nominale `n1` dans `H3` avec `u`, ce qui nous donne une nouvelle hypothèse `H4`.

```
test_pure_beta <  inst H3 with n1 = u.

H2 : {pure u}
H4 : {pure u |- pure (t u)}
============================
 {pure (t u)}
```

Puis, avec une simple coupure de l'hypothèse `H4` par l'hypothèse `H2`, nous obtenons l'hypothèse `H5`, qui est notre objectif de preuve. Ainsi, la preuve est terminée.

```
test_pure_beta < cut H4 with H2.

H5 : {pure (t u)}
============================
 {pure (t u)}
```

Comme nous venons de le voir dans l'exemple précédent, le raisonnement par cas sur le fait que `t` est pur nous a conduit à un cas avec un contexte.

```
{pure n1 |- pure (t n1)}
```

Cela vient du fait que, dans la règle correspondante, la prémisse a la forme d'une implication. Dans le cas de l'abstraction, la règle a pour prémisse que, pour tout `x` pur, `U x` est pur. Cela se traduit dans les preuves par un jugement où `U x` est pur dans le contexte où `x` est pur.

Les contextes sont de type `olist`, ce qui signifie qu'ils sont représentés sous la forme d'une liste de prédicats de la logique de spécification. Il n'y a pas de restriction sur la composition de cette liste.

De manière plus générale, nos lemmes doivent être énoncés avec un contexte quelconque, car nous ne savons pas quel sera l'état du contexte lorsque nous utiliserons le lemme. Avec l'énoncé actuel, notre lemme ne peut être utilisé qu'avec un contexte vide. Nous allons donc généraliser l'énoncé de notre lemme.

```
Theorem test_pure_beta :
  forall t u L, {L |- pure (abs t)} ->
  {L |- pure u} -> {L |- pure(t u)}.
```

En ajoutant un contexte, le raisonnement par cas nous a créé un deuxième sous-but à prouver.

```
Subgoal 2:

H2 : {L |- pure u}
H3 : {L, [F] |- pure (abs t)}
```

```
H4 : member F L
============================
 {L |- pure (t u)}
```

Dans ce deuxième sous-cas, nous voyons une nouvelle notation avec des crochets appelés «focus» et un nouveau prédicat présent dans le langage de base d'Abella, `member`, qui signifie que l'élément `F` est présent dans la liste `L`. Dans ce cas, les prédicats avec un contexte et un focus dans ce contexte signifient que le prédicat est prouvé avec l'élément du contexte qui se trouve dans le focus. Cela nous permet, lors de la preuve, de raisonner sur le contexte d'un prédicat.

Ce dernier cas représente donc le cas où la pureté de `abs t` est déduite uniquement du contexte `L`, par exemple car ce contexte contiendrait précisément la formule `pure (abs t)`. Une telle hypothèse tautologique `H3` serait cependant inutile pour poursuivre la preuve. Nous allons donc plutôt restreindre le contenu possible de `L` pour éliminer ces cas.

5.1.3 Gestion du contexte

C'est pourquoi nous avons défini des prédicats qui permettent de préciser la nature des contextes. Par exemple, le prédicat `ctx_pure` nous indique que tous les éléments de `L` sont de la forme `pure x`, avec `x` une variable fraîche.

```
Define ctx_pure: olist -> prop by
  ctx_pure nil;
  nabla x, ctx_pure (pure x :: L) := ctx_pure L.
```

On modifie donc encore une fois l'énoncé de notre lemme pour contraindre `L` avec le prédicat `ctx_pure`.

```
Theorem test_pure_beta :
  forall t u L, ctx_pure L -> {L |- pure (abs t)} ->
  {L |- pure u} -> {L |- pure(t u)}.
```

Le premier cas est inchangé. Le deuxième cas est semblable à l'original, mais avec l'hypothèse supplémentaire `H1`, qui précise la nature de `L`.

```
Subgoal 2:

H1 : ctx_pure L
H3 : {L |- pure u}
H4 : {L, [F] |- pure (abs t)}
H5 : member F L
============================
 {L |- pure (t u)}
```

Comme nous l'avons vu, ce cas est celui où `abs t` est pur, à cause de `F` dans le contexte `L`. Nous avons donc besoin d'un théorème qui précise la forme de `F`. Ce théorème `mem_pure` affirme que si un contexte `L` satisfait le prédicat `ctx_pure`, et que l'on trouve un élément `F` à l'intérieur de `L`, alors `F` est nécessairement de la forme `pure x` et `x` satisfait le prédicat `fresh`, ce qui signifie simplement que `x` est une variable nominale.

```
Define fresh : trm -> prop by
  nabla x, fresh x.

Theorem mem_pure : forall F L,
  ctx_pure L -> member F L ->
  exists x, F = pure x /\ fresh x.
```

Nous utilisons donc notre théorème `mem_pure`, ce qui remplace `F` par `pure x` et qui nous donne l'hypothèse que `x` est `fresh`.

```
test_pure_beta < apply mem_pure to H1 H5.
Subgoal 2:

H4 : {L, [pure x] |- pure (abs t)}
H5 : member (pure x) L
H6 : fresh x
============================
 {L |- pure (t u)}
```

Puisque le focus dans `H4` dit que `pure (abs t)` a été déduit de `pure x`, cela oblige à unifier `pure x` et `pure (abs t)`. La seule manière de les unifier est que `abs t = x`. Ainsi, Abella remplace `x` par `abs t`.

```
test_pure_beta < case H4.
Subgoal 2:

H5 : member (pure (abs t)) L
H6 : fresh (abs t)
============================
 {L |- pure (t u)}
```

Nous concluons en montrant une contradiction dans les hypothèses. L'hypothèse `H6` nous dit que `abs t` respecte le prédicat `fresh`. Or, ce n'est pas possible, car `abs t` n'est pas une variable nominale, ce qui met fin à la preuve.

Nous venons de voir comment nous traitons le cas où le prédicat de l'une de nos hypothèses est déduit de son propre contexte Ce cas apparaît à chaque fois qu'une preuve est énoncée avec des contextes (donc tout le temps). C'est pourquoi des variables du théorème `mem_pure` existent pour tous les prédicats `ctx_`....

5.1.4 Induction structurelle

Pour parler de l'induction, nous allons détailler la preuve du lemme `mem_pure` que nous avons utilisé pour la preuve précédente.

```
Theorem mem_pure : forall F L,
  ctx_pure L -> member F L ->
  exists x, F = pure x /\ fresh x.
```

Pour commencer cette preuve par induction sur la liste `L`, nous allons utiliser la tactique `induction` sur notre première prémisse.

```
mem_pure < induction on 1.
```

```
IH : forall F L, ctx_pure L * -> member F L ->
       (exists x, F = pure x /\ fresh x)
============================
 forall F L, ctx_pure L @ -> member F L ->
 (exists x, F = pure x /\ fresh x)
```

On peut constater des étoiles sur certaines hypothèses, notamment dans l'hypothèse d'induction. Ces étoiles sont les marqueurs de la décroissance structurelle sur laquelle repose l'induction. Abella nous indique donc par ces étoiles quels termes peuvent être utilisés pour l'hypothèse d'induction.

Une hypothèse avec un @ est précisément l'hypothèse sur laquelle on fait l'induction. C'est en la déconstruisant que l'on obtient les hypothèses étoiles auxquelles peuvent s'appliquer les hypothèses d'induction. Par conséquent, il faut procéder à une étude de cas sur cette hypothèse @. C'est pourquoi, le schéma de preuve le plus courant est d'effectuer une induction puis de procéder à un raisonnement par cas directement après.

```
mem_pure < intros.

H1 : ctx_pure L @
H2 : member F L
============================
 exists x, F = pure x /\ fresh x

mem_pure < case H1.
Subgoal 1:

H2 : member F nil
============================
 exists x, F = pure x /\ fresh x

Subgoal 2 is:
 exists x, F n1 = pure x /\ fresh x
```

Ici, l'induction est effectuée sur une liste, donc il y a deux sous-buts. Le premier sous-but est le cas où la liste `L` est vide. Dans ce cas, `F` ne peut pas être dans la liste vide, donc il y a une contradiction et le cas est éliminé.

```
Subgoal 2:

IH : forall F L, ctx_pure L * -> member F L ->
       (exists x, F = pure x /\ fresh x)
H2 : member (F n1) (pure n1 :: L1)
H3 : ctx_pure L1 *
============================
 exists x, F n1 = pure x /\ fresh x
```

Une fois ici, on constate que l'hypothèse `H2` est un peu plus précise. Elle nous dit que `F n1` est dans la liste `pure n1 :: L1`. On note que `F` a été lifté avec `n1` lorsque la variable nominale `n1` est apparue. Le lambda-lifting est une technique utilisée dans le prouveur Abella pour permettre à un terme abstrait de dépendre d'une variable nominale : le terme `F n1` peut parler de `n1`, tandis que `n1` est fraîche par rapport à `L1`

Maintenant, on va faire un raisonnement par cas sur l'hypothèse `H2`. Soit `F n1` vaut pure n1 (but 2.1), soit `F n1` est dans `L1` (but 2.2).

```
mem_pure < case H2.
Subgoal 2.1:

H3 : ctx_pure L1 *
============================
 exists x, pure n1 = pure x /\ fresh x

Subgoal 2.2 is:
 exists x, F n1 = pure x /\ fresh x
```

Si `F n1 = pure n1`, alors nous faisons le remplacement et la preuve est finie. Sinon, on utilise l'hypothèse d'induction, ce qui permet de terminer la preuve.

```
Subgoal 2.2:

IH : forall F L, ctx_pure L * -> member F L ->
       (exists x, F = pure x /\ fresh x)
H3 : ctx_pure L1 *
H4 : member (F n1) L1
============================
 exists x, F n1 = pure x /\ fresh x

mem_pure < apply IH to H3 H4.
Subgoal 2.2:

H4 : member (pure (x n1)) L1
H5 : fresh (x n1)
============================
 exists x1, pure (x n1) = pure x1 /\ fresh x1
```

On note que l'hypothèse `H3` peut utiliser l'hypothèse d'induction car elle a une étoile ce qui veut dire qu'elle est un sous-cas de l'hypothèse `H1`. En l'occurrence, ici `L1` est une sous-liste de `L`.

La preuve est finie en utilisant la tactique `search`.

```
mem_pure < search.
Proof completed.
```

5.1.5 Relations et fonctions

La logique minimale d'Abella ne permet pas d'écrire une fonction qui prend un argument et renvoie un résultat, comme on le ferait en programmation. Cependant, il est possible de définir des relations qui prennent des arguments et établissent des liens entre eux.

Dans la version papier, le dépliage (Fig. 2.46) est considéré comme une fonction, dans le sens où le dépliage d'un terme t donne un terme t'. Dans la logique minimale d'Abella, cette fonction est vue comme une relation.

Cette relation `star` nous dit que deux termes sont en relation si le deuxième terme est le dépliage du premier. Par exemple, si on a `star t1 t2`, cela signifie que $t_1^\star = t_2$. La définition de `star` en Abella repose sur les mêmes critères que la version papier.

```
type star  trm -> trm -> o.

star (app U V) (app X Y) :- star U X, star V Y.
star (abs U) (abs X) :- pi x\ star x x => star (U x) (X x).
star (es U V) (X Y) :- star V Y, pi x\ star x x =>
                                         star (U x) (X x).
```

On constate que `star` a des prémisses avec des implications. Donc une preuve qui effectue un raisonnement sur ce prédicat va introduire des prédicats avec contextes. De la même manière que pour les prédicats précédents, nous définissons un prédicat `ctx_star` qui garantit que le contexte ne contient que des prédicats `star`.

Pour illustrer le fait que les fonctions sont représentées par des relations, nous allons parler de trois lemmes sur le prédicat `star`. Nous commençons par le lemme `star_app`.

Ce lemme nous dit que l'étoile d'un terme de la forme `app T1 T2` est forcément un terme de la forme `app U1 U2`, avec `U1` qui est l'étoile de `T1` et `U2` l'étoile de `T2`, le tout sous un contexte `L` qui est `ctx_star`.

```
Theorem star_app :
  forall L T1 T2 U,
  ctx_star L -> { L |- star (app T1 T2) U } ->
  exists U1 U2, U = app U1 U2 /\
  { L |- star T1 U1 } /\ { L |- star T2 U2 }.
```

Pour l'abstraction, le principe est le même, nous avons en conclusion le fait que les sous-termes sont en relation par `star`. Ici, le sous-terme de l'abstraction `T` appliqué à une variable nominale `x` doit être en relation `star` avec le sous-terme `U'` de l'autre abstraction `U` appliqué à la même variable nominale `x`. De plus, `x` est en relation avec elle-même.

```
Theorem star_abs :
  forall L T U,
  ctx_star L -> { L |- star (abs T) U } ->
  exists U', U = abs U' /\ nabla x,
  { L, star x x |- star (T x) (U' x) }.
```

Pour la substitution explicite, c'est une combinaison des deux lemmes précédents. Le premier sous-terme `T1` a les mêmes contraintes que l'application, et le deuxième sous-terme `T2` a les mêmes contraintes que l'abstraction.

```
Theorem star_es :
  forall L T1 T2 U,
  ctx_star L -> { L |- star (es T1 T2) U } ->
  exists U1 U2, U = U1 U2 /\ nabla x,
  { L, star x x |- star (T1 x) (U1 x) } /\ { L |- star T2 U2 }.
```

5.1.6 Induction sur les dérivations

Nous allons examiner la preuve du théorème `star_uniqueness`, qui rappelle l'aspect fonctionnel de la relation `star`, afin de mieux comprendre certaines notions telles

que l'induction sur les dérivations, dans une preuve plus complexe et plus proche de ce qui est généralement fait dans le reste de notre formalisation.

```
Theorem star_uniqueness : forall T U W C,
  ctx_star C -> {C |- star T U} -> {C |- star T W} -> U = W.
```

Dans un premier temps, on peut remarquer que le théorème `star_uniqueness` est énoncé pour un contexte `C` quelconque qui ne contient que des prédicats de la forme `star x y`. La définition de `ctx_star` est établie de la même manière que pour tous les autres prédicats `ctx_...`. Toutefois, ce prédicat a une arité de deux. De plus, lorsque l'on observe la définition de `star`, on remarque que ce prédicat sur les variables nominales n'est vrai que pour une même variable nominale, de sorte que la liste `C` ne contient plus précisément que des prédicats de la forme `star x x`.

Nous commençons par faire une induction sur la dérivation du prédicat `{C |- star T U}`, puis nous faisons un raisonnement par cas sur ce même prédicat. Il y aura donc quatre sous-cas correspondant à chaque manière possible de dériver le prédicat : le cas où `T` est une application, une abstraction, une substitution explicite et le dernier cas qui apparaît systématiquement lorsque l'on utilise des contextes, où le prédicat est directement déduit du contexte.

Le premier cas est celui de l'application.

```
IH : forall T U W C, ctx_star C -> {C |- star T U}* ->
  {C |- star T W} -> U = W
H1 : ctx_star C
H3 : {C |- star (app U1 V) W}
H4 : {C |- star U1 X}*
H5 : {C |- star V Y}*
============================
 app X Y = W
```

On voit que le raisonnement par cas sur `star T U` a bien transformé `T` en `app U1 V`, mais n'a rien fait sur `W`. On utilise donc le lemme `star_app` pour décomposer `W` en `app U2 U5` (dans `H3`) avec les nouvelles hypothèses `star U1 U2` et `star V U3`.

```
star_uniqueness < apply star_app to H1 H3.
Subgoal 1:

H3 : {C |- star (app U1 V) (app U2 U3)}
...
H6 : {C |- star U1 U2}
H7 : {C |- star V U3}
============================
 app X Y = app U2 U3
```

Nous voyons que l'utilisation du lemme `star_app` nous a donné les hypothèses `H6` et `H7` et a transformé `W` en une application.

De là, nous pouvons utiliser l'hypothèse d'induction sur les hypothèses `H4` et `H6` puis `H5` et `H7`, ce qui fait le lien entre `X` et `U2` et entre `Y` et `U3`. Cela nous permet de conclure le sous-cas de l'application.

Les deux sous-cas suivants fonctionnent de manière identique, aussi nous ne les détaillerons pas. Le dernier cas, traitant le contexte, est géré de la même façon que dans le lemme `mem_pure`.

5.2 Formalisation des calculs

Dans cette section, nous présenterons dans un premier temps la formalisation des éléments liés au lambda-calcul pur, puis les éléments qui font le lien avec le lambda-calcul étendu avec la substitution explicite, et enfin les prédicats sur notre calcul $\lambda_{\mathsf{sn}+}$ tels que les règles de réduction et les formes normales.

5.2.1 Formalisation du lambda-calcul pur

On a déjà vu la dérivation des termes avec substitutions explicites et le prédicat `pure` qui nous indique que le terme est un lambda-terme pur. Pour pouvoir énoncer et résoudre nos preuves de correction de notre calcul $\lambda_{\mathsf{sn}+}$ par rapport au lambda-calcul pur, nous avons besoin de formaliser les éléments du lambda-calcul pur tels que la bêta-réduction et la caractérisation des formes normales.

Nous commençons par la bêta-réduction. Comme vu précédemment, on ne peut pas définir de fonction, donc la bêta-réduction est représentée comme une relation. La relation `beta T U` nous dit que `T` se réduit en `U` en une étape de bêta-réduction.

Cette relation `beta` a quatre règles comme la bêta-réduction classique. Le premier cas est celui où l'on va réduire le sous-terme de gauche dans une application. Le deuxième est le cas où l'on va réduire à droite dans une application. Le troisième cas est celui où l'on va réduire sous une abstraction. Et le quatrième cas est celui où l'on a une application avec comme sous-terme gauche une abstraction, et là nous faisons la bêta-réduction.

```
type beta    trm -> trm -> o.

beta (app M N) (app M' N) :- beta M M'.
beta (app M N) (app M N') :- beta N N'.
beta (abs R) (abs R') :- pi x\ beta (R x) (R' x).
beta (app (abs R) M) (R M).
```

On note dans le cas de l'abstraction, comme on a déjà pu le voir, que les deux sous-termes `R` et `R'` sont des fonctions et que pour pouvoir les mettre en relation on doit leur appliquer une variable nominale.

On définit la relation `betas` pour parler d'une séquence de réduction `beta`.

```
type betas   trm -> trm -> o.

betas M M.
betas M N :- beta M P, betas P N.
```

Le prédicat `nfb` dit qu'un terme pur est en forme normale. Le prédicat `notabs` nous dit qu'un terme n'est pas une abstraction. On s'en sert caractériser les applications en forme normale. Si les deux sous-termes de l'application sont en forme normale et que le sous-terme de gauche n'est pas une abstraction, alors on est sûr qu'il n'y a pas de bêta-réduction possible.

```
type frozen  trm -> o.
type notabs  trm -> o.
type nfb     trm -> o.
```

```
notabs T :- frozen T.
notabs (app T U).

nfb X :- frozen X.
nfb (abs T) :- pi x\ frozen x => nfb (T x).
nfb (app T U) :- nfb T, nfb U, notabs T.
```

On voit que dans les deux prédicats `notabs` et `nfb`, on utilise le prédicat `frozen`. En effet, nous avons besoin de marquer les variables pour les cas de base car il n'y a pas de constructeur «variable» pour les termes. Donc, les variables nominales sont introduites avec `pi` en utilisant la propriété `frozen`. Le cas de base nous dit que tous les termes qui sont marqués par `frozen` (c'est-à-dire les variables nominales) respectent la propriété. On note que la prémisse de la règle de forme normale sur les abstractions a une implication avec à gauche un prédicat `frozen`, ce qui laisse présager qu'il va y avoir un nouveau contexte pour gérer cela, `ctx_frozen`.

5.2.2 Formalisation du calcul amélioré $\lambda_{\mathsf{sn}+}$

Pour formaliser notre calcul, nous allons commencer par les prédicats `frozen`, `omega`, et `active`. Les termes qui sont `frozen` représentent des variables qui sont gelées (dans φ) et les termes qui sont `omega` représentent des variables locales non gelées (dans ω), comme dans la version papier, alors que les termes qui sont `active` représentent les variables qui ne sont ni gelées (dans φ) ni locales (dans ω).

```
type frozen  trm -> o.
type omega   trm -> o.
type active  trm -> o.
```

On constate que l'aspect gelé d'une variable n'est pas géré par un ensemble que l'on déplace comme dans la version papier, mais comme un prédicat sur un terme. Donc, cela va être géré de la même manière que le prédicat `pure` par exemple, c'est-à-dire que l'on va stocker ces informations dans le contexte. Et comme pour les autres prédicats, nous aurons un prédicat `ctx` pour chacun.

Dans la version papier, nous utilisons la convention de Barendregt qui nous garantit la fraîcheur des variables nominales. Ainsi, si une variable n'est pas explicitement liée à un ensemble, elle ne peut pas en faire partie. Cependant, avec Abella, si nous ne précisons pas la nature d'une variable, elle peut se retrouver dans n'importe quel ensemble. Afin de préciser qu'une variable n'est pas dans ω ou dans φ, nous utilisons le prédicat `active`.

Les deux propriétés `struct` et `struct_omega` sont les mêmes que dans la version papier (Fig. 2.48) au détail près qu'elles ne sont pas paramétrées par un ensemble. En effet, comme indiqué précédemment, le statut gelé ou non des variables n'est plus géré par des ensembles. Les structures ne sont donc plus paramétrées par ces ensembles, mais sont maintenant considérées dans un certain contexte.

```
type struct        trm -> o.
type struct_omega  trm -> o.
```

```
struct X :- frozen X.
struct (app U V) :- struct U.
struct (es U V) :- pi x\ struct (U x).
struct (es U V) :- pi x\ frozen x => struct (U x), struct V.

struct_omega X :- omega X.
struct_omega (app U V) :- struct_omega U.
struct_omega (es U V) :- pi x\ struct_omega (U x).
struct_omega (es U V) :- pi x\ omega x => struct_omega (U x),
                          struct_omega V.
```

On voit dans les définitions des prédicats `struct` que les prémisses introduisent des prédicats avec contextes `frozen` et `omega`. Donc, il faudra utiliser les contextes `ctx` correspondants dans les preuves qui parlent de structures.

On introduit aussi un nouveau type atomique qui est `bool` et deux instances de ce type `top` et `bot` qui représentent le mode, de la même manière que dans la version papier.

```
kind bool type.
type top bool.
type bot bool.
```

Une fois que les prédicats sur les variables, les structures et les modes ont été introduits, nous pouvons formuler nos règles de réduction.

On commence par introduire un nouveau type atomique `step_type`, qui est le type des réductions. On définit ensuite quatre types de réduction : `db`, `lsv`, `idx` et `sub`. Ce sont les mêmes que ceux de la version papier (Fig. 3.4). On constate que `db` et `lsv` ne sont pas paramétrés, tandis que `idx` est associé à un paramètre id_x et que `sub` en a deux, $\mathsf{sub}_{x\backslash v}$.

```
kind step_type type.

type db   step_type.
type lsv  step_type.
type idx  trm -> step_type.
type sub  trm -> trm -> step_type.
```

On définit ensuite les réductions. On a une relation `step` qui correspond à la flèche $\xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn+}}$ (Fig. 3.3 et 3.4), prenant en paramètres `step_type` (correspondant à ρ), un booléen pour le mode et les deux termes. De la même manière que pour les structures, l'ensemble φ n'est pas passé en paramètre. Les deux règles auxiliaires $\rightarrow_{\mathsf{db}}$ et $\xrightarrow{\varphi,\mu}_{\mathsf{lsv}}$ (Fig. 3.5) sont représentées par les deux relations `aux_db` et `aux_lsv`.

```
type step     step_type -> bool -> trm -> trm -> o.
type aux_db   trm -> trm -> o.
type aux_lsv  bool -> trm -> trm -> o.
```

La majorité des règles sont identiques, mais il y a quand même quelques différences que nous allons préciser. Pour commencer, nous allons réexaminer la règle ES-LEFT-FROZEN et la règle d'Abella correspondante :

$$\text{ES-LEFT-FROZEN}\quad \frac{t \xrightarrow{\rho,\varphi\cup\{x\},\mu}_{\mathsf{sn}} t' \qquad u \in \mathcal{S}_\varphi}{t[x\setminus u] \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t'[x\setminus u]}$$

```
step R B (es T U) (es T' U) :-
  pi x\ frozen x => step R B (T x) (T' x), struct U.
```

L'expression $t[x\backslash u] \xrightarrow{\rho,\varphi,\mu}_{\mathsf{sn}} t'[x\backslash u]$ correspond à `step R B (es T U) (es T' U)`. On constate que les deux représentations sont similaires, la seule différence réside dans l'absence de φ. Enfin, l'expression $t \xrightarrow{\rho,\varphi\cup\{x\},\mu}_{\mathsf{sn}} t'$ correspond à `pi x\ frozen x => step R B (T x) (T' x)`. Dans cette dernière prémisse, on observe que l'ajout d'une variable dans le contexte est représenté par une condition `forzen x` dans la prémisse.

On présente ensuite les autres règles qui ne changent pas par rapport à la version papier.

```
step R top (abs T) (abs T') :-
  pi x\ frozen x => step R top (T x) (T' x).

step R B (app T U) (app T' U) :- step R bot T T'.

step R B (app T U) (app T U') :- struct T, step R top U U'.

step R B (es T U) (es T' U) :-
  pi x\ frozen x => step R B (T x) (T' x), struct U.

step lsv B T T' :- aux_lsv B T T'.
step db B T T' :- aux_db T T'.

aux_db (app (abs T) U) (es T U).
aux_db (app (es T W) U) (es T' W) :-
  pi x\ aux_db (app (T x) U) (T' x).

aux_lsv B (es T (es U W)) (es T' W) :-
  pi x\ frozen x => aux_lsv B (es T (U x)) (T' x), struct W.
```

Ensuite, il y a des règles où nous avons ajouté `active` à certains endroits. Comme éxpliqué précédemment, `active` permettra de garantir que la variable n'est pas dans `frozen`. Ce prédicat intervient lorsque l'on utilise une réduction `sub` ou `idx`.

```
step R B (es T U) (es T U') :-
  pi x\ active x => step (idx x) B (T x) (T x), step R bot U U'.

step (idx X) B X X :- active X.
step (sub X V) B X V :- active X.

aux_lsv B (es T (abs V)) (es T' (abs V)) :-
  pi x\ active x =>
  step (sub x (abs V)) B (T x) (T' x), lnf bot (abs V).
```

On note également que la modalité de la règle ABS-$\bot$ n'est pas la même que dans la règle Abella. Cependant, ce n'est pas un problème car nous avons la monotonie des modes, ce qui signifie que tout ce qui est réductible $\bot$ est également réductible $\top$. Ainsi, un terme réductible $\bot$ est réductible quel que soit le mode.

```
step R B (abs T) (abs T') :-
  pi x\ omega x => step R bot (T x) (T' x).

step R B (es T U) (es T' U) :-
  pi x\ omega x => step R B (T x) (T' x).

aux_lsv B (es T (es U W)) (es T' W) :-
  pi x\ omega x => aux_lsv B (es T (U x)) (T' x).
```

On voit que dans le prédicat `step` on utilise le prédicat `lnf` comme dans la dernière version du calcul sur papier (Fig. 4.1). Le prédicat `lnf` est défini comme suit.

```
type lnf  bool -> trm -> o.

lnf B X :- frozen X.
lnf B X :- omega X.
lnf B (app T U) :- lnf B T, struct T, lnf top U.
lnf B (app T U) :- lnf B T, struct_omega T.
lnf bot (abs T) :- pi x\ omega x => lnf bot (T x).
lnf B (abs T) :- pi x\ frozen x => lnf top (T x).
lnf B (es T U) :- pi x\ lnf B (T x).
lnf B (es T U) :- pi x\ omega x => lnf B (T x), struct_omega U.
lnf B (es T U) :-
  pi x\ frozen x => lnf B (T x), lnf bot U, struct U.
```

On note cependant deux petites différences de modalité. La raison pour laquelle il y a ces différences est purement technique, mais nous allons voir en quoi cela ne pose pas de problème et représente bien la même chose. Tout d'abord, dans la version papier, la règle LNF-ABS-$\top$ ne s'applique qu'au mode $\top$, tandis que dans les règles Abella, le mode n'est pas précisé. Cela est possible car, comme nous l'avons vu dans la version papier, il y a une monotonie par rapport au mode. Cela signifie que si un terme est en forme normale locale avec le mode $\top$, alors il l'est aussi avec le mode $\bot$, par conséquent nous pouvons remplacer le mode $\top$ par n'importe quel mode.

Ensuite, dans la version papier de la règle LNF-ES-φ, le corps de la substitution explicite doit être en forme normale locale, quelque soit le mode. Cela vient du fait que ce même terme doit également être une structure. Or, si un terme est une structure, le mode dans lequel on le regarde pour déterminer s'il est en forme normale locale n'a pas d'importance, alors que dans les règles Abella, le mode est fixé à $\bot$, pour la même raison de monotonie. Ainsi, cela ne pose pas de problème.

On a également utilisé le prédicat `omega` qui n'est pas présent dans la version papier. De la même manière que l'on a utilisé le prédicat `frozen` dans la définition des lambda-termes purs, ici on utilise `omega` pour parler des variables qui ne sont pas `frozen`. Il y a donc une surcharge du prédicat `omega`. Mais contrairement au cas précédent, ici le prédicat `omega` est utilisé dans le prédicat `lnf` qui est utilisé dans `step`. Dans la version papier, quand on utilise le prédicat `lnf`, on l'utilise avec

un omega vide. Ici, omega n'est pas vide, mais cela ne pose pas de problème car nous avons vu dans le papier que les formes normales locales étaient monotones par rapport à omega.

On a vu plusieurs fois que nous surchargeons la signification d'un prédicat. Il serait possible de corriger ce problème en créant plusieurs prédicats. Par exemple, nous pourrions remplacer la première utilisation du prédicat frozen par un nouveau prédicat var. Mais cela multiplierait les prédicats ctx_ et mem_.

La propriété nf dit qu'un terme est en forme normale. Dans la version papier (Fig. 3.10), le fait qu'une variable soit à une position vivante est géré par la propriété $\mathscr{V}$, et toutes les variables sont en forme normale. Dans Abella, les choses sont gérées différemment : on applique le tag frozen aux variables qui sont introduites par un lieur (abs ou es), et seules les variables frozen sont en forme normale. Cela a pour effet que si une variable n'est pas taguée et est vivante, le terme ne peut pas être en forme normale. Ainsi, la règle LNF-ES qui a comme prémisse $t \notin \mathscr{V}_x$ est parfaitement traduite par la règle Abella `nf (es U V) :- pi x\ nf (U x)`, car la seule manière pour que `U x` soit en forme normale alors que x n'est pas frozen est que x ne soit pas vivante dans U.

```
type nf   trm -> o.

nf X :- frozen X.
nf (app U V) :- nf U, nf V, struct U.
nf (abs U) :- pi x\ frozen x => nf (U x).
nf (es U V) :- pi x\ frozen x => nf (U x), nf V, struct V.
nf (es U V) :- pi x\ nf (U x).
```

Le prédicat red représente le cas particulier de step en mode $\top$ avec db ou lsv. On utilise ce prédicat car, lorsqu'on veut déterminer si un terme est réductible (notamment dans la preuve de correction), on parle d'une vraie réduction à top-level, c'est-à-dire une réduction db ou lsv en mode $\top$.

```
type red      trm -> trm -> o.

red T T' :- step db top T T'.
red T T' :- step lsv top T T'.
```

Le prédicat lnf dit qu'un terme est en forme normale locale, exactement comme dans la version papier (Fig. 4.4).

5.2.3 Fichier sig et mod

Le fichier de signature (Fig. 5.1) donne le type des différents éléments que nous allons manipuler lors de nos preuves. Les fichiers de spécification (Fig. 5.2 et 5.3) donnent leurs définitions. Comme le fichier qui contient les preuves est trop volumineux pour être présenté ici, nous ne présentons que les résultats.

```
kind step_type, bool, trm  type.

type top  bool.
type bot  bool.

type app  trm -> trm -> trm.
type abs  (trm -> trm) -> trm.
type es   (trm -> trm) -> trm -> trm.

type frozen  trm -> o.
type omega   trm -> o.
type active  trm -> o.

type trm   trm -> o.
type pure  trm -> o.
type star  trm -> trm -> o.

type answer        trm -> o.
type struct        trm -> o.
type struct_omega  trm -> o.

type db   step_type.
type lsv  step_type.
type idx  trm -> step_type.
type sub  trm -> trm -> step_type.

type red      trm -> trm -> o.
type step     step_type -> bool -> trm -> trm -> o.
type aux_db   trm -> trm -> o.
type aux_lsv  bool -> trm -> trm -> o.

type nf   trm -> o.
type lnf  bool -> trm -> o.

type beta    trm -> trm -> o.
type betas   trm -> trm -> o.
type nfb     trm -> o.
type notabs  trm -> o.
```

FIGURE 5.1 – Fichier de signature

```
trm (app U V) :- trm U, trm V.
trm (abs U) :- pi x\ trm x => trm (U x).
trm (es U V) :- pi x\ trm x => trm (U x), trm V.

pure (app U V) :- pure U, pure V.
pure (abs U) :- pi x\ pure x => pure (U x).

star (app U V) (app X Y) :- star U X, star V Y.
star (abs U) (abs X) :- pi x\ star x x => star (U x) (X x).
star (es U V) (X Y) :- star V Y, pi x\ star x x => star (U x) (X x).

answer (abs T).
answer (es T U) :- pi x\ answer (T x).

struct X :- frozen X.
struct (app U V) :- struct U.
struct (es U V) :- pi x\ struct (U x).
struct (es U V) :- pi x\ frozen x => struct (U x), struct V.

struct_omega X :- omega X.
struct_omega (app U V) :- struct_omega U.
struct_omega (es U V) :- pi x\ struct_omega (U x).
struct_omega (es U V) :- pi x\ omega x => struct_omega (U x),
                         struct_omega V.

red T T' :- step db top T T'.
red T T' :- step lsv top T T'.

step R top (abs T) (abs T') :-
  pi x\ frozen x => step R top (T x) (T' x).
step R B (abs T) (abs T') :- pi x\ omega x => step R bot (T x) (T' x).
step R B (app T U) (app T' U) :- step R bot T T'.
step R B (app T U) (app T U') :- struct T, step R top U U'.
step R B (es T U) (es T' U) :- pi x\ omega x => step R B (T x) (T' x).
step R B (es T U) (es T' U) :-
  pi x\ frozen x => step R B (T x) (T' x), struct U.
step R B (es T U) (es T U') :-
  pi x\ active x => step (idx x) B (T x) (T x), step R bot U U'.

step (idx X) B X X :- active X.
step (sub X V) B X V :- active X.
step db B T T' :- aux_db T T'.
step lsv B T T' :- aux_lsv B T T'.
```

FIGURE 5.2 – Fichier de spécification partie 1

```
aux_db (app (abs T) U) (es T U).
aux_db (app (es T W) U) (es T' W) :-
  pi x\ aux_db (app (T x) U) (T' x).

aux_lsv B (es T (abs V)) (es T' (abs V)) :-
  pi x\ active x =>
  step (sub x (abs V)) B (T x) (T' x), lnf bot (abs V).
aux_lsv B (es T (es U W)) (es T' W) :-
  pi x\ omega x => aux_lsv B (es T (U x)) (T' x).
aux_lsv B (es T (es U W)) (es T' W) :-
  pi x\ frozen x => aux_lsv B (es T (U x)) (T' x), struct W.

nf X :- frozen X.
nf (app U V) :- nf U, nf V, struct U.
nf (abs U) :- pi x\ frozen x => nf (U x).
nf (es U V) :- pi x\ frozen x => nf (U x), nf V, struct V.
nf (es U V) :- pi x\ nf (U x).

lnf B X :- frozen X.
lnf B X :- omega X.
lnf B (app T U) :- lnf B T, struct T, lnf top U.
lnf B (app T U) :- lnf B T, struct_omega T.
lnf B (abs T) :- pi x\ frozen x => lnf top (T x).
lnf bot (abs T) :- pi x\ omega x => lnf bot (T x).
lnf B (es T U) :- pi x\ lnf B (T x).
lnf B (es T U) :-
  pi x\ frozen x => lnf B (T x), lnf bot U, struct U.
lnf B (es T U) :- pi x\ omega x => lnf B (T x), struct_omega U.

beta (app M N) (app M' N) :- beta M M'.
beta (app M N) (app M N') :- beta N N'.
beta (abs R) (abs R') :- pi x\ beta (R x) (R' x).
beta (app (abs R) M) (R M).

betas M M.
betas M N :- beta M P, betas P N.

nfb X :- frozen X.
nfb (abs T) :- pi x\ frozen x => nfb (T x).
nfb (app T U) :- nfb T, nfb U, notabs T.

notabs T :- frozen T.
notabs (app T U).
```

Figure 5.3 – Fichier de spécification partie 2

5.3 Preuve de correction

Dans cette section, nous allons formaliser la preuve de correction de notre calcul par rapport au lambda-calcul pur.

Pour commencer, nous utilisons la relation fonctionelle `star` que nous avons déjà vue plus haut. Elle nous permet de traduire les termes avec substitution explicite en termes du lambda-calcul pur en transformant les substitutions explicites en substitutions implicites.

Ensuite, nous allons prouver que la traduction de nos formes normales est bien une forme normale du lambda-calcul pur. Enfin, nous allons prouver que si un terme T de notre calcul $\lambda_{\mathsf{sn}+}$ se réduit en un terme U, alors la traduction de ce terme T se réduit bien dans le lambda-calcul pur au terme U traduit. Enfin, nous prouvons l'adéquation de la caractérisation des formes normales locales en mode $\top$, ce qui nous permet de prouver l'adéquation de la caractérisation des formes normales de notre calcul.

5.3.1 Correspondance des formes normales

Grâce à cette relation fonctionnelle `star`, on prouve le théorème qui dit que si `T` est en forme normale pour notre calcul, alors `T*` est en forme normale dans le lambda-calcul pur.

On énonce le théorème sans les contextes pour obtenir un théorème final plus esthétique.

```
Theorem nf_star' : forall T T*,
  { nf T } -> { star T T* } -> { nfb T* }.
```

Mais, pour le prouver par induction, nous devons énoncer le théorème (Lem. 31) avec son contexte.

```
Theorem nf_star : forall T T* L1 L2,
  ctx_frozen L1 -> ctx_star L2 ->
  { L1 |- nf T } -> { L2 |- star T T* } -> { L1 |- nfb T* }.
```

Cette preuve est une preuve par induction classique sur `nf T` qui utilise certains lemmes tels que `star_es`, comme nous l'avons vu avec la fonction `star`.

5.3.2 Simulation

Ensuite, nous formalisons la preuve de simulation. Cette preuve fonctionne de la même manière que dans la version papier (Lem. 4). Pour les mêmes raisons que pour `nf_star`, nous avons la version finale et la version avec les contextes.

```
Theorem simulation' : forall T U T* U*,
  { star T T* } -> { star U U* } -> { red T U } -> { betas T* U* }.

Theorem simulation : forall T T' T* T'* C1 C2,
  ctx_red C1 -> ctx_star C2 ->
  {C2 |- star T T*} -> {C2 |- star T' T'*} ->
  {C1 |- red T T'} -> {betas T* T'*}.
```

On voit apparaître un nouveau prédicat sur les contextes, `ctx_red`. Ce prédicat nous dit qu'un contexte peut contenir tout ce que les réductions utilisent, c'est-à-dire `frozen`, `omega` et `active`.

De manière plus générale, tous les énoncés de lemme qui utilisent `red` utiliseront des contextes `ctx_red`.

```
Define ctx_red : olist -> prop by
  ctx_red nil ;
  nabla x, ctx_red (frozen x :: L) := ctx_red L ;
  nabla x, ctx_red (omega x :: L) := ctx_red L ;
  nabla x, ctx_red (active x :: L) := ctx_red L.
```

Comme dans la version papier, les deux cas dB et lsv sont traités dans des lemmes à part. Nous avons le lemme `simulation_db` (Lem. 5) qui traite le cas de la réduction dB. Cette partie va traiter tous les cas récursifs en propageant la réduction dB, et le cas particulier de dB-BASE est traité dans un autre lemme, `simulation_aux_db`.

```
Theorem simulation_db : forall T T' T* T'* C1 C2 B,
  ctx_red C1 -> ctx_star C2 ->
  {C2 |- star T T*} -> {C2 |- star T' T'*} ->
  {C1 |- step db B T T'} -> {betas T* T'*}.

Theorem simulation_aux_db : forall T T' T* T'* C1 C2,
  ctx_red C1 -> ctx_star C2 -> {C2 |- star T T*} ->
  {C2 |- star T' T'*} -> {C1 |- aux_db T T'} -> {beta T* T'*}.
```

La partie `lsv` (Lem. 7) est découpée de la même manière que pour `db`, avec en plus le fait que `aux_lsv` utilise la réduction `sub`, donc il faut prévoir en plus le lemme avec la réduction `sub`.

```
Theorem simulation_lsv : forall T T' T* T'* C1 C2 B,
  ctx_red C1 -> ctx_star C2 ->
  {C2 |- star T T*} -> {C2 |- star T' T'*} ->
  {C1 |- step lsv B T T'} -> T* = T'*.

Theorem simulation_aux_lsv : forall T T' T* T'* C1 C2 B,
  ctx_red C1 -> ctx_star C2 ->
  {C2 |- star T T*} -> {C2 |- star T' T'*} ->
  {C1 |- aux_lsv B T T'} -> T* = T'*.

Theorem simulation_sub : forall T U T' T'* T* C1 C2 B U*, nabla x,
  ctx_red C1 -> ctx_star C2 -> {C2, star x x |- star U U*} ->
  {C2, star x x |- star (T x) (T* x)} ->
  {C2, star x x |- star (T' x) (T'* x)} ->
  {C1, active x |- step (sub x U) B (T x) (T' x) } ->
  (T* U*) = (T'* U*).
```

5.3.3 Adéquation de la caractérisation des formes normales locales

Nous allons prouver l'adéquation de la caractérisation des formes normales locales grâce à la combinaison des deux théorèmes `lnf_nand_red` (Th. 10) et `lnf_or_red` (Th. 12)

comme dans la version papier.

```
Theorem lnf_nand_red :
  forall T U, { lnf top T } -> { red T U } -> false.

Theorem lnf_or_red :
  forall T, { trm T } ->
  { lnf top T } \/ exists U, { red T U }.
```

Nous ne rentrerons pas dans les détails des deux preuves, mais nous allons présenter plusieurs lemmes qui ont permis de les prouver.

Un premier lemme intéressant est le lemme `lnf_struct` (Lem. 36), qui nous indique que le mode de la forme normale locale n'a pas d'importance lorsqu'on considère une structure. Cela justifie pourquoi la différence de mode dans certaines règles d'Abella ne pose pas de problème.

```
Theorem lnf_struct :
  forall LT L B T, ctx_trm LT -> ctx_red L -> {LT |- trm T} ->
  {L |- lnf B T} -> {L |- struct T} -> {L |- lnf top T}.
```

Comme dans la preuve papier, nous avons besoin du lemme de classification des formes normales locales (Lem. 40).

```
Theorem lnf_struct_or_answer:
  forall T L B, ctx_red L ->
  { L |- lnf B T } ->
  { L |- struct T } \/ { L |- struct_omega T } \/ { answer T }.
```

On remarque ici l'utilisation du prédicat `answer`. La propriété `answer` nous indique qu'un terme est une réponse exactement comme dans la version papier (Fig. 3.13).

```
type answer   trm -> o.
answer (abs T).
answer (es T U) :- pi x\ answer (T x).
```

On utilise aussi des lemmes d'affaiblissement tels que le lemme d'affaiblissement des variables dans les formes normales locales (Lem. 40).

```
Theorem frozen_lnf_weakening:
  forall L B T, nabla n, ctx_red L ->
  { L, frozen n |- lnf B (T n) } -> { L, omega n |- lnf B (T n) }.
```

Deux lemmes nous disent que les règles dB et lsv s'appliquent lorsqu'une réponse se trouve à une position adéquate (Lem. 39).

```
Theorem applied_answer_db:
  forall T U, { answer T } ->
  exists T', { aux_db (app T U) T' }.

Theorem subst_answer_lsv:
  forall L B T T' V, nabla x, ctx_red L ->
  { L, active x |- step (idx x) B (T x) (T' x) } ->
  { L |- lnf bot V } ->
  { answer V } ->
  exists U, { L |- aux_lsv B (es T V) U }.
```

Nous avons également le lemme qui nous dit qu'un terme réductible `idx` est réductible par `sub` (Lem. 15). Comme dans la version papier, nous avons besoin de faire le lien entre `sub` et `idx`, car nos lemmes sont énoncés avec `idx`, tandis que certaines règles utilisent `sub`.

```
Theorem idx_sub:
  forall L X B V T T', ctx_red L ->
  { L |- step (idx X) B T T' } ->
  exists U, { L |- step (sub X V) B T U }.
```

Nous avons également des lemmes de classification et des lemmes de monotonie sur les réductions. Par exemple, le lemme `step_non_answer_bot` (Lem. 16) nous dit que si un terme est réductible, alors soit c'est une réponse, soit il est également réductible en mode $\bot$.

```
Theorem step_non_answer_bot:
  (forall L R B T T', ctx_red L ->
   { L |- step R B T T' } ->
   { answer T } \/ { L |- step R bot T T' })
  /\
  (forall L B T T', ctx_red L ->
   { L |- aux_lsv B T T' } ->
   { answer T } \/ { L |- aux_lsv bot T T' }).
```

Dans la version papier, nous présentons dans un premier temps une version stratifiée des formes normales locales pour illustrer la monotonie (Fig. 4.2 et Fig. 4.3). Nous avons également ce lemme dans Abella sous le nom de `lnf_bot` (Lem. 45).

```
Theorem lnf_bot :
  forall L B T, ctx_red L ->
  { L |- lnf B T } -> { L |- lnf bot T }.
```

5.3.4 Adéquation de la caractérisations des formes normales

La preuve d'adéquation de la caractérisation des formes normales est quant à elle assez simple car elle repose entièrement sur l'utilisation du théorème d'adéquation de la caractérisation des formes normales locales en mode $\top$.

```
Theorem nf_nand_red :
  forall T U, { nf T } -> { red T U } -> false.

Theorem nf_or_red :
    forall T, { trm T } -> { nf T } \/ exists U, { red T U }.
```

5.4 Conclusion

Dans ce chapitre, nous avons formalisé une variante très proche de notre calcul $\lambda_{\mathsf{sn}+}$. Les différences entre la version papier et la version formalisée, sont principalement dues à des aspects techniques. Il serait possible de continuer à améliorer et à affiner notre formalisation. Ensuite, nous avons prouvé la correction de cette formalisation.

Le théorème qui était invalide pour la version papier de $\lambda_{\mathsf{sn}+}$ est valide dans la version Abella car cette dernière autorise un peu plus de réductions. Ainsi, cette variante très proche faite en Abella est une correction alternative du problème du chapitre 4, puisqu'elle donne un calcul correct et (conjecture) complet ressemblant à $\lambda_{\mathsf{sn}+}$, mais on considère que cette solution n'est pas définitive puisqu'elle n'a pas la propriété du diamant qu'a $\lambda_{\mathsf{sn}+}$.

Chapitre 6

Machine abstraite

Après avoir défini et formalisé notre calcul $\lambda_{\mathsf{sn+}}$, nous avons voulu l'implémenter dans une machine abstraite. À la différence du calcul théorique, une machine abstraite doit être déterministe, donc nous avons dû, par moment, faire des choix. Dans ce chapitre, nous allons d'abord parler de la configuration de la machine dans la section 6.1, puis des règles de réduction de la machine abstraite dans la section 6.2.

Enfin, dans la section 6.3, nous parlerons des détails d'implémentation car, dans la version purement théorique, certaines opérations sont considérées comme instantanées alors que dans l'implémentation, nous nous sommes rendu compte que des opérations que nous pensions triviales pouvaient entraîner une augmentation considérable du temps de calcul.

6.1 Configuration de la machine

Pour cette machine, on écrit d'abord une sémantique à grands pas dont les règles sont définies dans la figure 6.3. Les configurations (Σ, Π, μ, t) sont formées à partir d'un environnement Σ de variables, d'une pile Π de termes, d'un mode μ et du terme actuellement visité t. La sémantique à petits pas sera donnée par le code OCaml.

Si la machine termine, elle retourne une paire (Σ', t') composée d'un nouvel environnement Σ' et d'un terme t' équivalent à t, que l'on espère réduit.

L'environnement est défini par le type OCaml `env`, qui est une liste d'éléments du type `e_env`. L'environnement est une liste car nous utilisons les indices de De Bruijn donc le i-ème élément de la liste correspond au i-ème indice. Les éléments de la liste représentent la manière dont la variable associée a été introduite. Si elle a été introduite par une abstraction en mode $\top$, alors l'élément vaudra $Lambda_Frozen$; si elle est liée à une abstraction en mode $\bot$, alors ce sera *Lambda*. Et dans le cas où la variable est créée par une substitution explicite, alors elle sera liée à une référence vers une fermeture de type `closure`.

On utilise les références pour gérer le partage et on utilise une fermeture plutôt qu'un simple terme pour pouvoir bénéficier de l'environnement adéquat. En effet, la fermeture inclut à la fois le terme et l'environnement dans lequel il a été défini. Cela permet d'éviter les problèmes liés à la portée des variables et de garantir que le terme

sera évalué avec les bonnes variables associées. La pile Π de termes est en fait une liste de fermetures pour les mêmes raisons.

```
type e_env =
  | Bind of closure ref
  | Lambda
  | Lambda_Frozen

and env = e_env list

and closure = term_es * env
```

Les termes sont représentés par le type OCaml `term_es`, qui a la même définition que dans la version papier, à la différence que les liens sont gérés par les indices de De Bruijn. Cette représentation permet de simplifier la gestion des lieurs et d'éviter les problèmes de capture de variable.

```
type term_es =
  | EVar of int
  | EAbs of term_es
  | EApp of term_es * term_es
  | ES   of term_es * term_es
```

Pour finir, nous avons le type `mode` qui possède deux constructeurs : `Top` et `Bot`, qui correspondent aux modes que nous connaissons bien.

```
type mode =
  | Top
  | Bot
```

La configuration initiale est un environnement vide, une pile d'appel vide, le mode $\top$ et le terme à évaluer.

6.2 Règles de réductions

Pour présenter les règles de la machine abstraite (Fig. 6.3), nous allons suivre la réduction de plusieurs termes. Dans un premier temps, nous allons voir comment le terme $(\lambda x.x)\ (\lambda y.y\ t)$, avec t un terme quelconque, est réduit. L'arbre de dérivation de ce terme complet est présenté dans la figure 6.1.

Comme nous l'avons dit dans la section précédente, nous allons évaluer le terme avec un environnement et une pile vides en mode $\top$, ce qui donne comme configuration de départ : $(\emptyset, Nil, \top, (\lambda x.x)\ (\lambda y.y\ t))$.

Pour commencer, nous avons une application et il n'y a qu'une règle qui s'applique aux applications : la règle APP. La règle APP consiste à mettre le sous-terme de droite sur la pile et à évaluer le sous-terme de gauche en utilisant la pile mise à jour.

$$\text{APP}\ \frac{(\Sigma, u \cdot \Pi, \mu, t) \rightarrow (\Sigma', t')}{(\Sigma, \Pi, \mu, t\ u) \rightarrow (\Sigma', t')}$$

La nouvelle configuration est $(\emptyset, [\lambda x.x], \top, \lambda y.y\ t)$.

Ensuite, il y a deux règles pour les abstractions, Abs-cons lorsqu'un terme est présent sur la pile et Abs-nil dans le cas où la pile est vide. Les termes de la pile sont des arguments qui n'ont pas encore été passés à une fonction. Comme nous avons le terme $[\lambda y.y\ t]$ sur la pile, nous allons utiliser la règle Abs-cons.

$$\text{Abs-cons}\ \frac{(\Sigma, \Pi, \mu, t[x \setminus u]) \rightarrow (\Sigma', t')}{(\Sigma, u \cdot \Pi, \mu, \lambda x.t) \rightarrow (\Sigma', t')}$$

De manière plus générale, la règle Abs-cons est l'équivalent de la règle dB-BASE. Lorsqu'une abstraction est sous une application, cette règle la transforme en une substitution explicite. La nouvelle configuration est $(\emptyset, Nil, \top, x[x \setminus \lambda y.y\ t])$.

Ensuite, il n'y a qu'une règle qui gère les substitutions explicites, la règle ES.

$$\text{ES}\ \frac{(\Sigma; x \mapsto u, \Pi, \mu, t) \rightarrow (\Sigma'; x \mapsto u', t')}{(\Sigma, \Pi, \mu, t[x \setminus u]) \rightarrow (\Sigma', t'[x \setminus u'])}$$

Cette règle prend une substitution explicite et la place dans l'environnement. Elle évalue ensuite le corps de la substitution dans ce nouvel environnement et, une fois que l'évaluation a renvoyé un résultat, elle transforme à nouveau le lien présent dans l'environnement en une substitution explicite. Nous aurions pu conserver les substitutions explicites dans l'environnement, mais cela aurait été difficile à gérer au moment de fermer les abstractions. La nouvelle configuration est $(\{x \mapsto \lambda y.y\ t\}, Nil, \top, x)$.

Lorsqu'on évalue une variable, il y a quatre cas possibles : Var-λ-Frozen si la variable est liée à $\lambda_\top$, Var-λ-NonFrozen si la variable est liée à $\lambda_\perp$, Var-$\mathcal{S}$ si la variable est liée à un terme qui est une structure, et Var-abs si la variable est liée à un terme qui est une réponse.

Ici, notre variable est liée à un terme qui est une abstraction. Donc, nous utilisons la règle Var-abs.

$$\text{Var-abs}\ \frac{\Sigma(x) = u \qquad (\Sigma, Nil, \perp, u) \rightarrow (\Sigma', u') \qquad u' = (\lambda y.t)\mathcal{L} \qquad (\Sigma'[x \mapsto u'], \Pi, \mu, u') \rightarrow (\Sigma'', u'')}{(\Sigma, \Pi, \mu, x) \rightarrow (\Sigma'', u'')}$$

La nouvelle configuration est $(\{x \mapsto \lambda y.y\ t\}, Nil, \perp, \lambda y.y\ t)$.

Cette règle commence par évaluer le terme qui est lié à la variable avec une pile vide et en mode $\perp$. Ensuite, on met à jour dans l'environnement la variable avec le terme évalué. Ensuite, on substitue cette occurrence de la variable par le nouveau terme évalué. Enfin, le résultat est l'évaluation de ce nouveau terme dans le nouveau contexte.

À ce moment-là, il y a donc branches qui se créent dans l'arbre de dérivation car nous avons deux prémisses. On commence par décrire le sous-arbre qui est créé pour la première prémisse. On évalue le terme qui est lié à la variable, c'est-à-dire $\lambda y.y\ t$. La configuration est la pile est vide, donc la seule règle qui s'applique dans ce cas est Abs-nil.

Abs-nil

$$\frac{(\Sigma; x \mapsto \lambda_\mu, Nil, \mu, t) \to (\Sigma'; x \mapsto \lambda_\mu, t')}{(\Sigma, Nil, \mu, \lambda x.t) \to (\Sigma', \lambda x.t')}$$

Cette règle est assez simple : elle lie dans l'environnement la variable de l'abstraction à un λ paramétré par le mode dans lequel la règle est appelée. Ici, c'est en mode $\perp$, donc la variable y est ajoutée dans le contexte et est liée à $\lambda_\perp$.

Nous évaluons alors le corps de l'abstraction dans ce nouvel environnement. La nouvelle configuration est $(\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\perp\}, [t], \perp, y)$.

On repasse dans le cas de l'application qui met le sous-terme de droite sur la pile et qui nous rend l'évaluation du sous-terme de gauche. Ici, le sous-terme de gauche est une variable. Donc, on se place dans le cadre d'une variable. Il faut choisir parmi les quatre règles qui traitent les variables. Ici, notre variable est liée à un $\lambda_\perp$, donc on utilisera la règle Var-λ-NonFrozen qui ne fait rien de plus que d'appeler la règle s-omega.

Var-λ-NonFrozen

$$\frac{\Sigma(x) = \lambda_\perp \qquad (\Sigma, \Pi, \mu, x) \to_{\mathcal{S}_\omega} (\Sigma', t')}{(\Sigma, \Pi, \mu, x) \to (\Sigma', t')}$$

La règle s-omega dépile toute la pile pour recréer des applications sans toucher à l'environnement.

s-omega

$$\frac{}{(\Sigma, t_1 \cdot ... \cdot t_n, \mu, x) \to_{\mathcal{S}_\omega} (\Sigma, x\ t_1\ ...\ t_n)}$$

Ensuite, nous passons dans la deuxième branche de la règle Var-abs et nous allons évaluer le même terme $\lambda y.y\ t$ mais dans un contexte différent, et surtout en mode $\top$. La nouvelle configuration est $(\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\perp\}, Nil, \perp, y\ t)$ puis après Abs-nil, nous avons $(\{x \mapsto \lambda y.y\ t\}, Nil, \perp, \lambda y.y\ t)$. Ce nouveau passage sur le terme $\lambda y.yt$ correspond cette fois à une occurrence substituable.

Du fait que le mode change, la règle que l'on applique n'est plus tout à fait la même. Ici, nous allons utiliser la règle Var-λ-Frozen qui gère les variables qui sont liées à un $\lambda_\top$.

Et de la même manière que pour la règle Var-λ-NonFrozen, la règle Var-λ-Frozen se contente d'appeler la règle s-phi.

Var-λ-Frozen

$$\frac{\Sigma(x) = \lambda_\top \qquad (\Sigma, \Pi, \mu, x) \to_{\mathcal{S}_\varphi} (\Sigma', t')}{(\Sigma, \Pi, \mu, x) \to (\Sigma', t')}$$

s-phi

$$\frac{(\Sigma_0, Nil, \top, t_1) \to (\Sigma_1, t'_1) \qquad ... \qquad (\Sigma_{n-1}, Nil, \top, t_n) \to (\Sigma_n, t'_n)}{(\Sigma_0, t_1 \cdot ... \cdot t_n, \mu, x) \to_{\mathcal{S}_\varphi} (\Sigma_n, x\ t'_1\ ...\ t'_n)}$$

La nouvelle configuration est $(\{x \mapsto \lambda y.y\ t\}, Nil, \top, y\ t)$ puis $(\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\top\}, [t], \top, y)$.

La règle s-phi, comme la règle s-omega, dépile toute la pile pour refaire des applications, mais en plus, elle évalue tous les termes qui sont sur cette pile avant de les remettre sous la forme d'application. On note que tous les termes sont évalués en mode $\top$ et que l'environnement obtenu après évaluation du du k-ème terme est utilisé pour le $k+1$-ème terme.

On considère t comme un terme quelconque qui s'évalue en un terme t' et nous ne détaillerons pas son évaluation. La nouvelle configuration est $(\{x \mapsto \lambda y.y\ t'; y \mapsto \lambda_\top\}, Nil, \top, y\ t')$ puis $(\{x \mapsto \lambda y.y\ t'\}, Nil, \top, \lambda y.y\ t')$. Et pour finir le résultat est $(\emptyset, (\lambda y.y\ t')[x \setminus \lambda y.y\ t'])$

Pour parler de la règle de réduction Var-$\mathcal{S}$, qui est la quatrième et dernière règle pour les variables, nous utilisons un autre exemple qui est présenté dans la figure 6.2. Lorsqu'une variable est liée dans le contexte à un terme qui se réduit en une structure, on met à jour le terme lié à la variable avec le résultat de la réduction du terme, puis on invoque la règle s-phi ou s-omega, selon le terme, pour dépiler toute la pile et reconstruire une structure.

$$\text{Var-}\mathcal{S}\ \frac{\Sigma(x) = u \qquad (\Sigma, Nil, \bot, u) \to (\Sigma', u') \qquad \Sigma' \vdash u' \in \mathcal{S}_\alpha \qquad (\Sigma'[x \mapsto u'], \Pi, \mu, x) \to_{\mathcal{S}_\alpha} (\Sigma'', u'')}{(\Sigma, \Pi, \mu, x) \to (\Sigma'', u'')}$$

Nous avons les règles des structures illustrées dans la figure 6.4 qui sont exactement les mêmes que dans la version papier. Les deux premières règles distinguent si le terme est une structure φ ou une structure ω, tandis que les trois dernières règles sont les règles de propagation classiques des structures.

Pour finir, nous allons reparler de la règle Var-abs. La première fois que nous avons vu cette règle, nous n'avions qu'une abstraction à la place de u. Cependant, cette règle traite également le cas des réponses.

$$\text{Var-abs}\ \frac{\Sigma(x) = u \qquad (\Sigma, Nil, \bot, u) \to (\Sigma', u') \qquad u' = (\lambda y.t)\mathcal{L} \qquad (\Sigma'[x \mapsto u'], \Pi, \mu, u') \to (\Sigma'', u'')}{(\Sigma, \Pi, \mu, x) \to (\Sigma'', u'')}$$

Nous sommes obligés d'énoncer cette règle avec les réponses car notre règle sur les substitutions explicites reconstruit les substitutions explicites pour les raisons que nous avons déjà vues.

Cela a un impact sur l'efficacité de la machine. Si nous n'autorisions que des abstractions, toutes les réponses devraient être traitées en utilisant seulement cette règle. Cela signifie que toutes les substitutions explicites seraient mises dans l'environnement, ce qui permettrait de gérer le partage. Toutefois, en autorisant également les réponses, le partage est limité, ce qui peut entraîner le recalcul de certains termes plusieurs fois.

$\Sigma = \{x \mapsto \lambda y.y\ t\}$
$\Sigma' = \{x \mapsto \lambda y.y\ t'\}$

$$\textsc{APP}\ \dfrac{\textsc{Abs-cons}\ \dfrac{\textsc{ES}\ \dfrac{\textsc{Var-abs}\ \dfrac{\textcircled{1}(\Sigma, Nil, \bot, \lambda y.y\ t) \to (\Sigma, \lambda y.y\ t) \qquad \textcircled{2}(\Sigma, Nil, \top, \lambda y.y\ t) \to (\Sigma', \lambda y.y\ t')}{(\{x \mapsto \lambda y.y\ t\}, Nil, \top, x) \to (\Sigma', \lambda y.y\ t')}}{(\emptyset, Nil, \top, x[x \setminus \lambda y.y\ t]) \to (\emptyset, (\lambda y.y\ t')[x \setminus \lambda y.y\ t'])}}{(\emptyset, [\lambda y.y\ t], \top, \lambda x.x) \to (\emptyset, (\lambda y.y\ t')[x \setminus \lambda y.y\ t'])}}{(\emptyset, Nil, \top, (\lambda x.x)\ (\lambda y.y\ t)) \to (\emptyset, (\lambda y.y\ t')[x \setminus \lambda y.y\ t'])}$$

$$\textsc{Abs-nil}\ \dfrac{\textsc{APP}\ \dfrac{\textsc{Var-}\lambda\textsc{-NonFrozen}\ \dfrac{\textsc{s-omega}\ \dfrac{}{(\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\bot\}, [t], \bot, y) \to_{\mathcal{S}_\omega} (\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\bot\}, y\ t)}}{(\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\bot\}, [t], \bot, y) \to (\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\bot\}, y\ t)}}{(\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\bot\}, Nil, \bot, y\ t) \to (\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\bot\}, y\ t)}}{\textcircled{1}(\Sigma, Nil, \bot, \lambda y.y\ t) \to (\Sigma, \lambda y.y\ t)}$$

$$\textsc{Abs-nil}\ \dfrac{\textsc{APP}\ \dfrac{\textsc{Var-}\lambda\textsc{-Frozen}\ \dfrac{\textsc{s-phi}\ \dfrac{\dfrac{...}{(\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\top\}, Nil, \top, t) \to (\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\top\}, t')}}{(\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\top\}, [t], \top, y) \to_{\mathcal{S}_\varphi} (\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\top\}, y\ t')}}{(\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\top\}, [t], \top, y) \to (\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\top\}, y\ t')}}{(\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\top\}, Nil, \top, y\ t) \to (\{x \mapsto \lambda y.y\ t; y \mapsto \lambda_\top\}, y\ t')}}{\textcircled{2}(\Sigma, Nil, \top, \lambda y.y\ t) \to (\Sigma', \lambda y.y\ t')}$$

Figure 6.1 – Réduction de $(\lambda x.x)(\lambda y.y\ t)$

$\Sigma = \{x \mapsto \lambda_\top; y \mapsto x\ t_1\}$
$\Sigma' = \{x \mapsto \lambda_\top; y \mapsto x\ t_1; ...\}$
$\Sigma'' = \{x \mapsto \lambda_\top; y \mapsto x\ s_1; ...\}$

$$\textsc{Var-}\mathcal{S}\ \dfrac{\dfrac{...}{(\Sigma, [t_1; t_2], \bot, x\ t_1) \to (\Sigma', x\ t'_1)} \quad \dfrac{\overline{\Sigma' \vdash x \in \mathcal{S}_\varphi}}{\Sigma' \vdash x\ t'_1 \in \mathcal{S}_\varphi} \quad ①(\Sigma'', [t_2; t_3], \top, y) \to_{\mathcal{S}_\varphi} (\Sigma''', y\ t'_2\ t'_3)}{(\Sigma, [t_2; t_3], \top, y) \to (\Sigma''', y\ t'_2\ t'_3)}$$

$$\textsc{s-phi}\ \dfrac{\dfrac{...}{(\Sigma'', Nil, \top, t_2) \to (\Sigma'''', t'_2)} \quad \dfrac{...}{(\Sigma'''', Nil, \top, t_3) \to (\Sigma''', t'_3)}}{①(\Sigma'', [t_2; t_3], \top, y) \to_{\mathcal{S}_\varphi} (\Sigma''', y\ t'_2\ t'_3)}$$

FIGURE 6.2 – Réduction de y qui est associée à $x\ t_1$ avec une pile $t_2\ t_3$

6.3 Détails d'implémentation

Pour commencer, toute notre implémentation est faite en OCaml et nous utilisons les indices de De Bruijn pour représenter les variables. En ce qui concerne l'environnement, la liste et la définition des termes, nous avons déjà vu leur définition, mais nous avons apporté quelques modifications pratiques pour rendre l'implémentation plus efficace.

Tout d'abord, dans la définition des termes, nous avons ajouté un constructeur supplémentaire : le constructeur `Lift`. Ce constructeur permet d'effectuer un lifting paresseux, où seules les variables utilisées sont liftées car de manière plus générale on n'applique le lifting que sur les sous-termes que l'on évalue. Nous avons introduit cette optimisation car nous avons constaté que le lifting prenait une part importante du temps d'exécution. En effet, lorsqu'un terme est déplacé, il est important de s'assurer que toutes les variables pointent vers le bon lieur, ce qui nécessite un lifting des indices de De Bruijn. Cependant, certaines variables ne seront plus utilisées et il est donc inutile de les lifter.

Le constructeur `Lift` prend deux entiers en paramètre : le premier indique de combien les variables doivent être liftées et le deuxième indique à partir de quel indice les variables doivent être liftées.

```
type term_es =
  | Lift of term_es * int * int
  | EVar of int
  | EAbs of term_es
  | EApp of term_es * term_es
```

```
  | ES    of term_es * term_es
```

L'environnement est géré par une liste d'éléments de type `e_env` ainsi qu'un entier qui correspond à la taille de cette liste. Cette taille est maintenue tout au long de l'exécution afin d'éviter d'avoir à la recalculer à chaque fois.

```
type env =  e_env list * int
```

Les éléments de l'environnement sont les mêmes que définis précédemment mais le constructeur `Bind` est maintenant paramétré par une référence vers un élément de type `bind`.

```
type e_env =
  | Bind of bind ref
  | Lambda
  | Lambda_Frozen

type bind =
  | NE of closure
  | E  of evaluated_closure
```

Le type `bind` contient deux constructeurs : `NE` pour non évalué et `E` pour évalué. En effet, pour optimiser l'exécution, nous distinguons les fermetures qui ont déjà été évaluées et qui sont en forme normale locale des fermetures qui ne l'ont pas encore été. Cette distinction nous permet de ne parcourir que les termes en forme `NE` pour y trouver une réduction . Par ailleurs, on constate que le constructeur `NE` est paramétré par une fermeture, tandis que le constructeur `E` est paramétré par un nouveau type `evaluated_closure`.

```
type closure = term_es * int
type evaluated_closure = term_es * int * head
```

Les fermetures ne sont plus paramétrées par un terme et un environnement, mais par un terme et un entier. C'est-à-dire que lorsque l'on sauvegarde un terme, on le sauvegarde avec la taille de l'environnement à ce moment-là. Lorsque l'on va réutiliser ce terme, on sait que l'on ne doit prendre que les n plus vieux éléments de l'environnement. Cette optimisation nous permet de ne pas dupliquer l'environnement.

Le nouveau type `closure_head` est paramétré par un terme et un entier de la même manière que le type `closure`, mais il a aussi un élément de type `head` en plus.

```
type head =
  | Lam
  | S
  | SO
```

Le type `head` contient trois constructeurs : `Lam`, `S` et `SO`, respectivement pour représenter les abstractions, les structures et les structures ω. Ce type nous permet de marquer les fermetures pour savoir quel est leur élément de tête, car parfois, cela peut être long de trouver la forme du terme. Et comme dans certains cas, nous avons besoin de connaître la forme du terme, nous avons cette marque qui nous permet de ne pas repartir à zéro inutilement. En effet, si la fermeture est de type `E`, cela veut dire que nous avons déjà parcouru le terme et que nous avons déjà cette information.

Dans la figure 6.5, on peut voir le squelette de notre fonction d'évaluation. Pour des raisons de lisibilité c'est une version sans lifting, sans représentation des environnements par un entier et sans distinction entre les termes déjà évalués (`E`) et les termes non évalués (`NE`). Notre fonction prend un environnement nommé `vars`, qui est de type `env`, une pile de fermetures nommée `args`, qui est de type `closure list`, un mode `m` de type `mode` et un terme `t` (ligne 8). Elle renvoie une paire composée d'un terme en forme normale et de la forme du terme représentée par le type `head` (ligne 9).

Ensuite, nous présentons le corps de la fonction. Nous commençons par regarder le terme à évaluer (ligne 11). Si le terme est une variable (ligne 12), nous allons voir dans l'environnement (ligne 14) si cette variable est liée à un terme (dans le cas où elle est liée à une substitution explicite) (ligne 15) ou si elle est liée à une abstraction (gelée ou non) (lignes 26-27).

Dans le cas où la variable est liée à un terme, on évalue ce terme dans le contexte dans lequel il a été enregistré (ligne 17) et on met à jour la variable dans l'environnement (ligne 18). Ensuite, si le terme retourné est une abstraction (ligne 21), nous renvoyons le terme évalué dans le contexte actuel, ce qui revient à effectuer la substitution (d'une occurence). Si le terme est une structure gelée (ligne 22), nous déplions la pile d'arguments à l'aide de la fonction `depile_struct` qui évalue chaque terme déplié. Sinon (ligne 23), `depile_struct_omega` qui ne les évalue pas.

Si la variable n'est pas liée à une abstraction mais à une structure, nous sommes exactement dans le même cas que précédemment. Si le terme évalué est une structure, on dépile (lignes 26-27).

Ensuite, si le terme est une abstraction (ligne 30), il existe deux cas : le cas où la pile d'arguments est vide (ligne 32) et le cas où elle ne l'est pas (ligne 34). Si la pile d'arguments est vide, nous évaluons le corps en mettant dans l'environnement la variable liée, puis nous reconstruisons l'abstraction avec le corps évalué (lignes 32-33). En revanche, si la pile n'est pas vide, nous allons déclencher ce qui correspond à la règle dB. Cela signifie que nous allons dépiler un argument pour le transformer en une substitution explicite et l'évaluer (ligne 34).

Le cas de l'application est très simple : nous mettons le sous-terme de droite sur la pile et nous évaluons le sous-terme de gauche (ligne 36).

Dans le cas où le terme est une substitution explicite (ligne 38), nous mettons le corps de la substitution dans l'environnement (ligne 39), puis nous évaluons le sous-terme de gauche (ligne 40). Ensuite, nous récupérons le terme qui se trouve dans l'environnement, qui a été mis à jour par l'évaluation, afin de reconstruire une substitution explicite (ligne 41) pour les raisons que nous avons déjà vues au moment de présenter la règle ES dans la section 6.2.

Il nous reste encore deux principes à examiner qui nous permettrons d'optimiser la machine abstraite : le lifting paresseux, représenté comme nous l'avons vu précédemment par un constructeur `Lift`, et la méthode que nous utilisons pour enregistrer l'environnement dans une fermeture.

Pour illustrer ces deux principes, nous utilisons la figure 6.6 qui représente l'évolution de trois éléments : le terme (Terme), la pile d'arguments (Args) et l'environnement (Vars). Les termes sont représentés par des indices de De Bruijn et la fonction

de lifting (Lift) prend trois arguments : le terme à lifter, de combien il faut le lifter et à partir de quelle profondeur. La pile d'arguments stocke des fermetures, c'est-à-dire des termes avec un entier qui correspond au nombre d'éléments dans l'environnement au moment où la fermeture a été mise sur la pile. Et l'environnement décrit à quoi sont liées les variables : `LF` pour une abstraction gelée, `L` pour une abstraction non gelée et `B` pour dire que la variable est liée à une fermeture. Commençons par parler des suffixes. Pour ne pas stocker des copies de l'environnement lorsque l'on enregistre une fermeture, on enregistre la taille de l'environnement à ce moment-là. En effet, étant donné que l'environnement ne peut pas diminuer, si l'on reprend les n premiers termes de l'environnement, on sait que l'on aura l'état de l'environnement au moment où l'on a enregistré la fermeture. On peut voir dans la figure ligne 6 que l'on met le terme $\lambda.0$ sur la pile et que, à ce moment, l'environnement Vars est de taille 1, donc on enregistre avec le nombre 1 et on obtient ainsi la fermeture $(\lambda.0 * 1)$. Puis, ligne 8, lorsque l'on va chercher la fermeture correspondante, on prend comme environnement les n premiers éléments. On voit donc ligne 9 que l'on prend le premier élément de l'environnement.

On ne redéfinit pas la notion de lifting, qui est un sujet à part entière bien connu. Mais nous considérons ici l'aspect paresseux du lifting. Nous créons un lifting paresseux dans l'environnement à la ligne 7, mais comme ce terme n'est jamais utilisé, le lifting n'est jamais effectué réellement. Cela nous permet d'éviter de faire un lifting de variables inutiles. Nous sommes parvenus à cette optimisation car le lifting des substitutions explicites devenait très coûteux.

6.4 Conclusion

Notre machine abstraite n'est pas tout à fait fidèle à notre calcul $\lambda_{\mathsf{sn+}}$ en raison de la règle Var-abs qui nous fait perdre un peu de partage en autorisant la substitution de réponse. Cependant, cela n'a qu'un impact sur l'efficacité du calcul, et il y a de bonnes chances que la machine ait les mêmes propriétés de correction et de complétude que le calcul. De plus, nous avons validé notre machine par rapport à une machine de référence en appel par nécessité itéré.

$$\textsc{Abs-cons}\ \frac{(\Sigma, \Pi, \mu, t[x \setminus u]) \to (\Sigma', t')}{(\Sigma, u \cdot \Pi, \mu, \lambda x.t) \to (\Sigma', t')} \qquad \textsc{Abs-nil}\ \frac{(\Sigma; x \mapsto \lambda_\mu, Nil, \mu, t) \to (\Sigma'; x \mapsto \lambda_\mu, t')}{(\Sigma, Nil, \mu, \lambda x.t) \to (\Sigma', \lambda x.t')}$$

$$\text{APP}\ \frac{(\Sigma, u \cdot \Pi, \mu, t) \to (\Sigma', t')}{(\Sigma, \Pi, \mu, t\ u) \to (\Sigma', t')} \qquad \text{ES}\ \frac{(\Sigma; x \mapsto u, \Pi, \mu, t) \to (\Sigma'; x \mapsto u', t')}{(\Sigma, \Pi, \mu, t[x \setminus u]) \to (\Sigma', t'[x \setminus u'])}$$

$$\textsc{Var-}\lambda\textsc{-Frozen}\ \frac{\Sigma(x) = \lambda_\top \qquad (\Sigma, \Pi, \mu, x) \to_{\mathcal{S}_\varphi} (\Sigma', t')}{(\Sigma, \Pi, \mu, x) \to (\Sigma', t')}$$

$$\textsc{Var-}\lambda\textsc{-NonFrozen}\ \frac{\Sigma(x) = \lambda_\bot \qquad (\Sigma, \Pi, \mu, x) \to_{\mathcal{S}_\omega} (\Sigma', t')}{(\Sigma, \Pi, \mu, x) \to (\Sigma', t')}$$

$$\textsc{Var-abs}\ \frac{\Sigma(x) = u \qquad (\Sigma, Nil, \bot, u) \to (\Sigma', u') \qquad u' = (\lambda y.t)\mathcal{L} \qquad (\Sigma'[x \mapsto u'], \Pi, \mu, u') \to (\Sigma'', u'')}{(\Sigma, \Pi, \mu, x) \to (\Sigma'', u'')}$$

$$\textsc{Var-}\mathcal{S}\ \frac{\Sigma(x) = u \qquad (\Sigma, Nil, \bot, u) \to (\Sigma', u') \qquad \Sigma' \vdash u' \in \mathcal{S}_\alpha \qquad (\Sigma'[x \mapsto u'], \Pi, \mu, x) \to_{\mathcal{S}_\alpha} (\Sigma'', u'')}{(\Sigma, \Pi, \mu, x) \to (\Sigma'', u'')}$$

$$\textsc{s-phi}\ \frac{(\Sigma_0, Nil, \top, t_1) \to (\Sigma_1, t_1') \qquad \dots \qquad (\Sigma_{n-1}, Nil, \top, t_n) \to (\Sigma_n, t_n')}{(\Sigma_0, t_1 \cdot \dots \cdot t_n, \mu, x) \to_{\mathcal{S}_\varphi} (\Sigma_n, x\ t_1' \dots t_n')}$$

$$\textsc{s-omega}\ \frac{}{(\Sigma, t_1 \cdot \dots \cdot t_n, \mu, x) \to_{\mathcal{S}_\omega} (\Sigma, x\ t_1 \dots t_n)}$$

FIGURE 6.3 – Sémantique à grand pas pour la machine abstraite

$$\frac{}{\Sigma; x \mapsto \lambda_\top \vdash x \in \mathcal{S}_\varphi} \qquad \frac{}{\Sigma; x \mapsto \lambda_\bot \vdash x \in \mathcal{S}_\omega}$$

$$\frac{\Sigma \vdash t \in \mathcal{S}_\alpha}{\Sigma; x \mapsto t \vdash x \in \mathcal{S}_\alpha} \qquad \frac{\Sigma \vdash t_1 \in \mathcal{S}_\alpha}{\Sigma \vdash t_1\ t_2 \in \mathcal{S}_\alpha} \qquad \frac{\Sigma; x \mapsto u \vdash t \in \mathcal{S}_\alpha}{\Sigma \vdash t[x \setminus u] \in \mathcal{S}_\alpha}$$

FIGURE 6.4 – Caractérisation des structures pour la machine abstraite

```
1  let depile_struct_omega args n  =
2    List.fold_left (fun a t -> EApp(a,t)) (EVar n) args
3
4  let rec  depile_struct vars args n =
5    List.fold_left (fun a t -> EApp(a,(fst eval vars [] Top t)))
6      (EVar n) args
7
8  and eval (vars:env) (args:closure list) (m:mode) (t:term_es)
9                : (term_es * head) =
10
11 match t with
12 | EVar n ->
13
14     match List.nth vars n with
15     | Bind (t,vars') ->
16       let r = !t in
17       let (t',h) = eval vars' [] Bot r in
18       t := t';
19       begin
20         match h with
21         | Lam -> eval vars args m t'
22         | S ->  (depile_struct vars  args n,S)
23         | SO -> (depile_struct_omega args n,SO)
24       end
25
26     | Lambda_Frozen -> (depile_struct vars  args n,S)
27     | Lambda -> (depile_struct_omega args n,SO)
28
29
30 | EAbs t ->
31   match args with
32   | [] -> let var = if m == Top then Lambda_Frozen else Lambda in
33           let t' = fst (eval (var::vars) args m t) in (EAbs t',Lam)
34   | u :: args -> eval vars args m (ES(t, u))
35
36 | EApp (t, u) -> eval vars args m (u::args) t
37
38 | ES (t, u) ->
39   let u' = ref u in
40   let (t',h) = eval ((Bind (u',vars))::vars) args m t in
41   (ES(t',!u'),h)
```

FIGURE 6.5 – Squelette de la fonction d'évaluation

N°	Terme	Args	Vars
1	$\lambda.((\lambda.\lambda.1)\ (\lambda.0))\ (\lambda.1)$	[ ]	[ ]
2	$((\lambda.\lambda.1)\ (\lambda.0))\ (\lambda.1)$	[ ]	[LF]
3	$(\lambda.\lambda.1)\ (\lambda.0)$	[(λ.1 * 1)]	[LF]
4	$\lambda.\lambda.1$	[(λ.0 * 1) ;(λ.1 * 1)]	[LF]
5	(λ.1)[/λ.0]	[(λ.1 * 1)]	[LF]
6	$\lambda.1$	[(λ.1 * 1)]	[B(λ.0 * 1) ;LF]
7	(1)[/Lift(λ.1,1,0)]	[ ]	[B(λ.0 * 1) ;LF]
8	1	[ ]	[B(Lift(λ.1,1,0) * 2) ;B(λ.0 * 1) ;LF]
9	$\lambda.0$	[ ]	[LF]
10	0	[ ]	[L ;LF]
11	Lift(λ.0,2,0) = λ.0	[ ]	[B(Lift(λ.1,1,0) * 2) ;B(λ.0 * 1) ;LF]
12	Lift(0,2,1) = 0	[ ]	[LF ;B(Lift(λ.1,1,0) * 2) ;B(λ.0 * 1) ;LF]

FIGURE 6.6 – Fonctionnement du suffixe et du lifting sur l'exemple $\lambda y.((\lambda z.\lambda w.z)\ (\lambda x.x))(\lambda k.y)$

Chapitre 7

Conclusion

7.1 Contribution

Ma thèse représente une avancée significative dans le domaine des stratégies de réduction. En contribuant à la compréhension et à l'amélioration des stratégies de réduction du lambda-calcul, en particulier l'appel par nécessité en réduction forte.

Nous avons créé la première caractérisation d'une famille de stratégies de réduction forte différentes des stratégies faibles itérées que nous avons appelé λ_{sn}. Pour ce faire, nous avons créé une approximation des positions nécessaires qui va également chercher les radicaux sous les abstractions. Pour cela, nous avons mis en place un système d'annotation des variables que nous avons nommé «variable gelée».

Nous avons prouvé la correction de notre calcul λ_{sn} par rapport au lambda-calcul en utilisant trois lemmes principaux : un lemme de simulation, un lemme de caractérisation des formes normales et un lemme de dépliage des formes normales. Nous avons également prouvé la complétude de notre calcul λ_{sn} par rapport au lambda-calcul grâce à des résultats de la littérature, notamment un calcul hôte λ_{c} et un système de types avec intersection non-idempotente. Ces deux résultats nous disent que si un terme admet une forme normale dans le lambda-calcul, alors toute réduction dans λ_{sn} atteint cette forme normale. Ces résultats nous ont aussi permis de prouver une notion de nécessité des réductions. De plus, nous faisons la conjecture que le calcul est bien confluent, modulo la règle gc.

Ensuite, nous avons défini un nouveau calcul $\lambda_{\mathsf{sn+}}$ pour réduire encore le nombre de calculs dupliqués grâce à une restriction sur le critère de substitution des termes.

Nous avons tenté de prouver la complétude de notre calcul $\lambda_{\mathsf{sn+}}$: on sait par construction que les réductions de $\lambda_{\mathsf{sn+}}$ sont incluses dans les réductions de λ_{sn}, mais nous n'avons pas pu compléter la preuve qui démontre que les formes normales de $\lambda_{\mathsf{sn+}}$ correspondent à celles de λ_{sn}, en raison du problème évoqué dans le chapitre 4. Nous avons formalisé une variante très proche de notre calcul $\lambda_{\mathsf{sn+}}$ et prouvé sa correction grâce à l'assistant de preuve Abella. Par ailleurs, nous avons prouvé la propriété du diamant sur notre calcul $\lambda_{\mathsf{sn+}}$.

Enfin, nous avons implémenté une machine abstraite pour ce calcul afin de le comparer à la stratégie de normalisation traditionnelle. Bien que l'implémentation de notre machine ne soit pas finie, cela nous a permis de faire quelques benchmarks

notamment en terme de nombre de réductions effectuées.

7.2 Pour la suite

Avant tout, il est nécessaire de compléter notre caractérisation des termes irréductibles afin de résoudre le problème qui est décrit dans le chapitre 4.

Puis par la suite, il reste à formaliser la preuve de confluence, qui ne devrait pas être très difficile, mais qui prendra vraisemblablement beaucoup de temps car il y a beaucoup de cas à traiter. En effet, pour la preuve de confluence, l'objectif est de tester toutes les possibilités pour qu'un terme soit réduit en une étape en deux termes différents. Donc, dans le pire des cas, nous devons traiter toutes les règles de réduction au carré. Il reste également, la preuve de complétude qui semble être plus difficile et d'une nature différente des preuves précédentes, comme nous pouvons le voir dans la version papier de la preuve. En effet, elle utilise un système de types avec intersection non-idempotente. Par conséquent, en plus de devoir faire la preuve, il faut formaliser tout le matériel utilisé. Cela rendra cette tâche assez longue et complexe.

Une chose à faire serait d'implémenter une machine abstraite correcte pour $\lambda_{\mathsf{sn}+}$. En effet, notre machine abstraite actuelle n'est pas entièrement conforme à notre calcul $\lambda_{\mathsf{sn}+}$ en raison de la règle Var-abs qui nous fait perdre un peu de partage en autorisant les réponses. Une autre possibilité pour la suite consisterait à implémenter un générateur de termes plus efficace, afin de pouvoir tester notre machine sur des ensembles de termes plus volumineux. Car pour le moment nous avons un prototype qui nous permet d'évaluer les gains en termes de réduction, mais si nous voulons de réels gains en terme de temps, il nous faut une véritable implémentation.

Une autre chose à finir serait de faire la preuve de minimalité sur le calcul $\lambda_{\mathsf{sn}+}$.

7.3 À étudier

Un autre aspect à prendre en considération est l'adaptation de la machine afin de tester la convertibilité entre les termes. Cela implique des étapes supplémentaires dans notre séquence de réduction, pour ne pas attendre jusqu'à la fin de la réduction pour savoir si les deux termes sont compatibles. Cela permettrait d'utiliser notre calcul $\lambda_{\mathsf{sn}+}$ dans, par exemple, un assistant de preuve basé sur la théorie des types.

Comme nous l'avons déjà dit, le problème de savoir si un terme est à une position nécessaire est un problème indécidable, ce qui signifie qu'il existe toujours une solution plus efficace. Nous proposons ici une solution, mais il reste du travail possible pour améliorer cette approximation. Par exemple, il serait possible d'améliorer le calcul en ne faisant pas l'hypothèse que les termes qui sont dans une substitution explicite sont systématiquement appliqués, mais d'avoir une analyse plus fine qui nous permettrait de pouvoir faire varier le mode et donc pouvoir éventuellement trouver plus de termes nécessaires. Où encore, comme l'ont montré les règles de la machine abstraite, notre calcul peut éviter de remplacer les variables qui ne sont pas appliquées (règle LSV-BASE), suivant [32, 3], mais cela ouvre la question de la façon de caractériser les formes normales alors. Une autre piste à explorer est la façon dont ce travail interagit

avec le partage pleinement paresseux, qui évite plus de duplication mais dont les propriétés sont étroitement liées à la réduction faible [7].

Il est aussi possible de voir ce que pourrait donner notre calcul $\lambda_{\mathsf{sn+}}$ sur d'autres types de lambda-termes comme par exemple avec le lambda-calcul simplement typé. Par exemple, savoir si le fait de ne pas avoir à traiter les termes qui divergent nous permettrait de trouver plus de positions nécessaires.

Bibliographie

[1] Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. A strong distillery. In Xinyu Feng and Sungwoo Park, editors, Programming Languages and Systems, volume 9458 of Lecture Notes in Computer Science, pages 231–250, 2015. doi:10.1007/978-3-319-26529-2_13.

[2] Beniamino Accattoli, Eduardo Bonelli, Delia Kesner, and Carlos Lombardi. A nonstandard standardization theorem. In 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, page 659–670, 2014. doi:10.1145/2535838.2535886.

[3] Beniamino Accattoli, Andrea Condoluci, and Claudio Sacerdoti Coen. Strong call-by-value is reasonable, implosively, February 2021. arXiv:2102.06928.

[4] Beniamino Accattoli and Delia Kesner. The structural λ-calculus. In Anuj Dawar and Helmut Veith, editors, Computer Science Logic, pages 381–395, 2010. doi:10.5555/1887459.1887491.

[5] Zena M. Ariola, John Maraist, Martin Odersky, Matthias Felleisen, and Philip Wadler. A call-by-need lambda calculus. In 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '95, pages 233–246, 1995. doi:10.1145/199448.199507.

[6] David Baelde, Kaustuv Chaudhuri, Andrew Gacek, Dale Miller, Gopalan Nadathur, Alwen Tiu, and Yuting Wang. Abella: A system for reasoning about relational specifications. Journal of Formalized Reasoning, 7(2):1–89, December 2014. doi:10.6092/issn.1972-5787/4650.

[7] Thibaut Balabonski. A unified approach to fully lazy sharing. In John Field and Michael Hicks, editors, 39th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL, pages 469–480, January 2012. doi:10.1145/2103656.2103713.

[8] Thibaut Balabonski. Weak optimality, and the meaning of sharing. In Greg Morrisett and Tarmo Uustalu, editors, ACM SIGPLAN International Conference on Functional Programming, ICFP'13, pages 263–274, September 2013. doi:10.1145/2500365.2500606.

[9] Thibaut Balabonski, Pablo Barenbaum, Eduardo Bonelli, and Delia Kesner. Foundations of strong call by need. Proc. ACM Program. Lang., 1(ICFP), August 2017. doi:10.1145/3110264.

[10] Hendrik Pieter Barendregt, J Bergstra, Jan Willem Klop, and Henri Volken. Some notes on lambda reduction. Degrees, reductions, and representability in the lambda calculus. Preprint, (22):13–53, 1976.

[11] Malgorzata Biernacka, Dariusz Biernacki, Witold Charatonik, and Tomasz Drab. An abstract machine for strong call by value. In Bruno C. d. S. Oliveira, editor, Programming Languages and Systems - 18th Asian Symposium, APLAS 2020, volume 12470 of Lecture Notes in Computer Science, pages 147–166. Springer, 2020. doi:10.1007/978-3-030-64437-6_8.

[12] Malgorzata Biernacka and Witold Charatonik. Deriving an Abstract Machine for Strong Call by Need. In Herman Geuvers, editor, 4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019), volume 131 of Leibniz International Proceedings in Informatics (LIPIcs), pages 8:1–8:20, 2019. doi:10.4230/LIPIcs.FSCD.2019.8.

[13] Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. Non-idempotent intersection types for the Lambda-Calculus. Logic Journal of the IGPL, 25(4):431–464, 07 2017. doi:10.1093/jigpal/jzx018.

[14] Stephen Chang and Matthias Felleisen. The call-by-need lambda calculus, revisited. In Helmut Seidl, editor, Programming Languages and Systems, pages 128–147. Springer Berlin Heidelberg, 2012.

[15] Arthur Charguéraud. The locally nameless representation. Journal of Automated Reasoning, 49(3):363–408, October 2012. doi:10.1007/s10817-011-9225-2.

[16] Adam Chlipala. Parametric higher-order abstract syntax for mechanized semantics. In 13th ACM SIGPLAN International Conference on Functional Programming, ICFP, pages 143–156, September 2008. doi:10.1145/1411204.1411226.

[17] Maddalena Coppo and Mariangiola Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. Notre Dame Journal of Formal Logic, 21(4):685–693, 1980. doi:10.1305/ndjfl/1093883253.

[18] Pierre Crégut. An abstract machine for lambda-terms normalization. In ACM Conference on LISP and Functional Programming, LFP '90, page 333–340, 1990. doi:10.1145/91556.91681.

[19] Daniel de Carvalho. Sémantiques de la logique linéaire et temps de calcul. PhD thesis, Université Aix-Marseille II, 2007.

[20] Philippa Gardner. Discovering needed reductions using type theory. In Masami Hagiya and John C. Mitchell, editors, Theoretical Aspects of Computer Software, pages 555–574, 1994.

[21] Benjamin Grégoire and Xavier Leroy. A compiled implementation of strong reduction. In 7th ACM SIGPLAN International Conference on Functional Programming, ICFP '02, page 235–246, 2002. doi:10.1145/581478.581501.

[22] Barendregt Hendrik Pieter. The Lambda Calculus – Its Syntax and Semantics. North-Holland Publishing Company, Amsterdam, revised edition, 1984.

[23] Carsten Kehler Holst and Darsten Krogh Gomard. Partial evaluation is fuller laziness. In ACM SIGPLAN Symposium on Partial Evaluation and Semantics-

Based Program Manipulation, PEPM ’91, page 223–233, 1991. doi:10.1145/115865.115890.

[24] Delia Kesner. A theory of explicit substitutions with safe and full composition. Logical Methods in Computer Science, 5(3), May 2009. doi:10.2168/LMCS-5(3:1)2009.

[25] Delia Kesner. Reasoning about call-by-need by means of types. In Bart Jacobs and Christof Löding, editors, Foundations of Software Science and Computation Structures, pages 424–441, 2016. doi:10.1007/978-3-662-49630-5_25.

[26] Delia Kesner and Daniel Ventura. Quantitative types for the linear substitution calculus. In Josep Diaz, Ivan Lanese, and Davide Sangiorgi, editors, Theoretical Computer Science, volume 8705 of Lecture Notes in Computer Science, pages 296–310, 2014. doi:10.1007/978-3-662-44602-7_23.

[27] John Maraist, Martin Odersky, and Philip Wadler. The call-by-need lambda calculus. J. Funct. Program., 8(3):275–317, May 1998. doi:10.1017/S0956796898003037.

[28] Robin Milner. Local bigraphs and confluence: Two conjectures. Electron. Notes Theor. Comput. Sci., 175(3):65–73, June 2007. doi:10.1016/j.entcs.2006.07.035.

[29] Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. Theoretical Computer Science, 1(2):125–159, 1975. doi:10.1016/0304-3975(75)90017-1.

[30] Gordon D. Plotkin. A structural approach to operational semantics. Technical report, DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[31] Christopher P. Wadsworth. Semantics and Pragmatics of the Lambda Calculus. PhD thesis, Oxford, 1971.

[32] Nobuko Yoshida. Optimal reduction in weak-lambda-calculus with shared environments. In Conference on Functional Programming Languages and Computer Architecture, FPCA ’93, page 243–252, 1993. doi:10.1145/165180.165217.

Index

Abella, 105
Alpha-conversion, 18
Appel par nom, 24
Appel par nécessité, 25
Appel par valeur, 22

Bêta-réduction, 17

Calcul, 19
Capture, 17
Complétude, 79
Confluence, 20
Contexte, 37
Contexte d'évaluation, 38
Convention de Barendregt, 18

Diamant, 98
Dépliage, 48

Forme normale, 25
Forme normale locale, 89

Indice de de Bruijn, 19

Lambda-calcul, 15
Lifting, 19

Mode, 39

Nabla, 107
Normalisation, 25

Position isolée, 34
Position nécessaire, 30

Règle dB, 46
Règle gc, 47
Règle lsv, 46
Réduction, 17
Réponse, 65

Stratégie d'évaluation, 22
Stratégie de réduction, 28
Structure, 35, 49
Substitution, 16
Substitution explicite, 44

Terme fortement normalisant, 26
Terme normalisable, 26
Top-level, 16

Valeur, 16
Valeur ouverte, 38
Variable gelée, 34
Variable libre, 16
Variable vivante, 64