From proof terms to programs : . an operational and quantitative study of intuistionistic Curry-Howard calculi

Loîc Peyrot

From Proof Terms to Programs

An operational and quantitative study of intuitionistic Curry-Howard calculi

Par Loïc Peyrot

Thèse de doctorat d'informatique

Université Paris Cité

Institut de recherche en informatique fondamentale
École doctorale 386 – Sciences mathématiques de Paris Centre

Dirigée par **Delia Kesner**

Présentée et soutenue publiquement le 18 novembre 2022 devant un jury composé de :

Mme Silvia Ghilezan	Professor, Univerzitet u Novom Sadu	Rapporteuse
M. Willem Heijltjes	Senior Lecturer, University of Bath	Rapporteur
M. Giulio Guerrieri	Maître de conférence, Université d'Aix-Marseille	Examinateur
M. Damiano Mazza	Directeur de recherche CNRS, Université Sorbonne Paris Nord	Examinateur
Mme Femke van Raamsdonk	Assistant Professor, Vrije Universiteit Amsterdam	Examinatrice
Mme Delia Kesner	Professeure, Université Paris Cité	Directrice de thèse
M. José Espírito Santo	Assistant Professor, Universidade do Minho	Membre invité

Résumé. Le λ-calcul est un modèle mathématique des langages de programmation fonctionnels, avec un accent sur l'application de fonctions. Il est intéressant de considérer des variants du λ-calcul pour modéliser des comportements calculatoires spécifiques.

Le λ-calcul atomique et les λ-calculs avec applications généralisées sont deux variants (indépendants) du λ-calcul provenant d'interprétations calculatoires de la théorie de la démonstration. Alors que les langages de programmation sont basés sur des stratégies d'évaluation déterministes spécifiques, la littérature existante sur le λ-calcul atomique et les applications généralisées ne s'étendent que sur la théorie générale des calculs. En particulier, la réduction des termes n'est pas restreinte, et seulement analysée qualitativement. Cela induit un écart entre la théorie et la pratique, que nous cherchons à diminuer dans cette thèse.

À partir du λ-calcul atomique, nous isolons la notion la plus saillante de sa sémantique opérationnelle, que nous appelons réplication par nœuds. C'est une procédure de substitution particulière, qui duplique les termes finement, un nœud de l'arbre syntaxique à la fois. Nous poursuivons avec les λ-calculs à applications généralisées. Ceux-ci utilisent une application ternaire qui ajoute une continuation à l'application binaire habituelle. Dans ce travail, nous développons les théories opérationnelles basées sur la réplication par nœuds et les applications généralisées séparément. Pour les deux : À un niveau opérationnel, nous donnons plusieurs stratégies d'évaluation, toutes observationnellement équivalentes aux stratégies correspondantes du λ-calcul. À un niveau logique, notre approche est guidée par les types quantitatifs. Nous définissons différents systèmes de types qui caractérisent des propriétés sémantiques par induction, mais donnent aussi des bornes quantitatives sur la longueur de réduction et la taille des formes normales.

Plus précisément, dans la première partie de cette thèse, nous implémentons la réplication par nœuds au moyen d'un calcul à substitutions explicites. Nous montrons en particulier comment la réplication par nœuds peut être utilisée pour implémenter la pleine paresse, une stratégie d'évaluation bien connue de langages de programmations comme Haskell. Nous montrons des propriétés d'équivalence observationnelle reliant la sémantique pleinement paresseuse aux sémantiques usuelles. Dans la deuxième partie de cette thèse, nous commençons par une caractérisation opérationnelle et logique de la solvabilité dans les λ-calculs à applications généralisées. Nous montrons comment ce cadre donne naissance à une remarquable sémantique opérationnelle de l'appel-par-valeur. La caractérisation en appel-par-nom s'appuie sur un nouveau calcul à applications généralisées. Nous prouvons dans les deux cas que les sémantiques opérationnelles sont compatibles avec un modèle quantitatif, au contraire de celle du calcul en appel-par-nom originel. Nous prouvons ensuite des propriétés essentielles de ce nouveau calcul en appel-par-nom, et montrons l'équivalence observationnelle avec l'original.

Mots-clefs : λ-calcul, réécriture, réplication par nœuds, applications généralisées, types intersection, substitutions explicites.

Abstract. The λ-calculus is a mathematical model of functional programming languages, with an emphasis on function application. Variants of the calculus are of interest to model specific computational behaviors.

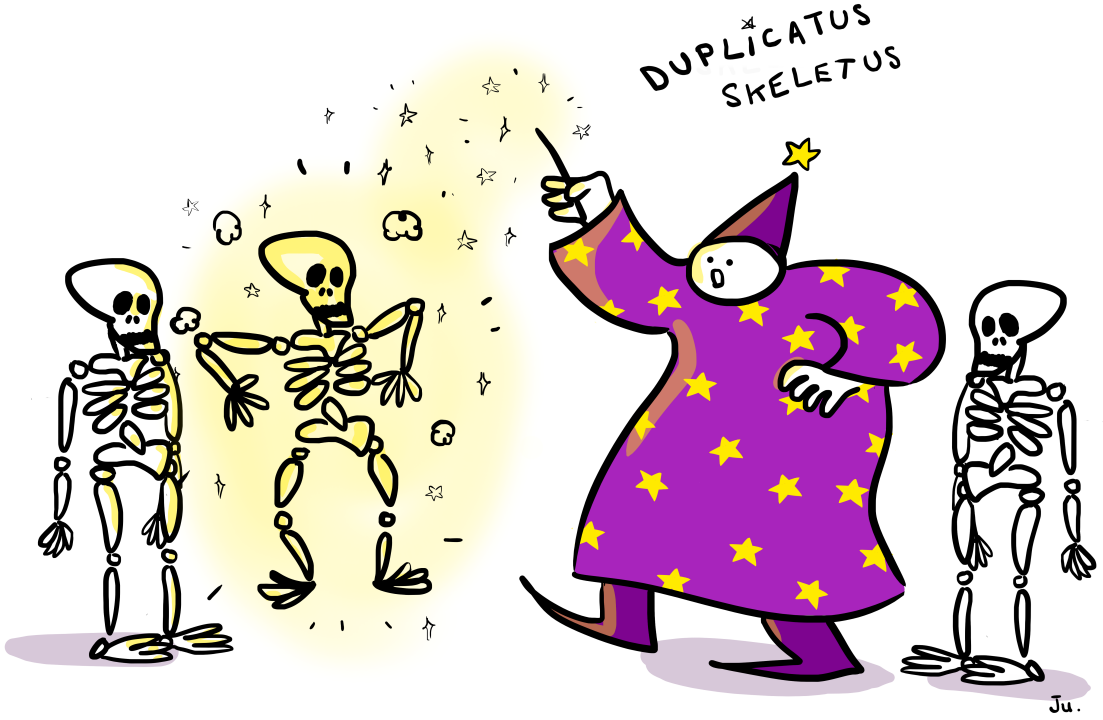
The atomic λ-calculus and the λ-calculus with generalized applications are two (independent) variants of the λ-calculus originating in computational interpretations of proof theory. While programming languages are built from specific deterministic evaluation strategies, the existing literature on the atomic λ-calculus and generalized applications only go over the general theory of the calculi. In particular, reduction of terms is unrestricted, and only analyzed qualitatively. This induces a gap between theory and practice, which we strive to diminish in this thesis.

Starting from the atomic λ-calculus, we isolate the most salient concept of its operational semantics, that we call node replication. This is a particular substitution procedure, which duplicates terms finely, one node of the syntax tree at a time. We follow up with λ-calculi with generalized applications. They use a ternary application constructor, adding a continuation to the usual binary application. In this work, we develop the operational theories built on node replication and generalized applications separately. For the two of them: On an operational level, we give several evaluation strategies, all observationally equivalent to the corresponding strategies in the λ-calculus. On a logical level, our approach is guided by quantitative types. We provide different type systems that inductively characterize semantic properties, but also give quantitative bounds on the length of reduction and the size of normal forms.

More precisely, in the first part of this thesis, we implement node replication by means of an explicit substitution calculus. We show in particular how node replication can be used to implement full laziness, a well-known evaluation strategy for functional programming languages like Haskell. We prove observational equivalence properties relating the fully lazy semantics with the standard ones. In the second part of the thesis, we start with an operational and logical characterization of solvability in λ-calculi with generalized applications. We show how this framework gives rise to a remarkable operational theory of call-by-value. The call-by-name characterization relies on an original calculus with generalized applications. We show in both cases that the operational semantics are compatible with a quantitative model, unlike the one of the original call-by-name calculus. We then prove essential properties of this new call-by-name calculus, and show observational equivalence with the original one.

Keywords: λ-calculus, rewriting, node replication, generalized applications, intersection types, explicit substitutions.

À Arthur.
Faute de pouvoir lire la tienne, cette thèse t'es dédiée.


DUPLICATUS
SKELETUS
Ju.

Remerciements

J'ai toujours pensé que ce chapitre serait le premier que j'écrirais de ma thèse. Pour le plaisir de faire une petite place dans ma bibliothèque aux gens qui m'ont connu pendant ces années. Pour qu'ils s'amusent aussi à voir si leur nom y figure. Mais sans surprise, c'est quelques heures seulement avant d'envoyer mon document pour impression que je me retrouve à écrire ces lignes. Les oublié·e·s me pardonneront donc.

Comment commencer sans remercier ma directrice de thèse ? Delia Kesner a su à la fois être une directrice disponible, compréhensive, patiente et pédagogue. Mais elle fut aussi une collaboratrice : le contenu de cette thèse a été travaillé à au moins quatre, au plus six mains, et ce fut un très grand plaisir d'avancer ensemble sur ces problèmes.

Thank you Silvia Ghilezan and Willem Heijltjes for accepting to review my thesis, for taking time to do so, and for your comments. Giulio Guerrieri, Damiano Mazza and Femke van Raamsdonk, thank you for accepting to be examiners. I must say that I am very proud and honored to have such a jury, and happy that everyone of you could make it to Paris.

Daniel Ventura and José Espírito Santo, I am glad that I had the opportunity to work with you. It is unfortunate that I had to cancel my plans to go to Goiânia. José, thank you again for your warm welcome in Braga, and for coming over to Paris to attend my defense.

Merci à Yann Régis-Gianas pour avoir encadré mon stage d'initiation à la recherche à la fin de mon M1, puis mon contrat à OCamlPro. Merci à Beppe Castagna de m'accueillir dans l'après-thèse, de l'autre côté du couloir.

Je remercie le personnel administratif de l'IRIF, sans lesquel·le·s toute cette recherche ne serait pas possible. En particulier Omur Avci et Dieneba Camara, avec qui j'ai été le plus en contact.

Le MPRI a été intensif, mais on y a survécu tou·te·s ensemble. Pour cela, merci à Aliaume, Antonin, Boris, Carine, Émilie, Gaëtan, Kostia, Pierre, Paul, Paul, Olivier, Yuan. Merci à tou·te·s les doctorant·e·s de l'IRIF, aux côtés desquel·le·s j'ai eu le plaisir de travailler (par intermittence forcée) tel·les que : Abishek, Alen, Antoine, Bernardo, Cédric, Chaitanya, Clément, Colin, El-Mehdi, Farzad, Félix, Jules, Hadrien, Hugo, Klara, Léo, Léonard, Maiana, Mickael, Moana, Nicolas, Pierre, Raphaëlle, Rémi, Simona, Théo, Victor, Vincent, Zeinab. Une pensée particulière à Victor, pour les discussions sans fin, et Adrienne, dont je suivrais les avancées avec attention.

Durant ces années un peu spéciales, les rencontres en dehors de l'université ont été assez rares. Celles que j'ai faites m'ont pourtant marqué. Merci aux titulaires et non titulaires qui ont permis d'élargir mon horizon sur la fin de la thèse, que ce soit à Fontainebleu, Nantes, Munich ou Haifa. Je veux nommer en particulier Giulio Guerrieri (encore), Romain Péchoux, Ambroise Lafont, Davide et Thiago.

Pour toutes nos victoires et pour toutes nos tentatives, merci aux camarades d'Amiens. À

~~Paris 7~~ l'université ~~de Paris~~ Paris Cité, je pense en particulier à Aemon, Bertille, Brice, Camille, Élise, Émilien, Juliette, Laurie, Lucas, Louane, Lorenzo, Margaux[1], Marion, Nour, Rémi[2], Zak et Zoé. Au sein de la fédération : Charlie, Étienne, Julien, Lucie, Myriam, Quentin, Saphire, Solène, Thomas, Tristan, Val, Victor, Yvain et les autres. Bon courage à la relève !

Merci à la Perra. À celleux que je connais depuis Jean Jaurès, que j'ai connu avant, ou qui sont venu·e·s ensuite. Merci de me rattacher à Villeurbanne, et d'être le meilleur groupe de potes qu'on puisse avoir. Merci à Adrien, Albé, Bastos, Céline, Clément, Clément, Coralie, Jenny, Johan, Leïla, Louise, Louison, Marianne, Marie, Nina, Philippe, Pierre, Ponthus, Tiago, Vincent, Yannick. Et à tou·te·s celles et ceux qui tournent autour.

La dernière série de remerciements va à la famille. Merci papa et maman. C'est bien sûr grâce à vous que j'en suis là. Merci Johann et Thomas, Mamie et Dédé. Merci à Éric et Isabelle pour m'avoir accueilli en mars 2020, et ensuite. Merci à la famille Courson en Sarthe et Conquérant à Saint-Jouan pour leur gentillesse. Merci aux chats pour leurs ronrons.

Et pour finir, merci Juliette [Abe+20]. Pour tout ce qu'on a vécu et vivra. Pour tout ce que tu m'apportes et pour ton soutien. Pour tout[3].

[1] Qui me précède et me succède.

[2] Et Hannah Arendt !

[3] Sans oublier le dessin qui précède ces remerciements.

Contents

CHAPITRE 0

Résumé en français

Les langages de programmation fonctionnels sont des langages avec un haut niveau d'abstraction. Ces langages sont dits descriptifs, c'est-à-dire qu'un programme fonctionnel décrira plutôt quel est le résultat que le ou la programmeuse veut obtenir, plutôt que comment l'ordinateur doit traiter les données pour obtenir le résultat souhaité.

Ce haut niveau d'abstraction a de nombreux avantages : les programmes sont succincts, plus facile à déboguer, portables entre différentes machines. Les langages fonctionnels offrent généralement de puissants mécanismes comme les fonctions d'ordre supérieur, le filtrage par motifs ou des structures de données spécifiques.

Bien sûr, un haut niveau d'abstraction a un désavantage immédiat : plus le code s'éloigne des instructions interne de la machine, plus il devient difficile de faire le lien entre les deux, par la compilation ou l'interprétation directe du programme. Pour savoir quoi faire d'un morceau de code écrit, il convient de lui donner un sens : une *sémantique*.

La recherche en sémantique des langages de programmation cherche à donner un sens aux programmes. Un des buts principaux est de savoir quelles transformations peuvent être appliquées aux programmes sans en modifier le sens. Ces transformations peuvent notamment correspondre à une compilation, une interprétation ou une optimisation.

En ce qui nous concerne, au lieu de s'intéresser à un ou des langages en particulier, nous allons adopter des modèles théoriques (mathématiques) du calcul et des langages. En effet, les modèles abstraits représentent une grande classe de langages et programmes simultanément. Par conséquent, une propriété dérivée pour un modèle sera vraie pour toute une classe de langages. Par ailleurs, les modèles abstraient aussi de nombreuses difficultés liées à des considérations plus pratiques, ce qui permet de concentrer l'attention sur des problèmes spécifiques. Enfin, ces modèles mathématiques nous donnent accès à de nombreux outils de raisonnement.

Deux types de sémantiques sous-tendent notre travail. La première est la *sémantique opérationnelle*. Celle-ci est centrée sur la syntaxe, et définit le sens d'un programme comme la façon dont les (sous-)expressions interagissent. Un programme est transformé par une séquence d'étapes de *réécriture*, qui permettent d'aboutir à un résultat nommé forme normale, atteint quand le programme ne peut plus être évalué par réduction. La forme du code est importante, puisqu'elle détermine les transformations ultérieures.

La seconde est la sémantique dénotationnelle. Celle-ci est axée sur des propriétés globales de programmes qui sont invariantes au cours de l'évaluation. De telles propriétés sont la terminaison (un programme s'arrête-t-il ?), ou l'équivalence observationnelle de deux programmes (ces programmes ont-ils le même comportement dans tous les contextes ?). La sémantique dénotationnelle peut aussi être utile pour vérifier qu'une sémantique opération-

nelle conserve bien le sens des programmes.

Dans cette thèse, notre outil principal sera le *λ-calcul*, et surtout des dérivés pour lesquels nous étudions et définissons des sémantiques opérationnelles. Des sémantiques dénotationnelles des programmes seront données sous la forme de systèmes de *types intersection non idempotents*.

Le λ-calcul et ses variants. Le λ-calcul, créé à la fin des années 1920 par Church [Chu32] est un modèle de calcul qui peut être vu à la fois comme le premier langage de programmation fonctionnel, et comme le noyau de tout langage fonctionnel. Nous l'utilisons comme modèle mathématique de ces langages. Le λ-calcul est lui-même un langage très élémentaire, dont les programmes, appelés *termes*, sont construits à l'aide de trois constructeurs seulement :

- des variables $x, y, z, \dots$,
- des abstractions $\lambda x.t$ entendues comme la fonction $x \mapsto t$ où le paramètre x peut apparaître dans le sous-terme t ou pas,
- des applications tu d'un terme t à un argument u.

Le λ-calcul possède en plus de la grammaire ci-dessus un aspect dynamique. Une seule règle de réduction est nécessaire pour l'évaluation d'un λ-terme, la règle β. Le terme à gauche de la flèche est appelé un *radical*.

$$(\lambda x.t)u \longrightarrow_\beta t\{x/u\}$$

L'opération de *substitution* $t\{x/u\}$ est définie comme le remplacement de toutes les occurrences de x dans t par le terme u. La règle β réduit donc une fonction de paramètre x appliquée à un argument u en cette même fonction instanciée par le paramètre u. Il est important de noter ici que la substitution est définie de manière *externe*, et pas directement par des règles données dans le calcul.

Il existe de nombreuses extensions et variations du λ-calcul. Celles-ci ont de nombreux buts, par exemple représenter explicitement certains comportements du calcul ou certaines structures, ou rapprocher le λ-calcul d'autres théories mathématiques. Dans notre travail, les calculs à *substitutions explicites* sont récurrents. Ces calculs intègrent l'opération de substitution afin d'offrir une plus grand maîtrise dessus qu'en λ-calcul simple.

Plus concrètement, les calculs à substitutions explicites possèdent un constructeur en plus : une substitution explicite $t[x/u]$, qui représente une substitution retardée. Ainsi, la réduction β est décomposée :

1. Une première étape $\longrightarrow_{\mathrm{dB}}$ réduit un radical en introduisant une substitution explicite : $(\lambda x.t)u \longrightarrow_{\mathrm{dB}} t[x/u]$. Dans le terme contracté, les occurrences de x dans t pointent toutes vers le sous-terme u, qui est dit *partagé*.
2. Une deuxième série d'étapes peut manipuler la substitution de la manière souhaitée.

Dans la première partie de cette thèse (chapitre 2), nous voyons comment implémenter ce que nous appelons « réplication par nœuds » à l'aide d'un nouveau calcul à substitutions explicites nommé λR, qui est inspiré du *λ-calcul atomique* de Gundersen, Heijltjes et Parigot [GHP13b].

Un calcul pour la réplication par nœuds. La réplication par nœuds est une décomposition de la substitution, où les termes sont substitués constructeur par constructeur, ou *nœud par nœud* en voyant les termes comme des arbres de syntaxe.

La réplication par nœuds est un outil d'optimisation important : elle apparaît dans la réduction optimale par les graphes de partage [Lam90], et, comme montré par Gundersen, Heijltjes et Parigot [GHP13b], permet d'implémenter la *pleine paresse*, qui évite de nombreuse duplications de calculs lors de la substitution.

Notre calcul est une réinterprétation du λ-calcul atomique qui est lui-même une interprétation calculatoire d'un système logique de la déduction ouverte [GGP10]. Gundersen, Heijltjes et Parigot [GHP13b] s'attachent à garder une correspondance directe entre la déduction ouverte et le λ-calcul atomique. Ainsi, en plus de la réplication par nœuds, les variables de leur calcul sont linéaires. Par exemple, le terme correspondant à $\lambda x.xx$ dans la grammaire du λ-calcul atomique est $\lambda x.(x_1x_2)[x_1, x_2 \longleftarrow x]$. Il y a donc dans ce calcul une notion de partage ressemblant aux substitutions explicites.

Dans notre travail, nous ne conservons que la réplication par nœuds. La suppression de la contrainte de linéarité nous permet de formuler la sémantique de la réplication par nœuds en termes de substitutions explicites. Nous obtenons une formulation originale et concise de la réplication par nœuds, qui est suffisamment simple pour modéliser différents langages de programmation. L'étude des propriétés de la réplication par nœuds est mise en avant et facilitée par ce nouveau calcul.

Applications généralisées. Dans la deuxième partie de la thèse (chapitres 3 et 4), nous nous intéressons à un second variant du λ-calcul : le λ-calcul à applications généralisées, introduit par Joachimski et Matthes [JM00], puis Espírito Santo [Esp20] dans une version en appel-par-valeur (voir plus loin).

La différence syntaxique avec le λ-calcul est le constructeur d'application. L'application binaire tu devient $t(u, x.r)$. Dans le sous-terme supplémentaire r, des occurrences de la variable x peuvent apparaître. L'application tu est donc partagée par toutes les occurrences de x. Il y a une notion de partage, telle que toutes les applications sont partagées, mais seulement elles. C'est là une différence avec les calculs à substitutions explicites où tous les types de constructeurs sont partageables.

La dynamique des calculs à applications généralisées est donnée par une règle β utilisant une opération de substitution externe, ainsi qu'une règle de permutation π permettant de convertir les termes à une *forme normale entière* (*full normal form*). De par ces particularités, ces calculs présentent une sémantique opérationnelle très intéressante.

La correspondance de Curry-Howard. La réplication par nœud (au sein du λ-calcul atomique) ainsi que les calculs à applications généralisées ont tous deux une origine commune.

Ces deux formalismes sont tous les deux le fruit d'une interprétation de systèmes de la *théorie de la démonstration* à travers la *correspondance de Curry-Howard*.

La théorie de la démonstration [Gir00] est une branche de la logique mathématique et une métamathématique dont les objets sont des représentations formelles du raisonnement, appelées preuves ou démonstrations. La correspondance de Curry-Howard [How80] met en relation les systèmes de cette théorie avec les systèmes de la théorie des langages de programmation. Par exemple le λ-calcul avec (le fragment implicatif de) la logique intuitionniste en déduction naturelle de [Gen35a ; Gen35b]. C'est à la fois une correspondance statique : les types correspondent aux formules logiques, les programmes typés aux preuves de ces formules. Mais aussi une correspondance dynamique : la réduction d'un programme correspond à la normalisation d'une preuve. Cette correspondance révèle donc le caractère *calculatoire* de la théorie de la démonstration.

Le lien entre calculs et démonstrations est fondamental. Par conséquent, des résultats ou des idées dans un domaine peuvent influencer des avancées dans l'autre. C'est ainsi que des avancées en théorie de la démonstration (inférence profonde [Gug15] et déduction ouverte, déduction naturelle avec règles d'élimination généralisées [vPla01]) ont mené à l'introduction des formalismes que nous considérons.

Cependant, ces calculs ont été étudiés de manière générale, du point de vue de la théorie de la preuve. Dans la littérature, les résultats portent presque exclusivement sur la normalisation forte, qui correspond à une évaluation non restreinte et non déterministe, au contraire de l'évaluation dans les langages de programmation (voir le paragraphe suivant). De même, des propriétés sémantiques importantes, comme la *résolubilité*, sont capturées par des notions de réduction plus fines absentes de la littérature. D'autres travaux, en particulier sur les calculs à applications généralisées, les considèrent comme un outil pour la théorie de la preuve [Esp09 ; EFP18].

Nous suivons une approche orienté vers les langages de programmation et étendons l'étude des sémantiques opérationnelles des formalismes de réplication par nœuds et applications généralisées, en nous intéressant à diverses notions de réduction et de normalisation. Dans cette même perspective, nous menons une analyse *quantitative* de la réduction, c'est-à-dire sensible au nombre d'étapes de réduction ou à la taille des formes normales, par l'intermédiaire des types intersection non idempotents.

Différentes notions de normalisation. La grande majorité des calculs existant sont non déterministes : un unique terme est souvent réductible de différentes manières. C'est le cas pour le λ-calcul atomique, et pour les calculs à applications généralisées. Ce non déterminisme n'est pas trivial, il arrive qu'un chemin de réduction ne termine jamais, alors qu'un résultat était à portée de main en empruntant un autre chemin. Heureusement, les calculs cités sont confluents : tout chemin de réduction partant d'un terme et qui termine arrive toujours sur un unique résultat. Il convient dès lors d'étudier avec attention la question de la réduction : quel chemin prendre pour être sûr d'arriver au résultat, ou même pour minimiser le nombre d'étape.

La construction d'une évaluation déterministe dans le λ-calcul se fait en trois étapes.

1. Choisir une modalité de passage des paramètres (appel par nom, valeur ou nécessité).

2. Définir le type de résultats souhaités et restreindre l'évaluation en conséquence.

3. Rendre la réduction déterministe en donnant un ordre sur les radicaux.

Le troisième point étant plus étroitement lié à la syntaxe du calcul, nous nous concentrons sur les deux premiers points dans ce résumé, que nous détaillons maintenant un peu.

La question du passage de paramètres est essentielle à la fois du point de vue du nombre d'étapes de réduction et de la normalisation. Le λ-calcul de Church est dit en *appel-par-nom* : l'opération de β-réduction $(\lambda x.t)u$ s'applique sur n'importe quel terme de cette forme, quelle que soit la forme de u. Ceci est généralement source d'inefficacité : le terme u peut lui même être un radical. Si la variable x intervient plusieurs fois dans t, le calcul à effectuer pour réduire u sera dupliqué en même temps que u.

Le λ-calcul en *appel-par-valeur* de Plotkin [Plo75] est un calcul différent qui ajoute la restriction que le terme u doit être une valeur, c'est-à-dire une variable ou une abstraction. Il faudra donc réduire le terme u d'abord si nécessaire, ce qui évite de dupliquer le travail à effectuer. L'appel-par-valeur est à la base de nombreux langages fonctionnels comme OCaml ou Lisp.

Ce calcul a un inconvénient par rapport à l'appel-par-nom : il y a des termes qui normalisent en appel-par-nom mais pas en appel-par-valeur. Cela arrive si la réduction de u ne termine jamais, alors même que x n'apparaît pas dans t. En appel-par-nom, u sera simplement effacé par l'opération de substitution. L'appel-par-valeur n'a donc pas le même comportement que l'appel-par-nom, même à un niveau sémantique. Cette différence est très importante et reviendra dans la partie de la thèse sur les applications généralisées où des calculs en appel-par-nom et des calculs en appel-par-valeur seront considérés.

L'*appel-par-nécessité* [CF12] prend le meilleur des deux : l'argument u du radical $(\lambda x.t)u$ est seulement réduit s'il est nécessaire dans le corps t de la fonction $\lambda x.t$, mais la réduction de u n'est effectuée qu'une seule fois, quel que soit le nombre d'occurrences de x dans t. L'appel-par-nécessité est généralement implémenté avec une substitution *linéaire* : seule une occurrence de x est remplacée à la fois. L'inconvénient de l'appel-par-nécessité est qu'il est plus difficile à implémenter, nécessitant des systèmes de partages de termes comme les substitutions explicites.

Cependant, même cette méthode peut être source de duplication de calculs. C'est le cas quand une abstraction est dupliquée mais que son corps contient des radicaux. Une optimisation nommée *pleine paresse* [Wad71] permet d'éviter certaines de ces duplications. L'appel-par-nécessité pleinement paresseux a été prouvé *optimal* pour l'évaluation faible confluente (voir la thèse de Balabonski [Bal12b]). L'appel-par-nécessité pleinement paresseux est à la base du langage Haskell. Néanmoins, la pleine paresse est habituellement implémentée par des opérations externes au calcul. Nous donnons dans le chapitre 2 une implémentation de l'appel-par-nécessité pleinement paresseux au sein d'un calcul à substitution explicites utilisant la réplication par nœuds.

Une fois une modalité de passage de paramètres choisie, nous pouvons nous interroger sur le type de résultat voulu, qui n'est pas universel. On peut considérer comme résultat les formes normales qui ne peuvent plus réduire. Ce n'est pas l'approche prise par les langages de programmation généralistes : dans ceux-ci, un terme qui peut seulement être réduit sous

les abstractions (c'est-à-dire dans le corps des fonctions) est un résultat (on parle de forme normale *faible*). De même l'évaluation peut être restreinte à la *tête* du terme, *grosso modo* le radical le plus à gauche. Ces différentes notions d'évaluation capturent différentes propriétés sémantiques : la réduction de tête par exemple capture la *résolubilité* en appel-par-nom, et la réduction faible la *valuation potentielle* en appel-par-valeur. De par leur simplicité, les λ-calculs sont un outil de choix pour étudier les différentes réductions possibles.

Types intersection. Pour capturer ces différentes notions de normalisation et les propriétés sémantiques associées, nous utilisons des systèmes de types intersection.

De manière générale, les types servent de garantie pour les programmes : un programme typé respecte certaines propriétés. Les types simples, qui sont le standard pour le λ-calcul, garantissent la terminaison : toutes les réductions à partir d'un terme simplement typé terminent.

Cependant, il existe des termes qui sont normalisables, voire en forme normale, mais ne sont pas simplement typables. C'est le cas par exemple du terme $\lambda x.xx$ où il faudrait donner deux types différents à la variable x. Dans les systèmes de types intersection [CD80], il est possible de donner plusieurs types à la même variable ou au même terme sous forme d'une intersection de types. Le terme $\lambda x.xx$ est donc typable. Plus généralement, exactement tous les termes normalisables sont typables. Autrement dit, les systèmes de types intersection capturent la normalisation. La propriété « t est normalisable » peut être exprimée de manière équivalente par « t est typable », sans avoir à réduire le terme t pour le vérifier.

Cette caractérisation est très utile pour prouver des propriétés sur la normalisation. Nous nous en servons pour valider nos stratégies d'évaluation en vérifiant qu'elles correspondent bien à la notion de normalisation souhaitée. Nous pouvons facilement comparer la normalisation de différentes stratégies ou calculs, pour lier nos formalismes aux formalismes existants.

Nous utilisons des systèmes de types intersection qui sont *non idempotents* [Gar94], aussi connus comme types *quantitatifs*. Les types intersection en général, que l'intersection soit idempotente ou non, donnent un modèle qualitatif du calcul, répondant à des questions comme : Ce terme normalise-t-il ? Deux termes sont-ils observationnellement équivalents ?

Les types intersection non idempotents raffinent cette analyse en une analyse quantitative, par exemple : Combien d'étapes de réduction faut-il pour réduire ce terme à une forme normale ? La réduction de ces deux termes est-elle de même longueur ? Ce type d'analyse est particulièrement intéressante quand on s'intéresse aux λ-calculs comme fondation des langages de programmation car c'est une première étape vers des analyses de complexité. Les types quantitatifs permettent aussi des preuves de normalisation des termes typés très simples car combinatoires, où la taille des dérivations de type décroît à chaque étape de réduction.

Contributions

Nous énonçons la problématique de cette thèse comme suit :

> Quelles contributions la réplication par nœuds et les applications généralisées, analysées quantitativement, apportent-elles à la théorie des langages de programmation ?

Plus précisément, nos contributions sont doubles. D'une part, nous donnons des sémantiques opérationnelles détaillées de calculs avec réplication par nœuds ou applications généralisées. Cela consiste notamment en la définition de relations de réduction correspondant à différentes notions d'évaluation et de normalisation, pertinentes pour la sémantique des langages de programmation. D'autre part, nous assignons des systèmes de types quantitatifs à ces relations. Nous les utilisons comme une inspiration pour raffiner les calculs, comme outil technique pour simplifier les preuves de normalisation et comme outil sémantique pour prouver l'équivalence de propriétés sémantiques entre différents calculs.

L'une des manières par laquelle le modèle quantitatif influence la définition des λ-calculs est l'utilisation de la *distance* [AK10 ; ABM14], afin de mettre l'accent sur la computation. Dans les calculs à substitutions explicites, ou à applications généralisées, des règles de permutation sont nécessaires pour *débloquer* certains radicaux. Prenons par exemple le terme $(\lambda x.t)[y/r]u$. Le terme u est l'argument de l'abstraction $\lambda x.t$, mais on n'a pas encore un radical, puisqu'une substitution explicite sépare $\lambda x.t$ et u. Nous pouvons cependant permuter la substitution explicite et faire émerger le radical :

$$(\lambda x.t)[y/r]u \to_{\sigma_1} ((\lambda x.t)u)[y/r] \to_{\mathrm{B}} t[x/u][y/r]$$

Les deux termes à gauche et à droite de la permutation σ_1 sont sémantiquement équivalent. Ils ont la même représentation dans de nombreuse représentations graphiques des termes. Dans celles-ci, l'étape de calcul serait exécutée en une unique étape.

La réduction à distance s'inspire de formalismes graphiques et rassemble les règles de calcul et les règles de permutation dans une unique étape de réduction. De cette façon, la sémantique opérationnelle des représentations séquentielles de termes se rapproche de celle des représentations graphiques, avec souvent une correspondance étape-par-étape [KL07 ; Acc18b ; Kes22].

Le choix de la distance reflète également mieux les modèles logiques : les types quantitatifs sont principalement insensibles aux règles de permutation. En effet, ces dernières ne sont pertinentes que d'un point de vue structurel, tandis que la quantitativité est uniquement liée aux règles de calcul. Avec la distance, chaque étape représente une étape de calcul significative.

La notion de distance introduite, nous pouvons donner précisément les calculs étudiés et définis dans cette thèse :

- Dans la première partie, un calcul original à substitutions explicites implémentant la *réplication par nœuds* et utilisant une sémantique à *distance*.

- Dans la deuxième partie, les calculs à applications généralisées en appel-par-nom et appel-par-valeur ainsi que des variantes originales utilisant une sémantique à *distance*.

Réplication par nœuds

L'objectif principal de la première partie de la thèse (chapitre 2) est d'introduire la théorie et la pratique de la réplication par nœuds, dans le cadre du λ-calcul. Nous utilisons un nouveau calcul à substitutions explicites λR, que nous introduisons en section 2.1. Ce calcul est une réinterprétation du λ-calcul atomique et utilise la distance pour mettre en évidence les mécanismes de la réplication par nœuds.

La perspective change par rapport au λ-calcul atomique. Alors que Gundersen, Heijltjes et Parigot [GHP13b] donnent une interprétation calculatoire de la déduction ouverte, nous voulons donner une analyse fine de la substitution dans le λ-calcul et les langages de programmation en général en ajoutant la possibilité de substitution nœud par nœud.

Nous donnons quelques propriétés générales du calcul λR dans la section 2.2 : terminaison de la procédure de substitution avec réplication par nœuds, confluence et simulations avec le λ-calcul.

Le calcul est ensuite affiné en deux stratégies d'évaluation déterministes. La première en appel-par-nom ne prend pas avantage des optimisations apportées par la réplication par nœuds (section 2.3.1). Cette relation de réduction simule la réduction de tête faible du λ-calcul. Elle sert de lien entre le λ-calcul et des stratégies plus élaborées utilisant la réplication de nœuds.

La deuxième stratégie implémente l'appel-par-nécessité pleinement paresseux faible (section 2.3.2). Plusieurs implémentations de la pleine paresse existent dans la littérature (voir section 2.6), à commencer par la première par Wadsworth [Wad71]. Mais dans celles-ci, le point crucial de la séparation du squelette et des expressions libres maximales est un calcul externe. Au contraire, [GHP13b] montre comment une extraction entièrement paresseuse peut être effectuée dans le λ-calcul atomique. Nous nous basons sur ces résultats et intégrons l'extraction du squelette dans une stratégie d'appel-par-nécessité pour construire une stratégie pleinement paresseuse. Dans cette stratégie, les étapes menant à l'extraction sont décrites au sein du calcul. Par conséquent, l'opération est autonome et décrite de manière totalement opérationnelle.

Nous donnons deux types de sémantique pour la séparation du squelette des expressions libres. La première est une sémantique à *grands pas* [Kah87] et reformule la preuve de [GHP13b] que l'extraction du squelette peut être implémentée par le λ-calcul atomique. La seconde est une sémantique à *petits pas* qui détaille comment extraire un squelette pas à pas en utilisant les règles du calcul λR. Nous montrons que ces deux sémantiques correspondent à deux définitions différentes mais équivalentes d'un squelette.

Ariola et Felleisen [AF97] ont démontré que l'appel-par-nom et l'appel-par-nécessité, utilisant des substitutions respectivement complètes et linéaires, sont équivalents du point de vue de l'observation. La même propriété s'applique-t-elle dans notre cas avec la réplication par nœuds ? Une de nos contributions est une preuve de résultat utilisant un système de type quantitatif, en section 2.5. Cette technique de preuve [Kes16] simplifie considérablement d'autres approches basées sur des outils syntaxiques [AF97 ; MOW98]. En outre, l'utilisation de types intersection a une autre conséquence importante : les appel-par-nom et appel-par-nécessité usuels s'avèrent être équivalents, du point de vue de l'observation, aux appel-par-nom et appel-par-nécessité avec réplication par nœud, ainsi qu'à la notion plus

sémantique de *needeedness* [KRV18]. Il s'agit à notre connaissance de la première caractérisation quantitative de la normalisation pleinement paresseuse.

Applications généralisée

Qu'apportent les applications généralisées à la théorie des langages de programmation ? Nous affirmons qu'elles offrent un niveau d'abstraction différent des formalismes existants. Elles sont caractérisées par deux éléments : une notion de partage restreinte aux applications, et une gestion interne simple de la recherche d'un radical.

Le partage est autorisé par le constructeur d'application généralisé $t(u, y.r)$, où le terme tu est partagé par les occurrences de y dans r. Ce partage est utile pour éviter de dupliquer certains calculs. Puisque les β-radicaux sont des applications, ils sont tous partagés par défaut. Cependant, le partage n'est pas aussi général que dans les calculs avec constructeurs `let`, où chaque type de terme peut être partagé, et comme celui des calculs avec substitutions explicites, qui possèdent également un traitement interne de la substitution. Les applications généralisées maintiennent la substitution à un niveau externe. En conséquence, le calcul est toujours effectué en une seule étape (une étape β généralisée présentée ci-dessous), plutôt qu'en deux phases, comme avec les substitutions explicites. La sémantique opérationnelle du calcul est donc plus simple :

$$(\lambda x.t)(u, y.r) \longrightarrow_\beta r\{y/t\{x/u\}\}$$

Le partage des applications est particulièrement utile pour l'appel-par-valeur. Contrairement à la plupart des calculs en appel-par-valeur [AG16], le calcul ΛJ_v (ou notre nouvelle version à distance) n'impose aucune restriction sur les radicaux. Cela signifie que chaque application d'une fonction à un argument est un radical qui peut être déclenché. Des inconvénients des formalismes en appel-par-valeur sont ainsi évités. Plus encore : la réduction en appel-par-valeur est effectué au moyen d'une règle presque identique à l'appel-par-nom, en s'appuyant uniquement sur une notion différente de substitution (définie dans la section 3.1.1) :

$$(\lambda x.t)(u, y.r) \longrightarrow_\beta r\{y\backslash\backslash t\{x\backslash\backslash u\}\}$$

Avoir les mêmes radicaux $(\lambda x.t)(u, y.r)$ en appel-par-nom et appel-par-valeur signifie également que pour toute notion de normalisation, définir une évaluation d'appel-par-valeur est simple. La définition des formes normale est la même, et pour de nombreuses stratégies intéressantes, les même règles de réduction inductives peuvent être choisies. C'est le cas par exemple des formes normales fortes, définies dans les sections 3.7 et 4.2, et de la stratégie normalisante « leftmost-outermost ».

La recherche d'un radical dans le calcul est assurée par la règle de permutation nommée π, qui est l'une des permutations cachées révélées par von Plato [vPla01] :

$$t(u, x.r)(u', y.r') \longrightarrow_\pi t(u, x.r(u', y.r'))$$

Concrètement, cette permutation déplace le radical le plus à gauche sur le dessus, comme dans l'exemple suivant.

Example 0.1. La réduction suivante est représentée dans la figure 0.1 sous forme graphique, de manière à faire apparaître la façon dont le radical à gauche remonte en haut de l'arbre.

$$\begin{aligned}(\lambda x.t)(u_1, y_1.y_1)(u_2, y_2.y_2)(u_3, y_3.y_3) &\to_\pi (\lambda x.t)(u_1, y_1.y_1)(u_2, y_2.y_2(u_3, y_3.y_3)) \\ &\to_\pi (\lambda x.t)(u_1, y_1.y_1(u_2, y_2.y_2(u_3, y_3.y_3)))\end{aligned}$$

Dans le cadre fermé (sans variables libres) et faible de tête, qui est celui des langages de programmation généralistes, la permutation π permet au calcul d'atteindre une forme normale sans rentrer dans le terme.

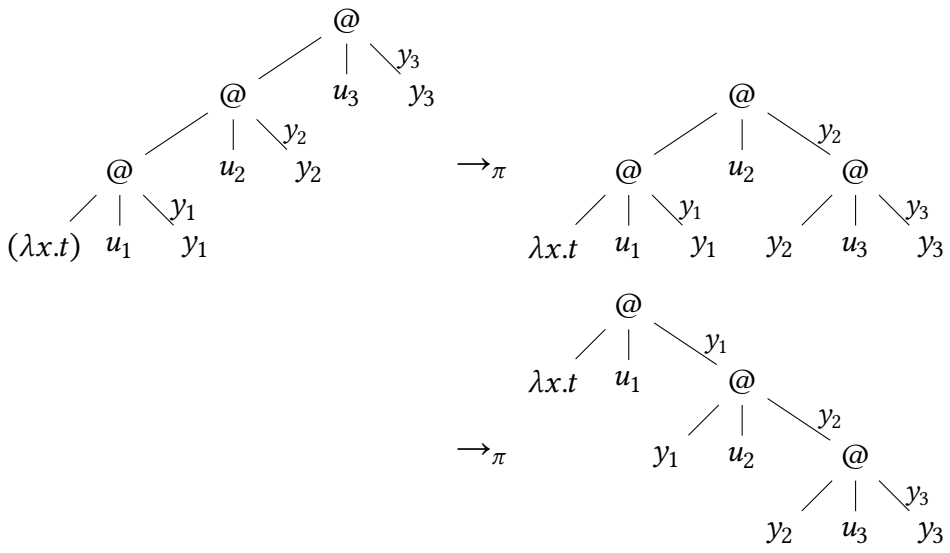


Fig. 0.1 : La permutation π illustrée sur un arbre de syntaxe.

Ce dernier exemple est une traduction d'un λ-terme $(\lambda x.t')u'_1u'_2u'_3$, dans lequel t' est traduit en t, et u'_i en u_i pour $1 \leq i \leq 3$. Dans une *machine abstraite* comme celle de Krivine [Kri07], les termes de u_1 à u_3 seraient successivement déplacés dans la *pile*. Les applications généralisées fournissent une représentation de la pile directement à l'intérieur du terme, et l'étape de réduction de la machine abstraite déplaçant l'élément de droite des applications à l'intérieur de celui-ci est remplacé par une permutation π. L'idée est similaire avec le style par passage de continuations (CPS) et les formes normales administratives (ANF) qui donnent un nom à chaque calcul intermédiaire pour encoder la pile.

L'utilisation de la distance permet de gagner en abstraction. En intégrant la permutation dans la règle β, il n'y a plus d'étape explicite révélant le radical le plus à gauche, mais seulement une règle de calcul. Ainsi, le calcul avec des applications généralisées se rapproche du λ-calcul, la seule différence étant que les applications sont nommées et partagées. Les applications généralisées avec distance peuvent alors aussi être considérées comme des versions plus abstraites et plus simples des calculs avec partage. Dans notre travail, nous donnons la priorité aux variations à distance des calculs à applications généralisées en appel-par-valeur

et appel-par-nom, pour rester aussi proche que possible du λ-calcul et du modèle donné par les types quantitatifs.

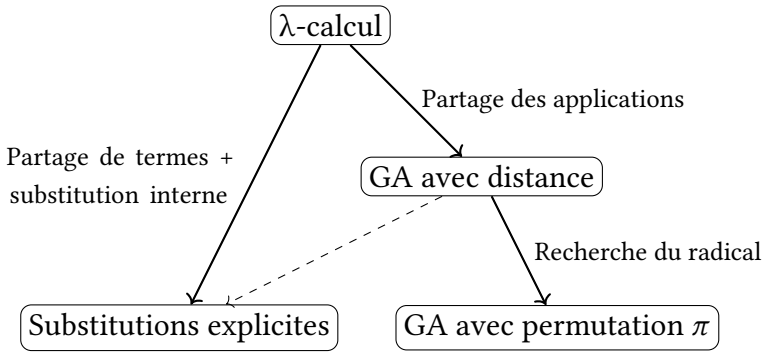


Fig. 0.2 : Plusieurs niveaux d'abstraction.

Malgré les aspects pratiques des applications généralisées, la littérature manque d'études détaillées de leur sémantique opérationnelle. Les travaux d'Espírito Santo [Esp09] ainsi que Espírito Santo, Frade et Pinto [EFP18] considèrent les applications généralisées comme un outil pour la théorie de la démonstration. Les travaux de Joachimski et Matthes [JM00 ; JM03] sur ΛJ, et d'Espírito Santo [Esp20] sur ΛJ_v introduisent le calcul, montrent la normalisation forte du calcul typé, ainsi que la confluence et la *standardisation* dans le premier cas. Cette approche centrée sur la normalisation forte est à nouveau orientée du point de vue de la théorie de la démonstration.

Nous adoptons une approche différente, inspirée de la sémantique des langages de programmation. Nous examinons la *résolubilité* pour les calculs à applications généralisées en appel-par-nom et appel-par-valeur, d'abord pour les versions distantes λJ_n et λJ_v, puis en transposant les résultats aux versions originales ΛJ et ΛJ_v. La résolubilité est une notion cruciale sur le plan dénotationnel et opérationnel, et implique des stratégies d'évaluation spécifiques, centrées sur l'évaluation de tête.

Le calcul à distance λJ_n est le résultat d'une analyse de ΛJ à travers le prisme de l'utilisation des ressources, et diffère substantiellement de l'original. Sa construction est décrite dans une deuxième partie.

Résolubilité des applications généralisées. La résolubilité est une notion sémantique qui identifie les termes *significatifs*, c'est-à-dire les termes qui contribuent au résultat final. Dans un modèle sémantique du λ-calcul, les termes non significatifs peuvent être égalisés, ce qui signifie qu'ils peuvent être librement intervertis sans affecter le résultat du calcul. Une première intuition nous dicterait de considérer tous les termes non normalisables comme non significatifs. Cependant, égaliser tous ces termes s'avère être incohérent, car les modèles construits ainsi s'effondrent.

La notion de termes significatifs est en fait donnée par l'ensemble des termes solubles, qui est strictement plus grand que l'ensemble des termes normalisables : la réduction de certains termes ne termine pas, mais peut cependant contribuer au résultat de la réduction globale. Tous les termes solubles dévoilent progressivement une structure stable tout au long

du processus de réduction : cela donne un résultat partiel progressif qui est plus tard intégré dans la structure finale de la forme normale. Au contraire, si un terme contenant un sous-terme insoluble u converge vers un résultat, alors u peut être remplacé par n'importe quel autre terme, donnant toujours le même résultat et justifiant ainsi l'appellation d'insoluble comme *non significatif* (lemme de généricité [Bar84]).

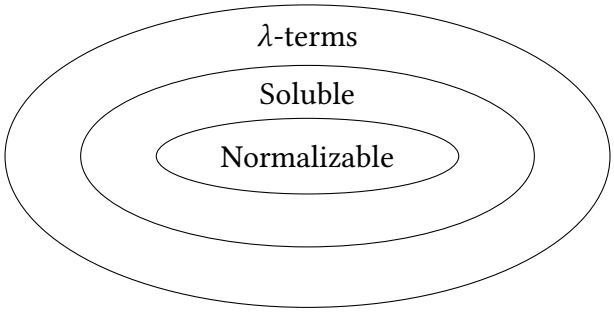


Fig. 0.3 : Il y a strictement plus de termes solubles que fortement normalisables.

Tout en étant une propriété sémantique importante, la résolubilité possède également une théorie opérationnelle très élégante. Un terme soluble peut se réduire à tout autre terme lorsque fermé par des abstractions et appliqué à une séquence d'arguments appropriée. Dans le λ-calcul en appel-par-nom, un terme t est soluble *si et seulement si* t a une forme normale de tête *si et seulement si* t normalise par l'évaluation de tête [Wad76].

La résolubilité peut être définie dans l'appel-par-nom, ainsi que dans l'appel-par-valeur. Mais en raison des comportements de normalisation différents, les notions correspondantes de résolubilité ne coïncident pas parfaitement [PR99].

L'étude de la résolubilité en appel-par-valeur est considérablement plus complexe, notamment à cause de l'absence de formalismes d'appel-par-valeur satisfaisants. En effet, une première caractérisation opérationnelle de la résolubilité par Paolini et Ronchi Della Rocca [PR99] utilise la réduction β, plutôt que βv de l'appel-par-valeur. Une caractérisation de la résolubilité en appel-par-valeur utilisant une notion de réduction en appel-par-valeur directement n'a été obtenue que récemment [AP12 ; CG14].

Le λ-calcul en appel-par-valeur de Plotkin est défectueux : certains termes qui sont insolubles d'un point de vue sémantique sont des formes normales *précoces*. Par exemple, dans un modèle sémantique des termes, le terme $(\lambda y.\lambda x.xx)(zz)(\lambda x.xx)$ se comporte comme le terme $\Omega = (\lambda x.xx)(\lambda x.xx)$ qui boucle et est insoluble. Pourtant, le premier terme ne se réduit pas car l'argument zz n'est pas une valeur, de sorte que la réduction βv ne se déclenche pas.

Dans le λ-calcul, la solution pour obtenir un calcul correct dans lequel la résolubilité peut être exprimée de manière opérationnelle est d'étendre le calcul de Plotkin. Une possibilité est d'étendre le calcul avec deux règles de permutation, dans l'esprit des règles σ de Regnier [Reg94], qui permettent de débloquer les formes normales prématurées. Cette solution est donnée par Carraro et Guerrieri [CG14].

$$(\lambda y.\lambda x.xx)(zz)(\lambda x.xx) \to_{\sigma_1} (\lambda y.\Omega)(zz) \to_{\beta\mathrm{v}} (\lambda y.\Omega)(zz) \to_{\beta\mathrm{v}} \dots$$

Une autre solution, d'Accattoli et Paolini [AP12] est d'utiliser un calcul à substitutions explicites, où chaque application d'une fonction à un argument est éliminée et où la distance peut être utilisée.

$$(\lambda y.\lambda x.xx)(zz)(\lambda x.xx) \to_{\mathrm{dB}} (\lambda x.xx)[y/zz](\lambda x.xx) \to_{\mathrm{dB}} \Omega[y/zz] \to_{\mathrm{dB}} \Omega[y/zz] \to_{\mathrm{dB}} \dots$$

Ainsi, la résolubilité est un bon critère pour juger un calcul (en appel-par-valeur), puisque sa caractérisation en tant que relation de réduction est suffisamment complexe pour mettre en évidence certains problèmes potentiels.

Pour le λ-calcul en appel-par-nom à applications généralisées, nous étendons les définitions et les techniques du λ-calcul dans la section 3.3.1 pour obtenir une relation *résolvante* capturant la résolubilité et étendant la réduction de tête du λ-calcul. Bien que le formalisme soit plus général, l'extension de la théorie est naturelle. La caractérisation est valable pour le variant à distance ainsi que que pour le calcul originel, pour lequel nous donnons une preuve directe dans la section 3.5.1. La résolubilité en appel-par-nom introduit des notions utiles à l'analyse plus complexe de la résolubilité en appel-par-valeur.

Pour l'appel-par-valeur, nous donnons une caractérisation opérationnelle interne de la résolubilité dans la section 3.4.2. Elle consiste en une relation de réduction qui ne possède pas les mêmes contextes d'évaluation et formes normales que son pendant en appel-par-nom. Ceci est dû au fait que les notions de normalisation ne sont pas les mêmes : la résolubilité en appel-par-nom est capturée par la normalisation de tête, alors que la résolubilité en appel-par-valeur correspond à la normalisation de tête plus l'évaluation faible sur tous les sous-termes effaçables. La similarité entre les sémantiques opérationnelles en appel-par-nom et par valeur dans les applications généralisées met en évidence les différences cruciales entre les deux notions de résolubilité, sur le plan opérationnel et syntaxique.

Par rapport au λ-calcul par valeur avec permutations, la relation résolvante présente l'avantage de ne pas impliquer de règles de permutation, de sorte que les transformations structurelles et calculatoires ne sont pas entrelacées. Les formes normales dans le calcul avec permutations sont plutôt complexes, en raison de la présence de radicaux bloqués. Ceux-ci contiennent en effet des applications d'abstractions telles que $(\lambda x.x)(yy)$. Au lieu de cela, nos formes normales sont simples et similaires à celles de la version en appel-par-nom et du λ-calcul : elles sont de la forme $\lambda x_1 \dots \lambda x_n.y(u_1, z_1.r_1) \dots (u_m, z_m.r_m)$ (avec même $m = 1$ quand la permutation π est utilisée indépendamment).

La caractérisation de la résolubilité dans les applications généralisées montre qu'il n'est pas nécessaire d'aller jusqu'aux substitutions explicites pour obtenir un bon formalisme pour l'appel par valeur. L'approche plus abstraite, où seules les applications sont partagées et où la réduction est effectué dans une règle unique, simplifie certains aspects de la théorie par rapport à celle décrite dans [AG22]. Avec l'application généralisée, il est également possible d'utiliser π comme une règle séparée, pour avoir des formes normales plus simples. En effet, une relation résolvante pour le calcul originel sans distance est donnée dans la section 3.5.1.

Nos deux notions de résolubilité en appel-par-nom et par valeur, caractérisées opérationnellement, correspondent-elles à la notion habituelle de résolubilité dans le λ-calcul ? Puisque la résolubilité est caractérisée en termes de normalisation, des systèmes de types intersection peuvent être donnés, où :

$$\text{Typabilité} \iff \text{normalisation} \iff \text{résolubilité}.$$

Nous donnons de tels systèmes de types dans la section 3.3.2 (appel-par-nom) et la section 3.4.3 (appel-par-valeur). Avec eux, nous identifions les nouvelles notions de résolubilité pour applications généralisées avec celles existantes (section 3.5.2). Cette notion sémantique de résolubilité en appel-par-valeur, caractérisée par des permutations, des substitutions explicites ou des applications généralisées correspond également à celle du calcul de Plotkin, bien qu'elle n'y soit pas directement exprimable. L'étude de la résolubilité en appel-par-valeur repose sur celle de la valuation potentielle, moins restrictive, et que nous capturons également de manière opérationnelle et logique.

Nous utilisons les caractérisations en appel-par-valeur pour deux autres résultats : la relation de résolution termine (propriété 3.67), et différentes définitions de la résolubilité sont équivalentes, ce qui est le cas en appel-par-nom mais pas toujours en appel-par-valeur [GN16].

Les types intersection non idempotents apportent aussi des preuves combinatoires courtes de la terminaison, ainsi que des limites sur la longueur de la réduction et la taille des formes normales.

À la fin du chapitre 3, nous comparons les réductions de λJ_v et $\lambda_{\mathtt{vsub}}$ opérationnellement par des simulations. Nous donnons également une bisimulation forte sur les termes avec applications généralisées et comparons les théories équationnelles de ces calculs avec l'addition des équivalences structurelles. Nous terminons en montrant une simple réduction normalisante pour l'évaluation forte dans l'appel-par-valeur avec des applications généralisées. De telles stratégies sont généralement beaucoup plus complexes dans d'autres calculs, tels que $\lambda_{\mathtt{vsub}}$ [Leb21].

Un calcul à application généralisées en appel-par-nom quantitatif. Les modèles donnés par les types quantitatifs ont les avantages des modèles qualitatifs donnés par les types intersection idempotents. En particulier, ils aident à comparer les propriétés de normalisation de différents formalismes. Mais ils permettent aussi de mesurer la différence du nombre d'étapes d'exécution entre différentes relations de réduction, et sont ainsi une première étape vers une analyse de complexité. Grâce à eux, nous pouvons associer des calculs à des systèmes sensibles à l'utilisation de ressources comme la logique linéaire.

Pourtant, le calcul en appel-par-nom originel ΛJ n'est pas compatible avec une sémantique quantitative. En effet, des propriétés cruciales du typage dans un système de types quantitatifs pour l'appel-par-nom échouent dans ΛJ (voir section 4.4.3). Cela se produit parce que π n'a pas un comportement adéquat quantitativement en appel-par-nom. Cette permutation a une nature appel-par-valeur qui affecte la durée d'exécution lorsqu'elle est utilisée dans un calcul en appel-par-nom. Cette permutation est acceptée par un système de type pour l'appel-par-valeur, mais pas par un système de type pour l'appel-par-nom. Il est intéressant de noter que [Mat00] a donné un système de type intersections idempotents pour ΛJ. Son système, qui n'est pas sensible au nombre d'étapes de réduction jusqu'à la forme normale, valide π, contrairement à notre système de types système de type quantitatif plus fin.

Nous ne pouvons pas nous passer complètement des permutations, car elles sont nécessaires pour débloquer certains radicaux bloquées. Nous introduisons donc une autre permu-

tation p2, qui n'affecte pas la longueur de la réduction dans une système en appel-par-nom :

$$t(u, y.\lambda x.r) \to_{\mathrm{p2}} \lambda x.t(u, y.r)$$

Nous intégrons cette permutation dans la règle β, selon le paradigme de la distance. La règle distante résultante ainsi que la syntaxe des applications généralisées donnent le nouveau calcul à distance en appel-par-nom λJ_n. Ce calcul est confluent (section 4.2) et les termes simplement typés terminent (section 4.2, théorème 4.4).

Nous montrons que λJ_n est compatible avec le modèle quantitatif dans la section 4.4. Pour la preuve de complétude (normalisable implique typable), nous donnons une définition inductive de la normalisation forte, qui est une contribution non triviale de ce travail.

Pour notre règle β à distance, nous nous inspirons des calculs à substitutions explicites, en ayant à l'esprit la traduction habituelle $t(u, y.r)^\star$ vers la substitution explicite $r[y/tu]$. Nous nous attendons à ce que le comportement dynamique de notre calcul soit *fidèle* aux substitutions explicites.

Une telle traduction, cependant, ne préserve pas en général la normalisation forte. En effet, dans un radical $(\lambda x.t)(u, y.r)$, l'interaction de $\lambda x.t$ avec l'argument u est matérialisée par la substitution interne dans le terme contracté $r\{y/t\{x/u\}\}$, comme mentionné précédemment. Mais une telle interaction est seulement apparente : si y n'est pas libre dans r, la réduction β effacera simplement l'abstraction $\lambda x.t$ et son argument u. Au contraire, $(\lambda x.t^\star)u^\star$ peut se réduire dans le contexte de la substitution explicite $r^\star[y/(\lambda x.t^\star)u^\star]$.

La différence d'interaction entre l'abstraction et son argument dans les deux modèles de calcul mentionnés a des conséquences importantes. Par exemple, soit $\delta \coloneqq \lambda x.x(x, z.z)$ le codage de $\lambda x.xx$ comme un terme à applications généralisées. Soit aussi r un terme normal sans occurrences libres libres de y, comme $r = \lambda x.x$. La seule réduction possible à partir de $\delta(\delta, y.r)$ est vers $r = \lambda x.x$, qui est une forme normale, alors que $\delta^\star\delta^\star$ peut se réduire à l'infini dans le contexte de la substitution explicite $r^\star[y/\delta^\star\delta^\star] \to^+ r^\star[y/\delta^\star\delta^\star]$.

C'est pourquoi nous proposons un nouvel encodage, préservant le type, des termes avec applications généralisées en termes avec substitutions explicites dans la section 4.5. En utilisant ce nouvel encodage et le système de types quantitatifs, nous montrons que la normalisation forte du terme source avec applications généralisées est équivalente à la normalisation forte du terme cible avec substitutions explicites, et donc aussi à celle du λ-calcul.

En guise de contribution finale, nous montrons la normalisation forte de λJ_n équivalente à celle du calcul originel ΛJ en section 4.6. En effet, nous souhaitons donner un calcul à applications généralisées quantitativement compatible avec un modèle pour l'appel-par-nom, mais sans perdre les propriétés sémantiques du calcul original. Nous extrayons de nouveaux résultats pour ce dernier, comme une traduction fidèle aux substitutions explicites et une nouvelle stratégie de normalisation. De plus, nous obtenons une caractérisation quantitative de la normalisation forte de ΛJ, où la taille des dérivations de types donne une borne supérieure au nombre de réductions β, mais pas π.

CHAPTER 1

Introduction

1.1 General Introduction

Imperative and declarative programming languages are often opposed, as they offer rather different styles of programming: machine versus specification-oriented.

Imperative programs are organized as a sequence of commands for manipulating the state of the environment where they run: storage access, input/output, jumps... using pointers, scan/print directives, exceptions... The execution of a program consists in the guided transformation of the state of the machine. Famous imperative languages include C, or the object-oriented C++ and Java.

On the other hand, *declarative* programs can be seen as a sequence of mathematical expressions not acting on the environment or execution flow. They take a high-level approach, so that a program resembles more to the specification of a problem in mathematical style. Control flow is left implicit: the programmer trusts the compiler or interpreter to execute the program on the machine in a reasonable way. The execution of a program consists in the *evaluation* of the expressions through their transformations to a result. Popular declarative languages are OCaml, Haskell or Coq (functional) and Prolog (logic).

```c
int factorial (int n) {
  int factn = 1;
  while (n >= 1) {
    factn = factn * n;
    n--;
  }
  return factn;
}
```

```
let rec factorial n =
  if (n = 1) then 1
  else n * factorial (n-1)
```

Figure 1.1: Imperative and declarative styles.

In this work, we focus on functional languages. Compared to imperative programs, functional ones are less error-prone thanks to their higher-level approach and powerful type systems, are easier to parallelize and prove correct. Functional languages produce smaller programs thanks to the use of first-class functions, pattern matching... They are actively developed, stimulated in part by the increasing demand for safety-critical applications. Meanwhile, modern imperative languages integrate more and more functional features and styles [vRos09; Hol16; KN19, §13].

1.1.1 Programming Language Semantics

A functional program is very far from a sequence of assembly instructions aimed at the processor. Focus is put on "what" rather than "how". But then, how do we make the transition to the machine, for which only the "how" matters? How do we specify the order of computation? What about potential optimizations? Parallelization? From the same program, two different compilers could give very different executions, sometimes with different results.

Researchers on *programming language semantics* are concerned with this kind of questions. Two important ones, that will also be apparent along this work, are:

1. Which transformations should we apply to programs, for evaluation, compilation, and optimizations?

2. How do we make sure that these transformations preserve the meaning of the original programs?

The word "semantics" in the term "semantics of programming languages" refers to *meaning*. The meaning of a program must be understood as what we want the program to do: light up a screen, loop forever, compute a result in less than ten seconds... Various kinds of semantics offer complementary point of views on programs. Two of them underlie our work.

Operational semantics is centered around syntax, and defines the meaning of programs as the way the expressions are to be computed, with syntactical rules, to obtain a result (called a *normal form*). It uses *rewriting relations* on programs, which model the transformations of the functional expressions of the program. The form of the code matters, since it determines further transformations.

Denotational semantics is concerned with properties on programs that are invariant along evaluation, such as termination (does a program eventually stop) or observational equivalence of two different programs (whether they have the same behavior). Checking that a transformation process conserves denotational properties is important. The study of denotational semantics is rooted in mathematics, as it interprets programs into an algebraic theory, abstract from the syntax. This interpretation should be the same for all correct transformations of the program.

The approach followed in this thesis is foundational and theoretical and follows ideas coming from operational semantics as well as denotational semantics. The first interest of modeling programs and languages in a mathematical language is to abstract over the necessary pragmatic details, which obstruct the peculiarities of the systems. A second interest is to appropriate the many tools of mathematics to prove correctness of program transformations, of programs themselves, or the efficiency of an implementation choice. Finally, this abstract model also offers the advantage that the results obtained are mathematically true, and will stay so, for many programs and languages at once.

Hence, rather than focusing on the implementation of a specific language, our work is centered around a pen-and-paper functional language: the *λ-calculus*.

The λ-calculus, a minimal functional language. The λ-calculus was created in the end of the 1920s by Church, and first published in 1932 [Chu32].[1] It was originally designed as a logical foundation of mathematics centered on functions, putting the emphasis on function application and the substitution process. It can be considered as the first functional programming language. This calculus provides both a mathematical theory of programs and computation, and an abstract model for (functional) programming languages. In this very elementary language, programs, named *terms*, are built out of only three constructors. We use the letters t, u, r and s to denote terms.

Variables such as $x, y, z, \ldots$ which range over terms.

Abstractions $\lambda x.t$, which can be understood as $x \mapsto t$: an anonymous function with parameter x and whose body is the term t. The occurrences of the variable x in the term t are said to be *bound* by the abstraction.

Applications tu, where the term t is applied to an argument u.

Programs of the λ-calculus are built inductively by nesting constructors on top of others. Examples are the term $\mathtt{I} \coloneqq \lambda x.x$ which is the identity function, $(\lambda x.x)(\lambda x.x)$ which is the identity function applied to itself, self-application $\delta \coloneqq \lambda x.xx$ and self-applied self-application $\Omega \coloneqq \delta\delta = (\lambda x.xx)(\lambda x.xx)$. The terms $\mathtt{I}$ and Ω will be used in this introduction.

In this model, the focus is put on the major component of computation in a functional program: the application of a function to an argument. Yet, this minimal language has as much computational power as any other programming language: every program and data can be encoded as a λ-term (although often tediously). For instance, the integer 3 can be encoded as $\lambda f.\lambda x.f(f(fx))$. In following examples, we use an extended syntax with integers and arithmetic operations, for the sake of illustration.

The λ-calculus has a syntactic part that we just described. But also a dynamic one, determined by the interaction of the different constructors. Computation in the λ-calculus is modeled as a *rewriting* sequence: steps in which the starting term is changed according to some predefined *reduction rules*. Rewriting models the evaluation of a functional program. Concretely, it is reminiscent of the way a simple arithmetic calculus is carried out, like:

$$5 \times 3 + 8 = 15 + 8 = 23. \tag{1.1}$$

In the first step of this computation, the subterm 5×3 gets rewritten to 15, according to the rule that $5 \times 3 = 15$. In the second step, the term is rewritten again according to another rule for addition. A result is reached when no more rules apply, upon the term 23 here.

Rewriting in the λ-calculus follows this scheme, with the difference that we use an arrow symbol $\longrightarrow$ instead of an equality, to emphasize the directed nature of rewriting. An example in the λ-calculus is the following one:

$$(\lambda x.x + 8)(5 \times 3) \longrightarrow 5 \times 3 + 8 \longrightarrow 15 + 8 \longrightarrow 23. \tag{1.2}$$

[1] A detailed history of the λ-calculus and combinatory logic is in [CH09].

The starting term is made of a function $\lambda x.x + 8$ with parameter x, applied to an argument 5×3. To evaluate it, we simply pass the argument as a parameter to the function by replacing the variable x by 5×3: we *substitute* 5×3 for x in $x + 8$. This first step is an example of the reduction rule of the λ-calculus. In the second step, our program is now $5 \times 3 + 8$. It is rewritten to $15 + 8$, then 23, using the encoding of natural numbers, addition and multiplication.

The dynamics of the λ-calculus are defined by a single reduction rule β:

$$(\lambda x.t)u \longrightarrow_\beta t\{x/u\}.$$

The term on the left is called a *redex*, for *red*ucible *ex*pression. A redex is an abstraction applied to one argument. This kind of term is not a definite result, and we can reduce it by substituting the argument u for x in t: this is denoted by $t\{x/u\}$, meaning that every occurrence of the variable x in t will be textually replaced by the term u. For instance:

$$(\lambda x.xx)(\lambda y.y) \longrightarrow_\beta (xx)\{x/\lambda y.y\} = (\lambda y.y)(\lambda y.y).$$

Here, $=$ denotes syntactical equality, since the substitution is a *meta-level* operation defined outside the calculus.

In the same way that the rule $5 \times 3 = 15$ was applied left of the addition in (1.1) and (1.2), reduction β can itself be applied anywhere inside a term. Take for instance the following, where the underlined redex occurs inside the body of the abstraction λy.

$$\lambda y.\underline{(\lambda x.xx)y} \longrightarrow_\beta \lambda y.(xx)\{x/y\} = \lambda y.yy$$

Extensions of the λ-calculus. The λ-calculus can be seen as a kernel of functional languages. However, concrete languages include many other constructors and data types, such as integers and recursion [Plo77], pattern matching [KvOdV08; AKV20] and monads [Mog91], among others. Each of these programming features can be considered in isolation inside minimal syntax for an extended λ-calculus. This enables one to focus on and study particular behaviors of the programming language by giving general results on them.

Abstract machines lie on an intermediate level of abstraction between the λ-calculus and concrete implementations. In particular, they provide an internal treatment of substitution and a mechanism for searching for a redex. Both of these mechanisms are specified by transformations that are executed stepwise at a local level, rather than on the whole term.

The λ-calculi with *ESs (explicit substitutions)* (see a survey in [Kes09]) are less concrete, as they only give an internal treatment of substitution. Their terms contain an additional constructor $t[x/u]$, the explicit substitution. This is a more compact notation for a let-binding `let` $x = u$ `in` t: the occurrences of x in t are bound to the term u.

The β-step is divided into two phases. A first one is the application of a B-rule which creates a new explicit substitution.

$$(\lambda x.t)u \mapsto_{\mathrm{B}} t[x/u]$$

This creates a sharing of the term, that is useful to avoid duplicating computations or grow the size of a term too much. The second phase consists in applying different reduction rules

to evaluate the previously fired substitution. The concrete evaluation steps in play vary according to the implementation of substitution considered.

Explicit substitutions give greater control over the substitution process. They enable the study of different flavors of substitution, like *linear* substitution which acts on one occurrence of a variable at a time, or, as we will see in the first part of this thesis, of *node replication*.

In the second part of the thesis, we investigate another extension of the λ-calculus. *Generalized applications* combine applications and ESs in one constructor. Both node replication and generalized applications arise from mathematical logic, by the *Curry-Howard* correspondence, a foundational link between logic and *type systems* of programming languages. One motivation of this work is to see in which way these features can be useful in a foundation of (functional) programming languages based on the λ-calculus.

Towards evaluation. When modeling programming languages with an abstract calculus, a difficulty is in finding a correct and interesting evaluation of the terms. Indeed, the λ-terms are descriptive programs that do not specify anything about the flow of execution. This is echoed by the nondeterminism of reduction: from a single starting term, several reductions are often possible. The example $(\lambda x.\mathtt{I}(xx))\mathtt{I}$ can be reduced either to $\mathtt{I}(\mathtt{II})$ by reducing the outer redex or to $(\lambda x.xx)\mathtt{I}$ by reducing under the left abstraction.

$$(\lambda x.\mathtt{I}(xx))\mathtt{I} \begin{array}{l} \nearrow \mathtt{I}(\mathtt{II}) \\ \searrow (\lambda x.xx)\mathtt{I} \end{array} \begin{array}{l} \searrow \\ \nearrow \end{array} \mathtt{II} \longrightarrow \mathtt{I}$$

Fortunately, each term always reduces to at most one result: the λ-calculus is *confluent*. Yet, choosing one execution path or the other has important implications.

First, while some reductions can lead to the unique result, some others may never reach it and reduce infinitely. For instance, take the following possible reduction paths, where $\Omega = \lambda x.xx$ is a term that reduces to itself: $(\lambda x.xx)(\lambda x.xx) \longrightarrow_\beta (\lambda x.xx)(\lambda x.xx)$.

$$(\lambda z.x)\Omega \begin{array}{l} \nearrow x \\ \searrow (\lambda z.x)\Omega \longrightarrow (\lambda z.x)\Omega \longrightarrow \cdots \end{array}$$

Second, take two terminating reduction paths. They can have arbitrarily different lengths: one could find the result in one step, and the other one in a million. In the illustration below, the loop on the starting term can be as long as we want, until we decide to reduce it to the normal form.

$$(\lambda z.x)\Omega \longrightarrow x$$

On the contrary, the evaluation of a program must be deterministic, as the compiler or interpreter must be able to find out what the next step is. Inside the λ-calculus, various deterministic *evaluation strategies* can be encoded by restricting evaluation. Including such

restrictions in the calculus is a crucial step in the modeling of programming languages. The minimal syntax of the λ-calculus makes it a tool of choice to inspect the various possibilities and relate them qualitatively or quantitatively.

The construction of a deterministic evaluation in the λ-calculus is done in three steps.

1. Choosing a parameter-passing policy (among call-by-name/value/need).
2. Defining the shape of the desired results and restricting evaluation accordingly.
3. Making reduction deterministic by giving an order on the redexes.

We will concentrate in the following on the first and second items, as the third is more closely related to the syntax of the calculus under consideration, and often straightforward.

Call-by-name, call-by-value and call-by-need. We consider three parameter-passing policies: CbN (call-by-name), CbV (call-by-value) and CbNeed (call-by-need). These three policies define three different λ-calculi.

Church's original λ-calculus implements *call-by-name* evaluation. Within the β-rule $(\lambda x.t)u \longrightarrow_\beta t\{x/u\}$, the arguments of functions are first copied, then evaluated. This is frequently expensive, as in the term $t = (\lambda x.xx)(\mathtt{II})$. The normal order (from left to right) CbN reduction sequence is the following, where the redex is underlined at each step. Remember that $\mathtt{II} \longrightarrow_\beta \mathtt{I}$.

$$t = \underline{(\lambda x.xx)(\mathtt{II})} \longrightarrow_\beta (xx)\{x/\mathtt{II}\} = \underline{(\mathtt{II})}(\mathtt{II}) \longrightarrow_\beta \underline{\mathtt{I}(\mathtt{II})} \longrightarrow_\beta \underline{\mathtt{II}} \longrightarrow_\beta \mathtt{I}$$

This happens because the argument II is itself a redex. Since there are several occurrences of x in the body of $\lambda x.xx$, the redex in the argument is copied naively, leading to a superfluous reduction step. In general, there are as many duplications of the argument as there are occurrences of the bound variable in the term. This can lead to an explosion of the number of steps, as well as of the size of the term.

This situation may be improved by *call-by-value*, in which arguments are evaluated first, then consumed. A CbV reduction from t contains one less reduction step.

$$t = (\lambda x.xx)(\underline{\mathtt{II}}) \longrightarrow_{\beta\mathrm{v}} \underline{(\lambda x.xx)\mathtt{I}} \longrightarrow_{\beta\mathrm{v}} (xx)\{x/\mathtt{I}\} = \underline{\mathtt{II}} \longrightarrow_{\beta\mathrm{v}} \mathtt{I}$$

The CbV λ-calculus uses a reduction rule βv, different to Church's original β.

$$(\lambda x.t)v \longmapsto_{\beta\mathrm{v}} t\{x/v\}$$

In this rule, the letter v denotes *values*: variables or abstractions $\lambda x.t$. This explains why the first step of reduction in the previous CbN reduction sequence is forbidden in CbV: the argument II is not a value. Call-by-value avoids many duplications of computation caused by the general β rule and is generally more efficient than CbN.

When talking about efficiency here, we are talking about the number of β/βv-steps. It should not be confused with a precise measure of complexity, as some of those steps could be costly to implement. We give an overview about cost models of the λ-calculus in section 2.6.

The CbV λ-calculus was introduced by Plotkin [Plo75] and underlies evaluation of languages like OCaml or Scheme. Like the CbN one, the CbV calculus is non-deterministic, and evaluation strategies need to be defined to implement a programming language.

Call-by-value is not always the best solution, though, because evaluating erasable arguments is useless. Compare for instance:

CbN $(\lambda x.z)(\mathtt{II}) \to_\beta z$, to:

CbV $(\lambda x.z)(\mathtt{II}) \to_{\beta_\mathrm{V}} (\lambda x.z)\mathtt{I} \to_{\beta_\mathrm{V}} z$.

This time, the CbV reduction takes one more step reducing an argument that is going to be erased anyway.

Crucially, some terms which normalize in CbN do not in CbV. Take again the term $(\lambda x.z)\Omega$. Now, compare:

CbN $(\lambda x.z)\Omega \to_\beta z$ to

CbV $(\lambda x.z)\Omega \to_{\beta_\mathrm{V}} (\lambda x.z)\Omega \to_{\beta_\mathrm{V}} \dots$.

We could see that the semantics of CbN and CbV are rather different. The CbV calculus in particular poses technical difficulties, and is still not as well understood as CbN. There is for instance no canonical well-behaved CbV λ-calculus (see section 1.2.2.1).

A third possibility is *call-by-need*, which takes the best of CbN and CbV: as in CbN, erasable arguments are not evaluated at all, and as in CbV, reduction of arguments occurs at most once. Precisely, CbNeed implements a *demand-driven* evaluation, in which erasable arguments are never needed (so they are not evaluated), and non-erasable arguments are evaluated only the first time they are needed, and the result of this evaluation is memoized for later uses. Call-by-need can intuitively be seen as CbN with memoization. Indeed, CbNeed evaluation finds the same results as CbN, and the set of normalizing terms is the same in both formalisms [Kes16].

Call-by-need is used in Haskell, under the name laziness. Besides efficiency in the number of β-steps, another possibility offered by lazy evaluation is the use of infinite data structures like streams. In eager languages, a special construction `lazy` must be added to delay evaluation of a subterm. However, semantical analysis of CbNeed is more complicated to carry out: knowing how long a program will run is difficult, due to the delays in the evaluation. Moreover, delay becomes problematic when introducing side-effects, since the order in which changes will be made on the state of the machine is less clear.

In the λ-calculus, some mechanism is needed to keep a unique shared copy of the argument after the β-reduction. The first instance of such a CbNeed reduction, devised by Wadsworth [Wad71], uses a representation of terms as directed acyclic graphs. A CbNeed reduction from the graph representation of $t = (\lambda x.xx)(\mathtt{II})$ is represented in figure 1.2 (with application nodes denoted by @).

The reduction is done at the level of the outermost redex, like in CbN, but thanks to the graphical representation, keeping a single instance of $\mathtt{II}$ shared over the occurrences of x is easy. In the next step, x is considered needed because it is located at the head of the term, in a position where a potential redex can be created. Therefore, we first reduce $\mathtt{II}$ to the

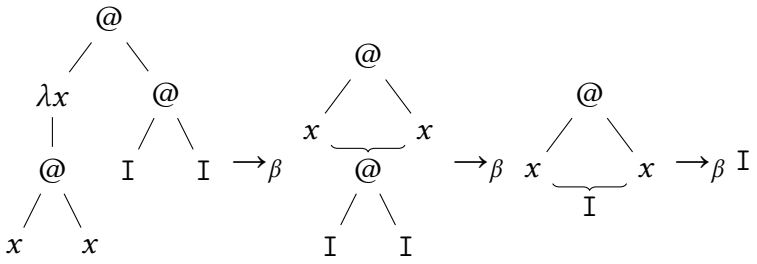


Figure 1.2: Graph reduction.

value I before replacing it. We only replace the occurrence of x that is needed: this is a *linear* substitution. After the second reduction, we have again a needed occurrence of x, so we replace it by the value I which was memoized.

A common way to implement CbNeed in the λ-calculus is to use explicit substitutions $t[x/u]$. Since in the term t, all occurrences of x are bound to the term u, this one only needs to appear once. The β-rule is decomposed in two: the rule creating an explicit substitution B, and another one linearly substituting values which we call sub.

$$(\lambda x.xx)y \longrightarrow_{\mathrm{B}} (xx)[x/y] \longrightarrow_{\mathrm{sub}} (yx)[x/y]$$

In a CbNeed reduction, the second possible B-step in the previous reduction is not fired, since the variable y is considered not needed. The graph reduction given above can be implemented with explicit substitutions, and only three B-rules will be necessary, as for CbV, which is one less than CbN.

However, CbNeed conserves the same notion of normalization as CbN, as the following example demonstrates.

$$t \coloneqq (\lambda x.z)\Omega \longrightarrow_{\mathrm{B}} x[z/\Omega] \not\to$$

The term $x[z/\Omega]$ is a normal form because z does not occur at the head of the term, so that the term Ω in the ES (explicit substitution) is not considered *needed*. In this way, CbNeed avoids the pitfall of CbV: the term t is strongly normalizing.

Even this wise evaluation scheme does not prevent unnecessary copies of redexes: while only *values* are duplicated, they may contain redexes as subterms, like $\lambda z.z(\mathtt{II})$ in which the subterm II is a redex. Duplicating this value will duplicate this inner redex (in color), as shown in figure 1.3.

Alas, keeping all values shared forever is impossible, typically when they potentially contribute to the creation of a future β-reduction step. The key idea to gain efficiency is then to keep the subterm II as a *shared* redex. For this, the value $\lambda z.z(\mathtt{II})$ to be copied is split into two separate parts. The first one, called *skeleton*, is $\lambda z.z\Diamond$, where $\Diamond$ is a placeholder. The skeleton contains the path from the top abstraction to all the occurrences of the bound variable z. It is highlighted in blue in the figure 1.4. The expression II is called a MFE (maximal free expression), and is the biggest expression that can stay shared without losing the scope of the abstraction. In general, there can be several separate MFEs for one term.

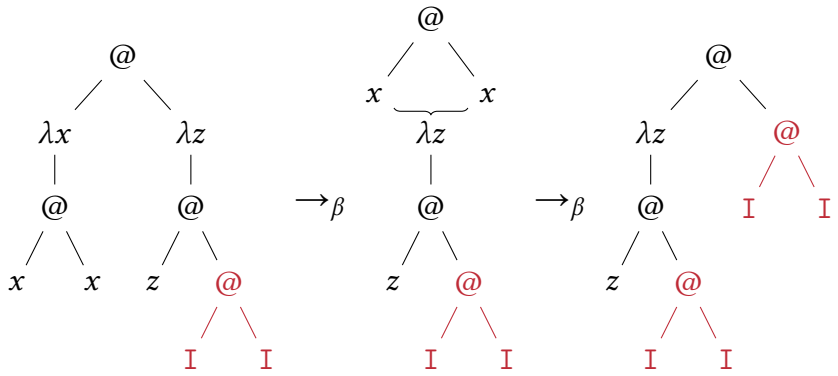


Figure 1.3: Redex duplication in CbNeed.

This optimization is called *fully lazy sharing* and is also due to Wadsworth [Wad71]. A fully lazy CbNeed reduction of the term $t = (\lambda x.xx)(\lambda z.z(\mathtt{II}))$ is shown in figure 1.4. Only the skeleton is copied, while the problematic redex $\mathtt{II}$ remains shared. When the subterm $\mathtt{II}$ is needed ahead, it is first reduced, as usual in CbNeed, thus avoiding to compute the redex twice.

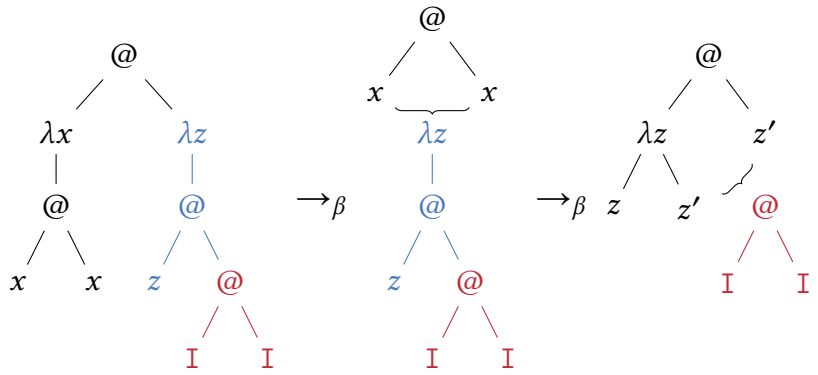


Figure 1.4: Fully lazy duplication.

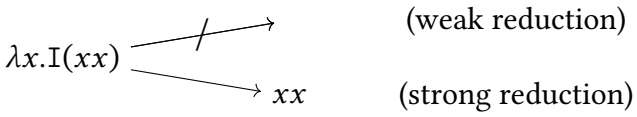
Call-by-name and call-by-value evaluation will appear through this work. As for full laziness, one of our results is its implementation in a λ-calculus with ESs.

Shaping results. Reducing a term to a full normal form (*i.e.* to a term that contains no redexes at all, at any position) is not always desired. Thus, once a parameter-passing policy is chosen, reduction can be refined, according to the shape of the results wanted.

A first possibility is to consider an abstraction, even with redexes inside, as a result. Indeed, in general-purpose programming languages, a function is a first-class element that can be used for instance as the argument of another function. There might be some computational steps left in the body of the function, but they are considered internal details that do

not appear in the interface of the function. The paradigm where reduction steps inside functions or abstractions are forbidden is called *weak reduction*. For instance, the term $\lambda x.\mathtt{I}(xx)$ is a normal form for weak reduction.

The unrestricted paradigm resulting in fully normalized results and where reduction is allowed also under abstractions, is known as *strong reduction*. In strong reduction, the weak normal form $\lambda x.\mathtt{I}(xx)$ reduces to $\lambda x.xx$. Strong reduction is crucial in the theory of the λ-calculus itself to get fully reduced results, but also for denotational studies, related in particular to *solvability*, which is the subject of chapter 3. Beyond theory, strong reduction, while more difficult to specify, has important use-cases in practice, notably the implementation of proof assistants,[2] and *partial evaluation* [JGS93]. In practice, strong reduction is generally implemented using the *leftmost-outermost* strategy, which reaches the normal form of a term every time there is one.



Weak and strong reduction can be combined with other guidelines on normal forms. An important one is restricting the calculus to *head* reduction to get head normal forms. An intuition is given by the following. Although $\mathtt{I}x$ is a redex, should the term $x(\mathtt{I}x)$ be a result? This redex is only an argument of the variable x. We can wait until x is replaced by an abstraction, rather than dealing with the arguments. Head reduction never reduces arguments. When combined with the constraint of weakness, we are talking about *weak-head* reduction, which is the one adopted (in a deterministic form) by general-purpose programming languages.

In the first part of this thesis, we consider an explicit order on the execution of redexes to obtain deterministic strategies that define a programming language. But in the second part, we instead consider reduction relations following principles like head or weak reduction, that are not deterministic. This enables a more general analysis of reduction, from which strategies can be easily derived, and the results hold for different implementations that follow the constraints.

One of the contributions of our work is the refinement of the calculi under study to reduction relations and strategies, aimed at the operational semantics of programming languages. Another one is to give them a particular kind of type systems rooted in denotational semantics. We now formally introduce type systems for the λ-calculus, and give an overview of their relation to logic.

1.1.2 Types

Typing is a guarantee. Types come up in most real-world programming languages, where they offer guarantees of well-behavior of programs. In these languages, every expression

[2] More precisely, of languages with dependent types, mainly proof assistants, in which type checking imposes to check syntactic $\beta\eta$-equality between terms [CH88; GL02].

has a type, that identifies the kind of data it represents: integers, characters, functions from integers to strings...

For the programmer, types offer a static guarantee: many bugs can be detected at compile time already, without needing to run the program. Therefore, the number of errors at execution is greatly lessened, and so is the debugging effort. For safety-critical programs, having a reliable static verification is mandatory, and types are an important part of it.

Types also offer an interface which guides the programmer. For functions, the type declaration tells them immediately what kind of datatype is needed as argument, and what kind of item the function will return. A few languages are dynamically typed: the correctness of instantiations, functions applications and forth, are only checked at execution time. But even there, statically typed variants, like TypeScript for JavaScript, are popular alternatives [Zap22].

Types are an essential element of the programming language semantics for at least three reasons. First, since many languages are typed, it is natural that a theoretical foundation of programming languages also consider types. Second, types are a tool to avoid bugs, by checking the correctness of written programs. One of the main goals of theoretical computer science is to distinguish correct from defective programs. Type systems can be formally inspected, ameliorated, or new ones can be proposed, sometimes for specific programming languages, other times for abstract models such as the λ-calculus and related systems. Thus, several type systems can exist for the same language.

The last reason is that types arise from the theory: decades before typed programming languages, types were devised as a logical tool for the foundation of mathematics by Russell and Whitehead [WR10]. They use types to restrict a too general foundational system that entails paradoxes. The principal role of types has not changed much: they forbid certain terms/programs which are syntactically constructible, but deemed semantically incorrect. So, type systems are objects of both mathematical logic and computer science, which is why they are at the core of the interface of these two disciplines with the Curry-Howard correspondence.

In the 1930s, both Church [Chu40] and Curry [Cur34] defined a *simple type system* for the λ-calculus, to reject certain terms expressing a logical paradox.

Simple types guarantee termination of λ-terms. In practical programming languages, the guarantees given by the type system can be manifold: using the correct methods on some data, not accessing an unallocated part of memory, testing all possible cases... The λ-calculus is a minimalist system, with no side-effects in particular. What behaviors can we consider unsound?

A first distinction between correct and incorrect programs is determined by the only observation we can make:[3] does a program terminate or not? Simple types offer a guarantee on strong normalization: every reduction of a typed program terminates. In other words every typed program can be eventually converted to a result. Yet, not every term whose reduction terminates is typable. Not every normal term is typable even: this is for example the case of $\lambda x.xx$ which is normal and untypable.

[3] Apart from confluence, valid for all terms.

The set of programs we would wish to type and the set of programs that are indeed accepted by a system commonly differ. There is a trade-off between:

1. the *expressiveness* of a type system: the amount of correct programs it recognizes and the flexibility and precision the programmer has, and

2. the computational *cost* of the associated typing algorithms: in particular *type checking*, *i.e.* verifying that the type annotations are valid for a program, and *type inference*, *i.e.* deducing the correct typing for a program.

Type systems in which all and only typable λ-terms are normalizable are seldom in practice, because of their undecidability [Urz99] and computational complexity [NM04], but present great theoretical advantages, as we will see in section 1.1.4 about intersection types.

Technically, type systems are defined as systems of *formal proofs* of mathematical logic. We thus introduce *proof theory* before discussing the simple type system for the λ-calculus and the underlying Curry-Howard correspondence.

Proof theory. Hilbert's 1901 program was a stepping stone in the search for a new foundation of mathematics. The logician wished to obtain a foundation of mathematics that would be:

Axiomatic, so that every theorem can be derived from a minimal amount of shared assumptions (axioms);

Complete, so that every statement can be proved or refuted;

Consistent, so that the same statement cannot be true and false; and

Computable, so that there is an algorithm deciding if a statement is provable or not.

Such a foundation should use a precise mathematical language. This means having non-ambiguous symbols for connectives such as ∨ (or), ⊃ (implies) or ¬ (not). For the computable part, there needed to be a mathematical description of algorithms and computation. This was given by the λ-calculus and Turing machines [Tur37] in particular. But these systems also allowed Church [Chu36] and [Tur37] to prove that Hilbert's deciding algorithm cannot exist. A previous objection to Hilbert's program is due to Gödel's incompleteness theorems [Göd31] stating that no system containing arithmetics can be proven complete and consistent. An important element of these metamathematics is the theory of formal proofs.

Proof theory is the line of research where the objects considered are syntactical representations of mathematical proofs. It is a tool to understand their logic and structure. Mathematical reasoning becomes itself an object of mathematics, conceptualized as a series of logical inferences. Every statement and hypothesis is represented as a formula. Proofs are (originally and in this work) finite and inductive, which means that they are built by assembling base elements together, in the same way that a program is constructed by assembling expressions and constructors. Many proof formalisms coexist, for different logics, with different inference rules.

In proof theory, the abstract notion of "truth" of a proof becomes an algorithmic notion of provability: given a proof system and a statement, can we prove this statement using the rules and axioms of that system? Proofs can be verified by computational methods: retracing the construction of the proof and checking if every step is well applied.

The existence of a formal theory of proofs makes possible computer-checked proofs, more reliable than human-checked proofs. Proofs can be either written by the user, or automatically generated. *Proof assistants* are in charge of verifying the "code" of the proof, while *automated theorem provers* can generate a correct proof of a statement. Proof generation is also used in *logic programming*, a descriptive programming paradigm notably present in Prolog [Mil21].

1.1.3 The Curry-Howard Correspondence

We will introduce the simple type system of the λ-calculus through its correspondence to *natural deduction* under the Curry-Howard correspondence.

Natural deduction and the λ-calculus. Natural deduction is a proof formalism, introduced by Gentzen [Gen35a; Gen35b]. In this system, proofs are represented as trees called *derivations*, with the leaves on top. The nodes of the tree are inference rules.

Within this formalism, different logics can be expressed. Of interest to us is the system for the implicational fragment of propositional logic, also called minimal logic, comprising only one connective: the implication $\supset$. The inference rules of that system are shown below.

$$\frac{}{\Gamma, A \vdash A} \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B} \qquad \frac{\Gamma \vdash A \supset B \qquad \Gamma \vdash A}{\Gamma \vdash B}$$

The variables A and B denote formulas of the minimal logic, which are built inductively using the connective $\supset$ from a set of arbitrary *atoms* $a, b, c, \ldots$. The basic atoms are left arbitrary, so that the focus is put on the steps of deduction, and is therefore also very generic.

The inference rules are represented using a horizontal bar separating *premises* from the conclusion. This bar means that if we can prove the premises inside the system, then we can prove the conclusion. The first rule comports no premise. Axioms serve as leafs of the derivation tree.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \supset B, A \vdash A \supset B} \qquad \dfrac{}{A \supset B, A \vdash A}}{A \supset B, A \vdash B}}{A \supset B \vdash A \supset B}}{\vdash (A \supset B) \supset A \supset B}$$

The relevance of this system for us is evident when considering the simple type system of the λ-calculus below.

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B} \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash tu : B}$$

The type system faithfully corresponds to the system of natural deduction. The only notable difference is that the type system can be seen as a version that is labeled with terms. Apart from this, the changes are mostly lexical: instead of *formulas*, we have *types*, the symbol for *implication* $\supset$ becomes a symbol for *functionality* $\longrightarrow$, and *proof derivations* become type derivations. What we derive is a *typing* for the term labeling the conclusion, based on the typings given for the terms in the premises.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{f : A \longrightarrow B, x : A \vdash f : A \longrightarrow B} \qquad \dfrac{}{f : A \longrightarrow B, x : A \vdash x : A}}{f : A \longrightarrow B, x : A \vdash fx : B}}{f : A \longrightarrow B \vdash \lambda x.fx : A \longrightarrow B}}{\vdash \lambda f.\lambda x.fx : (A \longrightarrow B) \longrightarrow A \longrightarrow B}$$

In the proof above, notice that the parameter f of the abstraction has a functional type $A \longrightarrow B$. The term $\lambda f.\lambda x.fx$ is an example of an higher-order function, characteristic of functional programs.

Lambda-terms act as compact representations of the proofs: each constructor of a term reflects one inference in the tree, since at each inference a constructor is removed from the term. This means that from a sequent $\Gamma \vdash t : A$ that is derivable in the simply typed system, we can reconstruct the full derivation, and obtain the natural deduction proof simply by removing the term and variable labels in the tree. A term seen as such a compact representation of a proof will be called *proof term*.

The connection between logic on one side and computer science on the other side with type systems, that is, the *Curry-Howard correspondence* [How80; SU06] is here rendered transparent by the presentation of both systems. Yet, it took several years for it to be worked out.

The Curry-Howard correspondence also lives at a dynamic level. In natural deduction, an introduction rule followed directly by an elimination acting on the same constructor is called a *detour*.

$$\dfrac{\dfrac{\Gamma, A \vdash B}{\Gamma \vdash A \supset B} \qquad \Gamma \vdash A}{\Gamma \vdash B} \qquad\qquad \dfrac{\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \supset B} \qquad \Gamma \vdash u : A}{\Gamma \vdash (\lambda x.t)u : B}$$

All detours can be eliminated by *detour conversion*, a process which reflects β-reduction of the λ-calculus.

In intuitionistic logic, detour conversion terminates, exactly like how reduction of simply typed λ-terms does. The Curry-Howard correspondence exhibits here the *computational content* of proofs: they are a static objects asserting a theorem as well as models of computation, related to the typed λ-calculus and similar systems.

Other correspondences. The connection between logic and computer science goes far beyond the λ-calculus and natural deduction. Indeed, many other systems than the simply typed λ-calculus and minimal logic have been brought in correspondence. Two examples are second-order logic (where quantifiers also act on formulas) and the System F of Girard

[Gir89] and Reynolds [Rey74], as well as linear logic and session types for the π-calculus, a calculus for concurrency [CPT14].

Thanks to the Curry-Howard correspondence, logical systems can be analyzed using the semantical tools of rewriting theory. The Curry-Howard correspondence is so ubiquitous that several important features of programming languages have been analyzed in logical terms. Likewise, recent or established logical systems have been used as an inspiration to devise new calculi. This has both a logical motivation, which is understanding the computational content of proofs, and a motivation in computer science, which is rooting programming ideas in the theory.

An example of a calculus extracted from some logic is the $\lambda\mu$-calculus of Parigot [Par92], which reveals that axioms of *classical* logic such as the excluded middle (either A is true, or $\neg A$ is) correspond to the control operators of programming, such as `call/cc`. Another example is the $\bar{\lambda}\mu\tilde{\mu}$-calculus [CH00], which is an interpretation of the presentation of classical logic in the *sequent calculus*.

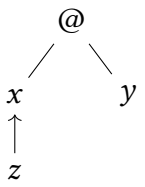
The sequent calculus is the other most important system of proofs, beyond natural deduction, and was also introduced by Gentzen [Gen35a; Gen35b]. As in the case of natural deduction, derivations are also trees with sequents for nodes, as in our presentation of natural deduction,[4] but inference rules are different, and can act either on the left or the right of the sequents. The dynamics of the $\bar{\lambda}\mu\tilde{\mu}$-calculus reflect the reduction of proofs by *cut-elimination* in the sequent calculus, in a similar way that the β-reduction simulates detour conversion in natural deduction.

After Gentzen's systems, alternative proof formalisms have been proposed, mainly to express different logics (like modal or multi-valued logics), but also to overcome syntactical limitations of existing systems. *Geometrical* formalisms like *proof nets* for *linear logic* represent proofs as graphs.

The construction of proofs as graphs has an advantage: it avoids some of the *bureaucracy* involved in sequential presentations of proofs, notably the sequent calculus. Indeed, it often happens that several proofs (even normal) are completely equivalent from a logical, semantical and dynamic point of view. Graphs offer a more flexible structure, which does not reflect the order of application of inference rules, and in which links can be drawn precisely between the relevant structures.

In sequential proof systems as well as in term calculi, some bureaucracy can be tamed with the addition of *permutation rules* enabling to rewrite one proof into an equivalent one, by moving inference rules or term constructors around. For the λ-calculus, permutations called σ-rules were given by Regnier [Reg94], inspired from linear logic proof nets. Another example is given by explicit substitutions. The two terms $x[x/z]y$ and $(xy)[x/z]$, where the variable z is shared over x, are represented by the same graph, with a direct link between x and z.

[4] We indeed presented natural deduction in a sequent-style, rather than the original presentation without sequents by Gentzen.



When x does not appear in t_2, a general equivalence on terms can be defined: $t_1[x/u]t_2 \sim (t_1 t_2)[x/u]$.

We now detail the two classes of languages we consider. They both have their origin in a Curry-Howard correspondence on two different proof systems for minimal logic.

Deep inference and node replication. The first calculus that we consider is a new calculus λR with ESs implementing what we call *node replication*. Node replication is a refinement of substitution, where terms are substituted constructor-by-constructor, or *node-by-node* if we see terms as trees.

Let us compare different mechanism to implement substitution of all the free occurrences of x by the term $u = y \cdot z$ (the multiplication dot denotes the application constructor): full (1.3), linear (1.4) and with node replication (1.5). The variable substituted at each reduction step is highlighted. Full substitution is the one of the λ-calculus, while linear substitution is the common model in well-known abstract machines for CbN and CbV.

$$(x \cdot x)[x/u] \longrightarrow u \cdot u \tag{1.3}$$

$$(x \cdot x)[x/u] \longrightarrow (u \cdot x)[x/u] \longrightarrow u \cdot u \tag{1.4}$$

$$(x \cdot x)[x/y \cdot z] \longrightarrow ((x_1 \cdot x_2) \cdot (x_1 \cdot x_2))[x_1/y][x_2/z] \longrightarrow ((y \cdot x_2) \cdot (y \cdot x_2))[x_2/z] \longrightarrow u \cdot u \tag{1.5}$$

Node replication offers the possibility to substitute only some part of the term, while keeping some subterms shared. In λR, the smallest part of the term that needs to be substituted is its *skeleton*. Using node replication, we will thus be able to define a fully lazy CbNeed evaluation strategy for the λ-calculus, with an operational semantics internal to the λR-calculus, whereas in the literature, full laziness is defined as an external function on terms.

Node replication seems to be a crucial element for *optimality*, in the sense of Lévy [Lév80]. A reduction is called optimal if for any term, it reaches a normal form in a number of steps equal to the length of the shortest of all reduction paths in the λ-calculus. In the (confluent) setting of the weak λ-calculus [LM99], the fully lazy optimization is optimal. This means that the fully lazy CbNeed strategy reaches the weak normal form in the same number of B-steps as the shortest possible weak reduction sequence in the usual λ-calculus without sharing.

Thus, fully lazy sharing turns out to be a *decidable* optimal strategy, in contrast to other weak evaluation strategies in the λ-calculus without sharing, which are also optimal but not decidable [Bal13], so for which it is mathematically impossible to give a specification. Node replication is also used in the graph reduction of Lamping [Lam90], which implements Lévy's optimal reduction [Lév80], optimal with respect to the full β-reduction. Again, being optimal does not mean that this strategy is the most cost-effective complexity-wise (see section 2.6).

Our calculus λR for node replication is a reinterpretation of the *atomic λ-calculus* λa of Gundersen, Heijltjes, and Parigot [GHP13b]. This calculus is itself a faithful Curry-Howard

$$\dfrac{\dfrac{(a \wedge b \wedge c) \vee (a \wedge b \wedge c)}{(a \wedge b) \vee (a \wedge b)}}{\dfrac{a \vee a}{a} \wedge \dfrac{b \vee b}{b}} \wedge \dfrac{c \vee c}{c}$$

Figure 1.5: An open-deduction proof.

interpretation of the *open-deduction* proof system for minimal logic, a proof system relying on *deep inference* [Gug07].

The intuition behind deep inference is rather simple. In Gentzen's systems, we are only able to apply inference rules for the outermost connectives. Deep inference permits the application of these inferences inside a context, so that they can act on deeper connectives. This paradigm is particularly useful to express modal logics [Brü10], or multiplicative linear logic with a sequential operator, which is not expressible with Gentzen's systems [Tiu06].

Open deduction is a proof formalism with a geometrical flavor, which avoids some bureaucracy imposed by the arbitrary order of applications of unrelated rules. It achieves this by allowing logical connectives to operate not only on formulas, but also on subproofs. An illustration of an open-deduction proof is given in figure 1.5 (a, b, c are atomic formulas). This example is taken from Guglielmi, Gundersen, and Parigot [GGP10], it is a derivation of $a \wedge b \wedge c$ from the assumption $(a \wedge b \wedge c) \vee (a \wedge b \wedge c)$.

The atomic λ-calculus was created as an interpretation of minimal logic formulated in open deduction. As a computational interpretation of a deep-inference system, the atomic λ-calculus has two main characteristics. The first one is of course node replication. The second is *linearity* of the variables: every variable appears exactly once in the term. For example, the term $\lambda x.xx$ is not valid, its translation in the atomic λ-calculus is: $\lambda x.(x_1 x_2)[x_1, x_2 \longleftarrow x]$. This term is reminiscent of calculi with ESs: indeed, in the atomic λ-calculus the constructor $t[x_1, \dots, x_n \longleftarrow u]$ shares u over the occurrences of $x_1, \dots, x_n$. Hence, a natural form of sharing appears in this calculus.

In our work, we only keep node replication, and reject linearity of variables. Removing the constraint on linearity enables us to formulate the semantics of node replication in terms of the well-known formalism of ESs, and to make connections to calculi using other forms of substitution. We obtain an original concise formulation of node replication which is simple enough to model different programming languages based on reduction strategies. In particular, full laziness can be implemented internally with the rules of the calculus, while in the literature this is realized with an ad-hoc meta-level operation on terms.

Generalized eliminations and applications. In the second part of this thesis, we consider calculi with *generalized applications*. The original calculus with generalized applications ΛJ was introduced by Joachimski and Matthes [JM03; JM00] as a Curry-Howard interpretation of the implicational fragment of von Plato's *natural deduction with generalized elimination*

rules [vPla01]. The difference with natural deduction lies in the implication elimination rule:

$$\frac{\Gamma \vdash A \longrightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B} \qquad\qquad \frac{\Gamma \vdash A \longrightarrow B \qquad \Gamma \vdash A \qquad \Gamma, B \vdash C}{\Gamma \vdash C}$$

The generalized rule on the right has one more premise. Instead of the usual *modus ponens* "from A implies B and A, we obtain B", in the generalized rule we derive a third formula C from a derivation in which B is assumed. Philosophically, the rule can be seen as a strict application of Prawitz's *inversion principle* [as given in NvP01]:

> Whatever follows from the direct grounds for deriving a proposition must follow from that proposition.

The idea of generalizing elimination in natural deduction starts before von Plato, notably with Schroeder-Heister [Sch84a], Prawitz [Pra79] and Tennant [Ten92]. In practice, generalized eliminations have several advantages. Proofs are in closer correspondence to sequent calculus ones, even non-normal (see [vPla01]). Furthermore, the rules unveil new permutation conversions, which enable the reduction of proofs to a so-called *full normal form*, that are in bijective correspondence with cut-free sequent calculus derivations, and where the main premise of each elimination is a leaf of the derivation tree. In the case of the implication this is:

$$\frac{\dfrac{}{\Gamma, A \longrightarrow B \vdash A \longrightarrow B} \qquad \dfrac{\vdots}{\Gamma \vdash A} \qquad \dfrac{\vdots}{\Gamma, B \vdash C}}{\Gamma \vdash C}$$

Interpreting generalized elimination in term syntax gives a λ-calculus with a generalized application constructor. Instead of tu, we have $t(u, y.r)$, which is typed with the following:

$$\frac{\Gamma \vdash t : A \longrightarrow B \qquad \Gamma \vdash u : A \qquad \Gamma, y : B \vdash r : C}{\Gamma \vdash t(u, y.r) : C}$$

Intuitively, this construction is to be understood as a let-binding $\texttt{let}\ y = tu\ \texttt{in}\ r$, or equivalently as an ES $r[y/tu]$. The application of t to u is bound to the variable y and *shared* over the occurrences of y in r.

Computationally, this calculus is interesting because it has a natural notion of sharing with its roots in proof theory. Sharing is useful or even necessary, in implementing CbV, CbNeed, or different kinds of optimizations. Espírito Santo [Esp20] indeed devised a CbV version of ΛJ (ΛJ_v), with an interesting operational semantics.

Unlike in calculi with ESs, only applications can be shared in ΛJ and ΛJ_v. In cases like CbV, this renders the syntax and semantics of the calculus less redundant than calculi with ES. Generalized applications present a slightly higher level of abstraction, closer in spirit to the λ-calculus, with a single β (or βv) rule for computation. Although ΛJ and ΛJ_v do not explicate the substitution process like calculi with ES, they can also be seen as an intermediate between the λ-calculus and abstract machines. They indeed posses a permutation rule, coming from proof theory, which implements the search for the leftmost redex.

Additionally, the natural deduction calculus ΛJ is a fragment of the calculus ΛJ_m [EFP18], an interpretation of a fragment of the intuitionistic sequent calculus. Insights on generalized applications could be useful to understand the intricate computational behavior of the sequent calculus. Espírito Santo [Esp09; Esp13] also use generalizations of ΛJ to study possible isomorphisms between natural deduction and the sequent calculus.

Finally, generalized applications have a flavor of ANFs [Fla+93] or CPS (continuation-passing style) [App91], both of which are important tools for the compilation of programs.

For generalized applications as well as node replication, our approach is guided by a *quantitative model* given by *non-idempotent intersection types*. We will finish this introduction by giving an overview of intersection types.

1.1.4 Intersection Types

Very concretely, intersection types systems [BDS09; vBak11] (introduced by Coppo and Dezani-Ciancaglini [CD80]) implement a simple idea: a term can be assigned several types simultaneously. In this way, more terms are covered by the system than with simple types. For instance, the normal form $\lambda x.xx$ is not simply typable. Indeed, suppose that we assign a type τ to the second x. Then the first one should be assigned a type $\tau \to \sigma$, for some σ. Then, x needs to be typed with the two types τ and $\tau \to \sigma$, which is not possible with a simple type system. In an intersection type system, the term $\lambda x.xx$ can instead be typed with $\tau \wedge (\tau \to \sigma)$.[5] Intersection types represent a kind of *ad hoc* polymorphism [Str00].

What we can see from this example is that intersection type systems validate more terms than the original simple-type one. In fact, the crucial property of intersection type systems, and what often motivates their use, is that *being normalizable* corresponds to *being typable*. More precisely, if we are considering the reduction $\mathscr{R}$ (that can be strong, head, weak-head...) of some calculus, then we are looking for an intersection type system $\mathscr{I}$ such that:

For any term t, t is typable in $\mathscr{I}$ *if and only if* t is $\mathscr{R}$-normalizable.

For head and strong reduction in the λ-calculus, such a property was first given by Coppo, Dezani-Ciancaglini, and Venneri [CDV81]. In other words, intersection type systems provide *logical models* of terms [PPR17].

Intersection types characterize semantical properties of terms: normalization, but also solvability, or observational equivalence. They are indeed syntactical representations of denotational models known as filter models [BCD83]. This makes them a simple tool to study properties of these models in a more syntactical way. Dually, semantical properties can be proved much more easily by going through typability instead of working directly on terms.

Intersection types can also be specified categorically, unveiling strong connections with models of linear logic [dCar17; GO21]. Mazza, Pellissier, and Vial [MPV18] give a general categorical approach to intersection types, from which type systems can be built, and normalization is proved for a class of systems in an abstract (extensional) way.

[5] Beware that the symbol $\wedge$ here does not denote conjunction, and the intersection and conjunction connectives are different connectives [Hin84].

In this work, we use the characterizations that we obtain for the different notions of normalization and the different calculi intensively. With it, we prove the correctness of our evaluation procedures (do we obtain a result of the desired shape?), and the equivalence between various notions of evaluation, such as CbN/CbNeed, and between the original calculi and the λ-calculus.

The expressiveness of intersection types, being able to type every normalizing term, has a drawback: type inference, as well as the dual problem of inhabitation (given any type, find a term that can be assigned this type) are undecidable [Urz99]. No algorithm can generate a solution for any input of these problems. This is an important drawback concerning practical matters, but is natural since the problem of normalization itself is undecidable. Still, types with intersection are used in some modern programming languages like TypeScript [Mic22], in conjunction with *union* types [BDD95; Cas21], where they are used to combine several object types together. The types considered in that case are of course restricted to a decidable fragment.

Non-idempotent intersections. The properties characterized by the idempotent intersection type systems are *qualitative*. They are yes/no questions such as: Does a term terminate? Are two terms observationally equivalent? We go further and use *non-idempotent* intersection types. Removing idempotence means that the type $\tau \wedge \tau$ is not equivalent to the type τ. Non-idempotent type systems were first defined by Gardner [Gar94], then used by Kfoury [Kfo00] and Neergaard and Mairson [NM04], and have since then been applied to a range of calculi [PR10; KV14; KV15; Dal+19; AKV20; RDF20] and to different formalisms such as call-by-value [Ehr12], call-by-need [Kes16; AGL19], call-by-push-value [Buc+20; KP22] and classical logic [KV20]. A survey is in [BKV17].

Each type in the intersection roughly tracks one "use" of the term it types along reduction. From there, a *quantitative* analysis arises. Besides asserting that a program terminates, non-idempotent intersection types also give a bound on the number of steps to normal form and on the size of the normal form [dCar07; dCar17]. For this reason, these type systems are also known as *quantitative type* systems, which is the name that we will mostly use in the following.

The choice of refining the qualitative model of (idempotent) intersection types into a quantitative one is consistent with our approach oriented toward programming languages, where the question of time and space complexity matters as much as bare termination. Having a logical model giving a bound on measures of the execution is indeed a first step toward precise complexity analysis of evaluation [Acc18a].

Another interesting feature of quantitative types is their sensitivity to quantitative properties: for instance, some permutations that have a CbV behavior might be rejected by a CbN type system, as in section 4.4.3. Moreover, the property of inhabitation becomes decidable. On a technical level, non-idempotence greatly simplifies proofs of normalization, generally automatic since type derivation decreases at each step of computation. Quantitative types are also a representation of denotational models, namely the *relational models* [dCar07].

The quantitative flavor of non-idempotent types will guide us into focusing the reductions we consider on computation, as well as being cautious with the permutations used. The

combinatorial nature of normalization proofs will also help diminish the amount of technical content.

1.2 Contributions

This thesis gathers and expands three articles (each in different a chapter). Chapter 2 is written in collaboration with Delia Kesner and Daniel Ventura, and is based on [KPV22], a submitted journal article following [KPV21]. Chapter 3 is based on [KP22], written in collaboration with Delia Kesner. Sections 3.6 and 3.7 are original to this thesis. Chapter 4 is based on [EKP22], written in collaboration with Delia Kesner and José Espírito Santo. The proof of confluence for the calculus (section 4.2) is original.

These different works originate from a quantitative analysis of recent calculi originating in proof theory (the atomic λ-calculus and generalized applications). We develop an operational and a quantitative theory influenced by an approach aimed towards the theory of programming languages.

We state the main research question of this thesis as follows:

> What contributions do node replication and generalized applications, analyzed quantitatively, provide to the theory of programming languages?

More precisely, our contributions are twofold. On one hand, we give detailed operational semantics of calculi with node replication and generalized applications. This consists in particular of the definition of reduction relations corresponding to different notions of evaluation and normalization, that are of interest for programming language semantics. On the other hand, we give quantitative type systems for these reductions relations. We use them as an inspiration to refine the calculi, as a technical tool to simplify proofs of normalization and as a semantical tool to prove equivalence of semantical properties among different calculi.

One way in which the quantitative model influences the definition of the calculi is through the use of *distance* [AK10; ABM14] to focus on computation. In calculi with ES, generalized applications or other sequential term calculi, permutation rules are necessary to *unblock* some expected redexes. Take for instance the term $(\lambda x.t)[y/r]u$. The term u is the argument of the abstraction $\lambda x.t$, but we do not have yet a B-redex, since an explicit substitution separates $\lambda x.t$ and u. We use a directed version of the equivalence of terms with ES defined before, to permute the explicit substitution and make the redex emerge.

$$(\lambda x.t)[y/r]u \to_{\sigma_1} ((\lambda x.t)u)[y/r] \to_{\mathrm{B}} t[x/u][y/r]$$

The two terms to the left and to the right side of the permutation σ_1 indeed have the same graph representation. In the graph, reduction can be done straightaway in a unique computational step.

Inspired from this formalism, the distant paradigm gathers meaningful and permutation rules in only one reduction step. In this way, the semantics of sequential representations of terms is brought closer to the graphical ones, with often a one-to-one correspondence [KL07; Acc18b; Kes22].

In calculi with explicit substitutions, the two steps are replaced by a single dB-step. Overall, only the permutations that are necessary to unblock meaningful reductions are fired.

$$(\lambda x.t)[y/r]u \longrightarrow_{\mathrm{dB}} t[x/u][y/r]$$

The choice of distance also reflects the logical models in a better way: quantitative types are mostly neutral to permutation rules which are only relevant from a structural point of view. Indeed, quantitativity is only related to the computational rules. With distance, every step now represents a meaningful computational step.

Now that distance is introduced, we can precisely name the calculi analyzed and introduced in this thesis:

- In the first part, an original calculus with ES implementing *node replication* and using a semantics at a *distance*.
- In the second part, the CbN and CbV calculi with *generalized applications*, and original variants using a semantics at a *distance*.

1.2.1 Node Replication

The main objective of the first part of the thesis (chapter 2) is to introduce the theory and practice of node replication, inside the framework of the λ-calculus. We use a novel calculus with explicit substitutions λR, which we introduce in section 2.1. This calculus is a reinterpretation of the atomic λ-calculus, and uses distance to highlight the mechanisms of node replication.

Compared to the atomic λ-calculus, the perspective changes. While Gundersen, Heijltjes, and Parigot [GHP13b] give a computational interpretation of open deduction, we want to give a fine analysis of substitution in the λ-calculus and programming languages in general by adding the possibility of substituting node by node.

We give some general properties of the calculus λR in section 2.2: termination of the process of substitution with node replication, confluence and simulations with the λ-calculus.

The calculus is then refined to two deterministic evaluation strategies. The first one is CbN (section 2.3.1), and does not make use of the optimizations brought by node replication. As an implementation of the CbN (weak-head) reduction of the λ-calculus, it serves as a link between the λ-calculus and more elaborate strategies using node replication.

The second strategy implements (weak) fully lazy CbNeed (section 2.3.2). Several implementations of full laziness exist in the literature (see section 2.6), starting with the original one by Wadsworth [Wad71]. But in them, the crucial point of the extraction of the maximal free expressions is done at meta-level, and relies on external definitions of the skeleton. On the contrary, Gundersen, Heijltjes, and Parigot [GHP13b] show how a fully lazy extraction can be performed within the atomic λ-calculus. We build on these results, and integrate skeleton extraction in a call-by-need strategy to construct a fully-lazy call-by-need strategy. This strategy formalizes an operational semantics in which the steps leading to this construction are internal. Therefore, the computation is self-contained and described fully operationally.

We give two kinds of semantics for the splitting of the skeleton and free expressions: the first one is a *big-steps* semantics [Kah87], that reformulates the proof of Gundersen, Heijltjes, and Parigot [GHP13b] that skeleton extraction can be implemented by the atomic λ-calculus. The second is a *small-step* semantics, that details how to extract a skeleton step-by-step using the rules of the calculus λR. We show that these two semantics correspond to two different but equivalent definitions of a skeleton.

While it has been shown that call-by-name and call-by-need specified by means of full and linear substitution (respectively) are observationally equivalent [AF97], it was not clear at first whether the same property would hold in our case. A further contribution is a proof of this result using a quantitative type system in section 2.5. This proof technique [Kes16] considerably simplifies other approaches [AF97; MOW98] based on syntactical tools. Moreover, the use of intersection types has another important consequence: standard CbN and CbNeed turn out to be observationally equivalent to CbN and CbNeed with node replication, as well as to the more semantical notion of neededness [KRV18]. This is to our knowledge the first quantitative characterization of fully lazy normalization.

1.2.2 Generalized Applications

What do generalized applications bring to the theory of programming languages? We argue that they offer a different level of abstraction compared to existing formalisms. They are characterized by two features: a notion of sharing restricted to applications, and a simple internal management of the search for a redex.

Sharing is permitted by the generalized application constructor $t(u, y.r)$, where the term tu is shared over the occurrences of y in r. This sharing is useful to avoid duplicating some computations. Since β-redexes are applications, they are all shared by default. Yet, sharing is not as general as in calculi with let-bindings, where every kind of term can be shared, and as the one of calculi with ES, which also posses an internal treatment of substitution. Generalized applications keep substitution at a meta-level. In consequence, the computation is still done in one unique step (a generalized β-step shown below), rather than in two phases, as with explicit substitutions. The operational semantics of the computation is thus simpler:

$$(\lambda x.t)(u, y.r) \longrightarrow_\beta r\{y/t\{x/u\}\}$$

Sharing applications is particularly useful for CbV. Unlike most CbV calculi [AG16], the calculus ΛJ_v (or our new distant version) does not impose any restriction on the redexes. This means that every function application is a redex that can be fired. Some shortcomings of CbV formalisms are thus avoided. More: CbV computation is done by means of a rule almost identical to CbN, only relying on a different notion of (meta-level) substitution (defined in section 3.1.1):

$$(\lambda x.t)(u, y.r) \longrightarrow_\beta r\{y\backslash\backslash t\{x\backslash\backslash u\}\}$$

Having the same redexes $(\lambda x.t)(u, y.r)$ in CbN and CbV means also that for any notion of normalization, defining a CbV strategy is simple: the definition of normal forms is the same, and for many interesting strategies, the same inductive reduction rules can be chosen.

This is the case for instance of strong normal forms, defined in section 3.7 and section 4.2, and of the leftmost-outermost CbV strategy, which uses the same inductive rules that a CbN strategy would.

The "search for a redex" is provided by the permutation rule of the calculus, named π, which is one of the hidden permutations revealed by von Plato [vPla01]:

$$t(u, x.r)(u', y.r') \to_\pi t(u, x.r(u', y.r'))$$

Concretely, this permutation moves the leftmost redex on top, as in the following example.

Example 1.1. The following reduction is represented in figure 1.6 graphically, to make apparent how the leftmost redex is brought on top of the term.

$$\begin{aligned}(\lambda x.t)(u_1, y_1.y_1)(u_2, y_2.y_2)(u_3, y_3.y_3) &\to_\pi (\lambda x.t)(u_1, y_1.y_1)(u_2, y_2.y_2(u_3, y_3.y_3)) \\ &\to_\pi (\lambda x.t)(u_1, y_1.y_1(u_2, y_2.y_2(u_3, y_3.y_3)))\end{aligned}$$

In the closed (no free variables) and weak-head setting which is the one general-purpose programming languages, permutation π enables computation to reach a normal form without diving inside the term.

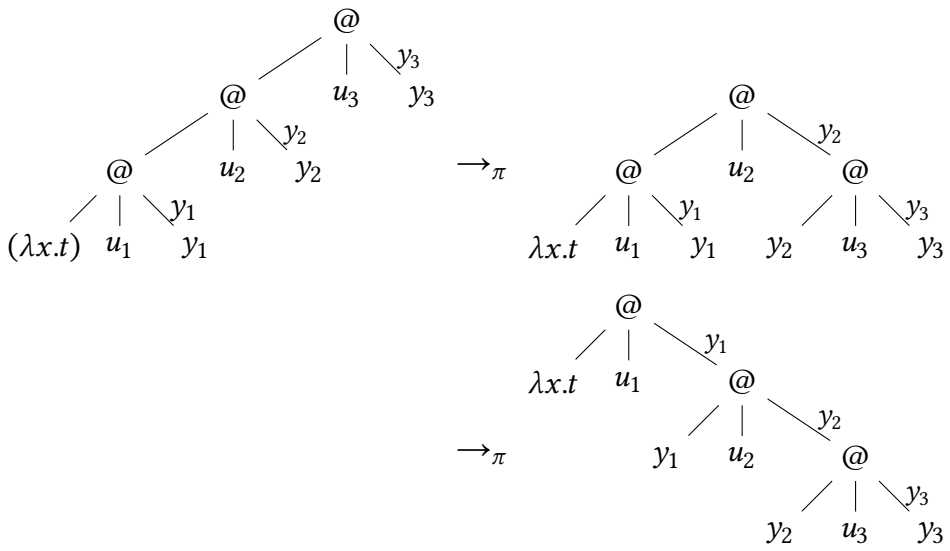


Figure 1.6: Permutation π illustrated on the syntax tree.

Interestingly, this last example is a translation of some λ-term $(\lambda x.t')u'_1u'_2u'_3$, in which t' is translated to t, and u'_i to u_i for $1 \leq i \leq 3$. Inside an abstract machine like the one of Krivine [Kri07], the terms u_1 to u_3 would successively be moved into the stack. Generalized applications provide a representation of the stack directly inside the term, and the reduction step of the abstract machine moving the right element of applications inside it is replaced by

a permutation π. The philosophy is similar with CPS and ANF (administrative normal form), which give a name to every intermediate computation to encode the stack internally.

Using distance enables to gain in abstraction. By integrating permutation into the rule β, there is no more an explicit step revealing the leftmost redex, but only a single computational rule. Thus, the calculus with generalized applications is made closer to the λ-calculus, with the only different feature being that applications are named and shared. Generalized applications with distance can then also be seen as more abstract and simple versions of calculi with sharing. In this work, we prioritize distant variants of the original CbN and CbV calculi, to stay as close to the λ-calculus and to the resource-aware model given by quantitative types as possible.

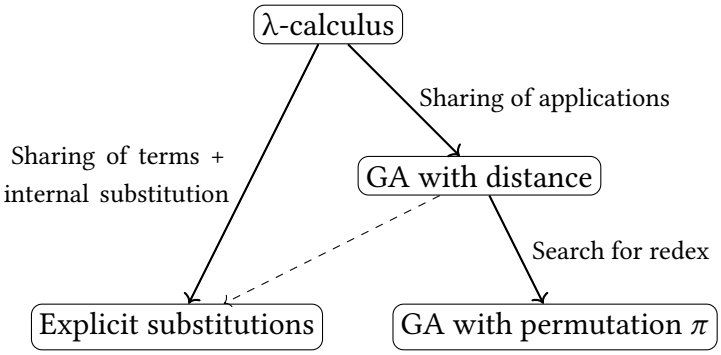


Figure 1.7: Different paths toward implementation.

Despite the practical aspects of generalized applications, detailed studies of their operational semantics lack in the literature. The works of [Esp09; EFP18] look at generalized applications as a tool for proof theory. The works of Joachimski and Matthes on ΛJ, and by Espírito Santo on ΛJ_v introduce the calculus, give strong normalization of the typed calculus, as well as confluence and *standardization* in the first case. This approach centered around strong normalization is again oriented from the point of view of proof theory.

We take a different approach, inspired by programming language semantics. We look at the notion of *solvability* for CbN and CbV calculi with generalized applications, first for the distant versions λJ_n and λJ_v, and then transpose the results to the original ΛJ and ΛJ_v. Solvability is crucial denotationally and operationally, and involves specific evaluation strategies, centered around head evaluation.

The distant calculus λJ_n is the result of an analysis of ΛJ through the lens of computation and resource usage, and differs substantially from the original. Its construction is described in a second part.

1.2.2.1 Solvability for Generalized Applications

Solvability is used to identify *meaningful* terms, that is, terms which contribute to the final result. In a semantical model of the λ-calculus, meaningless terms should be equated, meaning that they could be freely swapped without affecting the result of the computation. A first approach would consist in equating all strongly non-normalizing terms and deeming them

as meaningless. However, equating all non-normalizable terms turns out to be inconsistent, as the model would collapse.

The actual notion of meaningful terms is given by the set of solvable terms, which is strictly bigger than the set of normalizing terms: reduction of some terms do not terminate, but can still contribute to the result of the computation. All solvable terms progressively unveil a stable structure along the reduction process: this gives a step-by-step partial result that is later integrated into the definitive structure of the fully normalized term. On the contrary, if a term containing an unsolvable subterm u converges to a result, then u can be replaced by any other term, still giving the same result and thus justifying the designation of unsolvable as *meaningless* (Genericity Lemma [Bar84]).

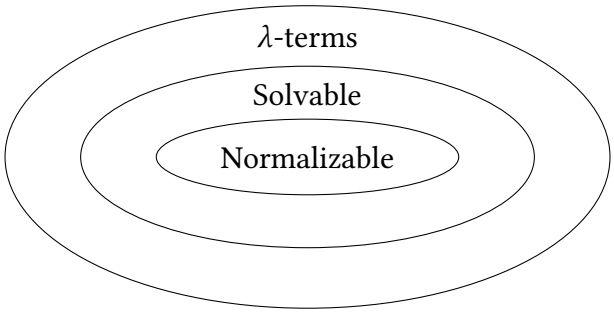


Figure 1.8: There are strictly more solvable than strongly normalizable terms.

Whilst being an important semantical property, solvability also has a very elegant operational theory. A solvable term may reduce to any other term when closed by abstractions and applied to a suitable sequence of arguments. In the CbN λ-calculus, a term t is solvable *iff* t has a *head normal form iff* t *head*-normalizes [Wad76].

Solvability can be defined in CbN as well as in CbV. But because of the different normalization behaviors of CbN and CbV, their corresponding notions of solvability do not perfectly coincide [PR99]. The study of CbV solvability is considerably more complex, due in part to the lack of satisfying CbV calculi for a long time. In fact, a first operational characterization of solvability by Paolini and Ronchi Della Rocca [PR99] uses β, rather than βv, reduction. A characterization of CbV solvability making use of some proper notion of CbV reduction was only achieved recently [AP12; CG14].

Plotkin's original CbV λ-calculus is defective: some terms which are unsolvable from a semantical point of view are *premature* normal forms. For instance, in a semantical analysis of terms, the term $(\lambda y.\lambda x.xx)(zz)(\lambda x.xx)$ behaves like the looping and unsolvable term $\Omega = (\lambda x.xx)(\lambda x.xx)$. Yet, the first term does not reduce because the argument zz is not a value, so that βv-reduction does not apply.

In the λ-calculus, the solution to obtain a correct calculus where solvability can be expressed operationally is to extend Plotkin's calculus. One possibility is to extend the calculus with two permutation rules, in the spirit of Regnier's σ-rules, which enable to unblock premature normal forms [CG14].

$$(\lambda y.\lambda x.xx)(zz)(\lambda x.xx) \to_{\sigma_1} (\lambda y.\Omega)(zz) \to_{\beta\mathrm{v}} (\lambda y.\Omega)(zz) \to_{\beta\mathrm{v}} \dots$$

Another solution is to use a calculus with ES [AP12], where every function application is eliminated and distance can be used.

$$(\lambda y.\lambda x.xx)(zz)(\lambda x.xx) \to_{\mathrm{dB}} (\lambda x.xx)[y/zz](\lambda x.xx) \to_{\mathrm{dB}} \Omega[y/zz] \to_{\mathrm{dB}} \Omega[y/zz] \to_{\mathrm{dB}} \dots$$

Thus, solvability is a good criterion to judge a (CbV) calculus, since its characterization as a reduction relation is complex enough to highlight some potential problems.

For the CbN λ-calculus with generalized applications, we extend the definitions and techniques from the λ-calculus in section 3.3.1 to obtain a *solving* relation characterizing solvability, extending the head reduction of the λ-calculus. Although the formalism is more general, the extension of the theory is natural. The characterization is valid for the distant as well as the original calculus, for which we give a direct proof in section 3.5.1. Call-by-name solvability introduces notions that are useful for the more complex analysis of call-by-value solvability.

For CbV, we give an internal operational characterization of solvability in section 3.4.2. It consists of a reduction relation which does not possess the same evaluation contexts and normal forms as its CbN counterpart. This is because the notions of normalization corresponding to CbN and CbV solvability are not the same: CbN solvability is captured by head normalization, while CbV solvability corresponds to head normalization plus weak evaluation on all the erasable subterms. The similarity between the CbN and CbV reductions in generalized applications highlights the crucial differences between the two notions of solvability, on operational and syntactical levels.

Compared to the CbV λ-calculus with permutations, the solving relation has the advantage that no permutation rules are involved, so that structural and computational transformations are not interleaved. Normal forms in the calculus with permutations are rather intricate, due to the presence of stuck redexes. They contain function applications such as $(\lambda x.x)(yy)$. Instead, solving normal forms are simple and similar to those of the previous CbN reduction and of the λ-calculus: they are of the shape $\lambda x_1 \dots \lambda x_n.y(u_1, z_1.r_1) \dots (u_m, z_m.r_m)$ (with even $m = 1$ when using π independently).

The characterization of solvability in generalized applications shows that going as far as explicit substitutions is not necessary to obtain a good formalism for call-by-value. The more abstract approach, where only applications can be shared and computation is done in a unique rule, simplifies some aspects of the theory with respect to the one described in [AG22]. With generalized application, it is also possible to use π as a separate rule, to have simpler normal forms. Indeed, a solving relation for the original calculus without distance is given in section 3.5.1.

We have two notions of CbN and CbV solvability with an operational characterization, but do they correspond to the usual notion of solvability in the λ-calculus? Since solvability is characterized in terms of normalization, intersection types systems can be given, where

$$\text{Typability} \iff \text{normalization} \iff \text{solvability}.$$

We give such type systems in section 3.3.2 (CbN) and section 3.4.3 (CbV). With them, we relate the new notions of solvability for generalized applications with the existing ones in section 3.5.2. This semantical notion of solvability in CbV, characterized with permutations,

explicit substitutions or generalized applications also corresponds to the one of Plotkin, despite it not being expressible in his original calculus. The study of CbV solvability relies on the one of potential valuability, less restrictive, and which we also capture operationally and logically.

We use the CbV characterizations for two more results: the solving relation is normalizing (property 3.67), and different definitions of solvability are equivalent, which is the case in CbN but not always in CbV [GN16].

Using non-idempotent intersection types here also brings of short combinatorial proofs of termination, as well as bounds on the length of reduction and size of normal forms.

In the end of chapter 3, we compare the CbV reductions of λJ_v and $\lambda_{\texttt{vsub}}$ operationally through simulations. We also give a strong bisimulation on $\mathtt{T}_J$ and compare the equational theories of these calculi augmented with structural equivalences. We finish by showing a simple normalizing reduction for strong evaluation in the CbV calculi with generalized applications. Such strategies are usually much more involved in other calculi, such as $\lambda_{\texttt{vsub}}$ [ACS21].

1.2.2.2 A Resource-Aware CbN Calculus with Generalized Applications

The models given by quantitative types have the advantages of the qualitative models of idempotent intersection types. In particular, they help in comparing normalization properties of different formalisms. But they also make short combinatorial proofs of normalization possible. They allow to measure the difference of the number of execution steps between different reduction relations, and are thus a first step toward complexity analysis. With them, we can relate calculi to resource-aware systems like linear logic.

Yet, the original CbN calculus ΛJ is not compatible with a resource-aware semantics. Indeed, crucial properties relating typing in a quantitative type system for CbN strong normalization and reduction in ΛJ fail (see section 4.4.3). This happens because π is not quantitatively well-behaved. This permutation has a CbV nature that affects the length of execution when used in a CbN calculus. This permutation is accepted by a CbV type system, but not a CbN one. Interestingly, Matthes [Mat00] gave an idempotent intersection type system for ΛJ. His system, which is not sensitive to the number of reduction steps to normal forms, validates π, unlike our finer quantitative type system.

We cannot dispense from permutations altogether, because they are necessary to unblock some stuck redexes. We introduce a different permutation p2, which does not affect the length of reduction in a CbN system.

$$t(u, y.\lambda x.r) \to_{\mathrm{p2}} \lambda x.t(u, y.r)$$

But we integrate this permutation inside the primary β rule, following the distance paradigm. The resulting distant rule together with the syntax of generalized applications gives the new distant calculus CbN calculus λJ_n. This calculus is confluent (section 4.2) and simply-typed terms terminate (section 4.2, theorem 4.4).

We show that λJ_n is compatible with the quantitative model in section 4.4. For the completeness proof (normalizable implies typable), we give an inductive definition of strong normalization, which is a non-trivial contribution of this work.

We draw inspiration for our distant β rule from calculi with explicit substitutions, having in mind the usual translation $t(u, y.r)^{\star}$ to the explicit substitution $r[y/tu]$. We expect the dynamic behavior of our calculus to be *faithful* to explicit substitutions.

Such translation, however, does not in general preserve strong normalization. Indeed, in a β-redex $(\lambda x.t)(u, y.r)$, the interaction of $\lambda x.t$ with the argument u is materialized by the internal substitution in the contractum term $r\{y/t\{x/u\}\}$, as mentioned before. But such interaction is elusive: if the external substitution is vacuous (that is, if y is not free in r), β-reduction will simply throw away the λ-abstraction $\lambda x.t$ and its argument u, whereas $(\lambda x.t^{\star})u^{\star}$ may reduce in the context of the explicit substitution $r^{\star}[y/(\lambda x.t^{\star})u^{\star}]$.

The different interaction between the abstraction and its argument in the two mentioned models of computation has important consequences. For instance, let $\delta \coloneqq \lambda x.x(x, z.z)$ be the encoding of $\lambda x.xx$ as a ΛJ-term. Then, let r be a normal term with no free occurrences of y, say $r = \lambda x.x$. The only possible reduction from $\delta(\delta, y.r)$ is to $r = \lambda x.x$, which is a normal form, whereas $\delta^{\star}\delta^{\star}$ may reduce forever in the context of the vacuous explicit substitution $r^{\star}[y/\delta^{\star}\delta^{\star}] \longrightarrow^{+} r^{\star}[y/\delta^{\star}\delta^{\star}]$.

That is why we propose a new, type-preserving, encoding of terms with generalized applications into terms with explicit substitutions in section 4.5. Using this new encoding and quantitative types, we show that strong normalization of the source term with generalized applications is equivalent to the strong normalization of the target term with explicit substitutions, and hence also of the CbN λ-calculus.

As a final contribution, we compare λJ_n-strong normalization to that of the original ΛJ in section 4.6. Indeed, we wish to give a quantitatively compatible calculus with generalized applications, but without losing semantical properties of the original calculus. We extract new results for the latter, as a faithful translation to ES, and a new normalizing strategy. Moreover, we obtain a quantitative characterization of ΛJ-strong normalization, where the size of type derivations bound the number of β-steps, but not π-steps.

1.3 Technical Preliminaries

We finish this introduction by giving first some standard definitions and the notations we use concerning the λ-calculus, explicit substitutions and rewriting. A formal introduction to quantitative type systems follows, where we detail the proof method that will be used recurrently in this work.

1.3.1 Calculi

Syntax. The syntax of the λ-calculus is defined inductively by the following grammar. The set of terms of the λ-calculus is named $\mathtt{T}_{\Lambda}$. From now on, we use uppercase letters for λ-terms to distinguish them from the terms of the calculi with node replication or generalized

applications. The set $\mathscr{V}$ is a countably infinite set of variables $x, y, z, \ldots$

$$\begin{array}{rl} \textbf{(Terms)} & M, N, P, Q ::= V \mid MN \\ \textbf{(Values)} & V ::= x \in \mathscr{V} \mid \lambda x.M \end{array}$$

The syntax of the λ-calculus with explicit substitution extends $\mathtt{T}_\Lambda$ with one constructor for explicit substitutions $[x/N]$. The set of terms is called $\mathtt{T}_{ES}$.

$$\textbf{(Terms)} \quad M, N, P, Q ::= \ldots \mid M[x/N]$$

Free and **bound** variables $\mathrm{fv}(M)$/$\mathrm{bv}(M)$ of a term M are defined as follows. A term M is **closed** if $\mathrm{fv}(M) = \emptyset$.

$$\begin{array}{rclcrcl} \mathrm{fv}(x) & = & \{x\} & & \mathrm{bv}(x) & = & \emptyset \\ \mathrm{fv}(\lambda x.M) & = & \mathrm{fv}(M) \setminus x & & \mathrm{bv}(\lambda x.M) & = & \mathrm{bv}(M) \cup \{x\} \\ \mathrm{fv}(MN) & = & \mathrm{fv}(M) \cup \mathrm{fv}(N) & & \mathrm{bv}(MN) & = & \mathrm{bv}(M) \cup \mathrm{bv}(N) \\ \mathrm{fv}(M[x/N]) & = & (\mathrm{fv}(M) \setminus x) \cup \mathrm{fv}(N) & & \mathrm{bv}(M[x/N]) & = & \mathrm{bv}(M) \cup \{x\} \cup \mathrm{bv}(N) \end{array}$$

Terms are implicitly equated by α-**conversion** $=_\alpha$:

$$\begin{array}{rll} \lambda x.M =_\alpha \lambda y.M\{x/y\} & & (y \notin \mathrm{fv}(M)) \\ M[x/N] =_\alpha M\{x/y\}[y/N] & & (y \notin \mathrm{fv}(M)) \end{array}$$

We use Barendregt's convention [Bar84], that the names of bound and free variables are assumed different. The *capture-avoiding* **meta-level substitution** $M\{x/N\}$ is defined by induction on terms and always defined, using α-conversion when necessary.

$$\begin{array}{rll} x\{x/N\} = N & & \\ y\{x/N\} = y & & (x \neq y) \\ (\lambda y.M)\{x/N\} = \lambda y.M\{x/N\} & & (x \neq y \text{ and } y \notin \mathrm{fv}(N)) \\ (MP)\{x/N\} = M\{x/N\}P\{x/N\} & & \\ M[y/P]\{x/N\} = M\{x/N\}[y/P\{x/N\}] & & (x \neq y \text{ and } y \notin fvN) \end{array}$$

Notice in particular that $(\lambda x.M)\{x/N\} = \lambda x.M$ and $M[x/P]\{x/N\} = M[x/P\{x/N\}]$.

We define the **size** $|M|$ of a term M as

$$|x| = 1 \qquad |\lambda x.M| = 1 + |M| \qquad |MN| = 1 + |M| + |N| \qquad |M[x/N]| = 1 + |M| + |N|$$

The number of free occurrences $|M|_x$ of a variable x in a term M is defined as follows. In the second, third and last cases, we suppose $x \neq y$.

$$|x|_x = 1 \qquad |y|_x = 0 \qquad |\lambda y.M|_x = |M|_x \qquad |MN|_x = |M|_x + |N|_x \qquad |M[y/N]|_x = |M|_x + |N|_x$$

$$\begin{aligned}
\textbf{(Full contexts)} \quad & \mathtt{C} ::= \Diamond \mid \lambda x.\mathtt{C} \mid \mathtt{C}N \mid M\mathtt{C} \\
\textbf{(Head contexts)} \quad & \mathtt{H} ::= \Diamond \mid \lambda x.\mathtt{H} \mid \mathtt{H}N \\
\textbf{(Weak-head contexts)} \quad & \mathtt{W} ::= \Diamond \mid \mathtt{W}N
\end{aligned}$$

Figure 1.9: Contexts for the λ-calculus.

Semantics. We denote a **reduction rule** r from terms to terms with $\mapsto_{\mathrm{r}}$. **Reduction relations** $\mathscr{R}$ are denoted with $\to_{\mathscr{R}}$. The reflexive-transitive closure of a reduction is denoted $\to^{*}_{\mathscr{R}}$, the transitive closure $\to^{+}_{\mathscr{R}}$. If a term M reduces to N in n steps, we write $M \to^{n}_{\mathscr{R}} N$. If M is strongly normalizing, we write $\|M\|_{\mathscr{R}}$ for the length of the longest reduction sequence starting at M. We write $\mathrm{SN}(\mathscr{R})$ the set of strongly normalizing terms on $\mathscr{R}$.

Reduction relations can be generated from a set of reduction rules in two ways. The first option is to use **contexts**. A context is is a special term with one hole $\Diamond$.[6] The **application** of a context to a term $\mathtt{C}\langle M\rangle$ denotes the syntactic replacement of the hole of $\mathtt{C}$ by the term M. Note that capture of variables may occur, for instance: $(\lambda x.\Diamond)\langle x\rangle = \lambda x.x$. We use the notation $\mathtt{C}\langle\!\langle M\rangle\!\rangle$ to indicate that no variable is captured. In that case, we may need to use α-equivalence. For example, $(\lambda x.\Diamond)\langle\!\langle x\rangle\!\rangle =_{\alpha} (\lambda y.\Diamond)\langle\!\langle x\rangle\!\rangle = \lambda y.x$.

For the λ-calculus, the general reduction relation β is defined as the closure of $\mapsto_{\beta}$ under all (full) contexts $\mathtt{C}$, given in figure 1.9. For a given reduction rule r, the reduction relation $\to_{\mathrm{r}}$ is defined as the closure of $\mapsto_{\mathrm{r}}$ under all contexts. Specific reductions have their own name. For instance, the head reduction relation $\to_{\mathrm{h}}$ is defined as the closure of $\mapsto_{\beta}$ under contexts $\mathtt{H}$, and the weak-head relation $\to_{\mathrm{whr}}$ as the closure under contexts $\mathtt{W}$, also defined in figure 1.9.

Example 1.2. Let $M = (\lambda x.\mathtt{I}x)\mathtt{I}N$. Then the reduction $M \to_{\beta} \mathtt{I}\mathtt{I}N$ is a valid weak-head reduction because $\mathtt{W} = \Diamond N$ is a weak-head context, so that the redex $(\lambda x.\mathtt{I}x)\mathtt{I}$ is directly surrounded by a weak-head context. On the contrary, the reduction $M \to_{\beta} (\lambda x.x)\mathtt{I}N$ is not a weak-head one because $\mathtt{H} = (\lambda x.\Diamond)\mathtt{I}N$ is only a head context.

The second way to detail a reduction relation is to use inference rules. This gives more flexibility and is useful in particular to give deterministic strategies. For instance, a deterministic head reduction for the λ-calculus can be given as follows.

$$\frac{M \mapsto_{\beta} N}{M \to_{\mathrm{h}} N} \qquad \frac{M \to_{\mathrm{h}} N}{\lambda x.M \to_{\mathrm{h}} \lambda x.N} \qquad \frac{M \to_{\mathrm{h}} M' \qquad M \neq \lambda x.P}{MN \to_{\mathrm{h}} M'N}$$

The set of **normal forms** of a reduction relation $\mathscr{R}$ is denoted $\mathrm{NF}_{\mathscr{R}}$. It often builds on a set of **neutral normal forms** (sometimes shortened to neutral forms) denoted $\mathrm{NE}_{\mathscr{R}}$. The neutrals are normal forms that cannot create a redex when put at the left of a term. For instance, the strong β-normal forms are defined below.

$$\begin{aligned}
\textbf{(Normal forms)} \quad & \mathrm{NF}_{\beta} ::= \mathrm{NE}_{\beta} \mid \lambda x.\,\mathrm{NF}_{\beta} \\
\textbf{(Neutral normal forms)} \quad & \mathrm{NE}_{\beta} ::= x \mid \mathrm{NE}_{\beta}\,\mathrm{NF}_{\beta}
\end{aligned}$$

[6] Contexts with several holes are only considered in section 2.3.2 for the formal definition of a skeleton.

In the next subsection, we present the quantitative type systems using a simple CbN calculus with explicit substitutions called λES. Its operational semantics are based on two rules.

$$\mathrm{L}\langle\lambda x.M\rangle N \longmapsto_{\mathrm{dB}} \mathrm{L}\langle M[x/N]\rangle$$
$$M[x/N] \longmapsto_{\mathrm{sub}} M\{x/N\}$$

The first rule dB uses distance, expressed with a notion of **list contexts** L.

$$\textbf{(List contexts)} \quad \mathrm{L} ::= \Diamond \mid \mathrm{L}[x/N]$$

A list context simply represents a series of ES $[x_1/N_1]\dots[x_N/N_n]$. In a verbose way, dB can be written as:

$$(\lambda x.M)[x_1/N_1]\dots[x_n/N_n]N \longrightarrow_{\mathrm{dB}} M[x/N][x_1/N_1]\dots[x_n/N_n]$$

We define the **domain** of a list context $\mathrm{L} = [x_1/N_1]\dots[x_1/N_n]$ as $\mathrm{dom}(\mathrm{L}) = \{x_1, \dots, x_n\}$.

1.3.2 Quantitative Types

The simple type system of the λ-calculus is the unique interpretation of minimal intuitionistic logic as types. On the contrary, intersection type systems are multiple. As they strive for the equivalence between normalization and typability, there are indeed as many type systems as there are (equivalent) notions of normalization.

In the next three sections, although only the calculus λES is used, we define three *quantitative* type systems. They correspond in order to CbN head, weak and strong normalization.

Each time, we will start by recalling the definition of the reduction we consider (that can be found in the literature), then give the type system and finally prove and discuss consequences. In the rest of the thesis, we will relate the type systems introduced to these ones. The considered notions of normalization in ES are equivalent to the corresponding ones in the λ-calculus. Therefore, choosing to equate normalization in our calculi to normalization in calculi with ES or the λ-calculus is equivalent.

Each of these notions correspond to the ones of the λ-calculus, so that equating normalization of our calculi to normalization in calculi with ES also equates it to normalization in the λ-calculus.

The grammar of types and multiset types is common to all the CbN systems (with only an added type constant for the weak systems), and reads as follows.

Definition 1.3 (Type grammar). Let an infinite set BTV of base type variables $a, b, c \dots$

$$\textbf{(Types)} \quad \sigma, \tau, \rho ::= a \in BTV \mid \mathscr{M} \longrightarrow \sigma$$
$$\textbf{(Multiset types)} \quad \mathscr{M}, \mathscr{N} ::= [\sigma_i]_{i\in I} \text{ where } I \text{ is a finite set}$$

Multiset types will also be called *multitypes*. The **empty multiset** is a valid multitype and is denoted $[\,]$. The number of elements of a multitype $\mathscr{M}$ is given by $|\mathscr{M}|$. The union of

multitypes $\mathscr{M}$ and $\mathscr{N}$ is denoted $\mathscr{M} \sqcup \mathscr{N}$. **Typing environments** $\Gamma, \Delta, \Lambda, \Pi$ are functions from variables to multiset types assigning the empty multiset to all but a finite set of variables. The **domain** of Γ is given by $\mathrm{dom}(\Gamma) \coloneqq \{x \mid \Gamma(x) \neq [\,]\}$. The **union of environments**, written $\Gamma \uplus \Delta$, is defined by $(\Gamma \uplus \Delta)(x) \coloneqq \Gamma(x) \sqcup \Delta(x)$, where $\sqcup$ denotes multiset union. This notion is extended to several environments as expected, so that $\uplus_{i \in I} \Gamma_i$ denotes a finite union of environments ($\uplus_{i \in I} \Gamma_i$ is to be understood as the empty environment $\emptyset$ when $I = \emptyset$). We write $\Gamma \setminus x$ for the environment such that $(\Gamma \setminus x)(y) = \Gamma(y)$ if $y \neq x$ and $(\Gamma \setminus x)(x) = [\,]$. We write $\Gamma; \Delta$ for $\Gamma \uplus \Delta$ when $\mathrm{dom}(\Gamma) \cap \mathrm{dom}(\Delta) = \emptyset$. The **typing** of a term in a sequent is a pair (environment, type).

1.3.2.1 Head Normalization

Head reduction. Starting with the head type system is natural: it is the simplest and oldest [CDV81] form of intersection type systems, upon which the others are built by slight alterations.

The formulation of head reduction for λES is taken from Bucciarelli, Kesner, Ríos, and Viso [Buc+20]. It uses a definition of head contexts $\mathtt{H}$ generalized to explicit substitutions.

Definition 1.4 (Head contexts for λES). $\mathtt{H} ::= \Diamond \mid \lambda x.\mathtt{H} \mid \mathtt{H}N \mid \mathtt{H}[x/N]$

The relation $\to_{\mathrm{hes}}$ is defined as the closure of rules dB and sub under contexts $\mathtt{H}$. The reductions modeled by quantitative types do not need to be deterministic, since quantitative types are a semantical tool, as long as reduction is confluent.

Normal forms are crucial in the completeness part of the proof (typable *implies* normalizable), which relies on their typability. They are the same head normal forms as in the λ-calculus.

Definition 1.5 (hes-normal forms).

$$\begin{aligned} &\textbf{(Neutral normal forms)} && \mathrm{NE}_{\mathrm{hes}} ::= x \mid \mathrm{NE}_{\mathrm{hes}}\, N \\ &\textbf{(Normal forms)} && \mathrm{NF}_{\mathrm{hes}} ::= \mathrm{NE}_{\mathrm{hes}} \mid \lambda x.\, \mathrm{NF}_{\mathrm{hes}} \end{aligned}$$

The type system. We introduce a type system $\mathscr{H}$, adapted from the original system [Gar94] by Kesner and Ventura [KV14]. Our presentation differs a little bit, because we take (MANY) as a separate rule, but this is only a stylistic matter.

Definition 1.6 (Head quantitative type system $\mathscr{H}$ for λES).

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (\mathrm{AX}) \qquad \frac{(\Gamma_i \vdash M : \sigma_i)}{\uplus_{i \in I} \Gamma_i \vdash M : [\sigma_i]_{i \in I}}\ (\mathrm{MANY}) \qquad \frac{\Gamma; x : \mathscr{M} \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \mathscr{M} \to \sigma}\ (\to_i)$$

$$\frac{\Gamma \vdash M : \mathscr{M} \to \sigma \qquad \Delta \vdash N : \mathscr{M}}{\Gamma \uplus \Delta \vdash MN : \sigma}\ (\to_e) \qquad \frac{\Gamma; x : \mathscr{M} \vdash M : \sigma \qquad \Delta \vdash N : \mathscr{M}}{\Gamma \uplus \Delta \vdash [N/x]M : \sigma}\ (\mathrm{ES})$$

Type derivations are named Φ or Ψ. For each system, we define a notion of size of a proof $\mathrm{sz}(\Phi)$. A derivation of the sequent $\Gamma \vdash M : \tau$ of size n in a system $\mathcal{S}$ is denoted

$\Gamma \Vdash^{n}_{\mathscr{H}} M : \sigma$. We sometimes omit the size when not relevant, or the type system when clear from the context and write $\Gamma \Vdash t : \tau$ a derivation of the sequent.

In this system, we have five rules: one for each constructor, and an auxiliary one called (MANY). Every rule apart from (MANY) infers a type σ. However, multi-types can appear inside derivations from the right premises of the inference rules for application ($\to_e$) and explicit substitutions (ES), which is where (MANY) is necessary. It would be equivalent to define a system with (MANY) embedded in these rules.

Although we can derive multitypes, a term $M \in \mathtt{T}_{ES}$ is said to be **typable** if and only if there is a derivation $\Gamma \Vdash M : \sigma$. Indeed, all terms are typable with the empty multitype in a CbN quantitative type system, so that a definition of typability considering both types and multitypes would be degenerate. This definition of typability will be valid for all kinds of CbN type systems, for all the calculi.

Example 1.7. The term $\lambda x.xx$ can be typed with the following derivation.

$$\dfrac{\dfrac{\dfrac{}{x : [[\sigma] \to \sigma] \vdash x : [\sigma] \to \sigma}\,(\text{AX}) \qquad \dfrac{\dfrac{}{x : [\sigma] \vdash x : \sigma}\,(\text{AX})}{x : [\sigma] \vdash x : [\sigma]}\,(\text{MANY})}{x : [[\sigma] \to \sigma, \sigma] \vdash xx : \sigma}\,(\to_e)}{\emptyset \vdash \lambda x.xx : [[\sigma] \to \sigma, \sigma] \to \sigma}\,(\to_i)$$

Since the set of simply typed terms is strictly contained in the set of normalizable terms, some terms can be typed with intersection types but not simple types. An example is given by the previous derivation. The term $\delta = \lambda x.xx$ is a normal form and thus trivially normalizing, but not simply typable. However, the looping term $\Omega = \delta\delta$ cannot be intersection-typed.

Terms typed with a multitype are the ones that can be duplicated or erased. Morally, each type of the multiset corresponds to one "use" of the term. Then, some subterms that can be erased along reduction may be left untyped, that is, typed with an empty intersection. The following derivation is possible because the subterm Ω is typed with an empty multitype.

$$\dfrac{\dfrac{\dfrac{}{x : [\sigma] \vdash x : \sigma}\,(\text{AX})}{x : [\sigma] \vdash \lambda z.x : [\,] \to \sigma}\,(\to_i) \qquad \dfrac{}{\emptyset \vdash \Omega : [\,]}\,(\text{MANY})}{x : [\sigma] \vdash (\lambda z.x)\Omega : \sigma}\,(\to_e)$$

Notice that in the axiom rule, the environment contains only the type of the variable. The system, and all those that we consider, is indeed *relevant*.

Property 1.8. *Let* $\Gamma \Vdash M : \sigma$. *Then* $\mathrm{dom}(\Gamma) \subseteq \mathrm{fv}(M)$.

Characterization. We now detail the lemmas necessary to prove correctness of the system. The general method transports to the different relations and calculi, with subtleties. The theorem that we want to prove is the following.

Theorem 1.9. *Let* $M \in \mathtt{T}_{ES}$. M *is typable in* $\mathscr{H} \iff M$ *is* hes*-normalizable.*

This theorem relies on two key lemmas, one for each direction of the implication.

1. *Weighted subject reduction.* Subject reduction states that the typing of a term is preserved along reduction, as is usual. This is often seen as a sanity property of type systems: changes in the type of subterms of a program are likely to cause incoherences, and thus bugs. The particularity of weighted subject reduction is that the size of a typing derivation reduces at each step. This gives normalization of typable terms as a direct corollary.

2. *Subject expansion* is the opposite of subject reduction: if M reduces to a typable term M' and M' is typable, then M is also typable with the same typing as M'. Attaching quantitative information to the subject expansion lemma is also possible, proving that the size of the reduced term is smaller than the one of the reducible one, but we do not do it.

Subject reduction and expansion themselves rely on two dual lemmas: substitution and anti-substitution. The first one builds a derivation for a meta-level substitution, while the second one dissociates two derivations entangled by a substitution.

Lemma 1.10 (Substitution for $\mathscr{H}$). *If* $\Gamma; x : \mathscr{M} \Vdash^{n}_{\mathscr{H}} M : \sigma$ *and* $\Delta \Vdash^{m}_{\mathscr{H}} N : \mathscr{M}$*, then there exists* $\Gamma \uplus \Delta \Vdash^{m+n}_{\mathscr{H}} M\{x/N\} : \sigma$.

Lemma 1.11 (Anti-substitution for $\mathscr{H}$). *If* $\Gamma \Vdash M\{x/N\} : \sigma$*, then there exists* Γ_M, Γ_N *and* $\mathscr{M}$ *such that* $\Gamma_M; x : \mathscr{M} \Vdash M : \sigma, \Gamma_N \Vdash N : \mathscr{M}$ *and* $\Gamma = \Gamma_M \uplus \Gamma_N$.

Both proofs are by induction on M.

Lemma 1.12 (Weighted subject reduction for $\mathscr{H}$). *If* $\Gamma \Vdash^{n_1}_{\mathscr{H}} M_1 : \sigma$ *and* $M_1 \to_{\text{hes}} M_2$*, then* $\Gamma \Vdash^{n_2}_{\mathscr{H}} M_2 : \sigma$ *with* $n_1 > n_2$.

We do not give a proof of this statement, but an example of the decreasing of the size of a proof.

Example 1.13. The first proof Φ_1 has size 3, since the applications of (MANY) are not counted. In the reduction, the rules $(\to_i)$ and $(\to_e)$ are erased, and replaced by a single rule (ES), so that $\text{sz}(\Phi_2) = 2$. Finally, the rule (ES) is erased in the last step, so that $\text{sz}(\Phi_3) = 1$.

$$\Phi_1 = \dfrac{\dfrac{\dfrac{}{x : [\sigma] \vdash x : \sigma}\,(\text{AX})}{x : [\sigma] \vdash \lambda z.x : [\,] \to \sigma}\,(\to_i) \qquad \dfrac{}{\emptyset \vdash y : [\,]}\,(\text{MANY})}{x : [\sigma] \vdash (\lambda z.x)y : \sigma}\,(\to_e)$$

$$\to_{\text{dB}} \quad \Phi_2 = \dfrac{\dfrac{}{x : [\sigma] \vdash x : \sigma}\,(\text{AX}) \qquad \dfrac{}{\emptyset \vdash y : [\,]}\,(\text{MANY})}{x : [\sigma] \vdash x[z/y] : \sigma}\,(\text{ES}) \quad \to_{\text{dB}} \quad \Phi_3 = \dfrac{}{x : [\sigma] \vdash x : \sigma}\,(\text{AX})$$

Type derivations become smaller along reduction. Then, the size of reduction from a typed term M itself cannot be bigger than the size of its smallest proof derivation. Therefore, all typable terms normalize: they have a finite reduction length bounded by the size of type derivations. This method of proving normalization only adds a small amount of effort to subject reduction, and avoids more involved techniques such as Tait's reducibility candidates ([Tai67]). Such combinatorial proofs are not available with idempotent intersection types, where usual methods are used.

Lemma 1.14 (Subject expansion for $\mathscr{H}$). *If* $\Gamma \Vdash_{\mathscr{H}} M_2 : \sigma$ *and* $M_1 \to_{\text{hes}} M_2$, *then* $\Gamma \Vdash_{\mathscr{H}} M_1 : \sigma$.

Subject expansion is not a usual property of type systems. Indeed, one generally considers typing as a guarantee of good behavior of programs along reduction only.[7] In the framework of intersection types however, we want completeness of the type system because we consider typings of terms as their model, which should be the same for two convertible terms.

Example 1.15. Subject expansion holds for the derivation $(\lambda z.x)y \to_{\text{dB}} x[z/y] \to_{\text{dB}} x$. From the derivation Φ_3 of example 1.13, we can derive Φ_2, with the same typing $(x : [\sigma], \sigma)$, and then Φ_3, still with the same typing.

However, in the simply typed system, a derivation for $x[z/y]$ or $(\lambda x.z)y$ must contain a type for the variable y in the environment:

$$\dfrac{\dfrac{}{x : A, y : B \vdash x : A}\,(\text{AX}) \qquad \dfrac{}{x : A, y : B \vdash y : B}\,(\text{AX})}{x : A, y : B \vdash x[z/y] : A}\,(\text{ES})$$

Starting from a derivation of x with typing $(x : A, A)$, we cannot find a derivation of $x[z/y]$ with the same typing, so that subject expansion fails. Still, subject reduction does hold, thanks to weakening: the derivation $\dfrac{}{x : A, y : B \vdash x : A}$ is valid.

To conclude the proof of completeness, we need to prove that hes-normal forms are typable. The proof is by induction on NE_{hes}, then on NF_{hes}.

Lemma 1.16 (Typing hes-nfs). *Let* $M \in \text{NF}_{\text{hes}}$. *Then there exists* Γ, σ *such that* $\Gamma \Vdash_{\mathscr{H}} M : \sigma$.

As we know that all normal forms are typable, we know by an arbitrary finite number of applications of subject expansion that all terms having a normal form are typable.

Putting everything together, theorem 1.9 follows from subject reduction in one direction and subject expansion and typability of normal forms in the other direction. This theorem relates normalization and typability in a qualitative manner. But thanks to non-idempotence, it can be extended with a quantitative property already evoked.

Theorem 1.17. *Let* $M \in \Lambda_{ES}$. M *is typable in* $\mathscr{H}$ *with a derivation of size* $n \iff$ *the size of the longest* hes *reduction sequence starting at* M *is bounded by* n.

[7] Expansion of terms is sometimes used during compilation; in this case, conservation of typing is checked *ad hoc* for the expansions considered.

The size of type derivations is also an upper bound on the size of normal forms, for an appropriate notion of size, depending on the reduction considered. The size of head normal forms for instance, does not include the size of arguments, so that xx and $x(xxx)$ are considered of the same size.

From the type system, we can build a *relational model* of the terms, that is, a model in the category of sets and relations of terms [BEM07]. The interpretation of a term is given by its set of possible typings (type environment, type).

1.3.2.2 Weak-head Normalization

We now detail weak-head reduction. This reduction is of particular interest to us because we are interested in λ-calculi as foundations of programming languages.

Weak-head reduction $\to_{\text{whes}}$ is defined as the reduction of dB and sub under weak-head contexts $\mathtt{W}$. The difference with head contexts is the absence of abstractions.

$$\textbf{(Weak-head contexts)} \quad \mathtt{W} ::= \Diamond \mid \mathtt{W}N \mid \mathtt{W}[x/N]$$

Definition 1.18. Normal forms are characterized by the following grammar.

$$\begin{aligned} \textbf{(Neutral normal forms)} \quad & \mathrm{NE}_{\text{hes}} ::= x \mid \mathrm{NE}_{\text{whes}}\, N \\ \textbf{(Normal forms)} \quad & \mathrm{NF}_{\text{hes}} ::= \mathrm{NE}_{\text{whes}}\, N \mid \lambda x.M \end{aligned}$$

A type system for weak-head reduction in CbN λES appears in [Kes16]. We first need to extend types with a constant $\mathtt{a}$ (for answer).

$$\textbf{(Weak-head types)} \quad \sigma, \tau, \delta, \rho ::= a \in BTV \mid \mathscr{M} \to \sigma \mid \mathtt{a}$$

The constant $\mathtt{a}$ types abstractions whose body will not be affected by reduction, that is, abstractions which will not be used on the left of a weak-head dB-redex. The following type system $\mathscr{W}$ is the same as $\mathscr{H}$, with one added rule to type any abstraction as an answer.

Definition 1.19 (Weak-head quantitative type system $\mathscr{W}$ for λES).

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (\text{AX}) \qquad \frac{(\Gamma_i \vdash M : \sigma_i)}{\uplus_{i\in I}\Gamma_i \vdash M : [\sigma_i]_{i\in I}}\ (\text{MANY})$$

$$\frac{}{\emptyset \vdash \lambda x.M : \mathtt{a}}\ (\text{ANS}) \qquad \frac{\Gamma; x : \mathscr{M} \vdash M : \sigma}{\Gamma \vdash \lambda x.M : \mathscr{M} \to \sigma}\ (\to_i)$$

$$\frac{\Gamma \vdash M : \mathscr{M} \to \sigma \qquad \Delta \vdash N : \mathscr{M}}{\Gamma \uplus \Delta \vdash MN : \sigma}\ (\to_e) \qquad \frac{\Gamma; x : \mathscr{M} \vdash M : \sigma \qquad \Delta \vdash N : \mathscr{M}}{\Gamma \uplus \Delta \vdash [N/x]M : \sigma}\ (\text{ES})$$

The possibility of typing every abstraction with a constant means that every abstraction is normalizable. Indeed, in the weak paradigm, every abstraction is considered a result, and thus a normal form.

Example 1.20. The term $\lambda x.\Omega$ is weak-head, but not head normalizable, and typable in $\mathscr{W}$ but not $\mathscr{H}$. Indeed, in $\mathscr{H}$, the content of an abstraction must be typable. In $\mathscr{W}$ instead, the whole abstraction can be typed with $\mathtt{a}$.

$$\frac{}{\emptyset \vdash \lambda x.\Omega \ : \ \mathtt{a}} \ (\textsc{ans})$$

The system is again relevant: in any sequent $\Gamma \vdash M \ : \ \sigma$, we have $\mathrm{dom}(\Gamma) \subseteq \mathrm{fv}(M)$. To prove that the type system is sound and complete for the relation $\to_{\mathrm{whes}}$, we adapt the lemmas of section 1.3.2.1 to the type system $\mathscr{H}$ and reduction $\to_{\mathrm{whes}}$: (anti-)substitution, weighted subject reduction, subject expansion and typability of whes-nfs.

Theorem 1.21. *Let $M \in \Lambda_{ES}$. M is typable in $\mathscr{W}$ with a derivation of size n $\iff$ M whes-normalizes in less than n steps.*

1.3.2.3 Strong Normalization

The last type system of the section characterizes strong normalization [BL13; BR13]. In other words, we must now make sure that every subterm normalizes. For instance, the term $(\lambda z.x)\Omega$ is not strongly normalizable, since we can keep reducing Ω forever, even though we can also erase it.

We write $\to_{\mathrm{es}}$ the full reduction in λES. The choice of reducing everything before it can be erased corresponds to the *perpetual strategy*, which is an operational characterization of strong normalization: for a term to normalize in the perpetual strategy, it must be strongly normalizable, that is having no infinite reduction sequence.

The fact that every subterm must be normalizable is specific to strong normalization. For instance, the head normalizable term $(\lambda z.x)\Omega$ was typed in system $\mathscr{H}$ by assigning the empty multitype to Ω. In the strong type system, we can still assign the empty multitype to a term, but must in addition show that this terms is typable. To do this, we use a **choice function** on multitypes, as in [KV20].

$$\#(\mathscr{M}) = \begin{cases} \mathscr{M}, & \text{if } \mathscr{M} \neq [\,] \\ [\sigma], & \text{otherwise, for an arbitrary } \sigma. \end{cases}$$

We use this choice operator in the rules where a premise must derive a multitype. The condition that $I \neq \emptyset$ in rule many is not necessary, but we add it to emphasize the idea that we cannot type subterms with the empty multiset.

Definition 1.22 (Strong quantitative type system $\cap ES$ for λES).

$$\frac{}{x \ : \ [\sigma] \vdash x \ : \ \sigma} \ (\textsc{ax}) \qquad \frac{(\Gamma_i \vdash M \ : \ \sigma_i) \qquad I \neq \emptyset}{\uplus_{i\in I}\Gamma_i \vdash M \ : \ [\sigma_i]_{i\in I}} \ (\textsc{many}) \qquad \frac{\Gamma; x \ : \ \mathscr{M} \vdash M \ : \ \sigma}{\Gamma \vdash \lambda x.M \ : \ \mathscr{M} \to \sigma} \ (\to_i)$$

$$\frac{\Gamma \vdash M \ : \ \mathscr{M} \to \sigma \qquad \Delta \vdash N \ : \ \#(\mathscr{M})}{\Gamma \uplus \Delta \vdash MN \ : \ \sigma} \ (\to_e) \qquad \frac{\Gamma; x \ : \ \mathscr{M} \vdash M \ : \ \sigma \qquad \Delta \vdash N \ : \ \#(\mathscr{M})}{\Gamma \uplus \Delta \vdash [N/x]M \ : \ \sigma} \ (\textsc{es})$$

Examples 1.23. The following terms show how an erasable argument can induce a difference in typing.

- $(\lambda z.x)\Omega$ cannot be typed, since we cannot provide a witness derivation for Ω as a premise of rule .
- A type derivation is given for the term $(\lambda z.x)y$ by providing a witness τ for y.

$$\dfrac{\dfrac{\dfrac{}{x : [\sigma] \vdash x : \sigma}\,(\text{AX})}{x : [\sigma] \vdash \lambda z.x : [\,] \to \sigma}\,(\to_e) \qquad \dfrac{\dfrac{}{y : [\tau] \vdash y : \tau}\,(\text{AX})}{y : [\tau] \vdash y : [\tau]}\,(\text{MANY})}{x : [\sigma], y : [\tau] \vdash (\lambda z.x)y : \sigma}$$

Relevance in the head and weak-head type systems were given by $\mathrm{dom}(\Gamma) \subseteq \mathrm{fv}(M)$ for a derivation $\Gamma \Vdash M : \sigma$. Interestingly, since all subterms are typed, here we have an equality.

Property 1.24 (Relevance). *Let* $\Gamma \Vdash M : \sigma$. *Then* $\mathrm{dom}(\Gamma) = \mathrm{fv}(M)$.

Naturally, the expected characterization of strong normalization holds, albeit more difficult to show. Here is why: weighted subject reduction and subject expansion hold only partially. The last example demonstrates a failure case. The free variable y is present in the typing environment with a non-empty multiset type. Reducing this terms to $x[z/y]$ then x deletes y, which should not appear anymore in the environment, following the relevance property. Subject reduction is thus not satisfied since typing can be modified (in fact only by removal) by so-called *erasing steps*.

Definition 1.25 (Erasing step). An erasing step in λES is a sub-step $M[x/N] \to_{\mathrm{sub}} M$, where $x \notin \mathrm{fv}(M)$.

The substitution and anti-substitution lemmas still hold, but with the condition that $x \in \mathrm{fv}(M)$.

Lemma 1.26. *Let* $M, N \in \mathtt{T}_{ES}$.

Substitution Lemma *If* $\Gamma; x : \mathscr{M} \Vdash^{n}_{\cap ES} M : \sigma$ *with* $x \in \mathrm{fv}(M)$ *and* $\Delta \Vdash^{m}_{\cap ES} N : \mathscr{M}$, *then there exists* $\Gamma \uplus \Delta \Vdash^{m+n}_{\cap ES} M\{x/N\} : \sigma$.

Anti-substitution Lemma *If* $\Gamma \Vdash M\{x/N\} : \sigma$ *with* $x \in \mathrm{fv}(M)$, *then there exists* Γ_M, Γ_N *and* $\mathscr{M}$ *such that* $\Gamma_M; x : \mathscr{M} \Vdash M : \sigma$, $\Gamma_N \Vdash N : \mathscr{M}$ *and* $\Gamma = \Gamma_M \uplus \Gamma_N$.

We can prove weighted subject reduction and expansion *for non-erasing steps*.

Lemma 1.27. *Let* $M_1, M_2 \in \mathtt{T}_{ES}$ *and* $M_1 \to_{\mathrm{es}} M_2$ *be a non-erasing step.*

Weighted Subject Reduction for non-erasing steps *If* $\Gamma \Vdash^{n_1}_{\cap ES} M_1 : \sigma$, *then* $\Gamma \Vdash^{n_2}_{\cap ES} M_2 : \sigma$ *with* $n_1 > n_2$.

Subject Expansion for non-erasing steps *If* $\Gamma \Vdash_{\cap ES} M_2 : \sigma$, *then* $\Gamma \Vdash_{\cap ES} M_1 : \sigma$.

A possible variation is to abandon relevance, so that we can prove subject reduction for every step, even the erasing ones. *Weakening* is added to the system by changing the conclusion of the axiom rule to $\Gamma \uplus x : [\sigma] \Vdash x : \sigma$. But doing so, we do not retrieve subject expansion.

The property that full normal forms are typable is valid. As all subterms are typed, the size of the type derivation is a bound on the size of the term, as defined in section 1.3.

Lemma 1.28 (Typing es-nfs). *Let $M \in \mathrm{NF}_{\mathrm{es}}$. Then there exists Γ, σ and n such that $\Gamma \Vdash^{n}_{\cap ES} M : \sigma$ and $|M| \leq n$.*

The proof of the characterization theorem must be completed by some inductive reasoning on type derivations. A proof for the strong quantitative type system $\cap J$ for the CbN calculus with generalized applications is in section 4.4. The size of the type derivation decreases at each step, even erasing ones, and the characterization theorem is still quantitative. So, while subject reduction and expansion do not hold for every step, the crucial characterization still holds.

Theorem 1.29. *Let $M \in \mathtt{T}_{ES}$. M is typable in $\cap ES$ with a derivation of size n $\iff$ the size of the longest* es *reduction sequence starting at M plus the size of the (unique)* es*-nf is bounded by n.*

CHAPTER 2

Node replication

In this chapter, we introduce the theory of node replication through the calculus λR. The first section describes the syntax (section 2.1.1) and operational semantics (section 2.1.2) of the calculus. It ends with the definition of the notion of levels (section 2.1.3), a crucial notion to prove termination of the substitution rules and to give a decreasing measure on type derivations.

Section 2.2 gives some general properties of the calculus: termination of substitution (section 2.2), a simulation between λR and the λ-calculus (section 2.2) –from which an indirect simulation to and from the atomic λ-calculus follows–, and finally confluence (section 2.2).

Section 2.3 introduces the CbN (section 2.3.1) and CbNeed (section 2.3.2) strategies, implementing full and fully lazy substitution respectively. The section starts with the definition of restricted syntax of terms. To simplify the presentation, we indeed implement substitutions on λ-terms rather than on the full syntax with explicit substitutions. We give two implementations of full laziness, as a big-steps (section 2.3.2) and a small-steps (section 2.3.2) semantics. The section ends with the fully lazy call-by-need strategy (section 2.3.2).

The final technical section (section 2.4) introduces a quantitative type system $\cap R$ for the CbN and fully lazy CbNeed strategies. We prove that the type system captures both CbN and CbNeed normalization in section 2.5. From this result, we deduce the equivalence of usual and fully lazy substitution (theorem 2.64).

2.1 A Calculus for Node Replication

We present the λR-calculus (as in Replication). From a syntactical point of view, we add two new constructors to the λ-calculus: explicit substitution and explicit distributors. From an operational point of view, we provide a rewriting system on λR-terms together with a notion of *levels* which will play a key role in the next sections.

2.1.1 Syntax

Given a countably infinite set of variables $x, y, z, ...$, we consider the following grammars.

(Terms)	t, u, r, s	$::=$	$x \mid \lambda x.t \mid tu \mid t[x/u] \mid t[x/\!\!/\lambda y.u]$
(Pure Terms)	p, q	$::=$	$x \mid \lambda x.p \mid pq$
(Term Contexts)	$\mathtt{C}$	$::=$	$\Diamond \mid \lambda x.\mathtt{C} \mid \mathtt{C}t \mid t\mathtt{C} \mid \mathtt{C}[x/t] \mid \mathtt{C}[x/\!\!/\lambda y.u] \mid t[x/\mathtt{C}] \mid t[x/\!\!/\lambda y.\mathtt{C}]$
(List Contexts)	$\mathtt{L}$	$::=$	$\Diamond \mid \mathtt{L}[x/u] \mid \mathtt{L}[x/\!\!/\lambda y.u]$

The set of terms is denoted by $\mathtt{T}_R$ and the subset of **pure** terms is denoted by $\mathtt{T}_P$. This set is isomorphic to the set of λ-terms $\mathtt{T}_\Lambda$, but we use lowercase p and q because it is a subset of $\mathtt{T}_R$.

The construction $[x/u]$ is an explicit substitution. The second construction $[x/\!\!/\lambda y.u]$ is an **explicit distributor** (or simply *distributor*), which is used specifically in the duplication of abstractions.

An **explicit cut** is written $[x \triangleleft u]$, which is either $[x/u]$, or $[x/\!\!/u]$ when u is $\lambda y.u'$, typically to factorize some definitions and proofs where ES and distributors behave similarly. A characterization function $\mathrm{es}([x \triangleleft u])$ on explicit cuts distinguishes these two cases: $\mathrm{es}([x \triangleleft u]) = 1$ if $[x \triangleleft u] = [x/u]$, and 0 otherwise. **Free** and **bound variables**, as well as α-conversion and meta-level substitution is extended to distributors in the same way as ESs.

Two notions of **contexts** are used. Term contexts $\mathtt{C}$ extend those of the λ-calculus to explicit cuts. List contexts $\mathtt{L}$ denote an arbitrary list of explicit cuts. They will be used in particular to implement reduction at a *distance*.

2.1.2 Operational Semantics

The atomic λ-calculus of Gundersen, Heijltjes, and Parigot [GHP13b] uses separate permutation rules to permute explicit substitutions and unblock reductions. Our calculus λR instead relies on a semantics at a distance, integrating permutations into meaningful reduction steps in order to put the focus on node replication mechanisms. The said permutations are the following:

$$\begin{aligned}
\lambda y.t[x \triangleleft u] &\mapsto_\rho (\lambda y.t)[x \triangleleft u] && \text{if } y \notin \mathrm{fv}(u)\\
t[x \triangleleft u]s &\mapsto_\rho (ts)[x \triangleleft u] && \text{if } x \notin \mathrm{fv}(s)\\
ts[x \triangleleft u] &\mapsto_\rho (ts)[x \triangleleft u] && \text{if } x \notin \mathrm{fv}(t)\\
t[x \triangleleft u[y \triangleleft s]] &\mapsto_\rho t[x \triangleleft u][y \triangleleft s] && \text{if } y \notin \mathrm{fv}(t)
\end{aligned}$$

The reduction relation $\longrightarrow_\rho$ is defined as the closure of the four rules $\mapsto_\rho$ under *all* contexts.

Example 2.1. In this reduction, both inner explicit cuts $[z_1/\!\!/\mathtt{I}]$ and $[z_2/z_3]$ are pushed outside the main ES, which results in a pure term followed by a list of explicit cuts.

$$\begin{aligned}
x[x/w[z_1/\!\!/\mathtt{I}](\lambda y.y[z_2/z_3])] &\longrightarrow_\rho x[x/w[z_1/\!\!/\mathtt{I}](\lambda y.y)[z_2/z_3]]\\
&\longrightarrow_\rho x[x/(w[z_1/\!\!/\mathtt{I}](\lambda y.y))[z_2/z_3]]\\
&\longrightarrow_\rho x[x/w[z_1/\!\!/\mathtt{I}](\lambda y.y)][z_2/z_3]\\
&\longrightarrow_\rho x[x/(w(\lambda y.y))[z_1/\!\!/\mathtt{I}]][z_2/z_3]\\
&\longrightarrow_\rho x[x/w(\lambda y.y)][z_1/\!\!/\mathtt{I}][z_2/z_3]
\end{aligned}$$

The **distant reduction relation** $\longrightarrow_{\mathrm{R}}$, is given by the closure under all contexts of the following rules.

$$
\begin{array}{rll}
\mathtt{L}\langle\lambda x.t\rangle u & \mapsto_{\mathrm{dB}} \mathtt{L}\langle t[x/u]\rangle & \\
t[x/\mathtt{L}\langle us\rangle] & \mapsto_{\mathrm{app}} \mathtt{L}\langle t\{x/yz\}[y/u][z/s]\rangle & \text{where } y \text{ and } z \text{ are fresh} \\
t[x/\mathtt{L}\langle\lambda y.u\rangle] & \mapsto_{\mathrm{dist}} \mathtt{L}\langle t[x/\!/\lambda y.z[z/u]]\rangle & \text{where } z \text{ is fresh} \\
t[x/\!/\lambda y.u] & \mapsto_{\mathrm{abs}} \mathtt{L}\langle t\{x/\lambda y.p\}\rangle & \text{where } u \longrightarrow_{\rho}^{*} \mathtt{L}\langle p\rangle \text{ and } y \notin \mathrm{fv}(\mathtt{L}) \\
t[x/\mathtt{L}\langle y\rangle] & \mapsto_{\mathrm{var}} \mathtt{L}\langle t\{x/y\}\rangle &
\end{array}
$$

The λR-**calculus** is defined by the set of terms $\mathtt{T}_R$ equipped with this reduction relation.

In the five rules just above, a list context $\mathtt{L}$ is pushed outside the term. We assume in all these cases that there is no capture of variables caused by this transformation, *e.g.* in rule dB this means that $\mathrm{dom}(\mathtt{L}) \cap \mathrm{fv}(u) = \emptyset$. Apart from the *distant* B*eta* rule dB used to fire β-reduction, there are four substitution rules used to copy nodes of *pure* terms while pushing outside all the cuts surrounding the node to be copied. Rule app copies one application node, while rule var copies one variable node. Notice that the (meta-level and capture-free) substitution is *full*, in the sense that it is performed simultaneously on all occurrences of the free variable x at the same time.

Example 2.2. This example illustrates the use of rules app and var to replicate application and variables nodes, as well as rule dB to fire reduction. No distance is involved in this example.

$$
\begin{aligned}
(\lambda x.xx)(yz) &\longrightarrow_{\mathrm{dB}} (xx)[x/yz] \\
&\longrightarrow_{\mathrm{app}} ((x_1x_2)(x_1x_2))[x_1/y][x_2/z] \\
&\longrightarrow_{\mathrm{var}} (yx_2)(yx_2)[x_2/z] \\
&\longrightarrow_{\mathrm{var}} (yz)(yz)
\end{aligned}
$$

Example 2.3. Replication of abstractions is more involved, as illustrated below. Distance is highlighted in green.

$$
\begin{aligned}
(\lambda x.xx)(\lambda y.(ww)y) &\longrightarrow_{\mathrm{dB}} (xx)[x/\lambda y.(ww)y] && (2.1)\\
&\longrightarrow_{\mathrm{dist}} (xx)[x/\!/\lambda y.z[z/(ww)y]] && (2.2)\\
&\longrightarrow_{\mathrm{app}} (xx)[x/\!/\lambda y.(z_1z_2)[z_1/ww][z_2/y]] && (2.3)\\
&\longrightarrow_{\mathrm{var}} (xx)[x/\!/\lambda y.(z_1y)[z_1/ww]] && (2.4)\\
&\longrightarrow_{\mathrm{app}} (xx)[x/\!/\lambda y.((z_3z_2)y)\,[z_3/w][z_2/w]\,] && (2.5)\\
&\longrightarrow_{\mathrm{abs}} ((\lambda y.(z_3z_2)y)(\lambda y.(z_3z_2)y))[z_3/w][z_2/w] && (2.6)\\
&\longrightarrow_{\mathrm{var}} ((\lambda y.(wz_2)y)(\lambda y.(wz_2)y))[z_2/w] && (2.7)\\
&\longrightarrow_{\mathrm{var}} (\lambda y.(ww)y)(\lambda y.(ww)y) && (2.8)
\end{aligned}
$$

The specificity in copying an abstraction $\lambda y.u$ is due to the (binding) relation between the binder λy and all the free occurrences of y in its body u. Abstractions are thus copied in two

stages. The first one is implemented by the rule dist, which creates a distributor in which a potentially replicable abstraction is placed, while moving its body inside a new ES. Thus, in line (2.2), we create a distributor over the abstraction λy, while $(ww)y$ is placed inside an ES $[z/(ww)y]$. Notice that this substitution is in the scope of abstraction λy. The distributor is marking the fact that the abstraction needs to be further duplicated. There are then two kinds of potentially replicable nodes shared in the body of the corresponding abstraction.

1. All free occurrences of the variable bound by the main abstraction (here λy) must be replicated by means of the rule var (2.4), so as to keep the correct binding structure. This means that all the nodes leading to these occurrences must also be duplicated: this is why rule app is first used in (2.3).

2. All nodes which are neither a free occurrence of the bound variable nor in the path to such a node can be arbitrarily copied inside the distributor (*e.g.* the internal application node in line (2.5)), or replicated later (*e.g.* the two variable nodes w in (2.7) and (2.8)).

Components which are not replicated inside the distributor form a list of explicit cuts, which can occur at different depths inside this distributor. Indeed, in (2.5), there are two ESs $[z_3/w]$ and $[z_2/w]$. The cuts can be gathered together into a list context, called L in the definition of rule abs, which is pushed outside by using permutation rules, before performing the substitution of the pure body containing all the bound occurrences of y (here $\lambda y.(z_1 z_2)y$). This operation is in general hard to specify using only distance since the cuts can appear at arbitrary depth in the distributor, and this is one of the reasons to introduce the use of permutation rules in rule abs.

On a technical note, notice that the nodes inside a distributor are not replicated yet, but rather moved in the main body of the distributor. The nodes will be replicated only when applying rule dist. This is a difference with the atomic λ-calculus, whose grammar has a tuple $\langle t_1, \dots, t_n \rangle$ containing replicated occurrences of the body of the abstraction inside the distributor. However, Gundersen, Heijltjes, and Parigot do not make use of that possibility to handle these replicated bodies differently while they are present in the distributor.

Other choices are possible, such as replicating all the nodes, or only the uppermost application and the node y (corresponding to fully lazy duplication), as long as at least all free occurrences of y are duplicated.

The substitution relation $\rightarrow_{\text{sub}}$ (resp. distant Beta relation $\rightarrow_{\text{dB}}$) is defined as the closure of $\mapsto_{\text{app}} \cup \mapsto_{\text{dist}} \cup \mapsto_{\text{abs}} \cup \mapsto_{\text{var}}$ (resp. $\mapsto_{\text{dB}}$) under *all* contexts, and the reduction relation $\rightarrow_{\text{R}}$ is the union of $\rightarrow_{\text{sub}}$ and $\rightarrow_{\text{dB}}$.

Example 2.4. This last example showcases different reduction steps with distance, high-

lighted in green.

$$\begin{aligned}(\lambda x.x)\,[z_4/z_5]\,(w[z_1/\!/\mathtt{I}](\lambda y.y[z_2/z_3])) &\to_{\mathrm{dB}} x[x/w[z_1/\!/\mathtt{I}](\lambda y.y[z_2/z_3])][z_4/z_5]\\ &\to_{\mathrm{app}} (x_1x_2)[x_1/w\,[z_1/\!/\mathtt{I}]\,][x_2/\lambda y.y[z_2/z_3]][z_4/z_5]\\ &\to_{\mathrm{var}} (wx_2)[z_1/\!/\mathtt{I}][x_2/\lambda y.y[z_2/z_3]][z_4/z_5]\\ &\to_{\mathrm{dist}} (wx_2)[z_1/\!/\mathtt{I}][x_2/\!/\lambda y.x[x/y\,[z_2/z_3]\,]][z_4/z_5]\\ &\to_{\mathrm{var}} (wx_2)[z_1/\!/\mathtt{I}][x_2/\!/\lambda y.y\,[z_2/z_3]\,][z_4/z_5]\\ &\to_{\mathrm{abs}} (w(\lambda y.y))[z_1/\!/\mathtt{I}][z_2/z_3][z_4/z_5]\end{aligned}$$

Notice that an R-step can be decomposed into some ρ-steps followed by a simpler step not involving any list context. Indeed, $t \to_{\mathrm{R}} u$ could be simulated by $t \to_{\rho}^{*} t' \to_{\mathrm{R}'} u$, where $\to_{\mathrm{R}'}$ is the closure under all contexts of the following set of rewriting rules:

$$\begin{array}{lll}(\lambda x.t)u & \mapsto_{\mathrm{dB}'} & t[x/u]\\ t[x/us] & \mapsto_{\mathrm{app}'} & t\{x/yz\}[y/u][z/s]\\ t[x/\lambda y.u] & \mapsto_{\mathrm{dist}'} & t[x/\!/\lambda y.z[z/u]]\\ t[x/\!/\lambda y.p] & \mapsto_{\mathrm{abs}'} & t\{x/\lambda y.p\}\\ t[x/y] & \mapsto_{\mathrm{var}'} & t\{x/y\}\end{array}$$

For instance, step (2.6) in example 2.3 can be decomposed as follows, where $r = \lambda y.(z_3z_2)y$:

$$(xx)[x/\!/r[z_3/w][z_2/w]] \to_{\pi}^{*} (xx)[x/\!/r][z_3/w][z_2/w] \to_{\mathrm{abs}'} (rr)[z_3/w][z_2/w].$$

This decomposition will be useful in some of our proofs, but we prefer to integrate distance inside the rules, as initially defined on page 59, to highlight the computational behavior and execute permutations only when strictly necessary.

2.1.3 Levels

This subsection introduces the syntactical notion of level and its associated properties. Intuitively, the level of a variable in a term indicates the maximal depth (*only* w.r.t. ESs and not w.r.t. explicit distributors) of its free occurrences. However, in order to be sound with respect to the permutation rules, levels do not consider depth in the usual sense only, but also across linked chains of ES. For instance, the level of z in both $(xx)[x/y[y/z]]$ and $(xx)[x/y][y/z]$ is the same. Levels will play a key role in the next sections: they will be the combinatorial witnesses of the progress of sub-substitution steps, necessary to prove termination of the sub-relation. They will also be helpful to define a decreasing measure on typing derivations in section 2.4. The **level** $\mathrm{lv}_z(t)$ of a variable z in a term t is defined by induction:

$$\begin{aligned}\mathrm{lv}_z(x) &\coloneqq 0\\ \mathrm{lv}_z(t_1t_2) &\coloneqq \max(\mathrm{lv}_z(t_1), \mathrm{lv}_z(t_2))\\ \mathrm{lv}_z(\lambda x.t) &\coloneqq \mathrm{lv}_z(t)\\ \mathrm{lv}_z(t[x \triangleleft u]) &\coloneqq \begin{cases}\mathrm{lv}_z(t) & \text{if } z \notin \mathrm{fv}(u)\\ \max(\mathrm{lv}_z(t), \mathrm{lv}_x(t) + \mathrm{lv}_z(u) + \mathrm{es}([x \triangleleft u])) & \text{otherwise}\end{cases}\end{aligned}$$

In the last two cases, we can always suppose $z \neq x$, because we work modulo α-conversion. Notice that $\mathrm{lv}_z(t) = 0$ whenever $z \notin \mathrm{fv}(t)$ or t is pure.

We illustrate the concept of level by an example. Consider $t = x[x/z[y/w]][w/w']$, then $\mathrm{lv}_z(t) = 1$, $\mathrm{lv}_{w'}(t) = 3$ and $\mathrm{lv}_y(t) = 0$ because $y \notin \mathrm{fv}(t)$.

This notion is also extended to contexts as expected, *i.e.* $\mathrm{lv}_\diamond(\mathtt{C}) = \mathrm{lv}_z(\mathtt{C}\langle z\rangle)$, where z is a fresh variable. Remark that for any variable x, $\mathrm{lv}_\diamond(\mathtt{C}) \leq \mathrm{lv}_x(\mathtt{C}\langle\!\langle x\rangle\!\rangle)$ and $\mathrm{lv}_x(\mathtt{C}\langle\!\langle p\rangle\!\rangle) \leq \mathrm{lv}_x(\mathtt{C}\langle\!\langle x\rangle\!\rangle)$ for any $p \in \mathtt{T}_P$.

Lemma 2.5. *Let $x \neq z$, $t \in \mathtt{T}_R$ and $p \in \mathtt{T}_P$:*

(i) *If $z \notin \mathrm{fv}(p)$, then $\mathrm{lv}_z(t\{x/p\}) = \mathrm{lv}_z(t)$.*

(ii) *If $z \in \mathrm{fv}(p)$, then $\mathrm{lv}_z(t\{x/p\}) = \max(\mathrm{lv}_z(t), \mathrm{lv}_x(t))$.*

Proof. If $x \notin \mathrm{fv}(t)$, then $t\{x/p\} = t$ and the property holds in both cases since $\mathrm{lv}_x(t) = 0$. Let $x \in \mathrm{fv}(t)$.

The first item where $z \notin \mathrm{fv}(p)$ is by induction on t. We detail the case where $t = t'[y \triangleleft u]$ and $z \in \mathrm{fv}(u\{x/p\})$. We have:

$$\begin{aligned}\mathrm{lv}_z(t'\{x/p\}[y \triangleleft u\{x/p\}]) &= \max(\mathrm{lv}_z(t'\{x/p\}), \mathrm{lv}_y(t'\{x/p\}) + \mathrm{lv}_z(u\{x/p\}) + \mathrm{es}([y \triangleleft u])) \\ &=_{i.h.} \max(\mathrm{lv}_z(t'), \mathrm{lv}_y(t') + \mathrm{lv}_z(u) + \mathrm{es}([y \triangleleft u])) \\ &= \mathrm{lv}_z(t'[y \triangleleft u])\end{aligned}$$

The second case where $z \in \mathrm{fv}(p)$ is also by induction on t. We detail the case where $t = t_1[y \triangleleft t_2]$.

Then, $t\{x/p\} = t_1\{x/p\}[y \triangleleft t_2\{x/p\}]$. By α-conversion we can assume $y \notin \mathrm{fv}(p)$. By *i.h.* we have $\mathrm{lv}_z(t_i\{x/p\}) = \max(\mathrm{lv}_z(t_i), \mathrm{lv}_x(t_i))$ for $i \in \{1, 2\}$, if $x \in \mathrm{fv}(t_i)$, $\mathrm{lv}_z(t_i\{x/p\}) = \mathrm{lv}_z(t_i)$ otherwise. There are two cases.

Case $z \notin \mathrm{fv}(t_2\{x/p\})$. Then $z \notin \mathrm{fv}(t_2)$ and necessarily $x \notin \mathrm{fv}(t_2)$ since $z \in \mathrm{fv}(p)$. Then,

$$\mathrm{lv}_z(t\{x/p\}) = \mathrm{lv}_z(t_1\{x/p\}) =_{i.h.} \max(\mathrm{lv}_z(t_1), \mathrm{lv}_x(t_1)) = \max(\mathrm{lv}_z(t_1[y \triangleleft t_2]), \mathrm{lv}_x(t_1[y \triangleleft t_2]))$$

Case $z \in \mathrm{fv}(t_2\{x/p\})$. There are three cases.

Subcase $x \notin \mathrm{fv}(t_1)$. Then $x \in \mathrm{fv}(t_2)$.

$$\begin{aligned}
\mathrm{lv}_z(t\{x/p\}) &= \max(\mathrm{lv}_z(t_1\{x/p\}), \mathrm{lv}_y(t_1\{x/p\}) + \mathrm{lv}_z(t_2\{x/p\}) + \mathrm{es}([y \triangleleft t_2])) \\
&=_{\text{(i)}} \max(\mathrm{lv}_z(t_1), \mathrm{lv}_y(t_1) + \mathrm{lv}_z(t_2\{x/p\}) + \mathrm{es}([y \triangleleft t_2])) \\
&=_{i.h.} \max(\mathrm{lv}_z(t_1), \mathrm{lv}_y(t_1) + \max(\mathrm{lv}_z(t_2), \mathrm{lv}_x(t_2)) + \mathrm{es}([y \triangleleft t_2])) \\
&= \max(\mathrm{lv}_z(t_1), \mathrm{lv}_y(t_1) + \mathrm{lv}_z(t_2) + \mathrm{es}([y \triangleleft t_2]), \\
&\qquad \mathrm{lv}_y(t_1) + \mathrm{lv}_x(t_2) + \mathrm{es}([y \triangleleft t_2])) \\
&= \begin{cases} \max(\max(\mathrm{lv}_z(t_1), \mathrm{lv}_y(t_1) + \mathrm{lv}_z(t_2) + \mathrm{es}([y \triangleleft t_2])), & \\ \qquad \mathrm{lv}_y(t_1) + \mathrm{lv}_x(t_2) + \mathrm{es}([y \triangleleft t_2])) & z \in \mathrm{fv}(t_2) \\ \max(\mathrm{lv}_z(t_1), \mathrm{lv}_y(t_1) + \mathrm{lv}_x(t_2) + \mathrm{es}([y \triangleleft t_2])) & z \notin \mathrm{fv}(t_2) \end{cases} \\
&= \max(\mathrm{lv}_z(t_1[y \triangleleft t_2]), \mathrm{lv}_x(t_1[y \triangleleft t_2]))
\end{aligned}$$

Subcase $x \notin \mathrm{fv}(t_2)$. Then $x \in \mathrm{fv}(t_1)$ and $z \in \mathrm{fv}(t_2)$:

$$\begin{aligned}
\mathrm{lv}_z(t\{x/p\}) &= \max(\mathrm{lv}_z(t_1\{x/p\}), \mathrm{lv}_y(t_1\{x/p\}) + \mathrm{lv}_z(t_2\{x/p\}) + \mathrm{es}([y \triangleleft t_2])) \\
&=_{i.h.+\text{(i)}} \max(\mathrm{lv}_z(t_1), \mathrm{lv}_x(t_1), \mathrm{lv}_y(t_1) + \mathrm{lv}_z(t_2) + \mathrm{es}([y \triangleleft t_2])) \\
&= \max(\max(\mathrm{lv}_z(t_1), \mathrm{lv}_y(t_1) + \mathrm{lv}_z(t_2) + \mathrm{es}([y \triangleleft t_2])), \mathrm{lv}_x(t_1[y \triangleleft t_2])) \\
&= \max(\mathrm{lv}_z(t_1[y \triangleleft t_2]), \mathrm{lv}_x(t_1[y \triangleleft t_2]))
\end{aligned}$$

Subcase $x \in \mathrm{fv}(t_1) \cap \mathrm{fv}(t_2)$.

$$\begin{aligned}
\mathrm{lv}_z(t\{x/p\}) &= \max(\mathrm{lv}_z(t_1\{x/p\}), \mathrm{lv}_y(t_1\{x/p\}) + \mathrm{lv}_z(t_2\{x/p\}) + \mathrm{es}([y \triangleleft t_2])) \\
&=_{i.h.} \max(\max(\mathrm{lv}_z(t_1), \mathrm{lv}_x(t_1)), \\
&\qquad \mathrm{lv}_y(t_1) + \max(\mathrm{lv}_z(t_2), \mathrm{lv}_x(t_2)) + \mathrm{es}([y \triangleleft t_2])) \\
&= \max(\mathrm{lv}_z(t_1), \mathrm{lv}_x(t_1), \mathrm{lv}_y(t_1) + \mathrm{lv}_z(t_2) + \mathrm{es}([y \triangleleft t_2]), \\
&\qquad \mathrm{lv}_y(t_1) + \mathrm{lv}_x(t_2) + \mathrm{es}([y \triangleleft t_2])) \\
&= \begin{cases} \max(\mathrm{lv}_z(t_1), \mathrm{lv}_y(t_1) + \mathrm{lv}_z(t_2) + \mathrm{es}([y \triangleleft t_2]), & \\ \qquad \mathrm{lv}_x(t_1), \mathrm{lv}_y(t_1) + \mathrm{lv}_x(t_2) + \mathrm{es}([y \triangleleft t_2])) & z \in \mathrm{fv}(t_2) \\ \max(\mathrm{lv}_z(t_1), \mathrm{lv}_x(t_1), \mathrm{lv}_y(t_1) + \mathrm{lv}_x(t_2) + \mathrm{es}([y \triangleleft t_2])) & z \notin \mathrm{fv}(t_2) \end{cases} \\
&= \max(\mathrm{lv}_z(t_1[y \triangleleft t_2]), \mathrm{lv}_x(t_1[y \triangleleft t_2]))
\end{aligned}$$

□

Lemma 2.6. *Let $t \in \mathtt{T}_R$ and w be any variable.*

(i) *If $t_0 \longrightarrow_\rho t_1$, then $\mathrm{lv}_w(t_0) \geq \mathrm{lv}_w(t_1)$.*

(ii) *If $t_0 \longrightarrow_{\mathrm{sub}} t_1$, then $\mathrm{lv}_w(t_0) \geq \mathrm{lv}_w(t_1)$.*

Proof. We start with item (i). Let $t_0 = \mathtt{C}\langle o \rangle$ and $t_1 = \mathtt{C}\langle o' \rangle$, where $o \longrightarrow_\rho o'$ is a root step. We reason by induction on $\mathtt{C}$. First we consider the base cases, where $\mathtt{C} = \Diamond$. We detail

two cases, where $w \in \mathrm{fv}(u)$.

Case $t_0 = t[x \triangleleft u]s \longrightarrow_\rho (ts)[x \triangleleft u] = t_1$, where $x \notin \mathrm{fv}(s)$.

$$\begin{aligned}
&\mathrm{lv}_w(t[x \triangleleft u]s)\\
&= \max(\mathrm{lv}_w(t[x \triangleleft u]), \mathrm{lv}_w(s))\\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \mathrm{lv}_w(u) + \mathrm{es}([x \triangleleft u]), \mathrm{lv}_w(s))\\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_w(s), \mathrm{lv}_x(t) + \mathrm{lv}_w(u) + \mathrm{es}([x \triangleleft u]))\\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_w(s), \max(\mathrm{lv}_x(t), 0) + \mathrm{lv}_w(u) + \mathrm{es}([x \triangleleft u])) && (x \notin \mathrm{fv}(s))\\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_w(s), \max(\mathrm{lv}_x(t), \mathrm{lv}_x(s)) + \mathrm{lv}_w(u) + \mathrm{es}([x \triangleleft u]))\\
&= \max(\mathrm{lv}_w(ts), \mathrm{lv}_x(ts) + \mathrm{lv}_w(u) + \mathrm{es}([x \triangleleft u]))\\
&= \mathrm{lv}_w((ts)[x \triangleleft u])
\end{aligned}$$

Case $t_0 = t[y \triangleleft s[x \triangleleft u]] \longrightarrow_\rho t[y \triangleleft s][x \triangleleft u] = t_1$, where $x \notin \mathrm{fv}(t)$. We only detail the case where $w \in \mathrm{fv}(s)$.

$$\begin{aligned}
\mathrm{lv}_w(t[y \triangleleft s[x \triangleleft u]]) &= \max(\mathrm{lv}_w(t), \mathrm{lv}_y(t) + \mathrm{lv}_w(s[x \triangleleft u]) + \mathrm{es}([y \triangleleft s]))\\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_y(t)\\
&\qquad + \max(\mathrm{lv}_w(s), \mathrm{lv}_x(s) + \mathrm{lv}_w(u) + \mathrm{es}([x \triangleleft u])) + \mathrm{es}([y \triangleleft s]))\\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_y(t) + \mathrm{lv}_w(s) + \mathrm{es}([y \triangleleft s]),\\
&\qquad \mathrm{lv}_y(t) + \mathrm{lv}_x(s) + \mathrm{lv}_w(u) + \mathrm{es}([x \triangleleft u]) + \mathrm{es}([y \triangleleft s]))\\
&= \max(\max(\mathrm{lv}_w(t), \mathrm{lv}_y(t) + \mathrm{lv}_w(s) + \mathrm{es}([y \triangleleft s])),\\
&\qquad \max(\mathrm{lv}_x(t), \mathrm{lv}_y(t) + \mathrm{lv}_x(s) + \mathrm{es}([y \triangleleft s])) + \mathrm{lv}_w(u) + \mathrm{es}([x \triangleleft u]))\\
&\geq \max(\mathrm{lv}_w(t[y \triangleleft s]), \mathrm{lv}_x(t[y \triangleleft s]) + \mathrm{lv}_w(u) + \mathrm{es}([x \triangleleft u]))\\
&= \mathrm{lv}_w(t[y \triangleleft s][x \triangleleft u])
\end{aligned}$$

The inductive cases are the following:

Case $\mathtt{C} = \lambda x.\mathtt{C}'$, where $x \neq w$. Then

$$\mathrm{lv}_w(\lambda x.\mathtt{C}'\langle o\rangle) = \mathrm{lv}_w(\mathtt{C}'\langle o\rangle) \geq_{i.h.} \mathrm{lv}_w(\mathtt{C}'\langle o'\rangle) = \mathrm{lv}_w(\mathtt{C}\langle o'\rangle)$$

Case $\mathtt{C} = \mathtt{C}'u$. Then

$$\mathrm{lv}_w(\mathtt{C}'\langle o\rangle u) = \max(\mathrm{lv}_w(\mathtt{C}'\langle o\rangle), \mathrm{lv}_w(u)) \geq_{i.h.} \max(\mathrm{lv}_w(\mathtt{C}'\langle o'\rangle), \mathrm{lv}_w(u)) = \mathrm{lv}_w(\mathtt{C}\langle o'\rangle)$$

Case $\mathtt{C} = u\mathtt{C}'$. Then

$$\mathrm{lv}_w(u\mathtt{C}'\langle o\rangle) = \max(\mathrm{lv}_w(u), \mathrm{lv}_w(\mathtt{C}'\langle o\rangle)) \geq_{i.h.} \max(\mathrm{lv}_w(u), \mathrm{lv}_w(\mathtt{C}'\langle o'\rangle)) = \mathrm{lv}_w(\mathtt{C}\langle o'\rangle)$$

Case $\mathtt{C} = \mathtt{C}'[x \lhd u]$. Then

Subcase $w \notin \mathrm{fv}(u)$. $\mathrm{lv}_w(\mathtt{C}'\langle o\rangle[x \lhd u]) = \mathrm{lv}_w(\mathtt{C}'\langle o\rangle) \geq_{i.h.} \mathrm{lv}_w(\mathtt{C}'\langle o'\rangle) = \mathrm{lv}_w(\mathtt{C}\langle o'\rangle)$.

Subcase $w \in \mathrm{fv}(u)$. Then

$$\begin{aligned}\mathrm{lv}_w(\mathtt{C}'\langle o\rangle[x \lhd u]) &= \max(\mathrm{lv}_w(\mathtt{C}'\langle o\rangle), \mathrm{lv}_x(\mathtt{C}'\langle o\rangle) + \mathrm{lv}_w(u) + \mathrm{es}([x \lhd u])) \\ &\geq_{i.h.} \max(\mathrm{lv}_w(\mathtt{C}'\langle o'\rangle), \mathrm{lv}_x(\mathtt{C}'\langle o'\rangle) + \mathrm{lv}_w(u) + \mathrm{es}([x \lhd u])) \\ &= \mathrm{lv}_w(\mathtt{C}'\langle o'\rangle[x \lhd u]) \\ &= \mathrm{lv}_w(\mathtt{C}\langle o'\rangle)\end{aligned}$$

Case $\mathtt{C} = u[x \lhd \mathtt{C}']$. Then

Subcase $w \notin \mathrm{fv}(\mathtt{C}'\langle o\rangle)$. Then $\mathrm{lv}_w(u[x \lhd \mathtt{C}'\langle o\rangle]) = \mathrm{lv}_w(u) = \mathrm{lv}_w(u[x \lhd \mathtt{C}'\langle o'\rangle]) = \mathrm{lv}_w(\mathtt{C}\langle o'\rangle)$.

Subcase $w \in \mathrm{fv}(\mathtt{C}'\langle o\rangle)$. Then $\mathrm{lv}_w(u[x \lhd \mathtt{C}'\langle o\rangle]) = \max(\mathrm{lv}_w(u), \mathrm{lv}_x(u) + \mathrm{lv}_w(\mathtt{C}'\langle o\rangle) + \mathrm{es}([x \lhd \mathtt{C}'\langle o\rangle])) \geq_{i.h.} \max(\mathrm{lv}_w(u), \mathrm{lv}_x(u) + \mathrm{lv}_w(\mathtt{C}'\langle o'\rangle) + \mathrm{es}([x \lhd \mathtt{C}'\langle o\rangle])) = \mathrm{lv}_w(u[x \lhd \mathtt{C}'\langle o'\rangle]) = \mathrm{lv}_w(\mathtt{C}\langle o'\rangle)$.

Now, we consider item (ii). We reason by induction on the reduction relation, *i.e.* by induction on the context $\mathtt{C}$ where the root reduction takes place. We detail the base case which is $\mathtt{C} = \Diamond$. In all such cases we use point (i) to push $\mathtt{L}$ outside, *i.e.* we can write $t_0 \to_{\mathrm{sub}} t_1$ as $t_0 \to_\rho \mathtt{L}\langle t_0'\rangle \to_{\mathrm{sub}'} \mathtt{L}\langle t_1'\rangle = t_1$, where $t_0' \to_{\mathrm{sub}'} t_1'$ does not push any list context outside. We then show the property for steps $t_0' \to_{\mathrm{sub}'} t_1'$ not pushing any substitution outside and we conclude by $\mathrm{lv}_w(t_0) \geq_{(i)} \mathrm{lv}_w(\mathtt{L}\langle t_0'\rangle) \geq \mathrm{lv}_w(\mathtt{L}\langle t_1'\rangle) = \mathrm{lv}_w(t_1)$. The inductive cases for $\mathtt{C}$ are treated as in point (i).

Case $t_0' = t[x/us] \to_{\mathrm{app}} t\{x/yz\}[y/u][z/s] = t_1'$, where y and z are fresh variables. We detail the case where $w \in \mathrm{fv}(u) \cap \mathrm{fv}(s)$ and $x \in \mathrm{fv}(t)$.

$$
\begin{aligned}
&\mathrm{lv}_w(t[x/us]) \\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \mathrm{lv}_w(us) + 1) \\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \mathrm{lv}_w(u) + 1, \mathrm{lv}_x(t) + \mathrm{lv}_w(s) + 1) \\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \mathrm{lv}_w(u) + 1, \max(0, \mathrm{lv}_x(t) + 0) + \mathrm{lv}_w(s) + 1) \\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \mathrm{lv}_w(u) + 1, \max(\mathrm{lv}_z(t), \mathrm{lv}_x(t) + \mathrm{lv}_z(yz)) + \mathrm{lv}_w(s) + 1) \\
&=_{2.5(ii)} \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \mathrm{lv}_w(u) + 1, \mathrm{lv}_z(t\{x/yz\}) + \mathrm{lv}_w(s) + 1) \\
&= \max(\mathrm{lv}_w(t), \max(0, \mathrm{lv}_x(t) + 0) + \mathrm{lv}_w(u) + 1, \mathrm{lv}_z(t\{x/yz\}) + \mathrm{lv}_w(s) + 1) \\
&= \max(\mathrm{lv}_w(t), \max(\mathrm{lv}_y(t), \mathrm{lv}_x(t) + \mathrm{lv}_y(yz)) + \mathrm{lv}_w(u) + 1, \mathrm{lv}_z(t\{x/yz\}) + \mathrm{lv}_w(s) + 1) \\
&=_{2.5(ii)} \max(\mathrm{lv}_w(t), \mathrm{lv}_y(t\{x/yz\}) + \mathrm{lv}_w(u) + 1, \mathrm{lv}_z(t\{x/yz\}) + \mathrm{lv}_w(s) + 1) \\
&= \max(\mathrm{lv}_w(t\{x/yz\}), \mathrm{lv}_y(t\{x/yz\}) + \mathrm{lv}_w(u) + 1, \mathrm{lv}_z(t\{x/yz\}) + \mathrm{lv}_w(s) + 1) \\
&= \max(\mathrm{lv}_w(t\{x/yz\}[y/u]), \mathrm{lv}_z(t\{x/yz\}) + \mathrm{lv}_w(s) + 1) \\
&= \max(\mathrm{lv}_w(t\{x/yz\}[y/u]), \mathrm{lv}_z(t\{x/yz\}[y/u]) + \mathrm{lv}_w(s) + 1) \\
&= \mathrm{lv}_w(t\{x/yz\}[y/u][z/s])
\end{aligned}
$$

Case $t_0' = t[x/\lambda y.u] \longrightarrow_{\mathrm{dist}} t[x/\!/\lambda y.z[z/u]] = t_1'$. There are two cases.

Subcase $w \notin \mathrm{fv}(\lambda y.u)$. $\mathrm{lv}_w(t[x/\lambda y.u]) = \mathrm{lv}_w(t) = \mathrm{lv}_w(t[x/\!/\lambda y.z[z/u]])$

Subcase $w \in \mathrm{fv}(\lambda y.u)$ (*i.e.* $w \in \mathrm{fv}(u)$ and $w \neq y$).

$$
\begin{aligned}
\mathrm{lv}_w(t[x/\lambda y.u]) &= \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \mathrm{lv}_w(\lambda y.u) + 1) \\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \mathrm{lv}_w(u) + 1) \\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \max(0, 0 + \mathrm{lv}_w(u) + 1)) \\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \max(\mathrm{lv}_w(z), \mathrm{lv}_z(z) + \mathrm{lv}_w(u) + 1)) \\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \mathrm{lv}_w(z[z/u])) \\
&= \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \mathrm{lv}_w(\lambda y.z[z/u])) \\
&= \mathrm{lv}_w(t[x/\!/\lambda y.z[z/u]])
\end{aligned}
$$

Case $t_0' = t[x/\!/\lambda y.u] \longrightarrow_{\mathrm{abs}} t\{x/\lambda y.u\} = t_1'$, where u is pure. There are two cases:

Subcase $w \notin \mathrm{fv}(\lambda y.u)$ or $x \notin \mathrm{fv}(t)$.

$$\mathrm{lv}_w(t[x/\!/\lambda y.u]) = \mathrm{lv}_w(t) =_{2.5(i)} \mathrm{lv}_w(t\{x/\lambda y.u'\}) = \mathrm{lv}_w(\mathrm{L}\langle t\{x/\lambda y.u'\}\rangle)$$

Subcase $w \in \mathrm{fv}(\lambda y.u)$ and $x \in \mathrm{fv}(t)$.

$$\mathrm{lv}_w(t[x/\!/\lambda y.u]) = \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t)) =_{2.5(ii)} \mathrm{lv}_w(t\{x/\lambda y.u\})$$

Case $t_0' = t[x/y] \longrightarrow_{\mathrm{var}} t\{x/y\} = t_1'$. There are three subcases.

Subcase $w \neq y$. $\mathrm{lv}_w(t[x/y]) = \mathrm{lv}_w(t) =_{2.5(i)} \mathrm{lv}_w(t\{x/y\})$

Subcase $w = y$ and $x \notin \mathrm{fv}(t)$.

$$\begin{aligned}\mathrm{lv}_w(t[x/y]) &= \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \mathrm{lv}_w(y) + 1)\\ &= \max(\mathrm{lv}_w(t), 1) \geq \mathrm{lv}_w(t) =_{2.5(i)} \mathrm{lv}_w(t\{x/y\})\end{aligned}$$

Subcase $w = y$ and $x \in \mathrm{fv}(t)$.

$$\begin{aligned}\mathrm{lv}_w(t[x/y]) &= \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t) + \mathrm{lv}_w(y) + 1)\\ &\geq \max(\mathrm{lv}_w(t), \mathrm{lv}_x(t)) =_{2.5(ii)} \mathrm{lv}_w(t\{x/y\})\end{aligned}$$

□

Notice that there are two cases when the level of a variable in a term may decrease:

- Moving an explicit cut out of another one with a permutation rule when the first cut is a void cut, *i.e.* its domain does not bind any other variable. Thus *e.g.* if $t = x[x/z[y/w]][w/w'] \to_\rho x[x/z][y/w][w/w'] = u$, then $\mathrm{lv}_{w'}(t) = 3 > 2 = \mathrm{lv}_{w'}(u)$.
- Using rule $\mapsto_{\mathrm{var}}$. Thus *e.g.* if $t = (xx)[x/y][y/z] \to_{\mathrm{var}} (yy)[y/z] = u$, then $\mathrm{lv}_z(t) = 2 > 1 = \mathrm{lv}_z(u)$.

Hence, levels alone are not enough to prove termination of $\to_{\mathrm{sub}}$. We thus define a decreasing measure for $\to_{\mathrm{sub}}$ in which not only variables are indexed by a level, but also constructors. For instance, in the term $t[x/\lambda y.yz]$, we can consider that the level of *all* the constructors of $\lambda y.yz$, including the abstraction and the application, have level $\mathrm{lv}_x(t)$. This will ensure that the level of an abstraction will decrease when applying rule dist, as well as the level of an application when applying rule app.

2.2 Operational Properties

We now prove three key properties of the λR-calculus: termination of the reduction system $\to_{\mathrm{sub}}$, relation between λR and the λ-calculus, and confluence of the reduction system $\to_{\mathrm{R}}$.

Termination of $\to_{\mathrm{sub}}$. Some (rather informal) arguments are provided in [GHP13b] to justify termination of the substitution subrelation of their calculus. We expand these ideas into an alternative full formal proof adapted to our case, which is based on a measure being strictly decreasing w.r.t. $\to_{\mathrm{sub}}$.

We consider a set $\mathcal{O}$ of objects of the form $\mathrm{a}(k,n)$ or $\mathrm{b}(k)$ ($k, n \in \mathbb{N}$), which is equipped with the following ordering $>^{\mathcal{O}}$ ($\geq^{\mathcal{O}}$ denotes its reflexive closure):

$$\begin{array}{llll}\mathrm{a}(k,n) >^{\mathcal{O}} \mathrm{a}(k',n) & \text{if } k > k', \text{ or } (k = k' \text{ and } n > n') & \mathrm{b}(k) >^{\mathcal{O}} \mathrm{a}(k',n) & \text{if } k \geq k'\\ \mathrm{a}(k,n) >^{\mathcal{O}} \mathrm{b}(k') & \text{if } k > k' & \mathrm{b}(k) >^{\mathcal{O}} \mathrm{b}(k') & \text{if } k > k'\end{array}$$

Lemma 2.7. *The order $>^{\mathcal{O}}$ on the set $\mathcal{O}$ is well-founded.*

Proof. Let us consider the set $\mathbb{N}$ equipped with the standard order $>_{\mathbb{N}}$ on natural numbers. Let us also consider the set $\mathbb{N}_\infty \coloneqq \mathbb{N} \uplus \{\infty\}$ equipped with the order $>_\infty \coloneqq >_{\mathbb{N}} \cup \{\langle \infty, n\rangle \mid n \in \mathbb{N}\}$. Since $>_{\mathbb{N}}$ and $>_\infty$ are both well-founded, then the lexicographic order induced by $\langle >_{\mathbb{N}}, >_\infty\rangle$ on $\mathbb{N}\times\mathbb{N}_\infty$, written $>_{\texttt{LEX}}$, is also well-founded. We show that $>^{\mathcal{O}}$ is well-founded by projecting it into the well-founded order $>_{\texttt{LEX}}$, *i.e.* we define a projection function P such that $s >^{\mathcal{O}} s'$ implies $\mathrm{P}(s) >_{\texttt{LEX}} \mathrm{P}(s')$, for any $s, s' \in \mathcal{O}$. Let us define $\mathrm{P}(s) = \langle \mathbb{L}(s), \mathbb{S}(s)\rangle$, where $\mathbb{L}(\mathrm{a}(k,n)) \coloneqq k$ and $\mathbb{L}(\mathrm{b}(k)) \coloneqq k$ while $\mathbb{S}(\mathrm{a}(k,n)) \coloneqq n$ and $\mathbb{S}(\mathrm{b}(k)) \coloneqq \infty$. We reason by cases.

Case $s_0 = \mathrm{a}(k,n) >^{\mathcal{O}} \mathrm{a}(k',n') = s_1$. Then $\langle \mathbb{L}(s_0), \mathbb{S}(s_0)\rangle = \langle k, n\rangle >_{\texttt{LEX}} \langle k', n'\rangle = \langle \mathbb{L}(s_1), \mathbb{S}(s_1)\rangle$ holds by definition since either $k > k'$ or $k = k'$ and $n > n'$.

Case $s_0 = \mathrm{b}(k) >^{\mathcal{O}} \mathrm{b}(k') = s_1$. Then $\langle \mathbb{L}(s_0), \mathbb{S}(s_0)\rangle = \langle k, \infty\rangle >_{\texttt{LEX}} \langle k', \infty\rangle = \langle \mathbb{L}(s_1), \mathbb{S}(s_1)\rangle$ holds by definition since $k > k'$.

Case $s_0 = \mathrm{a}(k,n) >^{\mathcal{O}} \mathrm{b}(k') = s_1$. Then $\langle \mathbb{L}(s_0), \mathbb{S}(s_0)\rangle = \langle k, n\rangle >_{\texttt{LEX}} \langle k', \infty\rangle = \langle \mathbb{L}(s_1), \mathbb{S}(s_1)\rangle$ holds by definition since $k > k'$.

Case $s_0 = \mathrm{b}(k) >^{\mathcal{O}} \mathrm{a}(k',n') = s_1$. Then $\langle \mathbb{L}(s_0), \mathbb{S}(s_0)\rangle = \langle k, \infty\rangle >_{\texttt{LEX}} \langle k', n'\rangle = \langle \mathbb{L}(s_1), \mathbb{S}(s_1)\rangle$ holds by definition since either $k > k'$ or $k = k'$ and $\infty > n$. □

We write $>^{\mathcal{O}}_{\texttt{MUL}}$ for the multiset extension of the order $>^{\mathcal{O}}$ on $\mathcal{O}$, which turns out to be well-founded [BN98] by lemma 2.7. Some operations on multisets are needed to build the measure $\mathrm{C}(\_)$ on terms. Indeed, let M be a multiset of objects in $\mathcal{O}$. Multiset union is denoted $\sqcup$. Furthermore:

1. The a-**elements** (resp. b-**elements**) of the multiset M are all the objects of the form $\mathrm{a}(k,n)$ (resp. $\mathrm{b}(k)$) in M. We then may write M as $M_{\mathrm{a}} \sqcup M_{\mathrm{b}}$, where M_{a} (resp. M_{b}) contains all the a-elements (resp b-elements) of M.

2. Given $K \in \mathbb{N}$, we write $M^{\leq K}$ (resp. $M^{>K}$) for the multiset containing all $o \in M$ such that the first element of o is less than K (resp. strictly greater than K). We write $M_{\mathrm{a}}^{>K}$ for $M^{>K} \sqcap M_{\mathrm{a}}$.

3. M can thus be decomposed in three disjoint multisets $M_{\mathrm{b}}, M_{\mathrm{a}}^{\leq K}$ and $M_{\mathrm{a}}^{>K}$, for every $K \in \mathbb{N}$.

4. We also define the following operation on M:

$$p \cdot M \coloneqq [\mathrm{a}(p+k, n) \mid \mathrm{a}(k,n) \in M] \sqcup [\mathrm{b}(p+k) \mid \mathrm{b}(k) \in M]$$

We are now ready to (inductively) define our **cuts level** measure $\mathrm{C}(\cdot)$ on terms.

$$\mathrm{C}(x) \coloneqq [\,] \qquad \mathrm{C}(\lambda x.t) \coloneqq \mathrm{C}(t) \qquad \mathrm{C}(tu) \coloneqq \mathrm{C}(t) \sqcup \mathrm{C}(u)$$
$$\mathrm{C}(t[x/u]) \coloneqq \mathrm{C}(t) \sqcup (\mathrm{lv}_x(t)+1) \cdot \mathrm{C}(u) \sqcup [\mathrm{a}(\mathrm{lv}_x(t)+1, |u|)]$$
$$\mathrm{C}(t[x/\!/u]) \coloneqq \mathrm{C}(t) \sqcup \mathrm{lv}_x(t) \cdot \mathrm{C}(u) \sqcup [\mathrm{b}(\mathrm{lv}_x(t))]$$

Intuitively, the integer k in $\mathrm{a}(k,n)$ and $\mathrm{b}(k)$ counts the level of variables bound by explicit substitutions, while n counts the size of terms to be substituted by an ES. Remark that for every pure term p we have $\mathrm{C}(p) = [\,]$.

Example 2.8. Consider the following reduction sequence:

$$\begin{aligned} t_0 \coloneqq (yy)[y/(\lambda z.x)w] &\to_{\mathrm{app}} (y_1y_2)(y_1y_2)[y_1/\lambda z.x][y_2/w] && \coloneqq t_1 \\ &\to_{\mathrm{var}} (y_1w)(y_1w)[y_1/\lambda z.x] && \coloneqq t_2 \\ &\to_{\mathrm{dist}} (y_1w)(y_1w)[y_1/\!\!/\lambda z.x'[x'/x]] && \coloneqq t_3 \\ &\to_{\mathrm{abs}} ((\lambda z.x')w)((\lambda z.x')w)[x'/x] && \coloneqq t_4 \\ &\to_{\mathrm{var}} ((\lambda z.x)w)((\lambda z.x)w) && \coloneqq t_5 \end{aligned}$$

We have $\mathrm{C}(t_0) = [\mathrm{a}(1,4)]$, $\mathrm{C}(t_1) = [\mathrm{a}(1,1), \mathrm{a}(1,2)]$, $\mathrm{C}(t_2) = [\mathrm{a}(1,2)]$, $\mathrm{C}(t_3) = [\mathrm{a}(1,1), \mathrm{b}(0)]$, $\mathrm{C}(t_4) = [\mathrm{a}(1,1)]$ and $\mathrm{C}(t_5) = [\,]$.

Fact 2.9. Some properties on multisets are straightforward:

(i) If $M_1 >^{\mathcal{O}}_{\mathtt{MUL}} M_2$, then $M_1 \sqcup M >^{\mathcal{O}}_{\mathtt{MUL}} M_2 \sqcup M$.

(ii) If $M_1 >^{\mathcal{O}}_{\mathtt{MUL}} M_2$, then $k \cdot M_1 >^{\mathcal{O}}_{\mathtt{MUL}} k \cdot M_2$ for any $k \in \mathbb{N}$.

(iii) $k_1 \cdot k_2 \cdot M = (k_1 + k_2) \cdot M$.

Lemma 2.10. *If* $\mathrm{C}(t) >^{\mathcal{O}}_{MUL} \mathrm{C}(u)$ *and* $\mathrm{lv}_x(t) \geq \mathrm{lv}_x(u)$ *holds for every* $x \in \mathrm{dom}(\mathtt{L})$*, then* $\mathrm{C}(\mathtt{L}\langle t\rangle) >^{\mathcal{O}}_{MUL} \mathrm{C}(\mathtt{L}\langle u\rangle)$.

Proof. By induction on $\mathtt{L}$. The property is straightforward. □

Lemma 2.11. *Let* t *be a term,* x *a variable and* p *a pure term. Let* $K = \mathrm{lv}_x(t)$*. Then* $\mathrm{C}(t\{x/p\}) \sqsubseteq \mathrm{C}(t)_{\mathrm{b}} \sqcup \mathrm{C}(t)_{\mathrm{a}}^{>K} \sqcup [\mathrm{a}(k,n) \mid k \leq K \text{ and } n \in \mathbb{N}]$.

Proof. By induction on t. In this proof, $\mathrm{fst}(o)$ denotes the first element of an object $o \in \mathcal{O}$: $\mathrm{fst}(\mathrm{a}(k,n)) = k$ and $\mathrm{fst}(\mathrm{b}(k)) = k$.

Case $t = y$. Then $\mathrm{C}(y) = \mathrm{C}(y\{x/p\}) = [\,]$ so the property is straightforward.

Case $t = \lambda y.u$. Then $\mathrm{C}(t\{x/p\}) = \mathrm{C}(u\{x/p\})$ and $\mathrm{lv}_x(t) = \mathrm{lv}_x(u)$. The property trivially holds by the *i.h.*

Case $t = u_1u_2$. Then we have $\mathrm{C}(t\{x/p\}) = \mathrm{C}(u_1\{x/p\}) \sqcup \mathrm{C}(u_2\{x/p\})$ and $\mathrm{lv}_x(u_1u_2) = \max(\mathrm{lv}_x(u_1), \mathrm{lv}_x(u_2))$. Let $o \in \mathrm{C}(t\{x/p\})$ thus $o \in \mathrm{C}(u_1\{x/p\}) \sqcup \mathrm{C}(u_2\{x/p\})$. Suppose w.l.o.g. that $o \in \mathrm{C}(u_1\{x/p\})$. Let $K_1 = \mathrm{lv}_x(u_1) \leq K$. By the *i.h.* we have either (1) $o \in \mathrm{C}(u_1)_{\mathrm{b}}$, (2) $o \in \mathrm{C}(u_1)_{\mathrm{a}}^{>K_1}$, or (3) $o = \mathrm{a}(k,n)$ where $k \leq K_1$. If (1) holds, then $o \in \mathrm{C}(t)_{\mathrm{b}}$ and we are done. Otherwise, $o = \mathrm{a}(k,n)$, and we consider two cases.

1. $k > K$. Then (2) implies $o \in \mathrm{C}(u_1)_\mathrm{a}^{>K}$ which implies $o \in \mathrm{C}(t)_\mathrm{a}^{>K}$ while (3) implies $k \leq K$ which leads to a contradiction.
2. $k \leq K$. We are done.

Case $t = u_1[y/u_2]$. Then we can assume by α-conversion that $y \notin \mathrm{fv}(p)$. Therefore,

$$\begin{aligned}\mathrm{C}(t) &= \mathrm{C}(u_1) \sqcup (\mathrm{lv}_y(u_1) + 1) \cdot \mathrm{C}(u_2) \sqcup [\mathrm{a}(\mathrm{lv}_y(u_1) + 1, |u_2|)] \text{ and}\\ \mathrm{C}(t\{x/p\}) &= \mathrm{C}(u_1\{x/p\}) \sqcup (\mathrm{lv}_y(u_1\{x/p\}) + 1) \cdot \mathrm{C}(u_2\{x/p\})\\ &\quad \sqcup [\mathrm{a}(\mathrm{lv}_y(u_1\{x/p\}) + 1, |u_2\{x/p\}|)]\\ &=_{2.5(\mathrm{i})} \mathrm{C}(u_1\{x/p\}) \sqcup (\mathrm{lv}_y(u_1) + 1) \cdot \mathrm{C}(u_2\{x/p\}) \sqcup [\mathrm{a}(\mathrm{lv}_y(u_1) + 1, |u_2\{x/p\}|)]\end{aligned}$$

There are two cases:

Subcase $x \notin \mathrm{fv}(u_2)$. Then $\mathrm{lv}_x(t) = \mathrm{lv}_x(u_1)$. Moreover, $\mathrm{C}(u_2\{x/p\}) = \mathrm{C}(u_2)$ and $|u_2\{x/p\}| = |u_2|$. Let $o \in \mathrm{C}(t\{x/p\})$.

Subsubcase $o \in \mathrm{C}(u_1\{x/p\})$. Then let $K_1 = \mathrm{lv}_x(u_1) = \mathrm{lv}_x(t) = K$, so that the *i.h.* gives either (1) $o \in \mathrm{C}(u_1)_\mathrm{b}$, (2) $o \in \mathrm{C}(u_1)_\mathrm{a}^{>K_1}$, or (3) $o = \mathrm{a}(k, n)$ where $k \leq K_1$. If (1) holds, then $o \in \mathrm{C}(t)_\mathrm{b}$ and we are done. If (2) holds, then $o \in \mathrm{C}(u_1)_\mathrm{a}^{>K}$ since $K_1 = K$, which implies $o \in \mathrm{C}(t)_\mathrm{a}^{>K}$ and we are done. Otherwise, (3) holds and $k \leq K_1 = K$ as required.

Subsubcase $o \in (\mathrm{lv}_y(u_1) + 1) \cdot \mathrm{C}(u_2\{x/p\}) = (\mathrm{lv}_y(u_1) + 1) \cdot \mathrm{C}(u_2)$. We have $o \in \mathrm{C}(t) = \mathrm{C}(t)_\mathrm{b} \sqcup \mathrm{C}(t)_\mathrm{a}^{>K} \sqcup \mathrm{C}(t)_\mathrm{a}^{\leq K}$, which particularly implies in the last case that $o = \mathrm{a}(k, n)$ and $k \leq K$.

Subsubcase $o = \mathrm{a}(\mathrm{lv}_y(u_1) + 1, |u_2\{x/p\}|) = \mathrm{a}(\mathrm{lv}_y(u_1) + 1, |u_2|)$. $o \in \mathrm{C}(t)$, thus either $o \in \mathrm{C}(t)^{>K}$ or $o \in \mathrm{C}(t)^{\leq K}$, which particularly implies in the last case that $\mathrm{fst}(o) \leq K$.

Subcase $x \in \mathrm{fv}(u_2)$. Then $\mathrm{lv}_x(t) = \max(\mathrm{lv}_x(u_1), \mathrm{lv}_y(u_1) + \mathrm{lv}_x(u_2) + 1)$. Let o be an object of $\mathrm{C}(t\{x/p\})$.

Subsubcase $o \in \mathrm{C}(u_1\{x/p\})$. Let $K_1 = \mathrm{lv}_x(u_1) = \mathrm{lv}_x(t) \leq K$, so that the *i.h.* gives either (1) $o \in \mathrm{C}(u_1)_\mathrm{b}$, (2) $o \in \mathrm{C}(u_1)_\mathrm{a}^{>K_1}$, or (3) $o = \mathrm{a}(k, n)$ where $k \leq K_1$. If (1) holds, then $o \in \mathrm{C}(t)_\mathrm{b}$ and we are done. Otherwise $o = \mathrm{a}(k, n)$ and we consider two cases.

1. $k > K$. Then (2) implies $o \in \mathrm{C}(u_1)_\mathrm{a}^{>K}$, and thus $o \in \mathrm{C}(t)_\mathrm{a}^{>K}$, while (3) implies $k \leq K$ which leads to a contradiction.
2. $k \leq K$. We are done.

Subsubcase $o \in (\mathrm{lv}_y(u_1) + 1) \cdot \mathrm{C}(u_2\{x/p\})$. There is $o' \in \mathrm{C}(u_2\{x/p\})$ such that $\mathrm{fst}(o) = \mathrm{fst}(o') + (\mathrm{lv}_y(u_1) + 1)$. Let $K_2 = \mathrm{lv}_x(u_2)$, so that the *i.h.* gives either (1) $o' \in \mathrm{C}(u_2)_\mathrm{b}$, (2) $o' \in \mathrm{C}(u_2)_\mathrm{a}^{>K_2}$, or (3) $o' = \mathrm{a}(k, n)$ where $k \leq K_2$. If (1) holds, then $o \in (\mathrm{lv}_y(u_1) + 1) \cdot \mathrm{C}(u_2)_\mathrm{b}$, thus $o \in \mathrm{C}(t)_\mathrm{b}$ and we are done. If (2)

holds, then $o \in (\mathrm{lv}_y(u_1)+1)\cdot \mathrm{C}\,(u_2)_{\mathrm{a}}^{>K_2}$ and thus $\mathrm{fst}(o) > K_2 + (\mathrm{lv}_y(u_1)+1)$. We consider two cases.

1. $\mathrm{fst}(o) > K \geq K_2 + \mathrm{lv}_y(u_1) + 1$. Then (2) implies $o \in \mathrm{C}\,(t)_{\mathrm{a}}^{>K}$ while (3) leads to a contradiction.
2. $\mathrm{fst}(o) \leq K$. We are done.

Subsubcase $o = \mathrm{a}(\mathrm{lv}_y(u_1)+1, |u_2\{x/p\}|)$. Then $\mathrm{fst}(o) = \mathrm{lv}_y(u_1)+1 \leq K$.

Case $t = u_1[y/\!\!/u_2]$. The analysis is similar. □

Lemma 2.12. *Let* $t \in \mathtt{T}_R$. *Then* $t \longrightarrow_\rho t'$ *implies* $\mathrm{C}\,(t) >_{\mathcal{O}_{MUL}} \mathrm{C}\,(t')$.

Proof. Let $t = \mathtt{C}\langle t_0\rangle \longrightarrow_\rho \mathtt{C}\langle t_1\rangle = t'$, where $t_0 \longrightarrow_\rho t_1$ is a reduction step at the root position. We proceed by induction on $\mathtt{C}$. We detail the base case where $\mathtt{C} = \Diamond$, by inspecting the cases where the explicit cuts are explicit substitutions, as the remaining cases for explicit distributors are similar.

Case $t_0 = \lambda y.t[x/u] \longrightarrow_\rho (\lambda y.t)[x/u] = t_1$, where $y \notin \mathrm{fv}(u)$.

$$\begin{aligned}\mathrm{C}\,(t_0) &= \mathrm{C}\,(t[x/u])\\ &= \mathrm{C}\,(t) \sqcup (\mathrm{lv}_x(t)+1)\cdot \mathrm{C}\,(u) \sqcup [\mathrm{a}(\mathrm{lv}_x(t)+1, |u|)]\\ &= \mathrm{C}\,(\lambda y.t) \sqcup (\mathrm{lv}_x(\lambda y.t)+1)\cdot \mathrm{C}\,(u) \sqcup [\mathrm{a}(\mathrm{lv}_x(\lambda y.t)+1, |u|)]\\ &= \mathrm{C}\,(t_1)\end{aligned}$$

Case $t_0 = t[x/u]s \longrightarrow_\rho (ts)[x/u] = t_1$, where $x \notin \mathrm{fv}(s)$.

$$\begin{aligned}\mathrm{C}\,(t_0) &= \mathrm{C}\,(t[x/u]) \sqcup \mathrm{C}\,(s)\\ &= \mathrm{C}\,(t) \sqcup (\mathrm{lv}_x(t)+1)\cdot \mathrm{C}\,(u) \sqcup \mathrm{C}\,(s)\\ &= \mathrm{C}\,(ts) \sqcup (\mathrm{lv}_x(ts)+1)\cdot \mathrm{C}\,(u) \sqcup [\mathrm{a}(\mathrm{lv}_x(ts)+1, |u|)]\\ &= \mathrm{C}\,(t_1)\end{aligned}$$

Case $t_0 = ts[x/u] \longrightarrow_\rho (ts)[x/u] = t_1$, where $x \notin \mathrm{fv}(t)$.

$$\begin{aligned}\mathrm{C}\,(t_0) &= \mathrm{C}\,(t) \sqcup \mathrm{C}\,(s[x/u])\\ &= \mathrm{C}\,(t) \sqcup \mathrm{C}\,(s) \sqcup (\mathrm{lv}_x(s)+1)\cdot \mathrm{C}\,(u) \sqcup [\mathrm{a}(\mathrm{lv}_x(s)+1, |u|)]\\ &= \mathrm{C}\,(ts) \sqcup (\mathrm{lv}_x(ts)+1)\cdot \mathrm{C}\,(u) \sqcup [\mathrm{a}(\mathrm{lv}_x(ts)+1, |u|)]\\ &= \mathrm{C}\,(t_1)\end{aligned}$$

Case $t_0 = t[y/s[x/u]] \longrightarrow_\rho t[y/s][x/u] = t_1$, where $x \notin \mathrm{fv}(t)$.

$$\begin{aligned}
\mathrm{C}(t_0) &= \mathrm{C}(t) \sqcup (\mathrm{lv}_y(t)+1)\cdot \mathrm{C}(s[x/u]) \sqcup [\mathrm{a}(\mathrm{lv}_y(t)+1, |s[x/u]|)] \\
&= \mathrm{C}(t) \sqcup (\mathrm{lv}_y(t)+1)\cdot (\mathrm{C}(s) \sqcup (\mathrm{lv}_x(s)+1)\cdot \mathrm{C}(u) \sqcup [\mathrm{a}(\mathrm{lv}_x(s)+1, |u|)]) \\
&\quad \sqcup [\mathrm{a}(\mathrm{lv}_y(t)+1, |s[x/u]|)] \\
&= \mathrm{C}(t) \sqcup (\mathrm{lv}_y(t)+1)\cdot \mathrm{C}(s) \sqcup (\mathrm{lv}_y(t)+\mathrm{lv}_x(s)+2)\cdot \mathrm{C}(u) \\
&\quad \sqcup [\mathrm{a}(\mathrm{lv}_y(t)+\mathrm{lv}_x(s)+2, |u|), \mathrm{a}(\mathrm{lv}_y(t)+1, |s[x/u]|)] \\
&= \left(\mathrm{C}(t) \sqcup (\mathrm{lv}_y(t)+1)\cdot \mathrm{C}(s) \sqcup [\mathrm{a}(\mathrm{lv}_y(t)+1, |s[x/u]|)]\right) \\
&\quad \sqcup (\mathrm{lv}_y(t)+\mathrm{lv}_x(s)+2)\cdot \mathrm{C}(u) \sqcup [\mathrm{a}(\mathrm{lv}_y(t)+\mathrm{lv}_x(s)+2, |u|)] \\
&>^{\mathcal{O}}_{\mathtt{MUL}} \left(\mathrm{C}(t) \sqcup (\mathrm{lv}_y(t)+1)\cdot \mathrm{C}(s) \sqcup [\mathrm{a}(\mathrm{lv}_y(t)+1, |s|)]\right) \\
&\quad \sqcup (\mathrm{lv}_x(t[y/s])+1)\cdot \mathrm{C}(u) \sqcup [\mathrm{a}(\mathrm{lv}_x(t[y/s])+1, |u|)] \\
&= \mathrm{C}(t_1)
\end{aligned}$$

The $>^{\mathcal{O}}_{\mathtt{MUL}}$ inequality is justified by the following facts:

1. $|s[x/u]| > |s|$.
2. $\mathrm{lv}_y(t) + \mathrm{lv}_x(s) + 2 = \max(0, \mathrm{lv}_y(t) + \mathrm{lv}_x(s) + 1) + 1 = \mathrm{lv}_x(t[y/s]) + 1$.

The inductive cases are straightforward. □

Lemma 2.13. *Let $t \in \mathtt{T}_R$. Then $t \longrightarrow_{\mathrm{sub}} t'$ implies $\mathrm{C}(t) >^{\mathcal{O}}_{\mathtt{MUL}} \mathrm{C}(t')$.*

Proof. Let $t = \mathtt{C}\langle t_0\rangle \longrightarrow_{\mathrm{sub}} \mathtt{C}\langle t_1\rangle = t'$, where $t_0 \longrightarrow_{\mathrm{sub}} t_1$ is a reduction step at the root position. We proceed by induction on $\mathtt{C}$. We detail the base case which is $\mathtt{C} = \Diamond$. In all such cases we use lemma 2.12 to push $\mathtt{L}$ outside, *i.e.* we can write $t_0 \longrightarrow_{\mathrm{sub}} t_1$ as $t_0 \longrightarrow_\pi \mathtt{L}\langle t_0'\rangle \longrightarrow_{\mathrm{sub}'} \mathtt{L}\langle t_1'\rangle = t_1$, where $t_0' \longrightarrow_{\mathrm{sub}'} t_1'$ is a root step which does not push any list context outside. We then show the property for root steps $t_0' \longrightarrow_{\mathrm{sub}'} t_1'$, and we conclude with lemma 2.12 then lemma 2.10 by $\mathrm{C}(t_0) >^{\mathcal{O}}_{\mathtt{MUL}} \mathrm{C}\left(\mathtt{L}\langle t_0'\rangle\right) >^{\mathcal{O}}_{\mathtt{MUL}} \mathrm{C}\left(\mathtt{L}\langle t_1'\rangle\right) = \mathrm{C}(t_1)$ since $\mathrm{lv}_x(t_0') \geq \mathrm{lv}_x(t_1')$ holds for every $x \in \mathrm{dom}(\mathtt{L})$ by lemma 2.6. Let us analyze all the cases $t_0' \longrightarrow_{\mathrm{sub}'} t_1'$.

Case $t_0' = t[x/us] \longrightarrow_{\mathrm{app}} t\{x/yz\}[y/u][z/s] = t_1'$, where y and z are fresh variables. Then

$$\mathrm{C}\left(t_0'\right) = \mathrm{C}(t) \sqcup (\mathrm{lv}_x(t)+1)\cdot \mathrm{C}(us) \sqcup [\mathrm{a}(\mathrm{lv}_x(t)+1, |us|)] \text{ and}$$

$$\begin{aligned}
\mathrm{C}\left(t_1'\right) &= \mathrm{C}\left(t\{x/yz\}[y/u]\right) \sqcup \left(\mathrm{lv}_z(t\{x/yz\}[y/u]) + 1\right) \cdot \mathrm{C}\left(s\right) \\
&\quad \sqcup \left[\mathrm{a}(\mathrm{lv}_z(t\{x/yz\}[y/u]) + 1, |s|)\right] \\
&= \left(\mathrm{C}\left(t\{x/yz\}\right) \sqcup \left(\mathrm{lv}_y(t\{x/yz\}) + 1\right) \cdot \mathrm{C}\left(u\right) \sqcup \left[\mathrm{a}(\mathrm{lv}_y(t\{x/yz\}) + 1, |u|)\right]\right) \\
&\quad \sqcup \left(\mathrm{lv}_z(t\{x/yz\}[y/u]) + 1\right) \cdot \mathrm{C}\left(s\right) \sqcup \left[\mathrm{a}(\mathrm{lv}_z(t\{x/yz\}[y/u]) + 1, |s|)\right] \\
&= \left(\mathrm{C}\left(t\{x/yz\}\right) \sqcup \left(\mathrm{lv}_x(t) + 1\right) \cdot \mathrm{C}\left(u\right) \sqcup \left[\mathrm{a}(\mathrm{lv}_x(t) + 1, |u|)\right]\right) \\
&\quad \sqcup \left(\mathrm{lv}_x(t) + 1\right) \cdot \mathrm{C}\left(s\right) \sqcup \left[\mathrm{a}(\mathrm{lv}_x(t) + 1, |s|)\right] \\
&= \mathrm{C}\left(t\{x/yz\}\right) \sqcup \left(\mathrm{lv}_x(t) + 1\right) \cdot \mathrm{C}\left(us\right) \sqcup \left[\mathrm{a}(\mathrm{lv}_x(t) + 1, |u|), \mathrm{a}(\mathrm{lv}_x(t) + 1, |s|)\right]
\end{aligned}$$

By lemma 2.11, $\mathrm{C}\left(t\{x/yz\}\right) \sqsubseteq \mathrm{C}\left(t\right)_{\mathrm{b}} \sqcup \mathrm{C}\left(t\right)_{\mathrm{a}}^{>\mathrm{lv}_x(t)} \sqcup \left[\mathrm{a}(k,n) \mid k \leq \mathrm{lv}_x(t)\right]$. We also have $\left[\mathrm{a}(\mathrm{lv}_x(t) + 1, |us|)\right] >_{\mathtt{MUL}}^{\mathcal{O}} \left[\mathrm{a}(\mathrm{lv}_x(t) + 1, |u|), \mathrm{a}(\mathrm{lv}_x(t) + 1, |s|)\right] >_{\mathtt{MUL}}^{\mathcal{O}} \left[\mathrm{a}(k,n) \mid k \leq \mathrm{lv}_x(t)\right]$. Moreover, $\mathrm{C}\left(t\right) \sqsupseteq \mathrm{C}\left(t\right)_{\mathrm{b}} \sqcup \mathrm{C}\left(t\right)_{\mathrm{a}}^{>\mathrm{lv}_x(t)}$ and $\left[\mathrm{a}(\mathrm{lv}_x(t) + 1, |us|)\right] >_{\mathtt{MUL}}^{\mathcal{O}} \left[\mathrm{a}(k,n) \mid k \leq \mathrm{lv}_x(t)\right]$. Thus we conclude $\mathrm{C}\left(t_0'\right) >_{\mathtt{MUL}}^{\mathcal{O}} \mathrm{C}\left(t_1'\right)$.

Case $t_0' = t[x/\lambda y.u] \longrightarrow_{\mathrm{dist}} t[x/\!/\lambda y.z[z/u]] = t_1'$. Then

$$\mathrm{C}\left(t_0'\right) = \mathrm{C}\left(t\right) \sqcup \left(\mathrm{lv}_x(t) + 1\right) \cdot \mathrm{C}\left(u\right) \sqcup \left[\mathrm{a}(\mathrm{lv}_x(t) + 1, |u| + 1)\right] \text{ and}$$

$$\begin{aligned}
\mathrm{C}\left(t_1'\right) &= \mathrm{C}\left(t\right) \sqcup \mathrm{lv}_x(t) \cdot \mathrm{C}\left(\lambda y.z[z/u]\right) \sqcup \left[\mathrm{b}(\mathrm{lv}_x(t))\right] \\
&= \mathrm{C}\left(t\right) \sqcup \mathrm{lv}_x(t) \cdot \mathrm{C}\left(z[z/u]\right) \sqcup \left[\mathrm{b}(\mathrm{lv}_x(t))\right] \\
&= \mathrm{C}\left(t\right) \sqcup \mathrm{lv}_x(t) \cdot \left(\mathrm{C}\left(z\right) \sqcup \left(\mathrm{lv}_z(z) + 1\right) \cdot \mathrm{C}\left(u\right) \sqcup \left[\mathrm{a}(\mathrm{lv}_z(z) + 1, |u|)\right]\right) \sqcup \left[\mathrm{b}(\mathrm{lv}_x(t))\right] \\
&= \mathrm{C}\left(t\right) \sqcup \mathrm{lv}_x(t) \cdot \left(1 \cdot \mathrm{C}\left(u\right) \sqcup \left[\mathrm{a}(1, |u|)\right]\right) \sqcup \left[\mathrm{b}(\mathrm{lv}_x(t))\right] \\
&= \mathrm{C}\left(t\right) \sqcup \left(\mathrm{lv}_x(t) + 1\right) \cdot \mathrm{C}\left(u\right) \sqcup \left[\mathrm{a}(\mathrm{lv}_x(t) + 1, |u|), \mathrm{b}(\mathrm{lv}_x(t))\right]
\end{aligned}$$

$\mathrm{C}\left(t_0'\right) >_{\mathtt{MUL}}^{\mathcal{O}} \mathrm{C}\left(t_1'\right)$ because the multisets are the same except for $\mathrm{a}(\mathrm{lv}_x(t) + 1, |u|)$ and $\mathrm{b}(\mathrm{lv}_x(t))$ on the right which are smaller than $\mathrm{a}(\mathrm{lv}_x(t) + 1, |u| + 1)$ on the left.

Case $t_0' = t[x/\!/\lambda y.u] \longrightarrow_{\mathrm{abs}} t\{x/\lambda y.u\} = t_1'$. Then we have $\mathrm{C}\left(t_0'\right) = \mathrm{C}\left(t\right) \sqcup \left[\mathrm{b}(\mathrm{lv}_x(t))\right]$. By lemma 2.11, $\mathrm{C}\left(t_1'\right) \sqsubseteq \mathrm{C}\left(t\right)_{\mathrm{b}} \sqcup \mathrm{C}\left(t\right)^{>\mathrm{lv}_x(t)} \sqcup \left[\mathrm{a}(k,n) \mid k \leq \mathrm{lv}_x(t)\right]$. Since $\left[\mathrm{b}(\mathrm{lv}_x(t))\right] >_{\mathtt{MUL}}^{\mathcal{O}} \left[\mathrm{a}(k,n) \mid k \leq \mathrm{lv}_x(t)\right]$ and $\mathrm{C}\left(t\right) \sqsupseteq \mathrm{C}\left(t\right)_{\mathrm{b}} \sqcup \mathrm{C}\left(t\right)^{>\mathrm{lv}_x(t)}$, then we conclude $\mathrm{C}\left(t_0'\right) >_{\mathtt{MUL}}^{\mathcal{O}} \mathrm{C}\left(t_1'\right)$.

Case $t_0' = t[x/y] \longrightarrow_{\mathrm{var}} t\{x/y\} = t_1'$. Then, $\mathrm{C}\left(t_0'\right) = \mathrm{C}\left(t\right) \sqcup \left[\mathrm{a}(\mathrm{lv}_x(t) + 1, 1)\right]$. By lemma 2.11, $\mathrm{C}\left(t_1'\right) \sqsubseteq \mathrm{C}\left(t\right)_{\mathrm{b}} \sqcup \mathrm{C}\left(t\right)_{\mathrm{a}}^{>\mathrm{lv}_x(t)} \sqcup \left[\mathrm{a}(k,n) \mid k \leq \mathrm{lv}_x(t)\right]$. Since $\left[\mathrm{a}(\mathrm{lv}_x(t) + 1, 1)\right] >_{\mathtt{MUL}}^{\mathcal{O}} \left[\mathrm{a}(k,n) \mid k \leq \mathrm{lv}_x(t)\right]$ and $\mathrm{C}\left(t\right) \sqsupseteq \mathrm{C}\left(t\right)_{\mathrm{b}} \sqcup \mathrm{C}\left(t\right)^{>\mathrm{lv}_x(t)}$, we conclude $\mathrm{C}\left(t_0'\right) >_{\mathtt{MUL}}^{\mathcal{O}} \mathrm{C}\left(t_1'\right)$ □

The sequence of example 2.8 illustrates this phenomenon: indeed, $\mathrm{C}\left(t_i\right) >_{\mathtt{MUL}}^{\mathcal{O}} \mathrm{C}\left(t_{i+1}\right)$ for $0 \leq i < 5$.

Corollary 2.14. *The reduction relation* $\longrightarrow_{\mathrm{sub}}$ *is terminating.*

Simulations. We show the relation between λR and λ, as well as the atomic λ-calculus λa. For that, we introduce a projection from $\mathtt{T}_R$ to $\mathtt{T}_P$ implementing the unfolding of all the explicit cuts:

$$x^{\downarrow} \coloneqq x \quad (\lambda x.t)^{\downarrow} \coloneqq \lambda x.t^{\downarrow} \quad (tu)^{\downarrow} \coloneqq t^{\downarrow}u^{\downarrow} \quad (t[x \triangleleft u])^{\downarrow} \coloneqq t^{\downarrow}\{x/u^{\downarrow}\}.$$

Thus *e.g.* $x[x/z[y/w]][w/w']^{\downarrow} = x\{x/z\{y/w\}\}\{w/w'\} = z$. The previous projection can be extended from list contexts to substitutions as follows: $\Diamond^{\downarrow} \coloneqq \{\}$ and $(\mathtt{L}[x \triangleleft u])^{\downarrow} \coloneqq \mathtt{L}^{\downarrow} \circ \{x/u^{\downarrow}\}$, where $\circ$ denotes standard composition of substitutions.

Lemma 2.15. *Let $t \in \mathtt{T}_R$. If $t \longrightarrow_{\mathrm{R}} t'$, then $t^{\downarrow} \longrightarrow_{\beta}^{*} t'^{\downarrow}$. In particular, if either $t \longrightarrow_{\rho} t'$ or $t \longrightarrow_{\mathrm{sub}} t'$, then $t^{\downarrow} = t'^{\downarrow}$.*

Proof. The proofs of the corresponding stated relations are by induction on them.

Case $t \longrightarrow_{\rho} t'$. Then $t = \mathtt{C}\langle t_0\rangle \longrightarrow_{\rho} \mathtt{C}\langle t_1\rangle = t'$, where $t_0 \longrightarrow_{\rho} t_1$ is a root step. If $\mathtt{C} = \Diamond$ we have the following cases:

- $(\lambda y.t[x \triangleleft u])^{\downarrow} = \lambda y.t^{\downarrow}\{x/u^{\downarrow}\} = (\lambda y.t^{\downarrow})\{x/u^{\downarrow}\} = ((\lambda y.t)[x \triangleleft u])^{\downarrow}$
- $(t[x \triangleleft u]v)^{\downarrow} = t^{\downarrow}\{x/u^{\downarrow}\}v^{\downarrow} = (t^{\downarrow}v^{\downarrow})\{x/u^{\downarrow}\} = ((tv)[x \triangleleft u])^{\downarrow}$
- $(tv[x \triangleleft u])^{\downarrow} = t^{\downarrow}v^{\downarrow}\{x/u^{\downarrow}\} = (t^{\downarrow}v^{\downarrow})\{x/u^{\downarrow}\} = ((tv)[x \triangleleft u])^{\downarrow}$
- $(t[y \triangleleft v[x \triangleleft u]])^{\downarrow} = t^{\downarrow}\{y/v^{\downarrow}\{x/u^{\downarrow}\}\} = t^{\downarrow}\{y/v^{\downarrow}\}\{x/u^{\downarrow}\} = (t[y \triangleleft v][x \triangleleft u])^{\downarrow}$

For the inductive cases, we reason as follows.

- If $\mathtt{C} = \lambda x.\mathtt{C}'$, then $(\lambda x.\mathtt{C}'\langle t_0\rangle)^{\downarrow} = \lambda x.(\mathtt{C}'\langle t_0\rangle)^{\downarrow} =_{i.h.} \lambda x.(\mathtt{C}'\langle t_1\rangle)^{\downarrow} = (\lambda x.\mathtt{C}'\langle t_1\rangle)^{\downarrow}$.
- If $\mathtt{C} = \mathtt{C}'u$, then $(\mathtt{C}'\langle t_0\rangle u)^{\downarrow} = (\mathtt{C}'\langle t_0\rangle)^{\downarrow}u^{\downarrow} =_{i.h.} (\mathtt{C}'\langle t_1\rangle)^{\downarrow}u^{\downarrow} = (\mathtt{C}'\langle t_1\rangle u)^{\downarrow}$.
- If $\mathtt{C} = u\mathtt{C}'$, then $(u\mathtt{C}'\langle t_0\rangle)^{\downarrow} = u^{\downarrow}(\mathtt{C}'\langle t_0\rangle)^{\downarrow} =_{i.h.} u^{\downarrow}(\mathtt{C}'\langle t_1\rangle)^{\downarrow} = (u\mathtt{C}'\langle t_1\rangle)^{\downarrow}$.
- If $\mathtt{C} = \mathtt{C}'[x \triangleleft u]$, then $(\mathtt{C}'\langle t_0\rangle[x \triangleleft u])^{\downarrow} = (\mathtt{C}'\langle t_0\rangle)^{\downarrow}\{x/u^{\downarrow}\} =_{i.h.} (\mathtt{C}'\langle t_1\rangle)^{\downarrow}\{x/u^{\downarrow}\} = (\mathtt{C}'\langle t_1\rangle[x \triangleleft u])^{\downarrow}$.
- If $\mathtt{C} = u[x \triangleleft \mathtt{C}']$, then $(u[x \triangleleft \mathtt{C}'\langle t_0\rangle])^{\downarrow} = u^{\downarrow}\{x/(\mathtt{C}')\langle t_0\rangle^{\downarrow}\} =_{i.h.} u^{\downarrow}\{x/(\mathtt{C}'\langle t_1\rangle)^{\downarrow}\} = (u[x \triangleleft \mathtt{C}'\langle t_1\rangle])^{\downarrow}$.

Case $t \longrightarrow_{\mathrm{sub}} t'$. Then $t = \mathtt{C}\langle t_0\rangle \longrightarrow_{\mathrm{sub}} \mathtt{C}\langle t_1\rangle = t'$, where $t_0 \longrightarrow_{\mathrm{sub}} t_1$ is a root step. We first consider $\mathtt{C} = \Diamond$. Let us call $\sigma_{\mathtt{L}}$ the substitution resulting from translating the list context $\mathtt{L}$. We use the last point on ρ to push out list contexts in these equations.

Subcase $t_0 = t[x/\mathtt{L}\langle uv\rangle] \longrightarrow \mathtt{L}\langle t\{x/yz\}[y/u][z/v]\rangle = t_1$. Then,

$$\begin{aligned}(t[x/\mathtt{L}\langle uv\rangle])^{\downarrow} &= (\mathtt{L}\langle t[x/uv]\rangle)^{\downarrow} = \sigma_{\mathtt{L}}(t^{\downarrow}\{x/u^{\downarrow}v^{\downarrow}\}) = \sigma_{\mathtt{L}}(t^{\downarrow}\{x/yz\}\{y/u^{\downarrow}\}\{z/v^{\downarrow}\})\\ &= \sigma_{\mathtt{L}}((t\{x/yz\}[y/u][z/v])^{\downarrow}) = (\mathtt{L}\langle t\{x/yz\}[y/u][z/v]\rangle)^{\downarrow}\end{aligned}$$

Subcase $t_0 = t[x/\mathtt{L}\langle\lambda y.u\rangle] \longrightarrow \mathtt{L}\langle t[x/\!\!/\lambda y.z[z/u]]\rangle = t_1$. Then,

$$\begin{aligned}(t[x/\mathtt{L}\langle\lambda y.u\rangle])^{\downarrow} = (\mathtt{L}\langle t[x/\lambda y.u]\rangle)^{\downarrow} = \sigma_{\mathtt{L}}(t^{\downarrow}\{x/\lambda y.u^{\downarrow}\}) = \sigma_{\mathtt{L}}(t^{\downarrow}\{x/\lambda y.z\{z/u^{\downarrow}\}\})\\ = \sigma_{\mathtt{L}}((t[x/\!\!/\lambda y.z[z/u]])^{\downarrow}) = (\mathtt{L}\langle t[x/\!\!/\lambda y.z[z/u]]\rangle)^{\downarrow}\end{aligned}$$

Subcase $t_0 = t[x/\!\!/\lambda y.u] \longrightarrow \mathtt{L}\langle t\{x/\lambda y.u'\}\rangle = t_1$, where $u \longrightarrow_{\rho} \mathtt{L}\langle u'\rangle$ and u' pure. Then

$$\begin{aligned}(t[x/\!\!/\lambda y.u])^{\downarrow} = (t[x/\!\!/\mathtt{L}\langle\lambda y.u'\rangle])^{\downarrow} = (\mathtt{L}\langle t[x/\!\!/\lambda y.u']\rangle)^{\downarrow} = \sigma_{\mathtt{L}}(t^{\downarrow}\{x/\lambda y.u'^{\downarrow}\})\\ = \sigma_{\mathtt{L}}((t\{x/\lambda y.u'\})^{\downarrow}) = (\mathtt{L}\langle t\{x/\lambda y.u'\}\rangle)^{\downarrow}\end{aligned}$$

Subcase $t_0 = t[x/\mathtt{L}\langle y\rangle] \longrightarrow \mathtt{L}\langle t\{x/y\}\rangle = t_1$. Then

$$\begin{aligned}(t[x/\mathtt{L}\langle y\rangle])^{\downarrow} = (\mathtt{L}\langle t[x/y]\rangle)^{\downarrow} = \sigma_{\mathtt{L}}((t[x/y])^{\downarrow}) = \sigma_{\mathtt{L}}(t^{\downarrow}\{x/y\})\\ = \sigma_{\mathtt{L}}((t\{x/y\})^{\downarrow}) = (\mathtt{L}\langle t\{x/y\}\rangle)^{\downarrow}\end{aligned}$$

The proof of the inductive cases is similar to the previous case.

Case $t \longrightarrow_{\mathrm{dB}} t'$. Then $t = \mathtt{C}\langle t_0\rangle \longrightarrow_{\mathrm{dB}} \mathtt{C}\langle t_1\rangle = t'$, where $t_0 \longrightarrow_{\mathrm{dB}} t_1$ is a root step. We first consider $\mathtt{C} = \Diamond$. As before, $\sigma_{\mathtt{L}}$ is the substitution resulting from translating the list context $\mathtt{L}$.
Then we have $(\mathtt{L}\langle\lambda x.t\rangle u)^{\downarrow} = \sigma_{\mathtt{L}}(((\lambda x.t)u)^{\downarrow}) = \sigma_{\mathtt{L}}((\lambda x.t^{\downarrow})u^{\downarrow}) \longrightarrow_{\beta} \sigma_{\mathtt{L}}(t^{\downarrow}\{x/u^{\downarrow}\}) = \sigma_{\mathtt{L}}((t[x/u])^{\downarrow}) = (\mathtt{L}\langle t[x/u]\rangle)^{\downarrow}$
For the inductive cases, we reason as follows.

- If $\mathtt{C} = \lambda x.\mathtt{C}'$, then $(\lambda x.\mathtt{C}'\langle t_0\rangle)^{\downarrow} = \lambda x.(\mathtt{C}'\langle t_0\rangle)^{\downarrow} \longrightarrow_{i.h.} \lambda x.(\mathtt{C}'\langle t_1\rangle)^{\downarrow} = (\lambda x.\mathtt{C}'\langle t_1\rangle)^{\downarrow}$.
- If $\mathtt{C} = \mathtt{C}'u$, then $(\mathtt{C}'\langle t_0\rangle u)^{\downarrow} = (\mathtt{C}'\langle t_0\rangle)^{\downarrow}u^{\downarrow} \longrightarrow_{i.h.} (\mathtt{C}'\langle t_1\rangle)^{\downarrow}u^{\downarrow} = (\mathtt{C}'\langle t_1\rangle u)^{\downarrow}$.
- If $\mathtt{C} = u\mathtt{C}'$, then $(u\mathtt{C}'\langle t_0\rangle)^{\downarrow} = u^{\downarrow}(\mathtt{C}'\langle t_0\rangle)^{\downarrow} \longrightarrow_{i.h.} u^{\downarrow}(\mathtt{C}'\langle t_1\rangle)^{\downarrow} = (u\mathtt{C}'\langle t_1\rangle)^{\downarrow}$.
- If $\mathtt{C} = \mathtt{C}'[x \triangleleft u]$, then $(\mathtt{C}'\langle t_0\rangle[x \triangleleft u])^{\downarrow} = (\mathtt{C}'\langle t_0\rangle)^{\downarrow}\{x/u^{\downarrow}\} \longrightarrow_{i.h.} (\mathtt{C}'\langle t_1\rangle)^{\downarrow}\{x/u^{\downarrow}\} = (\mathtt{C}'\langle t_1\rangle[x \triangleleft u])^{\downarrow}$.
- If $\mathtt{C} = u[x \triangleleft \mathtt{C}']$, then:
  - if $x \notin \mathrm{fv}(u)$: $(u[x \triangleleft \mathtt{C}'\langle t_0\rangle])^{\downarrow} = u^{\downarrow}\{x/\mathtt{C}'\langle t_0\rangle\} = u^{\downarrow} = (u[x \triangleleft \mathtt{C}'\langle t_1\rangle])^{\downarrow}$,
  - otherwise: $(u[x \triangleleft \mathtt{C}'\langle t_0\rangle])^{\downarrow} = u^{\downarrow}\{x/\mathtt{C}'\langle t_0\rangle\} \longrightarrow_{i.h.} u^{\downarrow}\{x/\mathtt{C}'\langle t_1\rangle\} = (u[x \triangleleft \mathtt{C}'\langle t_1\rangle])^{\downarrow}$. □

The relation $\longrightarrow_{\mathrm{sub}}$ enjoys **full composition** on *pure* terms. Namely:

Lemma 2.16. *For any $p \in \mathtt{T}_P$, $t[x/p] \longrightarrow_{\mathrm{sub}}^{+} t\{x/p\}$.*

Proof. By induction on p.

Case $p = y$. Then $t[x/y] \longrightarrow_{\mathrm{var}} t\{x/y\}$.

Case $p = p_1 p_2$. Then

$$\begin{aligned} t[x/p_1p_2] &\to_{\text{app}} t\{x/yz\}[y/p_1][z/p_2] \\ &\to^+_{i.h.} t\{x/yz\}\{y/p_1\}[z/p_2] \to^+_{i.h.} t\{x/yz\}\{y/p_1\}\{z/p_2\} \\ &= t\{x/p_1p_2\} \end{aligned}$$

Case $p = \lambda y.q$. Then $t[x/\lambda y.q] \to_{\text{abs}} t[x/\!/\lambda y.z[z/q]] \to^+_{i.h.} t[x/\!/\lambda y.q] \to_{\text{dist}} t\{x/\lambda y.q\}$. □

This property does not hold in general. Indeed, if $t = xx$, then $(xx)[x/y[y/z]]$ does not sub-reduce to $(y[y/z])(y[y/z])$, but to $(yy)[y/z]$. However, full composition restricted to pure terms is sufficient to prove simulation of the λ-calculus.

Lemma 2.17 (Simulation of the λ-calculus). *Let $p_0 \in \mathtt{T}_P$. If $p_0 \to_\beta p_1$, then $p_0 \to_{\text{dB}} \to^+_{\text{sub}} p_1$.*

Proof. Let $p_0 = \mathtt{C}\langle t_0\rangle \to_\beta \mathtt{C}\langle t_1\rangle = p_1$, where $t_0 = (\lambda x.q)p \mapsto_\beta q\{x/p\} = t_1$. By lemma 2.16, $t_0 \to_{\text{dB}} q[x/p] \to^+_{\text{sub}} t_1$. The inductive cases for $\mathtt{C}$ are straightforward. □

The previous results have an important consequence relating the atomic λ-calculus and the λR-calculus. Indeed, it can be shown that reduction in the atomic λ-calculus is captured by λa, and vice-versa. More precisely, the λR-calculus can be simulated into the atomic λ-calculus by lemma 2.15 and [GHP13b], while the converse holds by [GHP13b] and lemma 2.17.

However, this indirect result is vague, as it erases the specificities of the atomic and node replication calculi when going through the λ-calculus. We do not yet have a side-by-side comparison between both calculi. To this end, a more structural correspondence between λR and λa could be established. Indeed, λR can be first refined into a (non-linear) calculus *without* distance, let say $\lambda R'$, so that permutation rules are integrated in the intermediate calculus as independent rules. Then a structural relation can be established between λR and $\lambda R'$ on one side, and $\lambda R'$ and the atomic λ-calculus on the other side (as for example done in [KL07] for the λ-calculus).

Confluence. By corollary 2.14 the reduction relation $\to_{\text{sub}}$ is terminating. It is then not difficult to prove confluence of $\to_{\text{sub}}$ by using the unfolding function $\cdot^{\downarrow}$.

Lemma 2.18. *Let $t \in \mathtt{T}_R$. Then t is in* sub*-nf if and only if t is pure.*

Proof. It is obvious that a pure term is sub-normal. Let us show the left-to-right implication and consider a sub-normal term t. We reason by induction on t. Suppose that t is not pure, so that $t = \mathtt{C}\langle t_0[x \triangleleft u]\rangle$. If the explicit cut is an explicit substitution, then one of the rules app, dist, var apply, which contradicts the hypothesis. Otherwise the cut is a distributor, and u is an abstraction $\lambda y.u'$, where u' is in particular a sub-normal form. By the *i.h.* u' is pure so that the rule abs applies, which contradicts the hypothesis

again. □

Corollary 2.19. *Let $t \in \mathtt{T}_R$. If t is in* sub*-nf, then $t^{\downarrow} = t$.*

Lemma 2.20. *The reduction relation $\longrightarrow_{\mathrm{sub}}$ is terminating and confluent.*

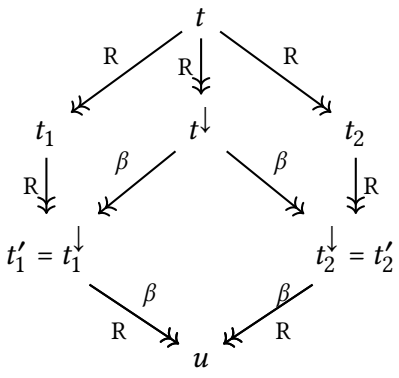
Proof. Termination holds by corollary 2.14. For confluence, suppose $t \longrightarrow^*_{\mathrm{sub}} t_1$ and $t \longrightarrow^*_{\mathrm{sub}} t_2$. Let $t_1 \longrightarrow^*_{\mathrm{sub}} t_1'$ and $t_2 \longrightarrow^*_{\mathrm{sub}} t_2'$, where t_1' and t_2' are in sub-nf. Then by corollary 2.19, $(t_i')^{\downarrow} = t_i'$ for both $i = 1, 2$. By lemma 2.15, $(t_i')^{\downarrow} = t_i^{\downarrow} = t^{\downarrow}$ so that $t_1' = t_2'$, closing the diagram. □

By termination of $\longrightarrow_{\mathrm{sub}}$ any $t \in \mathtt{T}_R$ has a sub-nf, and by confluence this sub-nf is unique. By lemma 2.15 and corollary 2.19 one obtains:

Corollary 2.21. *Let $t \in \mathtt{T}_R$. Then the unique* sub*-nf of t is $t^{\downarrow}$.*

Theorem 2.22. *The reduction relation $\longrightarrow_{\mathrm{R}}$ is confluent.*

Proof. Let $t \in \mathtt{T}_R$ such that $t \longrightarrow^*_{\mathrm{R}} t_1$ and $t \longrightarrow^*_{\mathrm{R}} t_2$. By simulation (lemma 2.15), we have $t^{\downarrow} \longrightarrow^*_{\beta} t_1^{\downarrow}$ and $t^{\downarrow} \longrightarrow^*_{\beta} t_2^{\downarrow}$. By lemma 2.20, there exist t_1' (resp. t_2') the unique sub-nf of t_1 (resp. t_2). By corollary 2.21 we have $t_1' = t_1^{\downarrow}$ and $t_2' = t_2^{\downarrow}$. Because $\longrightarrow_{\beta}$ is confluent, there is u such that $t_1^{\downarrow} \longrightarrow^*_{\beta} u$ and $t_2^{\downarrow} \longrightarrow^*_{\beta} u$, and by lemma 2.17, $t_1^{\downarrow} \longrightarrow^*_{\mathrm{R}} u$ and $t_2^{\downarrow} \longrightarrow^*_{\mathrm{R}} u$. The diagram is then closed by $t_1 \longrightarrow^*_{\mathrm{sub}} t_1' = t_1^{\downarrow} \longrightarrow^*_{\mathrm{R}} u$ and $t_2 \longrightarrow^*_{\mathrm{sub}} t_2' = t_2^{\downarrow} \longrightarrow^*_{\mathrm{R}} u$. Graphically,



□

2.3 Encoding Evaluation Strategies

Although the atomic λ-calculus was introduced as a technical tool to implement full laziness, only its (non-deterministic) equational theory was studied. We bridge the gap between the theoretical presentation of the atomic λ-calculus and concrete specifications of evaluation

strategies. Indeed, we use the λR-calculus to investigate two concrete cases: a call-by-name strategy implementing weak head reduction, based on full substitution, and the call-by-need fully lazy strategy, which uses linear substitution.

In this work, we choose to implement full laziness for pure terms, that is, for the usual λ-calculus without cuts. Indeed, we see explicit cuts as a tool for a fully lazy implementation of the λ-calculus. We thus keep in line with the definitions found in the literature. Defining full laziness for terms with explicit cuts also brings technical difficulties, which might divert from the main point: using node replication to implement a fully lazy strategy.

We then restrict the set of terms to a subset $\mathtt{U}$, which simplifies the formal reasoning of explicit cuts inside distributors. Indeed, distributors will all be of the shape $[x/\!\!/\lambda y.\mathtt{LL}\langle p\rangle]$, where p is a pure term containing the constructors that have been (symbolically) shared in the distributor, and $\mathtt{LL}$ is a *commutative list* (defined below). We argue that this restriction is natural in a weak implementation of the λ-calculus: it is true on pure terms and is preserved through evaluation. We consider the following grammars; recall that $|p|_x$ is the number of occurrences of x in p.

(Linear Cut Values)	$\mathtt{T}$	::=	$\lambda x.\mathtt{LL}\langle p\rangle$ where $y \in \mathrm{dom}(\mathtt{LL}) \implies \|p\|_y = 1$
(Commutative Lists)	$\mathtt{LL}$	::=	$\diamond \mid \mathtt{LL}[x/p] \mid \mathtt{LL}[x/\!\!/\mathtt{T}]$, where in both cases $\|\mathtt{LL}\|_x = 0$
(Values)	v	::=	$\lambda x.p$
(Restricted Terms)	$\mathtt{U}$	::=	$x \mid v \mid \mathtt{UU} \mid \mathtt{U}[x/\mathtt{U}] \mid \mathtt{U}[x/\!\!/\mathtt{T}]$

A term t generated by any of the grammars G defined above is written $t \in G$. Thus *e.g.* $\lambda x.(yz)[y/\mathtt{I}][z/\mathtt{I}] \in \mathtt{T}$ but $\lambda x.(yy)[y/\mathtt{I}] \notin \mathtt{T}$, $\diamond[x/yz][x'/\mathtt{I}] \in \mathtt{LL}$ but $\diamond[x/yz][y/\mathtt{I}] \notin \mathtt{LL}$, and $(yz)[y/\!\!/\mathtt{I}] \in \mathtt{U}$ but $(yz)[y/\!\!/\lambda x.(yy)[y/\mathtt{I}]] \notin \mathtt{U}$.

The set $\mathtt{T}$ is stable by the relation $\longrightarrow_{\mathrm{sub}}$ (lemma 2.23), but $\mathtt{U}$ is clearly not stable under the whole $\longrightarrow_{\mathrm{R}}$ relation, where dB-reductions may occur under abstractions. For instance, let $t_1 = (yz)[y/\!\!/\lambda x.(\lambda y.yy)\mathtt{I}] \longrightarrow_{\mathrm{dB}} (yz)[y/\!\!/\lambda x.(yy)[y/\mathtt{I}]] = t_2$. Then $t_1 \in \mathtt{U}$ but $t_2 \notin \mathtt{U}$, since $|yy|_y = 2$. However, $\mathtt{U}$ is stable under both weak strategies to be defined: call-by-name and call-by-need. We factorize the proofs by proving stability for a more general relation $\longrightarrow_{\mathrm{R}'}$, defined as the relation $\longrightarrow_{\mathrm{R}}$ with dB-reductions forbidden under abstractions and inside distributors.

Lemma 2.23. *If $t \in T$ and $t \longrightarrow_{\mathrm{sub}} t'$, then $t' \in T$.*

Proof. We first show a more general statement, namely that $t = \mathtt{LL}_0\langle p_0\rangle$ with $|p_0|_y = 1$ for every $y \in \mathrm{dom}(\mathtt{LL}_0)$, and $t \longmapsto_{\mathrm{sub}} t'$ imply $t' = \mathtt{LL}_1\langle p_1\rangle$ with $|p_1|_y = 1$ for every $y \in \mathrm{dom}(\mathtt{LL}_1)$. In the following rules var, app and dist, there is no $\mathtt{L}$ context inside the explicit substitutions because lists in $\mathtt{LL}$ only contain pure terms by definition.

Case $t = u[x/z] \longmapsto_{\mathrm{var}} u\{x/z\} = t'$. This is straightforward.

Case $t = \mathtt{LL}\langle p\rangle[x/q_1q_2] \longmapsto_{\mathrm{app}} \mathtt{LL}\langle p\{x/x_1x_2\}\rangle[x_1/q_1][x_2/q_2] = t'$. Freshness of both x_1 and x_2 implies $|p\{x/x_1x_2\}|_{x_1} = |p\{x/x_1x_2\}|_{x_2} = |p|_x = 1$, and $|q_1|_{x_2} = 0$.

Case $t = \mathtt{LL}\langle p\rangle[x/\lambda z.p'] \longmapsto_{\mathrm{dist}} \mathtt{LL}\langle p\rangle[x/\!\!/\lambda z.w[w/p']] = t'$. By hypothesis $|p|_x = 1$, $|\mathtt{LL}|_x =$

0 and $\lambda z.p'$ is pure. Then, $\lambda z.w[w/p'] \in \mathtt{T}$ because p' is pure and $|w|_w = 1$.

Case $t = \mathtt{LL}\langle p\rangle[x /\!\!/ \lambda z.\mathtt{LL}'\langle p'\rangle] \mapsto_{\text{abs}} \mathtt{LL}'\langle \mathtt{LL}\langle p\rangle\{x/\lambda z.p'\}\rangle = t'$. By hypothesis, we have that $\lambda z.\mathtt{LL}'\langle p'\rangle \in \mathtt{T}$. Thus $\lambda z.p'$ and $p\{x/\lambda z.p'\}$ are pure. We conclude since $|\mathtt{LL}|_x = 0$ by hypothesis.

Now we can lift the property to T by observing that we necessarily have $t = \lambda x.u \to_{\text{sub}} \lambda x.u'$, where $u \to_{\text{sub}} u'$. Then we conclude by the previous point. □

Lemma 2.24. *If $t \in U$ and $t \to_{\mathrm{R}'} t'$, then $t' \in U$.*

Proof. An easy induction proves that $t \in \mathtt{U}$ implies $t\{x/p\} \in \mathtt{U}$ for any x and pure term p. We show that $t' \in \mathtt{U}$ by induction on the reduction relation. First, the base cases.

Case $t = \mathtt{L}\langle(\lambda x.p)\rangle t_0 \mapsto_{\text{dB}} \mathtt{L}\langle p[x/t_0]\rangle = t'$. Since $p, t_0 \in \mathtt{U}$, then $t' \in \mathtt{U}$.

Case $t = t_0[x/\mathtt{L}\langle y\rangle] \mapsto_{\text{var}} \mathtt{L}\langle t_0\{x/y\}\rangle = t'$. Since $t_0 \in \mathtt{U}$ and y is pure then $t' \in \mathtt{U}$.

Case $t = t_0[x/\mathtt{L}\langle t_1 t_2\rangle] \mapsto_{\text{app}} \mathtt{L}\langle t_0\{x/yz\}[y/t_1][z/t_2]\rangle$. Since $t_0 \in \mathtt{U}$ and yz is pure then $t' \in \mathtt{U}$.

Case $t = t_0[x/\mathtt{L}\langle\lambda y.p\rangle] \mapsto_{\text{dist}} \mathtt{L}\langle t_0[x /\!\!/ \lambda y.z[z/p]]\rangle = t'$. Since $t_0 \in \mathtt{U}$ and $|z|_z = 1$, then we have $\lambda y.z[z/p] \in \mathtt{T}$ and thus $t' \in \mathtt{U}$.

Case $t = t_0[x /\!\!/ \lambda y.\mathtt{LL}\langle p\rangle] \mapsto_{\text{abs}} \mathtt{LL}\langle t_0\{x/\lambda y.p\}\rangle = t'$. Since $t_0 \in \mathtt{U}$ and $\lambda y.p$ is pure then $t' \in \mathtt{U}$.

Then, the inductive cases.

Case $t = \lambda x.u$. Then $t \in \mathtt{U}$ implies in particular that u is pure, and then u can only contain dB-redexes, so that t does not R′-reduce to any term t'.

Case $t = t_0 u$ or $t = u t_0$ or $t = t_0[x \triangleleft u]$ or $t = u[x/t_0]$, where $t_0 \to_{\mathrm{R}'} t_0'$. We have $t' = t_0' u$, $t' = u t_0'$, $t' = t_0'[x/u]$, or $t' = u[x/t_0']$ respectively. By hypothesis $t_0 \in \mathtt{U}$, so by the *i.h.* $t_0' \in \mathtt{U}$ and therefore $t' \in \mathtt{U}$.

Case $t = u[x /\!\!/ t_0] \to_{\mathrm{R}'} u[x /\!\!/ t_0'] = t'$, where $t_0 \to_{\text{sub}} t_0'$. By hypothesis, we have $t_0 \in \mathtt{T}$. By lemma 2.23, $t_0' \in \mathtt{T}$, so $t' \in \mathtt{U}$. □

2.3.1 Call-by-Name

The **call-by-name** strategy $\to_{\text{name}}$ (figure 2.1) is defined on the set of terms U as the union of the following relations $\to_{\text{ndB}}$ and $\to_{\text{nsub}}$. The strategy is *weak* as there is no reduction under abstractions. It is also worth noticing (as a particular case of lemma 2.24) that $t \in \mathtt{U}$ and $t \to_{\text{name}} t'$ implies $t' \in \mathtt{U}$.

Example 2.25. This example follows a call-by-name evaluation. The name of the contextual rule is written in the superscript of the arrow symbol, and the redex is underlined.

$$\begin{array}{lll}
\underline{(\lambda x_1.\mathtt{I}(x_1\mathtt{I}))(\lambda y.(\mathtt{II})y)} & \to^{\text{DB}} & \underline{(\mathtt{I}(x_1\mathtt{I}))[x_1/\lambda y.(\mathtt{II})y]} \\
& \to^{\text{S}} & (\mathtt{I}(x_1\mathtt{I}))[x_1/\!/\lambda y.z[z/(\mathtt{II})y]] \\
& \to^{\text{SUBS}} & (\mathtt{I}(x_1\mathtt{I}))[x_1/\!/\lambda y.\underline{(z_1z_2)[z_1/\mathtt{II}][z_2/y]}] \\
& \to^{\text{SUBS}} & \underline{(\mathtt{I}(x_1\mathtt{I}))[x_1/\!/\lambda y.(z_1y)[z_1/\mathtt{II}]]} \\
& \to^{\text{S}} & \underline{(\mathtt{I}((\lambda y.z_1y)\mathtt{I}))[z_1/\mathtt{II}]} \\
& \to^{\text{SUBDB}} & x_2[x_2/(\lambda y.z_1y)\mathtt{I}][z_1/\mathtt{II}] \\
& \to^{+} & ((\lambda y.z_1y)\mathtt{I})[z_1/\mathtt{II}] \\
& \to^{+} & \lambda y.(\mathtt{II})y
\end{array}$$

The strategy $\to_{\text{name}}$ does not impose duplication of all nodes in the body of an abstraction inside the distributor: only the skeleton of the abstraction $\lambda y.(\mathtt{II})y$ is replicated. But the strategy forbids dB-reductions inside explicit cuts, so that there is no benefit gained by keeping shared terms such as $\mathtt{II}$. Indeed, the main idea behind full laziness is that shared terms are only reduced once. The CbN strategy, on the contrary, duplicates arguments before reducing them. The absence of optimization is reflected by the fact that the strategy, although not deterministic, enjoys the remarkable *diamond* property, guaranteeing in particular that all reduction sequences starting from t and ending in a normal form have the same length.

Property 2.26 (Diamond). *The CbN strategy enjoys the diamond property, i.e. for any terms $t, u, s \in U$ such that $t \to_{\text{name}} u$, $t \to_{\text{name}} s$ and $u \neq s$, there exists t' such that $u \to_{\text{name}} t'$ and $s \to_{\text{name}} t'$.*

Proof. We split the statement above in three different properties, each one proved by induction on the involved relation relations.

1. If $t \to_{\text{ndB}} u$ and $t \to_{\text{ndB}} s$, then there exists t' such that $u \to_{\text{ndB}} t'$ and $s \to_{\text{ndB}} t'$. We consider the following cases:

   Case ((APPDB), (APPDB)). We then have $t = t_0t_1$ such that $t \to_{\text{ndB}} u_0t_1 = u$ and $t \to_{\text{ndB}} s_0t_1 = s$, where $t_0 \to_{\text{ndB}} u_0$ and $t_0 \to_{\text{ndB}} s_0$. By the *i.h.* there is t_0' such

$$\frac{t \mapsto_{\text{dB}} t'}{t \to_{\text{ndB}} t'}\ (\text{DB}) \qquad \frac{t \to_{\text{ndB}} t'}{tu \to_{\text{ndB}} t'u}\ (\text{APPDB}) \qquad \frac{t \to_{\text{ndB}} t'}{t[x \triangleleft u] \to_{\text{ndB}} t'[x \triangleleft u]}\ (\text{SUBDB})$$

$$\frac{t \mapsto_{\text{sub}} t'}{t \to_{\text{nsub}} t'}\ (\text{S}) \qquad \frac{t \to_{\text{nsub}} t'}{tu \to_{\text{nsub}} t'u}\ (\text{APPS}) \qquad \frac{t \to_{\text{nsub}} t'}{u[x/\!/\lambda y.t] \to_{\text{nsub}} u[x/\!/\lambda y.t']}\ (\text{SUBS})$$

Figure 2.1: call-by-name strategy.

that $s_0 \to_{\text{ndB}} t_0'$ and $u_0 \to_{\text{ndB}} t_0'$. Therefore $s \to_{\text{ndB}} t_0' t_1 = t'$ and $u \to_{\text{ndB}} t'$.

Case ((SUBDB), (SUBDB)). We then have $t = t_0[x \triangleleft t_1]$ such that $t \to_{\text{ndB}} u_0[x \triangleleft t_1] = u$ and $t \to_{\text{ndB}} s_0[x \triangleleft t_1] = s$, where $t_0 \to_{\text{ndB}} u_0$ and $t_0 \to_{\text{ndB}} s_0$. By the *i.h.* there is t_0' such that $s_0 \to_{\text{ndB}} t_0'$ and $u_0 \to_{\text{ndB}} t_0'$. Therefore $s \to_{\text{ndB}} t_0'[x \triangleleft t_1] = t'$ and $u \to_{\text{ndB}} t'$.

Cases ((DB), (DB)); ((DB), (APPDB)); ((DB), (SUBDB)) and ((APPDB), (SUBDB)). They are impossible cases.

2. If $t \to_{\text{nsub}} u$ and $t \to_{\text{nsub}} s$, then there exists t' such that $u \to_{\text{nsub}} t'$ and $s \to_{\text{nsub}} t'$. We consider the following cases:

Case ((S), (S)). Impossible since u and s are assumed to be different.

Case ((S), (SUBS)). We have $t \in \mathtt{U}$ then $t = t_0[x /\!\!/ \lambda y.\mathtt{LL}\langle p\rangle[z \triangleleft t_1]]$, where $y \notin \text{fv}(\mathtt{LL}) \cup \text{fv}(t_1)$ and such that $t \mapsto_{\text{sub}} \mathtt{LL}\langle t_0\{x/\lambda y.p\}\rangle[z \triangleleft t_1] = u$. There are three cases for t.

Subcases $[z \triangleleft t_1] = [z/\mathtt{L}\langle w\rangle]$ and $[z \triangleleft t_1] = [z/\mathtt{L}\langle p_1 p_2\rangle]$. In each case, the only possibility is (S) on term $\mathtt{LL}\langle p\rangle[z/t_1]$. We then have

$$t \to_{\text{nsub}} t_0[x /\!\!/ \lambda y.\mathtt{L}'\langle \mathtt{LL}\langle p\rangle\{z/q\}\rangle] = s$$

for some $\mathtt{L}'$ and some pure term q. So $u \to_{\text{nsub}} \mathtt{L}'\langle \mathtt{LL}\langle t_0\{x/\lambda y.p\}\rangle\{z/q\}\rangle = u'$ and $s \to_{\text{nsub}} \mathtt{L}'\langle \mathtt{LL}\langle t_0\{x/\lambda y.p\{z/q\}\}\rangle\rangle = s'$. The equality $u' = s'$ holds because we can assume $z \neq y$ by α-equivalence, and $z \notin \text{fv}(\mathtt{LL})$ by definition.

Subcase $[z \triangleleft t_1] = [z/\lambda w.t_1']$. The only possible case is (S) on $\mathtt{LL}\langle p\rangle[z/\lambda w.t_1']$. We then have

$$t = t_0[x /\!\!/ \lambda y.\mathtt{LL}\langle p\rangle[z/\lambda w.t_1']] \to_{\text{nsub}} t_0[x /\!\!/ \lambda y.\mathtt{LL}\langle p\rangle[z /\!\!/ \lambda w.w'[w'/t_1']]] = s$$

We close the diagram with $u \to_{\text{nsub}} \mathtt{LL}\langle t_0\{x/\lambda y.p\}\rangle[z /\!\!/ \lambda w.w'[w'/t_1']] = t'$ and $s \to_{\text{nsub}} t'$.

Subcase $[z \triangleleft t_1] = [z /\!\!/ \lambda w.t_1']$. We have two different cases:

a) If the reduction happens inside t_1', then

$$t = t_0[x /\!\!/ \lambda y.\mathtt{LL}\langle p\rangle[z /\!\!/ \lambda w.t_1']] \to_{\text{nsub}} t_0[x /\!\!/ \lambda y.\mathtt{LL}\langle p\rangle[z /\!\!/ \lambda w.s_1]] = s$$

where $t_1' \to_{\text{nsub}} s_1$. We close by $u \to_{\text{nsub}} \mathtt{LL}\langle t_0\{x/\lambda y.p\}\rangle[z /\!\!/ \lambda w.s_1] = t'$ and $s \to_{\text{nsub}} t'$.

b) Otherwise, the (S) case for $\mathtt{LL}\langle p\rangle[z /\!\!/ \lambda w.t_1']$ gives

$$t \to_{\text{nsub}} t_0[x /\!\!/ \lambda y.\mathtt{L}\langle \mathtt{LL}\langle p\rangle\{z/v\}\rangle] = s$$

for some $\mathtt{L}$ and some value v. So $u \to_{\text{nsub}} \mathtt{L}\langle \mathtt{LL}\langle t_0\{x/\lambda y.p\}\rangle\{z/v\}\rangle = u'$ and $s \to_{\text{nsub}} \mathtt{L}\langle \mathtt{LL}\langle t_0\{x/\lambda y.p\{z/v\}\}\rangle\rangle = s'$. The equality $u' = s'$ holds because we can assume $y \notin \text{fv}(v) \cup \{z\}$ by α-equivalence, and $z \notin \text{fv}(\mathtt{LL})$ by definition.

Case ((APPS), (APPS)). We then have $t = t_0 t_1$ such that $t \to_{\text{nsub}} u_0 t_1 = u$ and $t \to_{\text{nsub}} s_0 t_1 = s$, where $t_0 \to_{\text{nsub}} u_0$ and $t_0 \to_{\text{nsub}} s_0$. By the *i.h.* $s_0 \to_{\text{nsub}} t_0'$ and $u_0 \to_{\text{nsub}} t_0'$. Therefore $u \to_{\text{nsub}} t_0' t_1 = t'$ and $s \to_{\text{nsub}} t'$.

Case ((SUBS), (SUBS)). We have $t = t_0[x /\!\!/ \lambda y.t_1]$ such that $t \to_{\text{nsub}} t_0[x /\!\!/ \lambda y.u_1] = u$ and $t \to_{\text{nsub}} t_0[x /\!\!/ \lambda y.s_1] = s$, where $t_1 \to_{\text{nsub}} u_1$ and $t_1 \to_{\text{nsub}} s_1$. By the *i.h.* $s_1 \to_{\text{nsub}} t_1'$ and $u_1 \to_{\text{nsub}} t_1'$. Therefore $u \to_{\text{nsub}} t_0[x /\!\!/ \lambda y.t_1'] = t'$ and $s \to_{\text{nsub}} t'$.

Cases ((S), (APPS)) and ((SUBS), (APPS)). These are impossible cases.

3. If $t \to_{\text{ndB}} u$ and $t \to_{\text{nsub}} s$, then there exists t' such that $u \to_{\text{nsub}} t'$ and $s \to_{\text{ndB}} t'$. We consider the following cases:

   Case ((DB), (APPS)). We have $t = \mathtt{L}\langle \lambda x.t_0\rangle[y \triangleleft t_2]t_1$ such that $t \to_{\text{ndB}} \mathtt{L}\langle t_0[x/t_1]\rangle[y \triangleleft t_2] = u$. There are three cases for $t \to_{\text{nsub}} s$.

   Case $t = \mathtt{L}\langle \lambda x.t_0\rangle[y /\!\!/ \lambda z.t_2']t_1 \to_{\text{nsub}} \mathtt{L}\langle \lambda x.t_0\rangle[y /\!\!/ \lambda z.t_3']t_1 = s$, where $t_2 = \lambda z.t_2'$ and $t_2' \to_{\text{nsub}} t_3'$. Then $u \to_{\text{nsub}} \mathtt{L}\langle t_0[x/t_1]\rangle[y /\!\!/ \lambda z.t_3'] = t'$ and $s \to_{\text{ndB}} t'$.

   Case $t = \mathtt{L}\langle \lambda x.t_0\rangle[y/\lambda z.t_2']t_1 \to_{\text{nsub}} \mathtt{L}\langle \lambda x.t_0\rangle[y /\!\!/ \lambda z.w[w/t_2']]t_1 = s$, where $t_2 = \lambda z.t_2'$. Then $u \to_{\text{nsub}} \mathtt{L}\langle t_0[x/t_1]\rangle[y /\!\!/ \lambda z.w[w/t_2']] = t'$ and $s \to_{\text{ndB}} t'$.

   Otherwise we have $t \to_{\text{nsub}} \mathtt{L}'\langle \mathtt{L}\langle \lambda x.t_0\rangle\{y/p\}\rangle t_1 = s$, for some $\mathtt{L}'$ and some pure term p. Therefore, $u \to_{\text{nsub}} \mathtt{L}'\langle \mathtt{L}\langle t_0[x/t_1]\rangle\{y/p\}\rangle = t'$ and $s \to_{\text{ndB}} t'$ because $y \notin \text{fv}(t_1)$. Note that y may be free in $\mathtt{L}$.

   Case ((APPDB), (APPS)). We have $t = t_0 t_1$ such that $t \to_{\text{ndB}} u_0 t_1 = u$ and $t \to_{\text{nsub}} s_0 t_1 = s$, where $t_0 \to_{\text{ndB}} u_0$ and $t_0 \to_{\text{nsub}} s_0$. By *i.h.* there exists t_0' such that $s_0 \to_{\text{ndB}} t_0'$ and $u_0 \to_{\text{nsub}} t_0'$. Therefore, $u \to_{\text{nsub}} t_0' t_1 = t'$ and $s \to_{\text{ndB}} t'$.

   Case ((SUBDB), (S)). We have $t = t_0[x \triangleleft t_1]$ such that $t \to_{\text{ndB}} u_0[x \triangleleft t_1] = u$, where $t_0 \to_{\text{ndB}} u_0$. If $t = t_0[x/\mathtt{L}\langle \lambda y.t_2\rangle] \to_{\text{nsub}} \mathtt{L}\langle t_0[x /\!\!/ \lambda y.z[z/t_2]]\rangle = s$, where $t_1 = \lambda y.t_2$, then $s \to_{\text{ndB}} \mathtt{L}\langle u_0[x /\!\!/ \lambda y.z[z/t_2]]\rangle = t'$ and $u \to_{\text{nsub}} t'$. Otherwise, $t \to_{\text{nsub}} \mathtt{L}\langle t_0\{x/p\}\rangle = s$ for some $\mathtt{L}$ and some pure term p. We show that $t_0\{x/p\} \to_{\text{ndB}} u_0\{x/p\}$ by induction on $t_0 \to_{\text{ndB}} u_0$. From this, we can deduce $s \to_{\text{ndB}} \mathtt{L}\langle u_0\{x/p\}\rangle = t'$ and conclude because $u \to_{\text{nsub}} t'$.

   Subcase $t_0 = \mathtt{L}'\langle \lambda y.q\rangle t_2 \to_{\text{dB}} \mathtt{L}'\langle q[y/t_2]\rangle = u_0$. Without loss of generality, we can assume by α-conversion that $y \notin \text{fv}(p) \cup \{x\}$. Then

$$\begin{aligned} t_0\{x/p\} &= \mathtt{L}'\{x/p\}\langle \lambda y.q\{x/p\}\rangle t_2\{x/p\} \\ &\to_{\text{ndB}} \mathtt{L}'\{x/p\}\langle q\{x/p\}[y/t_2\{x/p\}]\rangle = u_0\{x/p\} \end{aligned}$$

Subcase $t_0 = t_0' t_2 \to_{\mathrm{ndB}} u_0' t_2 = u_0$ from $t_0' \to_{\mathrm{ndB}} u_0'$. Then by the induction hypothesis and by rule (APPDB) we can conclude

$$t_0\{x/p\} = t_0'\{x/p\}t_2\{x/p\} \to_{\mathrm{ndB}} u_0'\{x/p\}t_2\{x/p\} = u_0\{x/p\}$$

Subcase $t_0 = t_0'[y \lhd t_2] \to_{\mathrm{ndB}} u_0'[y \lhd t_2] = u_0$ from $t_0' \to_{\mathrm{ndB}} u_0'$. W.l.o.g. we assume by α-conversion that $x \neq y$, then by *i.h.* and rule ((SUBDB)) we conclude $t_0\{x/p\} = t_0'\{x/p\}[y \lhd t_2\{x/p\}] \to_{\mathrm{ndB}} u_0'\{x/p\}[y \lhd t_2\{x/p\}] = u_0\{x/p\}$.

Case ((SUBDB), (SUBS)). We have $t = t_0[x/\!/\lambda y.t_1]$ such that $t \to_{\mathrm{ndB}} u_0[x/\!/\lambda y.t_1] = u$ and $t \to_{\mathrm{nsub}} t_0[x/\!/\lambda y.s_1] = s$, where $t_0 \to_{\mathrm{ndB}} u_0$ and $t_1 \to_{\mathrm{nsub}} s_1$. Therefore $u \to_{\mathrm{nsub}} u_0[x/\!/\lambda y.s_1] = t'$ and $s \to_{\mathrm{ndB}} t'$.

Cases ((DB), (S)); ((DB), (SUBS)); ((APPDB), (S)); ((APPDB),(SUBS)) and ((SUBDB),(APPS)). These are impossible cases. □

It is worth noticing that call-by-name in the λ-calculus can be simulated by call-by-name in λR. The former can be defined by weak-head reduction, denoted $\to_{\mathrm{whr}}$, and generated by the following rules:

$$\frac{t \to_\beta t'}{t \to_{\mathrm{whr}} t'} \qquad \frac{t \to_{\mathrm{whr}} t'}{tu \to_{\mathrm{whr}} t'u}$$

There is in particular a one-to-one relation between β-steps and ndB-steps.

Lemma 2.27 (Relating call-by-name strategies).

(i) *Let* $p_0 \in \mathtt{T}_P$. *If* $p_0 \to_{\mathrm{whr}} p_1$, *then* $p_0 \to_{\mathrm{ndB}} \to^+_{\mathrm{nsub}} p_1$ *(thus* $p_0 \to^+_{\mathrm{name}} p_1$*).*

(ii) *Let* $t_0 \in \mathcal{U}$. *If* $t_0 \to_{\mathrm{ndB}} t_1$, *then* $t_0^{\downarrow} \to_{\mathrm{whr}} t_1^{\downarrow}$. *If* $t_0 \to_{\mathrm{nsub}} t_1$, *then* $t_0^{\downarrow} = t_1^{\downarrow}$.

Proof. The first item is by induction on $\to_{\mathrm{whr}}$.

Case $p_0 = (\lambda x.p)q \to_\beta p\{x/q\} = p_1$. Then $(\lambda x.p)q \to_{\mathrm{ndB}} p[x/q]$ and we need to verify that $p[x/q] \to^+_{\mathrm{nsub}} p\{x/q\}$. The proof of $t[x/q] \to^+_{\mathrm{nsub}} t\{x/q\}$ for any $t \in \mathtt{U}$ and pure term q is by induction on q:

Subcase $q = y$. Then $t[x/y] \to_{\mathrm{nsub}} t\{x/y\}$.

Subcase $q = q_0 q_1$. Then $t[x/q] \to_{\mathrm{nsub}} t\{x/z_0 z_1\}[z_0/q_0][z_1/q_1]$. By the *i.h.* we have

$$t\{x/z_0z_1\}[z_0/q_0][z_1/q_1] \to^+_{\mathrm{nsub}} (t\{x/z_0z_1\}[z_0/q_0])\{z_1/q_1\} = t\{x/z_0q_1\}[z_0/q_0]$$
$$\text{and } t\{x/z_0q_1\}[z_0/q_0] \to^+_{\mathrm{nsub}} t\{x/z_0q_1\}\{z_0/q_0\} = t\{x/q_0q_1\}$$

Therefore, $t[x/q_0q_1] \to^+_{\mathrm{nsub}} t\{x/q_0q_1\}$.

Subcase $q = \lambda y.q'$. Then $t[x/q] \to_{\mathrm{nsub}} t[x/\!/\lambda y.z[z/q']]$. By the *i.h.* we have that $z[z/q'] \to^+_{\mathrm{nsub}} z\{z/q'\} = q'$ thus $t[x/\!/\lambda y.z[z/q']] \to^+_{\mathrm{nsub}} t[x/\!/\lambda y.q'] \to_{\mathrm{nsub}} t\{x/\lambda y.q'\}$. Therefore, $t[x/\lambda y.q'] \to^+_{\mathrm{nsub}} t\{x/\lambda y.q'\}$.

Case $p_0 = pq \to_{\mathrm{whr}} p'q = p_1$ where $p \to_{\mathrm{whr}} p'$. By the *i.h.* we have that $p \to^{+}_{\mathrm{name}} p'$ then, by (APPDB) and (APPS), $p_0 = pq \to^{+}_{\mathrm{name}} p'q = p_1$.

The second item is by case analysis on $\to_{\mathrm{name}}$. If $t_0 \to_{\mathrm{nsub}} t_1$ then $t_0^{\downarrow} = t_1^{\downarrow}$ by lemma 2.15. If $t_0 \to_{\mathrm{ndB}} t_1$ then we prove the property by induction on $\to_{\mathrm{ndB}}$.

Case $t_0 = (\lambda x.t)u \to_{\mathrm{ndB}} t[x/u] = t_1$. Then $t_0^{\downarrow} = (\lambda x.t^{\downarrow})u^{\downarrow} \to_{\beta} t^{\downarrow}\{x/u^{\downarrow}\} = t_1^{\downarrow}$. Note that both $t^{\downarrow}$ and $u^{\downarrow}$ are pure terms.

Case $t_0 = tu \to_{\mathrm{ndB}} t'u = t_1$ where $t \to_{\mathrm{ndB}} t'$. Then $t^{\downarrow} \to_{\mathrm{whr}} t'^{\downarrow}$ by the *i.h.*, thus $t_0^{\downarrow} = t^{\downarrow}u^{\downarrow} \to_{\mathrm{whr}} t'^{\downarrow}u^{\downarrow} = t_1^{\downarrow}$.

Case $t_0 = t[x \triangleleft u] \to_{\mathrm{ndB}} t'[x \triangleleft u] = t_1$ where $t \to_{\mathrm{ndB}} t'$. Then $t^{\downarrow} \to_{\mathrm{whr}} t'^{\downarrow}$ by the *i.h.*, thus $t_0^{\downarrow} = t^{\downarrow}\{x/u^{\downarrow}\} \to_{\mathrm{whr}} t'^{\downarrow}\{x/u^{\downarrow}\} = t_1^{\downarrow}$. Note that the result depends on the closure of $\to_{\mathrm{whr}}$ by (implicit) substitutions, which has a straightforward proof by induction on (pure) term $t^{\downarrow}$, using substitution composition. □

The following grammar $\mathrm{NF}_{\mathrm{name}}$ intends to characterize normal forms with respect to the strategy $\to_{\mathrm{name}}$:

$$\begin{array}{lcl} \mathrm{NF}_{\mathrm{name}} & ::= & \lambda x.p \mid \mathrm{NE}_{\mathrm{name}} \\ \mathrm{NE}_{\mathrm{name}} & ::= & x \mid \mathrm{NE}_{\mathrm{name}}\ t \end{array}$$

Notice that all normal forms are pure terms: we unfold all explicit substitutions with substeps.

Lemma 2.28. *Let $t \in U$. Then $t \in \mathrm{NE}_{\mathrm{name}}$ iff t is in* name*-nf.*

Proof. The left-to-right implication is straightforward. The right-to-left implication is by induction on $\mathtt{U}$.

Case $t = x$. By definition, $t \in \mathrm{NE}_{\mathrm{name}}$.

Case $t = \lambda x.p$. Then $t \in \mathrm{NE}_{\mathrm{name}}$ by definition.

Case $t = t'u$, where $t', u \in \mathtt{U}$. By definition of $\to_{\mathrm{name}}$, t in name-nf implies t' is also in name-nf and t' is neither an explicit cut nor an abstraction. Thus $t' \in \mathrm{NE}_{\mathrm{name}}$ by the *i.h.* and we can conclude $t \in \mathrm{NE}_{\mathrm{name}}$.

Case $t = t'[x/u]$, where $t', u \in \mathtt{U}$. This is not possible because there is always an applicable structural rule which would contradict t to be in name-nf.

Case $t = t'[x/\!/\lambda y.u]$, where $\lambda y.u = \lambda y.\mathtt{LL}\langle p\rangle \in \mathtt{T}$. Then either we can apply a structural rule on u, or u is pure (*i.e.* $\mathtt{LL} = \Diamond$) and we can apply rule $\to_{\mathrm{abs}}$. In both cases we would have a contradiction with t in name-nf. □

2.3.2 Call-by-Need

We now specify a deterministic strategy flneed implementing demand-driven computations and only linearly replicating nodes of *values* (*i.e.* pure abstractions). Given a value $\lambda x.p$, only the piece of structure containing the paths between the binder λx and all the free occurrences of x in p, named *skeleton*, will be copied. All the other components of the abstraction will remain shared, thus avoiding some future duplications of redexes, as explained in the introduction. By copying only the smallest possible substructure of the abstraction, the strategy flneed implements an optimization of call-by-need called *fully lazy sharing* [Wad71]. First, we formally define the key notions we are going to use.

A **free expression** [Pey87; Bal12b] of a *pure* term p is a strict subterm q of p such that every free occurrence of a variable in q is also a free occurrence of the variable in p. A **free expression** of p is **maximal** if it is not a subterm of another free expression of p. From now on, we will consider the (ordered) list of all MFEs of a term. Thus *e.g.* the MFEs of $\lambda y.p$, where $p = (\mathtt{I}y)\mathtt{I}(\lambda z.zyw)$, is given by the list $[\mathtt{I}; \mathtt{I}; w]$.

An n**-ary context** ($n \geq 0$) is a term with n holes $\diamond$. A skeleton is an n-ary pure context where the maximal free expressions w.r.t. a variable set θ are replaced with holes. We introduce two different yet equivalent definitions of skeleton: we argue that they entail respectively a big-step and a small-step semantics. We thus a give operational perspectives to these two classical definitions.

A first definition of skeleton. The first notion of skeleton runs as follows. Given any set of variables θ, the θ-**skeleton** $\{\!\{p\}\!\}^\theta$ of a pure term p is an n-ary pure (*i.e.* without explicit cuts) context defined as $\{\!\{p\}\!\}^\theta := \diamond$ if $\theta \cap \mathrm{fv}(p) = \emptyset$; otherwise:

$$\{\!\{x\}\!\}^\theta := x \qquad \{\!\{\lambda x.p\}\!\}^\theta := \lambda x.\{\!\{p\}\!\}^{\theta\cup\{x\}} \qquad \{\!\{p_1 p_2\}\!\}^\theta := \{\!\{p_1\}\!\}^\theta \{\!\{p_2\}\!\}^\theta$$

Thus *e.g.* if $p = (Iy)I(\lambda z.zyw)$ as above, then $\{\!\{p\}\!\}^{\{y\}} = (\diamond y)\diamond(\lambda z.zy\diamond)$.

Splitting a term into a skeleton and a multiset of MFEs is at the core of full laziness. This can naturally be implemented in the node replication model, as observed in [GHP13b]. Here, we give two different (alternative) operational semantics to achieve it. The first one (figure 2.2), written $\Downarrow^\theta$, uses big-step semantics and implements the first definition of skeleton introduced above.

$$\frac{x \text{ fresh}}{p \Downarrow^\theta x[x/p]} \quad \text{when } \mathrm{fv}(p) \cap \theta = \emptyset\text{; otherwise:}$$

$$\frac{}{x \Downarrow^\theta x} \qquad \frac{p \Downarrow^{\theta\cup\{x\}} \mathtt{L}\langle p'\rangle}{\lambda x.p \Downarrow^\theta \mathtt{L}\langle\lambda x.p'\rangle} \qquad \frac{p \Downarrow^\theta \mathtt{L}_1\langle p'\rangle \qquad q \Downarrow^\theta \mathtt{L}_2\langle q'\rangle}{pq \Downarrow^\theta \mathtt{L}_2\langle \mathtt{L}_1\langle p'q'\rangle\rangle}$$

Figure 2.2: Relation $\Downarrow^\theta$: Splitting Skeleton and MFEs in Big-Step Semantics.

The big-steps semantics can be seen as a reformulation of the proof that splitting the skeleton can be done inside the atomic λ-calculus [GHP13b, lemma 30]. This proof proceeds by induction, and the premises of the inferences of our big-steps rules reflect the use of an induction hypothesis.

Each of the rules in figure 2.2 corresponds to a different case in the first definition of θ-skeleton. In the first rule, since there is no free variable of p in θ, p is thus an MFE kept shared in an explicit substitution. The other three rules correspond to each possible constructor, where all the explicit cuts created during the inductive cases are pushed out.

Example 2.29. Let $y, z \notin \mathrm{fv}(t)$, so that t is the MFE of $\lambda y.x[x/\lambda z.(yt)z]$. Then,

$$\dfrac{\dfrac{\dfrac{\overline{y \Downarrow^{\{y,z\}} y} \qquad \overline{t \Downarrow^{\{y,z\}} x[x/t]}}{yt \Downarrow^{\{y,z\}} (yx)[x/t]} \qquad \overline{z \Downarrow^{\{y,z\}} z}}{(yt)z \Downarrow^{\{y,z\}} ((yx)z)[x/t]}}{\lambda z.(yt)z \Downarrow^{\{y\}} (\lambda z.(yx)z)[x/t]}$$

Lemma 2.30 (Correctness of $\Downarrow^\theta$). *If $p \in \mathtt{T}_P$, then $\exists n \geq 0$ s.t. $p \Downarrow^\theta \{\!\{p\}\!\}^\theta\langle x_1, \dots, x_n\rangle[x_i/t_i]_{i\leq n}$, where $\{\!\{p\}\!\}^\theta\langle t_1, \dots, t_n\rangle = p$, and $(x_i)_{1\leq i\leq n}$ are fresh and pairwise distinct variables. Moreover, $\mathrm{fv}(t_i) \cap \theta = \emptyset$ for all $1 \leq i \leq n$.*

Proof. If $\mathrm{fv}(p) \cap \theta = \emptyset$, then $p \Downarrow^\theta x_1[x_1/p]$ and $\{\!\{p\}\!\}^\theta = \Diamond$, so that $\{\!\{p\}\!\}^\theta\langle p\rangle = p$ trivially holds. Otherwise, we reason by induction on p:

Case $p = x$. Then $\{\!\{x\}\!\}^\theta = x$, so the property holds for $n = 0$ because $x \Downarrow^\theta x$.

Case $p = p_1 p_2$. Then $\{\!\{p\}\!\}^\theta = \{\!\{p_1\}\!\}^\theta\{\!\{p_2\}\!\}^\theta$. By the *i.h.* we have

$$p_1 \Downarrow^\theta \{\!\{p_1\}\!\}^\theta\langle x_1, \dots, x_k\rangle[x_i/t_i]_{i\leq k} \text{ and } p_2 \Downarrow^\theta \{\!\{p_2\}\!\}^\theta\langle x_{k+1}, \dots, x_n\rangle[x_i/t_i]_{k<i\leq n}, \text{ where}$$
$$\{\!\{p_1\}\!\}^\theta\langle t_1, \dots, t_k\rangle = p_1 \text{ and } \{\!\{p_2\}\!\}^\theta\langle t_{k+1}, \dots, t_n\rangle = p_2.$$

Hence:

$$\begin{aligned} p_1 p_2 \Downarrow^\theta (\{\!\{p_1\}\!\}^\theta\langle x_1, \dots, x_k\rangle\{\!\{p_2\}\!\}^\theta\langle x_{k+1}, \dots, x_n\rangle)[x_i/t_i]_{i\leq k}[x_i/t_i]_{k<i\leq n} \\ = \{\!\{p\}\!\}^\theta\langle x_1, \dots, x_n\rangle[x_i/t_i]_{i\leq n} \end{aligned}$$

Case $p = \lambda x.p'$. Then $\{\!\{p\}\!\}^\theta = \lambda x.\{\!\{p'\}\!\}^{\theta\cup\{x\}}$. By the *i.h.* we have

$$p' \Downarrow^{\theta\cup\{x\}} \{\!\{p'\}\!\}^{\theta\cup\{x\}}\langle x_1, \dots, x_n\rangle[x_i/t_i]_{i\leq n}.$$

Moreover, $x \notin \bigcup_{i\leq n} \mathrm{fv}(t_i)$ by definition of $\Downarrow$ and every x_i is different from x. Hence:

$$\lambda x.p' \Downarrow^\theta (\lambda x.\{\!\{p'\}\!\}^{\theta\cup\{x\}}\langle x_1, \dots, x_n\rangle)[x_i/t_i]_{i\leq n} = \{\!\{\lambda x.p'\}\!\}^\theta\langle x_1, \dots, x_n\rangle[x_i/t_i]_{i\leq n}. \qquad \square$$

The correctness lemma states in particular that $p \Downarrow^\theta \mathtt{L}\langle p'\rangle$ implies p' is pure and $\mathrm{fv}(\mathtt{L}) \cap \theta = \emptyset$.

An alternative definition of skeleton. An *alternative* definition of θ-skeleton can be given by *removing* the maximal free expressions from a term. Indeed, the θ-**skeleton** $\{\!\{p\}\!\}^\theta$ of a pure term p, where $\theta = \{x_1 \dots x_n\}$, is the n-ary pure context $\{\!\{p\}\!\}^\theta$ such that $\{\!\{p\}\!\}^\theta\langle q_1, \dots, q_n\rangle = p$, for $[q_1; \dots; q_n]$ the maximal free expressions of $\lambda x_1. \dots \lambda x_n.p$.[1]. It is easy to show that both notions of skeleton are equivalent, *i.e.* $\{\!\{p\}\!\}^\theta = \{\!\{p\}\!\}^\theta$. Thus, for the same p as before, $\lambda y.\{\!\{p\}\!\}^{\{y\}} = \lambda y.(\diamond y)\diamond(\lambda z.zy\diamond)$.

The second strategy to split a term into a skeleton and its MFEs is the small-step strategy $\to_{\text{st}}$ on the set of terms Λ (figure 2.3), which is indeed a subset of the reduction relation $\to_{\text{R}}$. It implements the second definition of skeleton we have introduced. The relation $\to_{\text{st}}$ makes use of four basic rules which are parameterized by the variable y upon which the skeleton is built, written $\mapsto^y$. There are also two contextual (inductive) rules.

This definition is more subtle than the big-steps one. Indeed, it is necessary to handle contexts explicitly (by the last two rules), to pass the variable upon which to build the skeleton to a local level, and to encode determinism.

$$\frac{}{t[x/y] \mapsto^y_{\text{var}} t\{x/y\}} \qquad \frac{y \in \text{fv}(p_1 p_2)}{t[x/p_1 p_2] \mapsto^y_{\text{app}} t\{x/x_1 x_2\}[x_1/p_1][x_2/p_2]}$$

$$\frac{y \in \text{fv}(\lambda z.p)}{t[x/\lambda z.p] \mapsto^y_{\text{dist}} t[x/\!/\lambda z.w[w/p]]} \qquad \frac{y \in \text{fv}(\lambda z.\text{LL}\langle p\rangle) \qquad z \notin \text{fv}(\text{LL})}{t[x/\!/\lambda z.\text{LL}\langle p\rangle] \mapsto^y_{\text{abs}} \text{LL}\langle t\{x/\lambda z.p\}\rangle}$$

$$\frac{t \mapsto^y t' \qquad y \in \text{fv}(t) \qquad y \notin \text{fv}(\text{LL})}{\lambda y.\text{LL}\langle t\rangle \to_{\text{st}} \lambda y.\text{LL}\langle t'\rangle}\ (\text{CTX1}) \qquad \frac{t \to_{\text{st}} t' \qquad y \in \text{fv}(t) \qquad y \notin \text{fv}(\text{LL})}{\lambda y.\text{LL}\langle u[x/\!/t]\rangle \to_{\text{st}} \lambda y.\text{LL}\langle u[x/\!/t']\rangle}\ (\text{CTX2})$$

Figure 2.3: Relation $\to_{\text{st}}$: Splitting Skeleton and MFEs in Small-Step Semantics.

Example 2.31. Let $\lambda y.x[x/\lambda z.(yt)z]$ be as in example 2.29. Distance is highlighted.

$$\begin{aligned}\lambda y.\underline{x[x/\lambda z.(yt)z]} &\to^y_{\text{dist}} \lambda y.x[x/\!/\lambda z.\underline{w[w/(yt)z]}] \to^z_{\text{app}} \lambda y.x[x/\!/\lambda z.\underline{(w_1 w_2)[w_1/yt][w_2/z]}] \\ &\to^z_{\text{var}} \lambda y.\underline{x[x/\!/\lambda z.(w_1 z)\ [w_1/yt]\ ]} \to^y_{\text{abs}} \lambda y.\underline{(\lambda z.w_1 z)[w_1/yt]} \\ &\to^y_{\text{app}} \lambda y.\underline{(\lambda z.(x_1 x_2)z)[x_1/y]}[x_2/t] \to^y_{\text{var}} \lambda y.(\lambda z.(yx_2)z)[x_2/t]\end{aligned}$$

Notice that the focused variable changes from y to z, then back to y. This is because $\to_{\text{st}}$ constructs the innermost skeletons first. The small-step approach allows to parametrize the reduction relation by only one variable at a time, instead of a set.

Lemma 2.32. *If $t \in T$ and $t \to_{\text{st}} t'$, then $t' \in T$.*

[1] The order of the abstractions is irrelevant.

Proof. For the root $\mapsto^y$ rules, we first show that if $t = \mathrm{LL}_0\langle p_0\rangle$ with $|p_0|_z = 1$ for all $z \in \mathrm{dom}(\mathrm{LL}_0)$, and $t \mapsto^y t'$, then $t' = \mathrm{LL}_1\langle p_1\rangle$ with $|p_1|_z = 1$ for all $z \in \mathrm{dom}(\mathrm{LL}_1)$.

Case $t \mapsto^y_{\mathrm{var}} t'$. This is straightforward.

Case $t = \mathrm{LL}\langle p\rangle[x/q_1q_2] \mapsto^y_{\mathrm{app}} \mathrm{LL}\langle p\rangle\{x/x_1x_2\}[x_1/q_1][x_2/q_2] = t'$. Since $x \notin \mathrm{fv}(\mathrm{LL})$, we have $\mathrm{LL}\{x/x_1x_2\} = \mathrm{LL}$. Moreover, freshness of x_1, x_2 implies $|\mathrm{LL}\langle p'\rangle|_{x_1} = |\mathrm{LL}\langle p'\rangle|_{x_2} = |\mathrm{LL}\langle p\rangle|_x = 1$, where $p' = p\{x/x_1x_2\}$, and $|q_1|_{x_2} = 0$.

Case $t = u[x/\lambda x'.p] \mapsto^y_{\mathrm{dist}} u[x/\!/\lambda x'.w[w/p]] = t'$. This is true by hypothesis, where in particular $|u|_x = 1$, and $\lambda x'.w[w/p] \in \mathrm{T}$ because p is pure and $|w|_w = 1$.

Case $t = \mathrm{LL}_1\langle p_1\rangle[x/\!/\lambda z.\mathrm{LL}_2\langle p_2\rangle] \mapsto^y_{\mathrm{abs}} \mathrm{LL}_2\langle \mathrm{LL}_1\langle p_1\rangle\{x/\lambda z.p_2\}\rangle = t'$. By hypothesis $|p_1|_x = 1$ and $|\mathrm{LL}_1|_x = 0$, so that $t' = \mathrm{LL}_2\langle \mathrm{LL}_1\langle p_1\{x/\lambda z.p_2\}\rangle\rangle = \mathrm{LL}'_1\langle p'\rangle$, since for all $z_1 \in \mathrm{dom}(\mathrm{LL}_1)$ and all $z_2 \in \mathrm{dom}(\mathrm{LL}_2)$, $|p_1|_{z_1} = |p_2|_{z_2} = 1$ and, by α-conversion, $|p_1|_{z_2} = |p_2|_{z_1} = 0$ so that $|p_1\{x/\lambda z.p_2\}|_{z'} = 1$ for any $z' \in \mathrm{dom}(\mathrm{LL}')$.

Then, for the contextual rules, we show by induction on $t \to_{\mathrm{sub}} t'$: if $t \in \mathrm{T}$ and $t \to_{\mathrm{sub}} t'$, then $t' \in \mathrm{T}$.

Case (CTX1). We have $t = \lambda y.\mathrm{LL}\langle t_0\rangle \to_{\mathrm{sub}} \lambda y.\mathrm{LL}\langle t_1\rangle$. By the hypothesis that $t \in \mathrm{T}$ follows $t_0 = \mathrm{LL}_0\langle p_0\rangle$. By the previous case analysis, $t_1 = \mathrm{LL}_1\langle p_1\rangle$. Therefore $t' \in \mathrm{T}$.

Case (CTX2). We have $t = \lambda y.\mathrm{LL}\langle u[x/\!/t_0]\rangle \to_{\mathrm{sub}} \lambda y.\mathrm{LL}\langle u[x/\!/t_1]\rangle$. By the hypothesis that $t \in \mathrm{T}$ follows $t_0 \in \mathrm{T}$. By induction hypothesis, $t_1 \in \mathrm{T}$. Therefore $t' \in \mathrm{T}$. □

Lemma 2.33. *The reduction relation* $\to_{\mathrm{st}}$ *is confluent and terminating.*

Proof. To show termination it is sufficient to notice that $t \to_{\mathrm{st}} t'$ implies $t \to_{\mathrm{sub}} t'$. Since $\to_{\mathrm{sub}}$ is terminating (corollary 2.14) then we conclude termination of $\to_{\mathrm{st}}$. Next, we show that $\to_{\mathrm{st}}$ is confluent by observing that it is deterministic. Indeed,

- The base rules $\mapsto^y$ only reduce the outermost cut and they are all distinct: there is one rule for an outermost distributor, and three rules for outermost explicit substitutions, one for each possible form (variable, application, abstraction).
- Because of the condition $y \notin \mathrm{fv}(\mathrm{LL})$ in rules (CTX1) and (CTX2) the base rules are always applied from right to left inside an abstraction.
- Moreover, rule (CTX2) does not overlap with any other rule, in particular with $\mapsto^y_{\mathrm{abs}}$. Indeed, for a term $u[x/\!/\lambda z.z\mathrm{LL}\langle p\rangle]$, there are only two possibilities. Either z is a free variable of LL, and we cannot apply $\mapsto^y_{\mathrm{abs}}$, or z is not a free variable of LL, and we can apply $\mapsto^y_{\mathrm{abs}}$. In the latter, there is in particular no cut of LL for which z is free. Therefore, we cannot apply any base-rule recursively inside the distributor. So, we cannot apply rule (CTX2).

Since rule application is deterministic, then there is no possible diverging diagram, and thus confluence is trivial. □

Thus, from now on, we denote by $\Downarrow_{\mathrm{st}}$ the function relating a term of $\mathtt{T}$ to its unique st-nf. For instance, from example 2.31 we deduce $\lambda y.x[x/\lambda z.(yt)z] \Downarrow_{\mathrm{st}} \lambda y.(\lambda z.(yx_2)z)[x_2/t]$.

Lemma 2.34. *If p is a pure term and $\mathtt{LL}$ a (commutative) list context where $y \notin \mathrm{fv}(\mathtt{LL})$, then there exists n and an n-ary pure context c such that*

$$\lambda y.\mathtt{LL}\langle t[z/p]\rangle \to^*_{\mathrm{st}} \lambda y.\mathtt{LL}\langle t\{z/c\langle x_1,\dots,x_n\rangle\}[x_i/q_i]_{1\le i\le n}\rangle$$

where the variables $x_1,\dots,x_n$ are fresh and pairwise distinct and $[q_1;\dots;q_n]$ are the MFE of $\lambda y.p$ such that $c\langle q_1,\dots,q_n\rangle = p$.

Proof. If $y \notin \mathrm{fv}(p)$, then p is the MFE of $\lambda y.p$ and the property is satisfied by the empty reduction, with $n = 1$, $c = \Diamond$, and $q_1 = p$. Otherwise, we reason by induction on p.

Case $p = y$. Then $\lambda y.p$ has no MFE and $\lambda y.\mathtt{LL}\langle t[z/y]\rangle \to^{y}_{\mathrm{var}} \lambda y.\mathtt{LL}\langle t\{z/y\}\rangle$. Then the property holds for $n = 0$ and the nullary context y.

Case $p = p_1p_2$. Then by the *i.h.* on p_2 and on p_1 we have:

$$\begin{aligned}
\lambda y.\mathtt{LL}\langle t[z/p_1p_2]\rangle &\to^{y}_{\mathrm{app}} \lambda y.\mathtt{LL}\langle t\{z/z_1z_2\}[z_1/p_1][z_2/p_2]\rangle\\
&\to^*_{\mathrm{st}} \lambda y.\mathtt{LL}\langle t\{z/z_1c_2\langle x_{k+1},\dots,x_n\rangle\}[z_1/p_1][x_i/q_i]_{k<i\le n}\rangle\\
&\to^*_{\mathrm{st}} \lambda y.\mathtt{LL}\langle t\{z/c_1\langle x_1,\dots,x_k\rangle c_2\langle x_{k+1},\dots,x_n\rangle\}[x_i/q_i]_{1\le i\le k}[x_i/q_i]_{k<i\le n}\rangle\\
&= \lambda y.\mathtt{LL}\langle t\{z/c\langle x_1,\dots,x_n\rangle\}[x_i/q_i]_{1\le i\le n}\rangle
\end{aligned}$$

where $c\langle x_1,\dots,x_n\rangle = c_1\langle x_1,\dots,x_k\rangle c_2\langle x_{k+1},\dots,x_n\rangle$, and the variables $x_1,\dots,x_n$ are chosen to be pairwise distinct. To apply the *i.h.* on p_1, we take $\mathtt{LL}$ to be $\mathtt{LL}\langle\Diamond[x_i/q_i]_{k<i\le n}\rangle$, which verifies the hypothesis of the statement since by definition of the MFEs, $y \notin \cup_{k<i\le n}\mathrm{fv}(q_i)$. We can conclude since the maximal free expressions of $\lambda y.p_1p_2$ can be computed by considering the MFEs of $\lambda y.p_1$ and $\lambda y.p_2$ respectively, *i.e.* $[q_1;\dots;q_n]$.

Case $p = \lambda x.p'$. Then by the *i.h.* on p' we have:

$$\lambda x.z'[z'/p'] \to^*_{\mathrm{st}} \lambda x.c'\langle x_1,\dots,x_n\rangle[x_i/q_i]_{1\le i\le n}$$

where the terms $[q_1;\dots;q_n]$ are the MFEs of $\lambda x.p'$, so in particular $x \notin \cup_{1\le i\le n}\mathrm{fv}(q_i)$.

We can then apply the *i.h.* on $q_n, \dots, q_1$, thus for $t_0 = \lambda y.\mathrm{LL}\langle t[z/\lambda x.p']\rangle$ we have:

$$\begin{aligned}
t_0 &\to^{y}_{\mathrm{dist}} \lambda y.\mathrm{LL}\langle t[z/\!/\lambda x.z'[z'/p']]\rangle \\
&\to^{*}_{\mathrm{st}} \lambda y.\mathrm{LL}\langle t[z/\!/\lambda x.c'\langle x_1, \dots, x_n\rangle[x_i/q_i]_{1\le i\le n}]\rangle \\
&\to^{y}_{\mathrm{abs}} \lambda y.\mathrm{LL}\langle t\{z/\lambda x.c'\langle x_1, \dots, x_n\rangle\}[x_i/q_i]_{1\le i\le n}\rangle \\
&\to^{*}_{\mathrm{st}} \lambda y.\mathrm{LL}\langle t\{z/\lambda x.c'\langle x_1, \dots, x_{n-1}, c_n\langle x_n^1, \dots, x_n^{m_n}\rangle\rangle\}[x_i/q_i]_{1\le i<n}[x_n^j/q_n^j]_{1\le j\le m_n}\rangle \\
&\to^{*}_{\mathrm{st}} \lambda y.\mathrm{LL}\langle t\{z/\lambda x.c'\langle c_1\langle x_1^1, \dots, x_1^{m_1}\rangle, \dots, c_n\langle x_n^1, \dots, x_n^{m_n}\rangle\rangle\}[x_i^j/q_n^j]_{1\le j\le m_i, 1\le i\le n}\rangle \\
&= \lambda y.\mathrm{LL}\langle t\{z/c\langle x_1^1, \dots, x_n^{m_n}\rangle\}[x_i^j/q_n^j]_{1\le j\le m_i, 1\le i\le n}\rangle
\end{aligned}$$

where $c\langle x_1^1, \dots, x_n^{m_n}\rangle = \lambda x.c'\langle c_1\langle x_1^1, \dots, x_1^{m_1}\rangle, \dots, c_n\langle x_n^1, \dots, x_n^{m_n}\rangle\rangle$ and the variables x_1^1 to $x_n^{m_n}$ are taken pairwise distinct. To apply the *i.h.* on q_k ($1 \le k \le n$), we take the linear context to be $\mathrm{LL}\langle\Diamond[x_i^j/q_i^j]_{1\le j\le m_i, k<i\le n}\rangle$, which verifies the hypothesis of the statement since by definition of the MFEs, $y \notin \cup_{1\le j\le m_i, k<i\le n} \mathrm{fv}(q_i^j)$. By the *i.h.* $[q_i^1; \dots; q_i^{m_i}]$ are the MFEs of $\lambda y.q_i$ for each i. Therefore, since $[q_1; \dots; q_n]$ are the MFEs of $\lambda x.p'$, the terms $[q_1^1; \dots; q_n^{m_n}]$ are also the MFEs of $\lambda y.\lambda x.p'$. □

Corollary 2.35 (Correctness of $\to_{\mathrm{st}}$). *Let $p \in \mathtt{T}_P$ and $[q_1; \dots; q_n]$ be the MFEs of $\lambda y.p$. Then $\lambda y.z[z/p] \Downarrow_{\mathrm{st}} \lambda y.\{\!\{p\}\!\}^{\{y\}}\langle x_1, \dots, x_n\rangle[x_i/q_i]_{i\le n}$ where the variables $x_1, \dots, x_n$ are fresh and pairwise distinct.*

Proof. By lemma 2.34, there is an n-ary pure context c such that $\lambda y.z[z/p] \to^{*}_{\mathrm{nsub}} t = \lambda y.c\langle x_1, \dots, x_n\rangle[x_i/q_i]_{1\le i\le n}$, where $[q_1; \dots; q_n]$ are the MFEs of $\lambda y.p$. Thus, by the alternative definition of skeleton, c is $\{\!\{p\}\!\}^{\{y\}}$. Moreover, t is the nsub-nf of $\lambda y.z[z/p]$ because no more base $\mapsto^{y}$-reduction steps can be applied to the list of explicit substitutions since y is not free in $q_1, \dots, q_n$ by definition of MFE. □

From the fact that the two definitions of skeleton are equivalent, and from both proofs of correctness (lemma 2.30 and corollary 2.35), we infer the equivalence between the small-step and the big-step splitting semantics (figure 2.3 and figure 2.2 respectively). Since the small-step semantics is contained in λR, we use it to build our call-by-need strategy using node replication.

Another interesting question concerns the splitting semantics for terms with explicit cuts. It is not always clear what the maximal free expressions are, as this notion depends on the position of the explicit cuts in the term. For instance, take the term $t = \lambda y.z_1[w/xy]z_2$. What should be the MFEs of t? It could be $[z_1; x; z_2]$, or $[z_1 z_2; x]$, or even $[(z_1 z_2)[w/x]]$. Similarly for the skeleton, should it be respectively (1) $\lambda y.\Diamond[w/x\Diamond]\Diamond$, (2) $\lambda y.\Diamond\Diamond$ or (3) $\lambda y.\Diamond$? Solution (1) proposes to keep explicit substitutions in the skeleton. This is not coherent with the semantics of λR and λa, which only substitute pure terms. Solution (2) consists in unfolding the explicit cuts, so that the skeleton is pure. This can easily be obtained by adding the following rule to the definition.

$$\{\!\{t[x/u]\}\!\}^{\theta} := \begin{cases} \{\!\{t\}\!\}^{\theta\cup\{x\}}\{x/\{\!\{u\}\!\}^{\theta}\}, & \text{if } \theta \cup \mathrm{fv}(u) \neq \emptyset \\ \{\!\{t\}\!\}^{\theta}, & \text{otherwise} \end{cases}$$

Indeed, this is the definition of skeleton adopted for the atomic λ-calculus in [GHP13a], where the authors prove that the skeleton of a term with explicit substitutions (but without explicit distributors) can be split from the MFEs.

In cases involving explicit cuts binding no variable, like $[w/xy]$ in the term t above, this definition is a cause of inefficiency: we would prefer solution (3), which avoids duplication of the application node. More generally, many nodes can be duplicated inside a term to reach a bound variable that will finally be erased. For instance, in the term $\lambda y.x_1[w/y]x_2x_3 \dots x_n$, $n-1$ applications nodes will need to be duplicated, and the skeleton would be considered $\lambda y.\Diamond\Diamond\Diamond \dots \Diamond$ (n times) following (2), and simply $\lambda y.\Diamond$ following (3). As another example, the skeleton of $\lambda y.(\lambda z.z[w/y])x$ would be considered $\lambda y.(\lambda z.z)\Diamond$ following (2) and $\lambda y.\Diamond$ following (3). Unfortunately, this definition is hard to specify inductively (and therefore in a big-step semantics) without modifying the term first by permuting the cuts. Interestingly though, giving a small-step semantics for is possible by allowing $\longrightarrow_{\mathrm{st}}$-reduction deep inside the distributors. This is one advantage of the small-steps semantics, that is more flexible.

The call-by-need strategy. We have shown how to implement skeleton extraction. A call-by-need strategy depend on other elements: memoization (given by the explicit cuts), a notion of needed variables and need contexts, and linear substitution.

The **call-by-need strategy** $\longrightarrow_{\mathrm{flneed}}$ (figure 2.4) is defined on the set of terms $\mathtt{U}$, by using closure under the *need contexts*, given by the grammar

$$\mathtt{N} ::= \Diamond \mid \mathtt{N}t \mid \mathtt{N}[x \triangleleft t] \mid \mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\mathtt{N}]$$

where $\mathtt{N}\langle\!\langle \_ \rangle\!\rangle$ denotes capture-free application of contexts (section 2.1.1). Like call-by-name (section 2.3.1), the call-by-need strategy is *weak*, because no *meaningful* reduction steps are performed under abstractions.

$$\begin{array}{lll}
\mathtt{L}\langle \lambda x.p\rangle u & \mapsto_{\mathrm{dB}} & \mathtt{L}\langle p[x/u]\rangle \\
\mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\mathtt{L}\langle \lambda y.p\rangle] & \longrightarrow_{\mathrm{spl}} & \mathtt{L}\langle \mathtt{L}\mathtt{L}\langle \mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\!/\lambda y.p']\rangle\rangle \quad \text{if } \lambda y.z[z/p] \Downarrow_{\mathrm{st}} \lambda y.\mathtt{L}\mathtt{L}\langle p'\rangle \\
\mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\!/v] & \longrightarrow_{\mathrm{sub}} & \mathtt{N}\langle\!\langle v \rangle\!\rangle[x/\!/v]
\end{array}$$

Figure 2.4: call-by-need strategy.

Rule $\mapsto_{\mathrm{dB}}$ is the same one used to define $\longrightarrow_{\mathrm{name}}$. Rule $\mapsto_{\mathrm{spl}}$ only uses node replication operations to compute the skeleton of the abstraction, while rule $\mapsto_{\mathrm{sub}}$ implements one-shot *linear* substitution. There is no rule to substitute a variable, as it is usually done in call-by-need for closed terms [AF97].

Linear substitution as implemented in rule sub is out of scope of the calculus λR. This shows a limitation of λR and λa, both using full substitution to implement fully lazy sharing. Yet, the demand-driven philosophy of call-by-need is generally understood as replacing only some desired instance of one variable [AF97]. This corresponds in particular to the behavior of abstract machines, which make explicit some of the implementation features.

In this work, we have chosen to focus on node replication, and implement it in a generic explicit substitution calculus. A linear calculus for node replication could be considered. A

naive version of the calculus would however be very space-inefficient, as reduction would create a lot of explicit substitutions. Nonetheless, remark that the substitution used in the small-step semantics $\longrightarrow_{\mathrm{st}}$ is linear, thanks to the restriction on terms. This facilitates the design of $\longrightarrow_{\mathrm{flneed}}$ as a strategy of a linear calculus.

Notice that as a particular case of lemma 2.24, $t \in \mathtt{U}$ and $t \longrightarrow_{\mathrm{flneed}} t'$ implies $t' \in \mathtt{U}$. Another interesting property is that $t \longrightarrow_{\mathrm{sub}} t'$ implies $\mathrm{lv}_z(t) \geq \mathrm{lv}_z(t')$. Moreover, $\longrightarrow_{\mathrm{flneed}}$ is deterministic.

Lemma 2.36 (Determinism). *The strategy $\longrightarrow_{\mathrm{flneed}}$ is deterministic.*

Proof. The left hand sides of the rules dB, dist and sub are disjoint. On the other hand, the reduction relation $\longrightarrow_{\mathrm{st}}$ is confluent and terminating by lemma 2.33 so that $\Downarrow_{\mathrm{st}}$ defines a function, thus the relation $\longrightarrow_{\mathrm{flneed}}$ is deterministic. □

Example 2.37. Let $t_0 = (\lambda x.(\mathtt{I}(\mathtt{I}x)))(\lambda y.y\mathtt{I})$. Needed variable occurrences are highlighted in orange.

$$
\begin{aligned}
t_0 &\longrightarrow_{\mathrm{dB}} (\mathtt{I}(\mathtt{I}x))[x/\lambda y.y\mathtt{I}] \longrightarrow_{\mathrm{dB}} x_1[x_1/\mathtt{I}x][x/\lambda y.y\mathtt{I}] \\
&\longrightarrow_{\mathrm{dB}} x_1[x_1/x_2[x_2/x]][x/\lambda y.y\mathtt{I}] \longrightarrow_{\mathrm{spl}} x_1[x_1/x_2[x_2/x]][x/\!\!/\lambda y.yz_1][z_1/\mathtt{I}] \\
&\longrightarrow_{\mathrm{sub}} x_1[x_1/x_2[x_2/\lambda y.yz_1]][x/\!\!/\lambda y.yz_1][z_1/\mathtt{I}] \\
&\longrightarrow_{\mathrm{spl}} x_1[x_1/x_2[x_2/\!\!/\lambda y.yz_2][z_2/z_1]][x/\!\!/\lambda y.yz_1][z_1/\mathtt{I}] \\
&\longrightarrow_{\mathrm{sub}} x_1[x_1/(\lambda y.yz_2)[x_2/\!\!/\lambda y.yz_2][z_2/z_1]][x/\!\!/\lambda y.yz_1][z_1/\mathtt{I}] \\
&\longrightarrow_{\mathrm{spl}} x_1[x_1/\!\!/\lambda y.yz_3][z_3/z_2][x_2/\!\!/\lambda y.yz_2][z_2/z_1][x/\!\!/\lambda y.yz_1][z_1/\mathtt{I}] \\
&\longrightarrow_{\mathrm{sub}} (\lambda y.yz_3)[x_1/\!\!/\lambda y.yz_3][z_3/z_2][x_2/\!\!/\lambda y.yz_2][z_2/z_1][x/\!\!/\lambda y.yz_1][z_1/\mathtt{I}]
\end{aligned}
$$

In order to characterize flneed-nfs, we use the notion of **needed free variables** $\mathrm{ndv}(t)$ of a term t, defined as:

$$
\begin{aligned}
&\mathrm{ndv}(x) \coloneqq \{x\} && \mathrm{ndv}(t[y/u]) \coloneqq \begin{cases} \mathrm{ndv}(u) & \text{if } y \in \mathrm{ndv}(t) \\ \mathrm{ndv}(t) & \text{if } y \notin \mathrm{ndv}(t) \end{cases} \\
&\mathrm{ndv}(tu) \coloneqq \mathrm{ndv}(t) && \mathrm{ndv}(t[x/\!\!/u]) \coloneqq \mathrm{ndv}(t) \\
&\mathrm{ndv}(\lambda x.t) \coloneqq \emptyset
\end{aligned}
$$

Notice that $\mathrm{ndv}(t)$ is always either a singleton or the empty set. Thus *e.g.* $\mathrm{ndv}(x[y/\!\!/\mathtt{I}]\mathtt{I}) = \{x\}$ and $\mathrm{ndv}((xy_1)[x/zy_2]) = \{z\}$. In particular, $x \in \mathrm{ndv}(t)$ implies $x \in \mathrm{fv}(t)$.

Lemma 2.38. *Let $t \in U$. Then $x \in \mathrm{ndv}(t)$ iff there exists a context $\mathtt{N}$ such that $t = \mathtt{N}\langle\!\langle x \rangle\!\rangle$.*

Proof. We start with the left-to-right implication. Let $x \in \mathrm{ndv}(t)$. By induction on t.

Case $t = x$. We take $\mathtt{N} = \Diamond$.

Case $t = t'u$. By the *i.h.* there exists $\mathtt{N}'$ such that $t' = \mathtt{N}'\langle\!\langle x \rangle\!\rangle$. We then take $\mathtt{N} = \mathtt{N}'u$.

Case $t = t'[y/u]$. By α-conversion we can assume $x \neq y$. Either $x \in \mathrm{ndv}(t')$ or ($x \in$

$\mathrm{ndv}(u)$ and $y \in \mathrm{ndv}(t')$). In the first case, there exists by the *i.h.* on t' a context $\mathtt{N}'$ such that $t' = \mathtt{N}'\langle\!\langle x \rangle\!\rangle$. We then take $\mathtt{N} = \mathtt{N}'[y/u]$. In the second case, there exists by the *i.h.* on t' a context $\mathtt{N}_1$ such that $t' = \mathtt{N}_1\langle\!\langle y \rangle\!\rangle$. By the *i.h.* on u we have $u = \mathtt{N}_2\langle\!\langle x \rangle\!\rangle$. We then take $\mathtt{N} = \mathtt{N}_1\langle\!\langle y \rangle\!\rangle[y/\mathtt{N}_2]$.

Case $t = t'[x/\!/u]$. By the *i.h.* there exists $\mathtt{N}'$ such that $t' = \mathtt{N}'\langle\!\langle x \rangle\!\rangle$. We then take $\mathtt{N} = \mathtt{N}'[x/\!/u]$.

We continue with the right-to-left implication. Let $t = \mathtt{N}\langle\!\langle x \rangle\!\rangle$. By induction on $\mathtt{N}$.

Case $\mathtt{N} = \Diamond$. Then $t = x$ and $\mathrm{ndv}(t) = \{x\}$.

Case $\mathtt{N} = \mathtt{N}'u$. Then $t = t'u$ and by the *i.h.* $x \in \mathrm{ndv}(t')$, so $x \in \mathrm{ndv}(t)$ by definition.

Case $\mathtt{N} = \mathtt{N}'[x \triangleleft u]$. Then $t = t'[x \triangleleft u]$ and by the *i.h.* $x \in \mathrm{ndv}(t')$, so $x \in \mathrm{ndv}(t)$ by definition.

Case $\mathtt{N} = \mathtt{N}_1\langle\!\langle y \rangle\!\rangle[y/\mathtt{N}_2]$. Then $t = t'[y/u]$, where $y \in \mathrm{fv}(t')$. By the *i.h.* $x \in \mathrm{ndv}(u)$, so $x \in \mathrm{ndv}(t)$. □

Terms of $\mathtt{U}$ in flneed-nf can be characterized by the grammar $\mathrm{NF}_{\mathrm{flneed}}$, defined upon the grammar of neutral terms $\mathrm{NE}_{\mathrm{flneed}}$. Notice that name-nfs are also flneed-nfs.

$$\begin{aligned}
\mathrm{NF}_{\mathrm{flneed}} &::= \mathrm{L}\langle \lambda x.t \rangle \mid \mathrm{NE}_{\mathrm{flneed}} \\
\mathrm{NE}_{\mathrm{flneed}} &::= x \mid \mathrm{NE}_{\mathrm{flneed}}\ t \mid \mathrm{NE}_{\mathrm{flneed}}[x \triangleleft u] \text{ where } x \notin \mathrm{ndv}(\mathrm{NE}_{\mathrm{flneed}}) \\
&\quad \mid \mathrm{NE}_{\mathrm{flneed}}[x/\,\mathrm{NE}'_{\mathrm{flneed}}] \text{ where } x \in \mathrm{ndv}(\mathrm{NE}_{\mathrm{flneed}})
\end{aligned}$$

Lemma 2.39. *Let $t \in U$. Then $t \in \mathrm{NF}_{\mathrm{flneed}}$ iff t is in* flneed-*nf.*

Proof. We first show that $t \in \mathrm{NE}_{\mathrm{flneed}}$ iff t is in flneed-nf and t is not an answer. The left-to-right implication is by induction on $t \in \mathrm{NE}_{\mathrm{flneed}}$.

Case $t = x$. This case is straightforward.

Case $t = t'u$ where $t' \in \mathrm{NE}_{\mathrm{flneed}}$. By the *i.h.* t' is in flneed-nf and is not an answer, so it is not possible to apply any dB-reduction at the root. Then t is in flneed-nf, and since it is an application it is not an answer.

Case $t = t'[x \triangleleft u]$ where $t' \in \mathrm{NE}_{\mathrm{flneed}}$ and $x \notin \mathrm{ndv}(t')$. By the *i.h.* t' is in flneed-nf and it is not an answer. Moreover, we cannot apply rules $\to_{\mathrm{spl}}$ nor $\to_{\mathrm{sub}}$ because by lemma 2.38 there is no context $\mathtt{N}$ surrounding x. Then t is in flneed-nf and is not an answer.

Case $t = t'[x/u]$ where $t', u \in \mathrm{NE}_{\mathrm{flneed}}$ and $x \in \mathrm{ndv}(t')$. By the *i.h.* t' and u are in flneed-nf and are not answers. We cannot apply rule $\to_{\mathrm{spl}}$ because u is not an answer. Then t is in flneed-nf and is not an answer.

The right-to-left implication is by induction on t.

Case $t = x$. Immediate.

Case $t = t'u$. Then t' is in flneed-nf and is not an answer (otherwise dB would be applicable). By the *i.h.* $t' \in \mathrm{NE}_{\mathrm{flneed}}$ and thus $t \in \mathrm{NE}_{\mathrm{flneed}}$.

Case $t = t'[x/u]$. Then t' is in flneed-nf and is not an answer. By the *i.h.* $t' \in \mathrm{NE}_{\mathrm{flneed}}$. There are two cases. If $x \notin \mathrm{ndv}(t')$, then $t \in \mathrm{NE}_{\mathrm{flneed}}$ by definition and we are done. Otherwise $x \in \mathrm{ndv}(t')$, and we get $t' = \mathbb{N}\langle\langle x \rangle\rangle$ by lemma 2.38. Thus u cannot be an answer because $\to_{\mathrm{spl}}$ would apply. Moreover, u is in flneed-nf because otherwise t would not be in flneed-nf. Thus, $u \in \mathrm{NE}_{\mathrm{flneed}}$ by the *i.h.* and we get $t \in \mathrm{NE}_{\mathrm{flneed}}$ by definition.

Case $t = t'[x/\!\!/u]$. We have $x \notin \mathrm{ndv}(t')$, because $\to_{\mathrm{sub}}$ does not apply. By the *i.h.* $t' \in \mathrm{NE}_{\mathrm{flneed}}$, so that $t \in \mathrm{NE}_{\mathrm{flneed}}$.

Neutral terms are also normal. Answers are normal because the calculus is weak and they belong to the grammar $\mathrm{NF}_{\mathrm{flneed}}$. □

2.4 A Type System for λR

This section introduces a quantitative type system $\cap R$ for λR. Non-idempotent intersection [Gar94] has one main advantage over the idempotent model [BDS09]: it gives *quantitative* information about the length of reduction sequences to normal forms [dCar07]. Indeed, not only typability and normalization can be proved to be equivalent, but a measure based on type derivations provides an *upper bound* to normalizing reduction sequences. This was extensively investigated in different logical/computational frameworks [AGL19; Buc+20; CG14; Ehr12; Kes16; KV20]. However, no quantitative result based on types exists in the literature for the node replication model, even in the formulation of (non-idempotent) intersection types for open deduction [GHP21]. The typing rules of our system are in themselves not surprising (see [KV14]), but they provide a handy quantitative characterization of fully lazy normalization (section 2.5).

The **type system** is called $\cap R$ and presented in figure 2.5. The grammar of types is the same as in section 1.3.2.2, with in particular a special type constant $\mathtt{a}$ used to type terms reducing to normal abstractions. The only difference with the systems for ES introduced in section 1.3.2.2 is that the rule (CUT) is generalized to explicit cuts. The **size of a type derivation** $\mathrm{sz}(\Phi)$ is defined as the number of its rules (ABS), (APP) and (ANS).

As usual, the typing system is *relevant*:

Property 2.40 (Relevance). *If* $\Phi = \Gamma \Vdash t : \sigma$, *then* $\mathrm{dom}(\Gamma) \subseteq \mathrm{fv}(t)$.

Proof. Straightforward by induction on the typing derivation. □

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (\text{AX}) \qquad \frac{\Gamma; x : \mathscr{M} \vdash t : \sigma}{\Gamma \vdash \lambda x.t : \mathscr{M} \to \sigma}\ (\text{ABS}) \qquad \frac{}{\emptyset \vdash \lambda x.t : \mathtt{a}}\ (\text{ANS})$$

$$\frac{\Gamma \vdash t : \mathscr{M} \to \sigma \qquad \Delta \vdash u : \mathscr{M}}{\Gamma \uplus \Delta \vdash tu : \sigma}\ (\text{APP}) \qquad \frac{\Gamma; x : \mathscr{M} \vdash t : \sigma \qquad \Delta \vdash u : \mathscr{M}}{\Gamma \uplus \Delta \vdash t[x \triangleleft u] : \sigma}\ (\text{CUT})$$

$$\frac{(\Gamma_i \vdash t : \sigma_i)_{i \in I}}{\uplus_{i \in I} \Gamma_i \vdash t : [\sigma_i]_{i \in I}}\ (\text{MANY})$$

Figure 2.5: Typing System $\cap R$.

Example 2.41. The following tree is a type derivation (called Φ_u) in system $\cap R$ for the term $u = x[x/yz]$.

$$\dfrac{\dfrac{}{x : [\tau] \vdash x : \tau}\ (\text{AX}) \qquad \dfrac{\dfrac{\dfrac{}{y : [[\tau] \to \tau] \vdash y : [\tau] \to \tau}\ (\text{AX}) \qquad \dfrac{\dfrac{}{z : [\tau] \vdash z : \tau}\ (\text{AX})}{z : [\tau] \vdash z : [\tau]}\ (\text{MANY})}{y : [[\tau] \to \tau], z : [\tau] \vdash yz : \tau}\ (\text{APP})}{y : [[\tau] \to \tau], z : [\tau] \vdash yz : [\tau]}\ (\text{MANY})}{y : [[\tau] \to \tau], z : [\tau] \vdash x[x/yz] : \tau}\ (\text{CUT})$$

Type derivations can be measured by triples. We use a + operation on triples as pointwise addition: $(n_1, n_2, n_3) + (m_1, m_2, m_3) = (n_1 + m_1, n_2 + m_2, n_3 + m_3)$. These triples are computed by a **weighted derivation level** function defined on typing derivations as $\mathrm{D}(\Phi) \coloneqq \mathrm{M}(\Phi, 1)$, where $\mathrm{M}(\Phi, \cdot)$ is inductively defined below. In the cases (ABS), (APP) and (CUT), we let Φ_t (resp. Φ_u) be the subderivation of the type of t (resp. Φ_u) and in (MANY) we let Φ_t^i be the i-th derivation of the type of t for each $i \in I$.

Case (AX). $\mathrm{M}(\Phi_x, m) = (0, 0, 1)$,

Case (ABS). $\mathrm{M}(\Phi_{\lambda x.t}, m) = \mathrm{M}(\Phi_t, m) + (1, m, 0)$.

Case (ANS). $\mathrm{M}(\Phi_{\lambda x.t}, m) = (1, m, 0)$.

Case (APP). $\mathrm{M}(\Phi_{tu}, m) = \mathrm{M}(\Phi_t, m) + \mathrm{M}(\Phi_u, m) + (1, m, 0)$.

Case (CUT). $\mathrm{M}\left(\Phi_{t[x \triangleleft u]}, m\right) = \mathrm{M}(\Phi_t, m) + \mathrm{M}(\Phi_u, m + \mathrm{lv}_x(t) + \mathrm{es}([x \triangleleft u]))$.

Case (MANY). $\mathrm{M}(\Phi_t, m) = \sum_{i \in I} \mathrm{M}\left(\Phi_t^i, m\right)$.

Intuitively, the first component of the triple $\mathrm{M}(\Phi, m)$ counts the number of application and abstraction rules in the typing derivation. This is simply the size $\mathrm{sz}(\Phi)$ of the derivation,

does not depend on m and decreases at each dB-step. The second component also counts the number of application and abstractions rules, but *weighted* by the level of the constructor. This component decreases with $\to_{\text{abs}}$ and $\to_{\text{app}}$ reductions. The third component counts the number of axiom rules, and does not depend on m. It decreases with substitutions that occur in $\to_{\text{var}}$ and $\to_{\text{dist}}$-steps.

Example 2.42. Take the derivation Φ_u from example 2.41. Its measure is $\mathrm{D}(\Phi_u) = (1,2,3)$. Moreover, for $x[x/yz] \to_{\text{app}} (x_1x_2)[x_1/y][x_2/z]$ we have

$$\Phi_{u'} = y : [\sigma], z : [\tau] \Vdash (x_1x_2)[x_1/y][x_2/z] : \tau$$

and $\mathrm{D}(\Phi_{u'}) = (1,1,4)$.

Lemma 2.43. *For all derivation Φ and all $m, n \in \mathbb{N}$ with $m > n$, $\mathrm{M}(\Phi, m) = \mathrm{M}(\Phi, n) + (0, (m-n) * \mathrm{sz}(\Phi), 0)$.*

Proof. By induction on Φ.

Case $\Phi = \dfrac{}{x : [\sigma] \vdash x : \sigma}$. Then, $\mathrm{M}(\Phi, m) = (0,0,1) = (0,0,1) + (0, 0 + (m-n) * 0, 0)$.

Case $\Phi = \dfrac{\Phi_t = \Gamma; y : \mathscr{M} \Vdash t : \tau}{\Gamma \vdash \lambda y.t : \mathscr{M} \to \tau}$. Then

$$\begin{aligned}
\mathrm{M}(\Phi, m) &= \mathrm{M}(\Phi_t, m) + (1, m, 0) \\
&=_{i.h.} \mathrm{M}(\Phi_t, n) + (0, (m-n) * \mathrm{sz}(\Phi_t), 0) + (1, n, 0) + (0, m-n, 0) \\
&= \mathrm{M}(\Phi, n) + (0, (m-n) * \mathrm{sz}(\Phi_t), 0) + (0, m-n, 0) \\
&= \mathrm{M}(\Phi, n) + (0, (m-n) * (\mathrm{sz}(\Phi_t) + 1), 0) \\
&= \mathrm{M}(\Phi, n) + (0, (m-n) * \mathrm{sz}(\Phi), 0)
\end{aligned}$$

Case $\Phi = \dfrac{}{\vdash \lambda x.t : \mathsf{a}}$. Then, $\mathrm{M}(\Phi, m) = (1, m, 0) = \mathrm{M}(\Phi, n) + (0, (m-n) * \mathrm{sz}(\Phi), 0)$.

Case $\Phi = \dfrac{\Phi_t = \Gamma \Vdash t : \mathscr{M} \to \tau \qquad \Phi_u = \Delta \Vdash u : \mathscr{M}}{\Gamma \uplus \Delta \vdash tu : \tau}$. Then

$$\begin{aligned}
\mathrm{M}(\Phi, m) &= \mathrm{M}(\Phi_t, m) + \mathrm{M}(\Phi_u, m) + (1, m, 0) \\
&=_{i.h.} \mathrm{M}(\Phi_t, n) + (0, (m-n) * \mathrm{sz}(\Phi_t), 0) \\
&\quad + \mathrm{M}(\Phi_u, n) + (0, (m-n) * \mathrm{sz}(\Phi_u), 0) + (1, n, 0) + (0, m-n, 0) \\
&= \mathrm{M}(\Phi, n) + (0, (m-n) * (\mathrm{sz}(\Phi_t) + \mathrm{sz}(\Phi_u) + 1), 0) \\
&= \mathrm{M}(\Phi, n) + (0, (m-n) * \mathrm{sz}(\Phi), 0)
\end{aligned}$$

Case $\Phi = \dfrac{\Phi_t = \Gamma; x : \mathscr{M} \Vdash t : \sigma \qquad \Phi_u = \Delta \Vdash u : \mathscr{M}}{\Gamma \uplus \Delta \vdash t[x \triangleleft u] : \tau}$. Then

$$\begin{aligned}
\mathsf{M}(\Phi, m) &= \mathsf{M}(\Phi_t, m) + \mathsf{M}(\Phi_u, m + \mathrm{lv}_x(t) + \mathrm{es}([x \triangleleft u])) \\
&=_{i.h.} \mathsf{M}(\Phi_t, n) + (0, (m-n) * \mathrm{sz}(\Phi_t), 0) + \mathsf{M}(\Phi_u, n + \mathrm{lv}_x(t) + \mathrm{es}([x \triangleleft u])) \\
&\quad + (0, (m-n) * \mathrm{sz}(\Phi_u), 0) \\
&= \mathsf{M}(\Phi, n) + (0, (m-n) * (\mathrm{sz}(\Phi_t) + \mathrm{sz}(\Phi_u)), 0) \\
&= \mathsf{M}(\Phi, n) + (0, (m-n) * \mathrm{sz}(\Phi), 0)
\end{aligned}$$

□

Lemma 2.44 (Split). *Let $\Phi = \Delta \Vdash u : \mathscr{M}$ such that $\mathscr{M} = \sqcup_{i \in I} \mathscr{M}_i$ for $I \neq \emptyset$. Then there are derivations $\Phi_i = \Delta_i \Vdash u : \mathscr{M}_i$ such that $\Delta = +_{i \in I} \Delta_i$ and $\mathsf{M}(\Phi, m) = \sum_{i \in I} \mathsf{M}\left(\Phi^i, m\right)$.*

Proof. Straightforward. □

2.5 Observational Equivalence

The type system $\cap R$ characterizes normalization of both name and flneed strategies as follows: every typable term normalizes and every normalizable term is typable. In this sense, system $\cap R$ can be seen as a (quantitative) *model* [BE01] of our call-by-name and call-by-need strategies. We prove these results by studying the appropriate lemmas, notably weighted subject reduction and weighted subject expansion. We then deduce observational equivalence between the name and the flneed strategies from the fact that their associated normalization properties are both fully characterized by the same typing system.

Soundness. Soundness of system $\cap R$ w.r.t. both $\to_{\mathrm{name}}$ and $\to_{\mathrm{flneed}}$ is investigated in this section. More precisely, we show that typable terms are normalizing for both strategies. In contrast to reducibility techniques needed to show this kind of result for simple [GHP13a] or idempotent intersection types, soundness is achieved here by relatively simple combinatorial arguments based again on decreasing measures. We start by studying the interaction between system $\cap R$ and linear as well as full substitution.

Lemma 2.45 (Partial substitution). *Let $\Phi = \Gamma; x : \mathscr{M} \Vdash \mathsf{C}\langle\langle x \rangle\rangle : \sigma$ and $\sqsubseteq$ denote multiset inclusion. Then, there exists $\mathscr{N} \sqsubseteq \mathscr{M}$ such that for every $\Phi_u = \Delta \Vdash u : \mathscr{N}$ we have $\Psi = \Gamma \uplus \Delta; x : \mathscr{M} \setminus \mathscr{N} \Vdash \mathsf{C}\langle\langle u \rangle\rangle : \sigma$ and, for every $m \in \mathbb{N}$, $\mathsf{M}(\Psi, m) = \mathsf{M}(\Phi, m) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_\Diamond(\mathsf{C})\right) - (0, 0, |\mathscr{N}|)$.*

Proof. By induction on Φ.

Case (Ax). Then $\Phi = \dfrac{}{x : [\sigma] \vdash x : \sigma}$, $\mathscr{N} = [\sigma]$ and $\Psi = \Phi_u = \Delta \Vdash u : [\sigma]$. So, $\mathsf{M}(\Psi, m) = \mathsf{M}(\Phi_u, m) = (0,0,1) + \mathsf{M}(\Phi_u, m+0) - (0,0,1) = \mathsf{M}(\Phi, m) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_\Diamond(C)\right) - (0,0,|\mathscr{N}|)$.

Case (ABS). Then $\Phi = \dfrac{\Phi' = \Gamma; x : \mathscr{M}; y : \mathscr{M}_y \Vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : \tau}{\Gamma; x : \mathscr{M} \vdash \lambda y.\mathtt{C}'\langle\!\langle x \rangle\!\rangle : \mathscr{M}_y \to \tau}$. By α-conversion, we can assume that $y \notin \mathrm{dom}(\Delta)$ so that $(\Gamma; y : \mathscr{M}_y) \uplus \Delta = \Gamma \uplus \Delta; y : \mathscr{M}_y$. By using the *i.h.* we can then construct $\Psi = \dfrac{\Psi' = \Gamma \uplus \Delta; x : \mathscr{M} \setminus \mathscr{N}; y : \mathscr{M}_y \Vdash \mathtt{C}'\langle\!\langle u \rangle\!\rangle : \tau}{\Gamma \uplus \Delta; x : \mathscr{M} \setminus \mathscr{N} \vdash \lambda y.\mathtt{C}'\langle\!\langle u \rangle\!\rangle : \mathscr{M}_y \to \tau}$. Then,

$$\begin{aligned}
\mathsf{M}(\Psi, m) &= \mathsf{M}(\Psi', m) + (1, m, 0) \\
&=_{i.h.} \mathsf{M}(\Phi', m) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_\Diamond(\mathtt{C}')\right) - (0, 0, |\mathscr{N}|) + (1, m, 0) \\
&= \mathsf{M}(\Phi, m) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_\Diamond(\lambda y.\mathtt{C}')\right) - (0, 0, |\mathscr{N}|)
\end{aligned}$$

Case (ANS). Then $\Phi = \dfrac{}{\varnothing \vdash \lambda y.\mathtt{C}'\langle\!\langle x \rangle\!\rangle : \mathtt{a}}$. We can build $\Psi = \dfrac{}{\varnothing \vdash \lambda y.\mathtt{C}'\langle\!\langle u \rangle\!\rangle : \mathtt{a}}$. In particular, we have $\mathscr{M} = \mathscr{N} = [\,]$, and thus Φ_u comes from the application of the (MANY) rule to 0 premises, so that $\mathsf{M}\left(\Phi_u, m + \mathrm{lv}_\Diamond(\mathtt{C})\right) = (0, 0, 0)$. We have $\mathsf{M}(\Phi, m) = \mathsf{M}(\Psi, m) = \mathsf{M}(\Psi, m) + (0, 0, 0) - (0, 0, 0)$.

Case (APP) left. Then

$$\Phi = \frac{\Phi_1 = \Gamma_1; x : \mathscr{M}_1 \Vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : \mathscr{M}' \to \sigma \qquad \Phi_2 = \Gamma_2; x : \mathscr{M}_2 \Vdash t : \mathscr{M}'}{\Gamma_1 \uplus \Gamma_2; x : \mathscr{M} \vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle t : \sigma}$$

By *i.h.* there is $\mathscr{N} \sqsubseteq \mathscr{M}_1$ such that we can construct

$$\Psi = \frac{\Psi_1 = \Gamma_1 \uplus \Delta; x : \mathscr{M}_1 \setminus \mathscr{N} \Vdash \mathtt{C}'\langle\!\langle u \rangle\!\rangle : \mathscr{M}' \to \sigma \qquad \Phi_2 = \Gamma_2; x : \mathscr{M}_2 \Vdash t : \mathscr{M}'}{\Gamma_1 \uplus \Gamma_2 \uplus \Delta; x : \mathscr{M} \setminus \mathscr{N} \vdash \mathtt{C}'\langle\!\langle u \rangle\!\rangle t : \sigma}$$

because $\mathscr{M} \setminus \mathscr{N} = \mathscr{M}_1 \setminus \mathscr{N} \sqcup \mathscr{M}_2$. We have

$$\begin{aligned}
\mathsf{M}(\Psi, m) &= \mathsf{M}(\Psi_1, m) + \mathsf{M}(\Phi_2, m) + (1, m, 0) \\
&=_{i.h.} \mathsf{M}(\Phi_1, m) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_\Diamond(\mathtt{C}')\right) - (0, 0, |\mathscr{N}|) + \mathsf{M}(\Phi_2, m) + (1, m, 0) \\
&= \mathsf{M}(\Phi, m) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_\Diamond(\mathtt{C}'t)\right) - (0, 0, |\mathscr{N}|)
\end{aligned}$$

Case (APP) right. Then

$$\Phi = \frac{\Phi_1 = \Gamma_1; x : \mathscr{M}_1 \Vdash t : [\tau_i]_{i \in I} \to \sigma \qquad \dfrac{\left(\Phi_2^i = \Gamma_2^i; x : \mathscr{M}_2^i \Vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : \tau_i\right)_{i \in I}}{\Gamma_2; x : \mathscr{M}_2 \vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : [\tau_i]_{i \in I}}}{\Gamma_1 \uplus \Gamma_2; x : \mathscr{M} \vdash t\mathtt{C}'\langle\!\langle x \rangle\!\rangle : \sigma}$$

where $\mathscr{M}_2 = \sqcup_{i \in I} \mathscr{M}_2^i$ and $\Gamma_2 = \uplus_{i \in I} \Gamma_2^i$. lemma 2.44 gives $\Phi_u^i = \Delta^i \Vdash u : \mathscr{N}^i$ such that $\mathscr{N}^i \sqsubseteq \mathscr{M}_2^i$ for all $i \in I$ and $\mathscr{N} = \sqcup_{i \in I} \mathscr{N}^i$. Moreover, $\mathsf{M}(\Phi_u, m) = \sum_{i \in I} \mathsf{M}\left(\Phi_u^i, m\right)$. Using

the *i.h.* we can construct Ψ below:

$$\frac{\Phi_1 = \Gamma_1; x : \mathscr{M}_1 \Vdash t : [\tau_i]_{i\in I} \to \sigma \qquad \frac{\left(\Psi_2^i = \Gamma_2^i \uplus \Delta^i; x : \mathscr{M}_2^i \setminus \mathscr{N}^i \Vdash \mathtt{C}'\langle\!\langle u \rangle\!\rangle : \tau_i\right)_{i\in I}}{\Gamma_2 \uplus \Delta; x : \mathscr{M}_2 \setminus \mathscr{N} \vdash \mathtt{C}'\langle\!\langle u \rangle\!\rangle : [\tau_i]_{i\in I}}}{\Gamma_1 \uplus \Gamma_2 \uplus \Delta; x : \mathscr{M} \setminus \mathscr{N} \vdash t\mathtt{C}'\langle\!\langle u \rangle\!\rangle : \sigma}$$

where $\mathscr{M} \setminus \mathscr{N} = \mathscr{M}_1 \sqcup \mathscr{M}_2 \setminus \mathscr{N}$. We have

$$\begin{aligned}
\mathsf{M}(\Psi, m) &= \mathsf{M}(\Phi_1, m) + \sum_{i\in I} \mathsf{M}\left(\Psi_2^i, m\right) + (1, m, 0) \\
&=_{i.h.} \mathsf{M}(\Phi_1, m) + \sum_{i\in I} \left(\mathsf{M}\left(\Phi_2^i, m\right) + \mathsf{M}\left(\Phi_u^i, m + \mathrm{lv}_\Diamond(\mathtt{C}')\right) - (0, 0, |\mathscr{N}^i|)\right) + (1, m, 0) \\
&= \mathsf{M}(\Phi, m) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_\Diamond(t\mathtt{C}')\right) - (0, 0, |\mathscr{N}|)
\end{aligned}$$

Case (CUT) left. Then

$$\Phi = \frac{\Phi_1 = \Gamma_1; x : \mathscr{M}_1; y : \mathscr{M}_y \Vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : \sigma \qquad \Phi_2 = \Gamma_2; x : \mathscr{M}_2 \Vdash t : \mathscr{M}_y}{\Gamma_1 \uplus \Gamma_2; x : \mathscr{M} \vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle[y \triangleleft t] : \sigma}$$

We can assume by α-conversion that $x \notin \mathrm{fv}(u)$ and $y \notin \mathrm{fv}(u)$ thus, by the Relevance property 2.40, $y \notin \mathrm{dom}(\Delta)$ so that in particular $(\Gamma_1; y : \mathscr{M}_y) \uplus \Delta = \Gamma_1 \uplus \Delta; y : \mathscr{M}_y$. By using the *i.h.* we can then construct

$$\Psi = \frac{\Psi_1 = \Gamma_1 \uplus \Delta; x : \mathscr{M}_1 \setminus \mathscr{N}; y : \mathscr{M}_y \Vdash \mathtt{C}'\langle\!\langle u \rangle\!\rangle : \sigma \qquad \Phi_2 = \Gamma_2; x : \mathscr{M}_2 \Vdash t : \mathscr{M}_y}{\Gamma_1 \uplus \Gamma_2 \uplus \Delta; x : \mathscr{M} \setminus \mathscr{N} \vdash \mathtt{C}'\langle\!\langle u \rangle\!\rangle[y \triangleleft t] : \sigma}$$

because $\mathscr{M} \setminus \mathscr{N} = \mathscr{M}_1 \setminus \mathscr{N} \sqcup \mathscr{M}_2$. We have:

$$\begin{aligned}
\mathsf{M}(\Psi, m) &= \mathsf{M}(\Psi_1, m) + \mathsf{M}\left(\Phi_2, m + \mathrm{lv}_y(\mathtt{C}'\langle\!\langle u \rangle\!\rangle) + \mathrm{es}([y \triangleleft t])\right) \\
&=_{i.h.} \mathsf{M}(\Phi_1, m) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_\Diamond(\mathtt{C}')\right) - (0, 0, |\mathscr{N}|) \\
&\quad + \mathsf{M}\left(\Phi_2, m + \mathrm{lv}_y(\mathtt{C}'\langle x \rangle)\right) + \mathrm{es}([y \triangleleft t]) \\
&= \mathsf{M}(\Phi, m) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_\Diamond(\mathtt{C}'[y \triangleleft t])\right) - (0, 0, |\mathscr{N}|)
\end{aligned}$$

Case (CUT) right. Then

$$\Phi = \frac{\Phi_1 = \Gamma_1; x : \mathscr{M}_1; y : [\tau_i]_{i\in I} \Vdash t : \sigma \qquad \frac{\left(\Phi_2^i = \Gamma_2^i; x : \mathscr{M}_2^i \Vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : \tau_i\right)_{i\in I}}{\Gamma_2; x : \mathscr{M}_2 \vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : [\tau_i]_{i\in I}}}{\Gamma_1 \uplus \Gamma_2; x : \mathscr{M} \vdash t[y \triangleleft \mathtt{C}'\langle\!\langle x \rangle\!\rangle] : \sigma}$$

where $\mathscr{M} = \mathscr{M}_1 \sqcup \mathscr{M}_2$, $\mathscr{M}_2 = \sqcup_{i\in I} \mathscr{M}_2^i$ and $\Gamma_2 = \uplus_{i\in I} \Gamma_2^i$. lemma 2.44 gives $\Phi_u^i = \Delta^i \Vdash u : \mathscr{N}^i$ for all $i \in I$. Moreover, $\mathsf{M}(\Phi_u, m) = \sum_{i\in I} \mathsf{M}\left(\Phi_u^i, m\right)$. By using the *i.h.* we can

construct Ψ below:

$$\frac{\Phi_1 = \Gamma_1; x : \mathscr{M}_1; y : [\tau_i]_{i\in I} \Vdash t : \sigma \qquad \frac{\left(\Psi_2^i = \Gamma_2^i \uplus \Delta^i; x : \mathscr{M}_2^i \setminus \mathscr{N}^i \Vdash \mathtt{C}'\langle\!\langle u \rangle\!\rangle : \tau_i\right)_{i\in I}}{\Gamma_2 \uplus \Delta; x : \mathscr{M}_2 \setminus \mathscr{N} \vdash \mathtt{C}'\langle\!\langle u \rangle\!\rangle : [\tau_i]_{i\in I}}}{\Gamma_1 \uplus \Gamma_2 \uplus \Delta; x : \mathscr{M} \setminus \mathscr{N} \vdash t[y \triangleleft \mathtt{C}'\langle\!\langle u \rangle\!\rangle] : \sigma}$$

because $\mathscr{M} \setminus \mathscr{N} = \mathscr{M}_1 \sqcup \mathscr{M}_2 \setminus \mathscr{N}$, where $\mathscr{N} = \sqcup_{i\in I}\mathscr{N}^i$. We have

$$\begin{aligned}
\mathsf{M}(\Psi, m) &= \mathsf{M}(\Phi_1, m) + \sum_{i\in I} \mathsf{M}\left(\Psi_2^i, m + \mathrm{lv}_y(t) + \mathrm{es}([y \triangleleft \mathtt{C}'\langle\!\langle u \rangle\!\rangle])\right) \\
&=_{i.h.} \mathsf{M}(\Phi_1, m) + \sum_{i\in I} (\mathsf{M}\left(\Phi_2^i, m + \mathrm{lv}_y(t) + \mathrm{es}([y \triangleleft \mathtt{C}'\langle\!\langle u \rangle\!\rangle])\right) \\
&\qquad + \mathsf{M}\left(\Phi_u^i, m + \mathrm{lv}_y(t) + \mathrm{es}([y \triangleleft \mathtt{C}'\langle\!\langle u \rangle\!\rangle]) + \mathrm{lv}_\Diamond(\mathtt{C}')\right) - (0, 0, |\mathscr{N}^i|)) \\
&= \mathsf{M}(\Phi, m) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_\Diamond(t[y \triangleleft \mathtt{C}'])\right) - (0, 0, |\mathscr{N}|)
\end{aligned}$$

Notice that a special case is when $y \notin \mathrm{fv}(t)$. Then, $I = \emptyset$, $\Gamma = \Gamma_1$, $\mathscr{N} = [\,]$ and $\Phi_u = \emptyset \Vdash u : [\,]$ is made only of a nullary (MANY) rule. Hence, $\Phi = \Phi_1 = \Psi$. □

Corollary 2.46 (Substitution). *If $\Phi_t = \Gamma; x : \mathscr{M} \Vdash t : \sigma$ and $\Phi_u = \Delta \Vdash u : \mathscr{M}$, then $\Phi = \Gamma \uplus \Delta \Vdash t\{x/u\} : \sigma$, and for all $m \in \mathbb{N}$ we have $\mathsf{M}(\Phi, m) \leq \mathsf{M}(\Phi_t, m) + \mathsf{M}(\Phi_u, m + \mathrm{lv}_x(t))$. Moreover, $|\mathscr{M}| > 0$ iff the inequality is strict.*

Proof. The proof is by induction on $|t|_x$.

If $|t|_x = 0$, then by the relevance property 2.40 $\mathscr{M} = [\,]$, so that Φ_u necessarily comes from a (MANY) rule without any premise and thus $\Phi = \Phi_t$. We have $\mathsf{M}(\Phi, m) = \mathsf{M}(\Phi_t, m) + \mathsf{M}(\Phi_u, m + \mathrm{lv}_x(t))$ because $\mathsf{M}(\Phi_u, m + \mathrm{lv}_x(t)) = (0, 0, 0)$.

Otherwise, $|t|_x > 0$ and we can write t as $\mathtt{C}\langle\!\langle x \rangle\!\rangle$. By the partial substitution lemma 2.45, there exists $\mathscr{N} \sqsubseteq \mathscr{M}$ such that for all $\Phi_u^0 = \Delta_0 \Vdash u : \mathscr{N}$, there is $\Phi' = \Gamma \uplus \Delta_0; x : \mathscr{M} \setminus \mathscr{N} \Vdash \mathtt{C}\langle\!\langle u \rangle\!\rangle : \sigma$. By the split lemma 2.44, there are derivations $\Phi_u^1 = \Delta_1 \Vdash u : \mathscr{N}$ and $\Phi_u^2 = \Delta_2 \Vdash u : \mathscr{M} \setminus \mathscr{N}$, where $\Delta = \Delta_1 \uplus \Delta_2$ so that we can apply the partial substitution Lemma to Φ_t and Φ_u^1, and we obtain $\Phi' = \Gamma \uplus \Delta_1; x : \mathscr{M} \setminus \mathscr{N} \Vdash \mathtt{C}\langle\!\langle u \rangle\!\rangle : \sigma$.

Since $\mathrm{lv}_\Diamond(\mathtt{C}) \leq \mathrm{lv}_x(t)$, then $\mathsf{M}(\Phi', m) = \mathsf{M}(\Phi_t, m) + \mathsf{M}\left(\Phi_u^1, m + \mathrm{lv}_\Diamond(\mathtt{C})\right) - (0, 0, |\mathscr{N}|) \leq_{2.43} \mathsf{M}(\Phi_t, m) + \mathsf{M}\left(\Phi_u^1, m + \mathrm{lv}_x(t)\right) - (0, 0, |\mathscr{N}|) \leq \mathsf{M}(\Phi_t, m) + \mathsf{M}\left(\Phi_u^1, m + \mathrm{lv}_x(t)\right)$. Because $x \notin \mathrm{fv}(u)$, $|\mathtt{C}\langle\!\langle u \rangle\!\rangle|_x = |t|_x - 1$. We conclude by applying the *i.h.* on Φ' and Φ_u^2. We get $\Phi = \Gamma \uplus \Delta_1 \uplus \Delta_2 \Vdash \mathtt{C}\langle\!\langle u \rangle\!\rangle\{x/u\} : \sigma = \Gamma \uplus \Delta \vdash t\{x/u\} : \sigma$.

For the measure, we use $\mathrm{lv}_x(\mathtt{C}\langle\!\langle u \rangle\!\rangle) \leq \mathrm{lv}_x(t)$ in order to get $\mathsf{M}(\Phi, m) \leq \mathsf{M}(\Phi', m) + \mathsf{M}\left(\Phi_u^2, m + \mathrm{lv}_x(\mathtt{C}\langle\!\langle u \rangle\!\rangle)\right) \leq \mathsf{M}(\Phi_t, m) + \mathsf{M}\left(\Phi_u^1, m + \mathrm{lv}_x(t)\right) + \mathsf{M}\left(\Phi_u^2, m + \mathrm{lv}_x(t)\right) =_{2.44} \mathsf{M}(\Phi_t, m) + \mathsf{M}(\Phi_u, m + \mathrm{lv}_x(t))$. If $\mathscr{M} \neq [\,]$, then either $\mathscr{N}$ or $\mathscr{M} \setminus \mathscr{N}$ is non-empty, so at least one of the two previous inequalities is strict. □

The key idea to show soundness is that the measure $\mathsf{D}(\cdot)$ decreases w.r.t. the reduction relations $\longrightarrow_{\mathrm{name}}$ and $\longrightarrow_{\mathrm{flneed}}$:

Lemma 2.47 (Weighted subject reduction for $\to_\rho$). *Let* $\Phi_{t_0} = \Gamma \Vdash t_0 : \sigma$ *and* $t_0 \to_\rho t_1$. *Then there exists* $\Phi_{t_1} = \Gamma \Vdash t_1 : \sigma$ *such that* $\mathit{M}\left(\Phi_{t_0}, m\right) = \mathit{M}\left(\Phi_{t_1}, m\right)$ *for every* $m \in \mathbb{N}$.

Proof. Let $t_0 = \mathtt{C}\langle t_0' \rangle$ and $t_1 = \mathtt{C}\langle t_1' \rangle$, where $t_0' \to_\rho t_1'$ is a root step. We reason by induction on $\mathtt{C}$. We first consider the base cases where $\mathtt{C} = \Diamond$.

Case $t_0' = \lambda y.t[x \lhd u] \mapsto_\rho (\lambda y.t)[x \lhd u] = t_1'$, where $y \notin \mathrm{fv}(u)$. There are two possible typing derivations.

1. The typing derivation Φ is equal to

$$\dfrac{\dfrac{\Phi_t = \Gamma'; y : \mathscr{N}; x : \mathscr{M} \Vdash t : \tau \qquad \Phi_u = \Delta_u \Vdash u : \mathscr{M}}{\Gamma' \uplus \Delta_u; y : \mathscr{N} \vdash t[x \lhd u] : \tau}\,(\textsc{cut})}{\Gamma' \uplus \Delta_u \vdash \lambda y.t[x \lhd u] : \mathscr{N} \to \tau}\,(\textsc{abs})$$

   We construct the following derivation Ψ.

$$\dfrac{\dfrac{\Phi_t = \Gamma'; y : \mathscr{N}; x : \mathscr{M} \Vdash t : \tau}{\Gamma'; x : \mathscr{M} \vdash \lambda y.t : \mathscr{N} \to \tau}\,(\textsc{abs}) \qquad \Phi_u = \Delta_u \Vdash u : \mathscr{M}}{\Gamma' \uplus \Delta_u \vdash (\lambda y.t)[x \lhd u] : \mathscr{N} \to \tau}\,(\textsc{cut})$$

   Moreover,

$$\begin{aligned} \mathit{M}(\Phi, m) &= \mathit{M}(\Phi_t, m) + \mathit{M}(\Phi_u, m + \mathrm{lv}_x(t) + \mathrm{es}([x \lhd u])) + (1, m, 0) \\ &= \mathit{M}(\Phi_t, m) + (1, m, 0) + \mathit{M}(\Phi_u, m + \mathrm{lv}_x(\lambda y.t) + \mathrm{es}([x \lhd u])) \\ &= \mathit{M}(\Psi, m)\,. \end{aligned}$$

2. The typing derivation is of the form

$$\Phi = \dfrac{}{\vdash \lambda y.t[x \lhd u] : \mathtt{a}}\,(\textsc{ans})$$

   We construct the following derivation that has the same measure.

$$\Psi = \dfrac{\dfrac{}{\vdash \lambda y.t : \mathtt{a}}\,(\textsc{ans}) \qquad \dfrac{}{\vdash u : [\,]}\,(\textsc{many})}{\vdash (\lambda y.t)[x \lhd u] : \mathtt{a}}\,(\textsc{cut})$$

Case $t_0' = t[x \lhd u]s \mapsto_\rho (ts)[x \lhd u] = t_1'$, where $x \notin \mathrm{fv}(s)$. The typing derivation Φ is equal to:

$$\dfrac{\dfrac{\Phi_t = \Gamma'; x : \mathscr{M} \Vdash t : \mathscr{N} \to \sigma \qquad \Phi_u = \Delta_u \Vdash u : \mathscr{M}}{\Gamma' \uplus \Delta_u \vdash t[x \lhd u] : \mathscr{N} \to \sigma}\,(\textsc{cut}) \qquad \Phi_s = \Delta_s \Vdash s : \mathscr{N}}{\Gamma' \uplus \Delta_u \uplus \Delta_s \vdash t[x \lhd u]s : \sigma}\,(\textsc{app})$$

We construct the following derivation Ψ.

$$\dfrac{\dfrac{\Phi_t = \Gamma'; x : \mathscr{M} \Vdash t : \mathscr{N} \longrightarrow \sigma \qquad \Phi_s = \Delta_s \Vdash s : \mathscr{N}}{\Gamma' \uplus \Delta_s; x : \mathscr{M} \vdash ts : \sigma}\,(\textsc{app}) \quad \Phi_u = \Delta_u \Vdash u : \mathscr{M}}{\Gamma' \uplus \Delta_u \uplus \Delta_s(ts)[x \lhd u] : \sigma}\,(\textsc{cut})$$

Moreover, since $\mathrm{lv}_x(t) = \mathrm{lv}_x(ts)$,

$$\begin{aligned} \mathit{M}(\Phi, m) &= \mathit{M}(\Phi_t, m) + \mathit{M}(\Phi_s, m) + (1, m, 0) + \mathit{M}(\Phi_u, m + \mathrm{lv}_x(ts) + \mathrm{es}([x \lhd u])) \\ &= \mathit{M}(\Psi, m) \end{aligned}$$

Case $t_0' = ts[x \lhd u] \mapsto_\rho (ts)[x \lhd u] = t_1'$, where $x \notin \mathrm{fv}(t)$. Let

$$\Phi_{s[x \lhd u]} = \dfrac{\left(\dfrac{\Phi_s^i = \Delta_s^i; x : \mathscr{M}_i \Vdash s : \rho_i \qquad \dfrac{\left(\Phi_u^{i,j} = \Delta_u^{i,j} \Vdash u : \tau_j\right)_{j \in J_i}}{\Delta_u^i \vdash u : \mathscr{M}_i}\,(\textsc{many})}{\Delta_u^i \uplus \Delta_s^i \vdash s[x \lhd u] : \rho_i}\,(\textsc{cut})\right)_{i \in I}}{\Delta_u \uplus \Delta_s \vdash s[x \lhd u] : \mathscr{N}}\,(\textsc{many})$$

The typing derivation Φ is of the form

$$\dfrac{\Phi_t = \Gamma' \Vdash t : \mathscr{N} \longrightarrow \sigma \qquad \Phi_{s[x \lhd u]} = \Delta_u \uplus \Delta_s \Vdash s[x \lhd u] : \mathscr{N}}{\Gamma' \uplus \Delta_u \uplus \Delta_s \vdash ts[x \lhd u] : \sigma}\,(\textsc{app})$$

where $\mathscr{M}_i = [\tau_j]_{j \in J_i}$, $\mathscr{N} = [\rho_i]_{i \in I}$, $\Delta_u^i = \uplus_{j \in J_i} \Delta_u^{i,j}$, $\Delta_u = \uplus_{i \in I} \Delta_u^i$, and $\Delta_s = \uplus_{i \in I} \Delta_s^i$.
Now, let

$$\Phi_s = \dfrac{\dfrac{\left(\Phi_s^i = \Delta_s^i; x : \mathscr{M}_i \Vdash s : \rho_i\right)_{i \in I}}{\Delta_s; x : \mathscr{M} \vdash s : \mathscr{N}}\,(\textsc{many})}{\Gamma' \uplus \Delta_s; x : \mathscr{M} \vdash ts : \sigma}\,(\textsc{app}) \qquad \Phi_u = \dfrac{\left(\Phi_u^{i,j} = \Delta_u^{i,j} \Vdash u : \tau_j\right)_{j \in J_i, i \in I}}{\Delta_u \vdash u : \mathscr{M}}\,(\textsc{many})$$

We construct the following derivation Ψ.

$$\dfrac{\Phi_t = \Gamma' \Vdash t : \mathscr{N} \longrightarrow \sigma \qquad \Phi_s = \Gamma' \uplus \Delta_s; x : \mathscr{M} \Vdash ts : \sigma \qquad \Phi_u = \Delta_u \Vdash u : \mathscr{M}}{\Gamma' \uplus \Delta_u \uplus \Delta_s \vdash (ts)[x \lhd u] : \sigma}$$

where $\mathscr{M} = \sqcup_{i\in I}\mathscr{M}_i$, so that $\mathscr{M} = [\tau_j]_{j\in J_i, i\in I}$. Moreover, because $\mathrm{lv}_x(s) = \mathrm{lv}_x(ts)$,

$$\begin{aligned}\mathsf{M}(\Phi, m) &= \mathsf{M}(\Phi_t, m) + (1, m, 0) \\ &\quad + \sum_{i\in I}\left(\mathsf{M}\left(\Phi_s^i, m\right) + \sum_{j\in J_i}\mathsf{M}\left(\Phi_u^{i,j}, m + \mathrm{lv}_x(s) + \mathrm{es}([x \triangleleft u])\right)\right) \\ &= \mathsf{M}(\Psi, m)\end{aligned}$$

Case $t_0' = t[x \triangleleft u[y \triangleleft s]] \mapsto_\rho t[x \triangleleft u][y \triangleleft s] = t_1'$, where $y \notin \mathrm{fv}(t)$. Let

$$\Phi_{u[y\triangleleft s]} = \dfrac{\left(\dfrac{\Phi_u^i = \Delta_u^i; y : \mathscr{N}_i \Vdash u : \rho_i \qquad \dfrac{\left(\Phi_s^{i,j} = \Delta_s^{i,j} \Vdash s : \tau_j\right)_{j\in J_i}}{\Delta_s^i \vdash s : \mathscr{N}_i}\,(\textsc{many})}{\Delta_u^i \uplus \Delta_s^i \vdash u[y \triangleleft s] : \rho_i}\,(\textsc{cut})\right)_{i\in I}}{\Delta_u \uplus \Delta_s \vdash u[y \triangleleft s] : \mathscr{M}}\,(\textsc{many})$$

The typing derivation Φ is of the form

$$\dfrac{\Phi_t = \Gamma'; x : \mathscr{M} \Vdash t : \sigma \qquad \Phi_{u[y\triangleleft s]} = \Delta_u \uplus \Delta_s \Vdash u[y \triangleleft s] : \mathscr{M}}{\Gamma' \uplus \Delta_u \uplus \Delta_s \vdash t[x \triangleleft u[y \triangleleft s]] : \sigma}\,(\textsc{cut})$$

where $\mathscr{M} = [\rho_i]_{i\in I}$, $\mathscr{N}_i = [\tau_j]_{j\in J_i}$, $\Delta_u = \uplus_{i\in I}\Delta_u^i$, $\Delta_s^i = \uplus_{j\in J_i}\Delta_s^{i,j}$, and $\Delta_s = \uplus_{i\in I}\Delta_s^i$.
Now, let

$$\Phi_{t[x\triangleleft u]} = \dfrac{\Phi_t = \Gamma'; x : \mathscr{M} \Vdash t : \sigma \qquad \dfrac{\left(\Phi_u^i = \Delta_u^i; y : \mathscr{N}_i \Vdash u : \rho_i\right)_{i\in I}}{\Delta_u; y : \mathscr{N} \vdash u : \mathscr{M}}\,(\textsc{many})}{\Gamma' \uplus \Delta_u; y : \mathscr{N} \vdash t[x \triangleleft u] : \sigma}\,(\textsc{cut})$$

We then construct the following derivation Ψ.

$$\dfrac{\Phi_{t[x\triangleleft u]} = \Gamma' \uplus \Delta_u; y : \mathscr{N} \Vdash t[x \triangleleft u] : \sigma \qquad \dfrac{\left(\Phi_s^{i,j} = \Delta_s^{i,j} \Vdash s : \tau_j\right)_{j\in J_i, i\in I}}{\Delta_s \vdash s : \mathscr{N}}\,(\textsc{many})}{\Gamma' \uplus \Delta_u \uplus \Delta_s \vdash t[x \triangleleft u][y \triangleleft s] : \sigma}\,(\textsc{cut})$$

where $\mathscr{N} = \sqcup_{i\in I}\mathscr{N}_i$, so that $\mathscr{N} = [\tau_j]_{j\in J_i, i\in I}$. Moreover, because $y \notin \mathrm{fv}(t)$, we have that $\mathrm{lv}_y(t[x \triangleleft u]) = \mathrm{lv}_x(t) + \mathrm{lv}_y(u) + \mathrm{es}([x \triangleleft u])$ if $y \in \mathrm{fv}(u)$, and $\mathrm{lv}_y(t[x \triangleleft u]) = 0$ otherwise. Now, we show that $\mathsf{M}\left(\Phi_s^{i,j}, m + \mathrm{lv}_x(t) + \mathrm{es}([x \triangleleft u]) + \mathrm{lv}_y(u) + \mathrm{es}([y \triangleleft s])\right) = \mathsf{M}\left(\Phi_s^{i,j}, m + \mathrm{lv}_y(t[x \triangleleft u]) + \mathrm{es}([y \triangleleft s])\right)$. If $y \in \mathrm{fv}(u)$, this is immediate. Otherwise,

by the relevance property 2.40 we have $J_i = [\,]$ for any i thus s is not typed, so that both measures are equal to (0,0,0). Then,

$$\begin{aligned}
\mathsf{M}(\Phi, m) &= \mathsf{M}(\Phi_t, m) + \sum_{i\in I} \mathsf{M}\left(\Phi_u^i, m + \mathrm{lv}_x(t) + \mathrm{es}([x \triangleleft u])\right) \\
&\quad + \sum_{i\in I}\sum_{j\in J_i} \mathsf{M}\left(\Phi_s^{i,j}, m + \mathrm{lv}_x(t) + \mathrm{es}([x \triangleleft u]) + \mathrm{lv}_y(u) + \mathrm{es}([y \triangleleft s])\right) \\
&= \mathsf{M}(\Phi_t, m) + \sum_{i\in I} \mathsf{M}\left(\Phi_u^i, m + \mathrm{lv}_x(t) + \mathrm{es}([x \triangleleft u])\right) \\
&\quad + \sum_{i\in I}\sum_{j\in J_i} \mathsf{M}\left(\Phi_s^{i,j}, m + \mathrm{lv}_y(t[x \triangleleft u]) + \mathrm{es}([y \triangleleft s])\right) \\
&= \mathsf{M}(\Psi, m)
\end{aligned}$$

Now, we analyze all the inductive cases:

Case $\mathtt{C} = \lambda x.\mathtt{C}'$. We have $\sigma = \mathscr{M} \longrightarrow \tau$ and $\Phi' = \Gamma; x : \mathscr{M} \Vdash \mathtt{C}'\langle o\rangle : \tau$. By the *i.h.* there is $\Psi' = \Gamma; x : \mathscr{M} \Vdash \mathtt{C}'\langle o'\rangle : \tau$ and therefore $\Psi = \Gamma \Vdash \lambda x.\mathtt{C}'\langle o'\rangle : \tau$. Moreover, $\mathsf{M}(\Phi, m) = \mathsf{M}(\Phi', m) + (1, m, 0) =_{i.h.} \mathsf{M}(\Psi', m) + (1, m, 0) = \mathsf{M}(\Psi, m)$.

Case $\mathtt{C} = \mathtt{C}'u$. We have $\Phi' = \Gamma' \Vdash \mathtt{C}'\langle o\rangle : \mathscr{N} \longrightarrow \sigma$ and $\Phi_u = \Delta \Vdash u : \mathscr{N}$. By the *i.h.* there is $\Psi' = \Gamma' \Vdash \mathtt{C}'\langle o'\rangle : \mathscr{N} \longrightarrow \sigma$, so $\Psi = \Gamma' \uplus \Delta \Vdash \mathtt{C}'\langle o'\rangle u : \sigma$. Moreover, $\mathsf{M}(\Phi, m) = \mathsf{M}(\Phi', m) + \mathsf{M}(\Phi_u, m) + (1, m, 0) =_{i.h.} \mathsf{M}(\Psi', m) + \mathsf{M}(\Phi_u, m) + (1, m, 0) = \mathsf{M}(\Psi, m)$.

Case $\mathtt{C} = u\mathtt{C}'$. The case is similar to the previous one.

Case $\mathtt{C} = \mathtt{C}'[x \triangleleft u]$. We have $\Phi' = \Gamma'; x : \mathscr{M} \Vdash \mathtt{C}'\langle o\rangle : \sigma$ and $\Phi_u = \Delta \Vdash u : \mathscr{M}$. By the *i.h.* there is $\Psi' = \Gamma'; x : \mathscr{M} \Vdash \mathtt{C}'\langle o'\rangle : \sigma$, so $\Psi = \Gamma' \uplus \Delta \Vdash \mathtt{C}'\langle o'\rangle[x \triangleleft u] : \sigma$. Moreover, $\mathsf{M}(\Phi, m) = \mathsf{M}(\Phi', m) + \mathsf{M}(\Phi_u, m + \mathrm{lv}_x(t) + \mathrm{es}([x \triangleleft u])) =_{i.h.} \mathsf{M}(\Psi', m) + \mathsf{M}(\Phi_u, m + \mathrm{lv}_x(t) + \mathrm{es}([x \triangleleft u])) = \mathsf{M}(\Psi, m)$.

Case $\mathtt{C} = u[x \triangleleft \mathtt{C}']$. We have $\Phi_u = \Delta; x : \mathscr{M} \Vdash u : \sigma$ and $\Phi' = \Gamma' \Vdash \mathtt{C}'\langle o\rangle : \mathscr{M}$. By the *i.h.* there is $\Psi' = \Gamma' \Vdash \mathtt{C}'\langle o'\rangle : \mathscr{M}$, so $\Psi = \Gamma' \uplus \Delta \Vdash u[x \triangleleft \mathtt{C}'\langle o'\rangle] : \sigma$. Moreover, $\mathsf{M}(\Phi, m) = \mathsf{M}(\Phi_u, m) + \mathsf{M}(\Phi', m + \mathrm{lv}_x(u) + \mathrm{es}([x \triangleleft u])) =_{i.h.} \mathsf{M}(\Phi_u, m) + \mathsf{M}(\Psi', m + \mathrm{lv}_x(u) + \mathrm{es}([x \triangleleft u])) = \mathsf{M}(\Psi, m)$. □

Lemma 2.48 (Weighted subject reduction for $\longrightarrow_{\mathrm{sub}}$). *Let* $\Phi_{t_0} = \Gamma \Vdash t_0 : \sigma$. *If* $t_0 \longrightarrow_{\mathrm{sub}} t_1$, *then there exists* $\Phi_{t_1} = \Gamma \Vdash t_1 : \sigma$ *such that* $\mathsf{M}\left(\Phi_{t_0}, m\right) \geq \mathsf{M}\left(\Phi_{t_1}, m\right)$ *for every* $m \in \mathbb{N}$.

Proof. As remarked in section 2.1.2, $t_0 \longrightarrow_{\mathrm{sub}} t_1$ implies $t_0 \longrightarrow_\rho^* t' \longrightarrow_{\mathrm{sub}'} t_1$. By lemma 2.47, weighted subject reduction holds for $t_0 \longrightarrow_\rho^* t'$, so it is sufficient to show the statement for the relation $\longrightarrow_{\mathrm{sub}'}$. We reason by induction on this relation. We show the base cases for $\mapsto_{\mathrm{app}'}$ and $\mapsto_{\mathrm{dist}'}$, the cases $\mapsto_{\mathrm{abs}'}$ and $\mapsto_{\mathrm{var}'}$ are simply by the substitution corollary 2.46, and the inductive cases are straightforward by the *i.h.*

Case $t_0 = t[x/us] \longrightarrow_{\text{app}} t\{x/yz\}[y/u][z/s] = t_1$, where y and z are fresh variables. Let the following derivations:

$$\Phi_i = \frac{\Phi^i_u = \Gamma^i_u \Vdash u : \mathscr{N}_i \longrightarrow \rho_i \qquad \Phi^i_s = \Gamma^i_s \Vdash s : \mathscr{N}_i}{\Gamma^i_u \uplus \Gamma^i_s \vdash us : \rho_i}\ (\textsc{app})$$

then the typing derivation Φ_{t_0} is of the form

$$\frac{\Phi_t = \Gamma'; x : \mathscr{M} \Vdash t : \sigma \qquad \dfrac{\left(\Phi_i = \Gamma^i_u \uplus \Gamma^i_s \Vdash us : \rho_i\right)_{i\in I}}{\Gamma_u \uplus \Gamma_s \vdash us : \mathscr{M}}\ (\textsc{many})}{\Gamma' \uplus \Gamma_u \uplus \Gamma_s \vdash t[x/us] : \sigma}\ (\textsc{cut})$$

where $\mathscr{M} = [\rho_i]_{i\in I}$, $\Gamma_u = \uplus_{i\in I}\Gamma^i_u$ and $\Gamma_s = \uplus_{i\in I}\Gamma^i_s$. We have

$$\begin{aligned}\mathrm{M}\left(\Phi_{t_0}, m\right) = \mathrm{M}\left(\Phi_t, m\right) + \sum_{i\in I}\left(\mathrm{M}\left(\Phi^i_u, m + \mathrm{lv}_x(t) + 1\right) + \mathrm{M}\left(\Phi^i_s, m + \mathrm{lv}_x(t) + 1\right)\right)\\ + |I| \times (1, m + \mathrm{lv}_x(t) + 1, 0)\end{aligned}$$

Let us consider $\Phi' = \Gamma'; y : \mathscr{N}_u; z : \mathscr{N}_s \Vdash t\{x/yz\} : \sigma$, obtained by corollary 2.46 from Φ_t and $\Phi_{yz} = y : \mathscr{N}_u; z : \mathscr{N}_s \Vdash yz : [\rho_i]_{i\in I}$, where $\mathscr{N}_u = [\mathscr{N}_i \longrightarrow \rho_i]_{i\in I}$ and $\mathscr{N}_s = \sqcup_{i\in I}\mathscr{N}_i$. Let

$$\Phi_u = \frac{\left(\Phi^i_u = \Gamma^i_u \Vdash u : \mathscr{N}_i \longrightarrow \rho_i\right)_{i\in I}}{\Gamma_u \vdash u : \mathscr{N}_u}\ (\textsc{cut}) \qquad \Phi_s = \frac{\left(\Phi^i_s = \Gamma^i_s \Vdash s : \mathscr{N}_i\right)_{i\in I}}{\Gamma_s \vdash s : \mathscr{N}_s}\ (\textsc{cut})$$

We construct the following derivation Φ_{t_1} with two applications of rule (CUT).

$$\frac{\dfrac{\Phi' = \Gamma'; z : \mathscr{N}_s; y : \mathscr{N}_u \Vdash t\{x/yz\} : \sigma \qquad \Phi_u = \Gamma_u \Vdash u : \mathscr{N}_u}{(\Gamma'; z : \mathscr{N}_s) \uplus \Gamma_u \vdash t\{x/yz\}[y/u] : \sigma} \qquad \Phi_s = \Gamma_s \Vdash s : \mathscr{N}_s}{\Gamma' \uplus \Gamma_u \uplus \Gamma_s \vdash t\{x/yz\}[y/u][z/s] : \sigma}$$

We consider two cases to conclude:

Subcase $\mathscr{M} = [\,]$. Then

$$\begin{aligned}\mathrm{M}\left(\Phi_{t_1}, m\right) &= \mathrm{M}\left(\Phi', m\right) + \sum_{i\in I}\mathrm{M}\left(\Phi^i_u, m + \mathrm{lv}_y(t\{x/yz\}) + 1\right)\\ &\quad + \sum_{i\in I}\mathrm{M}\left(\Phi^i_s, m + \mathrm{lv}_z(t\{x/yz\}[y/u]) + 1\right)\\ &= \mathrm{M}\left(\Phi', m\right) =_{2.46} \mathrm{M}\left(\Phi_t, m\right) = \mathrm{M}\left(\Phi_{t_0}, m\right)\end{aligned}$$

Subcase $\mathscr{M} \neq [\,]$. Then

$$\begin{aligned}
\mathrm{M}\left(\Phi_{t_1}, m\right) &= \mathrm{M}\left(\Phi', m\right) + \sum_{i\in I} \mathrm{M}\left(\Phi_u^i, m + \mathrm{lv}_y(t\{x/yz\}) + 1\right) \\
&\quad + \sum_{i\in I} \mathrm{M}\left(\Phi_s^i, m + \mathrm{lv}_z(t\{x/yz\}[y/u]) + 1\right) \\
&= \mathrm{M}\left(\Phi', m\right) \\
&\quad + \sum_{i\in I} \mathrm{M}\left(\Phi_u^i, m + \mathrm{lv}_y(t\{x/yz\}) + 1\right) + \sum_{i\in I} \mathrm{M}\left(\Phi_s^i, m + \mathrm{lv}_z(t\{x/yz\}) + 1\right) \\
&=_{2.5(\mathrm{ii})} \mathrm{M}\left(\Phi', m\right) + \sum_{i\in I} \left(\mathrm{M}\left(\Phi_u^i, m + \mathrm{lv}_x(t) + 1\right) + \mathrm{M}\left(\Phi_s^i, m + \mathrm{lv}_x(t) + 1\right)\right) \\
&\leq_{2.45} \mathrm{M}\left(\Phi_t, m\right) + \mathrm{M}\left(\Phi_{yz}, m + \mathrm{lv}_x(t)\right) \\
&\quad + \sum_{i\in I} \left(\mathrm{M}\left(\Phi_u^i, m + \mathrm{lv}_x(t) + 1\right) + \mathrm{M}\left(\Phi_s^i, m + \mathrm{lv}_x(t) + 1\right)\right) \\
&= \mathrm{M}\left(\Phi_t, m\right) + (1, m + \mathrm{lv}_x(t), 2) \\
&\quad + \sum_{i\in I} \left(\mathrm{M}\left(\Phi_u^i, m + \mathrm{lv}_x(t) + 1\right) + \mathrm{M}\left(\Phi_s^i, m + \mathrm{lv}_x(t) + 1\right)\right) \\
&< \mathrm{M}\left(\Phi_t, m\right) + \sum_{i\in I} \left(\mathrm{M}\left(\Phi_u^i, m + \mathrm{lv}_x(t) + 1\right) + \mathrm{M}\left(\Phi_s^i, m + \mathrm{lv}_x(t) + 1\right)\right) \\
&\quad + |I| \times (1, m + \mathrm{lv}_x(t), 2) \\
&< \mathrm{M}\left(\Phi_{t_0}, m\right).
\end{aligned}$$

Case $t_0 = t[x/\lambda y.u] \longrightarrow_{\mathrm{dist}} t[x/\!/\lambda y.z[z/u]] = t_1$. The typing derivation is of the form

$$\Phi_{t_0} = \frac{\Phi_t = \Gamma'; x : \mathscr{M} \Vdash t : \sigma \qquad \Phi_{\lambda y.u} = \Gamma_{\lambda y.u} \Vdash \lambda y.u : \mathscr{M}}{\Gamma' \uplus \Gamma_{\lambda y.u} \vdash t[x/\lambda y.u] : \sigma}\ (\textsc{cut})$$

where

$$\Phi_{\lambda y.u} = \frac{\left(\dfrac{\Phi^i = \Gamma^i_{\lambda y.u}; y : \mathscr{N}_i \Vdash u : \rho_i}{\Gamma^i_{\lambda y.u} \vdash \lambda y.u : \mathscr{N}_i \longrightarrow \rho_i}\ (\textsc{abs})\right)_{i\in I} \left(\dfrac{}{\vdash \lambda y.u : \mathtt{a}}\ (\textsc{ans})\right)^k}{\Gamma_{\lambda y.u} \vdash \lambda y.u : \mathscr{M}}\ (\textsc{many})$$

and $\mathscr{M} = [\mathscr{N}_i \longrightarrow \rho_i]_{i\in I} \sqcup [\underbrace{\mathtt{a}, \ldots, \mathtt{a}}_{k}]$, with $\Gamma_{\lambda y.u} = \uplus_{i\in I} \Gamma^i_{\lambda y.u}$. Moreover,

$$\begin{aligned}
\mathrm{M}\left(\Phi_{t_0}, m\right) &= \mathrm{M}\left(\Phi_t, m\right) + k * (1, m + \mathrm{lv}_x(t) + 1, 0) \\
&\quad + \sum_{i\in I} \left(\mathrm{M}\left(\Phi^i, m + \mathrm{lv}_x(t) + 1\right) + (1, m + \mathrm{lv}_x(t) + 1, 0)\right)
\end{aligned}$$

We construct the following derivation

$$\Phi_{t_1} = \dfrac{\Phi_t = \Gamma'; x : \mathscr{M} \Vdash t : \sigma \qquad \Phi_\lambda = \Gamma_{\lambda y.u} \Vdash \lambda y.z[z/u] : \mathscr{M}}{\Gamma' \uplus \Gamma_{\lambda y.u} \vdash t[x /\!\!/ \lambda y.z[z/u]] : \sigma}\ (\textsc{cut})$$

where

$$\Phi_\lambda = \dfrac{\left(\dfrac{\Phi^i_\lambda = \Gamma^i_{\lambda y.u}; y : \mathscr{N}_i \Vdash z[z/u] : \rho_i}{\Gamma^i_{\lambda y.u} \vdash \lambda y.z[z/u] : \mathscr{N}_i \to \rho_i}\ (\textsc{abs})\right)_{i \in I} \left(\dfrac{}{\vdash \lambda y.z[z/u] : \mathtt{a}}\ (\textsc{ans})\right)^k}{\Gamma_{\lambda y.u} \vdash \lambda y.z[z/u] : \mathscr{M}}\ (\textsc{many})$$

with Φ^i_λ of the form

$$\dfrac{\dfrac{}{z : [\rho_i] \vdash z : \rho_i}\ (\textsc{ax}) \qquad \Phi^i = \Gamma^i_{\lambda y.u}; y : \mathscr{N}_i \Vdash u : \rho_i}{\Gamma^i_{\lambda y.u}; y : \mathscr{N}_i \vdash z[z/u] : \rho_i}\ (\textsc{cut})$$

We have

$$\begin{aligned} \mathsf{M}\left(\Phi_{t_1}, m\right) &= \mathsf{M}\left(\Phi_t, m\right) + k * (1, m + \mathrm{lv}_x(t), 0) \\ &\quad + \sum_{i \in I} \left(\mathsf{M}\left(\Phi^i, m + \mathrm{lv}_x(t) + 1\right) + (0, 0, 1) + (1, m + \mathrm{lv}_x(t), 0)\right) \\ &\leq \mathsf{M}\left(\Phi_{t_0}, m\right) \end{aligned}$$

□

Lemma 2.49 (Weighted subject reduction for $\to_{\mathrm{ndB}}$). *Let $\Phi_{t_0} = \Gamma \Vdash t_0 : \sigma$. If $t_0 \to_{\mathrm{ndB}} t_1$, then there exists $\Phi_{t_1} = \Gamma \Vdash t_1 : \sigma$ such that $\mathsf{M}\left(\Phi_{t_0}, m\right) > \mathsf{M}\left(\Phi_{t_1}, m\right)$ for every $m \in \mathbb{N}$.*

Proof. We prove that $\mathsf{M}\left(\Phi_{t_0}, m\right) > \mathsf{M}\left(\Phi_{t_1}, m\right)$ by showing in particular that it is the first component of the 3-tuple that strictly decreases.

We reason by induction on the reduction relation $\to_{\mathrm{ndB}}$.

Case $t_0 = \mathtt{L}\langle \lambda x.t \rangle u \to_{\mathrm{dB}} \mathtt{L}\langle t[x/u] \rangle = t_1$. We reason by induction on $\mathtt{L}$. The inductive step follows from lemma 2.47, so we only show the base case $\mathtt{L} = \Diamond$. The typing derivation Φ_{t_0} is of the form

$$\dfrac{\dfrac{\Phi_t = \Gamma'; x : \mathscr{M} \Vdash t : \sigma}{\Gamma' \vdash \lambda x.t : \mathscr{M} \to \sigma}\ (\textsc{abs}) \qquad \Phi_u = \Gamma_u \Vdash u : \mathscr{M}}{\Gamma' \uplus \Gamma_u \vdash (\lambda x.t)u : \sigma}\ (\textsc{app})$$

and $\mathsf{M}\left(\Phi_{t_0}, m\right) = \mathsf{M}\left(\Phi_t, m\right) + \mathsf{M}\left(\Phi_u, m\right) + (2, 2 * m, 0)$.

We construct the following derivation.

$$\Phi_{t_1} = \dfrac{\Phi_t = \Gamma'; x : \mathscr{M} \Vdash t : \sigma \qquad \Phi_u = \Gamma_u \Vdash u : \mathscr{M}}{\Gamma' \uplus \Gamma_u \vdash t[x/u] : \sigma}\ (\textsc{cut})$$

We have

$$\begin{aligned} \mathsf{M}\left(\Phi_{t_1}, m\right) &= \mathsf{M}\left(\Phi_t, m\right) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_x(t) + 1\right) \\ &=_{2.43} \mathsf{M}\left(\Phi_t, m\right) + \mathsf{M}\left(\Phi_u, m\right) + \left(0, \left(\mathrm{lv}_x(t) + 1\right) * \mathrm{sz}(\Phi_u), 0\right) \\ &< \mathsf{M}\left(\Phi_{t_0}, m\right) \end{aligned}$$

Notice that it is the first component of the first 3-tuple that strictly decreases by 2.

Case $t_0 = tu \longrightarrow_{\mathrm{ndB}} t'u = t_1$, where $t \longrightarrow_{\mathrm{ndB}} t'$. Then the property trivially holds by the *i.h.*

Case $t_0 = t[x/u] \longrightarrow_{\mathrm{ndB}} t'[x/u] = t_1$, where $t \longrightarrow_{\mathrm{ndB}} t'$. Then $\Gamma = \Gamma' \setminus x \uplus \Delta$ and $\Phi_t = \Gamma'; x : \mathscr{M} \Vdash t : \sigma$ and $\Phi_u = \Delta \Vdash u : \mathscr{M}$. Also, $\mathsf{M}\left(\Phi_{t_0}, m\right) = \mathsf{M}\left(\Phi_t, m\right) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_x(t) + 1\right)$. By the *i.h.* we have $\Phi_{t'} = \Gamma'; x : \mathscr{M} \Vdash t' : \sigma$ and $\mathsf{M}\left(\Phi_t, m\right) >_{i.h.} \mathsf{M}\left(\Phi_{t'}, m\right)$, where in particular it is the first component of the first 3-tuple that strictly decreases. Derivation Φ_{t_1} is then obtained by rule ($\textsc{cut}$) from $\Phi_{t'}$ and Φ_u. We can conclude since:

$$\begin{aligned} \mathsf{M}\left(\Phi_{t_1}, m\right) &= \mathsf{M}\left(\Phi_{t'}, m\right) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_x(t') + 1\right) \\ &=_{2.43} \mathsf{M}\left(\Phi_{t'}, m\right) + \mathsf{M}\left(\Phi_u, m\right) + \left(0, \left(\mathrm{lv}_x(t') + 1\right) * \mathrm{sz}(\Phi_u), 0\right) \\ &<_{i.h.} \mathsf{M}\left(\Phi_t, m\right) + \mathsf{M}\left(\Phi_u, m\right) + \left(0, \left(\mathrm{lv}_x(t) + 1\right) * \mathrm{sz}(\Phi_u), 0\right) \\ &=_{2.43} \mathsf{M}\left(\Phi_t, m\right) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_x(t) + 1\right) \\ &= \mathsf{M}\left(\Phi_{t_0}, m\right) \end{aligned}$$

Note that even when $\mathrm{lv}_x(t') > \mathrm{lv}_x(t)$, the inequality $\mathsf{M}\left(\Phi_{t_1}, m\right) < \mathsf{M}\left(\Phi_{t_0}, m\right)$ is determined by the strict relation between the first components of the 3-tuples, that is, the unweighted number of abstraction and application rules. □

Lemma 2.50. *Let* $\Phi = \Gamma \Vdash \mathtt{N}\langle\langle x \rangle\rangle : \tau$. *Then there exists* Γ', $I \neq \emptyset$ *and* $[\sigma_i]_{i \in I}$ *such that* $\Gamma = \Gamma' \uplus x : [\sigma_i]_{i \in I}$ *and for any variable* z *there is a proof* $\Phi = \Gamma' \uplus z : [\sigma_i]_{i \in I} \Vdash \mathtt{N}\langle\langle z \rangle\rangle : \tau$. *In particular, if* z *is fresh, then* $\Gamma' \uplus z : [\sigma_i]_{i \in I} = \Gamma'; z : [\sigma_i]_{i \in I}$.

Proof. By induction on $\mathtt{N}$.

Case $\mathtt{N} = \Diamond$. This is straightforward by taking $\Gamma' = \emptyset$ and $[\sigma_i]_{i \in I} = [\tau]$.

Cases $\mathtt{N} = \mathtt{N}'t$ and $\mathtt{N} = \mathtt{N}'[x \triangleleft t]$. There is a derivation $\Phi' = \Gamma_1 \Vdash \mathtt{N}'\langle\langle x \rangle\rangle : \tau'$, such that $\Gamma = \Gamma_1 \uplus \Gamma_2$ and $\tau' = \mathscr{M} \longrightarrow \tau$ or $\tau = \tau'$, respectively. By the *i.h.* $\Gamma_1 = \Gamma_1' \uplus x : [\sigma_i]_{i \in I}$, so that $\Gamma' = \Gamma_1' \uplus \Gamma_2$.

Case $\mathtt{N} = \mathtt{N}_1\langle\!\langle y \rangle\!\rangle[y/\mathtt{N}_2]$. The derivation is as follows.

$$\dfrac{\Gamma_1; y : [\rho_j]_{j\in J} \vdash \mathtt{N}_1\langle\!\langle y \rangle\!\rangle : \tau \qquad \dfrac{\left(\Gamma_j \vdash \mathtt{N}_2\langle\!\langle x \rangle\!\rangle : \rho_j\right)_{j\in J}}{\uplus_{j\in J}\Gamma_j \vdash \mathtt{N}_2\langle\!\langle x \rangle\!\rangle : [\rho_j]_{j\in J}}\,(\textsc{many})}{\Gamma \vdash \mathtt{N}_1\langle\!\langle y \rangle\!\rangle[y/\mathtt{N}_2\langle\!\langle x \rangle\!\rangle] : \tau}\,(\textsc{cut})$$

Where $\Gamma = \Gamma_1 \uplus \Gamma_2$ and $\Gamma_2 = \uplus_{j\in J}\Gamma_j$. By the *i.h.* on $\mathtt{N}_1$, $\Gamma_1; y : [\rho_j]_{j\in J} = \Gamma' \uplus y : [\rho_j]_{j\in J'}$ for some $\emptyset \neq J' \subseteq J$. Thus $J \neq \emptyset$. By the *i.h.* on $\mathtt{N}_2$, for every $j \in J$ we have $\Gamma_j = \Gamma'_j \uplus x : [\sigma_i]_{i\in I_j}$, where $I_j \neq \emptyset$ and a proof $\Phi_j = \Gamma'_j \uplus z : [\sigma_i]_{i\in I_j} \Vdash \mathtt{N}_2\langle\!\langle z \rangle\!\rangle : \rho_j$ for a variable z. We then take $I = \cup_{j\in J} I_j$ and $\Gamma' = \Gamma_1 \uplus_{j\in J} \Gamma'_j$. □

Lemma 2.51 (Weighted subject reduction for flneed). *Let* $\Phi_{t_0} = \Gamma \Vdash t_0 : \sigma$. *If* $t_0 \longrightarrow_{\text{flneed}} t_1$, *then there exists* $\Phi_{t_1} = \Gamma \Vdash t_1 : \sigma$ *such that* $\mathtt{M}\left(\Phi_{t_0}, m\right) > \mathtt{M}\left(\Phi_{t_1}, m\right)$ *for every* $m \in \mathbb{N}$.

Proof. We prove that $\mathtt{M}\left(\Phi_{t_0}, m\right) > \mathtt{M}\left(\Phi_{t_1}, m\right)$ by showing in particular that the first component of the first 3-tuple strictly decreases when the reduction is dB. We reason by induction on the reduction relation, *i.e.* by induction on the context $\mathtt{N}$ where the root reduction takes place. We first detail the base case when $\mathtt{N} = \Diamond$.

Case $t_0 = \mathtt{L}\langle\lambda x.t\rangle u \longrightarrow_{\text{dB}} \mathtt{L}\langle t[x/u]\rangle = t_1$. This case is the same as for name.

Case $t_0 = \mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\lambda y.p] \longrightarrow_{\text{spl}} \mathtt{LL}\langle\mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\!/\lambda y.p']\rangle = t_1$, where $\lambda y.z[z/p] \Downarrow_{\text{st}} \lambda y.\mathtt{LL}\langle p'\rangle$. The typing derivation Φ_{t_0} is of the form

$$\dfrac{\Phi = \Gamma'; x : \mathcal{N} \Vdash \mathtt{N}\langle\!\langle x \rangle\!\rangle : \sigma \qquad \dfrac{\left(\Phi^i_{\lambda y.p} = \Delta_i \Vdash \lambda y.p : \sigma_i\right)_{i\in I}}{\Delta \vdash \lambda y.p : \mathcal{N}}\,(\textsc{many})}{\Gamma' \uplus \Delta \vdash \mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\lambda y.p] : \sigma}\,(\textsc{cut})$$

where $\mathcal{N} = [\sigma_i]_{i\in I}$, $\Delta = \uplus_{i\in I}\Delta_i$ and $\Gamma = \Gamma' \uplus \Delta$. Moreover, $I \neq \emptyset$ by lemma 2.50. For each σ_i we build the following derivations $\Phi^i_{p_0}$:

Subcase $\sigma_i = \mathscr{M}_i \longrightarrow \tau_i$. Then $\Phi^i_{p_0}$ is of the form

$$\dfrac{\dfrac{\dfrac{}{z : [\tau_i] \vdash z : \tau_i}\,(\textsc{ax}) \qquad \Phi^i_p = \Delta_i; y : \mathscr{M}_i \Vdash p : \tau_i}{\Delta_i; y : \mathscr{M}_i \vdash z[z/p] : \tau_i}\,(\textsc{cut})}{\Delta_i \vdash \lambda y.z[z/p] : \mathscr{M}_i \longrightarrow \tau_i}\,(\textsc{abs})$$

where Φ^i_p is obtained from $\Phi^i_{\lambda y.p}$ by reversing the (ABS) rule.

Subcase $\sigma_i = \mathtt{a}$. Then $\Phi^i_{p_0} = \dfrac{}{\vdash \lambda y.z[z/p] : \mathtt{a}}\,(\textsc{ans})$.

By hypothesis, $\lambda y.z[z/p] \to_{\mathrm{st}}^{*} \lambda y.\mathtt{LL}\langle p'\rangle$. Since $\to_{\mathrm{st}}$ is included in $\to_{\mathrm{sub}}$, then we know by lemma 2.48 that there are derivations $\Phi^i_{p_1} = \Delta_i \Vdash \lambda y.\mathtt{LL}\langle p'\rangle : \sigma_i$ such that $\mathrm{M}\left(\Phi^i_{p_0}, m\right) \geq \mathrm{M}\left(\Phi^i_{p_1}, m\right)$. Thus, we can build the following derivation.

$$\Phi'_{t_1} = \dfrac{\Phi = \Gamma'; x : \mathscr{N} \Vdash \mathtt{N}\langle\!\langle x\rangle\!\rangle : \sigma \qquad \dfrac{(\Phi^i_{p_1} = \Delta_i \Vdash \lambda y.\mathtt{LL}\langle p'\rangle : \sigma_i)_{i\in I}}{\Delta \vdash \lambda y.\mathtt{LL}\langle p'\rangle : \mathscr{N}}\,(\textsc{many})}{\Gamma' \uplus \Delta \vdash \mathtt{N}\langle\!\langle x\rangle\!\rangle[x/\!/\lambda y.\mathtt{LL}\langle p'\rangle] : \sigma}\,(\textsc{cut})$$

Let $n = \mathrm{lv}_x(\mathtt{N}\langle\!\langle x\rangle\!\rangle)$. We begin showing that $\mathrm{M}\left(\Phi^i_{\lambda y.p}, m+n+1\right) > \mathrm{M}\left(\Phi^i_{p_0}, m+n\right)$ for every $i \in I$. There are two cases.

Subcase $\sigma_i = \mathtt{a}$. Then $\mathrm{M}\left(\Phi^i_{\lambda y.p}, m+n+1\right) = (1, m+n+1, 0)$, while $\mathrm{M}\left(\Phi^i_{p_0}, m+n\right) = (1, m+n, 0)$.

Subcase $\sigma_i = \mathscr{M}_i \to \tau_i$. Then $\mathrm{M}\left(\Phi^i_{\lambda y.p}, m+n+1\right) = (1, m+n+1, 0) + \mathrm{M}\left(\Phi^i_p, m+n+1\right)$ and

$$\begin{aligned}\mathrm{M}\left(\Phi^i_{p_0}, m+n\right) &= (1, m+n, 0) + (0,0,1) + \mathrm{M}\left(\Phi^i_p, m+n+\mathrm{lv}_z(z)+1\right)\\ &= (1, m+n, 1) + \mathrm{M}\left(\Phi^i_p, m+n+1\right).\end{aligned}$$

So that $\mathrm{M}\left(\Phi^i_{\lambda y.p}, m+n+1\right) > \mathrm{M}\left(\Phi^i_{p_0}, m+n\right)$ since $(1, m+n+1, 0) > (1, m+n, 1)$.

Finally, we have:

$$\begin{aligned}\mathrm{M}\left(\Phi'_{t_1}, m\right) &= \mathrm{M}(\Phi, m) + \sum_{i\in I} \mathrm{M}\left(\Phi^i_{p_1}, m+n\right)\\ &\leq_{2.48} \mathrm{M}(\Phi, m) + \sum_{i\in I} \mathrm{M}\left(\Phi^i_{p_0}, m+n\right)\\ &< \mathrm{M}(\Phi, m) + \sum_{i\in I} \mathrm{M}\left(\Phi^i_{\lambda y.p}, m+n+1\right)\\ &= \mathrm{M}\left(\Phi_{t_0}, m\right)\end{aligned}$$

By lemma 2.47, we can finally construct $\Phi_{t_1} = \Gamma' \uplus \Delta \Vdash \mathtt{LL}\langle \mathtt{N}\langle\!\langle x\rangle\!\rangle[x/\!/\lambda y.p']\rangle : \sigma$, where $\mathrm{M}\left(\Phi_{t_1}, m\right) = \mathrm{M}\left(\Phi'_{t_1}, m\right)$.

Case $t_0 = \mathtt{N}\langle\!\langle x\rangle\!\rangle[x/\!/v] \to_{\mathrm{sub}} \mathtt{N}\langle\!\langle v\rangle\!\rangle[x/\!/v] = t_1$. The typing derivation Φ_{t_0} is of the form

$$\dfrac{\Phi = \Gamma'; x : \mathscr{M} \Vdash \mathtt{N}\langle\!\langle x\rangle\!\rangle : \sigma \qquad \dfrac{(\Phi^i_v = \Delta_i \Vdash v : \tau_i)_{i\in I}}{\Phi_v = \Delta \Vdash v : \mathscr{M}}\,(\textsc{many})}{\Gamma' \uplus \Delta \vdash \mathtt{N}\langle\!\langle x\rangle\!\rangle[x/\!/v] : \sigma}\,(\textsc{cut})$$

where $\mathscr{M} = [\tau_i]_{i\in I}$ and $\Delta = \uplus_{i\in I}\Delta_i$. By lemma 2.50 we know that there is a non-empty $\mathscr{N} \sqsubseteq \mathscr{M}$ which types the variable x in the hole of the context $\mathtt{N}$. We can then write $\mathscr{M}$ as $\mathscr{N} \sqcup \mathscr{N}'$. By lemma 2.44 there are two derivations $\Phi_{v_1} = \Delta_1 \Vdash v : \mathscr{N}$ and $\Phi_{v_2} = \Delta_2 \Vdash v : \mathscr{N}'$ such that $\Delta = \Delta_1 \uplus \Delta_2$ and $\mathsf{M}(\Phi_v, m) = \mathsf{M}\left(\Phi_{v_1}, m\right) + \mathsf{M}\left(\Phi_{v_2}, m\right)$. Using lemma 2.45, we can construct:

$$\Phi_{t_1} = \dfrac{\Psi = \Gamma' \uplus \Delta_1; x : \mathscr{N}' \Vdash \mathtt{N}\langle\!\langle v \rangle\!\rangle : \sigma \qquad \Phi_{v_2} = \Delta_2 \Vdash v : \mathscr{N}'}{\Gamma' \uplus \Delta; x : \mathscr{N}' \vdash \mathtt{N}\langle\!\langle v \rangle\!\rangle[x/\!/v] : \sigma}\ (\textsc{cut})$$

We clearly have $\mathrm{lv}_{\Diamond}(\mathtt{N}) \leq \mathrm{lv}_x(\mathtt{N}\langle\!\langle x \rangle\!\rangle)$ and, because $x \notin \mathrm{fv}(v)$, also have $\mathrm{lv}_x(\mathtt{N}\langle\!\langle v \rangle\!\rangle) \leq \mathrm{lv}_x(\mathtt{N}\langle\!\langle x \rangle\!\rangle)$. Then,

$$\begin{aligned}
\mathsf{M}\left(\Phi_{t_1}, m\right) &= \mathsf{M}(\Psi, m) + \mathsf{M}\left(\Phi_{v_2}, m + \mathrm{lv}_x(\mathtt{N}\langle\!\langle v \rangle\!\rangle)\right) \\
&=_{2.45} \mathsf{M}(\Phi, m) + \mathsf{M}\left(\Phi_{v_1}, m + \mathrm{lv}_{\Diamond}(\mathtt{N})\right) - (0, 0, |\mathscr{N}|) + \mathsf{M}\left(\Phi_{v_2}, m + \mathrm{lv}_x(\mathtt{N}\langle\!\langle v \rangle\!\rangle)\right) \\
&\leq \mathsf{M}(\Phi, m) + \mathsf{M}\left(\Phi_{v_1}, m + \mathrm{lv}_x(\mathtt{N}\langle\!\langle x \rangle\!\rangle)\right) - (0, 0, |\mathscr{N}|) + \mathsf{M}\left(\Phi_{v_2}, m + \mathrm{lv}_x(\mathtt{N}\langle\!\langle x \rangle\!\rangle)\right) \\
&< \mathsf{M}(\Phi, m) + \mathsf{M}\left(\Phi_{v_1}, m + \mathrm{lv}_x(\mathtt{N}\langle\!\langle x \rangle\!\rangle)\right) + \mathsf{M}\left(\Phi_{v_2}, m + \mathrm{lv}_x(\mathtt{N}\langle\!\langle x \rangle\!\rangle)\right) \\
&= \mathsf{M}\left(\Phi_{t_0}, m\right)
\end{aligned}$$

Now, we analyze all the inductive cases of the form $t_0 = \mathtt{N}\langle t_0' \rangle \longrightarrow_{\text{flneed}} \mathtt{N}\langle t_1' \rangle = t_1$, where $t_0' \longrightarrow_{\text{flneed}} t_1'$.

Case $\mathtt{N} = \mathtt{N}'u$. We have $\Phi_{t_0'} = \Gamma' \Vdash \mathtt{N}'\langle t_0' \rangle : \mathscr{N} \longrightarrow \sigma$ and $\Phi_u = \Delta \Vdash u : \mathscr{N}$. By the *i.h.* there is $\Phi_{t_1'} = \Gamma' \Vdash \mathtt{N}'\langle t_1' \rangle : \mathscr{N} \longrightarrow \sigma$, so $\Phi_{t_1} = \Gamma' \uplus \Delta \Vdash \mathtt{N}'\langle t_1' \rangle u : \sigma$. Moreover, $\mathsf{M}\left(\Phi_{t_0}, m\right) = \mathsf{M}\left(\Phi_{t_0'}, m\right) + \mathsf{M}(\Phi_u, m) + (1, m, 0) >_{\textit{i.h.}} \mathsf{M}\left(\Phi_{t_1'}, m\right) + \mathsf{M}(\Phi_u, m) + (1, m, 0) = \mathsf{M}\left(\Phi_{t_1}, m\right)$.

Case $\mathtt{N} = \mathtt{N}'[x \triangleleft u]$. We have $\Phi_{t_0'} = \Gamma'; x : \mathscr{M} \Vdash \mathtt{N}'\langle t_0' \rangle : \sigma$ and $\Phi_u = \Delta \Vdash u : \mathscr{M}$. By the *i.h.* there is $\Phi_{t_1'} = \Gamma'; x : \mathscr{M} \Vdash \mathtt{N}'\langle t_1' \rangle : \sigma$, so $\Phi_{t_1} = \Gamma' \uplus \Delta \Vdash \mathtt{N}'\langle t_1' \rangle[x \triangleleft u] : \sigma$. We distinguish three different cases:

Subcase $t_0' \longrightarrow_{\text{flneed}} t_1'$ is a dB-step. We know by the *i.h.* that $\mathsf{M}\left(\Phi_{t_0'}, m\right) > \mathsf{M}\left(\Phi_{t_1'}, m\right)$ strictly decreases the first component of the first 3-tuple. We then have

$$\begin{aligned}
\mathsf{M}\left(\Phi_{t_1}, m\right) &= \mathsf{M}\left(\Phi_{t_1'}, m\right) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_x(\mathtt{N}'\langle t_1' \rangle) + 1\right) \\
&=_{2.43} \mathsf{M}\left(\Phi_{t_1'}, m\right) + \mathsf{M}(\Phi_u, m) + (0, (\mathrm{lv}_x(\mathtt{N}'\langle t_1' \rangle) + 1) * \mathrm{sz}(\Phi_u), 0) \\
&<_{\textit{i.h.}} \mathsf{M}\left(\Phi_{t_0'}, m\right) + \mathsf{M}(\Phi_u, m) + (0, (\mathrm{lv}_x(\mathtt{N}'\langle t_0' \rangle) + 1) * \mathrm{sz}(\Phi_u), 0) \\
&=_{2.43} \mathsf{M}\left(\Phi_{t_0'}, m\right) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_x(\mathtt{N}'\langle t_0' \rangle) + 1\right) \\
&= \mathsf{M}\left(\Phi_{t_0}, m\right)
\end{aligned}$$

Subcase $t_0' \longrightarrow_{\text{flneed}} t_1'$ is an spl-step. Then $t_0' \longrightarrow_{\text{sub}}^* t_1'$, so that $\mathtt{N}'\langle t_0'\rangle \longrightarrow_{\text{sub}}^* \mathtt{N}'\langle t_1'\rangle$, and thus $\mathrm{lv}_x(\mathtt{N}'\langle t_0'\rangle) \geq \mathrm{lv}_x(\mathtt{N}'\langle t_1'\rangle)$ holds by lemma 2.6. We then conclude by:

$$\begin{aligned}
\mathsf{M}\left(\Phi_{t_1}, m\right) &= \mathsf{M}\left(\Phi_{t_1'}, m\right) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_x(\mathtt{N}'\langle t_1'\rangle) + 1\right) \\
&<_{i.h.} \mathsf{M}\left(\Phi_{t_0'}, m\right) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_x(\mathtt{N}'\langle t_1'\rangle) + 1\right) \\
&\leq \mathsf{M}\left(\Phi_{t_0'}, m\right) + \mathsf{M}\left(\Phi_u, m + \mathrm{lv}_x(\mathtt{N}'\langle t_0'\rangle) + 1\right) \\
&= \mathsf{M}\left(\Phi_{t_0}, m\right)
\end{aligned}$$

Subcase $t_0' \longrightarrow_{\text{flneed}} t_1'$. is a sub-step. Then we know that $\mathtt{N}'\langle t_0'\rangle \longrightarrow_{\text{flneed}} \mathtt{N}'\langle t_1'\rangle$ also holds, then $\mathrm{lv}_x(\mathtt{N}'\langle t_0'\rangle) \geq \mathrm{lv}_x(\mathtt{N}'\langle t_1'\rangle)$. We conclude as before.

Case $\mathtt{N} = \mathtt{N}_1\langle\!\langle x \rangle\!\rangle[x/\mathtt{N}_2]$. Then we have $\Phi_1 = \Delta; x : \mathscr{M} \Vdash \mathtt{N}_1\langle\!\langle x \rangle\!\rangle : \sigma$ and $\Phi_{t_0'} = \Gamma' \Vdash \mathtt{N}_2\langle t_0'\rangle : \mathscr{M}$. By the *i.h.* there is $\Phi_{t_1'} = \Gamma' \Vdash \mathtt{N}_2\langle t_1'\rangle : \mathscr{M}$, so $\Phi_{t_1} = \Gamma' \uplus \Delta \Vdash \mathtt{N}_1\langle\!\langle x \rangle\!\rangle[x \lhd \mathtt{N}_2\langle t_1'\rangle] : \sigma$. Moreover,

$$\begin{aligned}
\mathsf{M}\left(\Phi_{t_0}, m\right) &= \mathsf{M}\left(\Phi_1, m\right) + \mathsf{M}\left(\Phi_{t_0'}, m + \mathrm{lv}_x(\mathtt{N}_1\langle\!\langle x \rangle\!\rangle) + \mathrm{es}([x \lhd \mathtt{N}_2\langle t_0'\rangle])\right) \\
&>_{i.h.} \mathsf{M}\left(\Phi_1, m\right) + \mathsf{M}\left(\Phi_{t_1'}, m + \mathrm{lv}_x(\mathtt{N}_1\langle\!\langle x \rangle\!\rangle) + \mathrm{es}([x \lhd \mathtt{N}_2\langle t_1'\rangle])\right) \\
&= \mathsf{M}\left(\Phi_{t_1}, m\right)
\end{aligned}$$

□

Example 2.52. Consider the following reduction sequence:

$$(\mathtt{I}(x_1\mathtt{I}))[x_1/\lambda y.\mathtt{I}y] \longrightarrow_{\text{dB}} x_2[x_2/x_1\mathtt{I}][x_1/\lambda y.\mathtt{I}y] \longrightarrow_{\text{spl}} x_2[x_2/x_1\mathtt{I}][x_1/\!\!/\lambda y.zy][z/\mathtt{I}]$$

We have $\Phi_1 = \emptyset \Vdash (\mathtt{I}(x_1\mathtt{I}))[x_1/\lambda y.\mathtt{I}y] : \mathtt{a}$ with Φ_1 of the form

$$\dfrac{\dfrac{\Phi_{\mathtt{I}} \qquad \Phi_{x_1\mathtt{I}}}{x_1 : [[\mathtt{a}] \longrightarrow \mathtt{a}] \vdash \mathtt{I}(x_1\mathtt{I}) : \mathtt{a}}\,(\text{APP}) \qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{y : [\mathtt{a}] \vdash y : \mathtt{a}}\,(\text{AX})}{y : [\mathtt{a}] \vdash y : [\mathtt{a}]}\,(\text{MANY}) \quad \Phi_{\mathtt{I}}}{y : [\mathtt{a}] \vdash \mathtt{I}y : \mathtt{a}}\,(\text{APP})}{\emptyset \vdash \lambda y.\mathtt{I}y : [\mathtt{a}] \longrightarrow \mathtt{a}}\,(\text{ABS})}{\emptyset \vdash \lambda y.\mathtt{I}y : [[\mathtt{a}] \longrightarrow \mathtt{a}]}\,(\text{MANY})}{\emptyset \vdash (\mathtt{I}(x_1\mathtt{I}))[x_1/\lambda y.\mathtt{I}y] : \mathtt{a}}\,(\text{CUT})$$

where

$$\Phi_{\mathtt{I}} = \dfrac{\dfrac{}{x : [\mathtt{a}] \vdash x : \mathtt{a}}\,(\text{AX})}{\emptyset \vdash \mathtt{I} : [\mathtt{a}] \longrightarrow \mathtt{a}}\,(\text{ABS})$$

and

$$\Phi_{x_1\mathtt{I}} = \dfrac{\dfrac{\dfrac{}{x_1 \vdash [[\mathtt{a}] \longrightarrow \mathtt{a}] : x_1[\mathtt{a}] \longrightarrow \mathtt{a}}\,(\text{AX}) \qquad \dfrac{\dfrac{}{\emptyset \vdash \mathtt{I} : \mathtt{a}}\,(\text{ANS})}{\emptyset \vdash \mathtt{I} : [\mathtt{a}]}\,(\text{MANY})}{x_1 : [[\mathtt{a}] \longrightarrow \mathtt{a}] \vdash x_1\mathtt{I} : \mathtt{a}}\,(\text{APP})}{x_1 : [[\mathtt{a}] \longrightarrow \mathtt{a}] \vdash x_1\mathtt{I} : [\mathtt{a}]}\,(\text{MANY})$$

We also have $\Phi_2 = \emptyset \Vdash x_2[x_2/x_1\mathtt{I}][x_1/\lambda y.\mathtt{I}y] : \mathtt{a}$ with Φ_2 of the form

$$\dfrac{\dfrac{}{x_2 : [\mathtt{a}] \vdash x_2 : \mathtt{a}}\,(\text{AX}) \qquad \Phi_{x_1\mathtt{I}}}{x_1 : [[\mathtt{a}] \to \mathtt{a}] \vdash x_2[x_2/x_1\mathtt{I}] : \mathtt{a}}\,(\text{CUT}) \qquad \dfrac{\dfrac{\Phi_{\mathtt{I}} \qquad \dfrac{\dfrac{}{y : [\mathtt{a}] \vdash y : \mathtt{a}}\,(\text{AX})}{y : [\mathtt{a}] \vdash y : [\mathtt{a}]}\,(\text{MANY})}{y : [\mathtt{a}] \vdash \mathtt{I}y : \mathtt{a}}\,(\text{APP})}{\emptyset \vdash \lambda y.\mathtt{I}y : [\mathtt{a}] \to \mathtt{a}}\,(\text{ABS})}{\emptyset \vdash \lambda y.\mathtt{I}y : [[\mathtt{a}] \to \mathtt{a}]}\,(\text{MANY})}{\emptyset \vdash x_2[x_2/x_1\mathtt{I}][x_1/\lambda y.\mathtt{I}y] : \mathtt{a}}\,(\text{CUT})$$

Concerning the measures we have $\mathrm{D}(\Phi_1) = (7, 10, 4) > (5, 13, 4) = \mathrm{D}(\Phi_2)$. The first element of the 3-tuple decreases from 7 to 5 because we lost an abstraction and an application constructors during dB-reduction. Note also that in Φ_1 we have $\mathrm{M}\left(\Phi_{x_1\mathtt{I}}, 1\right) = (2, 2, 1)$ while in Φ_2 we have $\mathrm{M}\left(\Phi_{x_1\mathtt{I}}, 2\right) = (2, 4, 1) = \mathrm{M}\left(\Phi_{x_1\mathtt{I}}, 1\right) + (0, \mathrm{sz}(\Phi_{x_1\mathtt{I}}), 0)$. Besides, we have $\Phi_3 = \emptyset \Vdash x_2[x_2/x_1\mathtt{I}][x_1/\!\!/\lambda y.zy][z/\mathtt{I}] : \mathtt{a}$ where Φ_3 is of the form

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x_2 : [\mathtt{a}] \vdash x_2 : \mathtt{a}}\,(\text{AX}) \qquad \Phi_{x_1\mathtt{I}}}{x_1 : [[\mathtt{a}] \to \mathtt{a}] \vdash x_2[x_2/x_1\mathtt{I}] : \mathtt{a}}\,(\text{CUT}) \qquad \Phi'_3}{z : [[\mathtt{a}] \to \mathtt{a}] \vdash x_2[x_2/x_1\mathtt{I}][x_1/\!\!/\lambda y.zy] : \mathtt{a}}\,(\text{CUT}) \qquad \Phi_{\mathtt{I}}}{\emptyset \vdash x_2[x_2/x_1\mathtt{I}][x_1/\!\!/\lambda y.zy][z/\mathtt{I}] : \mathtt{a}}\,(\text{CUT})$$

where Φ'_3 is

$$\dfrac{\dfrac{\dfrac{\dfrac{}{z : [[\mathtt{a}] \to \mathtt{a}] \vdash z : [\mathtt{a}] \to \mathtt{a}}\,(\text{AX}) \qquad \dfrac{\dfrac{}{y : [\mathtt{a}] \vdash y : \mathtt{a}}\,(\text{AX})}{y : [\mathtt{a}] \vdash y : [\mathtt{a}]}\,(\text{MANY})}{z : [[\mathtt{a}] \to \mathtt{a}]; y : [\mathtt{a}] \vdash zy : \mathtt{a}}\,(\text{APP})}{z : [[\mathtt{a}] \to \mathtt{a}] \vdash \lambda y.zy : [\mathtt{a}] \to \mathtt{a}}\,(\text{ABS})}{z : [[\mathtt{a}] \to \mathtt{a}] \vdash \lambda y.zy : [[\mathtt{a}] \to \mathtt{a}]}\,(\text{MANY})$$

Therefore $\mathrm{D}(\Phi_3) = (5, 11, 5) < (5, 13, 4) = \mathrm{D}(\Phi_2)$, where the second element of the 3-tuple has decreased from 13 to 11 because two nodes of the term $\lambda y.\mathtt{I}y$, namely the binder and the application, have moved from the explicit substitution of level 3 to the distributor of level 2.

Theorem 2.53 (Typability implies name-normalization). *Let $\Phi_t = \Gamma \Vdash t : \sigma$. Then t is* name*-normalizing. Moreover, the first element of $\mathrm{D}(\Phi_t)$ is an upper bound for the number of* dB*-steps to* name*-nf.*

Proof. Suppose t is not name-normalizing. Since $\to_{\mathrm{sub}}$ is terminating by corollary 2.14, then every infinite $\to_{\mathrm{name}}$-reduction sequence starting at t must necessarily have an infinite number of dB-steps. Moreover, all terms in such an infinite sequence are typed by lemma 2.49 and lemma 2.48. Therefore, these lemmas guarantee that all dB/sub reduction steps involved in such $\to_{\mathrm{name}}$-reduction sequence do not increase the measure $\mathrm{D}(\cdot)$, and

that, in particular, dB-steps strictly decrease it by decreasing the first element of the triple. This leads to a contradiction because the order $>$ on 3-tuples $\mathrm{D}(\cdot)$ is well-founded. Then t is necessarily name-normalizing. □

Theorem 2.54 (Typability implies flneed-normalization). *Let* $\Phi_t = \Gamma \Vdash t : \sigma$. *Then* t *is* flneed-*normalizing. Moreover, the first element of* $\mathrm{D}(\Phi_t)$ *is an upper bound for the number of* dB-*steps to* flneed-*nf.*

Proof. The property trivially holds by lemma 2.51 since the lexicographic order on 3-tuples is well-founded. □

Completeness. We address here completeness of system $\cap R$ with respect to $\to_{\mathrm{name}}$ and $\to_{\mathrm{flneed}}$. More precisely, we show that normalizing terms in each strategy are typable. The basic property in showing that consists in guaranteeing that normal forms are typable.

Lemma 2.55 (flneed-nfs are typable). *Let* t *be in* flneed-*nf. Then there exists a derivation* $\Phi = \Gamma \Vdash t : \tau$ *such that for any* $x \notin \mathrm{ndv}(t)$, $\Gamma(x) = [\,]$.

Proof. First, we show that if t is an answer $\mathtt{L}\langle\lambda x.p\rangle$, we can type it with type $\mathtt{a}$ and $\Gamma = \emptyset$. We reason by induction on $\mathtt{L}$. If $\mathtt{L} = \Diamond$, this is immediate. Otherwise, using the induction hypothesis, we build:

$$\dfrac{\dfrac{\emptyset \vdash \mathtt{L}\langle\lambda x.p\rangle : \mathtt{a} \qquad \dfrac{}{\emptyset \vdash u : [\,]}\,(\textsc{many})}{\emptyset \vdash \mathtt{L}\langle\lambda x.p\rangle[y \triangleleft u] : \mathtt{a}}\,(\textsc{cut})}$$

The statement is then trivial since $\Gamma = \emptyset$. For neutral terms, we use induction on $\mathrm{NE}_{\mathrm{flneed}}$ with a stronger hypothesis: there exists a derivation for any given type τ.

Case $t = x$. We can build $\Phi = x : [\tau] \Vdash x : \tau$. Note that $x \in \mathrm{ndv}(t)$.

Case $t = t'u$, where $t' \in \mathrm{NE}_{\mathrm{flneed}}$. By the *i.h.* there is a derivation $\Phi' = \Gamma \Vdash t' : [\,] \to \tau$ verifying the statement. We then build:

$$\dfrac{\Phi' = \Gamma \Vdash t' : [\,] \to \tau \qquad \dfrac{}{\emptyset \vdash u : [\,]}\,(\textsc{many})}{\Gamma \vdash t'u : \tau}\,(\textsc{app})$$

The statement holds by the *i.h.* because $\mathrm{ndv}(t) = \mathrm{ndv}(t')$.

Case $t = t'[x \triangleleft u]$, where $t' \in \mathrm{NE}_{\mathrm{flneed}}$. By the *i.h.* there is a derivation $\Phi' = \Gamma_{t'} \Vdash t' : \tau$ verifying the statement. Let $\Gamma_{t'} = \Gamma'; x : [\sigma_i]_{i \in I}$. There are two cases.

Subcase $x \notin \mathrm{ndv}(t')$. By the *i.h.* $I = [\,]$. We can then build the following derivation.

$$\dfrac{\dfrac{\Phi' = \Gamma' \Vdash t' : \tau : \qquad}{\Gamma' \vdash t'[x \triangleleft u] : \tau}\,(\textsc{cut}) \qquad \dfrac{}{\emptyset \vdash u : [\,]}\,(\textsc{many})}{}$$

The property holds for $\Gamma = \Gamma'$ because $\mathrm{ndv}(t) = \mathrm{ndv}(t')$.

Subcase $x \in \mathrm{ndv}(t')$. Then, $t = t'[x/u]$, and $u \in \mathrm{NE}_{\mathrm{flneed}}$. We apply the *i.h.* on u. There are derivations $\Phi_u^i = \Delta_i \Vdash u : \sigma_i$. We take $\Gamma = \Gamma_{t'} \uplus_{i \in I} \Delta_i$ and we build:

$$\dfrac{\Phi' = \Gamma_{t'} \Vdash t' : \tau \qquad \dfrac{(\Phi_u^i = \Delta_i \Vdash u : \sigma_i)}{\uplus_{i \in I} \Delta_i \vdash u : [\sigma_i]_{i \in I}}\,(\textsc{many})}{\Gamma \vdash t'[x/u] : \tau}\,(\textsc{cut})$$

where $\Gamma = \Gamma' \uplus_{i \in I} \Delta_i$. Moreover, $\mathrm{ndv}(t) = \mathrm{ndv}(u)$ so the second property holds on Γ by the two induction hypothesis. □

Example 2.56. Remember that $\mathrm{ndv}((xy_1)[x/z]y_1) = \{z\}$ and note that $\mathrm{ndv}(xy_1) = \{x\}$.

$$\dfrac{\dfrac{\dfrac{}{x : [\,] \to \tau \vdash x : [\,] \to \tau} \qquad \dfrac{}{\emptyset \vdash y_1 : [\,]}}{x : [\,] \to \tau \vdash xy_1 : \tau} \qquad \Phi}{z : [\,] \to [\,] \to \tau \vdash (xy_1)[x/zy_2] : \tau}$$

With

$$\Phi = \dfrac{\dfrac{}{z : [\,] \to [\,] \to \tau \vdash z : [\,] \to [\,] \to \tau} \qquad \dfrac{}{\emptyset \vdash y_2 : [\,]}}{z : [\,] \to [\,] \to \tau \vdash zy_2 : [\,] \to \tau}$$

Because name-nfs are also flneed-nfs, we infer the following corollary for free.

Corollary 2.57 (name-nfs are typable). *Let t be in* name*-nf. Then there is a derivation $\Phi = \Gamma \Vdash t : \tau$.*

We need lemmas stating the behavior of partial and full (anti-)substitution w.r.t. typing.

Lemma 2.58 (Partial anti-substitution). *Let $\mathtt{C}\langle\!\langle x \rangle\!\rangle$ and u be terms s.t. $x \notin \mathrm{fv}(u)$ and $\Phi = \Gamma \Vdash \mathtt{C}\langle\!\langle u \rangle\!\rangle : \sigma$. Then $\exists \Gamma', \exists \Delta, \exists \mathscr{M}, \exists \Phi', \exists \Phi_u$ s.t. $\Gamma = \Gamma' \uplus \Delta$, $\Phi' = \Gamma' \uplus x : \mathscr{M} \Vdash \mathtt{C}\langle\!\langle x \rangle\!\rangle : \sigma$ and $\Phi_u = \Delta \Vdash u : \mathscr{M}$.*

Proof. By induction on the structure of $\mathtt{C}$.

Case $\mathtt{C} = \Diamond$. The property trivially holds taking $\Gamma' = \emptyset$, $\Delta = \Gamma$, $\mathscr{M} = [\sigma]$, $\Phi' = x : [\sigma] \Vdash x : \sigma$ and $\Phi_u = \Phi$.

Case $\mathtt{C} = \lambda y.\mathtt{C}'$. then $y \notin \mathrm{fv}(u)$ and by α-conversion we can assume that $x \neq y$. There are two cases:

Subcase $\Phi = \dfrac{\Phi_0 = \Gamma; y : \mathscr{M}_y \Vdash \mathtt{C}'\langle\langle u\rangle\rangle : \tau}{\Gamma \vdash \lambda y.\mathtt{C}'\langle\langle u\rangle\rangle : \mathscr{M}_y \longrightarrow \tau}$. By the *i.h.* there are $\Gamma', \Delta, \mathscr{M}, \Phi'_0$ and Φ_u such that $\Gamma; y : \mathscr{M}_y = \Gamma'_0 \uplus \Delta, \Phi'_0 = \Gamma'_0 \uplus x : \mathscr{M} \Vdash \mathtt{C}'\langle\langle x\rangle\rangle : \tau$ and $\Phi_u = \Delta \Vdash u : \mathscr{M}$. By the relevance property 2.40 $y \notin \mathrm{dom}(\Delta)$ thus $\Gamma'_0 = \Gamma'; y : \mathscr{M}_y$. Therefore, $\Gamma'_0 \uplus x : \mathscr{M} = (\Gamma' \uplus x : \mathscr{M}); y : \mathscr{M}_y$ and

$$\Phi' = \frac{\Phi'_0 = (\Gamma' \uplus x : \mathscr{M}); y : \mathscr{M}_y \Vdash \mathtt{C}'\langle\langle x\rangle\rangle : \tau}{\Gamma' \uplus x : \mathscr{M} \vdash \lambda y.\mathtt{C}'\langle\langle x\rangle\rangle : \mathscr{M}_y \longrightarrow \tau}$$

Subcase $\Phi = \dfrac{}{\emptyset \vdash \lambda y.\mathtt{C}'\langle\langle u\rangle\rangle : \mathtt{a}}$. Taking $\Gamma', \Delta = \emptyset, \mathscr{M} = [\,]$ and $\Phi_u = \emptyset \Vdash u : [\,]$ we have

$$\Phi' = \frac{}{\emptyset \vdash \lambda y.\mathtt{C}'\langle\langle x\rangle\rangle : \mathtt{a}}$$

Case $\mathtt{C} = \mathtt{C}'t$. Then $\Phi = \dfrac{\Phi_1 = \Gamma_1 \Vdash \mathtt{C}'\langle\langle u\rangle\rangle : \mathscr{M}' \longrightarrow \sigma \qquad \Phi_2 = \Gamma_2 \Vdash t : \mathscr{M}'}{\Gamma_1 \uplus \Gamma_2 \vdash \mathtt{C}'\langle\langle u\rangle\rangle t : \sigma}$, where $\Gamma = \Gamma_1 \uplus \Gamma_2$. By *i.h.* there are $\Gamma'_1, \Delta, \mathscr{M}, \Phi'_1$ and Φ_u such that $\Gamma_1 = \Gamma'_1 \uplus \Delta$, $\Phi'_1 = \Gamma'_1 \uplus x : \mathscr{M} \Vdash \mathtt{C}'\langle\langle x\rangle\rangle : \mathscr{M}' \longrightarrow \sigma$ and $\Phi_u = \Delta \Vdash u : \mathscr{M}$. Therefore, taking $\Gamma' = \Gamma'_1 \uplus \Gamma_2$ we have

$$\Phi' = \frac{\Phi'_1 = \Gamma'_1 \uplus x : \mathscr{M} \Vdash \mathtt{C}'\langle\langle x\rangle\rangle : \mathscr{M}' \longrightarrow \sigma \qquad \Phi_2 = \Gamma_2 \Vdash t : \mathscr{M}'}{(\Gamma'_1 \uplus x : \mathscr{M}) \uplus \Gamma_2 \vdash \mathtt{C}'\langle\langle x\rangle\rangle t : \sigma}$$

where $(\Gamma'_1 \uplus x : \mathscr{M}) \uplus \Gamma_2 = \Gamma' \uplus x : \mathscr{M}$.

Case $\mathtt{C} = t\mathtt{C}'$. Then Φ is of the form

$$\frac{\Phi_1 = \Gamma_1 \Vdash t : [\tau_i]_{i\in I} \longrightarrow \sigma \qquad \dfrac{(\Phi_i = \Gamma_i \Vdash \mathtt{C}'\langle\langle u\rangle\rangle : \tau_i)_{i\in I}}{\Gamma_2 \vdash \mathtt{C}'\langle\langle u\rangle\rangle : [\tau_i]_{i\in I} :}}{\Gamma_1 \uplus \Gamma_2 \vdash t\mathtt{C}'\langle\langle u\rangle\rangle : \sigma}$$

where $\Gamma_2 = \uplus_{i\in I}\Gamma_i$ and $\Gamma = \Gamma_1 \uplus \Gamma_2$. There are two cases:

Subcase $I \neq \emptyset$. By *i.h.* $\exists\Gamma'_i, \exists\Delta_i, \exists\mathscr{M}_i, \exists\Phi'_i, \exists\Phi^i_u$ s.t. $\Gamma_i = \Gamma'_i \uplus \Delta_i$, $\Phi'_i = \Gamma'_i \uplus x : \mathscr{M}_i \Vdash \mathtt{C}'\langle\langle x\rangle\rangle : \tau_i$ and $\Phi^i_u = \Delta_i \Vdash u : \mathscr{M}_i$, for all $i \in I$. Let $\Delta = \uplus_{i\in I}\Delta_i$ and $\mathscr{M} = \sqcup_{i\in I}\mathscr{M}_i$ then from split lemma 2.44 we have $\Phi_u = \dfrac{\left(\Phi^i_u = \Delta_i \Vdash u : \mathscr{M}_i\right)_{i\in I}}{\Delta \vdash u : \mathscr{M}}$. Let $\Gamma'_2 = \uplus_{i\in I}\Gamma'_i$ then $\Gamma'_2 \uplus \Delta = \Gamma_2$ and Φ' is defined by

$$\frac{\Phi_1 = \Gamma_1 \Vdash t : [\tau_i]_{i\in I} \longrightarrow \sigma \qquad \dfrac{\left(\Phi'_i = \Gamma'_i \uplus x : \mathscr{M}_i \Vdash \mathtt{C}'\langle\langle x\rangle\rangle : \tau_i\right)_{i\in I}}{\Gamma'_2 \uplus x : \mathscr{M} \vdash \mathtt{C}'\langle\langle x\rangle\rangle : [\tau_i]_{i\in I}}}{(\Gamma_1 \uplus \Gamma'_2) \uplus x : \mathscr{M} \vdash t\mathtt{C}'\langle\langle x\rangle\rangle : \sigma}$$

where $\Gamma' = \Gamma_1 \uplus \Gamma_2'$.

Subcase $I = \emptyset$. Then $[\tau_i]_{i\in I} = [\,]$, $\Gamma_2 = \emptyset$ and $\Gamma = \Gamma_1$. Therefore, taking $\Gamma' = \Gamma_1$, $\Delta = \emptyset$, $\mathscr{M} = [\,]$, $\Phi_u = \emptyset \vdash u : [\,]$, we have $\Gamma_1 = \Gamma_1 \uplus x : [\,] = \Gamma' \uplus x : [\,]$ and $\Gamma' \uplus \Delta = \Gamma_1 \uplus \emptyset = \Gamma$. We take

$$\Phi' = \frac{\Phi_1 = \Gamma_1 \Vdash t : [\,] \longrightarrow \sigma \qquad \emptyset \vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : [\,]}{\Gamma_1 \vdash t\mathtt{C}'\langle\!\langle x \rangle\!\rangle : \sigma}.$$

Case $\mathtt{C} = \mathtt{C}'[y \lhd t]$. Then $\Phi = \dfrac{\Phi_1 = \Gamma_1; y : \mathscr{M}_y \Vdash \mathtt{C}'\langle\!\langle u \rangle\!\rangle : \sigma \qquad \Phi_2 = \Gamma_2 \Vdash t : \mathscr{M}_y}{\Gamma_1 \uplus \Gamma_2 \vdash \mathtt{C}'\langle\!\langle u \rangle\!\rangle[y \lhd t] : \sigma}$ where $\Gamma = \Gamma_1 \uplus \Gamma_2$. Moreover, $y \notin \mathrm{fv}(u)$ and by α-conversion we can assume that $x \neq y$. By *i.h.* there are $\Gamma_1', \Delta, \mathscr{M}, \Phi_1'$ and Φ_u such that $\Gamma_1; y : \mathscr{M}_y = \Gamma_1' \uplus \Delta$, $\Phi_1' = \Gamma_1' \uplus x : \mathscr{M} \Vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : \sigma$ and $\Phi_u = \Delta \Vdash u : \mathscr{M}$. By the relevance property 2.40 $y \notin \mathrm{dom}(\Delta)$ thus $\Gamma_1' = \Gamma''; y : \mathscr{M}_y, \Gamma_1' \uplus x : \mathscr{M} = (\Gamma'' \uplus x : \mathscr{M}); y : \mathscr{M}_y$ and $\Gamma'' \uplus \Delta = \Gamma_1$. Therefore, taking $\Gamma' = \Gamma'' \uplus \Gamma_2$ we have

$$\Phi' = \frac{\Phi_1' = (\Gamma'' \uplus x : \mathscr{M}); y : \mathscr{M}_y \Vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : \sigma \qquad \Phi_2 = \Gamma_2 \Vdash t : \mathscr{M}_y}{(\Gamma'' \uplus x : \mathscr{M}) \uplus \Gamma_2 \vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle[y \lhd t] : \sigma}$$

where $(\Gamma'' \uplus x : \mathscr{M}) \uplus \Gamma_2 = \Gamma' \uplus x : \mathscr{M}$.

Case $\mathtt{C} = t[y \lhd \mathtt{C}']$. Then Φ is of the form

$$\frac{\Phi_1 = \Gamma_1; y : [\tau_i]_{i\in I} \Vdash t : \sigma \qquad \dfrac{(\Phi_i = \Gamma_i \Vdash \mathtt{C}'\langle\!\langle u \rangle\!\rangle : \tau_i)_{i\in I}}{\Gamma_2 \vdash \mathtt{C}'\langle\!\langle u \rangle\!\rangle : [\tau_i]_{i\in I}}}{\Gamma_1 \uplus \Gamma_2 \vdash t[y \lhd \mathtt{C}'\langle\!\langle u \rangle\!\rangle] : \sigma}$$

where $\Gamma_2 = \uplus_{i\in I}\Gamma_i$ and $\Gamma = \Gamma_1 \uplus \Gamma_2$. There are two cases:

Subcase $I \neq \emptyset$. By *i.h.* there are $\Gamma_i', \Delta_i, \mathscr{M}_i, \Phi_i'$ and Φ_u^i such that $\Gamma_i = \Gamma_i' \uplus \Delta_i$, $\Phi_i' = \Gamma_i' \uplus x : \mathscr{M}_i \Vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : \tau_i$ and $\Phi_u^i = \Delta_i \Vdash u : \mathscr{M}_i$, for all $i \in I$. Let $\Delta = \uplus_{i\in I}\Delta_i$ and $\mathscr{M} = \sqcup_{i\in I}\mathscr{M}_i$ then from split lemma 2.44 we have $\Phi_u = \dfrac{\left(\Phi_u^i = \Delta_i \Vdash u : \mathscr{M}_i\right)_{i\in I}}{\Delta \vdash u : \mathscr{M}}$. Let $\Gamma_2' = \uplus_{i\in I}\Gamma_i'$ then $\Gamma_2' \uplus \Delta = \Gamma_2$ and Φ' is defined by

$$\frac{\Phi_1 = \Gamma_1; y : [\tau_i]_{i\in I} \Vdash t : \sigma \qquad \dfrac{\left(\Phi_i' = \Gamma_i' \uplus x : \mathscr{M}_i \Vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : \tau_i\right)_{i\in I}}{\Gamma_2' \uplus x : \mathscr{M} \vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : [\tau_i]_{i\in I}}}{(\Gamma_1 \uplus \Gamma_2') \uplus x : \mathscr{M} \vdash t[y \lhd \mathtt{C}'\langle\!\langle x \rangle\!\rangle] : \sigma}$$

where $\Gamma' = \Gamma_1 \uplus \Gamma_2'$.

Subcase $I = \emptyset$. Then $[\tau_i]_{i\in I} = [\,]$, $\Gamma_2 = \emptyset$ and $\Gamma = \Gamma_1$. Moreover, $y \notin \mathrm{dom}(\Gamma_1)$. Therefore, taking $\Gamma' = \Gamma_1$, $\Delta = \emptyset$, $\mathscr{M} = [\,]$, $\Phi_u = \emptyset \vdash u : [\,]$, we have $\Gamma_1 = \Gamma_1 \uplus x : [\,] = \Gamma' \uplus x : [\,]$ and $\Gamma' \uplus \Delta = \Gamma_1 \uplus \emptyset = \Gamma$. We take

$$\Phi' = \frac{\Phi_1 = \Gamma_1 \Vdash t : \sigma \qquad \emptyset \vdash \mathtt{C}'\langle\!\langle x \rangle\!\rangle : [\,]}{\Gamma_1 \vdash t[y \triangleleft \mathtt{C}'\langle\!\langle x \rangle\!\rangle] : \sigma}$$ □

Corollary 2.59 (Anti-substitution). *Let u be a term s.t. $x \notin \mathrm{fv}(u)$ and $\Phi = \Gamma \Vdash t\{x/u\} : \sigma$. Then $\exists\Gamma', \exists\Delta, \exists\mathscr{M}, \exists\Phi', \exists\Phi_u$ s.t. $\Gamma = \Gamma' \uplus \Delta$, $\Phi' = \Gamma'; x : \mathscr{M} \Vdash t : \sigma$ and $\Phi_u = \Delta \Vdash u : \mathscr{M}$.*

Proof. The proof is by induction on $|t|_x$.

Case $|t|_x = 0$. Then $t\{x/u\} = t$ and, by property 2.40, $x \notin \mathrm{dom}(\Gamma)$ then $\Gamma = \Gamma; x : [\,]$. Therefore, for $\Gamma' \coloneqq \Gamma$, $\Delta \coloneqq \emptyset$, $\mathscr{M} = [\,]$, $\Phi' \coloneqq \Phi$ and $\Phi_u \coloneqq \dfrac{}{\vdash u : [\,]}$ the result holds.

Case $|t|_x \geq 1$. Then let $\mathtt{C}\langle\!\langle x \rangle\!\rangle$ such that $t\{x/u\} = \mathtt{C}\langle\!\langle u \rangle\!\rangle$. For any fresh y, we have that $t\{x/u\} = \mathtt{C}\langle\!\langle y \rangle\!\rangle\{y/u\}$ where $\mathtt{C}\langle\!\langle y \rangle\!\rangle = t'\{x/u\}$ s.t. $t = t'\{y/x\}$. Note that $|t'|_x < |t|_x$. Then by lemma 2.58 $\exists\Gamma''$, $\exists\Delta'$, $\exists\mathscr{N}$, $\exists\Phi''$, $\exists\Phi'_u$ s.t. $\Gamma = \Gamma'' \uplus \Delta'$, $\Phi'' = \Gamma'' \uplus y : \mathscr{N} \Vdash \mathtt{C}\langle\!\langle y \rangle\!\rangle : \sigma$ and $\Phi'_u = \Delta' \Vdash u : \mathscr{N}$ where, by freshness of y, $\Gamma'' \uplus y : \mathscr{N} = \Gamma''; y : \mathscr{N}$. Therefore, by the *i.h.* on Φ'' $\exists\Gamma'''$, $\exists\Delta''$, $\exists\mathscr{N}'$, $\exists\Phi'''$, $\exists\Phi''_u$ s.t. $\Gamma''; y : \mathscr{N} = \Gamma''' \uplus \Delta''$, $\Phi''' = \Gamma'''; x : \mathscr{N}' \Vdash t' : \sigma$ and $\Phi''_u = \Delta'' \Vdash u : \mathscr{N}'$. By freshness of y and relevance, we have $y \notin \mathrm{dom}(\Delta'')$. Then $\Gamma''' = \Gamma^{iv}; y : \mathscr{N}$ where $\Gamma'' = \Gamma^{iv} \uplus \Delta''$. From Φ''' and lemma 2.45 we have $\Phi' = (\Gamma^{iv}; x : \mathscr{N}') \uplus x : \mathscr{N} \Vdash t : \sigma$ while from Φ'_u and Φ''_u we obtain $\Phi_u = \Delta' \uplus \Delta'' \Vdash u : \mathscr{N} \sqcup \mathscr{N}'$. Finally, for $\Gamma' \coloneqq \Gamma^{iv}$, $\Delta \coloneqq \Delta' \uplus \Delta''$, $\mathscr{M} = \mathscr{N} \sqcup \mathscr{N}'$ the result holds, since $(\Gamma^{iv}; x : \mathscr{N}') \uplus x : \mathscr{N} = \Gamma'; x : \mathscr{M}$ and $\Gamma' \uplus \Delta = \Gamma^{iv} \uplus \Delta'' \uplus \Delta' = \Gamma'' \uplus \Delta' = \Gamma$. □

To achieve completeness, we show that typing is preserved by anti-reduction.

Lemma 2.60 (Subject expansion). *Let $\Phi_{t_1} = \Gamma \Vdash t_1 : \sigma$ and $\mathrm{r} \in \{\rho, \mathrm{sub}, \mathrm{ndB}, \mathrm{flneed}\}$. If $t_0 \longrightarrow_{\mathrm{r}} t_1$, then there exists $\Phi_{t_0} = \Gamma \Vdash t_0 : \sigma$.*

Proof. The proof is by induction on $\longrightarrow_{\mathrm{r}}$ and uses lemma 2.58 and corollary 2.59. We detail some interesting cases of the proof. In all the cases shown, we suppose that the list context $\mathtt{L}$ of the general rule is empty ($\mathtt{L} = \Diamond$), since we can use subject expansion for $\longrightarrow_\rho$ to manipulate it.

Case $t_0 = t[x/us] \mapsto_{\mathrm{sub}} t\{x/yz\}[y/u][z/s] = t_1$. Then Φ_{t_1} is of the form

$$\dfrac{\dfrac{\Phi = \Gamma'; z : \mathscr{N}_s; y : \mathscr{N}_u \Vdash t\{x/yz\} : \sigma \qquad \Phi_u = \Delta_u \Vdash u : \mathscr{N}_u}{(\Gamma' \uplus \Delta_u); z : \mathscr{N}_s \vdash t\{x/yz\}[y/u] : \sigma} \qquad \Phi_s = \Delta_s \Vdash s : \mathscr{N}_s}{\Gamma' \uplus \Delta_u \uplus \Delta_s \vdash t\{x/yz\}[y/u][z/s] : \sigma}$$

where $\Gamma = \Gamma' \uplus \Delta_u \uplus \Delta_s$. Also $(\Gamma'; z : \mathcal{N}_s) \uplus \Delta_u = (\Gamma' \uplus \Delta_u); z : \mathcal{N}_s$ since $z \notin \mathrm{dom}(\Delta_u)$ by the relevance property 2.40. By corollary 2.59 $\exists \Gamma''$, $\exists \Delta$, $\exists \mathscr{M}$, $\exists \Phi'$, $\exists \Phi_{yz}$ s.t. $\Gamma'; z : \mathcal{N}_s; y : \mathcal{N}_u = \Gamma'' \uplus \Delta$, $\Phi' = \Gamma''; x : \mathscr{M} \Vdash t : \sigma$ and $\Phi_{yz} = \Delta \Vdash yz : \mathscr{M}$. By freshness of y, z and property 2.40 we have that $y, z \notin \mathrm{dom}(\Gamma'') \cup \{x\}$. Then $\Gamma'' = \Gamma'$ and $\Delta = z : \mathcal{N}_s; y : \mathcal{N}_u$. From Φ_{yz}, Φ_u, Φ_s and lemma 2.45 we obtain $\Phi_{us} = \Delta_u \uplus \Delta_s \Vdash us : \mathscr{M}$ and construct Φ_{t_0} as:

$$\Phi_{t_0} = \frac{\Phi' = \Gamma'; x : \mathscr{M} \Vdash t : \sigma \qquad \Phi_{us} = \Delta_u \uplus \Delta_s \Vdash us : \mathscr{M}}{\Gamma' \uplus \Delta_u \uplus \Delta_s \vdash t[x/us] : \sigma}$$

Case $t_0 = (\lambda x.t)u \longrightarrow_{\mathrm{dB}} t[x/u] = t_1$. Then Φ_{t_1} is of the form

$$\frac{\Phi_t = \Gamma'; x : \mathscr{M} \Vdash t : \sigma \qquad \Phi_u = \Gamma_u \Vdash u : \mathscr{M}}{\Gamma' \uplus \Gamma_u \vdash t[x/u] : \sigma}\ (\textsc{cut})$$

Therefore, we construct Φ_{t_0} as follows:

$$\frac{\dfrac{\Phi_t = \Gamma'; x : \mathscr{M} \Vdash t : \sigma}{\Gamma' \vdash \lambda x.t : \mathscr{M} \longrightarrow \sigma}\ (\textsc{abs}) \qquad \Phi_u = \Gamma_u \Vdash u : \mathscr{M}}{\Gamma' \uplus \Gamma_u \vdash (\lambda x.t)u : \sigma}\ (\textsc{app})$$

Case $t_0 = \mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\lambda y.p] \longrightarrow_{\mathrm{spl}} \mathtt{LL}\langle \mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\!/\lambda y.p'] \rangle = t_1$, where $\lambda y.z[z/p] \Downarrow_{\mathrm{st}} \lambda y.\mathtt{LL}\langle p' \rangle$. By subject expansion for $\longrightarrow_\rho$, there is $\Phi_{t_1'} = \Gamma \Vdash \mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\!/\lambda y.\mathtt{LL}\langle p' \rangle] : \sigma$ and it is of the form

$$\frac{\Phi = \Gamma'; x : \mathcal{N} \Vdash \mathtt{N}\langle\!\langle x \rangle\!\rangle : \sigma \qquad \dfrac{(\Phi_i = \Delta_i \Vdash \lambda y.\mathtt{LL}\langle p' \rangle : \sigma_i)_{i \in I}}{\Delta \vdash \lambda y.\mathtt{LL}\langle p' \rangle : \mathcal{N}}\ (\textsc{many})}{\Gamma' \uplus \Delta \vdash \mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\!/\lambda y.\mathtt{LL}\langle p' \rangle] : \sigma}\ (\textsc{cut})$$

where $\Delta = \uplus_{i \in I} \Delta_i$ and $\mathcal{N} = [\sigma_i]_{i \in I}$ where, by lemma 2.50, $\mathcal{N} \neq [\,]$. Then, for each $i \in I$ we have by subject expansion for $\longrightarrow_{\mathrm{sub}}$ (of which $\longrightarrow_{\mathrm{st}}$ is a subrelation) that $\Phi_i' = \Delta_i \Vdash \lambda y.z[z/p] : \sigma_i$ which has two different shapes, depending on σ_i.

Subcase $\sigma_i = \mathscr{M}_i \longrightarrow \tau_i$. Then Φ_i' is of the form

$$\frac{\dfrac{\dfrac{}{z : [\tau_i] \vdash z : \tau_i}\ (\textsc{ax}) \qquad \Phi_p^i = \Delta_i; y : \mathscr{M}_i \Vdash p : \tau_i}{\Delta_i; y : \mathscr{M}_i \vdash z[z/p] : \tau_i}\ (\textsc{cut})}{\Delta_i \vdash \lambda y.z[z/p] : \mathscr{M}_i \longrightarrow \tau_i}\ (\textsc{abs})$$

Therefore we have Ψ_i of the form

$$\frac{\Phi^i_p = \Delta_i; y : \mathscr{M}_i \Vdash p : \tau_i}{\Delta_i \vdash \lambda y.p : \mathscr{M}_i \longrightarrow \tau_i}\ (\text{ABS})$$

Subcase $\sigma_i = \mathtt{a}$. Then $\Delta_i = \emptyset$ and we obtain Ψ_i of the form $\dfrac{}{\vdash \lambda y.p : \mathtt{a}}\ (\text{ANS})$.

We can then construct Φ_{t_0} as follows

$$\frac{\Phi = \Gamma'; x : \mathscr{N} \Vdash \mathtt{N}\langle\!\langle x \rangle\!\rangle : \sigma \qquad \dfrac{(\Psi_i = \Delta_i \Vdash \lambda y.p : \sigma_i)_{i\in I}}{\Delta \vdash \lambda y.p : \mathscr{N}}\ (\text{MANY})}{\Gamma' \uplus \Delta \vdash \mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\lambda y.p] : \sigma}\ (\text{CUT})$$

Case $t_0 = \mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\!/v] \longrightarrow_{\text{sub}} \mathtt{N}\langle\!\langle v \rangle\!\rangle[x/\!/v] = t_1$. Then Φ_{t_1} is of the form

$$\frac{\Phi = \Gamma'; x : \mathscr{N}' \Vdash \mathtt{N}\langle\!\langle v \rangle\!\rangle : \sigma \qquad \Phi'_v = \Delta' \Vdash v : \mathscr{N}'}{\Gamma' \uplus \Delta' \vdash \mathtt{N}\langle\!\langle v \rangle\!\rangle[x/\!/v] : \sigma}\ (\text{CUT})$$

By lemma 2.58 $\exists\Gamma'', \exists\Delta'', \exists\mathscr{N}'', \exists\Phi', \exists\Phi''_v$ s.t. $\Gamma'; x : \mathscr{N} = \Gamma'' \uplus \Delta', \Phi' = \Gamma'' \uplus x : \mathscr{N}'' \Vdash \mathtt{N}\langle\!\langle x \rangle\!\rangle : \sigma$ and $\Phi''_v = \Delta'' \Vdash v : \mathscr{N}''$. From $x \notin \text{fv}(v)$ and the relevance property 2.40 we have that $x \notin \text{dom}(\Delta'')$. Thus $\Gamma'' = \Gamma'''; x : \mathscr{N}'$ and then $\Gamma'' \uplus x : \mathscr{N}' = \Gamma'''; x : \mathscr{N}$ where $\mathscr{N} = \mathscr{N}' \sqcup \mathscr{N}''$. From Φ'_v and Φ''_v derivations we obtain $\Phi_v = \Delta \Vdash v : \mathscr{N}$, where $\Delta = \Delta' \uplus \Delta''$. Then Φ_{t_0} is of the form

$$\frac{\Phi' = \Gamma'''; x : \mathscr{N} \Vdash \mathtt{N}\langle\!\langle x \rangle\!\rangle : \sigma \qquad \Phi_v = \Delta \Vdash v : \mathscr{N}}{\Gamma''' \uplus \Delta \vdash \mathtt{N}\langle\!\langle x \rangle\!\rangle[x/\!/v] : \sigma}\ (\text{CUT})$$

where $\Gamma''' \uplus \Delta = \Gamma' \uplus \Delta'$. □

Property 2.61. *Let $t \in \mathtt{T}_R$. If t is* name*-normalizing, then t is $\cap R$-typable.*

Proof. Let t be name-normalizing. Then $t \longrightarrow^n_{\text{name}} u$ and u is a name-nf. We reason by induction on n. If $n = 0$, then $t = u$ is typable by corollary 2.57. Otherwise, we have $t \longrightarrow_{\text{name}} t' \longrightarrow^{n-1}_{\text{name}} u$. By the *i.h.* t' is typable and thus by lemma 2.60 (because $\longrightarrow_{\text{nsub}}$ is included in $\longrightarrow_{\text{sub}}$), t turns out to be also typable. □

Property 2.62. *Let $t \in \mathtt{T}_R$. If t is* flneed*-normalizing, then t is $\cap R$-typable.*

Proof. Similar to the previous proof but using lemma 2.55 instead of corollary 2.57. □

Summing up, theorems 2.53 and 2.54 and properties 2.61 and 2.62 give:

Theorem 2.63. *$t \in \mathtt{T}_R$ is* name*-normalizing iff t is* flneed*-normalizing iff t is $\cap R$-typable.*

All the technical tools are now available to conclude observational equivalence between our two evaluation strategies based on node replication. Let $\mathscr{R}$ be any reduction notion on $\mathtt{T}_R$. Then, two terms $t, u \in \mathtt{T}_R$ are said to be **$\mathscr{R}$-observationally equivalent**, written $t \equiv u$, if for any context $\mathtt{C}$, $\mathtt{C}\langle t\rangle$ is $\mathscr{R}$-normalizing *iff* $\mathtt{C}\langle u\rangle$ is $\mathscr{R}$-normalizing.

Theorem 2.64. *For all terms $t, u \in \mathtt{T}_R$, t and u are* name*-observationally equivalent iff t and u are* flneed*-observationally equivalent.*

Proof. By theorem 2.63, $t \equiv_{\text{name}} u$ means that $\mathtt{C}\langle t\rangle$ is $\cap R$-typable *iff* $\mathtt{C}\langle u\rangle$ is $\cap R$-typable, for all $\mathtt{C}$. By the same theorem, this is also equivalent to say that $\mathtt{C}\langle t\rangle$ is flneed-normalizing *iff* $\mathtt{C}\langle u\rangle$ is flneed-normalizing for any $\mathtt{C}$, *i.e.* $t \equiv_{\text{flneed}} u$. □

2.6 Conclusion

Several calculi with ES bridge the gap between formal higher-order calculi and concrete implementations of programming languages (see a survey in [Kes07]). The first of such calculi, *e.g.* [Aba+91; BR95], were all based on *structural* substitution, in the sense that the ES operator is syntactically propagated step-by-step through the term structure until a variable is reached, when the substitution finally takes place. The correspondence between ES and linear logic proof-nets [CKP00] led to the more recent notion of calculi *at a distance* [AK10; ABM14; Acc18b], enlightening a natural and new application of the Curry-Howard interpretation. These calculi implement linear/partial substitution *at a distance*, where the search of variable occurrences is abstracted out with context-based rewriting rules, and thus no ES propagation rules are necessary. A third model was introduced by the seminal work of Gundersen, Heijltjes, and Parigot [GHP13b; GHP13a], introducing the atomic λ-calculus to implement node replication.

Inspired by the last approach we introduced the calculus λR, capturing the essence of node replication. Unlike [GHP13b], we work with an implicit (structural) mechanism of weakening and contraction, a design choice which aims at focusing and highlighting the node replication model, which is the core of our calculus, so that we obtain a rather simple and natural formalism used in particular to specify evaluation strategies. Indeed, besides the proof of the main operational meta-level properties of our calculus (confluence, termination of the substitution calculus, simulations), we use linear and non-linear versions of λR to specify evaluation strategies based on node replication, namely call-by-name and call-by-need evaluation strategies.

The first description of call-by-need was given by Wadsworth [Wad71], where reduction is performed on *graphs* instead of terms. Weak call-by-need on *terms* was then introduced by Ariola and Felleisen [AF97], and by Maraist, Odersky, and Wadler [MOW98] and Maraist, Odersky, Turner, and Wadler [Mar+99]. Reformulations were introduced by Accattoli, Barenbaum, and Mazza [ABM14] and by Chang and Felleisen [CF14]. Our call-by-need strategy is inspired by the calculus in [ABM14], which uses the distance paradigm [AK10] to gather

together meaningful and permutation rules, by clearly separating *multiplicative* from *exponential* rules, in the sense of linear logic [Gir87].

Full laziness has been formalized in different ways. Pointer graphs [Wad71; SW10] are DAGs allowing for an elegant representation of sharing. Labeled calculi [BLM05; BLM07] implement pointer graphs by adding annotations to λ-terms, which makes the syntax more difficult to handle. Lambda-lifting [Hug83; Pey87] implements full laziness by resorting to translations from λ-terms to supercombinators. In contrast to all the previous formalisms, our calculus is defined on standard λ-terms with explicit cuts, without the use of any complementary syntactical tool. So is Ariola and Felleisen's call-by-need [AF97]; however, their notion of full laziness relies on external (ad-hoc) meta-level operations used to extract the skeleton. Our specification of call-by-need enjoys fully lazy sharing, where the skeleton extraction operation is internally encoded in the term calculus operational semantics. Last but not least, our calculus has strong links with proof-theory, notably deep inference.

Balabonski [Bal12a; Bal12b] relates many formalisms of full laziness and shows that they are equivalent when considering the number of β-steps to a normal form. It would then be interesting to understand if his unified approach, (abstractly) stated by means of the theory of residuals [Lév78; Lév80], applies to our own strategy.

Balabonski shows that full laziness is optimal with respect to the confluent version of the weak λ-calculus [BLM05; BLM05]. Yet, this does not mean that full laziness is necessarily the most efficient way to implement weak evaluation. Indeed, the overhead due to the process of skeleton extraction could be more significant than the gain in the number of β-steps. To effectively compare efficiency of CbN, CbV or CbNeed to full laziness by looking at the number of steps, we would need to show that fully lazy CbNeed is *reasonable*. This mean that we should be able to give an implementation whose (time) complexity is polynomially related to Turing machines. Accattoli and Dal Lago [AD16] and Accattoli, Condoluci, and Sacerdoti Coen [ACS21] have shown that CbN and CbV are *reasonable*. They rely on a particular implementation using an explicit substitution calculus, and differentiating *useful* from *useless* substitutions. For a reasonable strategy, all steps can be considered as atomic operations, which justifies that fewer steps mean more efficiency [Acc18a].

A Curry-Howard interpretation of the logical *switch* rule of deep inference is given as an end-of-scope operator in [She19; She+20], thus introducing the *spinal atomic λ-calculus*. The calculus implements a refined optimization of call-by-need, where only the *spine* of the abstraction (tighter than the skeleton) is duplicated. It would be interesting to adapt λR to spine duplication using an appropriate end-of-scope operator, such as the one in [HvO03]. Further optimizations might also be considered. Extending full laziness to classical logic would be another interesting research direction, possibly taking preliminary ideas from He [He18].

Finally, we only consider weak evaluation strategies, *i.e.* with reductions forbidden under abstractions, but it would be interesting to extend our notions to full (strong) evaluations too [GL02; Bal+17; BLM21].

We have also studied the calculus from a semantical point of view, by means of intersection types. Indeed, the type system can be seen as a model of our implementations of call-by-name and call-by-need, in the sense that typability and normalization turn out to be equivalent.

Those characterizations provided by intersection type systems sometimes lead to observational equivalence results (*e.g.* [Kes16]). We succeed to prove observational equivalence related to a fully lazy implementation of weak call-by-need, a result which would be extremely involved to prove by syntactical tools of rewriting, as done for weak call-by-need in [AF97]. Moreover, our result implies that our node replication implementation of full laziness is observationally equivalent to standard call-by-name and to weak call-by-need (see [Kes16]), as well as to the more semantical notion of neededness (see [KRV18]).

While our type system provides upper bounds on the number of dB steps, we would also like to investigate (quantitative) *tight* types for our fully lazy strategy, as done for weak call-by-need by [AGL19]. Tight types [AGK20] provide exact bounds on the length of reduction and the size of normal forms. However, this does not seem evident in our node replication framework.

A quantitative type system formulated in open deduction has been defined by Guerrieri, Heijltjes, and Paulus [GHP21], independently of this work. This framework is parametrized by algebraic rules, allowing to encode, notably, idempotence or non-idempotence. Their type system can be interpreted both as a simple type system for a resource calculus, or as a quantitative type system for a calculus with linear substitution. However, none of these two calculi use node replication. It would be interesting to understand if a quantitative system can be derived for node replication inside the open-deduction formalism, to type the original atomic λ-calculus, or our calculus λR. An interesting exercise would be to capture our fully lazy strategy, which relies both on linear substitution and on node replication.

CHAPTER 3

Solvability for Generalized Applications

The next two chapters of this thesis are centered around λ-calculi with generalized application (often shortened to generalized applications). Chapter 3 describes a study of solvability in these calculi, while chapter 4 details our quantitative approach to CbN in generalized applications.

This chapter starts with a formal introduction of the calculi with generalized applications: the syntax and the original (section 3.1.1) and distant (section 3.1.2) semantics. The general definitions of solvability in that context are given in section 3.2.

The next two sections are built in a symmetrical way, the first one (section 3.3) dealing with CbN, and the second one (section 3.4) with CbV. We first present a *solving* reduction capturing solvability (section 3.3.1 and section 3.4.2), then a quantitative type system as a logical characterization (section 3.3.2 and section 3.4.3). Call-by-value solvability is built up on the notion of *potential valuability*, presented and operationally characterized in section 3.4.1, and for which a logical characterization is achieved with the same type system as CbV solvability.

We extend the tools and technique concerning solvability to the original calculi ΛJ and ΛJ_v in section 3.5.1: we introduce appropriate reduction relations and normal forms, and derive direct characterizations, thanks to modular proofs of the previous section. The equivalence between the distant and non-distant notions of solvability follow from the logical characterizations. Then, in section 3.5.2, we prove equivalence between the new notions of CbN and CbV solvability for generalized applications and the one for the λ-calculus, using the type systems.

In section 3.6, we compare the calculi λJ_v and $\lambda_{\mathtt{vsub}}$ on an operational level with simulations. We also introduce a CbV strong bisimulation on $\mathtt{T}_J$, that give rise to a powerful equational theory on terms.

Finally, we illustrate the versatility of the CbV calculi with generalized applications by defining a *normalizing* strategy, akin to leftmost-outermost evaluation in the λ-calculus (section 3.7). We capture the existence of a strong normal form operationally with this relation, and logically with the CbV type system and a special restriction on types.

3.1 The Calculi with Generalized Applications

The set of terms with generalized applications is called T_J, and is generated by the following grammar, given a countably infinite set $\mathscr{V}$ of variables $x, y, z \ldots$

$$\textbf{(Values)} \quad v ::= x \in \mathscr{V} \mid \lambda x.t$$
$$\textbf{(Terms)} \quad t, u, r, s ::= v \mid t(u, x.r)$$

The grammar for values is the same as in the λ-calculus. Indeed, values are interpreted from the axiom and introduction of implication rules of natural deduction with generalized elimination rules, which are the same as for usual natural deduction.

$$\frac{}{\Gamma; x : A \vdash x : A} \qquad \frac{\Gamma; x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \longrightarrow B}$$

Generalized applications $t(u, x.r)$, on the other hand, contain three subterms t, u and r, corresponding to the three premises of the implication elimination rule in von Plato's system.

$$\frac{\Gamma \vdash t : A \longrightarrow B \qquad \Gamma \vdash u : A \qquad \Gamma; x : B \vdash r : C}{\Gamma \vdash t(u, x.r) : C}$$

We call the part $x.r$ a continuation; note that it is not a subterm. The variable x binds the possible free occurrences of x in r. Indeed, a generalized application can be seen as a let-binding under the (informal) translation of $t(u, x.r)$ to $\texttt{let}\ x = tu\ \texttt{in}\ r$. The presence of sharing of all and only applications in these calculi is one of their very specific features.

Definition 3.1. A formal translation $(\cdot)^\star$ to explicit substitution can be given [Esp07]:

$$x^\star := x \qquad (\lambda x.t)^\star := \lambda x.t^\star \qquad t(u, x.r)^\star := r^\star[x/t^\star u^\star]$$

We use this translation in this chapter, but detail how it does not scale to strong normalization and propose a faithful one in chapter 4.

A translation $(\cdot)^\#$ to the λ-calculus will also be useful:

$$x^\# := x \qquad (\lambda x.t)^\# := \lambda x.t^\# \qquad t(u, x.r)^\# := (\lambda y.r^\#)(t^\# u^\#)$$

Free and **bound** variables of terms are defined as expected, in particular, $\mathrm{fv}(t(u, x.r)) := \mathrm{fv}(t) \cup \mathrm{fv}(u) \cup (\mathrm{fv}(r) \setminus \{x\})$. A generalized application $t(u, x.r)$ is said to be **non-relevant** if $x \notin \mathrm{fv}(r)$.

Definition 3.2. We also define a translation $(\cdot)^\circ$ from the λ-calculus with ES to generalized applications, which consists in giving a "dummy" continuation $z.z$. This translation extends the one from the λ-calculus, for which we use the same notation.

$$x^\circ := x \qquad (\lambda x.M)^\circ := \lambda x.M^\circ \qquad (MN)^\circ := M^\circ(N^\circ, z.z) \qquad (M[x/N])^\circ := \mathtt{I}(N^\circ, x.M^\circ)$$

We reuse the names δ and Ω for the corresponding terms $\delta := \lambda x.x(x, z.z)$ and $\Omega := \delta(\delta, z.z)$. We also define a family of projection terms $\mathtt{o}^n := \lambda x_n \ldots \lambda x_0.x_0$ parametrized by a natural number n.

Contexts $\mathtt{C}$ are extended to generalized applications:

$$\mathtt{C} ::= \Diamond \mid \lambda x.\mathtt{C} \mid \mathtt{C}(u, x.r) \mid t(\mathtt{C}, x.r) \mid t(u, x.\mathtt{C})$$

3.1.1 The Original Semantics

The CbN operational semantics relies on a notion of **right substitution**, which is the expected capture-free meta-level substitution on $\mathtt{T}_J$-terms:

$$\begin{array}{rrcl@{\qquad}rrcl} & x\{x/u\} & \coloneqq & u & & (\lambda y.t)\{x/u\} & \coloneqq & \lambda y.t\{x/u\} \\ (x \neq y) & y\{x/u\} & \coloneqq & y & & (t(s, y.r))\{x/u\} & \coloneqq & (t\{x/u\})(s\{x/u\}, y.r\{x/u\}) \end{array}$$

Computation is done with the following rule β:

$$(\lambda x.t)(u, y.r) \mapsto_\beta r\{y/t\{x/u\}\}$$

This rule is a generalization of the usual β of the λ-calculus, as depicted below.

$$\mathtt{let}\ y = (\lambda x.t)u\ \mathtt{in}\ r \longrightarrow \mathtt{let}\ y = t\{x/u\}\ \mathtt{in}\ r \longrightarrow r\{y/t\{x/u\}\}$$

This gives an intuitive explanation of this rule through the previous informal translation of generalized applications to let-bindings: $(\lambda x.t)(u, y.r)$ corresponds to $\mathtt{let}\ y = (\lambda x.t)u\ \mathtt{in}\ r$. In this first term, the computation in the foreground comes from the abstraction $\lambda x.t$ and its argument u. We get an intermediate result by substituting u for x in t, thus obtaining $\mathtt{let}\ y = t\{x/u\}\ \mathtt{in}\ r$. This intermediate result can then be fed to the continuation by unfolding the let-binding, which means substituting it for y in r, thus obtaining the contractum $r\{y/t\{x/u\}\}$. The term $t\{x/u\}$ may be duplicated, or, on the contrary, may be simply erased, as shown in the next examples.

Examples 3.3. The first example depicts erasure, and the second duplication.

- $(\lambda x.x(\mathtt{I}, y.y))(\mathtt{I}, z.z') \longrightarrow_\beta z'\{z/(x(\mathtt{I}, y.y))\{x/\mathtt{I}\}\} = z'\{z/\mathtt{I}(\mathtt{I}, y.y)\} = z'$
- $(\lambda x.x(\mathtt{I}, y.y))(\mathtt{I}, z.z(z, z'.z')) \longrightarrow_\beta (z(z, z'.z'))\{z/\mathtt{I}(\mathtt{I}, y.y)\} = \mathtt{I}(\mathtt{I}, y.y)(\mathtt{I}(\mathtt{I}, y.y), z'.z')$

As mentioned in the introduction, calculi ΛJ and ΛJ_v also use the permutation rule π:

$$t(u, x.r)(u', y.r') \mapsto_\pi t(u, x.r(u', y.r'))$$

This rule brings the leftmost application of a term to the top of its syntax tree, while stacking the list of arguments inside the continuation of the application. This particular feature brings generalized applications closer to abstract machines, and is reminiscent of continuation-passing style, as well as administrative normal forms [Fla+93], in which every intermediate computation is named (see section 3.8).

Example 3.4. Recall the example from the introduction:

$$\begin{aligned}(\lambda x.t)(u_1, y_1.y_1)(u_2, y_2.y_2)(u_3, y_3.y_3) &\longrightarrow_\pi (\lambda x.t)(u_1, y_1.y_1)(u_2, y_2.y_2(u_3, y_3.y_3)) \\ &\longrightarrow_\pi (\lambda x.t)(u_1, y_1.y_1(u_2, y_2.y_2(u_3, y_3.y_3)))\end{aligned}$$

The reduction relation $\to_{\mathrm{jn}}$ of the original CbN calculus ΛJ [JM00; JM03] is defined as the closure under all contexts $\mathtt{C}$ of the two reduction rules β and π.

A concrete advantage of using generalized applications is the simplicity of normal forms, which are usually either a value or an application in which the left element is a variable. For instance, the jn-normal forms are given by the following inductive definition.

$$\mathrm{NF}_{\mathrm{jn}} ::= x \mid \lambda x.\, \mathrm{NF}_{\mathrm{jn}} \mid x(\mathrm{NF}_{\mathrm{jn}}, y.\, \mathrm{NF}_{\mathrm{jn}})$$

In the λ-calculus instead, inductive definitions of normal forms rely on mutually recursive definitions of *neutral normal* and *normal* terms, since the element at the left of an application is not necessarily a variable.

The CbV semantics is based on a notion of **left substitution** $t\{x\backslash\backslash u\}$:

$$t\{x\backslash\backslash v\} := t\{x/v\} \qquad\qquad t\{x\backslash\backslash s(u, y.r)\} := s(u, y.t\{x\backslash\backslash r\})$$

Left substitution of a value invokes the right substitution, and left substitution of a generalized application performs a commutative/permutative conversion. This conversion prevents duplication of the potential redex between s and u. In other words, if the argument is an application, a unique copy of it is kept, which corresponds to CbV, which neither duplicates nor erases computations.

Examples 3.5. The first example demonstrates the left substitution of a value, the second of an application. In the second example, the application $\mathtt{I}(\mathtt{I}, y.\_)$ is not erased, and it is not duplicated in the third example, as that would be the case with right substitution.

- $(x(\mathtt{I}, y.y))\{x\backslash\backslash\mathtt{I}\} = (x(\mathtt{I}, y.y))\{x/\mathtt{I}\} = \mathtt{I}(\mathtt{I}, y.y)$
- $z'\{z\backslash\backslash\mathtt{I}(\mathtt{I}, y.y)\} = \mathtt{I}(\mathtt{I}, y.z'\{z\backslash\backslash y\}) = \mathtt{I}(\mathtt{I}, y.z'\{z/y\}) = \mathtt{I}(\mathtt{I}, y.z')$
- $(z(z, z'.z'))\{z\backslash\backslash\mathtt{I}(\mathtt{I}, y.y)\} = \mathtt{I}(\mathtt{I}, y.(z(z, z'.z'))\{z\backslash\backslash y\}) = \mathtt{I}(\mathtt{I}, y.y(y, z'.z'))$

The reduction relation $\to_{\mathrm{jv}}$ of the original CbV calculus ΛJ_v [Esp20] is defined as the closure under all contexts $\mathtt{C}$ of the following rule $\beta\mathrm{v}$ and of π.

$$(\lambda x.t)(u, y.r) \mapsto_{\beta\mathrm{v}} r\{y\backslash\backslash t\{x\backslash\backslash u\}\}$$

Notice the strong similarity between CbN and CbV. The only difference is the substitution used. This is unlike most CbV calculi, which put a restriction on the β rule stating that the argument must be a value. Here instead, values and applications are handled differently inside the $\beta\mathrm{v}$ rule. In generalized applications, every function application is a redex that can be fired. Some defects of call-by-value calculi, due to redexes stuck because of the condition on the argument are avoided. Moreover, CbN and CbV reductions and normal forms look very similar. We take advantage from this to highlight the significant differences between CbN and CbV, notably in the context of solvability.

Let us compare ΛJ and ΛJ_v by CbV-reducing the terms from examples 3.3 (we use the equations in examples 3.5).

Examples 3.6. In the first term, the argument is not erased until a second reduction step: CbV needs one more reduction step.

$$\begin{aligned}(\lambda x.x(\mathtt{I}, y.y))(\mathtt{I}, z.z') &\to_{\beta_\mathrm{v}} z'\{z\backslash\backslash(x(\mathtt{I}, y.y))\{x\backslash\backslash\mathtt{I}\}\} = \mathtt{I}(\mathtt{I}, y.z')\\ &\to_{\beta_\mathrm{v}} z'\{y\backslash\backslash x\{x\backslash\backslash\mathtt{I}\}\} = z'\{y\backslash\backslash\mathtt{I}\} = z'\end{aligned}$$

In the second term, the argument is not duplicated: we can reach the normal form in one less reduction step than CbN, where the reduction of the argument must be done twice.

$$\begin{aligned}(\lambda x.x(\mathtt{I}, y.y))(\mathtt{I}, z.z(z, z'.z')) &\to_{\beta} (z(z, z'.z'))\{z\backslash\backslash x(\mathtt{I}, y.y)\{x\backslash\backslash\mathtt{I}\}\} = \mathtt{I}(\mathtt{I}, y.y(y, z'.z'))\\ &\to_{\beta_\mathrm{v}} y\{y\backslash\backslash x\{x\backslash\backslash\mathtt{I}\}\}(y(y, z'.z')) = \mathtt{I}(\mathtt{I}, z'.z')\end{aligned}$$

As usual, CbV does not duplicate computations (outside abstractions), but tries to reduce every argument to a value, and this may create divergent computations. Take for instance $t = \delta(\delta, z.y)$, which translates to $t^{\star} = y[z/\delta]\delta$. In CbN, t normalizes to y, while in CbV, t loops infinitely.

$$\begin{aligned}&\text{(CBN)} \quad \delta(\delta, z.y) \mapsto_{\beta} y\{z/(x(x, z.z))\{x/\delta\}\} = y\{z/\delta(\delta, z.z)\} = y\\ &\text{(CBV)} \quad \delta(\delta, z.y) \mapsto_{\beta_\mathrm{v}} y\{z\backslash\backslash(x(x, z.z))\{x\backslash\backslash\delta\}\} = y\{z\backslash\backslash\delta(\delta, z.z)\} = \delta(\delta, z.y\{z\backslash\backslash z\}) = \delta(\delta, z.y)\end{aligned}$$

3.1.2 The Distant Semantics

In this work, we concentrate on distant variants λJ_n and λJ_v. Our approach is guided by quantitative types as a resource-aware model, which is neutral with respect to quantitatively correct permutations. The goal is to obtain a higher level of abstraction, closer to the λ-calculus and reflecting the quantitative model, through a calculus with a single computational rule.

A naive approach consisting in simply removing rule π is not satisfactory. Indeed, some expected computations can be stuck by the syntax until unblocked by a permutation.

Example 3.7. The following example does not β-reduce, because the application $z(u_1, y_1.\_)$ around $\lambda x.x$ prevents from applying β-reduction on u_2. Rule π moves this argument next to the abstraction, from where computation can continue.

$$z(u_1, y_1.\lambda x.x)(u_2, y_2.y_2) \to_{\pi} z(u_1, y_1.(\lambda x.x)(u_2, y_2.y_2)) \to_{\beta} z(u_1, y_1.u_2)$$

Our solution is to use *distance*, relying on a new notion of **distant contexts**:

$$\mathtt{D} ::= \Diamond \mid t(u, x.\mathtt{D})$$

Distant contexts are reminiscent of *list contexts* of explicit substitutions: they represent a list of shared terms, that can sometimes stand between an abstraction and its argument.

The CbN/CbV **distant calculi** λJ_n **and** λJ_v are based on the reduction relations $\to_{\mathrm{djn}}$ and $\to_{\mathrm{djv}}$ respectively, generated by the closure of the following rules $\mathrm{d}\beta$ and $\mathrm{d}\beta_\mathrm{v}$ under all contexts.

$$\begin{array}{lllll}(\mathrm{d}\beta) & \mathtt{D}\langle\lambda x.t\rangle(u, y.r) & \mapsto_{\mathrm{d}\beta} & r\{y/\mathtt{D}\langle t\{x/u\}\rangle\}\\ (\mathrm{d}\beta_\mathrm{v}) & \mathtt{D}\langle\lambda x.t\rangle(u, y.r) & \mapsto_{\mathrm{d}\beta_\mathrm{v}} & \mathtt{D}\langle r\{y\backslash\backslash t\{x\backslash\backslash u\}\}\rangle\end{array}$$

The distant CbV rule integrates π inside the rule $\mathrm{d}\beta_{\mathrm{v}}$. However, the distant CbN one integrates a different rule p2. Reasons for this are given at length in chapter 4.

$$t(u, y.\lambda x.r) \longmapsto_{\mathrm{p2}} \lambda x.t(u, y.r)$$

Remark that the distant context $\mathtt{D}$ appears in different places in the right side of the rules $\mathrm{d}\beta$ and $\mathrm{d}\beta_{\mathrm{v}}$. Indeed, the distant context $\mathtt{D}$ should not be duplicated or erased in CbV, on the contrary to CbN. However, by definition of left substitution, the equation $\mathtt{D}\langle r\{y\backslash\backslash t\{x\backslash\backslash u\}\}\rangle = r\{y\backslash\backslash \mathtt{D}\langle t\{x\backslash\backslash u\}\rangle\}$ holds. Thus, the symmetry between the CbN and CbV rules is preserved.

The notion of distant context is prevalent in our analysis of generalized applications. Notice in particular that every term can be (uniquely) decomposed into $\mathtt{D}\langle v\rangle$, where $\mathtt{D}$ is a distant context and v a value. Thus for example, let $t := x_1(u, y.x_2(y, z.z))$. Then, there are three possible decompositions of t in terms of distance contexts: $t = \mathtt{D}_0\langle x_1(u, y.x_2(y, z.z))\rangle$ with $\mathtt{D}_0 = \diamond$, $t = \mathtt{D}_1\langle x_2(y, z.z)\rangle$ with $\mathtt{D}_1 = x_1(u, y.\diamond)$ and $t = \mathtt{D}_2\langle z\rangle$ with $\mathtt{D}_2 = x_1(u, y.x_2(y, z.\diamond))$. We say in particular that a term t has an **abstraction shape** if $t = \mathtt{D}\langle\lambda x.t'\rangle$.

This decomposition of any term u into $\mathtt{D}\langle v\rangle$ enables us to give an alternative definition of left substitution: $t\{x\backslash\backslash\mathtt{D}\langle v\rangle\} = \mathtt{D}\langle t\{x/v\}\rangle$. We can see clearly how left substitution pushes the list of applications represented by the context $\mathtt{D}$ outside, before substituting only the value contained at the core of the term, thus following CbV principles.

Example 3.8. Take again $t = x_1(u, y.x_2(y, z.z))$ with the decomposition $t = \mathtt{D}_2\langle z\rangle$, where $\mathtt{D}_2 = x_1(u, y.x_2(y, z.\diamond))$.

$$w'\{w\backslash\backslash x_1(u, y.x_2(y, z.z))\} = w'\{w\backslash\backslash\mathtt{D}_2\langle z\rangle\} = \mathtt{D}_2\langle w'\{w/z\}\rangle = \mathtt{D}_2\langle w'\rangle = x_1(u, y.x_2(y, z.w'))$$

Going further, we could replace the use of left substitution in $\beta\mathrm{v}$ (and $\mathrm{d}\beta_{\mathrm{v}}$) by a finer analysis of the structure of the term. We also use the property that values are closed under substitution, and suppose that $x \notin \mathrm{fv}(r)$ by α-conversion.

$$(\lambda x.\mathtt{D}_1\langle v_1\rangle)(\mathtt{D}_2\langle v_2\rangle, y.r) \longrightarrow_{\beta\mathrm{v}'} \mathtt{D}_2\langle\mathtt{D}_1\langle r\{y/v_1\}\rangle\{x/v_2\}\rangle$$

3.2 Solvability of Generalized Applications

To begin our study of solvability, we start by giving the appropriate tools. Like in the λ-calculus, solvability is defined thanks to a specific notion of context. We will here also call them *head contexts*. There are two reasons:

1. Head contexts on terms in $\mathtt{T}_J$ are a strict generalization of head contexts in λ.

2. Their role is the same: they are used in the definition of solvability, entail the CbN solving relation and are at the core of CbV solving contexts.

In contrast to the λ-calculus, the syntax of generalized applications makes the identification of the *head* of a term very subtle. In particular, whereas it is possible to use vectorial meta-notations in the λ-calculus, we must use inductive definitions in this framework. **Head contexts** $\mathtt{H}$ are given by the following grammar:

$$\mathtt{H} ::= \Diamond \mid \lambda x.\mathtt{H} \mid \mathtt{H}(u, x.\mathtt{H}'\langle\!\langle x \rangle\!\rangle) \mid t(u, x.\mathtt{H})$$

While, in general, there are several possibilities to decompose a term into a head context surrounding a subterm, there is a closely related notion of **head variable** $\mathrm{hv}(t)$, which deterministically distinguishes a particular variable in the term t:

$$\frac{}{\mathrm{hv}(x) = x} \qquad \frac{\mathrm{hv}(t) = x}{\mathrm{hv}(\lambda y.t) = x} \qquad \frac{\mathrm{hv}(r) = y \qquad \mathrm{hv}(t) = x}{\mathrm{hv}(t(u, y.r)) = x} \qquad \frac{\mathrm{hv}(r) = x \qquad x \neq y}{\mathrm{hv}(t(u, y.r)) = x}$$

In the third rule we assume w.l.o.g. that y is not bound in r. Notice that the head variable may be either free or bound, since y can be equal to x in the second rule. To understand the last two rules, we use the previous analogy with let-bindings. To an application $t(u, y.r)$ corresponds a binding $\mathtt{let}\ y = tu\ \mathtt{in}\ r$, and to find the head variable of this term, we look inside r. For instance, the head variable of $\mathtt{let}\ x = zz\ \mathtt{in}\ y$, corresponding to $z(z, x.y)$, is y. But if we take $\mathtt{let}\ x = zz\ \mathtt{in}\ x$, corresponding to $z(z, x.x)$, its head variable is z. Thus, the head variable of a term with generalized applications is the head variable of the corresponding term where all the let-binding have been unfolded. In the example, z is the head variable of $z(z, x.x)$ because x is itself the head variable of the subterm x inside the continuation.

Lemma 3.9. *Let $t \in \mathtt{T}_J$ and $\mathrm{hv}(t) = x$. There is a unique decomposition $t = \mathtt{H}\langle x \rangle$. Moreover, if $x \in \mathrm{fv}(t)$, then $t = \mathtt{H}\langle\!\langle x \rangle\!\rangle$.*

Proof. By induction on t.

Case $t = x$. We take $\mathtt{H} = \Diamond$.

Case $t = \lambda y.u$. By the *i.h.* on u, $u = \mathtt{H}'\langle x \rangle$. We take $\mathtt{H} = \lambda y.\mathtt{H}'$ so that $t = \mathtt{H}\langle x \rangle$. If $x \in \mathrm{fv}(t)$, then $x \in \mathrm{fv}(u)$ and $x \neq y$. We then have $u = \mathtt{H}'\langle\!\langle x \rangle\!\rangle$ and thus $t = \mathtt{H}\langle\!\langle x \rangle\!\rangle$.

Case $t = s(u, y.r)$. Let $z = \mathrm{hv}(r)$. By the *i.h.* $r = \mathtt{H}_1\langle z \rangle$. There are two cases.

- **Subcase** $z = y$. Then $z \in \mathrm{fv}(r)$ and we have $r = \mathtt{H}_1\langle\!\langle z \rangle\!\rangle$. Therefore $\mathrm{hv}(t) = \mathrm{hv}(s) = x$. Thus by the *i.h.* $s = \mathtt{H}_2\langle x \rangle$. We then take $\mathtt{H} = \mathtt{H}_2(u, y.\mathtt{H}_1\langle\!\langle y \rangle\!\rangle)$ so that $t = \mathtt{H}\langle x \rangle$. If $x \in \mathrm{fv}(t)$, then $x \in \mathrm{fv}(s)$, we conclude $s = \mathtt{H}_2\langle\!\langle x \rangle\!\rangle$ and thus $t = \mathtt{H}\langle\!\langle x \rangle\!\rangle$.
- **Subcase** $z \neq y$. Then $\mathrm{hv}(t) = \mathrm{hv}(r) = z = x$. We take $\mathtt{H} = s(u, y.\mathtt{H}_1)$ so that $t = \mathtt{H}\langle x \rangle$. If $x \in \mathrm{fv}(t)$, then $x \in \mathrm{fv}(r)$, we conclude $s = \mathtt{H}_1\langle\!\langle x \rangle\!\rangle$ and thus $t = \mathtt{H}\langle\!\langle x \rangle\!\rangle$. □

Thus for example, given $t \coloneqq z(z, x.y)$, we have $\mathrm{hv}(t) = y$ and $t = \mathtt{H}\langle\!\langle y \rangle\!\rangle$ with $\mathtt{H} = z(z, x.\Diamond)$. Given $t' \coloneqq z(z, x.x)$, we have $\mathrm{hv}(t') = z$ as well as $t' = \mathtt{H}'\langle\!\langle z \rangle\!\rangle$ with $\mathtt{H}' = \Diamond(z, x.\mathtt{H}_0\langle\!\langle x \rangle\!\rangle)$ and $\mathtt{H}_0 = \Diamond$. An example where the head variable is bound is $\mathrm{hv}(\lambda y.t) = y$, where $\lambda y.t = \mathtt{H}''\langle y \rangle$ and $\mathtt{H}'' = \lambda y.z(z, x.\Diamond)$.

Definition 3.10 (Solvability). Let $t \in \mathtt{T}_J$. Then,

t is **CbN-solvable** *iff* there is a head and a distant context $\mathtt{H}$ and $\mathtt{D}$ such that $\mathtt{H}\langle t\rangle \to^*_{\mathrm{djn}} \mathtt{D}\langle \mathtt{I}\rangle$.

t is **CbV-solvable** *iff* there is a head context $\mathtt{H}$ such that $\mathtt{H}\langle t\rangle \to^*_{\mathrm{djv}} \mathtt{I}$.

Notice that although the two definitions of CbN/CbV solvability are slightly different, they both share the same notion of head context, which is independent from the calculus.

In the definition of CbN-solvability, the reduction yields an identity plugged inside a distant context, and not just an identity alone. Take *e.g.* the term $t = \Omega(y, z.\mathtt{I})$ containing a non-relevant continuation, as $z \notin \mathrm{fv}(\mathtt{I})$. In the λ-calculus, t translates to $(\lambda z.\mathtt{I})(\Omega y)$, which is solvable since $(\lambda z.\mathtt{I})(\Omega y) \to_\beta \mathtt{I}$. This suggests introducing a garbage collection-like rule for generalized applications which reduces in this case $\Omega(y, z.\mathtt{I}) \to_{\mathrm{gc}} \mathtt{I}$. This would be consistent with different models of CbN, such as our quantitative type system. However, we prefer to avoid such ad-hoc solution, which can be simply seen as an implementation detail, as it does not change the operational and denotational behavior of terms.

Now, why does our notion of CbV solvability not use this distant context? Take again the term $t = \Omega(y, z.\mathtt{I})$ and its translated λ-term $(\lambda z.\mathtt{I})(\Omega y)$. CbV reduction in the λ-calculus loops on the argument Ωy, that could only be erased if Ωy is reduced to a value. Therefore, having a definition of solvability which reduces to $\mathtt{D}\langle\mathtt{I}\rangle$ in CbV would be too liberal, and incoherent with the λ-calculus and its associated models.

3.3 Call-by-Name Solvability

This section is organized in two parts. We first give an operational characterization of solvability with a reduction relation that we call *solving*, and then a quantitative type system capturing solvability.

3.3.1 Operational Characterization of CbN Solvability

The CbN solving reduction relation is not based on the full dβ rule. Take for instance the term $t = \delta(\delta, y.\mathtt{I})$. This term is solvable, because $t \to^*_{\mathrm{djn}} \mathtt{D}\langle\mathtt{I}\rangle$ in zero steps, with $\mathtt{D} = \delta(\delta, y.\Diamond)$. Yet, d$\beta$-reduction at root loops on this term. Since we want all solvable terms to be normalizable with the solving reduction, we do not want to execute any dβ-reduction, even at the root of the term. In fact, we only want to dβ-reduce a term $\mathtt{D}\langle\lambda x.t\rangle(u, y.r)$ when y is the head variable of r. In t, this is not the case.

Definition 3.11. The **CbN solving reduction** $\to_{\mathrm{sn}}$ is defined as the closure of the following reduction rule dβh under head contexts.

$$\mathtt{D}\langle\lambda x.t\rangle(u, y.\mathtt{H}\langle\!\langle y\rangle\!\rangle) \mapsto_{\mathrm{d}\beta\mathrm{h}} \mathtt{H}\langle\!\langle y\rangle\!\rangle\{y/\mathtt{D}\langle t\{x/u\}\rangle\}$$

With this definition, the term $\delta(\delta, y.\mathtt{I})$ is sn-normal, while not djn-normal. The term $\delta(\delta, y.y)$ is neither sn-normal nor djn-normal.

Definition 3.12. The following grammar $\mathrm{NF}_{\mathrm{sn}}$ intends to capture sn-nfs.

$$\begin{array}{rl} \textbf{(CbN Neutral Normal Contexts)} & \mathtt{G} ::= \Diamond \mid \mathtt{G}(u, x.\mathtt{G}\langle\!\langle x \rangle\!\rangle) \mid t(u, y.\mathtt{G}) \\ \textbf{(CbN Solving Normal Terms)} & \mathrm{NF}_{\mathrm{sn}} ::= x \mid \lambda x.\, \mathrm{NF}_{\mathrm{sn}} \mid \mathtt{G}\langle\!\langle x \rangle\!\rangle(u, y.\, \mathrm{NF}_{\mathrm{sn}}) \\ & \qquad \mid t(u, x.\, \mathrm{NF}_{\mathrm{sn}}) \;\text{ where } x \neq \mathrm{hv}(\mathrm{NF}_{\mathrm{sn}}) \end{array}$$

A **(CbN) neutral normal term** is a term $\mathtt{G}\langle\!\langle x \rangle\!\rangle$ for some $\mathtt{G}, x$. For example, the neutral normal term $\mathtt{I}(\mathtt{I}, w.x(\mathtt{I}, y.y))(\lambda y.z, z.z)$ is of the shape $\mathtt{G}\langle\!\langle x \rangle\!\rangle$ with $\mathtt{G} = \mathtt{I}(\mathtt{I}, w.\Diamond)$.

Lemma 3.13. *Let $t \in \mathtt{T}_J$. Then $t \in \mathrm{NF}_{\mathrm{sn}}$ iff $t \in$ sn-nf.*

Proof. For the left-to-right implication, we show the following two stronger properties by simultaneous induction on $\mathtt{G}, \mathrm{NF}_{\mathrm{sn}}$:

(i) t neutral normal $\implies$ t does not have an abstraction shape and t is in sn-nf.

(ii) $t \in \mathrm{NF}_{\mathrm{sn}} \implies$ t is in sn-nf.

Case $t = x$. Both (i) and (ii) are straightforward.

Case $t = \lambda y.s$, where $s \in \mathrm{NF}_{\mathrm{sn}}$. Then t is not neutral normal. Item (i) does not apply, and (ii) is straightforward by the *i.h.*

Case $t = \mathtt{G}\langle\!\langle x \rangle\!\rangle(u, y.r)$, where $r \in \mathrm{NF}_{\mathrm{sn}}$. By the *i.h.* (i) $\mathtt{G}\langle\!\langle x \rangle\!\rangle$ does not have an abstraction shape and $\mathtt{G}\langle\!\langle x \rangle\!\rangle, r$ are in sn-nf. Then t has no root sn-redex, and therefore t is in sn-nf. Moreover, if t is neutral normal, then $r = \mathtt{G}\langle\!\langle y \rangle\!\rangle$, and by the *i.h.* (i) r does not have an abstraction shape. Thus neither does t.

Case $t = s(u, y.r)$, where $r \in \mathrm{NF}_{\mathrm{sn}}$ and $y \neq \mathrm{hv}(r)$. Therefore, the only possible reduction would be inside r. By the *i.h.* (ii), r is sn-normal, so that t is sn-normal too. Moreover, if t is neutral normal, then r is neutral normal, and by the *i.h.* (i) r does not have an abstraction shape. Thus neither does t.

For the right-to-left implication, we show the following two stronger properties by simultaneous induction on t:

(i) t does not have an abstraction shape and t is in sn-nf t is neutral normal.

(ii) t is in sn-nf $\implies$ $t \in \mathrm{NF}_{\mathrm{sn}}$.

Case $t = x$. Both (i) and (ii) are straightforward since $x = \Diamond\langle\!\langle x \rangle\!\rangle$ is neutral normal and $x \in \mathrm{NF}_{\mathrm{sn}}$.

Case $t = \lambda y.s$. Then we only need to show (ii). Since s is necessarily in sn-nf, then $s \in \mathrm{NF}_{\mathrm{sn}}$ by the *i.h.* (ii), thus we conclude $t \in \mathrm{NF}_{\mathrm{sn}}$.

Case $t = s(u, x.\mathtt{H}\langle\!\langle x\rangle\!\rangle)$. Since t is in sn-nf, then s and $\mathtt{H}\langle\!\langle x\rangle\!\rangle$ are in sn-nf. Therefore, $\mathtt{H}\langle\!\langle x\rangle\!\rangle \in \mathrm{NF}_{\mathrm{sn}}$ by the *i.h.* (ii). The subterm s does not have an abstraction shape, otherwise t would sn-reduce at the root position, thus s is neutral normal (*i.e.* $s = \mathtt{G}\langle\!\langle y\rangle\!\rangle$) by the *i.h.* (i). We conclude $t \in \mathrm{NF}_{\mathrm{sn}}$. Moreover, if t does not have an abstraction shape, the same holds for $\mathtt{H}\langle\!\langle x\rangle\!\rangle$. By the *i.h.* (i) $\mathtt{H}\langle\!\langle x\rangle\!\rangle$ is neutral normal (*i.e.* of the form $\mathtt{G}'\langle\!\langle x\rangle\!\rangle$). Then, t is neutral normal too with $\mathtt{G}'' = \mathtt{G}(u, x.\mathtt{G}'\langle\!\langle x\rangle\!\rangle)$.

Case $t = s(u, x.r)$, where $x \neq \mathrm{hv}(r)$. Since t is sn-normal, then r is also sn-normal. We then have $r \in \mathrm{NF}_{\mathrm{sn}}$ by the *i.h.* (ii) and thus $t \in \mathrm{NF}_{\mathrm{sn}}$. If t does not have an abstraction shape, then r does not have an abstraction shape. By the *i.h.* (i) r is neutral normal (*i.e.* $r = \mathtt{G}\langle\!\langle y\rangle\!\rangle$) and thus $t = \mathtt{G}'\langle\!\langle y\rangle\!\rangle$, with $\mathtt{G}' = s(u, x.\mathtt{G})$. So that t is neutral normal too. □

We now prove that sn-normalizable terms are solvable. The converse implication will be given later with the help of the logical characterization. We use the following notation: $t(u_1, u_2, \dots, u_n, z.z)$ for the term $t(u_1, z.z(u_2, z.z(\dots(u_n, z.z))\dots)$. Notice that

$$\mathtt{D}\langle\lambda x.t\rangle(u_1, u_2, \dots, u_n, z.z) \mapsto_{\mathrm{dB}} \mathtt{D}\langle t\{x/u_1\}(u_2, \dots, u_n, z.z)\rangle$$

We abbreviate as $\overline{t(u, z.z)}^{n}$ the term $t(\underbrace{u, \dots, u}_{n}, z.z)$.

The measure $|t|_@$ below gives the number of hereditary head variables of the term t. We use it for several inductions on terms.

Definition 3.14.

$$|x|_@ = 0 \qquad |\lambda x.t|_@ = |t|_@ \qquad |t(u, x.r)|_@ = \begin{cases} |r|_@ + |t|_@ + 1, & \text{if } x = \mathrm{hv}(r) \\ |r|_@, & \text{otherwise} \end{cases}$$

This first lemma states that $\mathrm{NF}_{\mathrm{sn}}$ is stable by substitution of any variable which is not the head variable. For instance, $x(y, z.z)\{y/u\} \in \mathrm{NF}_{\mathrm{sn}}$ for any $u \in \mathtt{T}_J$, but $x(y, z.z)\{x/\mathtt{I}\} = \mathtt{I}(y, z.z) \notin \mathrm{NF}_{\mathrm{sn}}$.

Lemma 3.15. *For any $t \in \mathrm{NF}_{\mathrm{sn}}$ with $x = \mathrm{hv}(t)$, any term u and variable $y \neq x$, let $t' = t\{y/u\}$. Then $t' \in \mathrm{NF}_{\mathrm{sn}}$, $\mathrm{hv}(t') = x$ and $|t'|_@ = |t|_@$. If moreover t is neutral normal, then so is t'.*

Proof. Straightforward by induction on $\mathrm{NF}_{\mathrm{sn}}$. □

Then comes the main technical lemma. There are many distant contexts involved in the statement, but they are only here because of the absence of garbage collection. In particular, the context $\mathtt{D}_0$ is needed for the induction hypothesis. We will ignore them for this explanation.

The goal is to show that a term $t = \mathtt{H}\langle\!\langle x\rangle\!\rangle$ in sn-nf can be reduced to a value of the shape $\lambda x_m \dots \lambda x_1.\mathtt{o}^{n-|t|_@}$ simply by replacing its head variable by a projection $\mathtt{o}^n$. This is a crucial step to reduce the term to the identity, in order to show that it is solvable.

The abstractions λx_m to λx_1 are the abstractions already present in $\mathtt{H}$ between the root of the term and the head variable x. By taking $n = |t|_@$ (the generality of n is needed for the induction), we get $t\{x/\mathtt{o}^n\} \to_\beta^* \lambda x_m \dots \lambda x_1.\mathtt{I}$. The dummy abstractions can then be erased by applying m arguments, say $(\mathtt{I}, z.z)$ to the term, in order to obtain the identity. This will be explained in more details in the construction of a head context in the proof of the main property (property 3.19).

The idea is similar as in the λ-calculus, where the property is immediate: a head-normal term whose head variable is free is of the shape $\lambda x_m \dots \lambda x_1.xN_1 \dots N_n$, where $x \notin \{x_1, \dots, x_m\}$. Replacing x by $\mathtt{o}^n$ gives $\lambda x_m \dots \lambda x_1.\mathtt{o}^n N_1 \dots N_n \to_\beta^* \lambda x_m \dots \lambda x_1.\mathtt{o}^0 = \lambda x_m \dots \lambda x_1.\mathtt{I}$. Here, because of the syntax of generalized applications and the shape of sn-nfs, we must do a careful inductive proof.

Remark that this lemma uses β-reduction instead of $\mathrm{d}\beta$. This way, we can use it to prove the property *modularly* for the distant and the original versions of the calculus.

Lemma 3.16. *For all $t = \mathtt{H}\langle\!\langle x \rangle\!\rangle \in \mathrm{NF}_{\mathrm{sn}}$, integer $n \geq |t|_@$ and distant context $\mathtt{D}_0$, there is an integer $m \geq 0$, there are variables $x_1, \dots, x_m$ and distant contexts $\mathtt{D}, \mathtt{D}_1, \dots \mathtt{D}_m$ such that $t\{x/\mathtt{D}_0\langle\mathtt{o}^n\rangle\} \to_\beta^* \mathtt{D}\langle\lambda x_m.\mathtt{D}_m\langle\dots\lambda x_1.\mathtt{D}_1\langle\mathtt{o}^{n-|t|_@}\rangle\rangle\rangle$. In particular, if t is neutral normal, then $m = 0$.*

Proof. By induction on $\langle |t|_@, t\rangle$. We reason by cases on the form of the normal term t.

Case $t = x$. So $|t|_@ = 0$ and this is the base case of the induction. We let $m = 0$, $\mathtt{D} = \mathtt{D}_0$ and conclude since $x\{x/\mathtt{D}_0\langle\mathtt{o}^n\rangle\} = \mathtt{D}\langle\mathtt{o}^n\rangle = \mathtt{D}\langle\mathtt{o}^{n-|x|_@}\rangle$.

Case $t = \lambda y.t'$, where $t' = \mathtt{H}'\langle\!\langle x \rangle\!\rangle$ with $x \neq y$. We suppose w.l.o.g that $y \notin \mathrm{fv}(\mathtt{D}_0\langle\mathtt{o}^n\rangle)$. Let $n \geq |t|_@ = |t'|_@$. By the *i.h.* there are m', $x_1, \dots, x_{m'}$ and $\mathtt{D}', \mathtt{D}_1, \dots, \mathtt{D}_{m'}$ such that $t'\{x/\mathtt{D}_0\langle\mathtt{o}^n\rangle\} \to_\beta^* \mathtt{D}'\langle\lambda x_{m'}.\mathtt{D}_{m'}\langle\dots\lambda x_1.\mathtt{D}_1\langle\mathtt{o}^{n-|t'|_@}\rangle\rangle\rangle$. Thus we obtain $t\{x/\mathtt{D}_0\langle\mathtt{o}^n\rangle\} \to_\beta^* \lambda y.\mathtt{D}'\langle\lambda x_{m'}.\mathtt{D}_{m'}\langle\dots\lambda x_1.\mathtt{D}_1\langle\mathtt{o}^{n-|t'|_@}\rangle\rangle\rangle$ since $|t|_@ = |t'|_@$. We conclude by taking $m = m' + 1$, $x_m = y$, $\mathtt{D}_m = \mathtt{D}'$ and $\mathtt{D} = \Diamond$.

Case $t = s(u, y.r)$, where $r = \mathtt{H}'\langle\!\langle y \rangle\!\rangle$ $(y \neq x)$ and $s = \mathtt{G}\langle\!\langle x \rangle\!\rangle$. We have $|t|_@ = |s|_@ + |r|_@ + 1$. Let $n \geq |t|_@ > |s|_@$. Applying the *i.h.* on the neutral normal term s, we know that for any $\mathtt{D}_0$, there is $\mathtt{D}'$ such that $s\{x/\mathtt{D}_0\langle\mathtt{o}^n\rangle\} \to_\beta^* \mathtt{D}'\langle\mathtt{o}^{n-|s|_@}\rangle$.

Let $n' = n - |s|_@ - 1$. Then $n' \geq |r|_@$ and by lemma 3.15, $r\{x/\mathtt{D}_0\langle\mathtt{o}^{n'+1}\rangle\} \in \mathrm{NF}_{\mathrm{sn}}$ and $|r\{x/\mathtt{D}_0\langle\mathtt{o}^{n'+1}\rangle\}|_@ = |r|_@$. Moreover, $r\{x/\mathtt{D}_0\langle\mathtt{o}^{n'+1}\rangle\}$ is of the form $\mathtt{H}''\langle\!\langle y \rangle\!\rangle$, for some $\mathtt{H}''$. We can then apply the *i.h.* on $r\{x/\mathtt{D}_0\langle\mathtt{o}^{n'+1}\rangle\}$, so there are m', $x_1, \dots, x_{m'}, \mathtt{D}, \mathtt{D}_1, \dots, \mathtt{D}_{m'}$ such that $r\{x/\mathtt{D}_0\langle\mathtt{o}^{n'+1}\rangle\}\{y/\mathtt{D}'\langle\mathtt{o}^{n'}\rangle\} \to_\beta^* \mathtt{D}\langle\lambda x_{m'}.\mathtt{D}_{m'}\langle\dots\lambda x_1.\mathtt{D}_1\langle\mathtt{o}^{n'-|r|_@}\rangle\rangle\rangle$. We take $m = m'$. In the case where t is neutral normal, we have $m = 0$ as required. Since

$n' - |r|_@ = n - |t|_@$, we conclude as follows:

$$\begin{aligned} t\{x/\mathtt{D}_0\langle \mathtt{o}^n\rangle\} &= s\{x/\mathtt{D}_0\langle \mathtt{o}^n\rangle\}(u\{x/\mathtt{D}_0\langle \mathtt{o}^n\rangle\}, y.r\{x/\mathtt{D}_0\langle \mathtt{o}^n\rangle\}) \\ &\to_\beta^* \mathtt{D}'\langle \mathtt{o}^{n'+1}\rangle(u\{x/\mathtt{D}_0\langle \mathtt{o}^n\rangle\}, y.r\{x/\mathtt{D}_0\langle \mathtt{o}^n\rangle\}) \\ &\to_\beta r\{x/\mathtt{D}_0\langle \mathtt{o}^n\rangle\}\{y/\mathtt{D}'\langle \mathtt{o}^{n'}\rangle\} \\ &\to_\beta^* \mathtt{D}\langle \lambda x_m.\mathtt{D}_m\langle \dots \lambda x_1.\mathtt{D}_1\langle \mathtt{o}^{n-|t|_@}\rangle\rangle\rangle. \end{aligned}$$

Case $t = s(u, y.t')$, where $y \neq \mathrm{hv}(t')$. Let $n \geq |t|_@ = |t'|_@$. By the *i.h.* on t', for all $\mathtt{D}_0$ there are $m', x_1, \dots, x_{m'}, \mathtt{D}', \mathtt{D}_1, \dots, \mathtt{D}_{m'}$ s.t. $t'\{x/\mathtt{D}_0\langle \mathtt{o}^n\rangle\} \to_\beta^* \mathtt{D}'\langle \lambda x_{m'}.\mathtt{D}_{m'}\langle \dots \lambda x_1.\mathtt{D}_1\langle \mathtt{o}^{n-|t'|_@}\rangle\rangle\rangle$. In particular $m' = 0$ if t' is neutral normal. We set $\mathtt{D} = s\{x/\mathtt{D}_0\langle \mathtt{o}^n\rangle\}(u\{x/\mathtt{D}_0\langle \mathtt{o}^n\rangle\}, y.\mathtt{D}')$ and $m = m'$. Since $|t'|_@ = |t|_@$, then $t\{x/\mathtt{D}_0\langle \mathtt{o}^n\rangle\} \to_\beta^* \mathtt{D}\langle \lambda x_m.\mathtt{D}_m\langle \dots \lambda x_1.\mathtt{D}_1\langle \mathtt{o}^{n-|s|_@}\rangle\rangle\rangle$. □

Example 3.17. Let $t = y_1(\mathtt{I}, z_1.x)(y_2(\mathtt{I}, z_2.z_2), z_3.\lambda y.z_3) \in \mathrm{NF}_{\mathrm{sn}}$. Notice that $\mathrm{hv}(t) = x$. Then,

$$t\{x/\mathtt{o}^1\} = y_1(\mathtt{I}, z_1.\mathtt{o}^1)(y_2(\mathtt{I}, z_2.z_2), z_3.\lambda y.z_3) \to_\beta \lambda y.y_1(\mathtt{I}, z_1.\mathtt{o}^0) = \lambda y.y_1(\mathtt{I}, z_1.\mathtt{I})$$

We have a term of the desired shape, with $m = 1, \mathtt{D} = \Diamond$ and $\mathtt{D}_1 = y_1(\mathtt{I}, z_1.\Diamond)$.

This next lemma states that every sn-normal term has a subterm of the shape $\mathtt{H}\langle\!\langle x\rangle\!\rangle$ potentially surrounded by abstractions. Finding this subterm is important to use lemma 3.16.

Lemma 3.18. *Let $t \in \mathrm{NF}_{\mathrm{sn}}$ such that $\mathrm{hv}(t) = x$. Then there is an integer $l \geq 0$, there are variables $x_1, \dots, x_l, x$, distant contexts $\mathtt{D}_1, \dots, \mathtt{D}_l$ and a head context $\mathtt{H}$ such that $t = \mathtt{D}_l\langle \lambda x_l. \dots \mathtt{D}_1\langle \lambda x_1.\mathtt{H}\langle\!\langle x\rangle\!\rangle\rangle\rangle$. Moreover, if $x \in \mathrm{fv}(t)$, then $l = 0$.*

Proof. By induction on t.

Case $t = x$. We take $l = 0$ and $\mathtt{H} = \Diamond$.

Case $t = \lambda y.t'$. By the *i.h.* $t' = \mathtt{D}_{l'}\langle \lambda x_{l'}. \dots \mathtt{D}_1\langle \lambda x_1.\mathtt{H}\langle\!\langle x\rangle\!\rangle\rangle\rangle$ with $l' \geq 0$. We take $l = l' + 1$, $\mathtt{D}_l = \Diamond$ and $x_l = y$. The *moreover* part does not apply since t is not neutral.

Case $t = s(u, y.t')$. There are two possibilities.

- **Subcase** $\mathrm{hv}(t') = y$. Then $\mathrm{hv}(t) = \mathrm{hv}(s) = x$. By lemma 3.9, $t' = \mathtt{H}'\langle\!\langle y\rangle\!\rangle$ for some $\mathtt{H}'$. Moreover, by construction of solving normal terms we necessarily have $s = \mathtt{G}\langle\!\langle x\rangle\!\rangle$ for some $\mathtt{G}$, so that $x \in \mathrm{fv}(t)$. We thus take $\mathtt{H} = \mathtt{G}(u, y.\mathtt{H}'\langle\!\langle y\rangle\!\rangle)$ and $l = 0$, thus $t = \mathtt{H}\langle\!\langle x\rangle\!\rangle$ as required.
- **Subcase** $\mathrm{hv}(t') \neq y$. The *i.h.* gives $t' = \mathtt{D}'_{l'}\langle \lambda x_{l'}. \dots \mathtt{D}_1\langle \lambda x_1.\mathtt{H}'\langle\!\langle x\rangle\!\rangle\rangle\rangle$. We conclude with $\mathtt{H} = \mathtt{H}'$, $l = l'$ and $\mathtt{D}_l = s(u, y.\mathtt{D}'_{l'})$. If $x \in \mathrm{fv}(t)$, then $x \in \mathrm{fv}(t')$. The *i.h.* gives $l' = 0$. We conclude with $l = l' = 0$ and $\mathtt{H} = s(u, y.\mathtt{H}')$. □

Now comes the main property. The proof of this lemma consists in building an appropriate head context for any sn-normalizable term.

Property 3.19. *Let t be an* sn*-normalizable term. Then t is CbN solvable.*

Proof. Since t is sn-normalizable, then there is a solving normal term $t' \in \mathrm{NF}_{\mathrm{sn}}$ such that $t \to_{\mathrm{sn}}^{*} t'$ (and thus $t \to_{\mathrm{djn}}^{*} t'$). Let $\mathrm{hv}(t') = x$. By lemma 3.18, t' can take two shapes:

1. $t' = \mathtt{H}'\langle\!\langle x \rangle\!\rangle$ if $x \in \mathrm{fv}(t')$;
2. $t' = \mathtt{D}_l\langle \lambda x_l. \dots \lambda x_2.\mathtt{D}_1\langle \lambda x_1.\mathtt{H}'\langle\!\langle x \rangle\!\rangle\rangle\rangle$ for $x \in \{x_1, \dots, x_l\}$, if $x \notin \mathrm{fv}(t')$.

In both cases, we must give a head context $\mathtt{H}$ such that $\mathtt{H}\langle t\rangle \to_{\mathrm{djn}}^{*} \mathtt{D}\langle \mathtt{I}\rangle$ for a distant context $\mathtt{D}$.

We start with the first case (x is free in t'). Let $n = |t'|_{@}$. By lemma 3.16, there are $m \geq 0$, variables $y_1, \dots, y_m$ and distant contexts $\mathtt{D}', \mathtt{D}_1, \dots, \mathtt{D}_m$ such that $t'\{x/\mathtt{o}^n\} \to_{\beta}^{*} \mathtt{D}'\langle \lambda y_m.\mathtt{D}_m\langle \dots \lambda y_1.\mathtt{D}_1\langle \mathtt{I}\rangle\rangle\rangle$, which is also a djn-step. We let $\mathtt{H} = (\lambda x.\Diamond)(\mathtt{o}^n, z.z)\overline{(\mathtt{I}, z.z)}^{m}$. Then, we have:

$$\begin{aligned}
\mathtt{H}\langle t\rangle &\to_{\mathrm{djn}}^{*} \mathtt{H}\langle t'\rangle \\
&= (\lambda x.t')(\mathtt{o}^n, z.z)\overline{(\mathtt{I}, z.z)}^{m} \\
&\to_{\mathrm{djn}} t'\{x/\mathtt{o}^n\}\overline{(\mathtt{I}, z.z)}^{m} \\
&\to_{\mathrm{djn}}^{*} \mathtt{D}'\langle \lambda y_m.\mathtt{D}_m\langle \dots \lambda y_1.\mathtt{D}_1\langle \mathtt{I}\rangle\rangle\rangle\overline{(\mathtt{I}, z.z)}^{m} \\
&\to_{\mathrm{djn}}^{m} \mathtt{D}'\langle \mathtt{D}_m\langle \dots \mathtt{D}_1\langle \mathtt{I}\rangle\{y_1/\mathtt{I}\}\rangle\{y_m/\mathtt{I}\}\rangle
\end{aligned}$$

We conclude by taking $\mathtt{D} = \mathtt{D}'\langle \mathtt{D}_m\langle \dots \mathtt{D}_1\{y_1/\mathtt{I}\}\rangle\{y_m/\mathtt{I}\}\rangle$.

In the second case (x is not free in t'), let $1 \leq i \leq l$ such that $x = x_i$. Let us consider the following reduction sequence:

$$t'\overline{(\mathtt{I}, z.z)}^{l-i} = \mathtt{D}_l\langle \lambda x_l. \dots \mathtt{D}_1\langle \lambda x_1.\mathtt{H}'\langle\!\langle x \rangle\!\rangle\rangle\rangle\overline{(\mathtt{I}, z.z)}^{l-i} \to_{\mathrm{djn}}^{l-i} \mathtt{D}''\langle \lambda x.\mathtt{H}''\langle\!\langle x \rangle\!\rangle\rangle$$

where $\mathtt{D}'' = \mathtt{D}_l\langle \mathtt{D}_{l-1}\langle \dots \mathtt{D}_i\{x_{i+1}/\mathtt{I}\}\rangle\{x_l/\mathtt{I}\}\rangle$ and $\mathtt{H}'' = \mathtt{D}_{i-1}\langle \lambda x_{i-1}. \dots \mathtt{D}_1\langle \lambda x_1.\mathtt{H}'\rangle\rangle\{x_{j_{i<j\leq l}}/\mathtt{I}\}$. The subterm $\mathtt{H}''\langle\!\langle x \rangle\!\rangle$ above is obtained by substituting a sn-normal term with variables different from the head variable. By lemma 3.15, this kind of substitution preserves the property of being sn-normal, so that $\mathtt{H}''\langle\!\langle x \rangle\!\rangle$ is sn-normal.

Let $n = |\mathtt{H}''\langle\!\langle x \rangle\!\rangle|_{@}$. Then the lemma 3.16 applied to $\mathtt{H}''\langle\!\langle x \rangle\!\rangle\{x/\mathtt{o}^n\}$ gives integers $m, y_1, \dots, y_m$ and distant contexts $\mathtt{D}', \mathtt{D}'_1, \dots, \mathtt{D}'_m$ such that (this is also a djn-step):

$$\mathtt{H}''\langle\!\langle x \rangle\!\rangle\{x/\mathtt{o}^n\} \to_{\beta}^{*} \mathtt{D}'\langle \lambda y_m.\mathtt{D}'_m\langle \dots \lambda y_1.\mathtt{D}'_1\langle \mathtt{I}\rangle\rangle\rangle.$$

To conclude, we let $\mathtt{H} = \diamond\overline{(\mathtt{I}, z.z)}^{l-i}(\mathtt{o}^n, z.z)\overline{(\mathtt{I}, z.z)}^{m}$, where m and n were obtained before. The whole reduction from $\mathtt{H}\langle t\rangle$ goes as follows:

$$\begin{aligned}
\mathtt{H}\langle t\rangle &\to^*_{\mathrm{djn}} \mathtt{H}\langle t'\rangle \\
&= t'\overline{(\mathtt{I}, z.z)}^{l-i}(\mathtt{o}^n, z.z)\overline{(\mathtt{I}, z.z)}^{m} \\
&= \mathtt{D}_l\langle \lambda x_l. \dots \mathtt{D}_1\langle \lambda x_1.\mathtt{H}'\langle\!\langle x\rangle\!\rangle\rangle\rangle\overline{(\mathtt{I}, z.z)}^{l-i}(\mathtt{o}^n, z.z)\overline{(\mathtt{I}, z.z)}^{m} \\
&\to^{l-i}_{\mathrm{djn}} \mathtt{D}''\langle \lambda x.\mathtt{H}''\langle\!\langle x\rangle\!\rangle\rangle(\mathtt{o}^n, z.z)\overline{(\mathtt{I}, z.z)}^{m} \\
&\to_{\mathrm{djn}} \mathtt{D}''\langle \mathtt{H}''\langle\!\langle x\rangle\!\rangle\{x/\mathtt{o}^n\}\rangle\overline{(\mathtt{I}, z.z)}^{m} \\
&\to^*_{\mathrm{djn}} \mathtt{D}''\langle \mathtt{D}'\langle \lambda y_m.\mathtt{D}'_m\langle \dots \lambda y_1.\mathtt{D}'_1\langle \mathtt{I}\rangle\rangle\rangle\rangle\overline{(\mathtt{I}, z.z)}^{m} \\
&\to^m_{\mathrm{djn}} \mathtt{D}''\langle \mathtt{D}'\langle \mathtt{D}'_m\langle \dots \mathtt{D}'_1\langle \mathtt{I}\rangle\{y_1/\mathtt{I}\}\rangle\{y_m/\mathtt{I}\}\rangle\rangle
\end{aligned}$$

where $\mathtt{D}'' = \mathtt{D}_l\langle \mathtt{D}_{l-1}\langle \dots \mathtt{D}_i\{x_{i+1}/\mathtt{I}\}\rangle\{x_l/\mathtt{I}\}\rangle$ and $\mathtt{H}'' = \mathtt{D}_{i-1}\langle \lambda x_{i-1}. \dots \mathtt{D}_1\langle \lambda x_1.\mathtt{H}'\rangle\rangle\{x_{j_{i<j\le l}}/\mathtt{I}\}$.

We conclude the proof by taking $\mathtt{D} = \mathtt{D}''\langle \mathtt{D}'\langle \mathtt{D}'_m\langle \dots \mathtt{D}'_1\{y_1/\mathtt{I}\}\rangle\{y_m/\mathtt{I}\}\rangle\rangle$ so that $\mathtt{H}\langle t\rangle \to^*_{\mathrm{djn}} \mathtt{D}\langle \mathtt{I}\rangle$. □

Example 3.20. Take again $t = y_1(\mathtt{I}, z_1.x)(y_2(\mathtt{I}, z_2.z_2), z_3.\lambda y.z_3) \in \mathrm{NF}_{\mathrm{sn}}$ from example 3.17. We take $\mathtt{H} = (\lambda x.\diamond)(\mathtt{o}^1, z.z)(\mathtt{I}, z.z)$. Then,

$$\begin{aligned}
\mathtt{H}\langle t\rangle &= (\lambda x.y_1(\mathtt{I}, z_1.x)(y_2(\mathtt{I}, z_2.z_2), z_3.\lambda y.z_3))(\mathtt{o}^1, z.z)(\mathtt{I}, z.z) \\
&\to_{\mathrm{djn}} y_1(\mathtt{I}, z_1.\mathtt{o}^1)(y_2(\mathtt{I}, z_2.z_2), z_3.\lambda y.z_3)(\mathtt{I}, z.z) \\
&\to_{\mathrm{djn}} (\lambda y.y_1(\mathtt{I}, z_1.\mathtt{I}))(\mathtt{I}, z.z) \\
&\to_{\mathrm{djn}} y_1(\mathtt{I}, z_1.\mathtt{I})
\end{aligned}$$

Taking $\mathtt{D} = y_1(\mathtt{I}, z_1.\diamond)$, we get a term of the expected form $\mathtt{D}\langle \mathtt{I}\rangle$.

3.3.2 Logical Characterization of CbN Solvability

We now give a type system, called $\cap N$, in which typability and normalization of solving reduction coincide, *i.e.* not only does typability imply normalization, but the converse implication also holds.

Types, multiset types and type derivations are defined as in section 1.3.2. The quantitative type system $\cap N$ is defined in figure 3.1. This system is a natural extension of Gardner's [Gar94] and De Carvalho's [dCar17] systems to generalized applications. Rule (MANY) may assign the empty multiset to any term (case $I = \emptyset$), so being typable with $[\,]$ means in fact being untyped. The interesting rule is (APP), where both t and u are assigned multiset types, since x is not necessarily linear in r. Because u is the argument of t, it is assigned all the types on the left of the arrow of t. The size of derivations is given by the number of rules (APP): we write $\Gamma \Vdash^n_{\cap N} t : \sigma$ a derivation of size n in the system. This derivation

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (\textsc{var}) \qquad \frac{\Gamma; x : \mathscr{M} \vdash t : \sigma}{\Gamma \vdash \lambda x.t : \mathscr{M} \to \sigma}\ (\textsc{abs}) \qquad \frac{(\Gamma_i \vdash t : \sigma_i)_{i\in I}}{\uplus_{i\in I}\Gamma_i \vdash t : [\sigma_i]_{i\in I}}\ (\textsc{many})$$

$$\frac{\Gamma \vdash t : [\mathscr{M}_i \to \sigma_i]_{i\in I} \qquad \Delta \vdash u : \sqcup_{i\in I}\mathscr{M}_i \qquad \Lambda; x : [\sigma_i]_{i\in I} \vdash r : \tau}{\Gamma \uplus \Delta \uplus \Lambda \vdash t(u, x.r) : \tau}\ (\textsc{app})$$

Figure 3.1: System $\cap N$.

measure is sufficient to capture the fact that each sn-step deletes at least one (app) rule (see lemma 3.28).

The system is relevant, as there is no weakening.

Lemma 3.21 (Relevance). *If* $\Gamma \Vdash_{\cap N} t : \sigma$, *then* $\mathrm{dom}(\Gamma) \subseteq \mathrm{fv}(t)$.

Proof. Straightforward by induction on the derivation. □

Example 3.22. Take $t = \Omega(y, z.\mathtt{I})$ (we expand $\mathtt{I}$ to $\lambda x.x$ in the derivation). Although the evaluation of the subterm Ω is not terminating (and thus Ω can only be typed with the empty multiset), t is typable:

$$\frac{\dfrac{}{\emptyset \vdash \Omega : [\,]}\ (\textsc{many}) \qquad \dfrac{}{\emptyset \vdash y : [\,]}\ (\textsc{many}) \qquad \dfrac{\dfrac{}{x : [\sigma] \vdash x : \sigma}\ (\textsc{var})}{\emptyset \vdash \lambda x.x : [\sigma] \to \sigma}\ (\textsc{abs})}{\emptyset \vdash \Omega(y, z.\lambda x.x) : [\sigma] \to \sigma}\ (\textsc{app})$$

Although not every subterm must be typed in a derivation of $\cap N$, any subterm that is at the head of the term, and in particular the head variable, must be.

Lemma 3.23.

(i) *For any context* $\mathtt{H}$ *and term* t, *if* $\Gamma \Vdash_{\cap N} \mathtt{H}\langle t\rangle : \sigma$, *then there are* Γ', τ *such that* $\Gamma' \Vdash_{\cap N} t : \tau$.

(ii) *If* $\Gamma \Vdash_{\cap N} \mathtt{H}\langle\langle x\rangle\rangle : \sigma$, *then* $\Gamma = \Delta; x : [\tau_i]_{i\in I}$ *and* $I \neq \emptyset$.

Proof. Straightforward by induction on $\mathtt{H}$. □

The split lemma will be needed for the proof.

Lemma 3.24 (Split).

(i) *If* $\Gamma \Vdash^{\cap N}_n t : \mathscr{M}$, *then for any decomposition* $\mathscr{M} = \sqcup_{i\in I}\mathscr{M}_i$, *then we have* $\Gamma_i \Vdash^{n_i}_{\cap N} t : \mathscr{M}_i$ *such that* $\sum_{i\in I} n_i = n$ *and* $\uplus_{i\in I}\Gamma_i = \Gamma$.

(ii) *If* $\Gamma_i \Vdash^{n_i}_{\cap N} t : \mathscr{M}_i$ *for all* $i \in I$, *then* $\Gamma \Vdash^{n}_{\cap N} t : \mathscr{M}$, *where* $\mathscr{M} = \sqcup_{i\in I}\mathscr{M}_i$, $n = \sum_{i\in I} n_i$ *and* $\Gamma = \uplus_{i\in I}\Gamma_i$.

Proof. Straightforward by induction on the derivation. □

We now prove that terms typable in $\cap N$ are exactly the ones that normalize with $\to_{\mathrm{sn}}$. The proof method is the same as in section 1.3.2.

Soundness

We first need to prove the *substitution lemma*, which also relates the sizes of the corresponding derivations.

Lemma 3.25 (Substitution for $\cap N$). *If* $\Gamma; x : \mathscr{M} \Vdash^{n}_{\cap N} t : \sigma$ *and* $\Delta \Vdash^{m}_{\cap N} u : \mathscr{M}$, *then there is a derivation* $\Gamma \uplus \Delta \Vdash^{m+n}_{\cap N} t\{x/u\} : \sigma$.

Proof. By induction on t.

Case $t = x$. By hypothesis, $\Gamma = \emptyset$, $\mathscr{M} = [\sigma]$ and $n = 0$. We can conclude with $\emptyset \uplus \Delta \Vdash^{0+m} x\{x/u\} : \sigma = \Delta \Vdash^{m} u : \sigma$, which we have by hypothesis.

Case $t = y \neq x$. By hypothesis, $\mathscr{M} = [\,]$, $\Gamma = y : \sigma$ and $n = 0$. We necessarily have $\emptyset \Vdash^{0} u : [\,]$ obtained by rule (MANY). We conclude with $\Gamma \uplus \emptyset \Vdash^{0+0} y\{x/u\} : \sigma = y : [\sigma] \Vdash^{0} y : \sigma$, which holds by hypothesis.

Case $t = \lambda y.s$, where $y \neq x$ and $y \notin \mathrm{fv}(u)$. By hypothesis we have $\sigma = \mathscr{N} \to \tau$ and $\Gamma; x : \mathscr{M}; y : \mathscr{N} \Vdash^{n} s : \tau$. By the *i.h.* we obtain $\Gamma \uplus \Delta; y : \mathscr{N} \Vdash^{n+m} s\{x/u\} : \tau$ because by lemma 3.21 $y \notin \mathrm{dom}(\Delta)$. By rule (ABS) and because $y \neq x$, we obtain $\Gamma \uplus \Delta \Vdash^{m+n} \lambda y.s\{x/u\} : \mathscr{N} \to \tau$. We can conclude because $\lambda y.s\{x/u\} = (\lambda y.s)\{x/u\}$.

Case $t = s(u', y.r)$, where $y \neq x$ and $y \notin \mathrm{fv}(u)$. By hypothesis, $\Gamma_1; x : \mathscr{M}_1 \Vdash^{n_1} s : [\mathscr{N}_i \to \tau_i]_{i\in I}$, $\Gamma_2; x : \mathscr{M}_2 \Vdash^{n_2} u' : \sqcup_{i\in I}\mathscr{N}_i$ and $\Gamma_3; x : \mathscr{M}_3; y : [\tau_i]_{i\in I} \Vdash^{n_3} r : \sigma$ where $\Gamma = \Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3$, $\mathscr{M} = \mathscr{M}_1 + \mathscr{M}_2 + \mathscr{M}_3$ and $n = n_1 + n_2 + n_3 + 1$. By lemma 3.24:1, $\Delta_i \Vdash^{m_i} u : \mathscr{M}_i$ $(i = 1, 2, 3)$ where $\Delta = \Delta_1 + \Delta_2 + \Delta_3$ and $m = m_1 + m_2 + m_3$. First, notice that lemma 3.21 states that $y \notin \mathrm{dom}(\Delta)$ as well as in particular $y \notin \mathrm{dom}(\Delta_3)$. By the *i.h.*, $\Gamma_1 \uplus \Delta_1 \Vdash^{n_1+m_1} s\{x/u\} : [\mathscr{N}_i \to \tau_i]_{i\in I}$, $\Gamma_2 \uplus \Delta_2 \Vdash^{n_2+m_2} u'\{x/u\} : \sqcup_{i\in I}\mathscr{N}_i$, and $\Gamma_3 \uplus \Delta_3; y : \mathscr{N}' \Vdash^{n_3+m_3} s\{x/u\} : \sigma$. We conclude using rule (APP), the fact that $(s(u', y.r))\{x/u\} = s\{x/u\}(u'\{x/u\}, y.r\{x/u\})$ and $n + m = 1 + \sum_{i=1}^{3}(n_i + m_i)$. □

In order to keep the proofs of characterization of the distant calculus and the original calculus modular, weighted subject reduction will be done in several steps. The first step is to prove it for root reduction only, for p2 and two new reduction steps: βh (dβh without distance) and πh (a head variant of π and p2).

Definition 3.26. The following rules are the core of head reduction in the original ΛJ.

$$(\lambda x.t)(u, y.\mathrm{H}\langle\!\langle y\rangle\!\rangle) \longmapsto_{\beta\mathrm{h}} \mathrm{H}\langle\!\langle y\rangle\!\rangle\{y/t\{x/u\}\}$$
$$t(u, x.r)(u', y.\mathrm{H}\langle\!\langle y\rangle\!\rangle) \longmapsto_{\pi\mathrm{h}} t(u, x.r(u', y.\mathrm{H}\langle\!\langle y\rangle\!\rangle))$$

Lemma 3.27. *Let* $\Gamma \Vdash^n_{\cap N} t_1 : \sigma$.

(i) *If* $t_1 \longmapsto_{\beta\mathrm{h}} t_2$, *then* $\Gamma \Vdash^{n-1}_{\cap N} t_2 : \sigma$.

(ii) *If* $t_1 \longmapsto_{\mathrm{p2}} t_2$, *then* $\Gamma \Vdash^{n}_{\cap N} t_2 : \sigma$.

(iii) *If* $t_1 \longmapsto_{\pi\mathrm{h}} t_2$, *then* $\Gamma \Vdash^{n'}_{\cap N} t_2 : \sigma$ *with* $n' \leq n$.

Proof. We prove each of the items successively.

Case $t_1 = (\lambda x.t)(u, y.r) \longmapsto_{\beta\mathrm{h}} r\{y/t\{x/u\}\} = t_2$, where $r = \mathrm{H}\langle\!\langle y\rangle\!\rangle$. We have the derivation below, with $\Gamma = \uplus_{i\in I}(\Sigma_i \uplus \Delta_i) \uplus \Lambda$, $n = \sum_{i\in I}(n^i_t + n^i_u) + n_r + 1$. Notice that I is never empty because y is the head variable of r and is thus always typed, by lemma 3.23.

$$\dfrac{\dfrac{\left(\Sigma_i \Vdash^{n^i_t} \lambda x.t : \mathscr{M}_i \longrightarrow \tau_i\right)_{i\in I}}{\uplus_{i\in I}\Sigma_i \vdash \lambda x.t : [\mathscr{M}_i \longrightarrow \tau_i]_{i\in I}} \qquad \dfrac{(\Delta_i \Vdash^{n^i_u} u : \mathscr{M}_i)_{i\in I}}{\uplus_{i\in I}\Delta_i \vdash u : \sqcup_{i\in I}\mathscr{M}_i} \qquad \Lambda; y : [\tau_i]_{i\in I} \Vdash^{n_r} r : \sigma}{\uplus_{i\in I}(\Sigma_i \uplus \Delta_i) \uplus \Lambda \vdash (\lambda x.t)(u, y.r) : \sigma}$$

The substitution lemma 3.25 gives $\Sigma_i \uplus \Delta_i \Vdash^{n^i_t + n^i_u} t\{x/u\} : \tau_i$, so that we have a derivation $\uplus_{i\in I}(\Sigma_i \uplus \Delta_i) \Vdash^{+_{i\in I}(n^i_t+n^i_u)} t\{x/u\} : [\tau_i]_{i\in I}$. Applying the substitution lemma 3.25 again gives $\Gamma \Vdash^{n-1} t_2 = r\{y/t\{x/u\}\} : \sigma$.

Case $t_1 = t(u, y.\lambda x.r) \longmapsto_{\mathrm{p2}} \lambda x.t(u, y.r)$. Notice that σ is necessarily an arrow type $\mathscr{N} \longrightarrow \tau$. We have the following derivation, with $n = n_t + n_u + n_r + 1$ and $\Gamma = \Sigma \uplus \Delta \uplus \Lambda$.

$$\dfrac{\Sigma \Vdash^{n_t} t : [\mathscr{M}_i \longrightarrow \tau_i]_{i\in I} \qquad \Delta \Vdash^{n_u} u : \sqcup_{i\in I}\mathscr{M}_i \qquad \dfrac{\Lambda; y : [\tau_i]_{i\in I}; x : \mathscr{N} \Vdash^{n_r} r : \tau}{\Lambda; y : [\tau_i]_{i\in I} \vdash \lambda x.r : \mathscr{N} \longrightarrow \tau}}{\Sigma \uplus \Delta \uplus \Lambda \vdash t(u, y.\lambda x.r) : \sigma}$$

By α-conversion, $x \notin \mathrm{fv}(t) \cup \mathrm{fv}(u)$, so that $x \notin \mathrm{dom}(\Sigma \uplus \Delta)$ by lemma 3.21. We can then build the following derivation of the same size:

$$\dfrac{\dfrac{\Sigma \Vdash^{n_t} t : [\mathscr{M}_i \longrightarrow \tau_i]_{i\in I} \qquad \Delta \Vdash^{n_u} u : \sqcup_{i\in I}\mathscr{M}_i \qquad \Lambda; y \Vdash^{n_r} [\tau_i]_{i\in I}; x : \mathscr{N} : r : \tau}{\Sigma \uplus \Delta \uplus (\Lambda; x : \mathscr{N}) \vdash t(u, y.r) : \tau}}{\Sigma \uplus \Delta \uplus \Lambda \vdash \lambda x.t(u, y.r) : \mathscr{N} \longrightarrow \tau}$$

Case $t_1 = t(u, x.r)(u', y.r') \mapsto_{\pi\mathrm{h}} t(u, x.r(u', y.r')) = t_2$, where $r' = \mathtt{H}\langle\!\langle y \rangle\!\rangle$. We have the following derivation:

$$\dfrac{\left(\dfrac{\Phi_t^i \qquad \Phi_u^i \qquad \Phi_r^i}{\Gamma_t^i \uplus \Gamma_u^i \uplus \Gamma_r^i \vdash t(u, x.r) : \mathscr{M}_i \to \sigma_i}\,(\textsc{app})\right)_{i\in I}}{\dfrac{\uplus_{i\in I}(\Gamma_t^i \uplus \Gamma_u^i \uplus \Gamma_r^i) \vdash t(u, x.r) : [\mathscr{M}_i \to \sigma_i]_{i\in I} \qquad \Phi_{u'} \qquad \Phi_{r'}}{\Gamma \vdash t(u, x.r)(u', y.r') : \sigma}\,(\textsc{app})}\,(\textsc{many})$$

where $\Phi_{u'} = \Gamma_{u'} \Vdash^{n_{u'}} u' : \sqcup_{i\in I}\mathscr{M}_i$, $\Phi_{r'} = \Gamma_{r'}; y : [\sigma_i]_{i\in I} \Vdash^{n_{r'}} r' : \sigma$ and for all $i \in I$: $\Phi_t^i = \Gamma_t^i \Vdash^{n_t^i} t : [\mathscr{N}_j \to \tau_j]_{j\in J_i}$, $\Phi_u^i = \Gamma_u^i \Vdash^{n_u^i} u : \sqcup_{j\in J_i}\mathscr{N}_j$, $\Phi_r^i = \Gamma_r^i; x : [\tau_j]_{j\in J_i} \Vdash^{n_r^i} r : \mathscr{M}_i \to \sigma_i$, such that $\Gamma = \uplus_{i\in I}(\Gamma_t^i \uplus \Gamma_u^i \uplus \Gamma_r^i) \uplus \Gamma_{u'} \uplus \Gamma_{r'}$ and $n = \sum_{i\in I}(n_t^i + n_u^i + n_r^i) + n_{u'} + n_{r'} + |I| + 1$. Notice that I is again never empty because y is the head variable of r and is thus always typed, by lemma 3.23.

Let $J = \uplus_{i\in I} J_i$, $n_t = \sum_{i\in I} n_t^i$, $n_u^i = \sum_{i\in I} n_u^i$ and $n_r = \sum_{i\in I} n_r^i$. By rule (MANY), we have derivations $\Phi_t = \uplus_{i\in I}\Gamma_t^i \Vdash^{n_t} t : [\mathscr{N}_j \to \tau_j]_{j\in J}$, $\Phi_u = \uplus_{i\in I}\Gamma_u^i \Vdash^{n_u} u : \sqcup_{j\in J}\mathscr{N}_j$ and $\Phi_r = \uplus_{i\in I}\Gamma_r^i; x : [\tau_j]_{j\in J} \Vdash^{n_r} r : [\mathscr{M}_i \to \sigma_i]_{i\in I}$. Using the fact that $x \notin \mathrm{fv}(u') \cup \mathrm{fv}(r')$ and the relevance lemma 3.21, we build the following derivation.

$$\dfrac{\Phi_t \qquad \Phi_u \qquad \dfrac{\Phi_r \qquad \Phi_{u'} \qquad \Phi_{r'}}{(\uplus_{i\in I}\Gamma_r^i; x : [\tau_j]_{j\in J}) \uplus \Gamma_{u'} \uplus \Gamma_{r'} \vdash r(u', y.r') : \sigma}\,(\textsc{app})}{\Gamma \vdash t(u, x.r(u', y.r')) : \sigma}\,(\textsc{app})$$

The derivation is of size $n' = n_t + n_u + n_r + n_{u'} + n_{r'} + 2 \le n$ since $|I| \ge 1$. □

We prove weighted subject reduction for the full sn relation by induction on the reduction step. In the base case we use weighted subject reduction for p2 and for βh, since a dβh-step is made of a potentially empty series of p2-steps followed by a βh-steps: $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u, y.\mathtt{H}\langle\!\langle y \rangle\!\rangle) \mapsto_{\mathrm{sn}} \mathtt{H}\langle\!\langle y \rangle\!\rangle\{y/\mathtt{D}\langle t\{x/u\}\rangle\} = t_2$ is decomposed into

$$t_1 \mapsto_{\mathrm{p2}}^* (\lambda x.\mathtt{D}\langle t\rangle)(u, y.\mathtt{H}\langle\!\langle y \rangle\!\rangle) \mapsto_{\beta\mathrm{h}} t_2$$

Lemma 3.28 (Weighted subject reduction for $\cap N$). *If* $\Gamma \Vdash^{n_1}_{\cap N} t_1 : \sigma$ *and* $t_1 \to_{\mathrm{sn}} t_2$, *then* $\Gamma \Vdash^{n_2}_{\cap N} t_2 : \sigma$ *with* $n_1 > n_2$.

Proof. By induction on $t_1 \to_{\mathrm{sn}} t_2$. We can generalize the statement to multi-types as follows: if $\Gamma \Vdash^{n_1}_{\cap N} t_1 : \mathscr{M}$ and $t_1 \to_{\mathrm{sn}} t_2$, then $\Gamma \Vdash^{n_2}_{\cap N} t_2 : \mathscr{M}$ with $n_1 > n_2$. We show the general statement by induction on $\to_{\mathrm{sn}}$.

Case $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u, y.r) \mapsto_{\mathrm{d}\beta\mathrm{h}} r\{y/\mathtt{D}\langle t\{x/u\}\rangle\} = t_2$ where $r = \mathtt{H}\langle\!\langle y \rangle\!\rangle$. Let $t_3 = \lambda x.\mathtt{D}\langle t\rangle(u, y.r)$. We have $t_1 \mapsto_{\mathrm{p2}}^* t_3 \mapsto_{\beta\mathrm{h}} r\{y/\{\langle x\rangle/u\}\mathtt{D}t\} = t_2$. By lemma 3.27(ii) we have $\Gamma \Vdash^{n_1} t_3 : \sigma$. By lemma 3.27(i) we have $\Gamma \Vdash^{n_2} t_2 : \sigma$ where $n_2 = n_1 - 1$.

Case $t_1 = \lambda x.t \to_{\mathrm{sn}} \lambda x.t' = t_2$, where $t \to_{\mathrm{sn}} t'$. By hypothesis, we have $\sigma = \mathscr{M} \to \tau$ and $\Gamma; x : \mathscr{M} \Vdash^{n_1} t : \sigma$. By the *i.h.* we have $\Gamma; x : \mathscr{M} \Vdash^{n_2} t' : \tau$ with $n_1 > n_2$. We use rule (ABS) to build a derivation of t_2 of size n_2.

Case $t_1 = t(u, x.r)$ and the reduction is internal. By hypothesis, we have the following derivations:

$$\frac{\Sigma \Vdash^{n_t} t : [\mathscr{M}_i \to \tau_i]_{i\in I} \qquad \Delta \Vdash^{n_u} u : \sqcup_{i\in I}\mathscr{M}_i \qquad \Lambda; y : [\tau_i]_{i\in I} \Vdash^{n_r} r : \sigma}{\Sigma \uplus \Delta \uplus \Lambda \vdash t(u, y.r) : \sigma}$$

where $\Gamma = \Gamma \uplus \Delta \uplus \Lambda$ and $n_1 = n_t + n_u + n_r + 1$. There are three possibilities:

Subcase $t_1 \to_{\mathrm{sn}} t'(u, x.r) = t_2$, where $t \to_{\mathrm{sn}} t'$ and $r = \mathtt{H}\langle\!\langle x \rangle\!\rangle$. By *i.h.* there is a derivation $\Sigma \Vdash^{n_{t'}} t' : [\mathscr{M}_i \to \tau_i]_{i\in I}$ such that $n_t \geq n_{t'}$. Since x is the head variable of r, we have $I \neq \emptyset$ by lemma 3.23, so that $n_t > n_{t'}$. We can build a derivation of t_2 of size $n_2 = 1 + n_{t'} + n_u + n_r$ and we get $n_1 > n_2$.

Subcase $t_1 \to_{\mathrm{sn}} t(u, x.r') = t_2$ where $r \to_{\mathrm{sn}} r'$. By the *i.h.* there is a derivation $\Lambda; x \Vdash^{n_{r'}} [\tau_i]_{i\in I} : r : \sigma$ such that $n_r > n_{r'}$. We can build a derivation of t_2 of size $n_2 = 1 + n_t + n_u + n_{r'}$ and we get $n_1 > n_2$. □

The size of type derivations is a natural number decreasing at every step, so that soundness is, as expected, a direct corollary.

Corollary 3.29 (Soundness for λJ_n). *If* $\Gamma \Vdash^n_{\cap N} t : \sigma$, *then* t *is* sn*-normalizable and the number of* sn*-steps needed to normalize* t *is bounded by* n.

Completeness

To prove completeness of the typing, we first need to show the anti-substitution lemma.

Lemma 3.30 (Anti-substitution for $\cap N$). *If* $\Gamma \Vdash t\{x/u\} : \sigma$, *then there exists* Γ_t, Γ_u *and* $\mathscr{M}$ *such that* $\Gamma_t; x : \mathscr{M} \Vdash t : \sigma, \Gamma_u \Vdash u : \mathscr{M}$ *and* $\Gamma = \Gamma_t \uplus \Gamma_u$.

Proof. By induction on the derivation $\Gamma \Vdash t\{x/u\} : \sigma$. We extend the statement to derivations ending with (MANY), for which the property is straightforward by the *i.h.* We reason by cases on t.

Case $t = x$. Then $t\{x/u\} = u$. We take $\Gamma_t = \emptyset, \Gamma_u = \Gamma, \mathscr{M} = [\sigma]$, and we have $x : [\sigma] \Vdash x : \sigma$ by rule (VAR) and $\Gamma \Vdash u : \mathscr{M}$ by rule (MANY) on the derivation of the hypothesis.

Case $t = y \neq x$. Then $t\{x/u\} = y$. We then have $\Gamma = y : [\sigma]$. We take $\Gamma_t = \Gamma, \Gamma_u = \emptyset$, $\mathscr{M} = [\,]$, and then we have $y : [\sigma]; x : [\,] \Vdash y : \sigma$ by hypothesis and $\emptyset \Vdash u : [\,]$ by rule (MANY).

Case $t = \lambda y.s$ where $y \neq x$ and $y \notin \mathrm{fv}(u)$ and $x \in \mathrm{fv}(s)$. Then $t\{x/u\} = \lambda y.s\{x/u\}$. We have $\sigma = \mathscr{N} \to \tau$ and $\Gamma; y : \mathscr{N} \Vdash s\{x/u\} : \tau$.

By the *i.h.* there exists $\Gamma', \Gamma_u, \mathscr{M}$ such that $\Gamma'; y : \mathscr{N}; x : \mathscr{M} \Vdash s : \tau, \Gamma_u \Vdash u : \mathscr{M}$, and $\Gamma; y : \mathscr{N} = (\Gamma'; y : \mathscr{N}) \uplus \Gamma_u$. Moreover, by α-conversion and lemma 3.21 we know that $y \notin \mathrm{dom}(\Gamma_u)$ so that $\Gamma = \Gamma' \uplus \Gamma_u$. We conclude by deriving $\Gamma'; y : \mathscr{N} \Vdash \lambda x.s : \mathscr{N} \to \tau$ with rule (ABS). Indeed, by letting $\Gamma_t = \Gamma'$ we have $\Gamma = \Gamma_t \uplus \Gamma_u$ as required.

Case $t = s(u', y.r)$ where $y \neq x$ and $y \notin \mathrm{fv}(u)$. By construction, we have derivations $\Gamma_1 \Vdash s\{x/u\} : [\mathscr{N}_i \to \tau_i]_{i\in I}$, $\Gamma_2 \Vdash u'\{x/u\} : \sqcup_{i\in I}\mathscr{N}_i$ and $\Gamma_3; y : [\tau_i]_{i\in I} \Vdash r\{x/u\} : \sigma$, with $\Gamma = \Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3$.

By the induction hypothesis there are environments $\Gamma_s, \Gamma_{u'}, \Gamma_r, \Gamma_u^1, \Gamma_u^2, \Gamma_u^3$ and multiset types $\mathscr{M}_1, \mathscr{M}_2, \mathscr{M}_3$ such that $\Gamma_s; x : \mathscr{M}_1 \Vdash s : [\mathscr{N}_i \to \tau_i]_{i\in I}$, $\Gamma_{u'}; x : \mathscr{M}_2 \Vdash u' : \sqcup_{i\in I}\mathscr{N}_i$, $\Gamma_r; x : \mathscr{M}_3 \Vdash r : \sigma$, $\Gamma_u^1 \Vdash u : \mathscr{M}_1$, $\Gamma_u^2 \Vdash u : \mathscr{M}_2$, $\Gamma_u^3 \Vdash u : \mathscr{M}_3$ and $\Gamma_1 = \Gamma_s \uplus \Gamma_u^1$, $\Gamma_2 = \Gamma_{u'} \uplus \Gamma_u^2$, $\Gamma_3 = \Gamma_r \uplus \Gamma_u^3$. Let $\Gamma_t = \Gamma_s \uplus \Gamma_{u'} \uplus \Gamma_r$, $\Gamma_u = \Gamma_u^1 \uplus \Gamma_u^2 \uplus \Gamma_u^3$ and $\mathscr{M} = \mathscr{M}_1 \sqcup \mathscr{M}_2 \sqcup \mathscr{M}_3$. We can build a derivation $\Gamma_t; x : \mathscr{M} \Vdash s(u', y.r) : \sigma$ with rule (APP) and a derivation $\Gamma_u \Vdash u : \mathscr{M}$ with lemma 3.24:2. We conclude since $\Gamma = \Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3 = \Gamma_s \uplus \Gamma_u^1 \uplus \Gamma_{u'} \uplus \Gamma_u^2 \uplus \Gamma_r \uplus \Gamma_u^3 = \Gamma_t \uplus \Gamma_u$. □

As for weighted subject reduction, we now prove subject expansion in several steps, the first one consisting of the root reductions of βh, πh and p2.

Lemma 3.31. *Let* $\Gamma \Vdash_{\cap N} t_2 : \sigma$ *and* $t_1 \mapsto_{\{\beta\mathrm{h},\mathrm{p2},\pi\mathrm{h}\}} t_2$. *Then* $\Gamma \Vdash_{\cap N} t_1 : \sigma$.

Proof. The three cases are shown successively.

Case $t_1 = (\lambda x.t)(u, y.r) \mapsto_{\beta\mathrm{h}} r\{y/t\{x/u\}\} = t_2$ where $r = \mathtt{H}\langle\!\langle y \rangle\!\rangle$. By lemma 3.30, there exist $\Gamma_r, \Gamma_{t\{x/u\}}$ and $\mathscr{N}$ such that $\Gamma_r; y : \mathscr{N} \Vdash r : \sigma$, $\Gamma_{t\{x/u\}} \Vdash t\{x/u\} : \mathscr{N}$ and $\Gamma = \Gamma_{t\{x/u\}} \uplus \Gamma_r$. By lemma 3.30 again, there exist Γ_t, Γ_u and $\mathscr{M}$ such that $\Gamma_t; x : \mathscr{M} \Vdash t : \mathscr{N}$, $\Gamma_u \Vdash u : \mathscr{M}$ and $\Gamma_{t\{x/u\}} = \Gamma_t \uplus \Gamma_u$. We thus have $\Gamma = \Gamma_t \uplus \Gamma_u \uplus \Gamma_r$. Let $\mathscr{N} = [\tau_i]_{i\in I}$. For each $i \in I$ there are derivations $\Gamma_t^i; x : \mathscr{M}_i \Vdash t : \tau_i$ with $\mathscr{M} = \sqcup_{i\in I}\mathscr{M}_i$ and $\Gamma_t = \uplus_{i\in I}\Gamma_t^i$. We can build the following derivation:

$$\dfrac{\dfrac{\left(\dfrac{\Gamma_t^i; x : \mathscr{M}_i \Vdash t : \tau_i}{\Gamma_t^i \vdash \lambda x.t : \mathscr{M}_i \to \tau_i}\,(\textsc{abs})\right)_{i\in I}}{\Gamma_t \vdash \lambda x.t : [\mathscr{M}_i \to \tau_i]_{i\in I}}\,(\textsc{many}) \qquad \Gamma_u \Vdash u : \mathscr{M} \quad \Gamma_r; y : \mathscr{N} \Vdash r : \sigma}{\Gamma \vdash (\lambda x.t)(u, y.r) : \sigma}\,(\textsc{app})$$

Case $t_1 = t(u, y.\lambda x.r) \mapsto_{\mathrm{p2}} \lambda x.t(u, y.r)$. Notice that σ is necessarily an arrow type $\mathscr{N} \to$

τ. We have the following derivation, with $\Gamma = \Sigma \uplus \Delta \uplus \Lambda$.

$$\dfrac{\dfrac{\Sigma \Vdash t : [\mathscr{M}_i \to \tau_i]_{i\in I} \qquad \Delta \Vdash u : \sqcup_{i\in I}\mathscr{M}_i \qquad \Lambda; y : [\tau_i]_{i\in I}; x : \mathscr{N} \Vdash r : \tau}{\Sigma \uplus \Delta \uplus \Lambda; x \vdash \mathscr{N} : t(u, y.r) : \tau}\,(\textsc{abs})}{\Sigma \uplus \Delta \uplus \Lambda \vdash \lambda x.t(u, y.r) : \mathscr{N} \to \tau}\,(\textsc{app})$$

By hypothesis, $x \notin \mathrm{fv}(t) \cup \mathrm{fv}(u)$, so that $x \notin \mathrm{dom}(\Sigma \uplus \Delta)$ by lemma 3.21. We can then build the following derivation:

$$\dfrac{\Sigma \Vdash t : [\mathscr{M}_i \to \tau_i]_{i\in I} \qquad \Delta \Vdash u : \sqcup_{i\in I}\mathscr{M}_i \qquad \dfrac{\Lambda; y : [\tau_i]_{i\in I}; x : \mathscr{N} \Vdash r : \tau}{\Lambda; y : [\tau_i]_{i\in I} \vdash \lambda x.r : \mathscr{N} \to \tau}\,(\textsc{abs})}{\Sigma \uplus \Delta \uplus \Lambda \vdash t(u, y.\lambda x.r) : \sigma}\,(\textsc{app})$$

Case $t_1 = t(u, x.r)(u', y.r') \mapsto_{\pi\mathrm{h}} t(u, x.r(u', y.r')) = t_2$ where $r' = \mathtt{H}\langle\!\langle y \rangle\!\rangle$. We have the following derivation:

$$\dfrac{\Phi_t \qquad \Phi_u \qquad \dfrac{\Phi_r \qquad \Phi_{u'} \qquad \Phi_{r'}}{\Gamma_r \uplus \Gamma_{u'} \uplus \Gamma_{r'}; x : [\tau_j]_{j\in J} \vdash r(u', y.r') : \sigma}\,(\textsc{app})}{\Gamma \vdash t(u, x.r(u', y.r')) : \sigma}\,(\textsc{app})$$

where $\Phi_t = \Gamma_t \Vdash t : [\mathscr{N}_j \to \tau_j]_{j\in J}$, $\Phi_u = \Gamma_u \Vdash u : \sqcup_{j\in J}\mathscr{N}_j$, $\Phi_r = \Gamma_r; x : [\tau_j]_{j\in J} \Vdash r : [\mathscr{M}_i \to \sigma_i]_{i\in I}$, $\Phi_{u'} = \Gamma_{u'} \Vdash u' : \sqcup_{i\in I}\mathscr{M}_i$, $\Phi_{r'} = \Gamma_{r'}; y : [\sigma_i]_{i\in I} \Vdash r' : \sigma$ and $\Gamma = \Gamma_t \uplus \Gamma_u \uplus \Gamma_r \uplus \Gamma_{u'} \uplus \Gamma_{r'}$.

By rule (MANY), there are derivations $\Phi_r^i = \Gamma_r^i; x : [\tau_j]_{j\in J_i} \Vdash r : \mathscr{M}_i \to \sigma_i$, where $\Gamma_r = \uplus_{i\in I}\Gamma_r^i$ and $J = \uplus_{i\in I} J_i$. By lemma 3.24, there are derivations $\Phi_t^i = \Gamma_t^i \Vdash t : [\mathscr{N}_j \to \tau_j]_{j\in J_i}$ and $\Phi_u^i = \Gamma_u^i \Vdash u : \sqcup_{j\in J_i}\mathscr{N}_j$, where $\Gamma_t = \uplus_{i\in I}\Gamma_t^i$ and $\Gamma_u = \uplus_{i\in I}\Gamma_u^i$, for each $i \in I$. We can build the following derivation:

$$\dfrac{\dfrac{\left(\dfrac{\Phi_t^i \qquad \Phi_u^i \qquad \Phi_r^i}{\Gamma_t^i \uplus \Gamma_u^i \uplus \Gamma_r^i \vdash t(u, x.r) : \mathscr{M}_i \to \sigma_i}\,(\textsc{app})\right)_{i\in I}}{\Gamma_t \uplus \Gamma_u \uplus \Gamma_r \vdash t(u, x.r) : [\mathscr{M}_i \to \sigma_i]_{i\in I}}\,(\textsc{many}) \qquad \Phi_{u'} \qquad \Phi_{r'}}{\Gamma \vdash t(u, x.r)(u', y.r') : \sigma}\,(\textsc{app})$$

□

We prove the subject expansion lemma by induction on the reduction step, using SE for βh and πh in the root case. Notice that the statement is about full djn reduction, which is useful in the proof of forthcoming theorem 3.37.

Lemma 3.32 (Subject expansion for $\cap N$). *If* $\Gamma \Vdash_{\cap N} t_2 : \sigma$ *and* $t_1 \to_{\mathrm{djn}} t_2$, *then* $\Gamma \Vdash_{\cap N} t_1 : \sigma$.

Proof. By induction on $t_1 \longrightarrow_{\mathrm{djn}} t_2$.

Case $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u, y.r) \mapsto_{\mathrm{dB}} r\{y/\mathtt{D}\langle t\{x/u\}\rangle\} = t_2$. Let $t_3 = \lambda x.\mathtt{D}\langle t\rangle(u, y.r)$. We have that $t_1 \mapsto^*_{\mathrm{p2}} t_3 \mapsto_{\beta\mathrm{h}} r\{y/\{\langle x\rangle/u\}\mathtt{D}t\} = t'$. Notice that $t' = t_2$ since $x \notin \mathrm{fv}(\mathtt{D})$. By multiple applications of lemma 3.31, we have $\Gamma \Vdash t_3 : \sigma$ and then $\Gamma \Vdash t_1 : \sigma$.

Case $t_1 = \lambda x.t \longrightarrow_{\mathrm{djn}} \lambda x.t' = t_2$, where $t \longrightarrow_{\mathrm{djn}} t'$. By hypothesis, we have $\sigma = \mathscr{M} \longrightarrow \tau$ and $\Gamma; x : \mathscr{M} \Vdash t' : \sigma$. By the *i.h.* we have $\Gamma; x : \mathscr{M} \Vdash t : \tau$. We use rule (ABS) to build a derivation of t_1.

Case $t_1 = t(u, y.r)$ and the reduction is internal. The derivation of t_2 ends with an (APP)-rule with premises: $\Gamma_t \Vdash t : [\mathscr{M}_i \longrightarrow \tau_i]_{i\in I}$, $\Gamma_u \Vdash u : \sqcup_{i\in I}\mathscr{M}_i$ and $\Gamma_r; y : [\tau_i]_{i\in I} \Vdash r : \sigma$. There are several cases:

Subcase $t_1 = t'(u, y.r) \longrightarrow_{\mathrm{djn}} t(u, y.r) = t_2$, where $t' \longrightarrow_{\mathrm{djn}} t$. By the *i.h.*, $\Gamma_t \Vdash t' : [\mathscr{M}_i \longrightarrow \tau_i]_{i\in I}$.

Subcase $t_1 = t(u', y.r) \longrightarrow_{\mathrm{djn}} t(u, y.r) = t_2$, where $u' \longrightarrow_{\mathrm{djn}} u$. By the *i.h.*, $\Gamma_u \Vdash u' : \sqcup_{i\in I}\mathscr{M}_i$.

Subcase $t_1 = t(u, y.r') \longrightarrow_{\mathrm{djn}} t(u, y.r) = t_2$, where $r' \longrightarrow_{\mathrm{djn}} r$. By the *i.h.*, $\Gamma_r; y : [\tau_i]_{i\in I} \Vdash r' : \sigma$. □

The other component of the completeness proof is the fact that sn-nfs are typable. For this, we define a notation for an arrow type made of only empty multitype, except for the last one.

Definition 3.33. $\sigma^k = \begin{cases} \sigma & \text{if } k = 0 \\ [\,] \longrightarrow \sigma^{k-1} & \text{otherwise.} \end{cases}$

We first show that neutral normal terms are typable, then general normal terms.

Lemma 3.34 (Typing neutral normal terms). *For any neutral term* $\mathtt{G}\langle\!\langle x\rangle\!\rangle$ *and any type* σ*, there exists* $k \geq 0$ *such that* $x : [\sigma^k] \Vdash_{\cap N} \mathtt{G}\langle\!\langle x\rangle\!\rangle : \sigma$.

Proof. By induction on $\mathtt{G}$.

Case $\mathtt{G} = \Diamond$. We get $x : [\sigma^0] \Vdash x : \sigma$ by rule (VAR).

Case $\mathtt{G} = \mathtt{G}''(u, y.\mathtt{G}'\langle\!\langle y\rangle\!\rangle)$. By the *i.h.* on $\mathtt{G}'$ there are $k_y \geq 0$ and a derivation $y : [\sigma^{k_y}] \Vdash_{\cap N} \mathtt{G}'\langle\!\langle y\rangle\!\rangle : \sigma$. By the *i.h.* on $\mathtt{G}''$ and then rule (MANY), we also have $k_x \geq 0$ and a derivation $x : [(\sigma^{k_y+1})^{k_x}] \Vdash \mathtt{G}''\langle\!\langle x\rangle\!\rangle : [\sigma^{k_y+1}]$. We conclude by setting $k = k_x + k_y + 1$ since $(\sigma^{k_y+1})^{k_x} = \sigma^{k_x+k_y+1}$.

$$\frac{x : [\sigma^k] \Vdash \mathtt{G}''\langle\!\langle x\rangle\!\rangle : [\sigma^{k_y+1}] \qquad \overline{\varnothing \vdash u : [\,]} \qquad y : [\sigma^{k_y}] \Vdash \mathtt{G}'\langle\!\langle y\rangle\!\rangle : \sigma}{x : [\sigma^k] \vdash \mathtt{G}''\langle\!\langle x\rangle\!\rangle(u, y.\mathtt{G}'\langle\!\langle y\rangle\!\rangle) : \sigma}\ (\mathrm{APP})$$

Case $t = t(u, y.\mathtt{H}'\langle\!\langle x\rangle\!\rangle)$ $(x \neq y)$. By the *i.h.* there is $k \geq 0$ and a derivation $x : [\sigma^k] \Vdash \mathtt{H}'\langle\!\langle x\rangle\!\rangle : \sigma$. We conclude as follows.

$$\frac{\overline{\emptyset \vdash t : [\,]} \qquad \overline{\emptyset \vdash u : [\,]} \qquad x : [\sigma^k] \Vdash \mathtt{H}'\langle\!\langle x\rangle\!\rangle : \sigma}{x : [\sigma^k] \vdash t(u, y.\mathtt{H}'\langle\!\langle x\rangle\!\rangle) : \sigma}\ (\textsc{app})$$

□

Lemma 3.35 (Typing sn-nfs). *Let $t \in \mathrm{NF}_{\mathrm{sn}}$. Then there exists σ such that*

(i) *If $t = \mathtt{H}\langle\!\langle x\rangle\!\rangle$ for some x, then there is τ such that $x : [\tau] \Vdash t : \sigma$.*

(ii) *Otherwise, $\emptyset \Vdash t : \sigma$.*

Proof. By induction on $t \in \mathrm{NF}_{\mathrm{sn}}$.

Case $t = x$. Then $t = \Diamond\langle\!\langle x\rangle\!\rangle$. We get $x : [\sigma] \Vdash x : \sigma$ by rule (VAR).

Case $t = \lambda y.s$ where $s \in \mathrm{NF}_{\mathrm{sn}}$. There are three possibilities.

Subcase $s = \mathtt{H}'\langle\!\langle x\rangle\!\rangle$ and $x \neq y$ (so that $t = \mathtt{H}\langle\!\langle x\rangle\!\rangle$ where $\mathtt{H} = \lambda y.\mathtt{H}'$). We need to prove (1)). By the *i.h.* (1) on s and x, there is τ such that $x : [\tau] \Vdash s : \sigma'$. We then get $x : [\tau] \Vdash \lambda y.s : [\,] \to \sigma'$ by rule (ABS). We conclude with $\sigma = [\,] \to \sigma'$.

Subcase $s = \mathtt{H}'\langle\!\langle y\rangle\!\rangle$. We need to prove (2). By the *i.h.* (1) on s and y, there is τ such that $y : [\tau] \Vdash s : \sigma'$. We then get $\emptyset \Vdash \lambda y.s : [\tau] \to \sigma'$ by rule (ABS). We conclude with $\sigma = [\tau] \to \sigma'$.

Subcase Otherwise. We need to prove (2). We apply *i.h.* (2) on s. We get a derivation $\emptyset \Vdash s : \sigma'$, and then $\emptyset \Vdash \lambda y.s : [\,] \to \sigma'$ by rule (ABS). We conclude with $\sigma = [\,] \to \sigma'$.

Case $t = \mathtt{G}\langle\!\langle x\rangle\!\rangle(u, y.r)$ where $r = \mathtt{H}\langle\!\langle y\rangle\!\rangle \in \mathrm{NF}_{\mathrm{sn}}$. We need to prove (1). By the *i.h.* on r there is a derivation $y : [\tau] \Vdash r : \sigma$. Applying lemma 3.34 on $\mathtt{G}\langle\!\langle x\rangle\!\rangle$ for the type $[\,] \to \tau$, and then rule (MANY), we have $k \geq 0$ such that $x : [([\,] \to \tau)^k] \vdash \mathtt{G}\langle\!\langle x\rangle\!\rangle : [[\,] \to \tau]$. We conclude as follows.

$$\frac{x : [([\,] \to \tau)^k] \vdash \mathtt{G}\langle\!\langle x\rangle\!\rangle : [[\,] \to \tau] \qquad \overline{\vdash u : [\,]} \qquad y : [\tau] \vdash r : \sigma}{x : [([\,] \to \tau)^k] \vdash \mathtt{G}\langle\!\langle x\rangle\!\rangle(u, y.r) : \sigma}$$

Case $t = s(u, y.r)$, where $r \neq \mathtt{H}\langle\!\langle y\rangle\!\rangle$ and $r \in \mathrm{NF}_{\mathrm{sn}}$. Let $\mathscr{M} = [\tau]$ in case 1, and $\mathscr{M} = [\,]$ in case (2). By *i.h.* there is a derivation $x : \mathscr{M} \vdash r : \sigma$. We conclude as follows.

$$\frac{\overline{\emptyset \vdash s : [\,]} \qquad \overline{\vdash u : [\,]} \qquad x : \mathscr{M} \vdash r : \sigma}{x : \mathscr{M} \vdash t(u, x.r) : \sigma}\ (\textsc{app})$$

□

Corollary 3.36 (Completeness for λJ_n). *Let $t \in \mathtt{T}_J$ be* sn*-normalizable. Then t is typable in system $\cap N$.*

Proof. By definition, the term t is reducible to a sn-normal form t'. By lemma 3.35, t' is typable. Subject expansion gives typability of t. □

Characterization of CbN Solvability

We can now derive the main theorem of this section.

Theorem 3.37 (CbN characterization). *Let $t \in \mathtt{T}_J$. Then t is CbN solvable iff t is $\cap N$-typable iff t is* sn*-normalizable.*

Proof. Typable $\Longrightarrow$ normalizable holds by corollary 3.29. Normalizable $\Longrightarrow$ solvable holds by property 3.19. For solvable $\Longrightarrow$ typable: take t solvable, so that there are contexts $\mathtt{H}, \mathtt{D}$ such that $\mathtt{H}\langle t\rangle \to^*_{\mathrm{djn}} \mathtt{D}\langle \mathtt{I}\rangle$. Since $\mathtt{D}\langle \mathtt{I}\rangle$ is $\cap N$-typable by lemma 3.35, and the system $\cap N$ satisfies subject expansion (lemma 3.32), then $\mathtt{H}\langle t\rangle$ is $\cap N$-typable, which implies t is $\cap N$-typable by lemma 3.23(i). □

3.4 Call-by-Value Solvability

We first give an operational characterization of CbV solvability and then a quantitative type system for it.

3.4.1 Potential Valuability

In CbN, a key element of the method to get the identity from a term plugged into a head context is to successively erase all the arguments, by replacing the head variable by a projection term $\mathtt{o}^n = \lambda x_n \dots x_0.x_0$. But in CbV, arguments which are not values cannot be erased.

For instance, let $t = x(\Omega, z.z)$. In CbN, we can substitute x by $\mathtt{o}^1$ to get

$$t\{x/\mathtt{o}^1\} = \mathtt{o}^1(\Omega, z.z) \to_{\mathrm{d}\beta} z.$$

In CbV, however, this is not possible since $t\{x/\mathtt{o}^1\}$ diverges.

$$\begin{aligned}\mathtt{o}^1(\Omega, z.z) &= (\lambda x_1 x_0.x_0)(\delta(\delta, y.y), z.z)\\ &\to_{\mathrm{d}\beta_{\mathrm{v}}} z\{z\backslash\backslash(\lambda x_0.x_0)\{x_1\backslash\backslash\delta(\delta, y.y)\}\} = \delta(\delta, y.z\{z/\lambda x_0.x_0\{x_1/y\}\}) = \delta(\delta, y.\lambda x_0.x_0)\\ &\to_{\mathrm{d}\beta_{\mathrm{v}}} \delta(\delta, y.\lambda x_0.x_0) \to_{\mathrm{d}\beta_{\mathrm{v}}} \dots\end{aligned}$$

The term t is only solvable in CbN. On the contrary, the term $x(\lambda y.\Omega, z.z)$ is solvable in both CbN and CbV because the argument $\lambda y.\Omega$ can be erased.

$$\mathtt{o}^1(\lambda y.\Omega, z.z) \to_{\mathrm{d}\beta_{\mathrm{v}}} z\{z\backslash\backslash\lambda x_0.x_0\{x_1\backslash\backslash\lambda y.\Omega\}\} = z\{z\backslash\backslash\lambda x_0.x_0\{x_1/\lambda y.\Omega\}\} = z\{z\backslash\backslash\lambda x_0.x_0\} = \lambda x_0.x_0$$

We will consider the set of *potentially valuable* terms [PR99]: terms which can be $\mathrm{d}\beta_{\mathrm{v}}$-reduced to a value under substitution, such as $\lambda y.\Omega$ and $x(\lambda y.\Omega, z.z)$. There are more potentially valuable terms than solvable terms, for instance $\lambda y.\Omega$ is not solvable. The potentially valuable terms are the terms that we will be able to erase when proving that a term is solvable.

Definition 3.38. A term t is **potentially valuable** iff there exist a distant context $\mathtt{D}$ and a value v such that $\mathtt{D}\langle t\rangle \to^{*}_{\mathrm{djv}} v$.

To reflect the definition of solvability, we do not use a list of (meta-level) substitutions here, but rather a distant context $\mathtt{D}$. This can be seen as a list of pending substitutions, that are fired with $\mathrm{d}\beta_{\mathrm{v}}$-steps. In particular, a substitution instance $t\{x_1/u_1\}\dots\{x_n/u_n\}$ can be expressed as $\mathtt{I}(u_n, x_n.\dots\mathtt{I}(u_1, x_1.t)\dots)$.

Interestingly, there is a (non-deterministic) reduction relation $\to_{\mathrm{ev}}$ such that the normalizing terms for $\to_{\mathrm{ev}}$ are exactly the potentially valuable terms (see theorem 3.66). It is in fact a *weak* reduction relation in which reduction can occur anywhere but below abstractions. We detail this result before tackling the CbV solving reduction.

Definition 3.39. Evaluation $\to_{\mathrm{ev}}$ is defined by the following rules:

$$\frac{t \mapsto_{\mathrm{d}\beta_{\mathrm{v}}} t'}{t \to_{\mathrm{ev}} t'} \qquad \frac{t \to_{\mathrm{ev}} t'}{t(u, y.r) \to_{\mathrm{ev}} t'(u, y.r)} \qquad \frac{u \to_{\mathrm{ev}} u'}{t(u, y.r) \to_{\mathrm{ev}} t(u', y.r)} \qquad \frac{r \to_{\mathrm{ev}} r'}{t(u, y.r) \to_{\mathrm{ev}} t(u, y.r')}$$

Lemma 3.40. *The following grammar characterizes* ev*-nfs:* $t \in \mathrm{NF}_{\mathrm{ev}}$ *iff* t *is in* ev*-nf.*

$$\begin{array}{lrcl} \textbf{\textit{(Valuable Neutral Normal Terms)}} & \mathrm{NE}_{\mathrm{ev}} & ::= & x \mid \mathrm{NE}_{\mathrm{ev}}(\mathrm{NF}_{\mathrm{ev}}, y.\,\mathrm{NE}_{\mathrm{ev}}) \\ \textbf{\textit{(Valuable Normal Terms)}} & \mathrm{NF}_{\mathrm{ev}} & ::= & x \mid \mathrm{NE}_{\mathrm{ev}}(\mathrm{NF}_{\mathrm{ev}}, y.\,\mathrm{NF}_{\mathrm{ev}}) \mid \lambda x.t \end{array}$$

Proof. For the left-to-right implication, we show the following stronger property:

(i) If $t \in \mathrm{NE}_{\mathrm{ev}}$, then t does not have an abstraction shape and t is in ev-nf.

(ii) If $t \in \mathrm{NF}_{\mathrm{ev}}$, then t is in ev-nf.

We proceed by induction on $\mathrm{NE}_{\mathrm{ev}}$ and $\mathrm{NF}_{\mathrm{ev}}$.

Case $t = x \in \mathrm{NF}_{\mathrm{ev}}$. Both statements are straightforward.

Case $t = \lambda x.t' \in \mathrm{NF}_{\mathrm{ev}}$. This is straightforward.

Case $t = s(u, y.r) \in \mathrm{NF}_{\mathrm{ev}}$ where $s \in \mathrm{NE}_{\mathrm{ev}}$, $u, r \in \mathrm{NF}_{\mathrm{ev}}$. By the *i.h.* (ii) s, u and r are in ev-nf (so that the contextual rules do not apply) and s does not have an abstraction shape (so that root reduction does not apply). Moreover, if $t \in \mathrm{NE}_{\mathrm{ev}}$, then in particular $r \in \mathrm{NE}_{\mathrm{ev}}$ and thus by the *i.h.* (i) r does not have an abstraction shape, so that t does not have this shape either.

For the right-to-left implication, we show the following stronger property:

(i) If t is in ev-nf and does not have an abstraction shape, then $t \in \mathrm{NE}_{\mathrm{ev}}$.

(ii) If t is in ev-nf, then $t \in \mathrm{NF}_{\mathrm{ev}}$.

We proceed by induction on t.

Case $t = x$. Then $t \in \mathrm{NE}_{\mathrm{ev}} \subseteq \mathrm{NF}_{\mathrm{ev}}$.

Case $t = \lambda x.t'$. Then $t \in \mathrm{NF}_{\mathrm{ev}}$ and the statement (i) does not apply.

Case $t = s(u, y.r)$. By hypothesis s, u and r are in ev-nf (otherwise the whole term would reduce). By the *i.h.* (ii), $u, r \in \mathrm{NF}_{\mathrm{ev}}$. Moreover, s does not have an abstraction shape (otherwise the whole term would ev-reduce at the root). By the *i.h.* (i), $s \in \mathrm{NE}_{\mathrm{ev}}$ and thus $t \in \mathrm{NF}_{\mathrm{ev}}$. Moreover, if t does not have an abstraction shape, then in particular r does not have an abstraction shape, so that by the *i.h.* (i) $r \in \mathrm{NE}_{\mathrm{ev}}$ and thus $t \in \mathrm{NE}_{\mathrm{ev}}$. □

We now show the main property: that ev-normalizable terms are potentially valuable. The converse is obtained in theorem 3.66. The next lemma resembles lemma 3.16 for CbN. We want to prove that a term $t \in$ ev-nf can be βv-reduced to a value with some well-chosen substitutions: $t\{x_1/\mathtt{o}^{n_1}\}\dots\{x_m/\mathtt{o}^{n_m}\} \to^*_{\beta\mathrm{v}} v$. Once again, this lemma is not on the general djv relation, but only βv (without distance): this allows us to use in the characterizations of potentially valuability for both the distant and the original calculus.

An important point to notice is that we do not only substitute one variable (the head variable in CbN), but the whole set of free variables of n. Why is it necessary? Take for instance $t = y_1(y_2(\mathtt{I}, x.x), z.z)$. In CbN, we would simply replace y_1 by $\mathtt{o}^1$, which would erase the argument $y_2(\mathtt{I}, x.x)$.

$$\mathtt{o}^1(y_2(\mathtt{I}, x.x), z.z) \to_\beta z\{z/(\lambda x_0.x_0)\{x_1/y_2(\mathtt{I}, x.x)\}\} = \lambda x_0.x_0$$

In CbV though, the argument needs to be a value to be erased. If we do the same substitution, we instead have:

$$\mathtt{o}^1(y_2(\mathtt{I}, x.x), z.z) \to_{\beta\mathrm{v}} z\{z\backslash\backslash(\lambda x_0.x_0)\{x_1\backslash\backslash y_2(\mathtt{I}, x.x)\}\} = y_2(\mathtt{I}, x.\lambda x_0.x_0).$$

We need to substitute y_2 also with a value such as $\mathtt{o}^0$. Then

$$\mathtt{o}^0(\mathtt{I}, x.\lambda x_0.x_0) \to_{\beta\mathrm{v}} (\lambda x_0.x_0)\{x\backslash\backslash x_0\{x_0\backslash\backslash \mathtt{I}\}\} = \lambda x_0.x_0.$$

Lemma 3.41. *For all $t \in \mathrm{NF}_{\mathrm{ev}}$ with $\mathrm{fv}(t) \subseteq \{x_1, \dots, x_m\}$, there exists $h \geq |t|_@$ such that for all $n_1, \dots, n_m \geq h$ there exists a value v such that $t\{x_1/\mathtt{o}^{n_1}\}\dots\{x_m/\mathtt{o}^{n_m}\} \to^*_{\beta\mathrm{v}} v$. If $t \in \mathrm{NE}_{\mathrm{ev}}$ with $\mathrm{hv}(t) = x_i$ (necessarily free), then $v = \mathtt{o}^{n_i - |t|_@}$.*

Proof. By induction on $t \in \mathrm{NF}_{\mathrm{ev}}$.

Case t is a variable. Thus $t = x_i \in \{x_1, \dots, x_m\}$. We take $h = 0 = |t|_@$ and for any $n_1, \dots, n_m \geq 0$ we have $t\{x_1/\mathtt{o}^{n_1}\}\dots\{x_m/\mathtt{o}^{n_m}\} = \mathtt{o}^{n_i - |x|_@} = \mathtt{o}^{n_i}$ which is a value.

Case $t = \lambda x.s$. Notice that $t \notin \mathrm{NE}_{\mathrm{ev}}$. We suppose w.l.o.g that $x \notin \{x_1, \dots, x_m\}$. We take $h = |s|_@$ and for any $n_1, \dots, n_m \geq h$ we conclude with an empty reduction since

$t\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\} = \lambda x.s\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}$ is a value.

Case $t = s(u, y.r)$, where $u, r \in \mathrm{NF}_{\mathrm{ev}}$ and $s \in \mathrm{NE}_{\mathrm{ev}}$. We suppose without loss of generality that $y \notin \{x_1, \dots, x_m\}$. Thus $\mathrm{fv}(r) \subseteq \{y, x_1, \dots, x_m\}$. Let $x_j = \mathrm{hv}(s)$ for some $1 \leq j \leq m$. By the *i.h.*:

1. There is $h_s \geq |s|_@$ s.t. for all $n_1^s, \dots, n_m^s \geq h_s$ we have $s\{x_1/\mathsf{o}^{n_1^s}\}\dots\{x_m/\mathsf{o}^{n_m^s}\} \to_{\beta\mathrm{v}}^* \mathsf{o}^{n_j^s - |s|_@}$.
2. There is $h_u \geq |u|_@$ such that for all $n_1^u, \dots, n_m^u \geq h_u$ there is a value v' such that $u\{x_1/\mathsf{o}^{n_1^u}\}\dots\{x_m/\mathsf{o}^{n_m^u}\} \to_{\beta\mathrm{v}}^* v'$.
3. There is $h_r \geq |r|_@$ such that for all $n_y, n_1^r, \dots, n_m^r \geq h_r$ there is a value v such that $r\{x_1/\mathsf{o}^{n_1^r}\}\dots\{x_m/\mathsf{o}^{n_m^r}\}\{y/\mathsf{o}^{n_y}\} \to_{\beta\mathrm{v}}^* v$.

We take $h = \max(h_s + h_r + 1, h_u) \geq |t|_@$ and we consider any $n_1, \dots, n_m \geq h$.

1. We have $h \geq h_s + h_r + 1$ and thus $n_1, \dots, n_m \geq h$ implies in particular $n_1, \dots, n_m \geq h_s$. This gives $s\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\} \to_{\beta\mathrm{v}}^* \mathsf{o}^{n_j - |s|_@}$ by the *i.h.* (1).
2. We have $h \geq h_u$ and thus $n_1, \dots, n_m \geq h$ implies in particular $n_1, \dots, n_m \geq h_u$. This gives $u\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\} \to_{\beta\mathrm{v}}^* v'$ by the *i.h.* (2).
3. We have $h \geq h_r + 1 > h_r$ and thus $n_1, \dots, n_m \geq h$ implies in particular $n_1, \dots, n_m \geq h_r + h_s + 1 \geq h_r + |s|_@ + 1 > h_r$. It gives $r\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}\{y/\mathsf{o}^{n_j - |s|_@ - 1}\} \to_{\beta\mathrm{v}}^* v$ by the *i.h.* (3).

Using the *i.h.*, we reduce as follows.

$$\begin{aligned} t\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\} &\to_{\beta\mathrm{v}}^* \mathsf{o}^{n_j - |s|_@}(v', y.r\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}) \\ &\to_{\beta\mathrm{v}} r\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}\{y/\mathsf{o}^{n_j - |s|_@ - 1}\} \\ &\to_{\beta\mathrm{v}}^* v \end{aligned}$$

We consider the particular case where $r \in \mathrm{NE}_{\mathrm{ev}}$. If $\mathrm{hv}(r) = x_i \neq y$ for some $1 \leq i \leq m$, then $\mathrm{hv}(t) = x_i$ and $|t|_@ = |r|_@$. We conclude by the *i.h.* (3) which gives $r\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}\{y/\mathsf{o}^{n_j - |s|_@ - 1}\} \to_{\beta\mathrm{v}}^* \mathsf{o}^{n_i - |r|_@}$. Otherwise, we have $\mathrm{hv}(r) = y$, $\mathrm{hv}(t) = \mathrm{hv}(s) = x_j$ for some $1 \leq i \leq m$ and $|t|_@ = |s|_@ + |r|_@ + 1$. The *i.h.* (3) gives $r\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}\{y/\mathsf{o}^{n_j - |s|_@ - 1}\} \to_{\beta\mathrm{v}}^* \mathsf{o}^{n_j - |s|_@ - 1 - |r|_@} = \mathsf{o}^{n_j - |t|_@}$. □

Lemma 3.42. *Let t be an* ev*-normalizable term. Then t is potentially valuable.*

Proof. Since t is ev-normalizable, then there is a ev-normal term t' such that $t \to_{\mathrm{ev}}^* t'$. Therefore $t' \in \mathrm{NF}_{\mathrm{ev}}$ by lemma 3.40. Let $\mathrm{fv}(t) = \{x_1, \dots, x_m\}$, so that $\mathrm{fv}(t') \subseteq \{x_1, \dots, x_m\}$. By lemma 3.41, there is $h \geq |t'|_@$ such that $t'\{x_1/\mathsf{o}^h\}\dots\{x_m/\mathsf{o}^h\} \to_{\beta\mathrm{v}}^* v$ for some value v.

Consider $\mathtt{D} = \mathtt{I}(\mathtt{o}^h, x_1.\mathtt{I}(\mathtt{o}^h, x_2. \dots \mathtt{I}(\mathtt{o}^h, x_m.\Diamond) \dots))$. Then,

$$\begin{aligned}
\mathtt{D}\langle t\rangle &\to^*_{\mathrm{ev}} \mathtt{I}(\mathtt{o}^h, x_1. \dots \mathtt{I}(\mathtt{o}^h, x_m.t')) \\
&\to_{\beta \mathrm{v}} \mathtt{I}(\mathtt{o}^h, x_2. \dots \mathtt{I}(\mathtt{o}^h, x_m.t\{x_1/\mathtt{o}^h\}') \dots) \\
&\to^*_{\beta \mathrm{v}} t'\{x_1/\mathtt{o}^h\} \dots \{x_m/\mathtt{o}^h\} \\
&\to^*_{\beta \mathrm{v}} v \text{ (by lemma 3.41)}
\end{aligned}$$

As a consequence, $\mathtt{D}\langle t\rangle \to^*_{\mathrm{djv}} v$. □

Example 3.43. Take again $t = y_1(\mathtt{I}, z_1.x)(y_2(\mathtt{I}, z_2.z_2), z_3.\lambda y.z_3)$ from example 3.17, which is also in $\mathrm{NF}_{\mathrm{ev}}$. We take $\mathtt{D} = \mathtt{I}(\mathtt{o}^1, y_1.\mathtt{o}^1(\mathtt{I}, x.\mathtt{o}^1(\mathtt{I}, y.\Diamond)))$.

$$\begin{aligned}
\mathtt{D}\langle t\rangle &\to^3_{\mathrm{djv}} \mathtt{o}^1(\mathtt{I}, z_1.\mathtt{o}^1)(\mathtt{o}^1(\mathtt{I}, z_2.z_2), z_3.\lambda y.z_3) \\
&\to_{\mathrm{djv}} \mathtt{o}^1(\mathtt{o}^1(\mathtt{I}, z_2.z_2), z_3.\lambda y.z_3) \\
&\to_{\mathrm{djv}} \mathtt{o}^1(\mathtt{I}, z_2.\lambda y.\lambda x_0.x_0) \\
&\to_{\mathrm{djv}} \lambda y.\lambda x_0.x_0
\end{aligned}$$

3.4.2 Operational Characterization of CbV Solvability

We are now ready to build the solving reduction on top of evaluation.

Definition 3.44. The **CbV solving reduction relation** $\to_{\mathrm{sv}}$ is defined as follows:

$$\frac{t \mapsto_{\mathrm{d}\beta_{\mathrm{v}}} t'}{t \to_{\mathrm{sv}} t'} \qquad \frac{t \to_{\mathrm{sv}} t'}{\lambda x.t \to_{\mathrm{sv}} \lambda x.t'}$$

$$\frac{t \to_{\mathrm{ev}} t'}{t(u, x.r) \to_{\mathrm{sv}} t'(u, x.r)} \qquad \frac{u \to_{\mathrm{ev}} u'}{t(u, x.r) \to_{\mathrm{sv}} t(u', x.r)} \qquad \frac{r \to_{\mathrm{sv}} r'}{t(u, x.r) \to_{\mathrm{sv}} t(u, x.r')}$$

An equivalent formulation can be given by a set of inductive rules identical to CbN head reduction, but using evaluation $\to_{\mathrm{ev}}$ as a base case. Thus, the CbV solving relation is more restrictive than the CbN one from the point of view of normalization, as it diverges on more term than the CbN solving relation $\to_{\mathrm{sn}}$.

To normalize, reduction must not only terminate under head contexts, but the subterms t and u in an application $t(u, x.r)$ must be ev-normalizable too. Semantically, this reflects the fact that for a term to be CbV solvable, the subterms t and u in the applications must be potentially valuable. With these rules, we make sure that in an application $t(u, x.r)$, the subterms t and u are ev-normalizable, and thus potentially valuable. In case there is a divergent term in u or t, the solving reduction will diverge.

For instance, the term $y(\Omega, z.\mathtt{I})$ loops because $\Omega \to_{\mathrm{ev}} \Omega$. However, the term $y(\lambda x.\Omega, z.\mathtt{I})$ does not reduce since $\lambda x.\Omega \not\to_{\mathrm{ev}}$. Finally, $(\lambda z_1.\lambda z_2.\Omega)(x, y.\mathtt{I}) \to_{\mathrm{sv}} \mathtt{I}$.

Lemma 3.45. *Let us consider the following grammar:*

$$\textbf{\textit{(CbV Solving Normal Terms)}} \quad \mathrm{NF}_{\mathrm{sv}} ::= x \mid \lambda x.\,\mathrm{NF}_{\mathrm{sv}} \mid \mathrm{NE}_{\mathrm{ev}}(\mathrm{NF}_{\mathrm{ev}}, y.\,\mathrm{NF}_{\mathrm{sv}})$$

Then, $t \in \mathrm{NF}_{\mathrm{sv}}$ iff t is in sv*-normal form. Notice that $\mathrm{NF}_{\mathrm{sv}} \subset \mathrm{NF}_{\mathrm{ev}}$.*

Proof. For the left-to-right implication, we proceed by induction on $\mathrm{NF}_{\mathrm{sv}}$. We proceed by induction on $\mathrm{NF}_{\mathrm{sv}}$.

Case $t = x \in \mathrm{NF}_{\mathrm{sv}}$. The statement is straightforward.

Case $t = \lambda x.t' \in \mathrm{NF}_{\mathrm{sv}}$. Then $t' \in \mathrm{NF}_{\mathrm{sv}}$, so that t' is in sv-nf by the *i.h.*, and thus t is in sv-nf by definition.

Case $t = s(u, y.r) \in \mathrm{NF}_{\mathrm{sv}}$. Then $s \in \mathrm{NE}_{\mathrm{ev}}$, $u \in \mathrm{NF}_{\mathrm{ev}}$ and $r \in \mathrm{NF}_{\mathrm{sv}}$. Since s is neutral normal, it does not have an abstraction shape, so that there is no root redex. Using the *i.h.* and lemma 3.40, we have t is in sv-nf.

For the right-to-left implication, we proceed by induction on t.

Case $t = x$. Then $t \in \mathrm{NF}_{\mathrm{sv}}$ holds trivially.

Case $t = \lambda x.t'$. Then t' is in sv-nf, which implies by the *i.h.* that $t' \in \mathrm{NF}_{\mathrm{sv}}$, thus $t \in \mathrm{NF}_{\mathrm{sv}}$.

Case $t = s(u, y.r)$. By hypothesis s is ev-normal and not an abstraction because t would be a root $\mathrm{d}\beta_{\mathrm{v}}$-redex, so that $s \in \mathrm{NE}_{\mathrm{ev}}$ by lemma 3.40. By hypothesis again u is in ev-nf. lemma 3.40 then gives $u \in \mathrm{NF}_{\mathrm{ev}}$. Finally, r is in sv-nf. The *i.h.* then gives $r \in \mathrm{NF}_{\mathrm{sv}}$. We thus conclude $t \in \mathrm{NF}_{\mathrm{sv}}$. □

As before, to prove the main property that sv-normalizable terms are solvable, we use an intermediate lemma to reduce sv-nfs to values. Like in lemma 3.41, we want to assign a value to every free variable of the term under consideration by substitution, as well as to the variables bound by abstractions by applying a series of arguments to the term.

Lemma 3.46. *For all $t \in \mathrm{NF}_{\mathrm{sv}}$ with $\mathrm{fv}(t) \subseteq \{x_1, \dots, x_m\}$, there exist $h \geq |t|_{@}, k \geq 0$ such that for all $n_1, \dots, n_{m+k} \geq h$ there exists $n \geq 0$ such that*

$$t\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}(\mathsf{o}^{n_{m+1}}, \dots, \mathsf{o}^{n_{m+k}}, z.z) \to_{\beta\mathrm{v}}^{*} \mathsf{o}^{n}.$$

Proof. By induction on $t \in \mathrm{NF}_{\mathrm{sv}}$.

Case t is a variable, thus $t = x_i$. We take $h = 0 = |x_i|_{@}, k = 0$ so that for all $n_1, \dots, n_m \geq 0$ we have $t\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\} = \mathsf{o}^{n_i}$. We let $n = n_i \geq 0$ and we conclude.

Case $t = \lambda x.s$ with $s \in \mathrm{NF}_{\mathrm{sv}}$. We suppose w.l.o.g that $x \notin \{x_1, \dots, x_m\}$. Then, $\mathrm{fv}(s) \subseteq \{x, x_1, \dots, x_m\}$. By the *i.h.*, there exist $h' \geq |s|_{@} = |t|_{@}, k' \geq 0$ such that for all

$n', n_1, \dots, n_{m+k} \geq h'$ there exists $n \geq 0$ such that

$$s\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}\{x/\mathsf{o}^{n'}\}(\mathsf{o}^{n_{m+1}}, \dots, \mathsf{o}^{n_{m+k'}}, z.z) \to_{\beta\mathrm{v}}^{*} \mathsf{o}^{n}.$$

Taking $h = h'$ and $k = k' + 1$ we have:

$$\begin{aligned}
&t\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}(\mathsf{o}^{n'}, \mathsf{o}^{n_{m+1}}, \dots, \mathsf{o}^{n_{m+k'}}, z.z)\\
&= \lambda x.s\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}(\mathsf{o}^{n'}, \mathsf{o}^{n_{m+1}}, \dots, \mathsf{o}^{n_{m+k'}}, z.z)\\
&\to_{\beta\mathrm{v}} s\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}\{x\backslash\!\backslash\mathsf{o}^{n'}\}(\mathsf{o}^{n_{m+1}}, \dots, \mathsf{o}^{n_{m+k'}}, z.z)\\
&= s\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}\{x/\mathsf{o}^{n'}\}(\mathsf{o}^{n_{m+1}}, \dots, \mathsf{o}^{n_{m+k'}}, z.z) \to_{\beta\mathrm{v}}^{*} \mathsf{o}^{n} \text{ (by the } \textit{i.h.}\text{)}
\end{aligned}$$

Case $t = s(u, y.r)$ with $s \in \mathrm{NE}_{\mathrm{ev}}$, $u \in \mathrm{NF}_{\mathrm{ev}}$ and $r \in \mathrm{NF}_{\mathrm{sv}}$. We suppose without loss of generality that $y \notin \{x_1, \dots, x_m\}$. Thus $\mathrm{fv}(r) \subseteq \{y, x_1, \dots, x_m\}$. Let $x_j = \mathrm{hv}(s)$ for some $1 \leq j \leq m$. By lemma 3.41 and the *i.h.* respectively:

1. There is $h_s \geq |s|_@$ such that for all $n_1^s, \dots, n_m^s \geq h_s$ we have $s\{x_1/\mathsf{o}^{n_1^s}\}\dots\{x_m/\mathsf{o}^{n_m^s}\} \to_{\beta\mathrm{v}}^{*} \mathsf{o}^{n_j^s - |s|_@}$.
2. There is $h_u \geq |u|_@$ such that for all $n_1^u, \dots, n_m^u \geq h_u$ there is a value v such that $u\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\} \to_{\beta\mathrm{v}}^{*} v$.
3. There are $h_r \geq |r|_@, k' \geq 0$ such that for all $n_y, n_1^r, \dots, n_{m+k'}^r \geq h_r$ there is $n \geq 0$ such that $r\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}\{y/\mathsf{o}^{n_y}\}(\mathsf{o}^{n_{m+1}}, \dots, \mathsf{o}^{n_{m+k'}}, z.z) \to_{\beta\mathrm{v}}^{*} \mathsf{o}^{n}$.

We take $h = \max(h_s + h_r + 1, h_u) \geq |t|_@$ and we consider any $n_1, \dots, n_m \geq h$.

1. We have $h \geq h_s + h_r + 1$ and thus $n_1, \dots, n_m \geq h$ implies in particular $n_1, \dots, n_m \geq h_s$. This gives $s\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\} \to_{\beta\mathrm{v}}^{*} \mathsf{o}^{n_j - |s|_@}$ by (1).
2. We have $h \geq h_u$ and thus $n_1, \dots, n_m \geq h$ implies in particular $n_1, \dots, n_m \geq h_u$. This gives $u\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\} \to_{\mathrm{ev}}^{*} v$ by (2).
3. We have $h \geq h_r + 1 > h_r$ and thus $n_1, \dots, n_m \geq h$ implies in particular $n_1, \dots, n_m \geq h_r + h_s + 1 \geq h_r + |s|_@ + 1 > h_r$. This gives $n \geq 0$ such that by the *i.h.* (3) $r\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}\{y/\mathsf{o}^{n_j - |s|_@ - 1}\}(\mathsf{o}^{n_{m+1}}, \dots, \mathsf{o}^{n_{m+k'}}, z.z) \to_{\beta\mathrm{v}}^{*} \mathsf{o}^{n}$.

In summary, we reduce as follows:

$$\begin{aligned}
&t\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}(\mathsf{o}^{n_{m+1}}, \dots, \mathsf{o}^{n_{m+k}}, z.z)\\
&\to_{\beta\mathrm{v}}^{*} \mathsf{o}^{n_j - |s|_@}(v, y.r\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\})(\mathsf{o}^{n_{m+1}}, \dots, \mathsf{o}^{n_{m+k}}, z.z)\\
&\to_{\beta\mathrm{v}} r(\mathsf{o}^{n_{m+1}}, \dots, \mathsf{o}^{n_{m+k}}, z.z)\{x_1/\mathsf{o}^{n_1}\}\dots\{x_m/\mathsf{o}^{n_m}\}\{y/\mathsf{o}^{n_j - |s|_@ - 1}\}\\
&\to_{\beta\mathrm{v}}^{*} \mathsf{o}^{n} \text{ (by the } \textit{i.h.}\text{ (3))}
\end{aligned}$$

□

Lemma 3.47. *Let t be an* sv*-normalizable term. Then t is CbV solvable.*

Proof. Since t is sv-normalizable, then there is a sv-normal term t' such that $t \to^*_{\mathrm{sv}} t'$. Therefore $t' \in \mathrm{NF}_{\mathrm{sv}}$ by lemma 3.45. Let $\mathrm{fv}(t) = \{x_1, \dots, x_m\}$, so that $\mathrm{fv}(t') \subseteq \{x_1, \dots, x_m\}$. By lemma 3.16, there are $h, k \in \mathbb{N}$ such that for all $n_1, \dots, n_{m+k} \geq h$ there is $n \geq 0$ such that $t'\{x_1/\mathtt{o}^{n_1}\}\dots\{x_m/\mathtt{o}^{n_m}\}(\mathtt{o}^{n_{m+1}}, \dots, \mathtt{o}^{n_{m+k}}, z.z) \to^*_{\beta\mathrm{v}} \mathtt{o}^n$, which is also a djv-step. We take $n_1, \dots, n_{m+k} = h$. We can then write $(\mathtt{o}^h, \dots, \mathtt{o}^h, z.z)$ as $\overline{(\mathtt{o}^h, z.z)}^{k}$. Let

$$\mathtt{H} = \mathtt{I}(\mathtt{o}^h, x_m.\dots\mathtt{I}(\mathtt{o}^h, x_1.\Diamond)\dots)\overline{(\mathtt{o}^h, z.z)}^{k}\,\overline{(\mathtt{I}, z.z)}^{n}.$$

Then:

$$\mathtt{H}\langle t\rangle \to^*_{\mathrm{sv}} \mathtt{H}\langle t'\rangle \to^m_{\beta\mathrm{v}} t'\{x_1/\mathtt{o}^h\}\dots\{x_m/\mathtt{o}^h\}\overline{(\mathtt{o}^h, z.z)}^{k}\,\overline{(\mathtt{I}, z.z)}^{n} \to^*_{\beta\mathrm{v}} \mathtt{o}^n\overline{(\mathtt{I}, z.z)}^{n} \to^n_{\beta\mathrm{v}} \mathtt{I}$$

As a consequence, $\mathtt{H}\langle t\rangle \to_{\mathrm{djv}} \mathtt{I}$. □

Example 3.48. Take again the term $t = y_1(\mathtt{I}, z_1.x)(y_2(\mathtt{I}, z_2.z_2), z_3.\lambda y.z_3)$ from example 3.43. We take $\mathtt{H} = \mathtt{D}(\mathtt{I}, z.z)$, where $\mathtt{D} = \mathtt{I}(\mathtt{o}^1, y_1.\mathtt{o}^1(\mathtt{I}, x.\mathtt{o}^1(\mathtt{I}, y.\Diamond)))$ is the context from that example. Then,

$$\mathtt{H}\langle t\rangle \to^*_{\mathrm{djv}} (\lambda y.\lambda x_0.x_0)(\mathtt{I}, z.z) \to_{\mathrm{djv}} \lambda x_0.x_0 = \mathtt{I}.$$

3.4.3 Logical Characterization of CbV Solvability

We will now define a quantitative type system characterizing CbV solvability. The grammar of types is different from section 3.3.2, as multiset types are considered as types and in particular may also occur on the right hand-side of an arrow.

(Types)	σ, τ, ρ	::=	$a \in BTV \mid \mathscr{M} \mid \mathscr{M} \to \sigma$
(Multiset types)	$\mathscr{M}, \mathscr{N}$	::=	$[\sigma_i]_{i\in I}$ where I is a finite set

$$\frac{}{x : \mathscr{M} \vdash x : \mathscr{M}}\ (\textsc{var}) \qquad \frac{(\Gamma_i; x : \mathscr{M}_i \vdash t : \sigma_i)_{i\in I}}{\uplus_{i\in I}\Gamma_i \vdash \lambda x.t : [\mathscr{M}_i \to \sigma_i]_{i\in I}}\ (\textsc{abs})$$

$$\frac{\Gamma \vdash t : [\mathscr{M} \to \mathscr{N}] \qquad \Delta \vdash u : \mathscr{M} \qquad \Lambda; x : \mathscr{N} \vdash r : \sigma}{\Gamma \uplus \Delta \uplus \Lambda \vdash t(u, x.r) : \sigma}\ (\textsc{app})$$

Figure 3.2: System $\cap V$.

We use a unique type system $\cap V$, defined in figure 3.2, to characterize both potential valuability and solvability. The type system is inspired from the system of Bucciarelli, Kesner,

Ríos, and Viso [Buc+20] for the bang calculus. Again, we write $\Gamma \Vdash^{n}_{\cap V} t : \sigma$ if the sequent $\Gamma \vdash t : \sigma$ is derivable in this system with a derivation of size n (containing n occurrences of (APP)). This system is relevant.

Lemma 3.49 (Relevance). *If* $\Gamma \Vdash_{\cap V} t : \sigma$, *then* $\mathrm{dom}(\Gamma) \subseteq \mathrm{fv}(t)$.

Proof. By induction on the derivation.

Case the derivation ends with (VAR). Then $t = x$, $\sigma = \mathscr{M}$ and $\Gamma = x : \mathscr{M}$ and we have $x : \mathscr{M} \Vdash_{\cap V} x : \mathscr{M}$. We have $\mathrm{dom}(x : \mathscr{M}) \subseteq \{x\} = \mathrm{fv}(x)$.

Case the derivation ends with (ABS). Then $t = \lambda y.u$, $\Gamma = \uplus_{i\in I}\Gamma_i$, $\sigma = [\mathscr{M}_i \to \tau_i]_{i\in I}$ and the premises are of the form $\Delta_i \Vdash u : \tau_i$, with $\Delta_i = \Gamma_i; y : \mathscr{M}_i$. We have $\mathrm{dom}(\Gamma_i; y : \mathscr{M}_i) \subseteq_{i.h.} \mathrm{fv}(u)$. If $\mathrm{dom}(\Gamma_i; y : \mathscr{M}_i) = \mathrm{dom}(\Gamma_i) \cup \{y\}$, then we get $\mathrm{dom}(\Gamma_i) \subseteq \mathrm{fv}(u) \setminus \{y\} = \mathrm{fv}(\lambda y.u)$. If $\mathrm{dom}(\Gamma_i; y : \mathscr{M}_i) = \mathrm{dom}(\Gamma_i)$, then we get $\mathrm{dom}(\Gamma_i) \subseteq \mathrm{fv}(u)$ with $y \notin \mathrm{dom}(\Gamma_i)$ so that $\mathrm{dom}(\Gamma_i) \subseteq \mathrm{fv}(u) \setminus \{y\} = \mathrm{fv}(\lambda y.u)$ also holds. Then $\mathrm{dom}(\Gamma) = \bigcup_{i\in I} \mathrm{dom}(\Gamma_i) \subseteq \mathrm{fv}(\lambda y.u)$.

Case the derivation ends with (APP). then $t = s(u, x.r)$ and the premises are of the form $\Gamma_s \Vdash s : [\mathscr{M} \to \mathscr{N}]$, $\Gamma_u \Vdash u : \mathscr{M}$ and $\Gamma_r; x : \mathscr{N} \Vdash r : \sigma$ where $\Gamma = \Gamma_s \uplus \Gamma_u \uplus \Gamma_r$. The *i.h.* gives $\mathrm{dom}(\Gamma_s) \subseteq \mathrm{fv}(s)$, $\mathrm{dom}(\Gamma_u) \subseteq \mathrm{fv}(u)$, and $\mathrm{dom}(\Gamma_r; x : \mathscr{N}) \subseteq \mathrm{fv}(r)$. If $\mathrm{dom}(\Gamma_r; x : \mathscr{N}) = \mathrm{dom}(\Gamma_r) \cup \{x\}$, then we get $\mathrm{dom}(\Gamma_r) \subseteq \mathrm{fv}(r) \setminus \{y\}$, which implies $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma_s) \cup \mathrm{dom}(\Gamma_u) \cup \mathrm{dom}(\Gamma_r) \subseteq \mathrm{fv}(s) \cup \mathrm{fv}(u) \cup (\mathrm{fv}(r) \setminus \{y\}) = \mathrm{fv}(t)$. If $\mathrm{dom}(\Gamma_r; x : \mathscr{N}) = \mathrm{dom}(\Gamma_r)$, then we get $\mathrm{dom}(\Gamma_r) \subseteq \mathrm{fv}(r)$ and $y \notin \mathrm{dom}(\Gamma_r)$ implies $\mathrm{dom}(\Gamma_r) \subseteq \mathrm{fv}(r) \setminus \{y\}$ and thus $\mathrm{dom}(\Gamma) = \mathrm{dom}(\Gamma_s) \cup \mathrm{dom}(\Gamma_u) \cup \mathrm{dom}(\Gamma_r) \subseteq \mathrm{fv}(s) \cup \mathrm{fv}(u) \cup (\mathrm{fv}(r) \setminus \{y\}) = \mathrm{fv}(t)$. □

We will show that typability in $\cap V$ is equivalent to normalization of evaluation. To logically characterize solvable terms, we constrain typability to a particular set of types, where the empty multiset type cannot appear anymore on the right-hand sides of arrows. We take this idea from Accattoli and Guerrieri [AG22], where these types are called *solvable*. This restriction originates from [PR99], (using an idempotent intersection type system), where the types are called proper.

Definition 3.50 (Solvable types). A **solvable type** σ^{s} is not an empty multiset, and has no empty multiset on the right of an arrow. Formally,

$$\begin{array}{rrcl} \textbf{(Solvable types)} & \sigma^{\mathrm{s}}, \tau^{\mathrm{s}} & ::= & a \in BTV \mid \mathscr{M}^{\mathrm{s}} \mid \mathscr{M} \to \sigma^{\mathrm{s}} \\ \textbf{(Solvable multiset types)} & \mathscr{M}^{\mathrm{s}}, \mathscr{N}^{\mathrm{s}} & ::= & [\sigma^{\mathrm{s}}_i]_{i\in I} \quad \text{where } I \text{ is a non-empty finite set} \end{array}$$

Unlike CbN, where the empty multiset $[\,]$ is used to mark *untyped* subterms, being typable in CbV with $[\,]$ is equivalent to being potentially valuable. The unsolvable term $\lambda x.\Omega$, for instance, can be typed with $[\,]$ by rule (ABS) with I empty. But it cannot be typed with any other type. In particular not with a solvable one, as this would require Ω to be typable. Notice also that the terms t and u in rule (APP) must always be typed, at least with type $[\,]$. That is why the term $t = \Omega(y, z.\mathtt{I})$ of example 3.22, typable in $\cap N$, is not typable in $\cap V$.

Example 3.51. Take $t = (\lambda x.x)(x, y.\lambda z.\Omega)$. Even when typing it with $[\,]$, premises must be given for rule (APP), that is, the subterms x, x and $\lambda z.\Omega$ must be typed.

$$\dfrac{\dfrac{\dfrac{}{\vdash x : [\,]}\,(\text{VAR})}{\vdash \lambda x.x : [[\,] \to [\,]]}\,(\text{ABS}) \qquad \dfrac{}{\vdash x : [\,]}\,(\text{VAR}) \qquad \dfrac{}{\vdash \lambda z.\Omega : [\,]}\,(\text{ABS})}{\vdash (\lambda x.x)(x, y.\lambda z.\Omega) : [\,]}\,(\text{APP})$$

Lemma 3.52 (Split for values). *If $\Gamma \Vdash^{n}_{\cap V} v : \mathscr{M}$, then for any decomposition $\mathscr{M} = +_{i\in I}\mathscr{M}_i$ we have $\Gamma_i \Vdash^{n_i}_{\cap V} v : \mathscr{M}_i$ such that $\sum_{i\in I} n_i = n$ and $\uplus_{i\in I}\Gamma_i = \Gamma$.*

Proof. Straightforward by induction on the derivation. □

We now prove that terms typable in $\cap V$ are exactly those that are normalizable for the valuable reduction, and among them, those typable with a solvable type are the ones normalizing for the solvable reduction. The proof method is the same as for CbN (section 3.3.2), but the statements cover both reduction relations at the same time, since both use the same type system.

Soundness

Soundness follows the same scheme as used for CbN (no reducibility proof is needed): a weighted subject reduction property is used to show that typability implies normalization.

Since two kinds of substitution are used in CbV, there are two corresponding substitution lemmas, the one for left substitution relying on the first one for the usual right substitution.

Lemma 3.53 (Right substitution lemma). *If $\Gamma; x : \mathscr{M} \Vdash^{n}_{\cap V} t : \sigma$ and $\Delta \Vdash^{m}_{\cap V} v : \mathscr{M}$, then $\Gamma \uplus \Delta \Vdash^{n+m}_{\cap V} t\{x/v\} : \sigma$.*

Proof. By induction on t.

Case $t = x$. By hypothesis, $\Gamma = \emptyset$, $n = 0$ and $\sigma = \mathscr{M}$. We conclude with $\emptyset \uplus \Delta \Vdash^{0+m} x\{x/v\} : \sigma = \Delta \Vdash^{m} v : \mathscr{M}$.

Case $t = y \neq x$. By hypothesis, $\mathscr{M} = [\,]$ and $n = 0$. We necessarily have $\Delta \Vdash^{0} v : [\,]$. Moreover, v is either a variable or an abstraction, which implies $\Delta = \emptyset$ by using rule (VAR) or (ABS). We conclude with $\Gamma \uplus \emptyset \Vdash^{0+0} y\{x/v\} : \sigma = \Gamma; x : [\,] \Vdash^{0} y : \sigma$.

Case $t = \lambda y.u$ where $y \neq x$ and $y \notin \mathrm{fv}(v)$. By hypothesis we have $\Gamma_i; x : \mathscr{M}_i; y : \mathscr{N}_i \Vdash^{n_i} u : \tau_i$ for all $i \in I$, where $\sigma = [\mathscr{N}_i \to \tau_i]_{i\in I}$, $\Gamma = \uplus_{i\in I}\Gamma_i$, $\mathscr{M} = +_{i\in I}\mathscr{M}_i$ and $\sum_{i\in I} n_i = n$. By lemma 3.52 $\Delta_i \Vdash^{m_i} v : \mathscr{M}_i$ where $\sum_{i\in I} m_i = m$ and $\Delta = \uplus_{i\in I}\Delta_i$. By the *i.h.* $\Gamma_i \uplus \Delta_i; y : \mathscr{N}_i \Vdash^{n_i+m_i} u\{x/v\} : \tau_i$ for $i \in I$. By rule (ABS) and because $y \neq x$, we obtain $\Gamma \uplus \Delta \Vdash^{n+m} \lambda y.u\{x/v\} : [\mathscr{N}_i \to \tau_i]_{i\in I}$. We conclude because $\lambda y.u\{x/v\} = (\lambda y.u)\{x/v\}$.

Case $t = s(u, y.r)$ where $y \neq x$ and $y \notin \mathrm{fv}(v)$. By hypothesis, $\Gamma_1; x : \mathscr{M}_1 \Vdash^{n_1} s : [\mathscr{N} \to \mathscr{N}']$, $\Gamma_2; x : \mathscr{M}_2 \Vdash^{n_2} u : \mathscr{N}$ and $\Gamma_3; x : \mathscr{M}_3; y : \mathscr{N}' \Vdash^{n_3} r : \sigma$ where $\Gamma = \Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3$, $\mathscr{M} = \mathscr{M}_1 + \mathscr{M}_2 + \mathscr{M}_3$ and $n = n_1 + n_2 + n_3 + 1$. By lemma 3.52, $\Delta_i \Vdash^{m_i} v : \mathscr{M}_i$ $(i = 1, 2, 3)$ where $\Delta = \Delta_1 + \Delta_2 + \Delta_3$ and $m = m_1 + m_2 + m_3$. By the *i.h.*, $\Gamma_1 \uplus \Delta_1 \Vdash^{n_1+m_1} s\{x/v\} : [\mathscr{N} \to \mathscr{N}']$, $\Gamma_2 \uplus \Delta_2 \Vdash^{n_2+m_2} u\{x/v\} : \mathscr{N}$, and $\Gamma_3 \uplus \Delta_3; y : \mathscr{N}' \Vdash^{n_3+m_3} s\{x/v\} : \sigma$. We conclude using rule (APP), the fact that $(s(u, y.r))\{x/v\} = s\{x/v\}(u\{x/v\}, y.r\{x/v\})$ and $1 + \sum_{i=1}^{3}(n_i + m_i) = m + n$. □

Lemma 3.54 (Left substitution lemma). *If* $\Gamma; x : \mathscr{M} \Vdash^{n}_{\cap V} t : \sigma$ *and* $\Delta \Vdash^{m}_{\cap V} u : \mathscr{M}$, *then* $\Gamma \uplus \Delta \Vdash^{n+m}_{\cap V} t\{x \backslash\backslash u\} : \sigma$.

Proof. By induction on u. If u is a value, then $t\{x \backslash\backslash u\} = t\{x/u\}$ and we use lemma 3.53. Otherwise, $u = s(u', y.r)$ so that by definition, $t\{x \backslash\backslash u\} = s(u', y.t\{x \backslash\backslash r\})$. The typing derivation of u ends with an (APP)-rule. We have $\Delta_s \Vdash^{m_1} s : [\mathscr{N} \to \mathscr{N}']$, $\Delta_{u'} \Vdash^{m_2} u' : \mathscr{N}$ and $\Delta_r; y : \mathscr{N}' \Vdash^{m_3} r : \mathscr{M}$ where $\Delta = \Delta_s \uplus \Delta_{u'} \uplus \Delta_r$ and $m = m_1 + m_2 + m_3 + 1$. By the *i.h.*, $\Gamma \uplus \Delta_r; y : \mathscr{N}' \Vdash^{n+m_3} t\{x \backslash\backslash r\} : \sigma$. We conclude with rule (APP) and the fact that $n + m = n + m_1 + m_2 + m_3 + 1$. □

We separate the proof of subject reduction in the same way as in CbN, starting with the base cases for βv and π separately, so that this proof can be used for the original and the distant calculus.

Lemma 3.55. *Let* $\Gamma \Vdash^{n}_{\cap V} t_1 : \sigma$.

(i) *If* $t_1 \mapsto_{\beta \mathrm{v}} t_2$, *then* $\Gamma \Vdash^{n-1}_{\cap V} t_2 : \sigma$.

(ii) *If* $t_1 \mapsto_{\pi} t_2$, *then* $\Gamma \Vdash^{n}_{\cap V} t_2 : \sigma$.

Proof. The items are proved successively.

Case $t_1 = (\lambda x.t)(u, y.r) \mapsto_{\beta \mathrm{v}} r\{y \backslash\backslash t\{x \backslash\backslash u\}\} = t_2$. We have the following derivation:

$$\dfrac{\dfrac{\Gamma_t; x : \mathscr{M} \Vdash^{n_t} t : \mathscr{N}}{\Gamma_t \vdash \lambda x.t : [\mathscr{M} \to \mathscr{N}]}\,(\text{ABS}) \qquad \Gamma_u \Vdash^{n_u} u : \mathscr{M} \qquad \Gamma_r; y : \mathscr{N} \Vdash^{n_r} r : \sigma}{\Gamma \vdash (\lambda x.t)(u, y.r) : \sigma}\,(\text{APP})$$

Where $\Gamma = \Gamma_t \uplus \Gamma_u \uplus \Gamma_r$ and $n = n_t + n_u + n_r + 1$. By two applications of lemma 3.54, $\Gamma \Vdash^{n-1} r\{y \backslash\backslash t\{x \backslash\backslash u\}\} : \sigma$.

Case $t_1 = t(u, x.r)(u', y.r') \mapsto_{\pi} t(u, x.r(u', y.r')) = t_2$. We have the following derivation:

$$\dfrac{\dfrac{\Phi_t \qquad \Phi_u \qquad \Phi_r}{\Gamma_t \uplus \Gamma_u \uplus \Gamma_r \vdash t(u, x.r) : [\mathscr{M} \to \mathscr{N}]}\,(\text{APP}) \qquad \Phi_{u'} \qquad \Phi_{r'}}{\Gamma \vdash t(u, x.r)(u', y.r') : \sigma}\,(\text{APP})$$

and $\Phi_t = \Gamma_t \Vdash^{n_t} t : [\mathscr{M}' \to \mathscr{N}']$, $\Phi_u = \Gamma_u \Vdash^{n_u} u : \mathscr{M}'$, $\Phi_r = \Gamma_r; x : \mathscr{N}' \Vdash^{n_r} r : [\mathscr{M} \to \mathscr{N}]$, $\Phi_{u'} = \Gamma_{u'} \Vdash^{n_{u'}} u' : \mathscr{M}$ and $\Phi_{r'} = \Gamma_{r'}; y : \mathscr{N} \Vdash^{n_{r'}} r' : \sigma$, such that $\Gamma = \Gamma_t \uplus \Gamma_u \uplus \Gamma_r \uplus \Gamma_{u'} \uplus \Gamma_{r'}$ and $n = n_t + n_u + n_r + n_{u'} + n_{r'} + 2$. Using the fact that $x \notin \mathrm{fv}(u') \cup \mathrm{fv}(r')$ and the relevance lemma 3.49, we build the following derivation of the same size.

$$\dfrac{\Phi_t \qquad \Phi_u \qquad \dfrac{\Phi_r \qquad \Phi_{u'} \qquad \Phi_{r'}}{(\Gamma_r; x : \mathscr{N}') \uplus \Gamma_{u'} \uplus \Gamma_{r'} \vdash r(u', y.r') : \sigma}\,(\textsc{app})}{\Gamma \vdash t(u, x.r(u', y.r')) : \sigma}\,(\textsc{app})$$

□

Subject reduction holds for the whole reduction relation djv. In particular, the size of the proof strictly decreases for evaluation and for the solving relation.

Lemma 3.56 (Weighted subject reduction for $\cap V$). *Let* $\Gamma \Vdash^{n_1}_{\cap V} t_1 : \sigma$ *and* $t_1 \to_{\mathrm{djv}} t_2$. *Then* $\Gamma \Vdash^{n_2}_{\cap V} t_2 : \sigma$ *with* $n_1 \geq n_2$. *Moreover:*

(i) *If* $t_1 \to_{\mathrm{ev}} t_2$, *then* $n_1 > n_2$.

(ii) *If* $t_1 \to_{\mathrm{sv}} t_2$ *and* σ *is a solvable type, then* $n_1 > n_2$.

Proof. By induction on $t_1 \to_{\mathrm{djv}} t_2$ (resp. $t_1 \to_{\mathrm{ev}} t_2$, $t_1 \to_{\mathrm{sv}} t_2$).

Case $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u, y.r) \mapsto_{\mathrm{d}\beta_{\mathrm{v}}} \mathtt{D}\langle r\{y \backslash\backslash t\{x \backslash\backslash u\}\}\rangle = t_2$. This is the base case. We have $t_1 \mapsto^*_\pi \mathtt{D}\langle(\lambda x.t)(u, y.r)\rangle = t_3$ (simple induction on $\mathtt{D}$). Thus $\Gamma \Vdash^{n_1} t_3 : \sigma$ by lemma 3.55(ii) It is straightforward that $\Gamma' \Vdash^{n} (\lambda x.t)(u, y.r) : \sigma$ for some Γ' and some $n \leq n_1$. By lemma 3.55(i) $\Gamma' \Vdash^{n-1} r\{y \backslash\backslash t\{x \backslash\backslash u\}\} : \sigma$. Thus, $\Gamma \Vdash^{n_2} t_2 : \sigma$, where $n_2 = n_1 - 1$.

Case $t_1 = t(u, y.r)$ and the reduction is internal. The derivation of t_1 ends with an (app)-rule with premises: $\Gamma_t \Vdash^{n_t} t : [\mathscr{M} \to \mathscr{N}]$, $\Gamma_u \Vdash^{n_u} u : \mathscr{M}$ and $\Gamma_r; x : \mathscr{N} \Vdash^{n_r} r : \sigma$ such that $n_1 = 1 + n_t + n_u + n_r$. There are several subcases:

Subcase $t_1 = t(u, y.r) \to_{\mathrm{djv}} t'(u, y.r) = t_2$ where $t \to_{\mathrm{djv}} t'$. By the *i.h.*, $\Gamma_t \Vdash^{n_{t'}} t' : [\mathscr{M} \to \mathscr{N}]$ such that $n_t \geq n_{t'}$. We can build a derivation of size $n_1 \geq 1 + n_{t'} + n_u + n_r = n_2$.

Subcase $t_1 = t(u, y.r) \to_{\mathrm{djv}} t(u', y.r) = t_2$, where $u \to_{\mathrm{djv}} u'$. By the *i.h.*, $\Gamma_u \Vdash^{n_{u'}} u' : \mathscr{M}$ such that $n_u \geq n_{u'}$, so that $n_1 \geq n_2$.

Subcase $t_1 = t(u, y.r) \to_{\mathrm{djv}} t(u, y.r') = t_2$, where $r \to_{\mathrm{djv}} r'$. By the *i.h.*, $\Gamma_r; x : \mathscr{N} \Vdash^{n_{r'}} r' : \sigma$ such that $n_r \geq n_{r'}$, so that $n_1 \geq n_2$.

For each of these subcases:

1. If $t_1 \to_{\mathrm{ev}} t_2$ the *i.h.* (i) gives $n_t > n_{t'}$ (resp. $n_u > n_{u'}$, $n_r > n_{r'}$), so that we conclude $n_1 > n_2$.

2. If $t_1 \to_{\mathrm{sv}} t_2$ and σ a solvable type, either we are in the case $t \to_{\mathrm{ev}} t'$ or $u \to_{\mathrm{ev}} u'$ and by the previous point $n_t > n_{t'}$ (resp. $n_u > n_{u'}$), or we are in the case $r \to_{\mathrm{sv}} r'$ and by the *i.h.* (ii) $n_r > n_{r'}$. In both cases $n_1 > n_2$.

Case $t_1 = \lambda x.t \to_{\mathrm{djv}} \lambda x.t' = t_2$. By hypothesis, we have $\sigma = [\mathscr{M}_i \to \sigma_i]_{i\in I}$. If I is empty, then $\Gamma = [\,]$, $n_1 = 0$ and we have $\Gamma \Vdash^0 \lambda x.t' : [\,]$ by using the (ABS) rule with no premise, so that in particular $n_1 = n_2$.

Otherwise, we have $\Gamma_i; x : \mathscr{M}_i \Vdash^{n_i} t : \sigma_i$ for $i \in I$, where $\Gamma = \uplus_{i\in I}\Gamma_i$ and $n_1 = \sum_{i\in I} n_i$. By the *i.h.*, we have $\Gamma_i; x : \mathscr{M}_i \Vdash^{n_i'} t' : \sigma_i$ for $i \in I$ such that $n_i \geq n_i'$. We can build a derivation of size $n_2 = \sum_{i\in I} n_i' \leq \sum_{i\in I} n_i = n_1$. In particular,

1. This step is never an valuable step.
2. If $t \to_{\mathrm{sv}} t'$ and σ is a solvable type, by definition $I \neq \emptyset$ and every σ_i is also solvable. Thus we can apply the *i.h.* (ii) to get $n_i > n_i'$ for each $i \in I$ so that $n_1 > n_2$. □

Corollary 3.57 (Soundness for $\cap V$). *Let* $\Gamma \Vdash^n_{\cap V} t : \sigma$. *Then,*

(i) *The term* t *is* ev*-normalizing and the number of* ev*-steps needed to normalize* t *is bound by* n.

(ii) *If* σ *is a solvable type, then* t *is* sv*-normalizing and the number of* sv*-steps needed to normalize* t *is bound by* n.

Completeness

For completeness we show that normal forms are typable, together with a subject expansion property, based on a right and left anti-substitution lemma.

Lemma 3.58 (Right anti-substitution). *If* $\Gamma \Vdash_{\cap V} t\{x/v\} : \sigma$, *then there exist* Γ_t, Γ_v *and* $\mathscr{M}$ *such that* $\Gamma_t; x : \mathscr{M} \Vdash_{\cap V} t : \sigma, \Gamma_v \Vdash_{\cap V} v : \mathscr{M}$ *and* $\Gamma = \Gamma_t \uplus \Gamma_v$.

Proof. By induction on t.

Case $t = z$. If $z = x$, then $t\{x/u\} = u$. We take $\Gamma_t = \emptyset$, $\Gamma_v = \Gamma$, $\mathscr{M} = \sigma$ and we have $x : \mathscr{M} \Vdash x : \mathscr{M}$ by (VAR) and $\Gamma \Vdash v : \mathscr{M}$ by hypothesis. Then $t\{x/v\} = v$. We take $\Gamma_t = \emptyset$, $\Gamma_v = \Gamma$, $\mathscr{M} = \sigma$ and we have $x : \mathscr{M} \Vdash x : \mathscr{M}$ by (VAR) and $\Gamma \Vdash v : \mathscr{M}$ by hypothesis.

Case $t = y \neq x$. Then $t\{x/v\} = y$. We take $\Gamma_v = \emptyset$, $\Gamma_t = y : \sigma$, $\mathscr{M} = [\,]$ and we have $y : \sigma \Vdash y : \sigma$ by hypothesis and $\emptyset \Vdash v : [\,]$ by lemma 3.62.

Case $t = \lambda y.s$ where $y \neq x$ and $y \notin \mathrm{fv}(v)$. Then $t\{x/v\} = \lambda y.s\{x/v\}$. We have $\sigma = [\mathscr{N}_i \to \sigma_i]_{i\in I}$ and $\Gamma_i; y : \mathscr{N}_i \Vdash s\{x/v\} : \sigma_i$ for $i \in I$ such that $\Gamma = \uplus_{i\in I}\Gamma_i$. By the *i.h.*, there exists Γ_s^i, Γ_v^i and $\mathscr{M}_i$ such that $\Gamma_s^i; y : \mathscr{N}_i; x : \mathscr{M}_i \Vdash s : \sigma_i$, $\Gamma_v^i \Vdash v : \mathscr{M}_i$ and $\Gamma_i = \Gamma_s^i \uplus \Gamma_v^i$,

for each $i \in I$. We conclude with rule (ABS), taking $\Gamma_t = \uplus_{i\in I}\Gamma_s^i$, $\Gamma_v = \uplus_{i\in I}\Gamma_v^i$ and $\mathscr{M} = +_{i\in I}\mathscr{M}_i$.

Case $t = s(u, y.r)$ where $y \neq x$ and $y \notin \mathrm{fv}(v)$. Then $t\{x/v\} = s\{x/v\}(u\{x/v\}, y.r\{x/v\})$. We have $\Gamma_1 \Vdash s\{x/v\} : [\mathscr{N} \longrightarrow \mathscr{N}']$, $\Gamma_2 \Vdash u\{x/v\} : \mathscr{N}$, $\Gamma_3; y : \mathscr{N}' \Vdash r\{x/v\} : \sigma$, where $\Gamma = \Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3$. By the *i.h.*, there exist $\Gamma_s, \Gamma_u, \Gamma_r, \Gamma_v^1, \Gamma_v^2, \Gamma_v^3$ and $\mathscr{M}_1, \mathscr{M}_2, \mathscr{M}_3$ such that $\Gamma_s; x : \mathscr{M}_1 \Vdash s : [\mathscr{N} \longrightarrow \mathscr{N}']$, $\Gamma_u; x : \mathscr{M}_2 \Vdash s : \mathscr{N}$, $\Gamma_r; y : \mathscr{N}'; x : \mathscr{M}_3 \Vdash s : \sigma$ and $\Gamma_v^i \Vdash v : \mathscr{M}_i$ for $i \in I$, where $\Gamma_1 = \Gamma_s \uplus \Gamma_v^1$, $\Gamma_2 = \Gamma_u \uplus \Gamma_v^2$ and $\Gamma_3 = \Gamma_r \uplus \Gamma_v^3$ for $i \in I$. We conclude with rule (APP), taking $\Gamma_t = \Gamma_s \uplus \Gamma_u \uplus \Gamma_r$, $\Gamma_v = \uplus_{i\in I}\Gamma_v^i$ and $\mathscr{M} = +_{i\in I}\mathscr{M}_i$. □

Lemma 3.59 (Left anti-substitution). *If* $\Gamma \Vdash_{\cap V} t\{x\backslash\backslash u\} : \sigma$, *then there exist* Γ_t, Γ_u *and* $\mathscr{M}$ *such that* $\Gamma_t; x : \mathscr{M} \Vdash_{\cap V} t : \sigma$, $\Gamma_u \Vdash_{\cap V} u : \mathscr{M}$ *and* $\Gamma = \Gamma_t \uplus \Gamma_u$.

Proof. By induction on u. If u is a value, then $t\{x\backslash\backslash u\} = t\{x/u\}$ and we conclude by lemma 3.58. Otherwise, $u = s(u', y.r)$ so that by definition $t\{x\backslash\backslash u\} = s(u', y.t\{x\backslash\backslash r\})$. The typing derivation of u ends with an (APP)-rule. We have $\Gamma_s \Vdash s : [\mathscr{N} \longrightarrow \mathscr{N}']$, $\Gamma_{u'} \Vdash u' : \mathscr{N}$ and $\Gamma'; y : \mathscr{N}' \Vdash r\{x/u\} : \sigma$ where $\Gamma = \Gamma_s \uplus \Gamma_{u'} \uplus \Gamma'$. By the *i.h.*, there exists Γ_r, Γ_u and $\mathscr{M}$ such that $\Gamma_r; y : \mathscr{N}'; x : \mathscr{M} \Vdash r : \sigma$, $\Gamma_u \Vdash u : \mathscr{M}$, where $\Gamma' = \Gamma_r \uplus \Gamma_u$. We conclude with rule (APP). □

We begin with the base cases of subject expansion.

Lemma 3.60. *Let* $\Gamma \Vdash_{\cap V} t_2 : \sigma$ *and* $t_1 \longmapsto_{\{\beta_V,\pi\}} t_2$. *Then* $\Gamma \Vdash_{\cap V} t_1 : \sigma$.

Proof. The cases are shown successively.

Case $t_1 = (\lambda x.t)(u, y.r) \longmapsto_{\beta V} r\{y\backslash\backslash t\{x\backslash\backslash u\}\} = t_2$. By lemma 3.59, there exist Γ_r, $\Gamma_{t\{x\backslash\backslash u\}}$ and $\mathscr{N}$ such that $\Gamma_r; y : \mathscr{N} \Vdash r : \sigma$, $\Gamma_{t\{x\backslash\backslash u\}} \Vdash t\{x\backslash\backslash u\} : \mathscr{N}$ and $\Gamma = \Gamma_{t\{x\backslash\backslash u\}} \uplus \Gamma_r$. By lemma 3.59 again, there exist Γ_t, Γ_u and $\mathscr{M}$ such that $\Gamma_t; x : \mathscr{M} \Vdash t : \mathscr{N}$, $\Gamma_u \Vdash u : \mathscr{M}$ and $\Gamma_{t\{x\backslash\backslash u\}} = \Gamma_t \uplus \Gamma_u$. We thus have $\Gamma = \Gamma_t \uplus \Gamma_u \uplus \Gamma_r$. We can build the following derivation:

$$\dfrac{\dfrac{\Gamma_t; x : \mathscr{M} \Vdash t : \mathscr{N}}{\Gamma_t \vdash \lambda x.t : [\mathscr{M} \longrightarrow \mathscr{N}]}\,(\text{ABS}) \qquad \Gamma_u \Vdash u : \mathscr{M} \qquad \Gamma_r; y : \mathscr{N} \Vdash r : \sigma}{\Gamma \vdash (\lambda x.t)(u, y.r) : \sigma}\,(\text{APP})$$

Case $t_1 = t(u, x.r)(u', y.r') \longmapsto_\pi t(u, x.r(u', y.r')) = t_2$. We have the following derivation:

$$\dfrac{\Phi_t \qquad \Phi_u \qquad \dfrac{\Phi_r \qquad \Phi_{u'} \qquad \Phi_{r'}}{\Gamma_{r'} \uplus \Gamma_{u'} \uplus \Gamma_{r'} \vdash r(u', y.r') : \sigma}\,(\text{APP})}{\Gamma \vdash t(u, x.r(u', y.r')) : \sigma}\,(\text{APP})$$

where $\Phi_t = \Gamma_t \Vdash t : [\mathscr{M}' \to \mathscr{N}']$, $\Phi_u = \Gamma_u \Vdash u : \mathscr{M}'$, $\Phi_r = \Gamma_r; x : \mathscr{N}' \Vdash r : [\mathscr{M} \to \mathscr{N}]$, $\Phi_{u'} = \Gamma_{u'} \Vdash u' : \mathscr{M}$ and $\Phi_{r'} = \Gamma_{r'}; y : \mathscr{N} \Vdash r' : \sigma$ such that $\Gamma = \Gamma_t \uplus \Gamma_u \uplus \Gamma_r \uplus \Gamma_{u'} \uplus \Gamma_{r'}$. We build the following derivation.

$$\dfrac{\dfrac{\Phi_t \quad \Phi_u \quad \Phi_r}{\Gamma_t \uplus \Gamma_u \uplus \Gamma_r \vdash t(u, x.r) : [\mathscr{M} \to \mathscr{N}]}\,(\text{APP}) \qquad \Phi_{u'} \quad \Phi_{r'}}{\Gamma \vdash t(u, x.r)(u', y.r') : \sigma}\,(\text{APP})$$ □

Subject expansion is also true for the whole reduction relation. Evaluation and the solving relation are special cases of the general statement for djv.

Lemma 3.61 (Subject expansion for $\cap V$). *Let* $\Gamma \Vdash_{\cap V} t_2 : \sigma$ *and* $t_1 \to_{\text{djv}} t_2$. *Then* $\Gamma \Vdash_{\cap V} t_1 : \sigma$.

Proof. By induction on $t_1 \to_{\text{djv}} t_2$.

Case $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u, y.r) \mapsto_{\mathrm{d}\beta_{\mathrm{v}}} \mathtt{D}\langle r\{y \backslash\backslash t\{x \backslash\backslash u\}\}\rangle = t_2$. This is the base case. By a simple induction on $\mathtt{D}$, there is $\Gamma' \Vdash r\{y \backslash\backslash t\{x \backslash\backslash u\}\} : \sigma$. Then, by lemma 3.60 for βv, $\Gamma' \Vdash (\lambda x.t)(u, y.r) : \sigma$. Let $t_3 = \mathtt{D}\langle(\lambda x.t)(u, y.r)\rangle$. We can easily show that $\Gamma \Vdash t_3 : \sigma$. Besides, $t_1 \mapsto^*_\pi t_3$. We conclude by lemma 3.60 for π.

Case $t_1 = t(u, y.r)$ and the reduction is internal. The derivation of t_2 ends with an (APP)-rule with premises: $\Gamma_t \Vdash t : [\mathscr{M} \to \mathscr{N}]$, $\Gamma_u \Vdash u : \mathscr{M}$ and $\Gamma_r; x : \mathscr{N} \Vdash r : \sigma$. There are several cases:

- **Subcase** $t_1 = t'(u, y.r) \to_{\text{djv}} t(u, y.r) = t_2$, where $t' \to_{\text{djv}} t$. By the *i.h.*, $\Gamma_t \Vdash t' : [\mathscr{M} \to \mathscr{N}]$.
- **Subcase** $t_1 = t(u', y.r) \to_{\text{djv}} t(u, y.r) = t_2$, where $u' \to_{\text{djv}} u$. By the *i.h.*, $\Gamma_u \Vdash u' : \mathscr{M}$.
- **Subcase** $t_1 = t(u, y.r') \to_{\text{djv}} t(u, y.r) = t_2$, where $r' \to_{\text{djv}} r$. By the *i.h.*, $\Gamma_r; x : \mathscr{N} \Vdash r' : \sigma$.

Case $t_1 = \lambda x.t \to_{\text{djv}} \lambda x.t' = t_2$. We have $\Gamma_i; x : \mathscr{M}_i \Vdash t' : \sigma_i$ for $i \in I$ (I can be empty), where $\Gamma = \uplus_{i \in I} \Gamma_i$. By the *i.h.*, we have $\Gamma_i; x : \mathscr{M}_i \Vdash^{n_1^i} t : \sigma_i$ for $i \in I$ and we conclude with rule (ABS). □

The two following lemmas state that NF_{ev} and NF_{sv} are typable in $\cap V$, with a solvable type for NF_{sv}.

Lemma 3.62.

(i) *Let* $t \in \text{NE}_{\text{ev}}$ *and* $k \geq 0$. *Then there exists* Γ *such that* $\Gamma \Vdash_{\cap V} t : [\,]^k$ *and every* $x \in \text{dom}(\Gamma)$ *has a type of the form* $[\,]^{k_x}$ $(k_x > 0)$.

(ii) *Let* $t \in \text{NF}_{\text{ev}}$. *Then there exists* Γ *such that* $\Gamma \Vdash_{\cap V} t : [\,]$ *and every* $x \in \text{dom}(\Gamma)$ *has a type of the form* $[\,]^{k_x}$ $(k_x > 0)$.

Proof. By mutual induction on $\mathrm{NF}_{\mathrm{ev}}$ and $\mathrm{NE}_{\mathrm{ev}}$. We start with the first item.

Case $t = x$. We get $x : [\,]^k \Vdash x : [\,]^k$ by rule (VAR). If $x \in \mathrm{dom}(\Gamma)$, then $k > 0$ and the statement holds for $k_x = k$.

Case $t = s(u, y.r)$ where $s, r \in \mathrm{NE}_{\mathrm{ev}}$ and $u \in \mathrm{NF}_{\mathrm{ev}}$. By the *i.h.* (2) we have $\Gamma_r; y : [\,]^{k_y} \Vdash r : [\,]^k, \Gamma_s \Vdash s : [\,]^{k_y+1}$, where $[\,]^{k_y+1} = [[\,] \to [\,]^{k_y}]$, and by the *i.h.* (1) $\Gamma_u \Vdash u : [\,]$. Notice that $\Gamma_s, \Gamma_r, \Gamma_u$ verify the statement by the *i.h.* We get $\Gamma_s \uplus \Gamma_u \uplus \Gamma_r \Vdash s(u, y.r) : [\,]^k$ by rule (APP).

Now, the second item.

Case $t = x$. We get $\emptyset \Vdash x : [\,]$ by rule (VAR).

Case $t = \lambda x.s$. We get $\emptyset \Vdash \lambda x.s : [\,]$ by rule (ABS).

Case $t = s(u, x.r)$ where $s \in \mathrm{NE}_{\mathrm{ev}}$, $u \in \mathrm{NF}_{\mathrm{ev}}$ and $r \in \mathrm{NF}_{\mathrm{ev}}$. By the *i.h.* (2) $\Gamma_u \Vdash u : [\,]$ and $\Gamma_r; y : [\,]^{k_y} \Vdash r : [\,]$. Then the *i.h.* (1) gives $\Gamma_s \Vdash s : [\,]^{k_y+1}$. Notice that $\Gamma_s, \Gamma_r, \Gamma_u$ verify the statement by the *i.h.*We get $\Gamma_s \uplus \Gamma_u \uplus \Gamma_r \Vdash s(u, x.r) : [\,]$ by rule (APP). □

Lemma 3.63. *If t is a* sv*-nf, then there exist Γ, σ^{s} solvable and $n \geq 0$ such that $\Gamma \Vdash^n_{\cap V} t : \sigma^{\mathrm{s}}$, where every $x \in \mathrm{dom}(\Gamma)$ has a type of the form $[\,]^{k_x}$ $(k_x > 0)$.*

Proof. By lemma 3.45 we can reason by induction on the grammar $\mathrm{NF}_{\mathrm{sv}}$.

Case $t = x$. Then $x : \mathscr{M}^{\mathrm{s}} \Vdash_{\cap V} x : \mathscr{M}^{\mathrm{s}}$ with $\mathscr{M}^{\mathrm{s}}$ a solvable multiset type is derivable.

Case $t = \lambda x.t'$, with $t' \in \mathrm{NF}_{\mathrm{sv}}$. Then $\Gamma' \Vdash_{\cap V} t' : \sigma$ with σ^{s} solvable holds by the *i.h.* so that we can write $\Gamma' = \Gamma; x : \mathscr{M}$. We obtain $\Gamma \Vdash_{\cap V} \lambda x.t' : [\mathscr{M} \to \sigma^{\mathrm{s}}]$ by applying rule (ABS). We conclude since $\mathscr{M} \to \sigma^{\mathrm{s}}$ is solvable because it is non-empty and σ^{s} is solvable. The domain is as expected by the *i.h.*

Case $t = s(u, y.r)$, where $s \in \mathrm{NE}_{\mathrm{ev}}$, $u \in \mathrm{NF}_{\mathrm{ev}}$ and $r \in \mathrm{NF}_{\mathrm{sv}}$. By lemma 3.62(ii), $\Gamma_u \Vdash_{\cap V} u : [\,]$. By the *i.h.* we have $\Gamma_r; y : [\,]^{k_y} \Vdash_{\cap V} r : \sigma^{\mathrm{s}}$ with σ^{s} solvable and $k_y \geq 0$ (since y may or not be in $\mathrm{fv}(r)$). Then by lemma 3.62(i) we have $\Gamma_s \Vdash_{\cap V} s : [\,]^{k_y+1}$, where $[\,]^{k_y+1} = [[\,] \to [\,]^{k_y}]$. We thus easily conclude as required, in particular, the domain is as required because Γ_u, Γ_r and Γ_s are as required by the *i.h.* and lemma 3.62. □

Corollary 3.64 (Completeness for $\cap V$). *Let $t \in \mathrm{T}_J$.*

(i) *If t is* ev*-normalizing, then t is typable in $\cap V$.*

(ii) *If t is* sv*-normalizing, then t is typable in $\cap V$ with a solvable type.*

Characterization of CbV Solvability

Before deriving the main theorem of this section, we introduce a last lemma.

Lemma 3.65. *Let* $\Gamma \Vdash_{\cap V} \mathtt{H}\langle t\rangle : \sigma^{\mathrm{s}}$. *Then* t *is typable with a solvable type.*

Proof. By induction on $\mathtt{H}$. The base case $\mathtt{H} = \Diamond$ is straightforward. In the other cases, we show that there is a derivation of a solvable type for $\mathtt{H}'\langle t\rangle$ and we conclude using the *i.h.* Indeed:

Case $\mathtt{H} = \lambda x.\mathtt{H}'$. By hypothesis, we have derivations $\left(\Gamma_i; x : \mathscr{M}_i \Vdash \mathtt{H}'\langle t\rangle : \sigma_i^{\mathrm{s}}\right)_{i\in I}$ where $\Gamma = \uplus_{i\in I}\Gamma_i$, $\sigma^{\mathrm{s}} = [\mathscr{M}_i \to \sigma_i^{\mathrm{s}}]_{i\in I}$ and $I \neq \emptyset$.

Case $\mathtt{H} = \mathtt{H}'(u, x.\mathtt{H}''\langle\!\langle x\rangle\!\rangle)$. By hypothesis there are derivations $\Gamma' \Vdash \mathtt{H}'\langle t\rangle : [\mathscr{M} \to \mathscr{N}]$ and $\Gamma''; x : \mathscr{N} \Vdash \mathtt{H}''\langle\!\langle x\rangle\!\rangle : \sigma^{\mathrm{s}}$ for some $\mathscr{M}, \mathscr{N}$. We use the *i.h.* to show that x is typable with a solvable type, *i.e.* to show that $\mathscr{N}$ is solvable. Thus, $[\mathscr{M} \to \mathscr{N}]$ is solvable and we can apply the *i.h.* on $\mathtt{H}'$ to conclude.

Case $\mathtt{H} = s(u, x.\mathtt{H}')$. By hypothesis there is a derivation $\Gamma'; x : \mathscr{N} \Vdash \mathtt{H}'\langle t\rangle : \sigma^{\mathrm{s}}$ for some $\mathscr{N}$. □

Theorem 3.66 (Characterization). *Let* $t \in \mathtt{T}_J$. *Then,*

(i) t *is potentially valuable iff* t *is* $\cap V$*-typable iff* t *is* ev*-normalizable, and*

(ii) t *is CbV solvable iff* t *is* $\cap V$*-typable with a solvable type iff* t *is* sv*-normalizable.*

Proof. Typable/Typable with a solvable type $\Longrightarrow$ ev/sv-normalizable: both hold by corollary 3.57. ev/sv-normalizable $\Longrightarrow$ potentially valuable/CbV solvable hold respectively by lemma 3.42/lemma 3.47.

For potentially valuable $\Longrightarrow$ typable: take t potentially valuable, so that there is a context $\mathtt{D}$ and a value v such that $\mathtt{D}\langle t\rangle \to_{\mathrm{djv}}^{*} v$. Since every value is $\cap V$-typable by lemma 3.62, and the system $\cap V$ satisfies subject expansion (lemma 3.61), then $\mathtt{D}\langle t\rangle$ is $\cap V$-typable, which implies t is $\cap V$-typable, by a straightforward induction on $\mathtt{D}$.

For solvable $\Longrightarrow$ typable with a solvable type: take t solvable, so that there is a context $\mathtt{H}$ such that $\mathtt{H}\langle t\rangle \to_{\mathrm{djv}}^{*} \mathtt{I}$. Since $\mathtt{I}$ is $\cap V$-typable by lemma 3.63, and the system $\cap V$ satisfies subject expansion (lemma 3.61), then $\mathtt{H}\langle t\rangle$ is $\cap V$-typable, which implies t is $\cap V$-typable by lemma 3.65. □

A direct consequence of this characterization is the important *normalization* property for $\to_{\mathrm{sv}}$. It states that if there is a reduction path from a term t to a sv-normal form, then a reduction sv from t necessarily reaches this (unique) sv-normal form. In other words, reduction $\to_{\mathrm{sv}}$ implements evaluation to a sv-normal form without failure.

Property 3.67 (Normalization for $\to_{\mathrm{sv}}$). *Let* $t \to_{\mathrm{djv}}^{*} u$ *and* $u \in \mathrm{NF}_{\mathrm{sv}}$. *Then there is* $s \in \mathrm{NF}_{\mathrm{sv}}$ *such that* $t \to_{\mathrm{sv}}^{*} s$.

Proof. Since $u \in \mathrm{NF}_{\mathrm{sv}}$, by lemma 3.63, u is typable with a solvable type. By lemma 3.61, subject expansion holds for the whole relation $\to_{\mathrm{djv}}$, so that t is also typable with a solvable type. Then by soundness (corollary 3.57), t is sv-normalizing. □

This elegant proof (available in [dCPT11, Corollary 22; MPV18]) is possible because subject expansion holds for the whole $\to_{\mathrm{djv}}$ relation. In fact, a normalization property also holds for evaluation, by a similar reasoning: $\to_{\mathrm{ev}}$ implements weak evaluation.

Equivalent definitions of solvability In the CbN λ-calculus, several equivalent definitions of solvability for a λ-term M coexist (where $\mathtt{H}$ are head contexts of the λ-calculus):

SOL-FA For all term N, there is a head context $\mathtt{H}$ such that $\mathtt{H}\langle M\rangle \to_\beta^* N$.

SOL-ID There is a head context $\mathtt{H}$ such that $\mathtt{H}\langle M\rangle \to_\beta^* \mathtt{I}$.

SOL-EX There is a β-normal term N and a head context $\mathtt{H}$ such that $\mathtt{H}\langle M\rangle \to_\beta^* N$.

Notice that the implications of (SOL-ID) from (SOL-FA) and (SOL-EX) from (SOL-ID) are trivial.

García-Pérez and Nogueira [GN16] observe that the three formulations are not equivalent in Plotkin's original CbV, where β is replaced by $\beta\mathrm{v}$ in the definitions above. For instance, the fact that (SOL-ID) implies (SOL-EX) is direct in CbN because for any term N, we have $\mathtt{I}N \mapsto_\beta N$. This is not the case in CbV as soon as N is not a value.

In our CbV framework, this equation is retrieved: for any term u, $\mathtt{I}(u, z.z) \mapsto_{\mathrm{d}\beta_{\mathrm{v}}} u$. The definitions (SOL-FA), (SOL-ID) and (SOL-EX) are obtained by textually replacing β by djv (or jv) and considering $\mathtt{H}$ as a head context in the grammar $\mathtt{T}_J$. These three definitions are equivalent not only in the CbN calculus with generalized applications (where β is replaced by djn or jn), but also in the CbV version. We give a proof for CbV, as it is a particular feature of generalized applications.

Lemma 3.68. *The definitions (1) SOL-FA, (2) SOL-ID and (3) SOL-EX are equivalent.*

Proof. The implications (1) $\implies$ (2) $\implies$ (3) are trivial. The implication (2) $\implies$ (1) is simply using the fact that $\mathtt{I}(u, z.z) \to_{\mathrm{d}\beta_{\mathrm{v}}} u$ (so $\mathtt{I}(u, z.z) \to_{\mathrm{djv}} u$) for any u. Only implication (3) $\implies$ (2) is left. Let $\mathtt{H}, t$ and $u \in \mathrm{NF}_{\mathrm{djv}}$ such that $\mathtt{H}\langle t\rangle \to_{\mathrm{djv}}^* u$. Since $\mathrm{NF}_{\mathrm{djv}} \subset \mathrm{NF}_{\mathrm{sv}}$, by lemma 3.63 u is typable with a solvable type. By subject expansion (lemma 3.61), $\mathtt{H}\langle t\rangle$ is typable with a solvable type. By lemma 3.65, so is t. By the logical characterization (theorem 3.66), t is solvable in the sense of (2). □

Remark 3.69. The equivalence between the notions of solvability also holds in the calculus with ESs of Accattoli and Guerrieri [AG22]. However, their calculus imposes the restriction on reduction that the term inside an ES must be a value. Because of this, their proof of SOL-ID $\implies$ SOL-FA is not obvious. Given $\mathtt{H}\langle t\rangle \to \mathtt{I}$ and any normal form u, they show $\mathtt{H}_u\langle t\rangle \to u$ with $\mathtt{H}_u \coloneqq ((\mathtt{H})\lambda x.u)\mathtt{I}$. In our proof, $\mathtt{H}_u$ is simply equal to $\mathtt{H}(u, z.z)$.

3.5 Extension to ΛJ, ΛJ_v and the λ-calculus

We have argued in favor of endowing generalized applications with a distant operational semantics: permutations are only used when they are necessary to unblock redexes, thus putting the focus on the computational content on the calculus, and also bringing the operational semantics of the calculus closer to the quantitative model. Nonetheless, this choice should not have an influence on overall properties such as strong normalization, solvability or potential valuability. We also wish to be conservative with respect to the original CbN and CbV calculi ΛJ and ΛJ_v.

We show this in this section. More precisely, we prove the equivalence of CbN/CbV solvability with and without distance using the quantitative type systems introduced in previous sections. We also show that our CbN/CbV notion of solvability is equivalent to the original one for the λ-calculus, a result which is expected but not evident.

3.5.1 Solvability for ΛJ and ΛJ_v

Remember that $\longrightarrow_{\mathrm{jn}}$ (resp. $\longrightarrow_{\mathrm{jv}}$) is the reduction relation associated to the original CbN (resp. CbV) calculus. In what follows we write *local* to mean *non-distant*.

Definition 3.70 (Local solvability). Let $t \in \mathtt{T}_J$.

(ΛJ) t is **CbN local solvable** *iff* there is a head context $\mathtt{H}$ and a distant context $\mathtt{D}$ such that $\mathtt{H}\langle t\rangle \longrightarrow_{\mathrm{jn}}^{*} \mathtt{D}\langle \mathtt{I}\rangle$.

(ΛJ_v) t is **CbV local solvable** *iff* there is a head context $\mathtt{H}$ such that $\mathtt{H}\langle t\rangle \longrightarrow_{\mathrm{jv}}^{*} \mathtt{I}$.

Notice that the terms $t = \Omega(y, z.\mathtt{I})$ and $t = x(\Omega, z.\mathtt{I})$ are CbN but not CbV locally solvable. The term $t = y_1(\mathtt{I}, z_1.x)(y_2(\mathtt{I}, z_2.z_2), z_3.\lambda y.z_3)$ is both CbN and CbV solvable.

Definition 3.71. The **CbN local solving reduction** $\longrightarrow_{\mathrm{lsn}}$ is generated by the closure of the rules βh and πh of definition 3.26 under head contexts. **Local evaluation** $\longrightarrow_{\mathrm{lev}}$ and the **CbV local solving reduction** $\longrightarrow_{\mathrm{lsv}}$ are defined by the closure of rules βv and π under the same contexts used in their distant counterparts (definition 3.39 and definition 3.44 respectively).

Lemma 3.72 (Local normal forms). *The following grammars generate* ***local CbN solving***, ***valuable*** *and* ***CbV solving normal forms*** *respectively. In the last case of* $\mathrm{NF}_{\mathrm{lsn}}$*, we have* $y \neq \mathrm{hv}(\mathrm{NF}_{\mathrm{lsn}})$.

$$\begin{aligned}
\mathrm{NF}_{\mathrm{lsn}} &::= x \mid \lambda x.\,\mathrm{NF}_{\mathrm{lsn}} \mid x(u, y.\,\mathrm{NF}_{\mathrm{lsn}}) \mid t(u, y.\,\mathrm{NF}_{\mathrm{lsn}})\\
\mathrm{NF}_{\mathrm{lev}} &::= x \mid \lambda x.t \mid x(\mathrm{NF}_{\mathrm{lev}}, y.\,\mathrm{NF}_{\mathrm{lev}})\\
\mathrm{NF}_{\mathrm{lsv}} &::= x \mid \lambda x.\,\mathrm{NF}_{\mathrm{lsv}} \mid x(\mathrm{NF}_{\mathrm{lev}}, y.\,\mathrm{NF}_{\mathrm{lsv}})
\end{aligned}$$

Proof. Let $t \in \mathrm{NF}_{\mathrm{lsn}} \cup \mathrm{NF}_{\mathrm{lev}} \cup \mathrm{NF}_{\mathrm{lsv}}$. It is immediate by induction on t that t does not reduce.

Let t be an lsn/lev/lsv-normal term. By induction on t:

Cases $t = x$ and $t = \lambda x.t'$. Then $t \in \mathrm{NF}_{\mathrm{lsn}} / \mathrm{NF}_{\mathrm{lev}} / \mathrm{NF}_{\mathrm{lsv}}$ is straightforward.

Case $t = t'(u, y.r)$. Since t does not β or π-reduce, t' is not an abstraction nor an application. Then $t = x(u, y.r)$. We conclude by *i.h.* □

Remark that all these reductions can be simplified a little by removing the closure rules on the left of an application, of the shape:

$$\frac{t \to_{\mathrm{lsn}} t'}{t(u, y.\mathtt{H}\langle\!\langle y \rangle\!\rangle) \to_{\mathrm{lsn}} t'(u, y.\mathtt{H}\langle\!\langle y \rangle\!\rangle)}\ (\mathrm{CbN}) \qquad \text{or} \qquad \frac{t \to_{\mathrm{lev}} t'}{t(u, y.r) \to_{\mathrm{lev/lsv}} t'(u, y.r)}\ (\mathrm{CbV})$$

Indeed, all terms can be π-reduced to terms of the shape $x(u, y.r)$ or $(\lambda x.t)(u, y.r)$. In the first case, we cannot reduce inside the first term which is a variable, and in the second case this is not necessary since we can βh/βv-reduce instead.

The proofs of operational and logical characterizations of local solvability are rather short, since they rely on the same lemmas as the distant versions. In particular, the lemmas normalizable implies solvable were stated for β/βv-reduction, and in subject reduction and expansion the base cases for β/βv and the permutations were separate. Finally, local normal forms form a subset of the distant normal forms.

3.5.1.1 Solvability in ΛJ

Lemma 3.73. *Let t be a* lsn*-normalizable term. Then t is solvable.*

Proof. Since t is lsn-normalizable, then there is a solving normal term $\mathrm{NF}_{\mathrm{lsn}}$ such that $t \to^*_{\mathrm{lsn}} \mathrm{NF}_{\mathrm{lsn}}$. We have $\mathrm{NF}_{\mathrm{lsn}} \in \mathrm{NF}_{\mathrm{sn}}$ so we can apply lemma 3.16 and conclude as in the proof of the corresponding property 3.19, because the last steps of the reduction are β-steps. We have $\mathtt{H}\langle t \rangle \to^*_{\mathrm{lsn}} \mathtt{H}\langle \mathrm{NF}_{\mathrm{lsn}} \rangle \to^*_{\beta} \mathtt{D}\langle \mathtt{I} \rangle$ and thus $\mathtt{H}\langle t \rangle \to^*_{\mathrm{jn}} \mathtt{D}\langle \mathtt{I} \rangle$. □

Lemma 3.74 (Weighted subject reduction for lsn). *Let $\Gamma \Vdash^{n_1}_{\cap N} t_1 : \sigma$ and $t_1 \to_{\mathrm{lsn}} t_2$. Then $\Gamma \Vdash^{n_2}_{\cap N} t_2 : \sigma$ with $n_1 \geq n_2$. Moreover, if $t_1 \to_{\beta\mathrm{h}} t_2$, then $n_1 > n_2$.*

Proof. By induction on $t_1 \to_{\mathrm{lsn}} t_2$ (resp. $t_1 \to_{\beta\mathrm{h}} t_2$).

- The base cases are inside lemma 3.27.
- The inductive cases are similar to the proof of lemma 3.28. □

Lemma 3.75 (Subject expansion for lsn). *Let $\Gamma \Vdash_{\cap N} t_2 : \sigma$ and $t_1 \to_{\mathrm{lsn}} t_2$. Then $\Gamma \Vdash_{\cap N} t_1 : \sigma$.*

Proof. By induction on $t_1 \to_{\mathrm{lsn}} t_2$.

- The base cases are inside lemma 3.31.
- The inductive cases are similar to the proof of lemma 3.32. □

Lemma 3.76 (Characterization in ΛJ). *Let $t \in \mathtt{T}_J$. The following are equivalent:*

(i) *t is CbN local solvable.*

(ii) *t is $\cap N$-typable.*

(iii) *t is* lsn*-normalizable.*

Proof. **Solvable $\implies$ typable** *t* solvable means there is a head context $\mathtt{H}$ s.t. $\mathtt{H}\langle t\rangle \to^*_{\mathrm{jn}} \mathtt{D}\langle \mathtt{I}\rangle$. But $\mathtt{D}\langle \mathtt{I}\rangle \in \mathrm{NF}_{\mathrm{lsn}} \subset \mathrm{NF}_{\mathrm{sn}}$ is $\cap N$-typable by lemma 3.35, and the system $\cap N$ has subject expansion (lemma 3.75), so that $\mathtt{H}\langle t\rangle$ is $\cap N$-typable which implies t is $\cap N$-typable by lemma 3.23.

Typable $\implies$ normalizable Holds by lemma 3.79 and the fact that πh terminates because π terminates [see JM00].

Normalizable $\implies$ solvable Holds by lemma 3.73. □

Lemma 3.77. *Let t be a* lev*-normalizable term. Then t is potentially valuable.*

Proof. Since t is lev-normalizable, then there is an lev-normal term t' such that $t \to^*_{\mathrm{lev}} t'$. Therefore $t' \in \mathrm{NF}_{\mathrm{lev}}$ by lemma 3.72. Moreover, $t' \in \mathrm{NF}_{\mathrm{ev}}$ since $\mathrm{NF}_{\mathrm{lev}} \subset \mathrm{NF}_{\mathrm{ev}}$ so that we can apply lemma 3.41 and conclude as in the proof of the corresponding lemma 3.42. We have $\mathtt{D}\langle t\rangle \to^*_{\mathrm{lev}} \mathtt{H}\langle t'\rangle \to^*_{\beta \mathrm{v}} \mathtt{I}$ and thus $\mathtt{D}\langle t\rangle \to^*_{\mathrm{jv}} v$. □

Lemma 3.78. *Let t be a* lsv*-normalizable term. Then t is CbV solvable.*

Proof. Since t is lsv-normalizable, then there is a lsv-normal term t' such that $t \to^*_{\mathrm{lsv}} t'$. Therefore $t' \in \mathrm{NF}_{\mathrm{lsv}}$ by lemma 3.72. Moreover, $t' \in \mathrm{NF}_{\mathrm{sv}}$ since $\mathrm{NF}_{\mathrm{lsv}} \subset \mathrm{NF}_{\mathrm{sv}}$ so that we can apply lemma 3.16 and conclude as in the proof of the corresponding lemma 3.47. We have $\mathtt{H}\langle t\rangle \to^*_{\mathrm{lsv}} \mathtt{H}\langle t'\rangle \to^*_{\beta \mathrm{v}} \mathtt{I}$ and thus $\mathtt{H}\langle t\rangle \to^*_{\mathrm{jv}} \mathtt{I}$. □

Lemma 3.79 (Weighted subject reduction for jv). *Let $\Gamma \Vdash^{n_1}_{\cap V} t_1 : \sigma$ and $t_1 \to_{\mathrm{jv}} t_2$. Then $\Gamma \Vdash^{n_2}_{\cap V} t_2 : \sigma$ with $n_1 \geq n_2$. Moreover:*

(i) *If $t_1 \to_\pi t_2$, then $n_1 = n_2$.*

(ii) *If $t_1 \to_{\beta \mathrm{v}} t_2$ is an valuable step, then $n_1 > n_2$.*

(iii) *If $t_1 \to_{\beta \mathrm{v}} t_2$ is a solving step and σ a solvable type, then $n_1 > n_2$.*

Proof. By induction on $t_1 \to_{\mathrm{jv}} t_2$ (resp. $t_1 \to_{\pi} t_2$, $t_1 \to_{\beta \mathrm{v}} t_2$).

- The base cases are inside lemma 3.55.
- The inductive cases are similar to the proof of lemma 3.56. □

Lemma 3.80 (Subject expansion for jv). *Let* $\Gamma \Vdash_{\cap V} t_2 : \sigma$ *and* $t_1 \to_{\mathrm{jv}} t_2$. *Then* $\Gamma \Vdash_{\cap V} t_1 : \sigma$.

Proof. By induction on $t_1 \to_{\mathrm{jv}} t_2$.

- The base cases are inside lemma 3.60.
- The inductive cases are similar to the proof of lemma 3.61. □

Lemma 3.81 (Characterization in ΛJ_v). *Let* $t \in \mathtt{T}_J$. *Then:*

(i) *t is local potentially valuable* $\iff$ *t is* $\cap V$*-typable* $\iff$ *t is* lev*-normalizable.*

(ii) *t is CbV local solvable* $\iff$ *t is* $\cap V$*-typable with a solvable type* $\iff$ *t is* lsv*-normalizable.*

Proof. The proof is similar to the one of theorem 3.66, but with its corresponding lemmas.

lev-**normalizable** $\implies$ **potentially valuable**: holds by lemma 3.77.

lsv-**normalizable** $\implies$ **CbV solvable**: holds by lemma 3.78.

Typable/typable with a solvable type $\implies$ lev/lsv-**normalizable**: both properties hold by lemma 3.79 and the fact that π terminates.

Potentially valuable/CbV solvable $\implies$ **typable**: uses lemma 3.80 and lemma 3.62. □

Theorem 3.82 (Local characterizations). *Let* $t \in \mathtt{T}_J$. *Then,*

CbN: *t is CbN local solvable iff t is* $\cap N$*-typable iff t is* lsn*-normalizable.*

CbV: *t is CbV local potentially valuable iff t is* $\cap V$*-typable iff t is* lev*-normalizable, and t is CbV local solvable iff t is* $\cap V$*-typable with a solvable type iff t is* lsv*-normalizable.*

Since the same notion of typability is used in the distant and local characterizations, this gives the following equivalences for free.

Corollary 3.83. *CbN solvability is equivalent to CbN local solvability, and so are CbV solvability and CbV local solvability.*

Remark 3.84. Normalization properties also hold for $\to_{\mathrm{lev}}$ and $\to_{\mathrm{lsv}}$, as well as the equivalence between the different definitions of local solvability (as in lemma 3.68).

3.5.2 Equivalence with Solvability in the λ-calculus

We also relate solvability of generalized applications to solvability in the λ-calculus. More precisely, we consider λ-calculi with explicit substitutions, whose notion of solvability corresponds to the one of the λ-calculus [GPD17]. Remember that λ-terms with explicit substitutions are denoted with uppercase letters M, N, P, and built with the following grammar (see section 1.3):

$$M, N, P ::= x \mid \lambda x.M \mid MN \mid M[x/N].$$

We show that the standard translations given in definitions 3.1 and 3.2 preserve solvability in both directions by comparing typability in type systems characterizing solvability of explicit substitutions calculi to typability in our type systems for generalized applications. Using this translation here is correct, as we do not consider strong normalization and counterexamples such as in section 4.5 do not apply.

Call-by-name

Remember that the type system $\mathscr{H}$ from section 1.3.2.1, originating from [KV14; Buc+20], characterizes head normalization in λES, and thus solvability.

Lemma 3.85. *Let $t \in \mathtt{T}_J$ and $M \in \mathtt{T}_{ES}$.*

(i) $\Gamma \Vdash_{\cap N} t : \tau$ *implies* $\Gamma \Vdash_{\mathscr{H}} t^\star : \tau$.

(ii) $\Gamma \Vdash_{\mathscr{H}} M : \tau$ *implies* $\Gamma \Vdash_{\cap N} M^\circ : \tau$.

Proof. Both statements are by induction on the type derivation. Note that we can extend the *i.h.* to multiset types in the expected way, using rule (MANY) on both sides. The base cases $t = x$ or $M = x$ are straightforward. The cases of the abstraction are straightforward by the *i.h.* The remaining cases are the following.

1. For (i), the derivation ends with rule (APP). Let $t = s(u, x.r)$, and $\Gamma \Vdash_{\cap N} s(u, x.r) : \tau$. We have $t^\star = r^\star[x/s^\star u^\star]$. By hypothesis we have $\Gamma = \Gamma' \uplus \Delta \uplus \Lambda$ and derivations $\Gamma' \Vdash_{\cap N} s : [\mathscr{M}_i \to \sigma_i]_{i\in I}$, $\Delta \Vdash_{\cap N} u : \sqcup_{i\in I}\mathscr{M}_i$ and $\Lambda; x : [\sigma_i]_{i\in I} \Vdash_{\cap N} r : \tau$. By the *i.h.* we obtain $\Gamma' \Vdash_{\mathscr{H}} s^\star : [\mathscr{M}_i \to \sigma_i]_{i\in I}$, $\Delta \Vdash_{\mathscr{H}} u^\star : \sqcup_{i\in I}\mathscr{M}_i$ and $\Lambda; x : [\sigma_i]_{i\in I} \Vdash_{\mathscr{H}} r^\star : \tau$. In particular, for each $i \in I$ we have derivations $\Gamma'_i \Vdash_{\mathscr{H}} s^\star : \mathscr{M}_i \to \sigma_i$, $\Delta_i \Vdash_{\mathscr{H}} u^\star : \mathscr{M}_i$ with $\Gamma' = \uplus_{i\in I}\Gamma'_i$ and $\Delta = \uplus_{i\in I}\Delta_i$. We build the following derivation in $\mathscr{H}$:

$$\dfrac{\Lambda; x : [\sigma_i]_{i\in I} \Vdash r^\star : \tau \qquad \dfrac{\left(\dfrac{\Gamma'_i \Vdash s^\star : \mathscr{M}_i \to \sigma_i \qquad \Delta_i \Vdash u^\star : \mathscr{M}_i}{\Gamma'_i \uplus \Delta_i \vdash s^\star u^\star : \sigma_i}\,(\to_e)\right)_{i\in I}}{\Gamma' \uplus \Delta \vdash s^\star u^\star : [\sigma_i]_{i\in I}}\,(\text{MANY})}{\Gamma' \uplus \Delta \uplus \Lambda \vdash r^\star[x/s^\star u^\star] : \tau}\,(\text{ES})$$

2. For (ii), there are two remaining cases.

   Case $(\to_e)$. . We have $M = M_1 M_2$, $\Gamma = \Gamma_1 \uplus \Gamma_2$ and $\Gamma_1 \uplus \Gamma_2 \Vdash_{\mathscr{H}} M_1 M_2 : \tau$. We have $M^\circ = {M_1}^\circ({M_2}^\circ, z.z)$. By hypothesis we have derivations $\Gamma_1 \Vdash_{\mathscr{H}} M_1 : \mathscr{M} \to \tau$ and $\Gamma_2 \Vdash_{\mathscr{H}} M_2 : \mathscr{M}$. By the *i.h.* we have $\Gamma_1 \Vdash_{\mathscr{H}} {M_1}^\circ : \mathscr{M} \to \tau$ and $\Gamma_2 \Vdash_{\mathscr{H}} {M_2}^\circ : \mathscr{M}$. We build the following derivation in $\cap N$:

$$\dfrac{\dfrac{\Gamma_1 \Vdash {M_1}^\circ : \mathscr{M} \to \tau}{\Gamma_1 \vdash {M_1}^\circ : [\mathscr{M} \to \tau]}\,(\textsc{many}) \qquad \Gamma_2 \Vdash {M_2}^\circ : \mathscr{M} \qquad \dfrac{}{z : [\tau] \vdash z : \tau}\,(\textsc{var})}{\Gamma_1 \uplus \Gamma_2 \vdash {M_1}^\circ({M_2}^\circ, z.z) : \tau}\,(\textsc{app})$$

   Case (ES). We have $M = M_1[x/M_2]$, $\Gamma = \Gamma_1 \uplus \Gamma_2$ and $\Gamma_1 \uplus \Gamma_2 \Vdash_{\mathscr{H}} M_1[x/M_2] : \tau$ and thus $M^\circ = \mathtt{I}({M_2}^\circ, x.M_1)$. By hypothesis we have $\Gamma_1; x : \mathscr{M} \Vdash_{\mathscr{H}} M_1 : \tau$ and $\Gamma_2 \Vdash_{\mathscr{H}} M_2 : \mathscr{M}$. By the *i.h.* we have $\Gamma_1; x : \mathscr{M} \Vdash_{\cap N} {M_1}^\circ : \tau$ and $\Gamma_2 \Vdash_{\cap N} {M_2}^\circ : \mathscr{M}$. Let $\mathscr{M} = [\tau_i]_{i \in I}$. We build the following derivation in $\cap N$:

$$\dfrac{\Phi \qquad \Gamma_2 \Vdash {M_2}^\circ : \mathscr{M} \qquad \Gamma_1; x : \mathscr{M} \Vdash {M_1}^\circ : \tau}{\Gamma_1 \uplus \Gamma_2 \vdash \mathtt{I}({M_2}^\star, x.{M_1}^\star) : \tau}\,(\to_e)$$

   With

$$\Phi = \dfrac{\left(\dfrac{\dfrac{}{y : [\tau_i] \vdash y : \tau_i}\,(\textsc{ax})}{\varnothing \vdash \lambda y.y : [\tau_i] \to \tau_i}\,(\to_i)\right)_{i \in I}}{\varnothing \vdash \lambda y.y : [[\tau_i] \to \tau_i]_{i \in I}}\,(\textsc{many})$$

□

Call-by-value

Despite the lack of operational characterization of CbV solvability in Plotkin's λ_v calculus, the notion is the same as in calculi with explicit substitutions or permutations.

In the literature, there is only one occurrence of a non-idempotent intersection type system for CbV solvability,[1] in [AG22]. This system $\mathscr{V}'$ is presented in figure 3.3. This system uses a different grammar of types than the one of our CbV type system. Let us call $\mathscr{G}_1$ our grammar of types, defined in section 3.4.3. Types and multi-types of $\mathscr{V}'$ are defined using the following grammar $\mathscr{G}_2$, where the ground types $a, b, c, \ldots$ still belong to the set BTV. Notice that we use different letters for types and multi-types.

$$\begin{aligned} &\textbf{(Types)} && A ::= a \in BTV \mid \mathscr{P} \to \mathscr{Q} \\ &\textbf{(Multi-types)} && \mathscr{P}, \mathscr{Q} ::= [A_1, \ldots, A_n] \quad (n \in \mathbb{N}) \end{aligned}$$

In order to show the equivalence of typability between the two systems, we will go through an intermediate type system $\mathscr{V}$ for explicit substitutions, which has rules similar to $\mathscr{V}'$, but uses the grammar $\mathscr{G}_1$, and has the same expressive power.

[1] An idempotent intersection type system was given already in [PR99].

$$\frac{}{x : [A] \vdash x : A}\ (\textsc{var}) \qquad \frac{(\Gamma_i \vdash V : A_i)_{i\in I}}{\uplus_{i\in I}\Gamma_i \vdash V : [A_i]_{i\in I}}\ (\textsc{val}) \qquad \frac{\Gamma; x : \mathscr{P} \vdash M : \mathscr{Q}}{\Gamma \vdash \lambda x.M : \mathscr{P} \to \mathscr{Q}}\ (\textsc{lam})$$

$$\frac{\Gamma \vdash M : [\mathscr{P} \to \mathscr{Q}] \qquad \Delta \vdash N : \mathscr{P}}{\Gamma \uplus \Delta \vdash MN : \mathscr{Q}}\ (@) \qquad \frac{\Gamma; x : \mathscr{P} \vdash M : \mathscr{Q} \qquad \Delta \vdash N : \mathscr{P}}{\Gamma \uplus \Delta \vdash M[x/N] : \mathscr{Q}}\ (\textsc{es})$$

Figure 3.3: System $\mathscr{V}'$.

We start by defining a function $\mathrm{fl}(\cdot)$ (*flatten*) from types in $\mathscr{G}_1$ to *multi-types* in $\mathscr{G}_2$:

$$\begin{aligned} \mathrm{fl}(a) &\coloneqq [a] \\ \mathrm{fl}(\mathscr{M} \to \sigma) &\coloneqq [\mathrm{fl}(\mathscr{M}) \to \mathrm{fl}(\sigma)] \\ \mathrm{fl}([\sigma_i]_{i\in I}) &\coloneqq \sqcup_{i\in I} \mathrm{fl}(\sigma_i) \end{aligned}$$

Notice that $\mathrm{fl}(\mathscr{M}_1 \sqcup \mathscr{M}_2) = \mathrm{fl}(\mathscr{M}_1) \sqcup \mathrm{fl}(\mathscr{M}_2)$ We extend this function to environments pointwise, by $\mathrm{fl}(x_1 : \mathscr{M}_1; \dots; x_n : \mathscr{M}_n) = x_1 : \mathrm{fl}(\mathscr{M}_1); \dots; x_n : \mathrm{fl}(\mathscr{M}_n)$. Thus, $\mathrm{fl}(\Gamma_1 \uplus \Gamma_2) = \mathrm{fl}(\Gamma_1) \uplus \mathrm{fl}(\Gamma_2)$.

System $\mathscr{V}$ is presented in figure 3.4. Compared to $\mathscr{V}'$, rule (MANY) is removed and integrated inside the rules (AX) and (λ).

$$\frac{}{x : \mathscr{M} \vdash x : \mathscr{M}}\ (\textsc{ax}) \qquad \frac{(\Gamma_i; x : \mathscr{M}_i \vdash M : \sigma_i)_{i\in I}}{\uplus_{i\in I}\Gamma_i \vdash \lambda x.M : [\mathscr{M}_i \to \sigma_i]_{i\in I}}\ (\lambda)$$

$$\frac{\Gamma \vdash M : [\mathscr{M} \to \mathscr{N}] \qquad \Delta \vdash N : \mathscr{M}}{\Gamma \uplus \Delta \vdash MN : \mathscr{N}}\ (@) \qquad \frac{\Gamma; x : \mathscr{M} \vdash M : \sigma \qquad \Delta \vdash N : \mathscr{M}}{\Gamma \uplus \Delta \vdash M[x/N] : \sigma}\ (\textsc{es})$$

Figure 3.4: System $\mathscr{V}$.

Lemma 3.86. *Let $M \in \Lambda_{ES}$. M is typable in $\mathscr{V}$ iff M is typable with a multiset type in $\mathscr{V}'$.*

Proof. For the right-to-left implication, we use the fact that $\mathscr{G}_2 \subsetneq \mathscr{G}_1$. We show the following statement: $\Gamma \Vdash_{\mathscr{V}'} M : \mathscr{P} \implies \Gamma \Vdash_{\mathscr{V}} M : \mathscr{P}$. The proof is by induction on the type derivation. If M is a value, the type derivation necessarily ends with rule (VAL) preceded by rule (VAR) or (LAM), to which corresponds a unique rule (AX) or (λ) in $\mathscr{V}$. If M is not a value, it is either an application or an explicit substitution, then the property holds by the *i.h.* since the inference rules are the same.

For the left-to-right implication, we prove the following statement: If $\Gamma \Vdash_{\mathscr{V}} M : \sigma$, then $\mathrm{fl}(\Gamma) \Vdash_{\mathscr{V}'} M : \mathrm{fl}(\sigma)$ (remember that $\mathrm{fl}(\sigma)$ is always a multiset). By induction on the

derivation.

Case (AX). Then $M = x$, $\sigma = \mathscr{M}$, $\Gamma = x : \mathscr{M}$ and $x : \mathscr{M} \Vdash x : \mathscr{M}$ ends with rule (AX). By definition, we have $\mathrm{fl}(\mathscr{M}) = [A_i]_{i\in I}$. We build the following derivation.

$$\dfrac{\left(\dfrac{}{x : [A_i] \vdash x : A_i}\ (\textsc{var})\right)_{i\in I}}{x : [A_i]_{i\in I} \vdash x : [A_i]_{i\in I}}\ (\textsc{val})$$

Case (λ). Then $M = \lambda x.M'$, $\sigma = [\mathscr{M}_i \to \sigma_i]_{i\in I}$, $\Gamma = \uplus_{i\in I}\Gamma_i$ and $\uplus_{i\in I}\Gamma_i \Vdash \lambda x.M' : [\mathscr{M}_i \to \sigma_i]_{i\in I}$ ends with rule (λ). We have $\Gamma_i; x : \mathscr{M}_i \Vdash M' : \sigma_i$ for each $i \in I$ by hypothesis and $\mathrm{fl}(\mathscr{M}_i \to \sigma_i) = [\mathrm{fl}(\mathscr{M}_i) \to \mathrm{fl}(\sigma_i)]$, so that $\mathrm{fl}(\sigma) = [\mathrm{fl}(\mathscr{M}_i) \to \mathrm{fl}(\sigma_i)]_{i\in I}$. By the *i.h.* we have $\mathrm{fl}(\Gamma_i); x : \mathrm{fl}(\mathscr{M}_i) \Vdash M' : \mathrm{fl}(\sigma_i)$ for each $i \in I$. We conclude by rule (LAM) for each $i \in I$ followed by rule (MANY).

Case (@). Then $M = M_1M_2$, $\Gamma = \Gamma_1 \uplus \Gamma_2$, $\sigma = \mathscr{N}$ and $\Gamma \Vdash M_1M_2 : \mathscr{N}$ ends with rule (@). By hypothesis we have derivations $\Gamma_1 \Vdash M_1 : [\mathscr{M} \to \mathscr{N}]$ and $\Gamma_2 \Vdash M_2 : \mathscr{M}$. By the *i.h.* we have $\mathrm{fl}(\Gamma_1) \Vdash M_1 : \mathrm{fl}([\mathscr{M} \to \mathscr{N}])$ and $\mathrm{fl}(\Gamma_2) \Vdash M_2 : \mathrm{fl}(\mathscr{M})$. Since $\mathrm{fl}([\mathscr{M} \to \mathscr{N}]) = [\mathrm{fl}(\mathscr{M}) \to \mathrm{fl}(\mathscr{N})]$ and $\mathrm{fl}(\Gamma_1 \uplus \Gamma_2) = \mathrm{fl}(\Gamma_1) \uplus \mathrm{fl}(\Gamma_2)$, we conclude with rule (@).

Case (ES). Then $M = M_1[x/M_2]$, $\Gamma = \Gamma_1 \uplus \Gamma_2$ and $\Gamma_1 \uplus \Gamma_2 \Vdash M_1[x/M_2] : \sigma$ ends with rule (ES). By hypothesis we have derivations $\Gamma_1; x : \mathscr{M} \Vdash M_1 : \sigma$ and $\Gamma_2 \Vdash M_2 : \mathscr{M}$. By *i.h.* we have $\mathrm{fl}(\Gamma_1); x : \mathrm{fl}(\mathscr{M}) \Vdash M_1 : \mathrm{fl}(\sigma)$ and $\mathrm{fl}(\Gamma_2) \Vdash M_2 : \mathrm{fl}(\mathscr{M})$. Since $\mathrm{fl}(\Gamma_1 \uplus \Gamma_2) = \mathrm{fl}(\Gamma_1) \uplus \mathrm{fl}(\Gamma_2)$, we conclude with rule (ES). □

In [AG22], CbV solvability is shown equivalent to being typable with a solvable multiset type in $\mathscr{V}'$. Moreover, solvable types in $\mathscr{G}_2$ are also solvable in $\mathscr{G}_1$, and if σ is a solvable type in $\mathscr{G}_1$, then we can show by induction that $\mathrm{fl}(\sigma)$ is also a solvable type in $\mathscr{G}_2$. Then, we get the following characterization.

Corollary 3.87. *Let $M \in \mathtt{T}_{ES}$. Then M is solvable iff M is typable in $\mathscr{V}$ with a solvable type.*

We are now ready to relate CbV solvability of ES and of generalized applications using the type system $\mathscr{V}$.

Lemma 3.88. *Let $t \in \mathtt{T}_J$ and $M \in \mathtt{T}_{ES}$.*

(i) $\Gamma \Vdash_{\cap V} t : \sigma$ *implies* $\Gamma \Vdash_{\mathscr{V}} t^{\star} : \sigma$

(ii) $\Gamma \Vdash_{\mathscr{V}} M : \sigma$ *implies* $\Gamma \Vdash_{\cap V} M^{\circ} : \sigma$.

Proof. Both statements are by induction on the type derivation. The base cases $t = x$ or $M = x$ are straightforward. The cases of the abstraction are straightforward by the *i.h.* The remaining cases are the following.

1. For (i), when the derivation ends with rule (@). Let $t = s(u, x.r)$, and $\Gamma \Vdash_{\cap V} s(u, x.r) : \sigma$. We have $t^\star = r^\star[x/s^\star u^\star]$. By hypothesis we have derivations $\Gamma' \Vdash_{\mathscr{V}} s : [\mathscr{M} \to \mathscr{N}]$, $\Delta \Vdash_{\mathscr{V}} u : \mathscr{M}$ and $\Lambda; x : \mathscr{N} \Vdash_{\mathscr{V}} r : \sigma$ with $\Gamma = \Gamma' \uplus \Delta \uplus \Lambda$. By the *i.h.* we obtain $\Gamma' \Vdash_{\mathscr{V}} s^\star : [\mathscr{M} \to \mathscr{N}]$, $\Delta \Vdash_{\mathscr{V}} u^\star : \mathscr{M}$ and $\Lambda; x : \mathscr{N} \Vdash_{\mathscr{V}} r^\star : \sigma$. We build the following derivation in $\mathscr{V}$:

$$\dfrac{\dfrac{\Gamma' \vdash s^\star : [\mathscr{M} \to \mathscr{N}] \qquad \Delta \vdash u^\star : \mathscr{N}}{\Gamma' \uplus \Delta \vdash s^\star u^\star : \mathscr{N}}\,(@) \qquad \Lambda; x : \mathscr{N} \vdash r : \sigma}{\Gamma' \uplus \Delta \uplus \Lambda \vdash r^\star[x/s^\star u^\star] : \sigma}\,(\textsc{es})$$

2. For (ii), there are two remaining cases.

   Case (@). We have $M = M_1 M_2$, $\sigma = \mathscr{N}$, $\Gamma = \Gamma_1 \uplus \Gamma_2$, $\Gamma_1 \uplus \Gamma_2 \Vdash_{\mathscr{V}} M_1 M_2 : \mathscr{N}$ and $M^\circ = {M_1}^\circ({M_2}^\circ, z.z)$. By hypothesis we have derivations $\Gamma_1 \Vdash_{\mathscr{V}} M_1 : [\mathscr{M} \to \mathscr{N}]$ and $\Gamma_2 \Vdash_{\mathscr{V}} M_2 : \mathscr{M}$. By the *i.h.* we have $\Gamma_1 \Vdash_{\cap V} {M_1}^\circ : [\mathscr{M} \to \mathscr{N}]$ and $\Gamma_2 \Vdash_{\cap V} {M_2}^\circ : \mathscr{M}$. We build the following derivation in $\cap V$:

$$\dfrac{\Gamma_1 \Vdash {M_1}^\circ : [\mathscr{M} \to \mathscr{N}] \qquad \Gamma_2 \Vdash {M_2}^\circ : \mathscr{M} \qquad \dfrac{}{z : \mathscr{N} \vdash z : \mathscr{N}}\,(\textsc{var})}{\Gamma_1 \uplus \Gamma_2 \vdash {M_1}^\circ({M_2}^\circ, z.z) : \mathscr{N}}\,(\textsc{app})$$

   Case (ES). We have $M = M_1[x/M_2]$, $\Gamma = \Gamma_1 \uplus \Gamma_2$, $\Gamma_1 \uplus \Gamma_2 \Vdash_{\mathscr{V}} M_1[x/M_2] : \sigma$ and $M^\circ = \mathtt{I}({M_2}^\circ, x.{M_1}^\circ)$. By hypothesis we have $\Gamma_1; x : \mathscr{M} \Vdash_{\mathscr{V}} M_1 : \sigma$ and $\Gamma_2 \Vdash_{\mathscr{V}} M_2 : \mathscr{M}$. By the *i.h.* we have $\Gamma_1; x : \mathscr{M} \Vdash_{\cap V} {M_1}^\circ : \sigma$ and $\Gamma_2 \Vdash_{\cap V} {M_2}^\circ : \mathscr{M}$. We build the following derivation in $\cap V$:

$$\dfrac{\dfrac{\dfrac{}{y : \mathscr{M} \vdash y : \mathscr{M}}\,(\textsc{var})}{\vdash \lambda y.y : [\mathscr{M} \to \mathscr{M}]}\,(\textsc{abs}) \qquad \Gamma_2 \Vdash {M_2}^\circ : \mathscr{M} \qquad \Gamma_1; x : \mathscr{M} \Vdash {M_1}^\circ : \sigma}{\Gamma_1 \uplus \Gamma_2 \vdash (\lambda y.y)({M_2}^\circ, x.{M_1}^\circ) : \sigma}\,(\textsc{app})$$

□

As CbN/CbV solvability in the λ-calculus is equivalent to $\mathscr{V}'$-typability/$\mathscr{V}$-typability with a solvable type, we get the final results:

Corollary 3.89. *Let t be a $\mathtt{T}_J$-term.*

(i) *t is CbN solvable if and only if $t^\#$ is CbN solvable in the λ-calculus.*

(ii) *t is CbV solvable if and only if $t^\#$ is CbV solvable in the λ-calculus.*

3.6 Comparison of the CbV Calculi with ES and Generalized Applications

We have shown equivalence between solvability and potential valuability of $\lambda_{\mathtt{vsub}}$ and of $\lambda J_v/\Lambda J_v$. Our characterizations of solvability and potential valuability were given by independent and semantic proofs. We now wish to compare both formalisms on an operational level. For this, we give simulations between the reductions. Simulations hold in both ways for the general reduction and weak evaluation, but not the solving reduction, because our formulation, albeit equivalent, is a tad more restricted.

For the simulations, we introduce an equivalence on $\mathtt{T}_J$-terms. This equivalence is a strong bisimulation, which gives a rich equational theory to $\mathtt{T}_J$ and is the main contribution of this section. We finally compare the equational theories of the calculi $\lambda_{\mathtt{vsub}}$ and λJ_v.

3.6.1 Simulations

We start with a simulation of λJ_v in $\lambda_{\mathtt{vsub}}$. When doing a simulation from generalized applications to explicit substitution in CbN, we need to resort to the *faithful* translation $(\cdot)^\star$ defined in section 4.5.1. In CbV instead, the original map $(\cdot)^{\star}$ already preserves strong normalization. Take for instance the counterexample of section 1.2.2.2, $t = \delta(\delta, y.r)$ where $y \notin r$. This term is strongly normalizing in the call-by-name λJ_n, but not in λES. In CbV, $t = \delta(\delta, y.r)$ is not strongly normalizing already in λJ_v, and stays so with the translation.

For the calculus $\lambda_{\mathtt{vsub}}$, we use the names of [AG22]: $\to_{\mathtt{vsub}}$ for the general reduction, $\to_{\mathtt{o}}$ for weak (open) evaluation corresponding to potential valuability and $\to_{\mathtt{s}}$ for the solving relation.

Lemma 3.90. *Let $t_1 \to_{\mathtt{djv}} t_2$. Then, ${t_1}^{\star} \to^{3}_{\mathtt{vsub}} {t_2}^{\star}$. In particular,*

(i) *If $t_1 \to_{\mathtt{ev}} t_2$, then ${t_1}^{\star} \to^{3}_{\mathtt{o}} {t_2}^{\star}$.*

(ii) *If $t_1 \to_{\mathtt{sv}} t_2$, then ${t_1}^{\star} \to^{3}_{\mathtt{s}} {t_2}^{\star}$.*

Proof. The base case is $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u, y.r) \mapsto_{\mathtt{d}\beta_{\mathtt{v}}} \mathtt{D}\langle r\{y \backslash\backslash t\{x \backslash\backslash u\}\}\rangle$. We decompose $t = \mathtt{D}_1\langle v_1\rangle$ and $u = \mathtt{D}_2\langle v_2\rangle$. Let $\mathtt{D}'_1 = \mathtt{D}_1\{x/v_2\}$.

$$t_2 = \mathtt{D}\langle r\{y \backslash\backslash \mathtt{D}_1\langle v_1\rangle\{x \backslash\backslash \mathtt{D}_2\langle v_2\rangle\}\}\rangle = \mathtt{D}\langle \mathtt{D}_2\langle r\{y \backslash\backslash \mathtt{D}_1\langle v_1\rangle\{x/v_2\}\}\rangle\rangle = \mathtt{D}\langle \mathtt{D}_2\langle \mathtt{D}'_1\langle r\{y/v_1\{x/v_2\}\}\rangle\rangle\rangle$$

For any $\mathtt{D}_0, t_0$, a simple induction on $\mathtt{D}_0$ shows that $\mathtt{D}_0\langle t_0\rangle^{\star} = \mathtt{L}_0\langle {t_0}^{\star}\rangle$ for some $\mathtt{L}_0$. Let $\mathtt{L}, \mathtt{L}_1, \mathtt{L}'_1, \mathtt{L}_2$ be the translations (extended to contexts) of $\mathtt{D}, \mathtt{D}_1, \mathtt{D}'_1, \mathtt{D}_2$. Because the translation commutes with substitution, we have ${t_2}^{\star} = \mathtt{L}\langle \mathtt{L}_2\langle \mathtt{L}'_1\langle r^{\star}\{y/{v_1}^{\star}\{x/{v_2}^{\star}\}\}\rangle\rangle\rangle$.

$$\begin{aligned}
t_1{}^\star &= r^\star[y/\mathrm{L}\langle(\lambda x.\mathrm{L}_1\langle v_1{}^\star\rangle)(\mathrm{L}_2\langle v_2{}^\star\rangle)\rangle] \\
&\to_{\mathrm{vsub}} r^\star[y/\mathrm{L}\langle(\mathrm{L}_1\langle v_1{}^\star\rangle)[x/\mathrm{L}_2\langle v_2{}^\star\rangle]\rangle] \\
&\to_{\mathrm{vsub}} r^\star[y/\mathrm{L}\langle\mathrm{L}_2\langle(\mathrm{L}_1\langle v_1\rangle)^\star\{x/v_2{}^\star\}\rangle\rangle] \\
&= r^\star[y/\mathrm{L}\langle\mathrm{L}_2\langle\mathrm{L}_1'\langle v_1{}^\star\{x/v_2{}^\star\}\rangle\rangle\rangle] \\
&\to_{\mathrm{vsub}} \mathrm{L}\langle\mathrm{L}_2\langle\mathrm{L}_1'\langle r^\star\{y/v_1{}^\star\{x/v_2{}^\star\}\}\rangle\rangle\rangle \\
&= t_2{}^\star
\end{aligned}$$

Notice that the rules applied are within the relation $\to_{\mathrm{o}}$, so that (i) and (ii) are verified.
Now, the inductive cases.

Case $t_1 = \lambda x.t \to_{\mathrm{djv}} \lambda x.t' = t_2$. This holds by *i.h.* This step is not an ev-step, but it is a sv-step if $t \to_{\mathrm{djv}} t'$ is. In this case, the steps $t_1{}^\star \to_{\mathrm{vsub}}^+ t_2{}^\star$ are s-steps.

Case $t_1 = t(u, y.r) \to_{\mathrm{djv}} t'(u', y.r') = t_2$ where $t \to_{\mathrm{djv}} t'$ or $u \to_{\mathrm{djv}} u'$ or $r \to_{\mathrm{djv}} r'$. We have $t_1{}^\star = r^\star[y/t^\star u^\star]$ and $t_2{}^\star = r'{}^\star[y/t'{}^\star u'{}^\star]$. We conclude by *i.h.* on t', u' or r'.

Suppose $t_1 \to_{\mathrm{ev}} t_2$. Then $t \to_{\mathrm{ev}} t'$, $u \to_{\mathrm{ev}} u'$ or $r \to_{\mathrm{ev}} r'$. We prove item (i) by *i.h.* because the terms $t^\star$, $u^\star$ and $r^\star$ are all in an open context of $\lambda_{\mathtt{vsub}}$.

Suppose $t_1 \to_{\mathrm{sv}} t_2$. There are two possibilities.

Case $t \to_{\mathrm{ev}} t'$ or $u \to_{\mathrm{ev}} t'$. As in the previous case, the $t^\star$ and $u^\star$ are in an open context, so they are in a solving context of $\lambda_{\mathtt{vsub}}$. We conclude (ii) by *i.h.*

Case $r \to_{\mathrm{sv}} r'$. In that case, $r^\star$ is in a solving context of $\lambda_{\mathtt{vsub}}$. We conclude (ii) by *i.h.* □

To establish an exact simulation of $\lambda_{\mathtt{vsub}}$ in λJ_v, we need two ingredients. The first one is a new translation $(\cdot)^\bullet$. Indeed, the original one $(\cdot)^\circ$ from definition 3.1 induces a simulation of each $\to_{\mathrm{sub}}$-reduction step on $\lambda_{\mathtt{vsub}}$ into a $\to_{\mathrm{d}\beta_{\mathrm{v}}}$-reduction step on $\mathtt{T}_J$, but cannot simulate the creation of an ES by rule $\to_{\mathrm{dB}}$. A solution is to refine the translation $(\cdot)^\circ$ for applications, yielding the following alternative $(\cdot)^\bullet$:

$$\begin{aligned}
x^\bullet &\coloneqq x & (\lambda x.M)^\bullet &\coloneqq \lambda x.M^\bullet \\
(MN)^\bullet &\coloneqq \mathtt{I}(N^\bullet, y.M^\bullet(y, z.z)) & M[x/N]^\bullet &\coloneqq \mathtt{I}(N^\bullet, x.M^\bullet)
\end{aligned}$$

Since the clause for ES is not changed, simulation of each sub-reduction step by a $\mathrm{d}\beta_{\mathrm{v}}$-reduction step holds as before. The improvement lies in the simulation of each dB-reduction step:

$$((\lambda x.M)N)^\bullet = \mathtt{I}(N^\bullet, y.(\lambda x.M^\bullet)(y, z.z)) \to_{\mathrm{d}\beta_{\mathrm{v}}} \mathtt{I}(N^\bullet, y.M^\bullet\{x/y\}) =_\alpha (M[x/N])^\bullet$$

The second ingredient is the following equivalence, where we assume no capture of variables, and where $z_2 \notin \mathrm{fv}(t_1) \cup \mathrm{fv}(u_1)$:

$$t_2(u_2, z_2.t_1(u_1, z_1.r)) \sim_{\mathrm{com}} t_1(u_1, z_1.t_2(u_2, z_2.r))$$

We write $\equiv_{\mathrm{com}}$ for the reflexive and transitive closure of $\sim_{\mathrm{com}}$. The congruence $\equiv_{\mathrm{com}}$ is a *strong bisimulation* with respect to $\to_{\mathrm{djv}}$, $\to_{\mathrm{ev}}$ and $\to_{\mathrm{sv}}$. We give a definition and properties of strong bisimulations in the next section section 3.6.2, where we define a larger strong bisimulation on $\mathtt{T}_J$, containing $\equiv_{\mathrm{com}}$. Roughly, two bisimilar terms will have the same observational and operational behavior; they may be represented by the same object in a graphical system. We write $\to_{\mathrm{djv}/\equiv_{\mathrm{jv}}}$ for the reduction $\to_{\mathrm{djv}}$ modulo $\equiv_{\mathrm{jv}}$, similarly for $\to_{\mathrm{djv}/\equiv_{\mathrm{com}}}$ and $\to_{\mathrm{ev}/\equiv_{\mathrm{com}}}$ modulo $\equiv_{\mathrm{com}}$. These relations are confluent, by the upcoming lemma 3.93 and confluence of the original relations.

Lemma 3.91. *Let $M_1 \to_{\mathrm{vsub}} M_2$. Then, $M_1^\bullet \to_{\mathrm{djv}/\equiv_{\mathrm{com}}} M_2^\bullet$. In particular, if $M_1 \to_{\mathrm{o}} M_2$, then $M_1^\bullet \to_{\mathrm{ev}/\equiv_{\mathrm{com}}} M_2^\bullet$.*

Proof. There are two base cases.

Case $M_1 = \mathtt{L}\langle\lambda x.M\rangle N \mapsto_{\mathrm{dB}} \mathtt{L}\langle M[x/N]\rangle$. For any list context $\mathtt{L}_0$ and term M_0, it is straightforward that $\mathtt{L}_0\langle M_0\rangle^\bullet = \mathtt{D}_0\langle M_0^\bullet\rangle$ for some $\mathtt{D}_0$. Then, for some $\mathtt{D}$:

$$\begin{aligned} M_1^\bullet &= \mathtt{I}(N^\bullet, y.\mathtt{D}\langle\lambda x.M^\bullet\rangle(y, z.z)) \\ &\to_{\mathrm{djv}} \mathtt{I}(N^\bullet, y.\mathtt{D}\langle M^\bullet\rangle\{x/y\}) \\ &=_\alpha \mathtt{I}(N^\bullet, x.\mathtt{D}\langle M^\bullet\rangle) \\ &\equiv_{\mathrm{com}} \mathtt{D}\langle\mathtt{I}(N^\bullet, x.M^\bullet)\rangle \\ &= \mathtt{L}\langle M[x/N]\rangle^\bullet = M_2^\bullet \end{aligned}$$

The rewrite step is done in a ev-context, so that the case of $\to_{\mathrm{ev}/\equiv_{\mathrm{com}}}$ is verified.

Case $M_1 = M[x/\mathtt{L}\langle V\rangle] \to_{\mathrm{sub}} \mathtt{L}\langle M\{x/V\}\rangle$. Then, for some $\mathtt{D}$, and because $(\cdot)^\bullet$ commutes with substitution:

$$\begin{aligned} \mathtt{I}(\mathtt{D}\langle V^\bullet\rangle, y.M^\bullet) &\to_{\mathrm{djv}} M^\bullet\{y \backslash\!\backslash \mathtt{D}\langle V^\bullet\rangle\} \\ &= \mathtt{D}\langle M^\bullet\{y/V^\bullet\}\rangle \\ &= \mathtt{L}\langle M\{x/V\}\rangle^\bullet \end{aligned}$$

The $\to_{\mathrm{djv}}$-step is a root step, thus in particular an $\to_{\mathrm{ev}}$-step.

We now consider the inductive cases.

Case $M_1 = \lambda x.M \to_{\mathrm{vsub}} \lambda x.M' = M_2$. By *i.h.* Moreover, the step $M \to_{\mathrm{vsub}} M'$ is not an open reduction.

Case $M_1 = MN \to_{\mathrm{vsub}} M'N'$ where $M \to_{\mathrm{vsub}} M'$ or $N \to_{\mathrm{vsub}} N'$. Then we have that $M_1^\bullet = \mathtt{I}(N^\bullet, y.M^\bullet(y, z.z))$. We conclude by *i.h.* on M' or N'. Suppose $M_1 \to_{\mathrm{o}} M_2$. $M^\bullet$ and $N^\bullet$ are in an ev context so the case for $\to_{\mathrm{ev}}$ is verified. □

Simulations hold between $\to_{\mathrm{djv}}$ and $\to_{\mathrm{vsub}}$, as well as $\to_{\mathrm{ev}}$ and $\to_{\mathrm{o}}$. However, simulation of $\to_{\mathrm{s}}$ in $\to_{\mathrm{sv}}$ fails, which is why the previous lemma does not treat it. This is because

in $\lambda_{\mathtt{vsub}}$, it is possible to reduce inside an abstraction that is on the left of an application. This is not the case in λJ_v. The absence of that special case does not affect normalization, since these abstractions can be destroyed by application of a $\mathrm{d}\beta_{\mathrm{v}}$-rule. It is possible to add a contextual rule for this case to the relation $\longrightarrow_{\mathrm{sv}}$ while keeping the operational and logical characterizations:

$$\frac{t \longrightarrow_{\mathrm{sv}} t'}{t(u, x.\mathrm{H}\langle\!\langle x \rangle\!\rangle) \longrightarrow_{\mathrm{sv}} t'(u, x.\mathrm{H}\langle\!\langle x \rangle\!\rangle)}$$

However, we prefer our formulation, in which it is never necessary to search for the head variable nested inside the term. On the other hand, it seems possible to restrict the solving reduction $\longrightarrow_{\mathrm{s}}$ of $\lambda_{\mathtt{vsub}}$ to correspond to $\longrightarrow_{\mathrm{sv}}$, without losing properties.

The simulations show that it is possible to relate generalized applications and explicit substitutions at a syntactic level. However, we can see that the relation is not straightforward, as we must be careful in crafting the translations.

The simulations between $\longrightarrow_{\mathrm{ev}}$ and $\longrightarrow_{\mathrm{o}}$ are an important element to derive the result of operational characterization of potential valuability by $\longrightarrow_{\mathrm{ev}}$ as a consequence of that result in [AG22]. The operational characterization of solvability could be derived from the one in [AG22] in the same way, supposing the extended definition of sv using the rule defined above.

3.6.2 Strong bisimulation for λJ_v

We now define a strong bisimulation $\equiv_{\mathrm{jv}}$ for λJ_v. This is to our knowledge the first (strong) bisimulation defined for generalized applications. A *strong bisimulation* is a congruence on terms that equates terms having the same behavior. It is defined as follows: For $\mathscr{R} \in \{\mathrm{djv}, \mathrm{ev}, \mathrm{sv}\}$, for any two terms in relation $t_1 \equiv_{\mathrm{jv}} t_2$ and $t_1 \longrightarrow_{\mathscr{R}} t_1'$, then there is t_2' such that $t_2 \longrightarrow_{\mathscr{R}} t_2'$.

The relation $\equiv_{\mathrm{jv}}$ is computationally irrelevant: it commutes with reduction steps, and can thus be postponed, does not change the number of steps in the reduction sequence, and preserves confluence and normalization.

It is one further advantage of the distant paradigm to allow such strong bisimulations on T_J: since no reduction is stuck, permutations can be included in a second phase as a strong bisimulation without effort since properties of the reduction are preserved. The generality of the calculus with arbitrary and separate permutation steps is thus retrieved. Reasoning can then also be done modulo bisimulation.

The equivalence $\equiv_{\mathrm{jv}}$ is defined by the reflexive, symmetric and transitive closure under all contexts of the following rules, where we suppose no capture of variables:

$$\begin{aligned}
&t_1(u_1, z_1.t_2)(u_2, z_2.r) \sim_{\pi} t_1(u_1, z_1.t_2(u_2, z_2.r)) \\
&t_2(t_1(u_1, z_1.u_2), z_2.r) \sim_{\mathrm{arg}} t_1(u_1, z_1.t_2(u_2, z_2.r)) \\
&t_2(u_2, z_2.t_1(u_1, z_1.r)) \sim_{\mathrm{com}} t_1(u_1, z_1.t_2(u_2, z_2.r)) \text{ where } z_2 \notin \mathrm{fv}(t_1) \cup \mathrm{fv}(u_1)
\end{aligned}$$

We also write $\equiv^1_{\mathrm{jv}}$ for the non-reflexive and non-transitive closure of $\sim_{\pi} \cup \sim_{\mathrm{arg}} \cup \sim_{\mathrm{com}}$ under all contexts (similarly for $\equiv^1_{\pi}$, $\equiv^1_{\mathrm{arg}}$ and $\equiv^1_{\mathrm{com}}$).

We now prove that $\equiv_{\mathrm{jv}}$ is a strong bisimulation. We will use the following auxiliary lemma.

Lemma 3.92. *Let $t, u, r, s \in \mathtt{T}_J$.*

(i) *The following equations hold:*

- $t\{x\backslash\backslash s\}(u, z.r) \equiv_\pi t(u, z.r)\{x\backslash\backslash s\}$, *when* $x \notin \mathrm{fv}(u) \cup \mathrm{fv}(r)$.
- $t(u\{x\backslash\backslash s\}, z.r) \equiv_{\mathrm{arg}} t(u, z.r)\{x\backslash\backslash s\}$, *when* $x \notin \mathrm{fv}(t) \cup \mathrm{fv}(r)$.
- $t(u, z.r\{x\backslash\backslash s\}) \equiv_{\mathrm{com}} t(u, z.r)\{x\backslash\backslash s\}$, *when* $x \notin \mathrm{fv}(t) \cup \mathrm{fv}(u)$.

(ii) *If $t \equiv^1_{\mathrm{jv}} t'$, then $t\{x\backslash\backslash u\} \equiv^1_{\mathrm{jv}} t'\{x\backslash\backslash u\}$.*

(iii) *If $u \equiv^1_{\mathrm{jv}} u'$, then $t\{x\backslash\backslash u\} \equiv_{\mathrm{jv}} t\{x\backslash\backslash u'\}$.*

Proof. (i) By induction on s. The cases where s is a value are direct by the hypothesis. Let $s = s_1(s_2, y.s_3)$. We have $t(u, z.r)\{x\backslash\backslash s\} = s_1(s_2, y.t(u, z.r)\{x\backslash\backslash s_3\})$. Then, using the *i.h.* on the last step:

- $t\{x\backslash\backslash s\}(u, z.r) = s_1(s_2, y.t\{x\backslash\backslash s_3\})(u, z.r) \equiv_\pi s_1(s_2, y.t\{x\backslash\backslash s_3\}(u, z.r)) \equiv_\pi t(u, z.r)\{x\backslash\backslash s\}$
- $t(u\{x\backslash\backslash s\}, z.r) = t(s_1(s_2, y.u\{x\backslash\backslash s_3\}), z.r) \equiv_{\mathrm{arg}} s_1(s_2, y.t(u\{x\backslash\backslash s_3\}, z.r)) \equiv_{\mathrm{arg}} t(u, z.r)\{x\backslash\backslash s\}$
- $t(u, z.r\{x\backslash\backslash s\}) = t(u, z.s_1(s_2, y.r\{x\backslash\backslash s_3\})) \equiv_{\mathrm{com}} s_1(s_2, y.t(u, z.r\{x\backslash\backslash s_3\})) \equiv_{\mathrm{com}} t(u, z.r)\{x\backslash\backslash s\}$

(ii) By induction on u. In the base case where $u = v$ is a value, by a nested induction on $t \equiv^1_{\mathrm{jv}} t'$. The base cases $t \sim_\pi t'$, $t \sim_{\mathrm{arg}} t'$ and $t \sim_{\mathrm{com}} t'$ are straightforward by definition of the substitution. The inductive cases are direct by *i.h.*

In the inductive case of the outer induction, let $u = s_1(s_2, y.s_3)$. Then, by the *i.h.* $t\{x\backslash\backslash u\} = s_1(s_2, y.t\{x\backslash\backslash s_3\}) \equiv^1_{\mathrm{jv}} s_1(s_2, y.t'\{x\backslash\backslash s_3\}) = t'\{x\backslash\backslash u\}$.

(iii) By induction on $u \equiv^1_{\mathrm{jv}} u'$. In all the base cases, let $u' = t_1(u_1, z_1.t_2(u_2, z_2.r))$, and thus $t\{x\backslash\backslash u'\} = t_1(u_1, z_1.t_2(u_2, z_2.t\{x\backslash\backslash r\}))$.

Case $u = t_1(u_1, z_1.t_2)(u_2, z_2.r) \sim_\pi u'$. We have $t\{x\backslash\backslash u\} = t_1(u_1, z_1.t_2)(u_2, z_2.t\{x\backslash\backslash r\}) \sim_\pi t\{x\backslash\backslash u'\}$.

Case $u = t_2(t_1(u_1, z_1.u_2), z_2.r) \sim_{\mathrm{arg}} u'$. We have $t\{x\backslash\backslash u\} = t_2(t_1(u_1, z_1.u_2), z_2.t\{x\backslash\backslash r\}) \sim_{\mathrm{arg}} t\{x\backslash\backslash u'\}$.

Case $u = t_2(u_2, z_2.t_1(u_1, z_1.r)) \sim_{\mathrm{com}} u'$. We have $t\{x\backslash\backslash u\} = t_2(u_2, z_2.t_1(u_1, z_1.t\{x\backslash\backslash r\})) \sim_{\mathrm{com}} t\{x\backslash\backslash u'\}$.

Case $u = u_1(u_2, y.u_3) \equiv^1_{\mathrm{jv}} u'_1(u'_2, y.u'_3)$. Where $u_i \equiv^1_{\mathrm{jv}} u'_i$ holds for exactly one $1 \leq i \leq 3$. We have $t\{x\backslash\backslash u\} = u_1(u_2, y.t\{x\backslash\backslash u_3\}) \equiv_{\mathrm{jv}} u'_1(u'_2, y.t\{x\backslash\backslash u'_3\}) = t\{x\backslash\backslash u'\}$. If $u_1 \equiv^1_{\mathrm{jv}} u'_1$ or $u_2 \equiv^1_{\mathrm{jv}} u'_2$, this is by hypothesis, if $u_3 \equiv^1_{\mathrm{jv}} u_3$ by *i.h.*

Case $u = \lambda y.s \equiv^1_{\mathrm{jv}} \lambda y.s'$ where $s \equiv^1_{\mathrm{jv}} s'$. Then, $t\{x\backslash\backslash u\} = t\{x/u\}$ and $t\{x\backslash\backslash u'\} = t\{x/u'\}$. We can show by a straightforward induction that for any value v such that $v \equiv^1_{\mathrm{jv}} v'$ (necessarily an abstraction), we have $t\{x/v\} \equiv_{\mathrm{jv}} t\{x/v'\}$. □

Lemma 3.93. *$\equiv_{\mathrm{jv}}$ is a strong bisimulation for $\to_{\mathrm{djv}}$, $\to_{\mathrm{ev}}$ and $\to_{\mathrm{sv}}$.*

Proof. We show that if $t_1 \to_{\mathrm{djv}} t_2$ and $t_1 \equiv^1_{\mathrm{jv}} t_1'$, then there is t_2' such that $t_1' \to_{\mathrm{djv}} t_2'$ and $t_2 \equiv_{\mathrm{jv}} t_2'$. From there, the strong bisimulation for the reflexive and transitive closure $t_1 \equiv_{\mathrm{jv}} t_1'$ is obtained by a simple induction.

For $\to_{\mathrm{ev}}$ and $\to_{\mathrm{sv}}$, simply notice that every ev-step is mapped to a ev-step and sv-step to a sv-step.

We reason by induction on $t_1 \equiv^1_{\mathrm{jv}} t_1'$. In each of the base cases, we do a case analysis on $t_1 \to_{\mathrm{djv}} t_2$.

Case $t_1 = s_1(u_1, z_1.s_2)(u_2, z_2.r) \sim_\pi s_1(u_1, z_1.s_2(u_2, z_2.r))$. The cases where $\equiv^1_\pi$ is inside a subterm are straightforward. There are two other subcases.

Subcase $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u_1, z_1.s_2)(u_2, z_2.r) \to_{\mathrm{djv}} \mathtt{D}\langle s_2\{z_1\backslash\backslash t\{x\backslash\backslash u_1\}\}\rangle(u_2, z_2.r) = t_2$. We have:

$$\begin{aligned} t_1' &= \mathtt{D}\langle\lambda x.t\rangle(u_1, z_1.s_2(u_2, z_2.r)) \\ &\to_{\mathrm{djv}} \mathtt{D}\langle s_2(u_2, z_2.r)\{z_1\backslash\backslash t\{x\backslash\backslash u_1\}\}\rangle \\ &\equiv_\pi \mathtt{D}\langle s_2\{z_1\backslash\backslash t\{x\backslash\backslash u_1\}\}(u_2, z_2.r)\rangle \text{ (by lemma 3.92(i))} \\ &\equiv_\pi t_2 \end{aligned}$$

Subcase $t_1 = s_1(u_1, z_1.\mathtt{D}\langle\lambda x.t\rangle)(u_2, z_2.r) \to_{\mathrm{djv}} s_1(u_1, z_1.\mathtt{D}\langle r\{z_2\backslash\backslash t\{x\backslash\backslash u_2\}\}\rangle) = t_2$. We have:

$$t_1' = s_1(u_1, z_1.\mathtt{D}\langle\lambda x.t\rangle(u_2, z_2.r)) \to_{\mathrm{djv}} t_2$$

Case $t_1 = s_2(s_1(u_1, z_1.u_2), z_2.r) \sim_{\mathrm{arg}} s_1(u_1, z_1.s_2(u_2, z_2.r)) = t_2$. The cases where $\equiv^1_{\mathrm{arg}}$ is inside a subterm are straightforward. There are two other subcases.

Subcase $t_1 = \mathtt{D}\langle\lambda x.t\rangle(s_1(u_1, z_1.u_2), z_2.r) \to_{\mathrm{djv}} r\{z_2\backslash\backslash t\{x\backslash\backslash s_1(u_1, z_1.u_2)\}\} = t_2$. We have:

$$t_1' = s_1(u_1, z_1.\mathtt{D}\langle\lambda x.t\rangle(u_2, z_2.r)) \to_{\mathrm{djv}} s_1(u_1, z_1.r\{z_2\backslash\backslash t\{x\backslash\backslash u_2\}\}) = t_2$$

Subcase $t_1 = s_2(\mathtt{D}\langle\lambda x.t\rangle(u_1, z_1.u_2), z_2.r) \to_{\mathrm{djv}} s_2(\mathtt{D}\langle u_2\{z_1\backslash\backslash t\{x\backslash\backslash u_1\}\}\rangle, z_2.r)$. We have:

$$\begin{aligned} t_1' &= \mathtt{D}\langle\lambda x.t\rangle(u_1, z_1.s_2(u_2, z_2.r)) \\ &\to_{\mathrm{djv}} \mathtt{D}\langle s_2(u_2, z_2.r)\{z_1\backslash\backslash t\{x\backslash\backslash u_1\}\}\rangle \\ &\equiv_{\mathrm{arg}} \mathtt{D}\langle s_2(u_2\{z_1\backslash\backslash t\{x\backslash\backslash u_1\}\}, z_2.r)\rangle \text{ (by lemma 3.92(iii))} \\ &\equiv_{\mathrm{arg}} t_2 \end{aligned}$$

Case $t_1 = s_2(u_2, z_2.s_1(u_1, z_1.r)) \sim_{\text{com}} s_1(u_1, z_1.s_2(u_2, z_2.r)) = t_1'$. The cases where $\equiv^1_{\text{com}}$ is inside a subterm are straightforward. There are two other subcases.

Subcase $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u_2, z_2.s_1(u_1, z_1.r)) \to_{\text{djv}} \mathtt{D}\langle s_1(u_1, z_1.r)\{z_2 \backslash\backslash t\{x \backslash\backslash u_2\}\}\rangle = t_2$. We have:

$$\begin{aligned} t_1' &= s_1(u_1, z_1.\mathtt{D}\langle\lambda x.t\rangle(u_2, z_2.r)) \\ &\to_{\text{djv}} s_1(u_1, z_1.\mathtt{D}\langle r\{z_2 \backslash\backslash t\{x \backslash\backslash u_2\}\}\rangle) \\ &\equiv_{\text{com}} \mathtt{D}\langle s_1(u_1, z_1.r\{z_2 \backslash\backslash t\{x \backslash\backslash u_2\}\})\rangle \\ &\equiv_{\text{com}} t_2 \text{ (by lemma 3.92(i))} \end{aligned}$$

Subcase $t_2 = s_2(u_2, z_2.\mathtt{D}\langle\lambda x.t\rangle(u_1, z_1.r)) \to_{\text{djv}} s_2(u_2, z_2.\mathtt{D}\langle r\{z_1 \backslash\backslash t\{x \backslash\backslash u_1\}\}\rangle) = t_1$. This case is symmetric to the previous.

We now analyze the inductive cases of $t_1 \equiv^1_{\text{jv}} t_1'$. We use a case analysis on $t_1 \to_{\text{djv}} t_2$.

Case $t_1 = \lambda x.s_1 \to_{\text{djv}} \lambda x.s_2 = t_2$. Straightforward by *i.h.*

Case $t_1 = s_1(u_1, x.r_1) \to_{\text{djv}} s_2(u_2, x.r_2) = t_2$, where $s_1 \to_{\text{djv}} s_2$, $u_1 \to_{\text{djv}} u_2$ or $r_1 \to_{\text{djv}} r_2$. Straightforward by *i.h.*

Case $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u, y.r) \to_{\text{djv}} r\{y \backslash\backslash t\{x \backslash\backslash u\}\} = t_2$. There are four subcases.

- **Subcase** $t_1' = \mathtt{D}\langle\lambda x.t'\rangle(u, y.r)$. Then $t_1' \to_{\text{djv}} \mathtt{D}\langle r\{y \backslash\backslash t'\{x \backslash\backslash u\}\}\rangle = t_2'$. By lemma 3.92(ii), we have $t'\{x \backslash\backslash u\} \equiv_{\text{jv}} t\{x \backslash\backslash u\}$. By lemma 3.92(iii), we have $t_2' \equiv_{\text{jv}} t_2$.
- **Subcase** $t_1' = \mathtt{D}\langle\lambda x.t\rangle(u', y.r)$. Then $t_1' \to_{\text{djv}} \mathtt{D}\langle r\{y \backslash\backslash t\{x \backslash\backslash u'\}\}\rangle = t_2'$. By two applications of lemma 3.92(iii), we have $t_2' \equiv_{\text{jv}} t_2$.
- **Subcase** $t_1' = \mathtt{D}\langle\lambda x.t\rangle(u, y.r')$. Then $t_1' \to_{\text{djv}} \mathtt{D}\langle r'\{y \backslash\backslash t\{x \backslash\backslash u\}\}\rangle = t_2'$. By lemma 3.92(ii), we have $t_2' \equiv_{\text{jv}} t_2$.
- **Subcase** $t_1' = \mathtt{D}'\langle\lambda x.t\rangle(u, y.r)$. Then $t_1' \to_{\text{djv}} \mathtt{D}'\langle r\{y \backslash\backslash t\{x \backslash\backslash u\}\}\rangle = t_2'$. We have $t_2' \equiv_{\text{jv}} t_2$ directly. □

In the bisimulation, notice that $\equiv_\pi$-equivalences are mapped to $\equiv_\pi$-equivalences, $\equiv_{\text{arg}}$-equivalences to $\equiv_{\text{arg}}$-equivalences and $\equiv_{\text{com}}$-equivalences to $\equiv_{\text{com}}$-equivalences. This is what allows us to define separate strong bisimulations for each of the equivalences, and in particular to consider $\to_{\text{djv}/\equiv_{\text{com}}}$ in lemma 3.91.

Remark 3.94. The relation $\to_{\text{djv}/\equiv_{\text{jv}}}$ allows us to simulate the calculus with permutations $\lambda^\sigma_{\text{v}}$ in λJ_v.

With the addition of the bisimulation to the calculus λJ_v, we obtain a rich equational theory for CbV generalized applications. An **equational theory** is the reflexive, symmetric, transitive and contextual closure of a rewriting relation $\mathscr{R}$. We write $=_{\text{jv}}$ the equational theory of λJ_v with reduction $\to_{\text{djv}/\equiv_{\text{jv}}}$.

Accattoli and Guerrieri [AG22] also define an equational theory $=_{\text{vsub}}$ of $\lambda_{\mathtt{vsub}}$ modulo a strong bisimulation $\equiv_{\text{vsub}}$ on explicit substitutions. It is easy to prove that $\equiv_{\text{jv}}$ simulates $\equiv_{\text{vsub}}$ and vice-versa. From this and the simulations given before, we can show the following:

- There are no terms $M, N \in \mathtt{T}_{ES}$ such that $M =_{\text{vsub}} N$ and $M^\bullet \neq N^\bullet$.
- There are no terms $t, u \in \mathtt{T}_J$ such that $t =_{\text{jv}} u$ and $M^\star \neq N^\star$.

However, there are terms $M, N \in \mathtt{T}_{ES}$ such that $M \neq_{\text{vsub}} N$ but $M^\bullet =_{\text{jv}} N^\bullet$.

Example 3.95. Let $M = x[x/yy]$ and $N = yy$. We have $M \neq_{\text{vsub}} N$: indeed M does not reduce because yy is not a value. Yet, $M^\bullet = \mathtt{I}(\mathtt{I}(y, z_1.y(z_1, z_2.z_2)), x.x) \to_{\text{djv}} \mathtt{I}(y(y, z_2.z_2), x.x) \to_{\text{djv}} y(y, x.x)$ and $N^\bullet = \mathtt{I}(y, z_1.y(z_1, x.x)) \to_{\text{djv}} y(y, x.x)$.

We also conjecture that the inverse is not true.

Conjecture 3.96. There are no two terms $t, u \in \mathtt{T}_J$ such that $t \neq_{\text{jv}} u$ but $t^\star =_{\text{vsub}} u^\star$.

In general, Moggi's identity rule $IN \to N$ is not always respected in $\lambda_{\mathtt{vsub}}$, despite its apparent simplicity. This happens because of the blocking character of CbV reduction in $\lambda_{\mathtt{vsub}}$. This equality always holds in λJ_v, as mentioned on page 165: $\mathtt{I}(u, z.z) \to_{\text{djv}} u$ for any $u \in \mathtt{T}_J$.

By adopting a restricted syntax, compared to calculi with explicit substitutions (the terms $x[x/yy]$ and yy have a unique representation $y(y, x.x)$), as well as non-blocking CbV rules, the calculus λJ_v avoids some of the flaws of $\lambda_{\mathtt{vsub}}$.

To repair Moggi's identity in $\lambda_{\mathtt{vsub}}$, Accattoli and Guerrieri [AG22] suggest to add a "glue" rule $\mathtt{O}\langle\!\langle x \rangle\!\rangle[x/N] \to \mathtt{O}\langle N \rangle$ (where $\mathtt{O}$ is a weak open context). Semantically, this rule is natural, and stays within the realm of CbV because it does not duplicate nor erase N. However, the addition of the glue rule is problematic: it breaks confluence of the relation $\to_{\text{vsub}}$. Accattoli and Guerrieri claim that confluence is retrieved when adding the equivalence $\equiv_{\text{vsub}}$. Yet, they do not give a proof, as proving confluence modulo equivalence is hard.

In λJ_v, there is no need to add such a rule, since Moggi's identity is already valid. The semantics is kept simple, and confluence holds.

We conclude with the following conjectures.

Conjecture 3.97. Let $=_{\text{vsub+glue}}$ be the equational theory of $\lambda_{\mathtt{vsub}}$ equipped with $\equiv_{\text{vsub}}$ and the glue rule.

- There are no terms $M, N \in \mathtt{T}_{ES}$ such that $M \neq_{\text{vsub+glue}} N$, but where $M^\bullet =_{\text{jv}} N^\bullet$.
- There are no terms $t, u \in \mathtt{T}_J$ such that $t \neq_{\text{jv}} u$ but where $t^\star =_{\text{vsub+glue}} u^\star$.

3.7 A Normalizing Strategy for Strong Evaluation

Accattoli, Guerrieri, and Leberle [AGL21] and Accattoli, Condoluci, and Sacerdoti Coen [ACS21] define a normalizing strategy for the CbV calculus with ES, called *external*. This strategy corresponds to the leftmost-outermost strategy of the λ-calculus, which reduces to a strong normal form every term that possesses one, without looping on subterms that could be erased.

However, their definition is complicated, as it resorts to two mutually recursive definitions of contexts (rigid and external), and a specific grammar of "rigid" terms. A difficulty is

that the only abstractions whose body must be reduced are the ones which are not applied. Applied and non-applied abstractions are not evident to distinguish because some that are inside explicit substitutions can be isolated from their argument.

We give new strategies for strong reductions for λJ_v and ΛJ_v. These strategies are remarkably simple, as they execute a transparent leftmost-outermost reduction. They constitute straightforward extensions of weak evaluation, obtained by adding reduction under abstractions. Normalizing strategies for CbN strong evaluation would be defined in exactly the same way, except for the base rules. Moreover, the grammars of strong normal forms are the same in CbN and in CbV, and in the non-distant case represent the fully normal derivations of von Plato. This shows again the advantages of generalized applications.

We prove the normalization property by characterizing the strategies in the quantitative type system $\cap V$, with a special notion of types, taken from [AGL21].

Definition 3.98. The strong normal forms are defined as follows.

$$\begin{aligned} &\textbf{(Neutral normal forms)} && \mathrm{NE}_{\mathrm{djv}} ::= x \mid \mathrm{NE}_{\mathrm{djv}}(\mathrm{NF}_{\mathrm{djv}}, y.\,\mathrm{NE}_{\mathrm{djv}}) \\ &\textbf{(Normal forms)} && \mathrm{NF}_{\mathrm{djv}} ::= x \mid \lambda x.\,\mathrm{NF}_{\mathrm{djv}} \mid \mathrm{NE}_{\mathrm{djv}}(\mathrm{NF}_{\mathrm{djv}}, y.\,\mathrm{NF}_{\mathrm{djv}}) \end{aligned}$$

Definition 3.99. The *distant* **leftmost-outermost value** reduction $\to_{\mathrm{lov}}$ is defined by the following rules.

$$\frac{t \mapsto_{\mathrm{d}\beta_v} t'}{t \to_{\mathrm{lov}} t'} \qquad \frac{t \to_{\mathrm{lov}} t'}{\lambda x.t \to_{\mathrm{lov}} \lambda x.t'} \qquad \frac{t \to_{\mathrm{lov}} t' \qquad t \neq \mathrm{D}\langle \lambda x.s \rangle}{t(u, y.r) \to_{\mathrm{lov}} t'(u, y.r)} \qquad \frac{u \to_{\mathrm{lov}} u' \qquad t \in \mathrm{NE}_{\mathrm{djv}}}{t(u, y.r) \to_{\mathrm{lov}} t(u', y.r)}$$

$$\frac{r \to_{\mathrm{lov}} r' \qquad t \in \mathrm{NE}_{\mathrm{djv}}}{t(u, y.r) \to_{\mathrm{lov}} t(u, y.r')}$$

Lemma 3.100. *Let $t \in \mathrm{T}_J$. Then $t \in \mathrm{NF}_{\mathrm{djv}}$ iff t is in* lov*-nf.*

Proof. We start with soundness: $t \in \mathrm{NF}_{\mathrm{djv}} \implies t$ is in lov-nf. We show the following two stronger properties:

(i) For all $t \in \mathrm{NE}_{\mathrm{djv}}$, t does not have an abstraction shape and t is in lov-nf.

(ii) For all $t \in \mathrm{NF}_{\mathrm{djv}}$, t is in lov-nf.

The proof is by simultaneous induction on $t \in \mathrm{NE}_{\mathrm{djv}}$ and $t \in \mathrm{NF}_{\mathrm{djv}}$.

Case $t = x$. Both statements are straightforward.

Case $t = \lambda x.s \in \mathrm{NF}_{\mathrm{lov}}$ with $s \in \mathrm{NF}_{\mathrm{djv}}$. By *i.h.* (ii), s is in lov-nf. Hence so is $\lambda x.s$.

Case $t = s(u, y.r)$ where $s \in \mathrm{NE}_{\mathrm{djv}}$ and $u, r \in \mathrm{NF}_{\mathrm{djv}}$. By *i.h.* (i), s is lov-normal and does not have an abstraction shape. By *i.h.* (ii), u and r are lov-normal. Hence, t is lov-normal. Moreover, if $t \in \mathrm{NE}_{\mathrm{lov}}$, then $r \in \mathrm{NE}_{\mathrm{lov}}$ and by *i.h.* (i), r does not have an abstraction shape, so that t does not either.

Now, completeness: t is in lov-nf $\implies$ $t \in \mathrm{NF}_{\mathrm{djv}}$. We show a stronger property: For all t,

(i) If t does not have an abstraction shape and t is in lov-nf, then $t \in \mathrm{NE}_{\mathrm{djv}}$; and

(ii) If t is in lov-nf, then $t \in \mathrm{NF}_{\mathrm{djv}}$.

The proof is by induction on t.

Case $t = x$. We have $x \in \mathrm{NE}_{\mathrm{djv}}$ and $x \in \mathrm{NF}_{\mathrm{djv}}$.

Case $t = \lambda x.s$. Item (i) does not apply. Suppose t is in djv-nf. Then so is s. By the *i.h.* (ii), $s \in \mathrm{NF}_{\mathrm{djv}}$. Hence $\lambda x.s \in \mathrm{NF}_{\mathrm{djv}}$.

Case $t = s(u, x.r)$. Suppose t is in djv-nf. Then s, u, r are in djv-nf, hence $u \in \mathrm{NF}_{\mathrm{lov}}$ and $r \in \mathrm{NF}_{\mathrm{lov}}$, by *i.h.* (i). The subterm s does not have an abstraction shape, otherwise t would be a $\mathrm{d}\beta_{\mathrm{v}}$-redex, thus $s \in \mathrm{NE}_{\mathrm{djv}}$, by the *i.h.* (i). Therefore, $t \in \mathrm{NF}_{\mathrm{djv}}$ and (i) is proved. Moreover, suppose t does not have an abstraction shape. Then the same holds for r. By *i.h.* (i) $r \in \mathrm{NE}_{\mathrm{lov}}$. Hence $t \in \mathrm{NE}_{\mathrm{djv}}$ and (i) is proved. □

Property 3.101 (Diamond). *The reduction* $\longrightarrow_{\mathrm{lov}}$ *is diamond.*

Proof. The only branching case is with a term $t = s(u, y.r)$, where $s \in \mathrm{NE}_{\mathrm{djv}}$ and $u \longrightarrow_{\mathrm{lov}} u'$, $r \longrightarrow_{\mathrm{lov}} r'$. Both terms $s(u', y.r)$ and $s(u, y.r')$ can be reduced in one lov-step to $s(u', y.r')$. □

The logical characterization of the leftmost-outermost value reduction is done again using the type system $\cap V$, with another restriction on types, as in the case of CbN [Kri02]. The normalizing terms are the ones that can be assigned a *shrinking* type derivation. Once again, this requirement can be verified locally on the last sequent of the derivation. The definition is usually given using a polarity on the occurrences of types [AGK20], but a grammar of types can be given directly, as done by Accattoli, Guerrieri, and Leberle [AGL21].

Definition 3.102 (Shrinking types). We distinguish **left** and **right shrinking types** σ^{l} and σ^{r}.

(Right shrinking types)	$\sigma^{\mathrm{r}}, \tau^{\mathrm{r}}$	::=	$a \in BTV \mid \mathscr{M}^{\mathrm{r}} \mid \mathscr{M}^{\mathrm{l}} \to \sigma^{\mathrm{r}}$
(Right shrinking multitypes)	$\mathscr{M}^{\mathrm{r}}, \mathscr{N}^{\mathrm{r}}$	::=	$[\sigma_i^{\mathrm{r}}]_{i \in I}$ where I is a non-empty finite set
(Left shrinking types)	$\sigma^{\mathrm{l}}, \tau^{\mathrm{l}}$	::=	$a \in BTV \mid \mathscr{M}^{\mathrm{l}} \mid \mathscr{M}^{\mathrm{r}} \to \sigma^{\mathrm{l}}$
(Left shrinking multitypes)	$\mathscr{M}^{\mathrm{l}}, \mathscr{N}^{\mathrm{l}}$	::=	$[\sigma_i^{\mathrm{l}}]_{i \in I}$ where I may be empty

A context Γ is *left shrinking* when for all $x : \mathscr{M} \in \Gamma$, $\mathscr{M}$ is left shrinking.

Definition 3.103 (Shrinking derivation). A derivation $\Gamma \Vdash t : \sigma$ is **shrinking** if Γ is left shrinking and τ is a right shrinking type.

Subject reduction as well as expansion were shown earlier to hold for the full $\longrightarrow_{\mathrm{djv}}$ relation. Thus, there are only two things we need to prove to achieve the characterization:

1. A $\longrightarrow_{\text{lov}}$-step diminishes the size of a shrinking derivation.
2. Terms in NF_{djv} are typable with a shrinking derivation.

First, we need a lemma on neutral normal forms.

Lemma 3.104. *Let $\Gamma \Vdash t : \mathscr{M}$ with Γ left shrinking and $t \in \text{NE}_{\text{djv}}$. Then $\mathscr{M}$ is left shrinking.*

Proof. By induction on NE_{djv}.

Case $t = x$. Then the derivation is

$$\frac{}{x : \mathscr{M} \vdash x : \mathscr{M}}$$

By definition, $\mathscr{M}$ is left shrinking.

Case $t = s(u, y.r)$, where $s \in \text{NE}_{\text{djv}}$ and $u, r \in \text{NF}_{\text{djv}}$. Then the derivation ends with

$$\frac{\Gamma_s \Vdash s : [\mathscr{N}_1 \longrightarrow \mathscr{N}_2] \qquad \Gamma_u \Vdash u : \mathscr{N}_1 \qquad \Gamma_r; y : \mathscr{N}_2 \Vdash r : \mathscr{M}}{\Gamma_s \uplus \Gamma_u \uplus \Gamma_r \Vdash s(u, y.r) : \mathscr{M}}$$

By *i.h.* $[\mathscr{N}_1 \longrightarrow \mathscr{N}_2]$ is left shrinking so $\mathscr{N}_2$ is left shrinking. Then $\Gamma_r; y : \mathscr{N}_2$ is left shrinking and we can use the *i.h.* on $\Gamma_r; y : \mathscr{N}_2 \Vdash r : \mathscr{M}$. □

Lemma 3.105 (Weighted subject reduction). *Let $\Gamma \Vdash^{n_1}_{\cap V} t_1 : \sigma$ with Γ left shrinking and, if $t_1 \neq \mathtt{D}\langle \lambda x.t \rangle$, σ right shrinking. Let $t_1 \longrightarrow_{\text{djv}} t_2$. Then $\Gamma \Vdash^{n_2}_{\cap V} t_2 : \sigma$ with $n_1 > n_2$.*

Proof. By induction on $t_1 \longrightarrow_{\text{djv}} t_2$. The existence of the derivation of t_2 is given by lemma 3.56. We focus on showing that the stronger induction hypothesis where the size of derivation decreases can be applied.

Case $t_1 \mapsto_{\text{djv}} t_2$. By lemma 3.55, where the size of the derivation decreases for any typing.

Case $t_1 = t(u, y.r)$, and the reduction is internal. The derivation of t_1 ends with an (APP)-rule with premises: $\Gamma_t \Vdash^{n_t} t : [\mathscr{M} \longrightarrow \mathscr{N}]$, $\Gamma_u \Vdash^{n_u} u : \mathscr{M}$ and $\Gamma_r; x : \mathscr{N} \Vdash^{n_r} r : \sigma$ such that $n_1 = 1 + n_t + n_u + n_r$. Moreover, Γ_t, Γ_u and Γ_r are left shrinking by hypothesis. There are several subcases:

- **Subcase** $t_1 = t(u, y.r) \longrightarrow_{\text{djv}} t'(u, y.r) = t_2$, where $t \longrightarrow_{\text{djv}} t'$. Since $t \neq \mathtt{D}\langle \lambda x.s \rangle$, we can apply the *i.h.* and obtain $n_t > n_{t'}$, so $n_1 > n_2$.
- **Subcase** $t_1 = t(u, y.r) \longrightarrow_{\text{djv}} t(u', y.r) = t_2$, where $u \longrightarrow_{\text{djv}} u'$. Since $t \in \text{NE}_{\text{djv}}$, by lemma 3.104 $[\mathscr{M} \longrightarrow \mathscr{N}]$ is left shrinking so that $\mathscr{M}$ is right shrinking. We can apply the *i.h.* and obtain $n_u > n_{u'}$, so $n_1 > n_2$.

Subcase $t_1 = t(u, y.r) \to_{\mathrm{djv}} t(u, y.r') = t_2$, where $r \to_{\mathrm{djv}} r'$. Since $t \neq \mathtt{D}\langle \lambda x.s \rangle$, by hypothesis σ is right shrinking. Moreover, since $t \in \mathrm{NE}_{\mathrm{djv}}$, by lemma 3.104 $[\mathscr{M} \to \mathscr{N}]$ is left shrinking so that $\mathscr{N}$ is left shrinking. Then $\Gamma_r; y : \mathscr{N}$ is left shrinking. We can apply the *i.h.* and obtain $n_r > n_{r'}$, so $n_1 > n_2$.

Case $t_1 = \lambda x.t \to_{\mathrm{djv}} \lambda x.t' = t_2$. By hypothesis, we have $\sigma = [\mathscr{M}_i \to \sigma_i]_{i \in I}$. Since σ is right shrinking, I is not empty. Thus, we have $\Gamma_i; x : \mathscr{M}_i \Vdash^{n_i} t : \sigma_i$ for $i \in I$, where $\Gamma = \uplus_{i \in I} \Gamma_i$ and $n_1 = \sum_{i \in I} n_i$. By definition, every Γ_i and $\mathscr{M}_i$ are left shrinking, and σ_i is right shrinking. By the *i.h.*, we have $\Gamma_i; x : \mathscr{M}_i \Vdash^{n_i'} t' : \sigma_i$ for $i \in I$ such that $n_i > n_i'$. We can build a derivation of size $n_2 = \sum_{i \in I} n_i' < \sum_{i \in I} n_i = n_1$. □

Lemma 3.106. *Let $t \in \mathrm{NF}_{\mathrm{djv}}$. Then t is typable in $\cap V$ with a shrinking derivation.*

Proof. We show the following statements by mutual induction on $\mathrm{NF}_{\mathrm{djv}}$ and $\mathrm{NE}_{\mathrm{djv}}$.

(i) Let $t \in \mathrm{NE}_{\mathrm{djv}}$. Then for all σ left shrinking, there is Γ left shrinking such that $\Gamma \Vdash_{\cap V} t : \sigma$.

(ii) Let $t \in \mathrm{NF}_{\mathrm{djv}}$. Then there are Γ left shrinking and σ right shrinking such that $\Gamma \Vdash_{\cap V} t : \sigma$.

Case $t = x$. For all σ, there is a derivation

$$\frac{}{x : [\sigma] \vdash x : \sigma}$$

with σ left shrinking by hypothesis, which concludes item (i). Item (ii) holds by taking σ different from $[\,]$.

Case $t = \lambda x.s$, where $s \in \mathrm{NF}_{\mathrm{djv}}$. Item (i) does not apply. By *i.h.* (ii), there are derivations $\Gamma_i; x : \mathscr{M}_i \Vdash s : \tau_i$ with all Γ_i and $\mathscr{M}_i$ left shrinking and τ_i right shrinking. Then there is a derivation $\Gamma \Vdash \lambda x.s : [\mathscr{M}_i \to \tau_i]_{i \in I}$. We have $\sigma = [\mathscr{M}_i \to \tau_i]$ right shrinking.

Case $t = s(u, y.r)$, where $s \in \mathrm{NE}_{\mathrm{djv}}$ and $u, r \in \mathrm{NF}_{\mathrm{djv}}$. By *i.h.* (ii), there are shrinking derivation $\Gamma_u \Vdash u : \mathscr{N}_1$ and $\Gamma_r; y : \mathscr{N}_2 \Vdash r : \sigma$. The type $[\mathscr{N}_1 \to \mathscr{N}_2]$ is left shrinking because $\mathscr{N}_1$ is right shrinking and $\mathscr{N}_2$ left shrinking. Thus, we can apply *i.h.* (i) and get a derivation $\Gamma_s \Vdash s(u, y.r) : \mathscr{N}_1 \to \mathscr{N}_2$. We conclude item (ii) by rule (APP). In case (i), we have $r \in \mathrm{NE}_{\mathrm{djv}}$, so that by *i.h.* for any $\mathscr{M}$ we have $\Gamma_r; y : \mathscr{N}_2 \Vdash r : \mathscr{M}$, and thus a derivation $\Gamma_s \uplus \Gamma_u \uplus \Gamma_r \Vdash s(u, y.r) : \mathscr{M}$. □

Theorem 3.107 (Logical characterization of $\to_{\mathrm{lov}}$-normalization). *Let $t \in \mathtt{T}_J$. Then t is typable iff t is* lov*-normalizable.*

Proof. Soundness is by lemma 3.105, and the fact that the size of the derivation diminishes at each lov-steps. Completeness is by lemma 3.106 and the subject expansion lemma 3.61. □

Property 3.108 (Normalization for $\longrightarrow_{\text{lov}}$). *Let* $t \longrightarrow^*_{\text{djv}} u$ *and* $u \in \text{NF}_{\text{djv}}$. *Then* $t \longrightarrow^*_{\text{lov}} u$.

Proof. Similar as property 3.67. However, since the calculus is confluent, proving that t necessarily lov-normalizes to the same term u. □

A non-distant definition of the normalizing strategy is possible, and even simpler.

Definition 3.109. The local strong normal forms are as follows.

$$\text{NF}_{\text{jv}} ::= x \mid \lambda x.\, \text{NF}_{\text{jv}} \mid x(\text{NF}_{\text{jv}}, y.\, \text{NF}_{\text{jv}})$$

Definition 3.110. The *local* **leftmost-outermost value** reduction $\longrightarrow_{\text{llov}}$ is defined by the following rules.

$$\frac{t \longmapsto_{\{\beta\text{v},\pi\}} t'}{t \longrightarrow_{\text{llov}} t'} \qquad \frac{t \longrightarrow_{\text{llov}} t'}{\lambda x.t \longrightarrow_{\text{llov}} \lambda x.t'} \qquad \frac{u \longrightarrow_{\text{llov}} u'}{x(u, y.r) \longrightarrow_{\text{llov}} x(u', y.r)} \qquad \frac{r \longrightarrow_{\text{llov}} r'}{x(u, y.r) \longrightarrow_{\text{llov}} x(u, y.r')}$$

This time, there is no side-condition on the first term of the application in the last two rules, because it can only be a variable, thanks to π-reduction. There is also no rule to go left of an application. This can be dispensed by applying permutations at root. A term $t(u, x.r)$ can always be reduced with π to a term $v(u', y.r')$. If v is a variable we apply one of the two inductive rules. If v is an abstraction we simply apply βv.

Lemma 3.111. *Let* $t \in \Lambda_J$. *Then* $t \in \text{NF}_{\text{jv}}$ *iff* t *is in* llov*-nf.*

Proof. It is immediate by induction that a term of NF_{jv} does not llov-reduce. Let t be an llov-normal term. By induction on t:

Case $t = x$. Implies $t \in \text{NF}_{\text{jv}}$.

Case $t = \lambda x.s$. By *i.h.*

Case $t = s(u, y.r)$, where u and r are llov-normal. Since t does not jv-reduce, s is not an abstraction nor an application. Then $t = x(u, y.r)$. We conclude since $u, r \in \text{NF}_{\text{jv}}$ by *i.h.* □

The characterization and normalization theorems follow in a similar way as for the distant version.

3.8 Conclusion

In this chapter, we have given a refined study of normalization of generalized applications centered around the notion of solvability. We have first adapted existing definitions and properties to our CbN calculus. The study of CbN solvability prepares the one of CbV, notably thanks to the similar reduction rules in both policies. This resemblance enables us to highlight the differences between the characterizations of CbN and CbV solvability. We have also extended the operational study of CbV generalized applications by defining a strong bisimulation on $\mathtt{T}_J$-terms, as well as a normalizing strategy for strong reduction.

Call-by-value solvability Finding good operational formalisms for CbV is an active topic of research (see [AG16]), with new insights from linear logic [Acc15; GPD17] and the sequent calculus [HZ09]. The calculus λJ_v holds a singular place, thanks to its natural way to deal with stuck redexes and the non-blocking character of βv-reduction.

Call-by-value solvability is captured operationally in two other calculi: $\lambda_{\mathrm{v}}^{\sigma}$ of Carraro and Guerrieri [CG14], relying on permutation rules, and $\lambda_{\mathtt{vsub}}$ of Accattoli and Paolini [AP12]. Let us compare our characterization to these ones.

The solving relation has two principal advantages compared to $\lambda_{\mathrm{v}}^{\sigma}$. The first one is the possibility to avoid independent permutation rules by adopting distance. This is useful since permutation rules are not measured quantitatively by intersection types, and seems difficult to implement in the calculus $\lambda_{\mathrm{v}}^{\sigma}$. Then, this calculus cannot be used for a quantitative analysis of CbV. The second advantage is that generalized applications exhibit normal forms of the shape $\lambda\vec{x}.y(u_1, y_1.r_1)\dots(u_n, y_n.r_n)$. This shape is the same as for CbN, and is reminiscent of the shape of normal terms in λ (both with different conditions on the subterms). On the contrary, normal forms of $\lambda_{\mathrm{v}}^{\sigma}$ are made complex by constructs of the shape $(\lambda x.t)(yu_1 \dots u_n)$. Yet, normal forms are a central notion when dealing with solvability in particular.

Normal forms in $\lambda_{\mathtt{vsub}}$ [AP12] do not contain function applications such as in $\lambda_{\mathrm{v}}^{\sigma}$ above. The solving normal forms in this calculus are similar to the ones of our distant solving reduction. However, an advantage of generalized applications is that π-permutation can be used separately, to obtain very elementary normal forms, of the shape $\lambda\vec{x}.y(u, z.r)$. The main drawback of $\lambda_{\mathtt{vsub}}$ is its lower level of abstraction: λJ_v and ΛJ_v allow us to study foundational concepts of CbV while keeping a level of abstraction close to the λ-calculus. Instead, $\lambda_{\mathtt{vsub}}$ deals with an explicit treatment of substitution, and two computational rules. Some practical matters blur the study of solvability, such as in [AG22], where some important properties do not hold for the full semantics, but only when variables are not substituted.

Solvability in the $\lambda_{\mathtt{vsub}}$-calculus has been captured logically by means of a quantitative type system by Accattoli and Guerrieri [AG22], a characterization that we have adapted to our setting. In the CbV type systems, solvability does not correspond to typability alone, but to typability with a solvable type. Considering a λ-calculus with pattern matching, Bucciarelli, Kesner, and Ronchi Della Rocca [BKR21] show that solvability in this calculus is captured by typability *and* inhabitation. We would like to know if this elegant solution extends to CbV.

Meaninglessness in CbV Now that CbV solvability is better understood, it appears that this notion does not correspond to CbN solvability in spirit. In CbN, solvability identifies *meaningless* terms, which can all be equated in a consistent theory of terms. The *genericity lemma* makes this property formal by specifying that in any normalizing computation, we can replace an unsolvable term by any other term.

Only a *partial* genericity lemma [GN16] can hold for CbV solvability, where the *order* (the number of abstractions on top of a term) matters. Take for instance the normalizing reduction $(\lambda z.x)(\lambda y.\Omega, y.y) \to_{\beta_{\mathrm{v}}} x$. The term $\lambda y.\Omega$ is unsolvable, but replacing it with an unsolvable of lesser order, such as Ω, gives rise to an infinite computation $(\lambda z.x)(\Omega, y.y) \to_{\beta_{\mathrm{v}}} \delta(\delta, z.y)$.

Meaninglessness in CbV is then still to be defined. Kennaway, van Oostrom, and de Vries [KvOdV96] present three axioms for meaninglessness, from which genericity follows. These axioms hold for potential valuability in λJ_v and ΛJ_v, but one of them fails for solvability. However, it is not clear whether genericity can be deduced from these axioms for our setting. In $\lambda_{\mathtt{vsub}}$, the situation is the same, according to Accattoli and Guerrieri [AG22]. They also prove that there are theories where potentially valuable terms can be consistently equated, on the contrary to solvable terms.

This gives the impression that the correct notion of meaningfulness is given by potential valuability (renamed *scrutability* by Accattoli and Guerrieri). To confirm this intuition, a genericity lemma should be proved. A possibility is to adapt the proof of Kennaway, van Oostrom, and de Vries [KvOdV96] to extensions of the λ-calculus. Another is to try to adapt the simple proof of genericity for the CbN λ-calculus given by Takahashi [Tak94], or the one of Kuper [Kup95] which takes advantage of the leftmost-outermost reduction.

A last argument in favor of potential valuability is the following. In CbN, the simplest and original system of intersection types captures head normalization and solvability, and other intersection type systems are derived by refinements. In CbV instead, the core type system captures potential valuability, and some restrictions are needed for solvability.

While solvability turns out not to correspond to meaninglessness in CbV, the notion is still interesting in its own right. It is a powerful property: a solvable term can be equated to any other term, given a suitable context. Potential valuability instead is tied with *weak* evaluation: a term is guaranteed to reduce to a value, but that value itself may diverge under an abstraction when considering strong evaluation. On the operational level, the CbV solving relation is an interesting intermediate between weak reduction and full reduction of terms, like head reduction is in the CbN λ-calculus. The solving one does not force divergence, while full reduction also reduces erasable subterms appearing as arguments of a variable.

A further open problem is to find a fully abstract model for the CbV λ-calculus. We would like to see whether generalized applications help in this quest. In particular, it would be interesting to understand CbV approximation for generalized applications, CbV Lévy-Longo trees [DG01] based on weak evaluation, and possibly CbV Böhm trees [Bar84; KMP20] based on the solvable reduction. We believe that the uncomplicated structure of jv-normal forms makes generalized applications a tool of choice to define trees. We would then like to see if one of those definitions helps in revisiting separability [Pao01] in the CbV setting.

Unlike our approach, which characterizes solvability in a calculus with an adequate semantics, García-Pérez and Nogueira [GN16] are concerned with Plotkin's original calculus. They define CbV solvability from the operational viewpoint, thus changing the semantical

model, and identify it to convertibility (as is usual) *plus* freezability. A partial genericity lemma holds for this notion of solvability. They fail to give an operational characterization of this new notion of CbV solvability, however, it might be easier to express inside generalized applications. We can also wonder what a notion of solvability defined from the operational semantics of ΛJ or ΛJ_v with π would be.

Abstract machines and relation to ANF In the introduction, we discussed how calculi with generalized applications equipped with permutation π implement sharing of applications and the search for a redex, making them an intermediate between the λ-calculus and abstract machines.

How do we obtain an abstract machine from a calculus with generalized applications? Every *transition* (reduction step) of a machine should be executed with elementary operations. Substitution, in particular, is delayed and done one occurrence at a time, on the variable under focus [ABM14]. Therefore, the principal missing ingredient to obtain an abstract machine from a calculus with rule π is an explicit treatment of substitutions that *linearizes* them.

The first concrete implementation of generalized applications to consider is weak-head evaluation on closed terms, that is adopted by general-purpose functional languages. Non-eager evaluation is implemented with CbNeed rather than CbN, to avoid code duplication. Adapting generalized applications to CbNeed remains future work. Thus, let us consider call-by-value.

In CbV (and CbNeed), rule π is quantitatively sound. Yet another simplification of the terms can be proposed, relying on the rule arg, defined as an equivalence in section 3.6.2.

$$t_2(t_1(u_1, x.u_2), y.r) \mapsto_{\text{arg}} t_1(u_1, x.t_2(u_2, y.r))$$

Why is this rule interesting? The (π, arg)-normal forms give a simpler grammar of terms, which is stable by βv-reduction. In that grammar, all applications are of the shape $v_1(v_2, x.r)$. Applications are always a value applied to a value and are named. This reveals the strong link between generalized applications and administrative normal forms [SF93; Fla+93] (or the closely related monadic languages [BKR98]). In ANF, the same restrictions on applications hold: all applications are made of a value applied to values, and are shared over a let-binding. Generalized eliminations could be understood as a proof-theoretical foundation of ANF, which were devised syntactically by simplifications of CPS.

Accattoli, Condoluci, Guerrieri, and Sacerdoti Coen [Acc+19] revisit ANF in a call-by-value calculus with ES, to derive simple and complexity-efficient abstract machines. They use a translation of terms with ES that they call *crumbling* to obtain the specific shape of ANF. We expect a CbV abstract machine for generalized applications to be similar to the crumbling machine of [Acc+19], with less overhead on the search for a redex.

In our case though, going from arbitrary terms with generalized applications to the restricted ANF form is very natural: it corresponds to a preliminary (π, arg)-full normalization. This preprocessing is even useful in more abstract studies of CbV, as it does not influence the qualitative and quantitative semantics of the reduction, but allow for a much simpler grammar of terms and normal forms.

One difference between ANF/crumbles and our grammar of terms, is in *tail calls*. The former languages accept tail calls, that are not named. In generalization applications, every

application is named, and tail calls are represented with a dummy continuation $z.z$. This feature is important, as it enables all terms to be of the shape $\mathtt{D}\langle v \rangle$. In other words, every term can be assimilated to a value surrounded by an environment.

In the literature and in practice, ANFs is used as intermediate compiler representation, alternative to CPS. ANFs adopt a direct style, rather than continuation-passing, which avoids the long terms of CPS, as well as bureaucratic reductions. In response to the long-standing debate between ANF and CPS [App91; Ken07] (see a summary in [Con+19]), Maurer, Downen, Ariola, and Jones [Mau+17] suggested using a direct style representation with explicit *join points*, while Cong, Osvald, Essertel, and Rompf [Con+19] propose a direct style representation with possibilities to perform CPS selectively. We would like to investigate generalized applications as an intermediate representation, and see in particular how it fits into the above. For this, extending the grammar of terms and the set of conversion rules to manage other constructors is necessary.

CHAPTER 4

A Quantitative Call-by-Name Calculus with Generalized Applications

In this chapter, we discuss the theory of the CbN variant of ΛJ called λJ_n, which uses distance based on rule p2 instead of π. Some properties of the calculus are given in section 4.2: termination of simply typed terms and normal forms, confluence and the subformula property. An inductive definition of strong normalization is given in section 4.3.

The calculus ΛJ is not quantitatively well-behaved, a concrete example of failure of subject reduction is given in section 4.4.3. We show that, on the contrary, it is the case for λJ_n by giving a non-idempotent intersection type system for the calculus in section 4.4.

Qualitatively, we show that strong normalization is preserved with respect to the λ-calculus (with explicit substitutions) in section 4.5. Yet, we need to define a different translation to explicit substitutions, as the usual one creates divergence. We also prove that the choice of distance does not influence strong normalization, as it is equivalent to a calculus with β and p2 separate (section 4.6.2).

We finish by equating strong normalization of the new and the original CbN calculus λJ_n and ΛJ in section 4.6.3. Thus, the changes to the calculus justified by the quantitative model do not affect qualitative properties. For this proof, we give a new inductive definition of strong normalization for $\to_{\mathrm{jn}}$.

4.1 Towards a Call-by-Name Operational Semantics

The syntax of $\mathtt{T}_J$ can be equipped with different rewriting rules. We use the generic notation $\mathtt{T}_J[\mathscr{R}]$to denote the calculus given by the syntax $\mathtt{T}_J$ equipped with the reduction relation $\to_{\mathscr{R}}$.

Now, if we consider $t_0 \coloneqq t(u, y.\lambda x.s)(u', z.r')$ in the calculus $\mathtt{T}_J[\beta]$, we can see that the term t_0 is stuck since the subterm $\lambda x.s$ is not close to u'. This is when rule π, plays the role of an unblocker of β-redexes:

$$t_0 \to_\pi t(u, y.(\lambda x.s)(u', z.r')) \to_\beta t(u, y.r'\{z/s\{x/u'\}\})$$

More generally, given $t \coloneqq \mathtt{D}\langle\lambda x.s\rangle(u, y.r)$ with $\mathtt{D} \neq \Diamond$, a sequence of π-steps reduces the term t above to $\mathtt{D}\langle(\lambda x.s)(u, y.r)\rangle$. A further β-step produces $\mathtt{D}\langle r\{y/s\{x/u\}\}\rangle$. So, the original ΛJ-

calculus, which is exactly $\mathtt{T}_J[\beta,\pi]$, has a derived notion of distant β rule, *based on* π. This rule $\mathrm{d}\beta\pi$ is specified as follows.

$$\mathtt{D}\langle\lambda x.s\rangle(u, y.r) \mapsto_{\mathrm{d}\beta\pi} \mathtt{D}\langle r\{y/s\{x/u\}\}\rangle \tag{4.1}$$

Still, we will not reduce as in (4.1) because such rule, as well as π itself, does not admit a quantitative semantics (see section 4.4.3). We then choose to unblock β-redexes with rule p2 given in section 3.1.2 instead:[1]

$$t(u, y.\lambda x.s) \mapsto_{\mathrm{p2}} \lambda x.t(u, y.s)$$

We retrieve rule $\mathrm{d}\beta$ by integrating p2 inside β:

$$\mathtt{D}\langle\lambda x.t\rangle(u, y.r) \to_{\mathrm{d}\beta} r\{y/t\{x/\mathtt{D}\langle u\rangle\}\}$$

Note that since the free variables in u cannot be captured by $\mathtt{D}$, the right-hand term is equal to $r\{y/\mathtt{D}\langle t\{x/u\}\rangle\}$.

Comparing the two rules $\mathrm{d}\beta\pi$ and $\mathrm{d}\beta$ gives a first intuition on why the first one is not quantitatively correct. In the rule $\mathrm{d}\beta\pi$, the distant context is put on the exterior of the two substitutions: a unique copy is kept, which is independent from the number of occurrences of y in r. This is a CbV behavior, that does not erase or duplicate computations. On the contrary, the distant context may be erased or duplicated in rule $\mathrm{d}\beta$, according to the number of occurrence of y in r. This is the situation that was already described in section 3.1.

In summary, applying a permutation π does not preserve the length of reduction to normal form in a CbN setting. Therefore, this semantics is not sound for a resource-aware model, such as the one given by a quantitative type system. In practice, subject reduction does not hold for (a rule relying on) π, as is shown in section 4.4.3.

4.2 Some (Un)typed Properties of λJ_n

Lemma 4.1. *The grammar* $\mathrm{NF}_{\mathrm{djn}}$ *characterizes* djn*-normal forms. Notice that the grammar is exactly the same as the one for* $\mathrm{NF}_{\mathrm{djv}}$.

$$\mathrm{NF}_{\mathrm{djn}} ::= x \mid \lambda x.\,\mathrm{NF}_{\mathrm{djn}} \mid \mathrm{NE}_{\mathrm{djn}}(\mathrm{NF}_{\mathrm{djn}}, x.\,\mathrm{NF}_{\mathrm{djn}})$$
$$\mathrm{NE}_{\mathrm{djn}} ::= x \mid \mathrm{NE}_{\mathrm{djn}}(\mathrm{NF}_{\mathrm{djn}}, x.\,\mathrm{NE}_{\mathrm{djn}})$$

Proof. We start with soundness: $t \in \mathrm{NF}_{\mathrm{djn}} \implies t$ is in djn-nf. We show the following two stronger properties:

(i) For all $t \in \mathrm{NE}_{\mathrm{djn}}$, t does not have an abstraction shape and t is in djn-nf.

(ii) For all $t \in \mathrm{NF}_{\mathrm{djn}}$, t is in djn-nf.

The proof is by simultaneous induction on $t \in \mathrm{NE}_{\mathrm{djn}}$ and $t \in \mathrm{NF}_{\mathrm{djn}}$.

[1] Rule p2 is used in [EP03; EFP06] along with two other permutation rules p1 and p3 to reduce $\mathtt{T}_J$-terms to a fragment isomorphic to natural deduction.

First, the cases relative to (i).

Case $t = x$. A variable x does not have an abstraction shape and is in djn-nf.

Case $t = s(u, x.r)$, with $s, r \in \mathrm{NE}_{\mathrm{djn}}$ and $u \in \mathrm{NF}_{\mathrm{djn}}$. The term t does not have an abstraction shape (because r does not have an abstraction shape, due to *i.h.* (i)). The term t is in djn-nf because s, u, r are in djn-nf (due to *i.h.* (i),(ii)) and because t itself is not a djn-redex (since s does not have an abstraction shape, by *i.h.* (i)).

Next, the cases relative to (ii).

Case $t = x$. A variable x is in djn-nf.

Case $t = \lambda x.s$, with $s \in \mathrm{NF}_{\mathrm{djn}}$. By *i.h.* (ii), s is in djn-nf. Hence so is $\lambda x.s$.

Case $t = s(u, x.r)$, with $s \in \mathrm{NE}_{\mathrm{djn}}$ and $u, r \in \mathrm{NF}_{\mathrm{djn}}$. The term t is in djn-nf because s, u, r are in djn-nf (due to *i.h.* (i),(ii)) and because t itself is not a djn-redex (since s does not have an abstraction shape, by *i.h.* (i)).

Now, completeness: t is in djn-nf $\implies t \in \mathrm{NF}_{\mathrm{djn}}$. We show a stronger property: For all t,

(i) If t does not have an abstraction shape and t is in djn-nf, then $t \in \mathrm{NE}_{\mathrm{djn}}$; and

(ii) If t is in djn-nf, then $t \in \mathrm{NF}_{\mathrm{djn}}$.

The proof is by induction on t.

Case $t = x$. We have $x \in \mathrm{NE}_{\mathrm{djn}}$ and $x \in \mathrm{NF}_{\mathrm{djn}}$.

Case $t = \lambda x.s$. Part (i) is trivial. Suppose t is in djn-nf. Then so is s. By the *i.h.* (i), $s \in \mathrm{NF}_{\mathrm{djn}}$. Hence $\lambda x.s \in \mathrm{NF}_{\mathrm{djn}}$.

Case $t = s(u, x.r)$. Suppose t is in djn-nf. Then s, u, r are in djn-nf, hence $u \in \mathrm{NF}_{\mathrm{djn}}$ and $r \in \mathrm{NF}_{\mathrm{djn}}$, by *i.h.* (i). The subterm s does not have an abstraction shape, otherwise t would be a dβ-redex, thus $s \in \mathrm{NE}_{\mathrm{djn}}$, by the *i.h.* (i). Therefore, $t \in \mathrm{NF}_{\mathrm{djn}}$ and (i) is proved. Moreover, suppose t does not have an abstraction shape. Then the same holds for r. By *i.h.* (i) $r \in \mathrm{NE}_{\mathrm{djn}}$. Hence $t \in \mathrm{NE}_{\mathrm{djn}}$ and (i) is proved. □

We already saw that, once β is generalized to dβ, π is not needed anymore to unblock β-redexes; the next lemma says that π preserves djn-nfs, so it does not bring anything new to djn-nfs either.

Lemma 4.2. *If t is a* djn*-nf, and $t \to_\pi t'$, then t' is a* djn*-nf.*

Proof. Given lemma 4.1, the proof proceeds by simultaneous induction on $\mathrm{NF}_{\mathrm{djn}}$ and $\mathrm{NE}_{\mathrm{djn}}$ (for $\mathrm{NE}_{\mathrm{djn}}$ one also proves that $\mathrm{NE}_{\mathrm{djn}}$ does not have an abstraction shape). □

Let us now discuss two properties related to (simple) typability for generalized applications, using the original system of Joachimski and Matthes [JM00], which we call here *ST*. Recall the following typing rules, where $A, B, C ::= a \mid A \longrightarrow B$, and a belongs to a set of base type variables:

$$\frac{}{\Gamma; x : A \vdash x : A} \qquad \frac{\Gamma; x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \longrightarrow B}$$

$$\frac{\Gamma \vdash t : A \longrightarrow B \qquad \Gamma \vdash u : A \qquad \Gamma; y : B \vdash r : C}{\Gamma \vdash t(u, y.r) : C}$$

Subformula property. The subformula property for normal forms is an important property of proof systems, being useful notably for proof search. It holds for von Plato's generalized natural deduction, and therefore also for the original calculus ΛJ. Despite the absence of full normal forms and the minimal amount of permutations used, this property is still true in our system.

Lemma 4.3 (Subformula property). *If* $\Phi = \Gamma \Vdash_{ST} \mathrm{NF}_{\mathrm{djn}} : \tau$ *then every formula in the derivation* Φ *is a subformula of* τ *or a subformula of some formula in* Γ.

Proof. The lemma is proved together with another statement: If $\Psi = \Gamma \Vdash_{ST} \mathrm{NE}_{\mathrm{djn}} : \tau$ then every formula in Ψ is a subformula of some formula in Γ. The proof is by simultaneous induction of Φ and Ψ. □

The subformula property confirms that executing only needed permutations still gives rise to a reasonable notion of normal form.

Termination of simply-typed terms. The second property we show is the typical property that simply typable terms are strongly normalizable. The proof is by the map into the λ-calculus which produces a simulation when the λ-calculus is equipped with the following σ-rules [Reg94]:

$$(\lambda x.M)NN' \mapsto_{\sigma_1} (\lambda x.MN')N \qquad (\lambda x.\lambda y.M)N \mapsto_{\sigma_2} \lambda y.(\lambda x.M)N$$

Theorem 4.4. *If* t *is simply typable, i.e.* $\Gamma \Vdash_{ST} t : \sigma$, *then* $t \in \mathrm{SN}(\mathrm{djn})$.

Proof. The proof uses the traditional map into the λ-calculus given in definition 3.1. This map produces the following simulation: if $t_1 \longrightarrow_{\mathrm{djn}} t_2$ then $t_1^{\#} \longrightarrow^{+}_{\beta\sigma_1} t_2^{\#}$. The proof of the simulation result is by induction on $t_1 \longrightarrow_{\mathrm{djn}} t_2$. The base case needs two lemmas: the first one states that map $(\_)^{\#}$ commutes with substitution; the other, proved by induction on $\mathtt{D}$, states that $\mathtt{D}\langle\lambda x.t\rangle^{\#}u^{\#} \longrightarrow^{+}_{\beta\sigma_1} \mathtt{D}\langle t\{x/u\}\rangle^{\#}$.

Now, given simply typable $t \in \mathtt{T}_J$, the λ-term $t^{\#}$ is also simply typable in the λ-

calculus. Hence, $t^{\#} \in \mathrm{SN}(\beta)$. It is well known that this is equivalent [Reg94] to $t^{\#} \in \mathrm{SN}(\beta, \sigma_1)$. By the simulation result, $t \in \mathrm{SN}(\mathrm{djn})$ follows. □

Confluence We now prove confluence of the calculus. For this, we adapt the proof of Takahashi [Tak95]. The same proof method is used for ΛJ by Joachimski and Matthes [JM00] and by Espírito Santo [Esp20] for ΛJ_v. We begin by defining the following **parallel reduction** $\Rightarrow$:

$$\frac{}{x \Rightarrow_{\mathrm{djn}} x}\ (\mathrm{VAR}) \qquad \frac{t \Rightarrow_{\mathrm{djn}} t'}{\lambda x.t \Rightarrow_{\mathrm{djn}} \lambda x.t'}\ (\mathrm{ABS})$$

$$\frac{t \Rightarrow_{\mathrm{djn}} t' \qquad u \Rightarrow_{\mathrm{djn}} u' \qquad r \Rightarrow_{\mathrm{djn}} r'}{t(u, x.r) \Rightarrow_{\mathrm{djn}} t'(u', x.r')}\ (\mathrm{APP})$$

$$\frac{\mathtt{D}\langle t\rangle \Rightarrow_{\mathrm{djn}} t' \qquad u \Rightarrow_{\mathrm{djn}} u' \qquad r \Rightarrow_{\mathrm{djn}} r'}{\mathtt{D}\langle \lambda x.t\rangle(u, y.r) \Rightarrow_{\mathrm{djn}} r'\{y/t'\{x/u'\}\}}\ (\mathrm{DB})$$

The particularity of our proof is the following lemma which deals with distance.

Lemma 4.5. *Let* $t_1 = \mathtt{D}\langle t\rangle \Rightarrow_{\mathrm{djn}} t_2$. *Then there are* $\mathtt{D}', t'$ *such that* $t_2 = \mathtt{D}'\langle t'\rangle$ *and* $\mathtt{D}\langle \lambda x.t\rangle \Rightarrow_{\mathrm{djn}} \mathtt{D}'\langle \lambda x.t'\rangle$.

Proof. By induction on $\mathtt{D}$.

Case $\mathtt{D} = \Diamond$. We take $\mathtt{D}' = \Diamond, t' = t_2$. We have $\lambda x.t_1 \Rightarrow_{\mathrm{djn}} \lambda x.t_2$ by rule (ABS).

Case $\mathtt{D} = s(u, y.\mathtt{D}_0)$ and $t_1 = s(u, y.\mathtt{D}_0\langle t\rangle) \Rightarrow_{\mathrm{djn}} s'(u', y.r) = t_2$ by rule (APP). By hypothesis, we have $s \Rightarrow_{\mathrm{djn}} s'$, $u \Rightarrow_{\mathrm{djn}} u'$ and $\mathtt{D}_0\langle t\rangle \Rightarrow_{\mathrm{djn}} r$. By *i.h.* $r = \mathtt{D}_1\langle t'\rangle$ and $\mathtt{D}_0\langle \lambda x.t\rangle \Rightarrow_{\mathrm{djn}} \mathtt{D}_1\langle \lambda x.t'\rangle$. We conclude by taking $\mathtt{D}' = s'(u', y.\mathtt{D}_1)$.

Case $\mathtt{D} = \mathtt{D}_0\langle \lambda z.s\rangle(u, y.\mathtt{D}_1)$ and $t_1 = \mathtt{D}_0\langle \lambda z.s\rangle(u, y.\mathtt{D}_1\langle t\rangle) \Rightarrow_{\mathrm{djn}} r\{y/s'\{x/u'\}\} = t_2$ by (ABS). By hypothesis, we have $\mathtt{D}_0\langle \lambda z.s\rangle \Rightarrow_{\mathrm{djn}} s'$, $u \Rightarrow_{\mathrm{djn}} u'$ and $\mathtt{D}_1\langle t\rangle \Rightarrow_{\mathrm{djn}} r$. By *i.h.* $r = \mathtt{D}_2\langle t''\rangle$ and $\mathtt{D}_1\langle \lambda x.t\rangle \Rightarrow_{\mathrm{djn}} \mathtt{D}_2\langle \lambda x.t''\rangle$. We can assume by α-equivalence that the free variables of u' and s' are not bound by $\mathtt{D}_2$. We take $\mathtt{D}' = \mathtt{D}_2\{y/s'\{z/u'\}\}$ and $t' = t''\{y/s'\{z/u'\}\}$. Thus, we have $\mathtt{D}'\langle \lambda x.t'\rangle = \mathtt{D}_2\langle \lambda x.t''\rangle\{y/s'\{z/u'\}\}$ and we can conclude $\mathtt{D}\langle \lambda x.t\rangle = \mathtt{D}_0\langle \lambda z.s\rangle(u, y.\mathtt{D}_1\langle \lambda x.t\rangle) \Rightarrow_{\mathrm{djn}} \mathtt{D}'\langle \lambda x.t'\rangle$ by *i.h.* and rule (ABS). □

Lemma 4.6. *Let* $y \notin \mathrm{fv}(u)$. *Then* $t\{y/r\}\{x/u\} = t\{x/u\}\{y/r\{x/u\}\}$.

Proof. Straightforward by induction on t. □

Lemma 4.7. *Let* $t_1, t_2, u_1, u_2 \in \mathtt{T}_J$. *Then:*

(i) *If* $t_1 \to_{\mathrm{djn}} t_2$, *then* $t_1 \Rightarrow_{\mathrm{djn}} t_2$.

(ii) *If* $t_1 \Rightarrow_{\mathrm{djn}} t_2$, *then* $t_1 \to^*_{\mathrm{djn}} t_2$.

(iii) *If* $t_1 \Rightarrow_{\mathrm{djn}} t_2$ *and* $u_1 \Rightarrow_{\mathrm{djn}} u_2$, *then* $t_1\{z/u_1\} \Rightarrow_{\mathrm{djn}} t_2\{z/u_2\}$.

Proof. The proof of the first statement is by induction on $t_1 \to_{\mathrm{djn}} t_2$. In the base case $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u, y.r) \to_{\mathrm{d}\beta} r\{y/\mathtt{D}\langle t\rangle\{x/u\}\} = t_2$, we use rule (DB) with premises $\mathtt{D}\langle t\rangle \Rightarrow_{\mathrm{djn}} \mathtt{D}\langle t\rangle$, $u \Rightarrow_{\mathrm{djn}} u$ and $r \Rightarrow_{\mathrm{djn}} r$. The other cases are straightforward by *i.h.* and rules (ABS) or (APP).

The proof of the second statement is by induction on $t_1 \Rightarrow_{\mathrm{djn}} t_2$. The base case (VAR) is by an empty reduction $t_1 = x = t_2$. The cases (ABS) and (APP) are direct by *i.h.* The case left is (DB), with $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u, y.r) \Rightarrow_{\mathrm{djn}} r'\{y/t'\{x/u'\}\} = t_2$ with hypothesis $\mathtt{D}\langle t\rangle \Rightarrow_{\mathrm{djn}} t'$, $\mathtt{D}\langle u\rangle \Rightarrow_{\mathrm{djn}} u'$ and $\mathtt{D}\langle r\rangle \Rightarrow_{\mathrm{djn}} r'$. By lemma 4.5, there are $\mathtt{D}', t''$ such that $\mathtt{D}\langle\lambda x.t\rangle \Rightarrow_{\mathrm{djn}} \mathtt{D}'\langle\lambda x.t''\rangle$ and $t' = \mathtt{D}'\langle t''\rangle$. By *i.h.* we have $\mathtt{D}\langle\lambda x.t\rangle \to^*_{\mathrm{djn}} \mathtt{D}'\langle\lambda x.t''\rangle$, $u \to^*_{\mathrm{djn}} u'$ and $r \to^*_{\mathrm{djn}} r'$. We have the following reduction:

$$t_1 \to^*_{\mathrm{djn}} \mathtt{D}'\langle\lambda x.t''\rangle(u', y.r') \to_{\mathrm{djn}} r'\{y/\mathtt{D}'\langle t''\rangle\{x/u'\}\} = t_2.$$

The proof of the third statement is also by induction on $t_1 \Rightarrow_{\mathrm{djn}} t_2$.

Case (VAR). Then t_1 is a variable. If $t_1 = z$, we have $t_1\{z/u_1\} = u_1$, $t_2\{z/u_2\} = u_2$ and this is direct by the second hypothesis. If $t_1 = y \neq z$, we have $t_1\{z/u_1\} = y = t_2\{z/u_2\}$, this is direct by (VAR).

Case (ABS). Then $t_1 = \lambda x.t \Rightarrow_{\mathrm{djn}} \lambda x.t' = t_2$, where w.l.o.g. $x \neq z$ and $x \notin \mathrm{fv}(u_1) \cup \mathrm{fv}(u_2)$ and such that $t \Rightarrow_{\mathrm{djn}} t'$. By *i.h.* we have $t_1\{z/u_1\} = \lambda x.t\{z/u_1\} \Rightarrow_{\mathrm{djn}} \lambda x.t'\{z/u_2\} = t_2\{z/u_2\}$.

Case (APP). Then $t_1 = t(u, x.r) \Rightarrow_{\mathrm{djn}} t'(u', x.r') = t_2$, where w.l.o.g. $x \neq z$ and $x \notin \mathrm{fv}(u_1) \cup \mathrm{fv}(u_2)$ and such that $t \Rightarrow_{\mathrm{djn}} t'$, $u \Rightarrow_{\mathrm{djn}} u'$ and $r \Rightarrow_{\mathrm{djn}} r'$. By *i.h.* we have $t_1\{z/u_1\} = t\{z/u_1\}(u\{z/u_1\}, x.r\{z/u_1\}) \Rightarrow_{\mathrm{djn}} t'\{z/u_1\}(u'\{z/u_1\}, x.r'\{z/u_1\}) = t_2\{z/u_1\}$.

Case (DB). Then $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u, y.r) \Rightarrow_{\mathrm{djn}} r\{y/t'\{x/u'\}\} = t_2$ where w.l.o.g $x, y \neq z$ and $x, y \notin \mathrm{fv}(u_1) \cup \mathrm{fv}(u_2)$, $\mathtt{D}$ does not capture free variables of u_1, u_2, and such that $\mathtt{D}\langle t\rangle \Rightarrow_{\mathrm{djn}} t'$, $u \Rightarrow_{\mathrm{djn}} u'$ and $r \Rightarrow_{\mathrm{djn}} r'$. By *i.h.* we have $\mathtt{D}\langle t\rangle\{z/u_1\} \Rightarrow_{\mathrm{djn}} t'\{z/u_2\}$, $u\{z/u_1\} \Rightarrow_{\mathrm{djn}} u'\{z/u_2\}$ and $r\{z/u_1\} \Rightarrow_{\mathrm{djn}} r'\{z/u_2\}$. Let $\mathtt{D}\langle t\rangle\{z/u_1\} = \mathtt{D}_{\{z/u_1\}}\langle t_{\{z/u_1\}}\rangle$. By rule (DB), we infer

$$\begin{aligned} t\{z/u_1\}_1 &= \mathtt{D}_{\{z/u_1\}}\langle\lambda x.t_{\{z/u_1\}}\rangle(u\{z/u_1\}, y.r\{z/u_1\}) \\ &\Rightarrow_{\mathrm{djn}} r\{z/u_2\}\{y/t'\{z/u_2\}\{x/u'\{z/u_2\}\}\} \\ &= t_2\{z/u_2\} \qquad \text{(by lemma 4.6 twice)} \end{aligned}$$

Lemmas 4.7(i) and 4.7(ii) imply that $\to^*_{\mathrm{djn}}$ is the transitive and reflexive closure of $\Rightarrow_{\mathrm{djn}}$. We now only need to prove the *diamond* property for $\Rightarrow_{\mathrm{djn}}$ to conclude. The difference between Takahashi's method and the more usual Tait and Martin-Löfs's method [Bar84, §3.2] is to replace the proof of diamond for the parallel reduction by a proof of the *triangle property*.

Definition 4.8 (Triangle property). Let $\to_{\mathscr{R}}$ be a reduction relation on $\mathtt{T}_J$ and f a function. $(\to_{\mathscr{R}}, f)$ satisfies the triangle property if, for any $t \in \mathtt{T}_J$, $t \to_{\mathscr{R}} t'$ implies $t' \to_{\mathscr{R}} f(t)$.

Definition 4.9 (Developments). The dβ-development $(t)^{\mathrm{d}\beta}$ of a $\mathtt{T}_J$-term t is defined as follows.

$$(x)^{\mathrm{d}\beta} = x \qquad (\lambda x.t)^{\mathrm{d}\beta} = \lambda x.(t)^{\mathrm{d}\beta} \qquad (t(u, y.r))^{\mathrm{d}\beta} = \begin{cases} (r)^{\mathrm{d}\beta}\{y/(\mathtt{D}\langle t'\rangle)^{\mathrm{d}\beta}\{x/(u)^{\mathrm{d}\beta}\}\}, & \text{if } t = \mathtt{D}\langle\lambda x.t'\rangle \\ (t)^{\mathrm{d}\beta}((u)^{\mathrm{d}\beta}, x.(r)^{\mathrm{d}\beta}), & \text{otherwise} \end{cases}$$

Lemma 4.10 (Triangle property of $(\Rightarrow_{\mathrm{djn}}, (\cdot)^{\mathrm{d}\beta})$). *Let* $t_1 \Rightarrow_{\mathrm{djn}} t_2$. *Then* $t_2 \Rightarrow_{\mathrm{djn}} (t_1)^{\mathrm{d}\beta}$.

Proof. By induction on t_1.

Case $t_1 = x$. Then $t_1 = t_2 = (t_1)^{\mathrm{d}\beta}$ and we conclude with rule (VAR).

Case $t_1 = \lambda x.t$. Then $t_1 \Rightarrow_{\mathrm{djn}} t_2 = \lambda x.t'$ by rule (ABS). We have $(t_1)^{\mathrm{d}\beta} = \lambda x.(t)^{\mathrm{d}\beta}$. By *i.h.* $t' \Rightarrow_{\mathrm{djn}} (t)^{\mathrm{d}\beta}$. By (ABS), $\lambda x.t' \Rightarrow_{\mathrm{djn}} \lambda x.(t)^{\mathrm{d}\beta}$.

Case $t_1 = t(u, y.r)$, where $t \neq \mathtt{D}\langle\lambda x.t'\rangle$. Then $t_1 \Rightarrow_{\mathrm{djn}} t_2 = t'(u', y.r')$ by rule (APP). We have $(t_1)^{\mathrm{d}\beta} = (t)^{\mathrm{d}\beta}((u)^{\mathrm{d}\beta}, y.(r)^{\mathrm{d}\beta})$. By *i.h.* $t' \Rightarrow_{\mathrm{djn}} (t)^{\mathrm{d}\beta}$, $u' \Rightarrow_{\mathrm{djn}} (u)^{\mathrm{d}\beta}$ and $r' \Rightarrow_{\mathrm{djn}} (r)^{\mathrm{d}\beta}$. By (APP), $t'(u', y.r') \Rightarrow_{\mathrm{djn}} (t_1)^{\mathrm{d}\beta}$.

Case $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u, y.r)$. Then $(t_1)^{\mathrm{d}\beta} = (r)^{\mathrm{d}\beta}\{y/(\mathtt{D}\langle t\rangle)^{\mathrm{d}\beta}\{x/(u)^{\mathrm{d}\beta}\}\}$. There are two subcases. In both cases we have $u \Rightarrow_{\mathrm{djn}} u'$ and $r \Rightarrow_{\mathrm{djn}} r'$ and thus by *i.h.* $u' \Rightarrow_{\mathrm{djn}} (u)^{\mathrm{d}\beta}$ and $r \Rightarrow_{\mathrm{djn}} (r)^{\mathrm{d}\beta}$.

- **Subcase** (APP). Then $\mathtt{D}\langle\lambda x.t\rangle \Rightarrow_{\mathrm{djn}} t'$. By a reasoning similar to lemma 4.5, we can show that $t' = \mathtt{D}'\langle\lambda x.t''\rangle$ and that $\mathtt{D}\langle t\rangle \Rightarrow_{\mathrm{djn}} \mathtt{D}'\langle t''\rangle$. Thus $t_2 = \mathtt{D}'\langle\lambda x.t''\rangle(u', y.r')$ and by *i.h.* $\mathtt{D}'\langle t''\rangle \Rightarrow_{\mathrm{djn}} (\mathtt{D}\langle t\rangle)^{\mathrm{d}\beta}$. We use rule (DB) with the three *i.h.* as premises to derive $t_2 \Rightarrow_{\mathrm{djn}} (r)^{\mathrm{d}\beta}\{y/(\mathtt{D}\langle t\rangle)^{\mathrm{d}\beta}\{x/(u)^{\mathrm{d}\beta}\}\} = (t_1)^{\mathrm{d}\beta}$.
- **Subcase** (DB). Then $t_2 = r'\{y/t'\{x/u'\}\}$ and $\mathtt{D}\langle t\rangle \Rightarrow_{\mathrm{djn}} t'$. By *i.h.* $t' \Rightarrow_{\mathrm{djn}} (\mathtt{D}\langle t\rangle)^{\mathrm{d}\beta}$. By *i.h.* and two applications of lemma 4.7(iii), we have $r'\{y/t'\{x/u'\}\} \Rightarrow_{\mathrm{djn}} (r)^{\mathrm{d}\beta}\{y/(\mathtt{D}\langle t\rangle)^{\mathrm{d}\beta}\{x/(u)^{\mathrm{d}\beta}\}\} = (t_1)^{\mathrm{d}\beta}$. □

Property 4.11. $\to_{\mathrm{djn}}$ *is confluent.*

Proof. The triangle property of $(\Rightarrow_{\mathrm{djn}}, (\cdot)^{\mathrm{d}\beta})$ implies that $\Rightarrow_{\mathrm{djn}}$ is diamond, since for any t_2 such that $t_1 \Rightarrow_{\mathrm{djn}} t_2$, $t_2 \Rightarrow_{\mathrm{djn}} (t_1)^{\mathrm{d}\beta}$. This implies in turn that $\Rightarrow_{\mathrm{djn}} = \to_{\mathrm{djn}}^*$ is diamond and thus that $\to_{\mathrm{djn}}$ is confluent. □

4.3 Inductive Characterization of Strong Normalization

In this section we give an inductive characterization of strong normalization (ISN) for λJ_n and prove it correct. This characterization will be useful to show completeness of the type system that we are going to present in section 4.4.1, as well as to compare strong normalization of λJ_n to the ones of $\mathtt{T}_\Lambda[\beta, \mathrm{p2}]$ and ΛJ.

4.3.1 ISN in the λ-Calculus with Weak-Head Contexts

We write $\mathrm{ISN}(\mathscr{R})$ the set of strongly normalizing terms under $\mathscr{R}$ given by the inductive definition. As an introduction, we first look at the case of ISN for the λ-calculus ($\mathrm{ISN}(\beta)$), on which our forthcoming definition of ISN(djn) elaborates. A usual way to define $\mathrm{ISN}(\beta)$ is by the following rules [vRaa96], where the general notation $M\vec{P}$ abbreviates $(\dots(MP_1)\dots)P_n$ for some $n \geq 0$.

$$\frac{P_1, \dots, P_n \in \mathrm{ISN}(\beta)}{x\vec{P} \in \mathrm{ISN}(\beta)} \qquad \frac{M \in \mathrm{ISN}(\beta)}{\lambda x.M \in \mathrm{ISN}(\beta)} \qquad \frac{M\{x/N\}\vec{P}, N \in \mathrm{ISN}(\beta)}{(\lambda x.M)N\vec{P} \in \mathrm{ISN}(\beta)}$$

One then shows that $M \in \mathrm{SN}(\beta)$ if and only if $M \in \mathrm{ISN}(\beta)$.

Notice that this definition is deterministic. Indeed, a reduction strategy emerges from this definition: weak-head reduction. The strategy is the following: reduce a term to a weak-head normal form $x\vec{P}$ or $\lambda x.M$, and then iterate reduction inside arguments and under abstractions, without any need to come back to the head of the term. Formally, **weak-head normal forms** are of two kinds:

$$\textbf{(Neutral terms)} \quad \mathtt{n} ::= x \mid \mathtt{n}M$$
$$\textbf{(Answers)} \quad \mathtt{a} ::= \lambda x.M$$

Neutral terms cannot produce any head β-redex. They are the terms of the shape $x\vec{P}$. On the contrary, answers can create a β-redex when given at least one argument. In the case of the λ-calculus, these are only abstractions. If the term is not a weak-head term, a redex can be located with a

$$\textbf{(Weak-head context)} \quad \mathtt{W} ::= \Diamond \mid \mathtt{W}t.$$

These concepts give rise to a different definition of $\mathrm{ISN}(\beta)$.

$$\frac{}{x \in \mathrm{ISN}(\beta)} \qquad \frac{\mathtt{n}, M \in \mathrm{ISN}(\beta)}{\mathtt{n}M \in \mathrm{ISN}(\beta)} \qquad \frac{M \in \mathrm{ISN}(\beta)}{\lambda x.M \in \mathrm{ISN}(\beta)} \qquad \frac{\mathtt{W}\langle M\{x/N\}\rangle, N \in \mathrm{ISN}(\beta)}{\mathtt{W}\langle(\lambda x.M)N\rangle \in \mathrm{ISN}(\beta)}$$

Weak-head contexts are an alternative to the meta-syntactic notation $\vec{r}$ of vectors of arguments. Notice that there is one rule for each kind of neutral term, one rule for answers and one rule for terms which are not weak-head normal forms.

4.3.2 ISN for $\mathrm{d}\beta$

We define ISN(djn) with the same tools as in the last subsection. Hence, we first have to define neutral terms, answers and a notion of contexts. We call the contexts left-right contexts ($\mathtt{R}$), and the underlying strategy the left-right strategy.

Definition 4.12. We consider the following grammars:

$$\begin{array}{rrcl} \textbf{(Neutral terms)} & \mathtt{n} & ::= & x \mid \mathtt{n}(u, x.\mathtt{n}) \\ \textbf{(Answers)} & \mathtt{a} & ::= & \lambda x.t \mid \mathtt{n}(u, x.\mathtt{a}) \\ \textbf{(Neutral distant contexts)} & \mathtt{D_n} & ::= & \Diamond \mid \mathtt{n}(u, x.\mathtt{D_n}) \\ \textbf{(Left-right contexts)} & \mathtt{R} & ::= & \Diamond \mid \mathtt{R}(u, x.r) \mid \mathtt{n}(u, x.\mathtt{R}) \end{array}$$

Notice that $\mathtt{n}$ and $\mathtt{a}$ are disjoint and stable by djn-reduction. Also $\mathtt{D_n} \subsetneq \mathtt{R}$.

Example 4.13 (Decomposition). Let $t = x_1(x_2, y_1.I(I, z.I))(x_3, y.II)$. Then, there are two decompositions of t in terms of a redex r and a left-right context $\mathtt{R}$: either $\mathtt{R} = \Diamond$ and $r = t$, or $\mathtt{R} = x_1(x_2, y_1.\Diamond)(x_3, y.II)$ and $r = I(I, z.I)$. In both cases $t = \mathtt{R}\langle r\rangle$. We will rule out the first possibility by defining next a restriction of the β-rule, securing uniqueness of such kind of decomposition in all cases.

The strategy underlying our definition of ISN($\mathrm{d}\beta$) is the **left-right strategy** $\longrightarrow_{\mathrm{lr}}$, defined as the closure under $\mathtt{R}$ of the following restricted β-rule:

$$\mathtt{D_n}\langle\lambda x.t\rangle(u, y.r) \mapsto r\{y/\mathtt{D_n}\langle t\{y/u\}\rangle\}.$$

The restriction of $\mathtt{D}$ to a neutral distant context $\mathtt{D_n}$ is what allows determinism of our forthcoming definition 4.17.

Remark that the strategy is not a weak-head strategy for generalized applications, given by the grammar: $\mathtt{W} ::= \Diamond \mid \mathtt{W}(u, x.\mathtt{W}'\langle\langle x\rangle\rangle)t(u, x.\mathtt{W})$. This is because we ultimately need to reduce all redexes, even the ones of the shape $\mathtt{D}\langle\lambda x.t\rangle(u, y.r)$ where y is not in r.

Lemma 4.14. *Let $t \in \mathtt{T}_J$. Then t is in* lr*-normal form iff $t \in \mathtt{n} \cup \mathtt{a}$.*

Proof. First, we show that t lr-normal implies $t \in \mathtt{n} \cup \mathtt{a}$, by induction on t. If $t = x$, then $t \in \mathtt{n}$. If $t = \lambda x.s$, then $t \in \mathtt{a}$. Let $t = s(u, x.r)$ where s and r are lr-normal. Then $s \notin \mathtt{a}$, otherwise the term would lr-reduce at root. Thus by the *i.h.* $s \in \mathtt{n}$. By the *i.h.* again $r \in \mathtt{n} \cup \mathtt{a}$ so that $t \in \mathtt{n} \cup \mathtt{a}$.

Second, we show that $t \in \mathtt{n} \cup \mathtt{a}$ implies t is lr-normal, by simultaneous induction on $\mathtt{n}$ and $\mathtt{a}$. The cases $t = x$ (*i.e.* $t \in \mathtt{n}$) and $t = \lambda x.s$ (*i.e.* $t \in \mathtt{a}$) are straightforward. Let $t = s(u, x.r)$ where $s \in \mathtt{n}$ and $r \in \mathtt{n} \cup \mathtt{a}$. Since $r, s \in \mathtt{n} \cup \mathtt{a}$, by the *i.h.* t does not lr-reduce in r or s. Since $s \in \mathtt{n}$, t does not lr-reduce at root either. Then, t is lr-normal. □

Lemma 4.15. *The reduction $\longrightarrow_{\mathrm{lr}}$ is deterministic.*

Proof. Let t be a lr-reducible term. We reason by induction on t. If t is a variable or an abstraction, then t does not lr-reduce so that t is necessarily an application $t'(u, y.r)$. By lemma 4.14 we have three possible cases for t'.

Case $t = t'(u, y.r)$ with $t' \in \mathtt{a}$. Then $t = \mathtt{D_n}\langle\lambda x.s\rangle(u, y.r)$, so t reduces at the root. Since $t' \in \mathtt{a}$, then we know by lemma 4.14 that (1) $t' \in \mathrm{NF_{lr}}$, (2) $t' \notin \mathtt{n}$, so that t does not lr-reduce in t' or r.

Case $t = t'(u, y.r)$ with $t' \in \mathtt{n}$. Then t does not lr-reduce at the root. By lemma 4.14, we know that $t' \in \mathrm{NF_{lr}}$ and thus t necessarily reduces in r. By the *i.h.* this reduction is deterministic.

Case $t = t'(u, y.r)$ with $t' \notin \mathrm{NF_{lr}}$. Then in particular by lemma 4.14 we know that (1) t' does not have an abstraction shape so that t does not reduce at the root, and (2) $t' \notin \mathtt{n}$ so that t does not reduce in r. Thus t lr-reduces only in t'. By the *i.h.* this reduction is deterministic. □

Symmetrically to the λ-calculus, left-right normal forms are either neutral terms or answers. This time, answers are not only abstractions, but also abstractions under a neutral distant context. Because of distance, these terms can also create a dβ-redex when applied to an argument, as seen in the next remark.

Remark 4.16. Consider again the term $t = x_1(x_2, y_1.I(I, z.I))(x_3, y.II)$ of example 4.13. If left-right contexts were taken to be a naive translation of the ones of the λ-calculus and the form $\mathtt{n}(u, x.\mathtt{R})$ of the grammar of $\mathtt{R}$ was disallowed, then it would not be possible to write t as $\mathtt{R}\langle r\rangle$, with r a restricted redex. In that case, the reduction strategy associated with ISN(djn) would consider t as a left-right normal form, and start reducing the subterms of t, including $I(I, z.I)$. Now, the latter would eventually reach I and suddenly the whole term $t' = x_1(x_2, y_1.I)(x_3, y.r')$ would be a left-right redex again: the typical separation between an initial external reduction phase and a later internal reduction phase, as it is the case in the λ-calculus, would be lost in our framework. This is a subtle point due to the *distant* character of rule dβ which explains the complexity of definition 4.12.

Our inductive definition of strong normalization follows.

Definition 4.17 (Inductive strong normalization). We consider the following inductive predicate:

$$\frac{}{x \in \mathrm{ISN(djn)}}(\textsc{snvar}) \qquad \frac{\mathtt{n}, u, r \in \mathrm{ISN(djn)} \qquad r \in \mathrm{NF_{lr}}}{\mathtt{n}(u, x.r) \in \mathrm{ISN(djn)}}(\textsc{snapp}) \qquad \frac{t \in \mathrm{ISN(djn)}}{\lambda x.t \in \mathrm{ISN(djn)}}(\textsc{snabs})$$

$$\frac{\mathtt{R}\langle r\{y/\mathtt{D_n}\langle t\{x/u\}\rangle\}\rangle, \mathtt{D_n}\langle t\rangle, u \in \mathrm{ISN(djn)}}{\mathtt{R}\langle \mathtt{D_n}\langle\lambda x.t\rangle(u, y.r)\rangle \in \mathrm{ISN(djn)}}(\textsc{snbeta})$$

Notice that every term can be written according to the conclusions of the previous rules, so that the following grammar also defines the syntax $\mathtt{T}_J$.

$$t, u, r ::= x \mid \lambda x.t \mid \mathtt{n}(u, x.\,\mathrm{NF_{lr}}) \mid \mathtt{R}\langle \mathtt{D_n}\langle\lambda x.t\rangle(u, y.r)\rangle \tag{4.2}$$

Moreover, at most one rule in the previous definition applies to each term, *i.e.* the rules are deterministic. An equivalent, but non-deterministic definition, can be given by removing the side condition "$r \in \mathrm{NF}_{\mathrm{lr}}$" in rule (SNAPP). Indeed, this (weaker) rule would overlap with rule (SNBETA) for terms in which the left-right context lies in the last continuation, as for instance in $x(u, y.y)(u', y'.\mathtt{I}\mathtt{I})$. Notice the difference with the λ-calculus: the head of a term with generalized applications can be either on the left of the term (as in the λ-calculus), or recursively on the left in a continuation.

To show that our definition corresponds to strong normalization, we need a few intermediate statements.

Lemma 4.18. *If* $t_0 \to_{\mathrm{djn}} t_1$, *then*

(i) $t_0\{x/u\} \to_{\mathrm{djn}} t_1\{x/u\}$, *and*

(ii) $u\{x/t_0\} \to^*_{\mathrm{djn}} u\{x/t_1\}$.

Proof. In the base cases, we have $t_0 = \mathtt{D}\langle\lambda z.t\rangle(s, y.r) \mapsto_{\mathrm{d}\beta} r\{y/\mathtt{D}\langle t\rangle\{z/s\}\} = t_1$. By α-equivalence we can suppose that $y, z \notin \mathrm{fv}(u)$ and $x \neq y$, $x \neq z$. The inductive cases and the base case for item (ii) are straightforward. We detail the base case of item (i).

$$\begin{aligned} t_0\{x/u\} &= \mathtt{D}\langle\lambda z.t\rangle\{x/u\}(s\{x/u\}, y.r\{x/u\}) \\ &\to_{\mathrm{djn}} r\{x/u\}\{y/\mathtt{D}\langle t\rangle\{x/u\}\{z/s\{x/u\}\}\} \\ &=_{4.6} r\{x/u\}\{y/(\mathtt{D}\langle t\rangle\{z/s\})\{x/u\}\} \\ &=_{4.6} r\{y/\mathtt{D}\langle t\rangle\{z/s\}\}\{x/u\} \\ &= t_1\{x/u\} \end{aligned}$$

□

Remark 4.19. For any $\mathtt{T}_J$-term $\mathtt{D}\langle\lambda x.t\rangle \in \mathrm{SN}(\mathrm{djn}) \iff \mathtt{D}\langle t\rangle \in \mathrm{SN}(\mathrm{djn})$.

Lemma 4.20. *Let* $t_0 = \mathtt{R}\langle r\{y/\mathtt{D}\langle t\rangle\{x/u\}\}\rangle, \mathtt{D}\langle t\rangle, u \in \mathrm{SN}(\mathrm{djn})$. *Then* $t'_0 = \mathtt{R}\langle\mathtt{D}\langle\lambda x.t\rangle(u, y.r)\rangle \in \mathrm{SN}(\mathrm{djn})$.

Proof. In this proof we use a notion of reduction of contexts which is the expected one: $\mathtt{C} \to \mathtt{C}'$ iff the hole in $\mathtt{C}$ is outside the redex contracted in the reduction step. By hypothesis we also have $r \in \mathrm{SN}(\mathrm{djn})$. We use the lexicographic order to reason by induction on $\langle \|t_0\|_{\mathrm{djn}}, \|\mathtt{D}\langle t\rangle\|_{\mathrm{djn}}, \|u\|_{\mathrm{djn}}\rangle$. To show $t'_0 \in \mathrm{SN}(\mathrm{djn})$ it is sufficient to show that all its reducts are in $\mathrm{SN}(\mathrm{djn})$. We analyze all possible cases.

Case $t'_0 \to_{\mathrm{djn}} t_0$. We conclude by the hypothesis.

Case $t'_0 \to_{\mathrm{djn}} \mathtt{R}\langle\mathtt{D}\langle\lambda x.t'\rangle(u, y.r)\rangle = t'_1$, where $t \to_{\mathrm{djn}} t'$. Thus also $\mathtt{D}\langle t\rangle \to_{\mathrm{djn}} \mathtt{D}\langle t'\rangle$. We then have $\mathtt{D}\langle t'\rangle \in \mathrm{SN}(\mathrm{djn})$ and $u \in \mathrm{SN}(\mathrm{djn})$ and by lemma 4.18(ii) we have $t_0 = \mathtt{R}\langle r\{y/\mathtt{D}\langle t\rangle\{x/u\}\}\rangle \to^*_{\mathrm{djn}} \mathtt{R}\langle r\{y/\mathtt{D}\langle t'\rangle\{x/u\}\}\rangle = t_1$, so that also $t_1 \in \mathrm{SN}(\mathrm{djn})$. We can con-

clude that $t_1' \in \mathrm{SN}(\mathrm{djn})$ by the *i.h.* since $\|t_1\|_{\mathrm{djn}} \leq \|t_0\|_{\mathrm{djn}}$ and $\|\mathtt{D}\langle t'\rangle\|_{\mathrm{djn}} < \|\mathtt{D}\langle t\rangle\|_{\mathrm{djn}}$.

Case $t_0' \to_{\mathrm{djn}} \mathtt{R}\langle \mathtt{D}\langle \lambda x.t\rangle(u', y.r)\rangle = t_1'$, where $u \to_{\mathrm{djn}} u'$. We have $\mathtt{D}\langle t\rangle, u' \in \mathrm{SN}(\mathrm{djn})$ and by lemma 4.18(ii) $t_0 = \mathtt{R}\langle r\{y/\mathtt{D}\langle t\rangle\{x/u\}\}\rangle \to_{\mathrm{djn}}^* \mathtt{R}\langle r\{y/\mathtt{D}\langle t\rangle\{x/u'\}\}\rangle = t_1$, so that also $t_1 \in \mathrm{SN}(\mathrm{djn})$. We conclude $t_1' \in \mathrm{SN}(\mathrm{djn})$ by the *i.h.* since $\|t_1\|_{\mathrm{djn}} \leq \|t_0\|_{\mathrm{djn}}$ and $\|u'\|_{\mathrm{djn}} < \|u\|_{\mathrm{djn}}$.

Case $t_0' \to_{\mathrm{djn}} \mathtt{R}\langle \mathtt{D}\langle \lambda x.t\rangle(u, y.r')\rangle = t_1'$, where $r \to_{\mathrm{djn}} r'$. We have $\mathtt{D}\langle t\rangle, u \in \mathrm{SN}(\mathrm{djn})$ and by lemma 4.18(i) $t_0 = \mathtt{R}\langle r\{y/\mathtt{D}\langle t\rangle\{x/u\}\}\rangle \to_{\mathrm{djn}} \mathtt{R}\langle r'\{y/\mathtt{D}\langle t\rangle\{x/u\}\}\rangle = t_1$. We conclude $t_1' \in \mathrm{SN}(\mathrm{djn})$ by the *i.h.* since since $\|t_1\|_{\mathrm{djn}} < \|t_0\|_{\mathrm{djn}}$.

Case $t_0' \to_{\mathrm{djn}} \mathtt{R}\langle \mathtt{D}'\langle \lambda x.t\rangle(u, y.r)\rangle = t_1'$, where $\mathtt{D} \to_{\mathrm{djn}} \mathtt{D}'$. We have $\mathtt{D}'\langle t\rangle, u \in \mathrm{SN}(\mathrm{djn})$ and by lemma 4.18 $t_0 = \mathtt{R}\langle r\{y/\mathtt{D}\langle t\rangle\{x/u\}\}\rangle \to_{\mathrm{djn}}^* \mathtt{R}\langle r\{y/\mathtt{D}'\langle t\rangle\{x/u\}\}\rangle = t_1$, so that also $t_1 \in \mathrm{SN}(\mathrm{djn})$. We conclude $t_1' \in \mathrm{SN}(\mathrm{djn})$ by the *i.h.* since $\|t_1\|_{\mathrm{djn}} \leq \|t_0\|_{\mathrm{djn}}$ and $\|\mathtt{D}'\langle t\rangle\|_{\mathrm{djn}} < \|\mathtt{D}\langle t\rangle\|_{\mathrm{djn}}$.

Case $t_0' \to_{\mathrm{djn}} \mathtt{R}'\langle \mathtt{D}\langle \lambda x.t\rangle(u, y.r)\rangle = t_1'$, where $\mathtt{R} \to_{\mathrm{djn}} \mathtt{R}'$. Thus $t_0 = \mathtt{R}\langle r\{y/\mathtt{D}\langle t\rangle\{x/u\}\}\rangle \to_{\mathrm{djn}} \mathtt{R}'\langle r\{y/\mathtt{D}\langle t\rangle\{x/u\}\}\rangle = t_1$. We have $t_1, \mathtt{D}\langle t\rangle, u \in \mathrm{SN}(\mathrm{djn})$. We conclude that $t_1' \in \mathrm{SN}(\mathrm{djn})$ by the *i.h.* since $\|t_1\|_{\mathrm{djn}} < \|t_0\|_{\mathrm{djn}}$.

Case $\mathtt{R} = \mathtt{R}'\langle \mathtt{D}_{\mathrm{n}}(u', y'.r')\rangle$ and $r = \mathtt{D}''\langle \lambda x'.t'\rangle$. This is the only case left. Indeed, there is no redex in $\mathtt{D}\langle \lambda x.t\rangle$ other than in $\mathtt{D}$ or $\lambda x.t$. Then,

$$t_0' = \mathtt{R}'\langle \mathtt{D}_{\mathrm{n}}\langle \mathtt{D}\langle \lambda x.t\rangle(u, y.\mathtt{D}''\langle \lambda x'.t'\rangle)\rangle(u', y'.r')\rangle$$

Let $\mathtt{D}' = \mathtt{D}_{\mathrm{n}}\langle \mathtt{D}\langle \lambda x.t\rangle(u, y.\mathtt{D}'')\rangle$. The reduction we need to consider is:

$$\begin{aligned} t_0' &= \mathtt{R}'\langle \mathtt{D}'\langle \lambda x'.t'\rangle(u', y'.r')\rangle \\ &\to_{\mathrm{djn}} \mathtt{R}'\langle r'\{y'/\mathtt{D}'\langle t'\rangle\{x'/u'\}\}\rangle \\ &= \mathtt{R}'\langle r'\{y'/\mathtt{D}_{\mathrm{n}}\langle \mathtt{D}\langle \lambda x.t\rangle(u, y.\mathtt{D}''\langle t'\rangle)\rangle\{x'/u'\}\}\rangle = t_1' \end{aligned}$$

We will show that $t_1' \in \mathrm{SN}(\mathrm{djn})$.

For this we show that $t_1 = \mathtt{R}'\langle r'\{y'/\mathtt{D}_{\mathrm{n}}\langle \mathtt{D}''\langle t'\rangle\{y/\mathtt{D}\langle t\rangle\{x/u\}\}\rangle\{x'/u'\}\}\rangle \in \mathrm{SN}(\mathrm{djn})$, that $\mathtt{D}'\langle t'\rangle \in \mathrm{SN}(\mathrm{djn})$ and that $u' \in \mathrm{SN}(\mathrm{djn})$. We have $t_0 \to_{\mathrm{djn}}^+ t_1$ so that $t_1 \in \mathrm{SN}(\mathrm{djn})$ and $\|t_1\|_{\mathrm{djn}} < \|t_0\|_{\mathrm{djn}}$. u' is a subterm of t_0, which is in $\mathrm{SN}(\mathrm{djn})$, so that $u' \in \mathrm{SN}(\mathrm{djn})$. To show that $\mathtt{D}'\langle t'\rangle \in \mathrm{SN}(\mathrm{djn})$, we consider $t_2 = \mathtt{D}_{\mathrm{n}}\langle \mathtt{D}''\langle \lambda x'.t'\rangle\{y/\mathtt{D}\langle t\rangle\{x/u\}\}\rangle$. We have $t_0 = \mathtt{R}'\langle t_2(u', y'.r')\rangle$. We can show that $\|t_2\|_{\mathrm{djn}} < \|t_0\|_{\mathrm{djn}}$ (so that $t_2 \in \mathrm{SN}(\mathrm{djn})$). Indeed, $\|\mathtt{R}'\langle t_2(u', y'.r')\rangle\|_{\mathrm{djn}} \geq \|t_2(u', y'.r')\|_{\mathrm{djn}} \geq \|t_2\|_{\mathrm{djn}} + 1 > \|t_2\|_{\mathrm{djn}}$. The second inequality holds since t_2 has an abstraction shape, and abstraction shapes are stable under substitution, and thus $t_2(u', y'.r')$ is also a redex. We can then conclude that $t_2' = \mathtt{D}_{\mathrm{n}}\langle \mathtt{D}\langle \lambda x.t\rangle(u, y.\mathtt{D}''\langle \lambda x'.t'\rangle)\rangle = \mathtt{D}'\langle \lambda x'.t'\rangle \in \mathrm{SN}(\mathrm{djn})$ by the *i.h.* since $u, \mathtt{D}\langle t\rangle \in \mathrm{SN}(\mathrm{djn})$. Thus, $\mathtt{D}'\langle t'\rangle \in \mathrm{SN}(\mathrm{djn})$ by remark 4.19.

We then have $t_1, \mathtt{D}'\langle t'\rangle, u' \in \mathrm{SN}(\mathrm{djn})$ and we can conclude $t_1' \in \mathrm{SN}(\mathrm{djn})$ since $\|t_1\|_{\mathrm{djn}} < \|t_0\|_{\mathrm{djn}}$. We conclude $t_1' \in \mathrm{SN}(\mathrm{djn})$ as required. □

Theorem 4.21. SN(djn) = ISN(djn).

Proof. First, we show ISN(djn) ⊆ SN(djn). We proceed by induction on $t \in \mathrm{ISN}(\mathrm{djn})$.

Case $t = x$. Straightforward.

Case $t = \lambda x.s$, where $s \in \mathrm{ISN}(\mathrm{djn})$. By the *i.h.* $s \in \mathrm{SN}(\mathrm{djn})$, so that $t \in \mathrm{SN}(\mathrm{djn})$ trivially holds.

Case $t = s(u, x.r) \in \mathrm{NF}_{\mathrm{lr}}$ where $s, u, r \in \mathrm{ISN}(\mathrm{djn})$. By lemma 4.14 we have $s \in \mathtt{n}$ and thus in particular s can not djn-reduce to an answer. Therefore any kind of reduction starting at t only occurs in the subterms s, u and r. We conclude since by the *i.h.* we have $s, u, r \in \mathrm{SN}(\mathrm{djn})$.

Case $t = \mathtt{R}\langle \mathtt{D}_{\mathtt{n}}\langle \lambda x.s\rangle(u, y.r)\rangle$, where $\mathtt{R}\langle r\{y/\mathtt{D}_{\mathtt{n}}\langle s\rangle\}\{x/u\}\}\rangle, \mathtt{D}_{\mathtt{n}}\langle s\rangle, u \in \mathrm{ISN}(\mathrm{djn})$. The *i.h.* gives $\mathtt{R}\langle r\{y/\mathtt{D}_{\mathtt{n}}\langle s\rangle\}\{x/u\}\}\rangle \in \mathrm{SN}(\mathrm{djn})$, $\mathtt{D}_{\mathtt{n}}\langle s\rangle \in \mathrm{SN}(\mathrm{djn})$ and $u \in \mathrm{SN}(\mathrm{djn})$ so that by lemma 4.20 $t = \mathtt{R}\langle \mathtt{D}_{\mathtt{n}}\langle \lambda x.s\rangle(u, y.r)\rangle \in \mathrm{SN}(\mathrm{djn})$ holds, with $\mathtt{D} = \mathtt{D}_{\mathtt{n}}$.

Next, we show SN(djn) ⊆ ISN(djn). Let $t \in \mathrm{SN}(\mathrm{djn})$. We reason by induction on $\langle \|t\|_{\mathrm{djn}}, |t|\rangle$ w.r.t. the lexicographic order. If $\langle \|t\|_{\mathrm{djn}}, |t|\rangle$ is minimal, *i.e.* $\langle 0, 1\rangle$, then t is a variable and thus in ISN(djn) by rule (SNVAR). Otherwise we proceed by case analysis.

Case $t = \lambda x.s$. Since $\|s\|_{\mathrm{djn}} \leq \|t\|_{\mathrm{djn}}$ and $|s| < |t|$, we conclude by the *i.h.* and rule (SNABS).

Case t is an application. There are two cases.

Subcase $t \in \mathrm{NF}_{\mathrm{lr}}$. Then $t = s(u, x.r)$ with $s, u, r \in \mathrm{SN}(\mathrm{djn})$ and $s \in \mathtt{n}$. We have $\|s\|_{\mathrm{djn}} \leq \|t\|_{\mathrm{djn}}$, $\|u\|_{\mathrm{djn}} \leq \|t\|_{\mathrm{djn}}$, $\|r\|_{\mathrm{djn}} \leq \|t\|_{\mathrm{djn}}$, $|s| < |t|$, $|u| < |t|$ and $|r| < |t|$. By the *i.h.* $s, u, r \in \mathrm{ISN}(\mathrm{djn})$ and thus we conclude by rule (SNAPP).

Subcase $t \notin \mathrm{NF}_{\mathrm{lr}}$. By definition there is a context $\mathtt{R}$ s.t. $t = \mathtt{R}\langle \mathtt{D}_{\mathtt{n}}\langle \lambda x.s\rangle(u, y.r)\rangle$. Moreover, $t \in \mathrm{SN}(\mathrm{djn})$ implies in particular $\mathtt{R}\langle r\{y/\mathtt{D}_{\mathtt{n}}\langle s\rangle\}\{x/u\}\}\rangle, u \in \mathrm{SN}(\mathrm{djn})$, so that they are in ISN(djn) by the *i.h.* Moreover, $t \in \mathrm{SN}(\mathrm{djn})$ also implies $\mathtt{D}_{\mathtt{n}}\langle \lambda x.s\rangle \in \mathrm{SN}(\mathrm{djn})$. Since the abstraction $\lambda x.s$ is never applied nor an argument, this is equivalent to $\mathtt{D}_{\mathtt{n}}\langle s\rangle \in \mathrm{SN}(\mathrm{djn})$, thus $\mathtt{D}_{\mathtt{n}}\langle s\rangle \in \mathrm{ISN}(\mathrm{djn})$ by the *i.h.* We conclude by rule (SNBETA). □

4.4 Quantitative Types Capture Strong Normalization

We proved that simply typable terms are strongly normalizing in section 4.2. In this section we use non-idempotent intersection types to fully characterize strong normalization, so that strongly normalizing terms are also typable. First we introduce the typing system, next we

prove the characterization and finally we study the quantitative behavior of π and give in particular an example of failure.

4.4.1 The Typing System

We now define our quantitative type system $\cap J$ for T_J-terms and we show that strong normalization in λJ_n exactly corresponds to $\cap J$ typability. As discussed in section 1.3.2.3, we introduce a **choice operator** on multiset types: if $\mathscr{M} \neq [\,]$, then $\#(\mathscr{M}) = \mathscr{M}$, otherwise $\#([\,]) = [\sigma]$, where σ is an arbitrary type. This operator is used to guarantee that there is always a typing witness for all the subterms of typed terms.

The **type system** $\cap J$ is given by the following typing rules.

$$\frac{}{x : [\sigma] \vdash x : \sigma}\ (\textsc{var}) \qquad \frac{\Gamma; x : \mathscr{M} \vdash t : \sigma}{\Gamma \vdash \lambda x.t : \mathscr{M} \to \sigma}\ (\textsc{abs}) \qquad \frac{(\Gamma_i \vdash t : \sigma_i)_{i\in I} \qquad I \neq \emptyset}{\uplus_{i\in I}\Gamma_i \vdash t : [\sigma_i]_{i\in I}}\ (\textsc{many})$$

$$\frac{\Gamma \vdash t : \#([\mathscr{M}_i \to \tau_i]_{i\in I}) \qquad \Delta \vdash u : \#(\sqcup_{i\in I}\mathscr{M}_i) \qquad \Lambda; x : [\tau_i]_{i\in I} \vdash r : \sigma}{\Gamma \uplus \Delta \uplus \Lambda \vdash t(u, x.r) : \sigma}\ (\textsc{app})$$

The use of the choice operator in rule (app) is subtle. If I is empty, then the multiset $[\mathscr{M}_i \to \tau_i]_{i\in I}$ typing t as well as the multiset $\sqcup_{i\in I}\mathscr{M}_i$ typing u are both empty, so that the choice operator must be used to type both terms. If I is not empty, then the multiset typing t is non-empty as well. However, the multiset typing u may or not be empty, *e.g.* if $[[\,] \to \alpha]$ types t. As before, the size of a type derivation $\mathrm{sz}(\Phi)$ is equal to the number of occurrences rules in the set $\{(\textsc{var}), (\textsc{abs}), (\textsc{app})\}$.

System $\cap J$ lacks weakening: it is *relevant*. Unlike the other systems in the thesis, not designed for strong normalization, the relevance property here uses an equality: this is because every subterm, every variable in particular, must be typed.

Lemma 4.22 (Relevance). *If* $\Gamma \Vdash t : \sigma$, *then* $\mathrm{fv}(t) = \mathrm{dom}(\Gamma)$.

Notice that the typing rules (and the choice operator) force all the subterms of a typed term to be also typed. Moreover, if $I = \emptyset$ in rule (app), then the types of t and u are not necessarily related. Indeed, let $t := \delta(\delta, x.z)$. Then t is djn-strongly-normalizing so it must be typed in system $\cap J$. However, since the set I of $x : [\tau_i]_{i\in I}$ in the typing of $r = z$ is necessarily empty (see lemma 4.22), then the unrelated types $\#([\mathscr{M}_i \to \tau_i]_{i\in I})$ and $\#(\sqcup_{i\in I}\mathscr{M}_i)$ of the two occurrences of δ witness the fact that these subterms will never interact during the reduction of t. Indeed, the term t can be typed as follows, where $\rho_i := [[\sigma_i] \to \sigma_i, \sigma_i] \to \sigma_i$ and $\tau_i := [\sigma_i] \to \sigma_i$, for $i = 1, 2$:

$$\frac{\dfrac{\dfrac{\emptyset \vdash \delta : \rho_1}{\emptyset \vdash \delta : [\rho_1]}\ (\textsc{many}) \qquad \dfrac{\emptyset \vdash \delta : \rho_2}{\emptyset \vdash \delta : [\rho_2]}\ (\textsc{many}) \qquad \dfrac{}{z : [\tau]; x : [\,] \vdash z : \tau}\ (\textsc{var})}{z : [\tau] \vdash \delta(\delta, x.z) : \tau}\ (\textsc{app})$$

where δ is typed with ρ_i as follows:

$$\dfrac{\dfrac{\dfrac{}{y : [\tau_i] \vdash y : \tau_i}\,(\textsc{var})}{y : [\tau_i] \vdash y : [\tau_i]}\,(\textsc{many}) \qquad \dfrac{\dfrac{}{y : [\sigma_i] \vdash y : \sigma_i}\,(\textsc{var})}{y : [\sigma_i] \vdash y : [\sigma_i]}\,(\textsc{many}) \qquad \dfrac{}{w : [\sigma_i] \vdash w : \sigma_i}\,(\textsc{var})}{\dfrac{y : [[\sigma_i] \to \sigma_i, \sigma_i] \vdash y(y, w.w) : \sigma_i}{\emptyset \vdash \lambda y.y(y, w.w) : [[\sigma_i] \to \sigma_i, \sigma_i] \to \sigma_i}\,(\textsc{abs})}\,(\textsc{app})$$

Lemma 4.23 (Split).

(i) *If* $\Gamma \Vdash^n t : \mathscr{M}$, *then for any decomposition* $\mathscr{M} = \sqcup_{i\in I}\mathscr{M}_i$ *where* $\mathscr{M}_i \neq \emptyset$ *for all* $i \in I$, *then we have* $\Gamma_i \Vdash^{n_i} t : \mathscr{M}_i$ *such that* $\sum_{i\in I} n_i = n$ *and* $\uplus_{i\in I}\Gamma_i = \Gamma$.

(ii) *If* $\Gamma_i \Vdash^{n_i} t : \mathscr{M}_i$ *for all* $i \in I$ *and* $I \neq \emptyset$, *then* $\Gamma \Vdash^n t : \mathscr{M}$, *where* $\mathscr{M} = \sqcup_{i\in I}\mathscr{M}_i$, $n = \sum_{i\in I} n_i$ *and* $\Gamma = \uplus_{i\in I}\Gamma_i$.

Proof. Straightforward by induction on the derivations. □

From now on we use the following notation to indicate that we have used lemma 4.23(ii).

$$\dfrac{(\Gamma_i \vdash t : \mathscr{M}_i)_{i\in I}}{\uplus_{i\in I}\Gamma_i \vdash t : \sqcup_i \mathscr{M}_i}$$

4.4.2 The Characterization of Strong djn-Normalization

The soundness property 4.32 is based on lemma 4.28, based in turn on lemma 4.24.

Lemma 4.24 (Substitution lemma). *Let* $t, u \in \mathrm{T}_J$ *with* $x \in \mathrm{fv}(t)$. *If both* $\Gamma; x : \mathscr{M} \Vdash^n t : \sigma$ *and* $\Delta \Vdash^m u : \mathscr{M}$ *hold, then* $\Gamma \uplus \Delta \Vdash^k t\{x/u\} : \sigma$ *where* $k = n + m - |\mathscr{M}|$.

Proof. By induction on the type derivation of t. We extend the statement to derivations ending with (many), for which the property is straightforward by the *i.h.*

Case $t = x$. Then $n = 1$ and by hypothesis $\Gamma = \emptyset$ and $\mathscr{M} = [\sigma]$ (so that $|\mathscr{M}| = 1$). Moreover, $\Delta \Vdash^m_{\cap J} u : \mathscr{M}$ necessarily comes from $\Delta \Vdash^m_{\cap J} u : \sigma$ by rule (many). Let $k = m$, then we conclude $\emptyset \uplus \Delta \Vdash^{1+m-1} u : \sigma = \Gamma \uplus \Delta \Vdash^k x\{x/u\} : \sigma$.

Case $t = \lambda y.s$ where $y \neq x$ and $y \notin \mathrm{fv}(u)$. By definition we have $\sigma = \mathscr{N} \to \tau$ and $\Gamma; x : \mathscr{M}; y : \mathscr{N} \Vdash^{n-1} s : \tau$.

By the *i.h.* $(\Gamma; y : \mathscr{N}) \uplus \Delta \Vdash^{k'} s\{x/u\} : \tau$ with $k' = n - 1 + m - |\mathscr{M}|$. By the relevance lemma 4.22 $y \notin \mathrm{dom}(\Delta)$ so that $(\Gamma; y : \mathscr{N}) \uplus \Delta = \Gamma \uplus \Delta; y : \mathscr{N}$. By rule (abs) we obtain $\Gamma \uplus \Delta \Vdash^{k'+1} \lambda y.s\{x/u\} : \mathscr{N} \to \tau$. Let $k = k' + 1$. We conclude because $\lambda y.s\{x/u\} = (\lambda y.s)\{x/u\}$ and $k = k' + 1 = n + m - |\mathscr{M}|$.

Case $t = s(o, y.r)$, where $y \neq x$ and $y \notin \mathrm{fv}(u)$. We only detail the case where $x \in \mathrm{fv}(s) \cap \mathrm{fv}(o) \cap \mathrm{fv}(r)$, the other cases being similar. By definition we have $\Gamma_1; x : \mathscr{M}_1 \Vdash^{n_1} s : \#([\mathscr{N}_i \longrightarrow \tau_i]_{i\in I})$, $\Gamma_2; x : \mathscr{M}_2 \Vdash^{n_2} o : \#(\sqcup_{i\in I}\mathscr{N}_i)$ and $\Gamma_3; x : \mathscr{M}_3; y : [\tau_i]_{i\in I} \Vdash^{n_3} r : \sigma$ where $\Gamma = \Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3$, $\mathscr{M} = \mathscr{M}_1 \sqcup \mathscr{M}_2 \sqcup \mathscr{M}_3$, and $n = 1 + n_1 + n_2 + n_3$. Moreover, by lemma 4.23 we have $\Delta_1 \Vdash^{m_1} u : \mathscr{M}_1$, $\Delta_2 \Vdash^{m_2} u : \mathscr{M}_2$ and $\Delta_3 \Vdash^{m_3} u : \mathscr{M}_3$ where $\Delta = \Delta_1 \uplus \Delta_2 \uplus \Delta_3$ and $m = m_1 + m_2 + m_3$. The *i.h.* gives $\Gamma_1 \uplus \Delta_1 \Vdash^{k_1} s\{x/u\} : \#([\mathscr{N}_i \longrightarrow \tau_i]_{i\in I})$, $\Gamma_2 \uplus \Delta_2 \Vdash^{k_2} o\{x/u\} : \#(\sqcup_{i\in I}\mathscr{N}_i)$ and $\Gamma_3 \uplus \Delta_3; y : [\tau_i]_{i\in I} \Vdash^{k_3} r\{x/u\} : \sigma$, where $k_i = n_i + m_i - |\mathscr{M}_i|$ for $i = 1, 2, 3$. Then we have a derivation $\Gamma_1 \uplus \Delta_1 \uplus \Gamma_2 \uplus \Delta_2 \uplus \Gamma_3 \uplus \Delta_3 \Vdash^{k} s\{x/u\}(o\{x/u\}, y.r\{x/u\}) : \sigma$ where $k = 1 +_{i=1,2,3} k_i$. We conclude since $\Gamma \uplus \Delta = \Gamma_1 \uplus \Delta_1 \uplus \Gamma_2 \uplus \Delta_2 \uplus \Gamma_3 \uplus \Delta_3$, $s(o, y.r)\{x/u\} = s\{x/u\}(o\{x/u\}, y.r\{x/u\})$ and $k = 1 +_{i=1,2,3} k_i = 1 +_{i=1,2,3} (n_i + m_i - |\mathscr{M}_i|) = n + m - |\mathscr{M}|$.

□

Lemma 4.25. *Let $t \in \mathtt{T}_J$, and $\mathtt{D}$ a list context. Then $\Gamma \Vdash^n_{\cap J} \mathtt{D}\langle\lambda x.t\rangle : \sigma$ if and only if $\Gamma \Vdash^n_{\cap J} \lambda x.\mathtt{D}\langle t\rangle : \sigma$.*

Proof. Both implications are proved by induction on $\mathtt{D}$. The base case $\mathtt{D} = \Diamond$ is trivial. Notice that we always have $\sigma = \mathscr{N} \longrightarrow \rho$. Let consider the inductive case $\mathtt{D} = s(u, y.\mathtt{D}')$.

We first consider the left-to-right implication. So that let $\Gamma \Vdash^n \mathtt{D}\langle\lambda x.t\rangle : \sigma$. We have the following derivation, with $n = k + l + m + 1$.

$$\dfrac{\Pi \Vdash^k s : \#([\mathscr{M}_i \longrightarrow \tau_i]_{i\in I}) \qquad \Delta \Vdash^l u : \#(\sqcup_{i\in I}\mathscr{M}_i) \qquad \Lambda; y : [\tau_i]_{i\in I} \Vdash^m \mathtt{D}'\langle\lambda x.t\rangle : \sigma}{\Pi \uplus \Delta \uplus \Lambda \vdash s(u, y.\mathtt{D}'\langle\lambda x.t\rangle) : \sigma}$$

The *i.h.* gives a derivation $\Lambda; y : [\tau_i]_{i\in I} \Vdash^m \lambda x.\mathtt{D}'\langle t\rangle : \sigma$ and thus a derivation $\Lambda; y : [\tau_i]_{i\in I}; x : \mathscr{N} \Vdash^{m-1} \mathtt{D}'\langle t\rangle : \rho$. By α-conversion, $y \notin \mathrm{fv}(s) \cup \mathrm{fv}(u)$, so that $y \notin \mathrm{dom}(\Pi \uplus \Delta)$ by lemma 4.22. We can then build the following derivation of the same size:

$$\dfrac{\dfrac{\Pi \Vdash^k s : \#([\mathscr{M}_i \longrightarrow \tau_i]_{i\in I}) \qquad \Delta \Vdash^l u : \#(\sqcup_{i\in I}\mathscr{M}_i) \qquad \Lambda; y : [\tau_i]_{i\in I}; x : \mathscr{N} \Vdash^m \mathtt{D}'\langle t\rangle : \rho}{\Pi \uplus \Delta \uplus (\Lambda; x : \mathscr{N}) \vdash s(u, y.\mathtt{D}'\langle t\rangle) : \rho}}{\Pi \uplus \Delta \uplus \Lambda \vdash \lambda x.s(u, y.\mathtt{D}'\langle t\rangle) : \sigma}$$

For the right-to-left implication, we build the first derivations from the second similarly to the previous case. □

By nature, subject reduction (or expansion) in the quantitative type system for strong normalization does not hold. Indeed, all subterms are typed, even the ones that will be erased. In most cases, these subterms have free variables, that are typed in the environment. When the term is erased, some bits of the environment are lost which means that the typing is not preserved by reduction steps.

Example 4.26. Let $t = \lambda x.\mathtt{I}(y, z.z) \longrightarrow_{\mathrm{d}\beta} \mathtt{I}$. The term t can be typed with the derivation below, with environment $y : [\sigma]$. However, by relevance, the term $\mathtt{I}$ can only be typed with

an empty environment since that term has no free variables.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : [\tau] \vdash x : \tau}}{\vdash \mathtt{I} : [\tau] \to \tau}}{\vdash \lambda x.\mathtt{I} : [\,] \to [\tau] \to \tau} \qquad \dfrac{\dfrac{}{y : [\sigma] \vdash y : \sigma}}{y : [\sigma] \vdash y : [\,]} \qquad \dfrac{}{z : [[\tau] \to \tau] \vdash z : [\tau] \to \tau}}{y : [\sigma] \vdash (\lambda x.\mathtt{I})(y, z.z) : [\tau] \to \tau}$$

We thus prove subject reduction only for *non-erasing* steps.

Definition 4.27 (Erasing step). A reduction step $t_1 \to_{\mathrm{djn}} t_2$ is said to be **erasing** iff the reduced dβ-redex in t_1 is of the form $\mathtt{D}\langle\lambda x.t\rangle(u, y.r)$ with $x \notin \mathrm{fv}(t)$ or $y \notin \mathrm{fv}(r)$.

Lemma 4.28 (Non-erasing subject reduction). *Let* $\Gamma \Vdash^{n_1}_{\cap J} t_1 : \sigma$. *If* $t_1 \to_{\mathrm{djn}} t_2$ *is a non-erasing step, then* $\Gamma \Vdash^{n_2}_{\cap J} t_2 : \sigma$ *with* $n_1 > n_2$.

Proof. By induction on $t_1 \to t_2$.

Case $t_1 = \mathtt{D}_{\mathtt{n}}\langle\lambda x.t\rangle(u, y.r) \mapsto_\beta r\{y/\mathtt{D}_{\mathtt{n}}\langle t\{x/u\}\rangle\} = t_2$. Because the step is non-erasing, the types of y and x are not empty by lemma 4.22, so that we have the following derivation, with $\Gamma = \uplus_{i\in I}\Sigma_i \uplus_{i\in I} \Delta_i \uplus \Lambda$, $n_1 = \sum_{i\in I}(n^i_t + 1 + n^i_u) + n_r + 1$ and $I \neq \emptyset$.

$$\dfrac{\dfrac{\left(\Sigma_i \Vdash^{n^i_\lambda} \mathtt{D}_{\mathtt{n}}\langle\lambda x.t\rangle : \mathscr{M}_i \to \tau_i\right)_{i\in I}}{\uplus_{i\in I}\Sigma_i \vdash \mathtt{D}_{\mathtt{n}}\langle\lambda x.t\rangle : [\mathscr{M}_i \to \tau_i]_{i\in I}} \qquad \dfrac{(\Delta_i \Vdash^{n^i_u} u : \mathscr{M}_i)_{i\in I}}{\uplus_{i\in I}\Delta_i \vdash u : \sqcup_{i\in I}\mathscr{M}_i} \qquad \Lambda; y : [\tau_i]_{i\in I} \Vdash^{n_r} r : \sigma}{\uplus_{i\in I}\Sigma_i \uplus_{i\in I} \Delta_i \uplus \Lambda \vdash \mathtt{D}_{\mathtt{n}}\langle\lambda x.t\rangle(u, y.r) : \sigma}$$

For each $i \in I$, lemma 4.25 gives a derivation $\Sigma_i \Vdash^{n^i_\lambda} \lambda x.\mathtt{D}_{\mathtt{n}}\langle t\rangle : \mathscr{M}_i \to \tau_i$ and therefore we have a derivation $\Sigma_i; x : \mathscr{M}_i \Vdash^{n^i_t} \mathtt{D}_{\mathtt{n}}\langle t\rangle : \tau_i$ where $n^i_t = n^i_\lambda - 1$. Moreover, the substitution lemma 4.24 gives $\Sigma_i \uplus \Delta_i \Vdash^{k_i} \mathtt{D}_{\mathtt{n}}\langle t\rangle\{x/u\} : \tau_i$, where $k_i = n^i_t + n^i_u - |\mathscr{M}_i|$, so that we have a derivation $\uplus_{i\in I}\Sigma_i \uplus_{i\in I} \Delta_i \Vdash^{+_{i\in I}k_i} \mathtt{D}_{\mathtt{n}}\langle t\rangle\{x/u\} : [\tau_i]_{i\in I}$. Applying the substitution lemma 4.24 again gives $\Gamma \Vdash^{n_2} t_2 = r\{y/\mathtt{D}_{\mathtt{n}}\langle t\rangle\{x/u\}\} : \sigma$ with $n_2 = n_r +_{i\in I} k_i < n_1$.

Case $t_1 = \lambda x.t \to \lambda x.t' = t_2$, where $t \to t'$. By hypothesis, we have $\sigma = \mathscr{M} \to \tau$ and $\Gamma; x : \mathscr{M} \Vdash^{n_1-1} t : \sigma$. By the *i.h.* we have $\Gamma; x : \mathscr{M} \Vdash^{k} t' : \tau$ for $n_1 - 1 > k$. We can build a derivation of size $n_2 = k + 1$ and we get $n_1 > n_2$.

Case $t_1 = t(u, x.r)$ and the reduction is internal. By hypothesis, we have the derivations $\Sigma \Vdash^{n_t} t : \#([\mathscr{M}_i \to \tau_i]_{i\in I})$, $\Delta \Vdash^{n_u} u : \#(\sqcup_{i\in I}\mathscr{M}_i)$ and $\Lambda; x : [\tau_i]_{i\in I} \Vdash^{n_r} r : \sigma$ with $\Gamma = \Sigma \uplus \Delta \uplus \Lambda$ and $n_1 = 1 + n_t + n_u + n_r$.

Subcase $t_1 \to t'(u, x.r) = t_2$, where $t \to t'$. If $I \neq \emptyset$, we have $\Sigma = \uplus_{i\in I}\Sigma_i$, $n_t = \sum_{i\in I} n^i_t$ and derivations $\Sigma_i \Vdash^{n^i_t} t : \mathscr{M}_i \to \tau_i$. If $I = \emptyset$, we have $\#([\mathscr{M}_i \to \tau_i]_{i\in I}) = [\tau]$ and a derivation $\Sigma \Vdash^{n_t} t : \tau$. In both cases, we apply the *i.h.* and derive $\Sigma \Vdash^{k} t' :$

$\#([\mathscr{M}_i \longrightarrow \tau_i]_{i\in I})$ with $k < n_t$. We can build a derivation of size $n_2 = 1+k+n_u+n_r$ and we get $n_1 > n_2$.

Subcase $t_1 \longrightarrow t(u', x.r) = t_2$, where $u \longrightarrow u'$. Let $\#(\sqcup_{i\in I}\mathscr{M}_i) = [\rho_j]_{j\in J}$. In particular, if $\sqcup_{i\in I}\mathscr{M}_i = [\,]$, then J is a singleton. We have $\Delta = \uplus_{j\in J}\Delta_j$, $n_u = \sum_{j\in J} n_u^j$ and derivations $\Delta_j \Vdash^{n_u^j} u : \rho_j$. We apply the *i.h.* and derive $\Delta \Vdash^{k} u : \#(\sqcup_{i\in I}\mathscr{M}_i)$ with $k < n_u$. We can build a derivation of size $n_2 = 1 + n_t + k + n_r$ and we get $n_1 > n_2$.

Subcase $t_1 \longrightarrow t(u, x.r') = t_2$, where $r \longrightarrow r'$. By the *i.h.* we have $\Lambda; x : [\tau_i]_{i\in I} \Vdash^{k} r : \sigma$ with $k < n_r$. We can build a derivation of size $n_2 = 1 + n_t + n_u + k$ and we get $n_1 > n_2$. □

Although subject reduction does not always hold, the characterization of normalizable terms as typable should. To prove this, we need a weaker form of subject reduction: the fact that the right-hand term of an erasing reduction is still typed. This is the goal of the following lemma. Notice that we do not consider any reduction, but one occurring inside a weak context $\mathtt{W}$. We will use the syntax of terms given in (4.2) on page 202 to conclude the proof (lemma 4.31).

Lemma 4.29. *Let* $t = \mathtt{D}_n\langle\lambda x.s\rangle(u, y.r)$ *and* $t' = r\{y/\mathtt{D}_n\langle s\{x/u\}\rangle\}$ *such that* $\Gamma \Vdash^{k}_{\cap J} \mathtt{W}\langle t\rangle : \sigma$. *Then,*

(i) *If* $y \notin \mathrm{fv}(r)$, *then there are typing derivations for* $\mathtt{W}\langle t'\rangle = \mathtt{W}\langle r\rangle, \mathtt{D}_n\langle s\rangle$ *and* u *having measures* $k_{\mathtt{W}\langle t'\rangle}, k_{\mathtt{D}_n\langle s\rangle}$ *and* k_u *resp. such that* $k > 1 + k_{\mathtt{W}\langle t'\rangle} + k_{\mathtt{D}_n\langle s\rangle} + k_u$.

(ii) *If* $y \in \mathrm{fv}(r)$ *and* $x \notin \mathrm{fv}(s)$, *then there are typing derivations for* $\mathtt{W}\langle t'\rangle = \mathtt{W}\langle r\{y/\mathtt{D}_n\langle s\rangle\}\rangle$ *and* u *having measures* $k_{\mathtt{W}\langle t'\rangle}$ *and* k_u *resp. such that* $k > 1 + k_{\mathtt{W}\langle t'\rangle} + k_u$.

Proof. We prove a stronger statement: the derivation for $\mathtt{W}\langle t'\rangle$ is of the shape $\Gamma' \Vdash^{k_{\mathtt{W}\langle t'\rangle}}_{\cap J} \mathtt{W}\langle t'\rangle : \sigma$ with the same σ but $\Gamma' \sqsubseteq \Gamma$. We proceed by induction on $\mathtt{W}$:

Case $\mathtt{W} = \Diamond$. (i) The derivation of t has premises $\Gamma_\lambda \Vdash^{k_\lambda} \mathtt{D}_n\langle\lambda x.s\rangle : \tau$, $\Delta \Vdash^{k_u} u : \rho$ and $\Lambda \Vdash^{k_{t'}} r : \sigma$, for some appropriate τ and ρ, such that $\Gamma = \Gamma_\lambda \uplus \Delta \uplus \Lambda$ and $k = k_\lambda + k_u + k_{t'} + 1$. By lemma 4.25, we have a derivation $\Gamma_\lambda \Vdash^{k_\lambda} \lambda x.\mathtt{D}_n\langle s\rangle : \tau$. Then, $\tau = \mathscr{M} \longrightarrow \tau'$ with $\mathscr{M}$ potentially empty and we have a derivation $\Gamma_\lambda; x : \mathscr{M} \Vdash^{k_\lambda - 1} \mathtt{D}_n\langle s\rangle : \tau'$. Let $k_{\mathtt{D}_n\langle s\rangle} = k_\lambda - 1$. We have $k > 1 + k_{t'} + k_{\mathtt{D}_n\langle s\rangle} + k_u$ and we let $\Gamma' = \Lambda$ since $t' = r$. We can conclude since $\Gamma' \sqsubseteq \Gamma$.

(ii) The derivation of t has premises $\Gamma_\lambda \Vdash^{k_\lambda} \mathtt{D}_n\langle\lambda x.s\rangle : [[\,] \longrightarrow \tau_i]_{i\in I}$, and thus $(\Gamma^i_\lambda \Vdash^{k^i_\lambda} \mathtt{D}_n\langle\lambda x.s\rangle : [\,] \longrightarrow \tau_i)_{i\in I}$ with $\Gamma_\lambda = \uplus_{i\in I}\Gamma^i_\lambda$ and $k_\lambda = +_{i\in I}k^i_\lambda$, as well as $\Delta \Vdash^{k_u} u : \rho$ and $\Lambda; [\tau_i]_{i\in I} \Vdash^{k_r} r : \sigma$, where $\Gamma = \Gamma_\lambda \uplus \Delta \uplus \Lambda$ and $k = k_\lambda + k_u + k_r + 1 = k$ and $I \neq \emptyset$. By lemma 4.25, we have derivations $(\Gamma^i_\lambda \Vdash^{k^i_\lambda} \lambda x.\mathtt{D}_n\langle s\rangle : [\,] \longrightarrow \tau_i)_{i\in I}$ and thus derivations $(\Gamma^i_\lambda \Vdash^{k^i_\lambda - 1} \mathtt{D}_n\langle s\rangle : \tau_i)_{i\in I}$. By rule (MANY) we have a derivation $\Gamma_\lambda \Vdash^{k_{\mathtt{D}_n\langle s\rangle}} \mathtt{D}_n\langle s\rangle : [\tau_i]_{i\in I}$ where $k_{\mathtt{D}_n\langle s\rangle} = +_{i\in I}(k^i_\lambda - 1) = k_\lambda - |I|$. Using the substitution lemma 4.24 we construct a derivation $\Lambda \uplus \Gamma_\lambda \Vdash^{k_{t'}} r\{y/\mathtt{D}_n\langle s\rangle\} : \sigma$

with $k_{t'} = k_r + k_{\mathtt{D_n}\langle s\rangle} - |I|$. We have $k = k_{\mathtt{D_n}\langle s\rangle} + |I| + k_u + k_r + 1 = 1 + k_{t'} + 2 \times |I| + k_u > 1 + k_{t'} + k_u$. We let $\Gamma' = \Lambda \uplus \Gamma_\lambda$. We can then conclude since $\Gamma' \sqsubseteq \Gamma$.

Case $\mathtt{W} = \mathtt{W}'(u', z.r')$. The derivation of $\mathtt{W}\langle t\rangle$ has three premises of the form: $\Gamma_1 \Vdash^{k_{\mathtt{W}'\langle t\rangle}} \mathtt{W}'\langle t\rangle : \#([\mathscr{M}_i \longrightarrow \tau_i]_{i\in I})$, $\Delta \Vdash^{k_{u'}} u' : \#(\sqcup_{i\in I}\mathscr{M}_i)$ and $\Lambda; z : [\tau_i]_{i\in I} \Vdash^{k_{r'}} r' : \sigma$ such that $k = 1 + k_{\mathtt{W}'\langle t\rangle} + k_{u'} + k_{r'}$ and $\Gamma = \Gamma_1 \uplus \Delta \uplus \Lambda$. By *i.h.* we get from the first premise:

1. In cases (i) and (ii) a derivation $\Gamma_2 \Vdash^{k_{\mathtt{W}'\langle t'\rangle}} \mathtt{W}'\langle t'\rangle : \#([\mathscr{M}_i \longrightarrow \tau_i]_{i\in I})$ such that $\Gamma_2 \sqsubseteq \Gamma_1$ and a typing derivation for u of measure k_u.
2. In case (i) a typing derivation for $\mathtt{D_n}\langle s\rangle$ of measure $k_{\mathtt{D_n}\langle s\rangle}$ and the fact that $k_{\mathtt{W}'\langle t\rangle} > 1 + k_{\mathtt{W}'\langle t'\rangle} + k_{\mathtt{D_n}\langle s\rangle} + k_u$.
3. In case (ii) the fact that $k_{\mathtt{W}'\langle t\rangle} > 1 + k_{\mathtt{W}'\langle t'\rangle} + k_u$.

Using the type derivations for $\mathtt{W}'\langle t'\rangle$, u' and r' we can build a derivation $\Gamma_2 \uplus \Delta \uplus \Lambda \Vdash^{k_{\mathtt{W}\langle t'\rangle}} \mathtt{W}'\langle t'\rangle(u', z.r') : \sigma$, where $k_{\mathtt{W}\langle t'\rangle} = 1 + k_{k_{\mathtt{W}'\langle t'\rangle}} + k_{u'} + k_{r'}$. We have $\Gamma_2 \uplus \Delta \uplus \Lambda \sqsubseteq \Gamma$. In case (i) we can conclude because $k = 1 + k_{\mathtt{W}'\langle t\rangle} + k_{u'} + k_{r'} >_{i.h.} 1 + (1 + k_{\mathtt{W}'\langle t'\rangle} + k_{\mathtt{D_n}\langle s\rangle} + k_u) + k_{u'} + k_{r'} = 1 + k_{\mathtt{W}\langle t'\rangle} + k_{\mathtt{D_n}\langle s\rangle} + k_u$. In case (ii) in the same way, but without adding $k_{\mathtt{D_n}\langle s\rangle}$ in the sum.

Case $\mathtt{W} = \mathtt{n}(u', z.\mathtt{W}')$. The derivation of $\mathtt{W}\langle t\rangle$ has premises: $\Gamma_\mathtt{n} \Vdash^{k_\mathtt{n}} \mathtt{n} : \#([\mathscr{M}_i \longrightarrow \tau_i]_{i\in I})$, $\Delta \Vdash^{k_{u'}} u' : \#(\sqcup_{i\in I}\mathscr{M}_i)$ and $\Lambda_1; z : [\tau_i]_{i\in I} \Vdash^{k_{\mathtt{W}'\langle t\rangle}} \mathtt{W}'\langle t\rangle : \sigma$. We have $\Gamma = \Gamma_\mathtt{n} \uplus \Delta \uplus \Lambda_1$ and $k = 1 + k_\mathtt{n} + k_{u'} + k_{\mathtt{W}'\langle t\rangle}$. By the *i.h.* we get from the third premise:

1. In cases (i) and (ii) a derivation $\Lambda_2; z : [\tau_i]_{i\in I'} \Vdash^{k_{\mathtt{W}'\langle t'\rangle}} \mathtt{W}'\langle t'\rangle : \sigma$ such that $\Lambda_2 \sqsubseteq \Lambda_1$, and $I' \subseteq I$ (I' possibly empty), and a typing derivation for u of measure k_u.
2. In case (i) a typing derivation for $\mathtt{D_n}\langle s\rangle$ of measure $k_{\mathtt{D_n}\langle s\rangle}$ and the fact that $k_{\mathtt{W}'\langle t\rangle} > 1 + k_{\mathtt{W}'\langle t'\rangle} + k_{\mathtt{D_n}\langle s\rangle} + k_u$.
3. In case (ii) the fact that $k_{\mathtt{W}'\langle t\rangle} > 1 + k_{\mathtt{W}'\langle t'\rangle} + k_u$.

To build a derivation for $\mathtt{W}\langle t'\rangle$, we need in particular derivations of type $\#([\mathscr{M}_i \longrightarrow \tau_i]_{i\in I'})$ for $\mathtt{n}$ and $\#(\sqcup_{i\in I'}\mathscr{M}_i)$ for u'.

Subcase $I' \neq \emptyset$. Then $\#([\mathscr{M}_i \longrightarrow \tau_i]_{i\in I'}) = [\mathscr{M}_i \longrightarrow \tau_i]_{i\in I'}$ and by lemma 4.23 it is possible to construct a derivation $\Gamma'_\mathtt{n} \Vdash^{k'_\mathtt{n}} \mathtt{n} : [\mathscr{M}_i \longrightarrow \tau_i]_{i\in I'}$ from the original one for $\mathtt{n}$ verifying $\Gamma'_\mathtt{n} \sqsubseteq \Gamma_\mathtt{n}$ and $k'_\mathtt{n} \leq k_\mathtt{n}$. For u' we build a derivation $\Delta' \Vdash^{k'_{u'}} u' : \#(\sqcup_{i\in I'}\mathscr{M}_i)$ verifying $\Delta' \sqsubseteq \Delta$ and $k'_{u'} \leq k_{u'}$. There are three cases:

Subsubcase $(\mathscr{M}_i)_{i\in I}$ are all empty, and therefore $(\mathscr{M}_i)_{i\in I'}$ are all empty. Then we set $\#(\sqcup_{i\in I'}\mathscr{M}_i) = \#(\sqcup_{i\in I}\mathscr{M}_i)$. We take the original derivation so that $\Delta' = \Delta$, $k'_{u'} = k_{u'}$.

Subsubcase $(\mathscr{M}_i)_{i\in I'}$ are all empty but $(\mathscr{M}_i)_{i\in I}$ are not all empty. As a consequence, $\sqcup_{i\in I}\mathscr{M}_i \neq \emptyset$ and we take an arbitrary type ρ of $\sqcup_{i\in I}\mathscr{M}_i$ as a witness

for u', so that, $\Delta_\rho \Vdash^{k_\rho} u' : \rho$ holds by lemma 4.23. We have the expected derivation with rule (MANY) taking $\Delta' = \Delta_\rho$, $\#(\sqcup_{i\in I'}\mathscr{M}_i) = [\rho]$ and $k'_{u'} = k_\rho$.

Subsubcase $\#(\sqcup_{i\in I'}\mathscr{M}_i) = \sqcup_{i\in I'}\mathscr{M}_i$. By lemma 4.23 it is possible to construct the expected derivation from the original ones for u'.

Finally, we conclude by the following derivation for $\mathtt{W}\langle t'\rangle$:

$$\frac{\Gamma'_{\mathtt{n}} \Vdash^{k'_{\mathtt{n}}} \mathtt{n} : [\mathscr{M}_i \longrightarrow \tau_i]_{i\in I'} \qquad \Delta' \Vdash^{k'_{u'}} u' : \#(\sqcup_{i\in I'}\mathscr{M}_i) \qquad \Phi}{\Gamma' \vdash \mathtt{n}(u', y.\mathtt{W}'\langle t'\rangle) : \sigma}$$

where $\Phi = \Lambda_2; z : [\tau_i]_{i\in I'} \Vdash^{k_{\mathtt{W}'\langle t'\rangle}} \mathtt{W}'\langle t'\rangle : \sigma$, where $\Gamma' = \Gamma'_{\mathtt{n}} \uplus \Delta' \uplus \Lambda_2$, and the total measure of the derivation is $k_{\mathtt{W}\langle t'\rangle} = 1 + k'_{\mathtt{n}} + k'_{u'} + k_{\mathtt{W}'\langle t'\rangle}$. We have $k > 1+k'_{\mathtt{n}}+k'_{u'}+k_{\mathtt{W}'\langle t\rangle} >_{i.h.} 1+k'_{\mathtt{n}}+k'_{u'}+1+k_{\mathtt{W}'\langle t'\rangle}+k_{\mathtt{D}_{\mathtt{n}}\langle s\rangle}+k_u > 1+k_{\mathtt{W}\langle t'\rangle}+k_{\mathtt{D}_{\mathtt{n}}\langle s\rangle}+k_u$ in case (i). Similarly but without $k_{\mathtt{D}_{\mathtt{n}}\langle s\rangle}$ in case (ii). We can conclude since $\Gamma' \sqsubseteq \Gamma$.

Case $I = I' = \emptyset$. We are done by taking the original derivations.

Case $I \neq \emptyset = I'$. Let us take an arbitrary $j \in I$: the type $[\mathscr{M}_j \longrightarrow \tau_j]$ is set as a witness for $\mathtt{n}$, whose derivation $\Gamma' \Vdash^{k_{\mathtt{n}'}} \mathtt{n}' : [\mathscr{M}_j \longrightarrow \tau_j]$ is obtained from the derivation $\Gamma_{\mathtt{n}} \Vdash^{k_{\mathtt{n}}} \mathtt{n} : \#([\mathscr{M}_i \longrightarrow \tau_i]_{i\in I})$ by the split lemma 4.23. For u', we take as a witness an arbitrary $\rho \in \#(\sqcup_{i\in I}\mathscr{M}_i)$ and we set $\#(\sqcup_{i\in I'}\mathscr{M}_i) = [\rho]$. If $\sqcup_{i\in I}\mathscr{M}_i = [\,]$, then ρ is the original witness. Otherwise ρ is a type of one of the $\mathscr{M}_i$'s. In both cases we use the split lemma 4.23 to get a derivation $\Delta' \Vdash^{k'_{u'}} u' : [\rho]$ where $\Delta' \sqsubseteq \Delta$ and $k'_{u'} \leq k_{u'}$. Using the type derivation given by the *i.h.* for $\mathtt{W}'\langle t'\rangle$, we conclude by the following derivation for $\mathtt{W}\langle t'\rangle$:

$$\frac{\Gamma'_{\mathtt{n}} \Vdash^{k_{\mathtt{n}'}} \mathtt{n}' : [\mathscr{M}_j \longrightarrow \tau_j] \qquad \Delta' \Vdash^{k'_{u'}} u' : [\rho] \qquad \Lambda_2; z : [\tau_i]_{i\in I'} \Vdash^{k_{\mathtt{W}'\langle t'\rangle}} \mathtt{W}'\langle t'\rangle : \sigma}{\Gamma' \vdash \mathtt{n}(u', y.\mathtt{W}'\langle t'\rangle) : \sigma}$$

where $\Gamma'_{\mathtt{n}} \sqsubseteq \Gamma_{\mathtt{n}}$, $\Delta' \sqsubseteq \Delta$, $k'_{\mathtt{n}} \leq k_{\mathtt{n}}$, $k'_{u'} \leq k_{u'}$. We have $\Gamma' = \Gamma'_{\mathtt{n}} \uplus \Delta' \uplus \Lambda_2 \sqsubseteq \Gamma$.

In case (i) we can conclude because $k = 1 + k_{\mathtt{n}} + k_{u'} + k_{\mathtt{W}'\langle t\rangle} > 1 + k'_{\mathtt{n}} + k'_{u'} + (1 + k_{\mathtt{W}'\langle t'\rangle} + k_{\mathtt{D}_{\mathtt{n}}\langle s\rangle} + k_u) = 1 + k_{\mathtt{W}\langle t'\rangle} + k_{\mathtt{D}_{\mathtt{n}}\langle s\rangle} + k_u$. Similarly but without $k_{\mathtt{D}_{\mathtt{n}}\langle s\rangle}$ in case (ii). □

We now finish the proof of soundness by proving that all typable terms have a finite reduction length, that is bounded by the maximum number of djn-steps until normal form. This maximal length is written $||t||_{\mathrm{djn}}$ for a term t.

Lemma 4.30. *The following equalities hold:*

$$\begin{aligned}
||x||_{\mathrm{djn}} &= 0\\
||\lambda x.t||_{\mathrm{djn}} &= ||t||_{\mathrm{djn}}\\
||\mathtt{n}(u, x.r)||_{\mathrm{djn}} &= ||\mathtt{n}||_{\mathrm{djn}} + ||u||_{\mathrm{djn}} + ||r||_{\mathrm{djn}}\\
||\mathtt{W}\langle \mathtt{D}_{\mathtt{n}}\langle \lambda x.s\rangle(u, y.r)\rangle||_{\mathrm{djn}} &= \begin{cases} 1 + ||\mathtt{W}\langle r\{y/\mathtt{D}_{\mathtt{n}}\langle s\{x/u\}\rangle\}\rangle||_{\mathrm{djn}} & \text{if } x \in \mathrm{fv}(s) \text{ and } y \in \mathrm{fv}(r)\\ 1 + ||\mathtt{W}\langle r\{y/\mathtt{D}_{\mathtt{n}}\langle s\rangle\}\rangle||_{\mathrm{djn}} + ||u||_{\mathrm{djn}} & \text{if } x \notin \mathrm{fv}(s) \text{ and } y \in \mathrm{fv}(r)\\ 1 + ||\mathtt{W}\langle r\rangle||_{\mathrm{djn}} + ||\mathtt{D}_{\mathtt{n}}\langle s\rangle||_{\mathrm{djn}} + ||u||_{\mathrm{djn}} & \text{if } y \notin \mathrm{fv}(r)\end{cases}
\end{aligned}$$

Proof. A consequence of definition 4.17 and theorem 4.21. □

Lemma 4.31. *If* $\Gamma \Vdash^{k}_{\cap J} t : \sigma$, *then* $||t||_{\mathrm{djn}} \leq k$.

Proof. We proceed by induction on k and we reason by case analysis on t according to the alternative grammar ((4.2) on page 202).

Case $t = x$. The derivation is just an axiom and $k = 1$, so that $||x||_{\mathrm{djn}} = 0 < 1 = k$.

Case $t = \lambda x.u$. There is a typing derivation for u of size $k - 1 < k$. The *i.h.* gives $||u||_{\mathrm{djn}} \leq k - 1$, so that $||t||_{\mathrm{djn}} = ||u||_{\mathrm{djn}} \leq k$.

Case $t = \mathtt{n}(u, x.r)$. There are typings of $\mathtt{n}$, u and r with measures $k_{\mathtt{n}}$, k_u and k_r resp. such that $k = 1 + k_{\mathtt{n}} + k_u + k_r$. We then get $||t||_{\mathrm{djn}} = ||\mathtt{n}||_{\mathrm{djn}} + ||u||_{\mathrm{djn}} + ||r||_{\mathrm{djn}} \leq_{i.h.} k_{\mathtt{n}} + k_u + k_r \leq k$.

Case $t = \mathtt{W}\langle \mathtt{D}_{\mathtt{n}}\langle \lambda x.s\rangle (u, y.r)\rangle$. There are three possible cases:

- **Subcase** $x \in \mathrm{fv}(s)$ and $y \in \mathrm{fv}(r)$. Then $t \longrightarrow_{\mathrm{djn}} \mathtt{W}\langle r\{y/\mathtt{D}_{\mathtt{n}}\langle s\{x/u\}\rangle\}\rangle = t_0$ and $||t||_{\mathrm{djn}} = 1 + ||t_0||_{\mathrm{djn}}$. Moreover, the subject reduction lemma 4.28 gives $\Gamma \Vdash^{k'}_{\cap J} t_0 : \sigma$ with $k' < k$. By the *i.h.* we have $||t_0||_{\mathrm{djn}} \leq k'$. Thus we conclude $||t||_{\mathrm{djn}} = 1 + ||t_0||_{\mathrm{djn}} \leq 1 + k' \leq k$.
- **Subcase** $y \notin \mathrm{fv}(r)$. Then $t \longrightarrow_{\mathrm{djn}} \mathtt{W}\langle r\rangle = t_0$ and $||t||_{\mathrm{djn}} = 1 + ||t_0||_{\mathrm{djn}} + ||\mathtt{D}_{\mathtt{n}}\langle s\rangle||_{\mathrm{djn}} + ||u||_{\mathrm{djn}}$. By subject reduction for erasing steps (lemma 4.29) there are typings of t_0, $\mathtt{D}_{\mathtt{n}}\langle s\rangle$ and u having measures k_{t_0}, $k_{\mathtt{D}_{\mathtt{n}}\langle s\rangle}$ and k_u resp. such that $k > 1 + k_{t_0} + k_{\mathtt{D}_{\mathtt{n}}\langle s\rangle} + k_u$. Thus we conclude $||t||_{\mathrm{djn}} = 1 + ||t_0||_{\mathrm{djn}} + ||\mathtt{D}_{\mathtt{n}}\langle s\rangle||_{\mathrm{djn}} + ||u||_{\mathrm{djn}} \leq_{i.h.} 2 + k_{t_0} + k_{\mathtt{D}_{\mathtt{n}}\langle s\rangle} + k_u \leq k$.
- **Subcase** $x \notin \mathrm{fv}(s)$ and $y \in \mathrm{fv}(r)$. Then $t \longrightarrow_{\mathrm{djn}} \mathtt{W}\langle r\{y/\mathtt{D}_{\mathtt{n}}\langle s\rangle\}\rangle = t_0$ and $||t||_{\mathrm{djn}} = 1 + ||t_0||_{\mathrm{djn}} + ||u||_{\mathrm{djn}}$. By subject reduction for erasing steps (lemma 4.29) there are typings of t_0 and u having measures k_{t_0} and k_u resp. such that $k > 1 + k_{t_0} + k_u$. Thus we conclude $||t||_{\mathrm{djn}} = 1 + ||t_0||_{\mathrm{djn}} + ||u||_{\mathrm{djn}} \leq_{i.h.} 1 + k_{t_0} + k_u < k$. □

As a corollary we obtain:

Property 4.32 (Soundness for λJ_n). *If* t *is* $\cap J$*-typable, then* $t \in \mathrm{SN}(\mathrm{djn})$.

The completeness lemma 4.36 is based on typability of normal forms (lemma 4.33) and non-erasing subject expansion (lemma 4.35). This last one is based itself on anti-substitution (lemma 4.34).

Lemma 4.33 (Typing normal forms).

(i) *For all* $t \in \mathrm{NF}_{\mathrm{djn}}$, *there exists* Γ, σ *such that* $\Gamma \Vdash_{\cap J} t : \sigma$.

(ii) *For all* $t \in \mathrm{NE}_{\mathrm{djn}}$, *for all* σ, *there exists* Γ *such that* $\Gamma \Vdash_{\cap J} t : \sigma$.

Proof. By simultaneous induction on $t \in \mathrm{NF}_{\mathrm{djn}}$ and $t \in \mathrm{NE}_{\mathrm{djn}}$.
First, the cases relative to statement (i).

Case $t = x$. Pick an arbitrary σ. We have $x : [\sigma] \Vdash x : \sigma$ by rule (VAR).

Case $t = \lambda x.s$ where $s \in \mathrm{NF}_{\mathrm{djn}}$. By *i.h.* on s there exists Γ' and τ such that $\Gamma' \Vdash s : \tau$. Let Γ and $\mathscr{N}$ be such that $\Gamma' = \Gamma; x : \mathscr{N}$ ($\mathscr{N}$ is possibly empty). We get $\Gamma \Vdash \lambda x.s : \mathscr{N} \to \tau$ by rule (ABS). We conclude by taking $\sigma = \mathscr{N} \to \tau$.

Case $t = s(u, y.r)$ where $u, r \in \mathrm{NF}_{\mathrm{djn}}$ and $s \in \mathrm{NE}_{\mathrm{djn}}$. By the *i.h.* on r there is a derivation of $\Lambda' \Vdash r : \sigma$. Let Λ and $[\tau_i]_{i\in I}$ be such that $\Lambda' = \Lambda; y : [\tau_i]_{i\in I}$. Now we construct a derivation $\Pi \Vdash s : \#([[\,] \to \tau_i]_{i\in I})$ as follows.

- If $I = \emptyset$, then the *i.h.* on s gives a derivation $\Pi \Vdash s : \tau$ and we use rule (MANY) to get $\Pi \Vdash s : [\tau]$. We conclude by setting $\#([[\,] \to \tau_i]_{i\in I}) = [\tau]$.
- If $I \neq \emptyset$, then the *i.h.* on s gives a derivation of $\Pi_i \Vdash s : [\,] \to \tau_i$ for each $i \in I$. We take $\Pi = \uplus_{i\in I}\Pi_i$ and we conclude with rule (MANY) since $\#([[\,] \to \tau_i]_{i\in I}) = [[\,] \to \tau_i]_{i\in I}$.

Finally, the *i.h.* on u gives a derivation $\Delta \Vdash u : \rho$ from which we get $\Delta \Vdash u : [\rho]$, by choosing $\#(\sqcup_{i\in I}[\,]) = [\rho]$. We conclude with rule (APP) as follows:

$$\frac{\Pi \Vdash s : \#([[\,] \to \tau_i]_{i\in I}) \qquad \Delta \Vdash u : \#(\sqcup_{i\in I}[\,]) \qquad \Lambda; y : [\tau_i]_{i\in I} \Vdash r : \sigma}{\Pi \uplus \Delta \uplus \Lambda \vdash s(u, y.r) : \sigma}$$

Next, the cases relative to statement (ii).

Case $t = x$. As seen above, given an arbitrary type σ, we can take $\Gamma = [\sigma]$.

Case $t = s(u, y.r)$ where $u \in \mathrm{NF}_{\mathrm{djn}}$ and $s, r \in \mathrm{NE}_{\mathrm{djn}}$. Pick an arbitrary σ. The proof proceeds *ipsis verbis* as in the case $t = s(u, y.r)$ above. □

Lemma 4.34 (Anti-substitution). *If* $\Gamma \Vdash t\{x/u\} : \sigma$ *where* $x \in \mathrm{fv}(t)$, *then there exist* Γ_t, Γ_u *and* $\mathscr{M} \neq [\,]$ *such that* $\Gamma_t; x : \mathscr{M} \Vdash t : \sigma$, $\Gamma_u \Vdash u : \mathscr{M}$ *and* $\Gamma = \Gamma_t \uplus \Gamma_u$.

Proof. By induction on the derivation $\Gamma \Vdash t\{x/u\} : \sigma$. We extend the statement to derivations ending with (MANY), for which the property is straightforward by the *i.h.* We reason by cases on t.

Case $t = x$. Then $t\{x/u\} = u$. We take $\Gamma_t = \emptyset$, $\Gamma_u = \Gamma$, $\mathscr{M} = [\sigma]$, and we have $x : [\sigma] \Vdash x : \sigma$ by rule (VAR) and $\Gamma \Vdash u : \mathscr{M}$ by rule (MANY) on the derivation of the hypothesis.

Case $t = \lambda y.s$ where $y \neq x$ and $y \notin \mathrm{fv}(u)$ and $x \in \mathrm{fv}(s)$. Then $t\{x/u\} = \lambda y.s\{x/u\}$. We have $\sigma = \mathscr{N} \to \tau$ and $\Gamma; y : \mathscr{N} \Vdash s\{x/u\} : \tau$.

By the *i.h.* there exists $\Gamma', \Gamma_u, \mathscr{M} \neq [\,]$ such that $\Gamma'; y : \mathscr{N}; x : \mathscr{M} \Vdash s : \tau, \Gamma_u \Vdash u : \mathscr{M}$, and $\Gamma; y : \mathscr{N} = (\Gamma'; y : \mathscr{N}) \uplus \Gamma_u$. Moreover, by α-conversion and lemma 4.22 we know that $y \notin \mathrm{dom}(\Gamma_u)$ so that $\Gamma = \Gamma' \uplus \Gamma_u$. We conclude by deriving $\Gamma'; y : \mathscr{N} \Vdash \lambda x.s : \mathscr{N} \to \tau$ with rule (ABS). Indeed, by letting $\Gamma_t = \Gamma'$ we have $\Gamma = \Gamma_t \uplus \Gamma_u$ as required.

Case $t = t_1(t_2, y.r)$, where $y \neq x$, $y \notin \mathrm{fv}(u)$ and $x \in \mathrm{fv}(t_1) \cup \mathrm{fv}(t_2) \cup (\mathrm{fv}(r) \setminus y)$. We detail the case where $x \in \mathrm{fv}(t_1) \cap \mathrm{fv}(t_2) \cap \mathrm{fv}(r)$, the other cases are similar. By construction, we have derivations $\Gamma_1 \Vdash t_1\{x/u\} : \#([\mathscr{N}_i \to \tau_i]_{i\in I})$, $\Gamma_2 \Vdash t_2\{x/u\} : \#(\sqcup_{i\in I}\mathscr{N}_i)$ and $\Gamma_3; y : [\tau_i]_{i\in I} \Vdash r\{x/u\} : \sigma$, with $\Gamma = \Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3$.

By the *i.h.* there are environments $\Gamma_{t_1}, \Gamma_{t_2}, \Gamma_r, \Gamma_u^1, \Gamma_u^2, \Gamma_u^3$ and multitypes $\mathscr{M}_1, \mathscr{M}_2, \mathscr{M}_3$ all different from $[\,]$ such that $\Gamma_{t_1}; x : \mathscr{M}_1 \Vdash t_1 : \#([\mathscr{N}_i \to \tau_i]_{i\in I})$, $\Gamma_{t_2}; x : \mathscr{M}_2 \Vdash t_2 : \#(\sqcup_{i\in I}\mathscr{N}_i)$, $\Gamma_r; x : \mathscr{M}_3 \Vdash r : \sigma$, $\Gamma_u^1 \Vdash u : \mathscr{M}_1$, $\Gamma_u^2 \Vdash u : \mathscr{M}_2$, $\Gamma_u^3 \Vdash u : \mathscr{M}_3$ and $\Gamma_1 = \Gamma_{t_1} \uplus \Gamma_u^1$, $\Gamma_2 = \Gamma_{t_2} \uplus \Gamma_u^2$, $\Gamma_3 = \Gamma_r \uplus \Gamma_u^3$. Let $\Gamma_t = \Gamma_{t_1} \uplus \Gamma_{t_2} \uplus \Gamma_r$, $\Gamma_u = \Gamma_u^1 \uplus \Gamma_u^2 \uplus \Gamma_u^3$ and $\mathscr{M} = \mathscr{M}_1 \sqcup \mathscr{M}_2 \sqcup \mathscr{M}_3$. We can build a derivation $\Gamma_t; x : \mathscr{M} \Vdash t_1(t_2, y.r) : \sigma$ with rule (APP) and a derivation $\Gamma_u \Vdash u : \mathscr{M}$ with lemma 4.23. We conclude since $\Gamma = \Gamma_1 \uplus \Gamma_2 \uplus \Gamma_3 = \Gamma_{t_1} \uplus \Gamma_u^1 \uplus \Gamma_{t_2} \uplus \Gamma_u^2 \uplus \Gamma_r \uplus \Gamma_u^3 = \Gamma_t \uplus \Gamma_u$. □

Lemma 4.35 (Non-erasing subject expansion). *If* $\Gamma \Vdash_{\cap J} t_2 : \sigma$ *and* $t_1 \to_{\mathrm{djn}} t_2$ *is a non-erasing step, then* $\Gamma \Vdash_{\cap J} t_1 : \sigma$.

Proof. By induction on $t_1 \to_{\mathrm{djn}} t_2$.

Case $t_1 = \mathtt{D}\langle\lambda x.t\rangle(u, y.r) \mapsto_\beta r\{y/\mathtt{D}\langle t\{x/u\}\rangle\} = t_2$. Since the reduction is non-erasing, we have $y \in \mathrm{fv}(r)$ and $x \in \mathrm{fv}(t)$. By lemma 4.34, there exists Γ_r, Γ' and $\mathscr{N}$ such that $\Gamma_r; y : \mathscr{N} \Vdash r : \sigma$, $\Gamma' \Vdash \mathtt{D}\langle t\{x/u\}\rangle : \mathscr{N}$ and $\Gamma = \Gamma' \uplus \Gamma_r$. Let $\mathscr{N} = [\tau_i]_{i\in I} \neq [\,]$ since $y \in \mathrm{fv}(r)$. By rule (MANY), we have a decomposition $(\Gamma_i' \Vdash \mathtt{D}\langle t\{x/u\}\rangle : \tau_i)_{i\in I}$ with $\Gamma' = \uplus_{i\in I}\Gamma_i'$. Since $\mathtt{D}\langle t\{x/u\}\rangle = \mathtt{D}\langle t\rangle\{x/u\}$, by lemma 4.34 again, for each $i \in I$ there are Γ_t^i, Γ_u^i and $\mathscr{M}_i \neq [\,]$ such that $\Gamma_t^i; x : \mathscr{M}_i \Vdash \mathtt{D}\langle t\rangle : \tau_i$, $\Gamma_u^i \Vdash u : \mathscr{M}_i$ and $\Gamma_i' = \Gamma_t^i \uplus \Gamma_u^i$. By rule (ABS) followed by (MANY), there are derivations $\Gamma_t \Vdash \lambda x.\mathtt{D}\langle t\rangle : [\mathscr{M}_i \to \tau_i]_{i\in I}$ with $\Gamma_t = \uplus_{i\in I}\Gamma_t^i$. By lemma 4.25, there is a derivation $\Gamma_t \Vdash \mathtt{D}\langle\lambda x.t\rangle : [\mathscr{M}_i \to \tau_i]_{i\in I}$. Finally, by lemma 4.23, there is a derivation $\Gamma_u \Vdash u : \sqcup_{i\in I}\mathscr{M}_i$ with $\Gamma_u = \uplus_{i\in I}\Gamma_u^i$. Since neither I nor the $\mathscr{M}_i$'s are empty, the choice operator is in both cases the identity and we can build the following derivation using rule (APP):

$$\frac{\Gamma_t \Vdash \mathtt{D}\langle\lambda x.t\rangle : [\mathscr{M}_i \to \tau_i]_{i\in I} \qquad \Gamma_u \Vdash u : \sqcup_{i\in I}\mathscr{M}_i \qquad \Gamma_r; y : [\tau_i]_{i\in I} \Vdash r : \sigma}{\Gamma \vdash \mathtt{D}\langle\lambda x.t\rangle(u, y.r) : \sigma}$$

We verify $\Gamma = \Gamma' \uplus \Gamma_r = \uplus_{i\in I}\Gamma_i' \uplus \Gamma_r = \uplus_{i\in I}(\Gamma_t^i \uplus \Gamma_u^i) \uplus \Gamma_r = \Gamma_t \uplus \Gamma_u \uplus \Gamma_r$.

Case $t_1 = \lambda x.t$ and $t_1 = t(u, x.r)$ and the reduction is internal. These cases are direct by the *i.h.* □

We cannot conclude completeness straightaway, given that subject expansion was only shown for non-erasing cases. Instead, we prove that from any term on the right of a reduction,

we can build a derivation for the term on the left. We rely on the previous lemma for the non-erasing steps, and construct derivations for erasing ones, in which the typing environment grows with anti-reduction. We use the inductive characterization of strong normalization ISN(djn) to recognize the left terms that are indeed strongly normalizing, which are the only ones for which we can build a typing derivation.

Lemma 4.36 (Completeness for λJ_n). *If $t \in \mathrm{SN}(\mathrm{djn})$, then t is $\cap J$-typable.*

Proof. In the statement, we replace SN(djn) by ISN(djn), using theorem 4.21. We use induction on ISN(djn) to show the following stronger property $\mathscr{P}$: If $t \in \mathrm{ISN}(\mathrm{djn})$ then there are Γ, σ such that $\Gamma \Vdash t : \sigma$, and if $t \in \mathtt{n}$, then the property holds for any σ.

Case $t = x$. We get $x : [\sigma] \vdash x : \sigma$ by rule (VAR), for any σ.

Case $t = \lambda x.s$, where $s \in \mathrm{ISN}(\mathrm{djn})$. By the *i.h.* we have $\Delta \Vdash s : \tau$. Let us write Δ as $\Gamma; x : \mathscr{M}$, where $\mathscr{M}$ is possibly empty. Then we get $\Gamma \Vdash \lambda x.s : \sigma$, where $\sigma = \mathscr{M} \longrightarrow \tau$, by using rule (ABS) on the previous derivation.

Case $t = \mathtt{n}(u, x.r)$, where $\mathtt{n}, u, r \in \mathrm{ISN}(\mathrm{djn})$ and $r \in \mathrm{NF}_{\mathrm{lr}}$. By the *i.h.* there are derivations $\Delta \Vdash u : \rho$ and $\Lambda; x : [\tau_i]_{i\in I} \Vdash r : \sigma$ with I possibly empty. Moreover, $\Delta \Vdash u : [\rho]$ holds by rule (MANY). If $r \in \mathtt{n}$, we have a derivation for any type σ by the stronger *i.h.*

We now construct a derivation $\Pi \Vdash \mathtt{n} : \#([[\,] \longrightarrow \tau_i]_{i\in I})$ as follows:

- If $I = \emptyset$, then the *i.h.* gives $\Pi \Vdash \mathtt{n} : \tau$ for an arbitrary τ, and then we obtain $\Pi \Vdash \mathtt{n} : [\tau]$ by rule (MANY). We conclude by setting $\#([[\,] \longrightarrow \tau_i]_{i\in I}) = [\tau]$.
- If $I \neq \emptyset$, then by the stronger *i.h.* we can derive $\Pi_i \Vdash \mathtt{n} : [\,] \longrightarrow \tau_i$ for each $i \in I$. We take $\Pi = \uplus_{i\in I}\Pi_i$ and we conclude with rule (MANY) since $\#([[\,] \longrightarrow \tau_i]_{i\in I}) = [[\,] \longrightarrow \tau_i]_{i\in I}$.

We conclude with rule (APP) as follows, by setting in particular $\#(\sqcup_{i\in I}[\,]) = [\rho]$.

$$\frac{\Pi \Vdash \mathtt{n} : \#([[\,] \longrightarrow \tau_i]_{i\in I}) \qquad \Delta \Vdash u : \#(\sqcup_{i\in I}[\,]) \qquad \Lambda; y : [\tau_i]_{i\in I} \Vdash r : \sigma}{\Pi \uplus \Delta \uplus \Lambda \vdash s(u, y.r) : \sigma}$$

Case $t \notin \mathrm{NF}_{\mathrm{lr}}$. That is, $t = \mathtt{W}\langle \mathtt{D}_{\mathtt{n}}\langle \lambda x.s\rangle(u, y.r)\rangle$, where $t' = \mathtt{W}\langle r\{y/\mathtt{D}_{\mathtt{n}}\langle s\rangle\{x/u\}\}\rangle \in \mathrm{ISN}(\mathrm{djn})$, $\mathtt{D}_{\mathtt{n}}\langle s\rangle \in \mathrm{ISN}(\mathrm{djn})$, and $u \in \mathrm{ISN}(\mathrm{djn})$. Notice that $t \notin \mathtt{n}$ by lemma 4.14. By the *i.h.* $t', \mathtt{D}_{\mathtt{n}}\langle s\rangle$ and u are typable. We show by a second induction on $\mathtt{W}$ that $\Sigma \Vdash t' : \sigma$ implies $\Gamma \Vdash t : \sigma$, for some Γ. For the base case $\mathtt{W} = \diamond$, there are three cases.

Subcase $x \in \mathrm{fv}(s)$ and $y \in \mathrm{fv}(r)$. Since $t' = r\{y/\mathtt{D}_{\mathtt{n}}\langle s\rangle\{x/u\}\}$ is typable and $t \longrightarrow_\beta t'$, then t is also typable with Σ and σ by the non-erasing subject expansion lemma 4.35. We conclude with $\Gamma = \Sigma$.

Subcase $x \notin \mathrm{fv}(s)$ and $y \in \mathrm{fv}(r)$. Then $t' = r\{y/\mathtt{D}_n\langle s\rangle\}$ and by *i.h.* there is a derivation $\Sigma \Vdash r\{y/\mathtt{D}_n\langle s\rangle\} : \sigma$. The anti-substitution lemma 4.34, gives $\Lambda; y : \mathscr{N} \Vdash r : \sigma$, $\Pi \Vdash \mathtt{D}_n\langle s\rangle : \mathscr{N}$ with $\Sigma = \Lambda \uplus \Pi$. Let $\mathscr{N} = [\sigma_i]_{i\in I}$. We have $I \neq \emptyset$ by lemma 4.22 since $y \in \mathrm{fv}(r)$. By the Split lemma 4.23 there are derivations $\Pi_i \Vdash \mathtt{D}_n\langle s\rangle : \sigma_i$ such that $\Pi = \uplus_{i\in I}\Pi_i$. Since $u \in \mathrm{ISN(djn)}$, the *i.h.* gives a derivation $\Delta \Vdash u : \rho$ and by rule (MANY) we get $\Delta \Vdash u : [\rho]$. Moreover, lemma 4.22 implies that $x \notin \mathrm{dom}(\Pi_i)$ for each $i \in I$ because $x \notin \mathrm{fv}(\mathtt{D}_n\langle s\rangle)$, then we can construct derivations $(\Pi_i \Vdash \lambda x.\mathtt{D}_n\langle s\rangle : [\,] \to \sigma_i)_{i\in I}$. By lemma 4.25 applied for each $i \in I$, we retrieve $(\Pi_i \Vdash \mathtt{D}_n\langle \lambda x.s\rangle : [\,] \to \sigma_i)_{i\in I}$. And by rule (MANY) we get $\Pi \Vdash \mathtt{D}_n\langle \lambda x.s\rangle : [[\,] \to \sigma_i]_{i\in I}$. Finally, since $\#([[\,] \to \sigma_i]_{i\in I}) = [[\,] \to \sigma_i]_{i\in I}$, it is sufficient to set $\#(\sqcup_{i\in I}[\,]) = [\rho]$ and we obtain the following derivation:

$$\frac{\Pi \Vdash \mathtt{D}_n\langle \lambda x.s\rangle : \#([[\,] \to \sigma_i]_{i\in I}) \qquad \Delta \Vdash u : \#(\sqcup[\,]) \qquad \Lambda; y : [\,] \Vdash r : \sigma}{\Gamma \vdash \mathtt{D}_n\langle \lambda x.s\rangle(u, y.r) : \sigma}$$

where $\Gamma = \Pi \uplus \Delta \uplus \Lambda$. We then conclude.

Subcase $y \notin \mathrm{fv}(r)$. Since $t' = r\{y/\mathtt{D}_n\langle s\rangle\{x/u\}\}$ is typable and $t' = r$, then there is a derivation $\Lambda \Vdash r : \sigma$ where $y \notin \mathrm{dom}(\Lambda)$ holds by relevance (so that $\Sigma = \Lambda$). We can then write $\Lambda; y : [\,] \Vdash r : \sigma$. We construct a derivation of t ending with rule (APP). For this we need two witness derivations for u and $\mathtt{D}_n\langle \lambda x.s\rangle$. Since $u \in \mathrm{ISN(djn)}$, the *i.h.* gives a derivation $\Delta \Vdash u : \rho$, and then we get $\Delta \Vdash u : [\rho]$ by application of rule (MANY). Similarly, since $\mathtt{D}_n\langle s\rangle \in \mathrm{ISN(djn)}$, the *i.h.* gives a derivation $\Pi; x : \mathscr{M} \Vdash \mathtt{D}_n\langle s\rangle : \tau$ where $\mathscr{M}$ can be empty. Thus $\Pi \Vdash \lambda x.\mathtt{D}_n\langle s\rangle : \mathscr{M} \to \tau$. By lemma 4.25, we get $\Pi \Vdash \mathtt{D}_n\langle \lambda x.s\rangle : \mathscr{M} \to \tau$, and then we get $\Pi \Vdash \mathtt{D}_n\langle \lambda x.s\rangle : [\mathscr{M} \to \tau]$ by application of rule (MANY). Finally, by setting $\#([\,]) = [\mathscr{M} \to \tau]$ and $\#(\sqcup[\,]) = [\rho]$ we construct the following derivation:

$$\frac{\Pi \Vdash \mathtt{D}_n\langle \lambda x.s\rangle : \#([\,]) \qquad \Delta \Vdash u : \#(\sqcup[\,]) \qquad \Lambda; y : [\,] \Vdash r : \sigma}{\Gamma \vdash \mathtt{D}_n\langle \lambda x.s\rangle(u, y.r) : \sigma}$$

where $\Gamma = \Pi \uplus \Delta \uplus \Lambda$. We then conclude.

Then, there are two inductive cases. We extend the second *i.h.* to multi-types trivially.

Subcase $\mathtt{W} = \mathtt{W}'(u', z.r')$. Let consider the terms $t_0 = \mathtt{W}'\langle \mathtt{D}_n\langle \lambda x.s\rangle(u, y.r)\rangle$ and $t_1 = \mathtt{W}'\langle r\{y/\mathtt{D}_n\langle s\rangle\{x/u\}\}\rangle$ so that $t = t_0(u', z.r')$ and $t' = t_1(u', z.r')$. The type derivation of t' ends with a rule (APP) with the premises: $\Sigma_1 \Vdash t_1 : \#([\mathscr{M}_i \to \tau_i]_{i\in I})$, $\Delta \Vdash u' : \#(\sqcup_{i\in I}\mathscr{M}_i)$ and $z : [\tau_i]_{i\in I}; \Lambda \Vdash r' : \sigma$, where $\Sigma = \Sigma_1 \uplus \Delta \uplus \Lambda$. By the second *i.h.* we get a derivation $\Gamma_0 \Vdash t_0 : \#([\mathscr{M}_i \to \tau_i]_{i\in I})$ for some Γ_0. We build a derivation for t with type σ ending with rule (APP) and using the derivations

for t_0 and the ones for u' and r', so that the corresponding typing environment is $\Gamma = \Gamma_0 \uplus \Delta \uplus \Lambda$. We then conclude.

Subcase $\mathtt{W} = \mathtt{n}(u', z.\mathtt{W}')$. Let t_0, t_1 be the same as before so that $t = \mathtt{n}(u', z.t_0)$ and $t' = \mathtt{n}(u', z.t_1)$. We detail the case where $z \in \mathrm{fv}(t_0)$ and $z \notin \mathrm{fv}(t_1)$, the other ones being similar to case 1. The type derivation of t' is as follows, with $\Sigma = \Gamma_{\mathtt{n}} \uplus \Delta \uplus \Sigma'$.

$$\frac{\Gamma_{\mathtt{n}} \Vdash \mathtt{n} : [\tau] \qquad \Delta \Vdash u' : [\rho] \qquad \Sigma' \Vdash t_1 : \sigma}{\Gamma_{\mathtt{n}} \uplus \Delta \uplus \Sigma' \Vdash s'(u', z.t_1) : \sigma}\ (\textsc{app})$$

By the second *i.h.* we have a derivation $z : [\tau_i]_{i\in I}; \Gamma' \Vdash t_0 : \sigma$ for some Γ'. Also by relevance lemma 4.22) we have $I \neq \emptyset$. By the *i.h.* on property $\mathscr{P}$, we can build derivations $\Pi_i \Vdash \mathtt{n} : [\,] \to \tau_i$ for each $i \in I$ and thus a derivation $\Pi \Vdash \mathtt{n} : [[\,] \to \tau_i]_{i\in I}$ by rule (MANY) with $\Pi = \uplus_{i\in I}\Pi_i$. Setting $\#(\sqcup_{i\in I}[\,]) = [\rho]$, we then build the following derivation:

$$\frac{\Pi \Vdash \mathtt{n} : [[\,] \to \tau_i]_{i\in I} \qquad \Delta \Vdash u' : \#(\sqcup_{i\in I}[\,]) \qquad z : [\tau_i]_{i\in I}; \Gamma' \Vdash t_0 : \sigma}{\Gamma \vdash \mathtt{n}(u', z.t_0) : \sigma}\ (\textsc{app})$$

where $\Gamma = \Pi \uplus \Delta \uplus \Gamma'$. We thus conclude. □

We finally obtain:

Theorem 4.37 (Characterization). *System $\cap J$ characterizes strong normalization, i.e. t is $\cap J$-typable if and only if t is $\to_{\mathrm{djn}}$-normalizing. Moreover, if $\Gamma \Vdash^n t : \sigma$ then the number of reduction steps in any reduction sequence from t to normal form is bounded by n.*

Proof. Soundness holds by property 4.32, while completeness holds by lemma 4.36. The bound is given by lemma 4.31. □

4.4.3 Quantitative Behavior of π

We have mentioned already that π is rejected by the quantitative type systems $\cap J$ for CbN. Concretely, this happens in the critical case when $x \notin \mathrm{fv}(r)$ and $y \in \mathrm{fv}(r')$ in

$$t_0 = t(u, x.r)(u', y.r') \to_\pi t(u, x.r(u', y.r')) = t_1$$

Example 4.38. We take $t_1 = x(y, a.z)(w, b.b(b, c.c)) \to_\pi x(y, a.z(w, b.b(b, c.c))) = t_2$. Let $\rho_1 = [\sigma] \to \tau$ and $\rho_2 = [\sigma] \to [\tau] \to \tau$. For each $i \in \{1, 2\}$ let $\Delta_i = x : [\sigma_1]; y : [\sigma_2]; z : [\rho_i]$. Consider

$$\Psi = \frac{b : [[\tau] \to \tau] \Vdash b : [[\tau] \to \tau] \qquad b : [\tau] \Vdash b : [\tau] \qquad \dfrac{}{c : [\tau] \vdash c : \tau}}{b : [[\tau] \to \tau, \tau] \vdash b(b, c.c) : \tau}$$

and the derivation Φ_i for $i \in \{1, 2\}$:

$$\Phi_i = \dfrac{x : [\sigma_1] \Vdash x : [\sigma_1] \qquad y : [\sigma_2] \Vdash y : [\sigma_2] \qquad \dfrac{}{z : [\rho_i] \vdash z : \rho_i}}{\Delta_i \vdash x(y, a.z) : \rho_i}$$

Then, for the term t_1, we have the following derivation:

$$\dfrac{\dfrac{\Phi_1 \qquad \Phi_2}{\Delta_1 \uplus \Delta_2 \vdash x(y, a.z) : [\rho_1, \rho_2]} \qquad w : [\sigma, \sigma] \Vdash w : [\sigma, \sigma] \qquad \Psi}{\Gamma_1 \vdash x(y, a.z)(w, b.b(b, c.c)) : \tau}$$

where $\Gamma_1 = z : [\rho_1, \rho_2]; w : [\sigma, \sigma]; x : [\sigma_1, \sigma_1]; y : [\sigma_2, \sigma_2]$.

While for the term t_2, we have:

$$\dfrac{x : [\sigma_1] \Vdash x : [\sigma_1] \qquad y : [\sigma_2] \Vdash y : [\sigma_2] \qquad \Phi}{\Gamma_2 \vdash x(y, a.z(w, b.b(b, c.c))) : \tau}$$

where

$$\Phi = \dfrac{z : [\rho_1, \rho_2] \Vdash z : [\rho_1, \rho_2] \qquad w : [\sigma, \sigma] \Vdash w : [\sigma, \sigma] \qquad \Psi}{\Gamma_2 \vdash z(w, b.b(b, c.c)) : \tau}$$

and $\Gamma_2 = z : [\rho_1, \rho_2]; w : [\sigma, \sigma]; x : [\sigma_1]; y : [\sigma_2]$.

Thus, the multiset types of x and y in Γ_1 and Γ_2 resp. are not the same. Despite the fact that the step $t_1 \to_\pi t_2$ does not erase any subterm, the typing environment is losing quantitative information.

Notice that by replacing non-idempotent types by idempotent ones, subject reduction (and expansion) would work for π-reduction: by assigning sets to variables instead of multisets, Γ_1 and Γ_2 would be equal.

Despite the fact that quantitative subject reduction fails for some π-steps, the following weaker property is sufficient to recover (qualitative) soundness of our typing system $\cap J$ w.r.t. the reduction relation $\to_{\mathrm{jn}}$. Soundness will be used later in section 4.6 to show equivalence between SN(djn) and SN(jn).

Lemma 4.39 (Typing behavior of π-reduction). *Let* $\Gamma \Vdash^{n_1}_{\cap J} t_1 : \sigma$. *If* $t_1 = t(u, x.r)(u', y.r') \mapsto_\pi t_2 = t(u, x.r(u', y.r'))$, *then there are* n_2 *and* $\Sigma \sqsubseteq \Gamma$ *such that* $\Sigma \Vdash^{n_2}_{\cap J} t_2 : \sigma$ *with* $n_1 \geq n_2$.

Proof. The derivation of t_1 ends with (APP), with $\Gamma = \Gamma' \uplus \Delta_{u'} \uplus \Lambda_{r'}$ and $n_1 = 1 + n' + n_{u'} + n_{r'}$.

$$\dfrac{\Gamma' \Vdash^{n'} t(u, x.r) : \#([\mathscr{M}_i \to \tau_i]_{i \in I}) \quad \Delta_{u'} \Vdash^{n_{u'}} u' : \#(\sqcup_{i \in I} \mathscr{M}_i) \quad \Lambda_{r'}; y : [\tau_i]_{i \in I} \Vdash^{n_{r'}} r' : \sigma}{\Gamma \vdash t(u, x.r)(u', y.r') : \sigma}$$

There are two possibilities.

Case $I \neq \emptyset$. Then $\#([\mathscr{M}_i \to \tau_i]_{i \in I}) = [\mathscr{M}_i \to \tau_i]_{i \in I}$ and for each $i \in I$ there is one

derivation of $t(u, x.r)$ having the following form:

$$\frac{\Gamma_t^i \Vdash^{n_t^i} t : \#([\mathcal{N}_j \longrightarrow \rho_j]_{j\in J_i}) \quad \Delta_u^i \Vdash^{n_u^i} u : \#(\sqcup_{j\in J_i}\mathcal{N}_j) \quad \Lambda_r^i; x : [\rho_j]_{j\in J_i} \Vdash^{n_r^i} r : \mathscr{M}_i \longrightarrow \tau_i}{\Gamma_t^i \uplus \Delta_u^i \uplus \Lambda_r^i \vdash t(u, x.r) : \mathscr{M}_i \longrightarrow \tau_i}\ (\textsc{app})$$

where $\Gamma' = \uplus_{i\in I}(\Gamma_t^i \uplus \Delta_u^i \uplus \Lambda_r^i)$ and $n' = \sum_{i\in I} n_t^i + n_u^i + n_r^i$. From $(\Lambda_r^i; x : [\rho_j]_{j\in J_i} \Vdash^{n_r^i} r : \mathscr{M}_i \longrightarrow \tau_i)_{i\in I}$ we can construct a derivation $\Phi_r = \uplus_{i\in I}\Lambda_r^i; x : [\rho_j]_{j\in J} \Vdash^{+_{i\in I}n_r^i} r : [\mathscr{M}_i \longrightarrow \tau_i]_{i\in I}$ using rule (MANY), where $J = \uplus_{i\in I} J_i$. We then construct the following derivation:

$$\Psi = \frac{\Phi_r \qquad \Delta'_u \Vdash^{n_{u'}} u' : \#(\sqcup_{i\in I}\mathscr{M}_i) \qquad \Lambda_{r'}; y : [\tau_i]_{i\in I} \Vdash^{n_{r'}} r' : \sigma}{\uplus_{i\in I}\Lambda_r^i \uplus \Delta_{u'} \uplus \Lambda_{r'}; x : [\rho_j]_{j\in J} \vdash r(u', y.r') : \sigma}$$

We then build two derivations $\Gamma_t \Vdash^{n_t} t : \#([\mathcal{N}_j \longrightarrow \rho_j]_{j\in J})$ with $\Gamma_t \sqsubseteq \uplus_{i\in I}\Gamma_t^i$ and $n_t \le +_{i\in I} n_t^i$ and $\Delta_u \Vdash^{n_u} u : \#(\sqcup_{j\in J}\mathcal{N}_j)$ with $\Gamma_u \sqsubseteq \uplus_{i\in I}\Gamma_u^i$ and $n_u \le +_{i\in I} n_u^i$ as follows:

- If $x \in \mathrm{fv}(r)$, then all the J_i's, and thus also J, are non-empty by relevance so that $\#([\mathcal{N}_j \longrightarrow \rho_j]_{j\in J_i}) = [\mathcal{N}_j \longrightarrow \rho_j]_{j\in J_i}$. Also, $\#([\mathcal{N}_j \longrightarrow \rho_j]_{j\in J}) = [\mathcal{N}_j \longrightarrow \rho_j]_{j\in J}$. We obtain the expected derivation for t by lemma 4.23, with $\Gamma_t = \uplus_{i\in I}\Gamma_t^i$, $n_t = +_{i\in I} n_t^i$. Now for u, notice that for each $i \in I$ we can have either $\#(\sqcup_{j\in J_i}\mathcal{N}_j) = \sqcup_{j\in J_i}\mathcal{N}_j$ or, if all the $\mathcal{N}_j$'s are empty, $\#(\sqcup_{j\in J_i}\mathcal{N}_j) = [\sigma_i]$ for some σ_i derived by $\Delta_u^k \Vdash^{n_u^k} u : [\sigma_k]$. Then, there are two possibilities.
  1. If $\sqcup_{j\in J}\mathcal{N}_j = [\,]$, we take an arbitrary $k \in I$ and let $\#(\sqcup_{j\in J}\mathcal{N}_j) = [\sigma_k]$ so that we can give a derivation $\Delta_u \Vdash^{n_u} u : [\sigma_k]$ with $\Delta_u = \Delta_u^k \sqsubseteq \uplus_{i\in I}\Delta_u^i$ and $n_u = n_u^k \le +_{i\in I} n_u^i$.
  2. Otherwise, we have $\#(\sqcup_{j\in J}\mathcal{N}_j) = \sqcup_{j\in J}\mathcal{N}_j$. Let I' be the subset of I such that for each $i \in I'$ we have $\sqcup_{j\in J_i}\mathcal{N}_j \neq [\,]$ and $J' = \uplus_{i\in I'} J_i$. By Lem. 30 we build a derivation $\Delta_u \Vdash^{n_u} u : \sqcup_{j\in J'}\mathcal{N}_j$ such that $\Delta_u = \uplus_{i\in I'}\Delta_u^i \sqsubseteq \uplus_{i\in I}\Delta_u^i$ and $n_u = +_{i\in I'} n_u^i \le \uplus_{i\in I} n_u^i$.
- If $x \notin \mathrm{fv}(r)$, then all the J_i's are empty by relevance. Therefore, for each $i \in I$ there are a σ_i, σ_i' such that $\#([\mathcal{N}_j \longrightarrow \rho_j]_{j\in J_i}) = [\sigma_i]$ is derived by $\Gamma_t^i \Vdash^{n_t^i} t : [\sigma_i]$ and $\#(\sqcup_{j\in J_i}\mathcal{N}_j) = [\sigma_i']$ is derived by $\Gamma_u^i \Vdash^{n_u^i} u : [\sigma_i']$. We take an arbitrary $k \in I$ and we take $\#([\mathcal{N}_j \longrightarrow \tau_j]_{j\in J}) = [\sigma_k]$ and $\#(\sqcup_{j\in J}\mathcal{N}_j) = [\sigma_k']$. We obtain the expected derivation by taking $\Gamma_t = \Gamma_t^k \sqsubseteq \uplus_{i\in I}\Gamma_t^i$, $n_t = n_t^k \le +_{i\in I} n_t^i$, $\Gamma_u = \Gamma_u^k \sqsubseteq \uplus_{i\in I}\Gamma_u^i$ and $n_u = n_u^k \le +_{i\in I} n_u^i$.

Finally, we build the following derivation of size n_2.

$$\frac{\Gamma_t \Vdash^{n_t} t : \#([\mathcal{N}_j \longrightarrow \tau_j]_{j\in J}) \qquad \Delta_u \Vdash^{n_u} u : \#(\sqcup_{j\in J}\mathcal{N}_j) \qquad \Psi}{\Sigma \vdash t(u, x.r(u', y.r')) : \sigma}$$

We have $\Sigma = \Gamma_t \uplus \Delta_u \uplus_{i\in I} \Lambda_r^i \uplus \Delta_{u'} \uplus \Lambda_{r'} \sqsubseteq \Gamma$ and $n_2 = n_t + n_u + _{i\in I} n_r^i + n_{u'} + n_{r'} \leq n_1$.

Case $I = \emptyset$. Then there is some τ such that $\#([\mathscr{M}_i \longrightarrow \tau_i]_{i\in I}) = [\tau]$ and the derivation of $t(u, x.r)$ ends as follows:

$$\dfrac{\dfrac{\Gamma_t \Vdash^{n_t} t : \#([\mathscr{N}_j \longrightarrow \rho_j]_{j\in J}) \quad \Delta_u \Vdash^{n_u} u : \#(\sqcup_{j\in J}\mathscr{N}_j) \quad \Lambda_r; x : [\rho_j]_{j\in J} \Vdash^{n_r} r : \tau}{\Gamma_t \uplus \Delta_u \uplus \Lambda_r \vdash t(u, x.r) : \tau}\,(\textsc{app})}{\Gamma_t \uplus \Delta_u \uplus \Lambda_r \vdash t(u, x.r) : [\tau]}\,(\textsc{many})$$

with $\Gamma' = \Gamma_t \uplus \Delta_u \uplus \Lambda_r$ and $n' = n_t + n_u + n_r$.

We construct the following derivation of size n_2:

$$\dfrac{\Gamma_t \Vdash^{n_t} t : \#([\mathscr{N}_j \longrightarrow \rho_j]_{j\in J}) \qquad \Delta_u \Vdash^{n_u} u : \#(\sqcup_{j\in J}\mathscr{N}_j) \qquad \Psi}{\Sigma \vdash t(u, x.r(u', y.r')) : \sigma}$$

where

$$\Psi = \dfrac{\Lambda_r; x : [\rho_j]_{j\in J} \Vdash^{n_r} r : [\tau] \qquad \Delta'_u \Vdash^{n_{u'}} u' : \#(\sqcup_{i\in I}\mathscr{M}_i) \qquad \Lambda_{r'} \Vdash^{n_{r'}} r' : \sigma}{\Lambda_r \uplus \Delta_{u'} \uplus \Lambda_{r'}; x : [\rho_j]_{j\in J} \vdash r(u', y.r') : \sigma}$$

We have $\Sigma = \Gamma_t \uplus \Delta_u \uplus \Lambda_r \uplus \Delta_{u'} \uplus \Lambda_{r'} = \Gamma$ and $n_2 = n_t + n_u + n_r + n_{u'} + n_{r'} = n_1$. □

We have proved that reducts of typed terms are also typed. To show that typed terms terminate, we will show that the maximal length of reduction to normal form is bounded by the size of the type derivation, so finite. This is similar to what we have done for $\longrightarrow_{\text{djn}}$.

We recall that for each $t \in \text{SN}(\text{jn})$, $\|t\|_{\text{jn}}$ represents the maximal length of a jn-reduction sequence to jn-nf starting at t. We also define $\|t\|_{\text{jn}}^{\beta}$ as the maximal number of β-steps in jn-reduction sequences from t to jn-normal form. Notice that, in general, $\|t\|_{\text{jn}}^{\beta} \neq \|t\|_{\beta}$, simply because π creates β-redexes, as already discussed. Lemmas 4.40 to 4.44 serve to define $\|t\|_{\text{jn}}$ inductively. We will write $\pi(t)$ for the (unique) π-normal form of t.

Lemma 4.40. *If $t_1 \longrightarrow_\beta t_2$ and $t_1 \longrightarrow_\pi t_3$, then there is t_4 such that $t_3 \longrightarrow_\beta t_4$ and $t_2 \longrightarrow_\pi^* t_4$.*

Proof. By case analysis of the possible overlaps of the two contracted redexes. □

Lemma 4.41. *If $t_1 \longrightarrow_\beta t_2$, then there is t_3 such that $\pi(t_1) \longrightarrow_\beta t_3$ and $t_2 \longrightarrow_\pi^* t_3$.*

Proof. By induction on the reduction sequence from t_1 to $\pi(t_1)$ using lemma 4.40 for the base case. □

Lemma 4.42. *If there is a* jn*-reduction sequence ρ starting at t and containing k β-steps, then there is a* jn*-reduction sequence ρ' starting at $\pi(t)$ and also containing k β-steps.*

Proof. By induction on the (necessarily finite) reduction sequence ρ. If the length of ρ is 0, then $k = 0$ and the property is trivial. If the length of ρ is $1 + n$, we analyze the two possible cases:

1. If ρ is $t \to_\beta t'$ followed by ρ_0 of length n and containing $k_0 = k - 1$ β-steps, then the property holds for t' w.r.t. $\pi(t')$. But lemma 4.41 gives a term t'' such that $\pi(t) \to_\beta t''$ and $t' \to_\pi^* t''$. Then we construct the jn-reduction sequence $\pi(t) \to_\beta t'' \to_\pi^* \pi(t'') = \pi(t')$ followed by the one obtained by the *i.h.* This new sequence has $1 + k_0 = k$ β-steps.
2. If ρ is $t \to_\pi t'$ followed by ρ_0 of length n and containing $k_0 = k$ β-steps, then the property holds for t' w.r.t. $\pi(t')$. Since $\pi(t) = \pi(t')$, we are done by the *i.h.* □

Lemma 4.43. $||t||^\beta_{\mathrm{jn}} = ||\pi(t)||^\beta_{\mathrm{jn}}$.

Proof. First we prove $||t||^\beta_{\mathrm{jn}} \le ||\pi(t)||^\beta_{\mathrm{jn}}$. If there is a jn-reduction sequence starting at t and containing k β-steps, then the same happens for $\pi(t)$ by lemma 4.42. Next we prove $||t||^\beta_{\mathrm{jn}} \ge ||\pi(t)||^\beta_{\mathrm{jn}}$. If there is a jn-reduction sequence starting at $\pi(t)$ and containing k β-steps, then the same happens for t because it is sufficient to prefix this sequence with the steps $t \to_\pi^* \pi(t)$. We conclude $||t||^\beta_{\mathrm{jn}} = ||\pi(t)||^\beta_{\mathrm{jn}}$. □

Lemma 4.44. *If* $t \to_\pi t'$, *then* $||t||^\beta_{\mathrm{jn}} = ||t'||^\beta_{\mathrm{jn}}$.

Proof. We have $||t||^\beta_{\mathrm{jn}} =_{4.43} ||\pi(t)||^\beta_{\mathrm{jn}} = ||\pi(t')||^\beta_{\mathrm{jn}} =_{4.43} ||t'||^\beta_{\mathrm{jn}}$. □

Lemma 4.45. *The following equalities hold:*

$$
\begin{aligned}
||x||^\beta_{\mathrm{jn}} &= 0\\
||\lambda x.t||^\beta_{\mathrm{jn}} &= ||t||^\beta_{\mathrm{jn}}\\
||x(u, y.r)||^\beta_{\mathrm{jn}} &= ||u||^\beta_{\mathrm{jn}} + ||r||^\beta_{\mathrm{jn}}\\
||(\lambda x.t)(u, y.r)||^\beta_{\mathrm{jn}} &= \begin{cases} 1 + ||r\{y/t\{x/u\}\}||^\beta_{\mathrm{jn}} & \text{if } x \in \mathrm{fv}(t) \text{ and } y \in \mathrm{fv}(r)\\ 1 + ||r\{y/t\}||^\beta_{\mathrm{jn}} + ||u||^\beta_{\mathrm{jn}} & \text{if } x \notin \mathrm{fv}(t) \text{ and } y \in \mathrm{fv}(r)\\ 1 + ||r||^\beta_{\mathrm{jn}} + ||t||^\beta_{\mathrm{jn}} + ||u||^\beta_{\mathrm{jn}} & \text{if } y \notin \mathrm{fv}(r) \end{cases}\\
||t(u, x.r)(u', y.r')||^\beta_{\mathrm{jn}} &= ||t(u, x.r(u', y.r'))||^\beta_{\mathrm{jn}}
\end{aligned}
$$

Proof. The proof follows from the inductive definition ISN(jn) and lemma 4.44. □

Lemma 4.46. *If* $\Gamma \Vdash^k_{\cap_J} t : \sigma$, *then* $||t||^\beta_{\mathrm{jn}} \le k$.

Proof. We define $|x|_l = |\lambda x.t|_l = 0, |t(u, x.r)|_l = |t|_l + 1$. We proceed by induction on the pair $\langle k, |t|_l \rangle$ with respect to the lexicographic order and we reason by case analysis on t. The proofs for cases $t = x$, $t = \lambda x.u$, $t = x(u, y.r)$ and $t = (\lambda x.s)(u, y.r)$ are similar to the ones in lemma 4.31, only replacing $\|t\|_{\text{djn}}$ by $\|t\|^{\beta}_{\text{jn}}$. We only show here the most interesting case which is $t = s(u, x.r)(u', y.r')$.

Let $t' = s(u, x.r(u', y.r'))$. By lemma 4.39 there is a type derivation $\Delta \Vdash^{\cap J}_{k'} t' : \sigma$ with $k' \leq k$ and $\Delta \sqsubseteq \Gamma$. Since $\|t\|^{\beta}_{\text{jn}} =_{4.45} \|t'\|^{\beta}_{\text{jn}}$ and $|t|_l > |t'|_l$ we can use the *i.h.* and we get $\|t'\|^{\beta}_{\text{jn}} \leq k'$, and thus $\|t\|^{\beta}_{\text{jn}} \leq k' \leq k$. □

As a corollary we obtain:

Lemma 4.47 (Soundness for ΛJ). *If t is $\cap J$-typable, then $t \in \text{SN}(\text{jn})$.*

Proof. By lemma 4.46, the number of β-reduction steps in any jn-reduction sequence starting at t is finite. So in any infinite jn-reduction sequence starting at t, there is necessarily a term u from which there is an infinite amount of π-steps only. But this is impossible since π terminates, so we conclude by contradiction. □

4.5 Faithfulness of the Translation

The natural translation of generalized applications into ES [see Esp07] is not conservative with respect to strong normalization. This is also true for the natural translation to λ-terms given by Joachimski and Matthes [JM03]. Indeed, recall the example from section 1.2.2.2, given by $t = \delta(\delta, y.r)$ with $y \notin \text{fv}(r)$ and $\delta = \lambda x.x(x, z.z)$. The term t is a dβ-redex, whose contraction throws away the two copies of δ. The naive translation of t gives $r^{\star}[y/\delta^{\star}\delta^{\star}]$, which diverges in λES.

In this section we define an alternative encoding and prove it faithful: a term in $\mathtt{T}_J$ is djn-strongly normalizing *iff* its alternative encoding is strongly normalizing in the ES framework. In a later subsection, we use this connection with ES to establish the equivalence between strong normalization of djn and $\mathtt{T}_J[\beta, \text{p2}]$.

4.5.1 A New Translation

We relate λJ_n to the simple calculus with ES, borrowed from Accattoli [Acc12], defined in section 1.3. Let us consider the (naive) translation from $\mathtt{T}_J$ to $\mathtt{T}_{ES}$ (section 3.1). According to it, the notion of distance in λES corresponds to our notion of distance for λJ_n. For instance, the application $t(u, x.\_)$ in the term $t(u, x.\lambda y.r)(u', z.r')$ can be seen as a substitution $[x/t^{\star}u^{\star}]$ inserted between the abstraction $\lambda y.r$ and the argument u'. But how can we now (informally) relate π to the notions of existing permutations for λES? Using the previous translation, we

can see that $t_0 = t(u, x.r)(u', y.r') \mapsto_\pi t(u, x.r(u', y.r')) = t_1$ simulates as

$$t_0{}^\star = r'{}^\star[y/(r^\star[x/t^\star u^\star])u'{}^\star] \to r'{}^\star[y/(r^\star u'{}^\star)[x/t^\star u^\star]] \to r'{}^\star[y/r^\star u'{}^\star][x/t^\star u^\star] = t_1{}^\star$$

The first step is an instance of a rule in ES known as σ_1: $(t[x/u])v \mapsto (tv)[x/u]$, and the second one of a rule we call σ_4: $v[y/t[x/u]] \mapsto v[y/t][x/u]$. Quantitative types for ES tell us that only rule σ_1, but not rule σ_4, is valid for a call-by-name calculus. This is why it is not surprising that π is rejected by our type system, as detailed in section 4.4.3.

The alternative encoding we propose is as follows (noted $(\cdot)^\star$) instead of $(\cdot)^\star$):

Definition 4.48 (Translation from $\mathtt{T}_J$ to $\mathtt{T}_{ES}$).

$$x^\star \coloneqq x \qquad (\lambda x.t)^\star \coloneqq \lambda x.t^\star \qquad t(u, x.r)^\star \coloneqq r^\star\{x/x^{\mathrm{l}}x^{\mathrm{r}}\}[x^{\mathrm{r}}/u^\star][x^{\mathrm{l}}/t^\star]$$

Notice the above π-reduction $t_0 \to t_1$ is still simulated: $t_0^\star \to^2_{\sigma_4} t_1^\star$.

Consider again the counterexample $t = \delta(\delta, y.r)$ to faithfulness discussed above. The alternative encoding of t is $r^\star\{y/y^{\mathrm{l}}y^{\mathrm{r}}\}[y^{\mathrm{r}}/\delta^\star][y^{\mathrm{l}}/\delta^\star]$, which is just $r^\star[y^{\mathrm{r}}/\delta^\star][y^{\mathrm{l}}/\delta^\star]$, because $y \notin \mathrm{fv}(r^\star)$. The only hope to have an interaction between the two copies of $\delta^\star$ in the previous term is to execute the ES, but such executions will just throw away those two copies, because $y^{\mathrm{l}}, y^{\mathrm{r}} \notin \mathrm{fv}(r^\star)$. This hopefully gives an intuitive idea of the faithfulness of our encoding.

4.5.2 Proof of Faithfulness

We need to prove the equivalence between two notions of strong normalization: the one of a term in λJ_n and the one of its encoding in λES. While this proof can be a bit involved using traditional methods, quantitative types will make it very straightforward.

For λES, we will use the type system in section 1.3.2.3, for which we recall the characterization.

Theorem 4.49. *Let $M \in \mathtt{T}_{ES}$. Then M is typable in $\cap ES$ iff $M \in \mathrm{SN}(\mathrm{dB}, \mathrm{sub})$.*

A simple induction on the type derivation shows that the encoding is sound.

Lemma 4.50. *Let $t \in \mathtt{T}_J$. Then $\Gamma \Vdash_{\cap J} t : \sigma \implies \Gamma \Vdash_{\cap ES} t^\star : \sigma$.*

Proof. By induction on the type derivation. Notice that the statement also applies by straightforward *i.h.* for rule (MANY).

Case (VAR). Then $t = x$ and we type $t^\star = x$ with rule (VAR).

Case (ABS). Then $t = \lambda x.s$ and $t^\star = \lambda x.s^\star$. We conclude by *i.h.* using $(\to_i)$.

Case (APP). Then $t = s(u, x.r)$ and $t^\star = r^\star\{x/x^{\mathrm{l}}x^{\mathrm{r}}\}[x^{\mathrm{r}}/u^\star][x^{\mathrm{l}}/s^\star]$. By the *i.h.* we have derivations $\Pi \Vdash_{\cap ES} s^\star : \#([\mathscr{M}_i \to \tau_i]_{i\in I})$, $\Delta \Vdash_{\cap ES} u^\star : \#(\sqcup_{i\in I}\mathscr{M}_i)$ and $\Lambda; x : [\tau_i]_{i\in I} \Vdash_{\cap ES} r^\star : \sigma$ with $\Gamma = \Pi \uplus \Delta \uplus \Lambda$.

If $I \neq \emptyset$, it is easy to construct a derivation $x^{\mathrm{l}} : [\mathscr{M}_i \to \tau_i]_{i\in I}; x^{\mathrm{r}} : \sqcup_{i\in I}\mathscr{M}_i \Vdash_{\cap ES} x^{\mathrm{l}}x^{\mathrm{r}} : [\tau_i]_{i\in I}$. By lemma 4.24, we get $\Phi = \Lambda; x^{\mathrm{l}} : [\mathscr{M}_i \to \tau_i]_{i\in I}; x^{\mathrm{r}} : \sqcup_{i\in I}\mathscr{M}_i \Vdash_{\cap ES}$

$r^\star\{x/x^\mathrm{l}x^\mathrm{r}\} : \sigma$. We conclude by building the following derivation.

$$\dfrac{\dfrac{\Phi \qquad \Delta \vdash u^\star : \#(\sqcup_{i\in I}\mathscr{M}_i)}{\Lambda \uplus \Delta; x^\mathrm{l} \vdash [\mathscr{M}_i \longrightarrow \tau_i]_{i\in I} : r^\star\{x/x^\mathrm{l}x^\mathrm{r}\}[x^\mathrm{r}/u^\star]\sigma} \qquad \Pi \Vdash s^\star : [\mathscr{M}_i \longrightarrow \tau_i]_{i\in I}}{\Pi \uplus \Delta \uplus \Lambda \vdash r^\star\{x/x^\mathrm{l}x^\mathrm{r}\}[x^\mathrm{r}/u^\star][x^\mathrm{l}/s^\star] : \sigma}$$

If $I = \emptyset$, then $x \notin \mathrm{fv}(r) = \mathrm{fv}(r^\star)$ by relevance, so that $t^\star = r^\star[x^\mathrm{r}/u^\star][x^\mathrm{l}/s^\star]$. By the *i.h.* we have derivations $\Pi \Vdash_{\cap ES} s^\star : [\tau]$, $\Delta \Vdash_{\cap ES} u^\star : [\rho]$ and $\Lambda \Vdash_{\cap ES} r^\star : \sigma$ with $\Gamma = \Pi \uplus \Delta \uplus \Lambda$. We conclude by building the following derivation.

$$\dfrac{\dfrac{\Lambda; x^\mathrm{l} : [\,]; x^\mathrm{r} : [\,] \Vdash r^\star : \sigma \qquad \Delta \vdash u^\star : [\rho]}{\Lambda \uplus \Delta; x^\mathrm{l} : [\,] \vdash r^\star[x^\mathrm{r}/u^\star] : \sigma} \qquad \Pi \Vdash s^\star : [\tau]}{\Lambda \uplus \Pi \uplus \Delta \vdash r^\star\{x/x^\mathrm{l}x^\mathrm{r}\}[x^\mathrm{r}/u^\star][x^\mathrm{l}/s^\star] : \sigma}$$

□

We show completeness by a detour through the encoding of $\mathtt{T}_{ES}$ to $\mathtt{T}_J$ (definition 3.2). The two following lemmas, shown by induction on the type derivations, give in particular that $t^\star$ typable implies t typable.

Lemma 4.51. *Let* $M \in \mathtt{T}_{ES}$. *Then* $\Gamma \Vdash_{\cap ES} M : \sigma \implies \Gamma \Vdash_{\cap J} M^\circ : \sigma$.

Proof. By induction on the derivation. The cases where the derivation ends with (VAR), (ABS) or (MANY) (generalizing the statement) are straightforward.

Case (APP). Then $M = PN$ and $M^\circ = P^\circ(N^\circ, z.z)$. By the *i.h.* we have derivations $\Lambda \Vdash_{\cap J} P^\circ : \mathscr{M} \longrightarrow \sigma$ and $\Delta \Vdash_{\cap J} N^\circ : \#(\mathscr{M})$ with $\Gamma = \Lambda \uplus \Delta$. By application of rule (MANY) we obtain $\Lambda \Vdash_{\cap J} P^\circ : [\mathscr{M} \longrightarrow \sigma]$. We conclude by building the following derivation.

$$\dfrac{\Lambda \Vdash P^\circ : [\mathscr{M} \longrightarrow \sigma] \qquad \Delta \Vdash N^\circ : \#(\mathscr{M}) \qquad \dfrac{}{x : [\sigma] \vdash x : \sigma}}{\Lambda \uplus \Delta \vdash P^\circ(N^\circ, x.x) : \sigma}$$

Case (ES). Then $M = P[x/N]$ and we have a translation of the form $M^\circ = (\lambda z.z)(N^\circ, x.P^\circ)$. By the *i.h.* we have derivations $\Lambda; x : \mathscr{M} \Vdash_{\cap J} P^\circ : \sigma$ and $\Delta \Vdash_{\cap J} N^\circ : \#(\mathscr{M})$ with $\Gamma = \Lambda \uplus \Delta$. Let $\mathscr{M} = [\tau_i]_{i\in I}$.

If $I \neq \emptyset$, We conclude by building the following derivation.

$$\dfrac{\dfrac{\left(\dfrac{\dfrac{}{z : [\tau_i] \vdash z : \tau_i}}{\emptyset \vdash \lambda z.z : [\tau_i] \to \tau_i}\right)_{i \in I}}{\emptyset \vdash \lambda z.z : [[\tau_i] \to \tau_i]_{i \in I}} \qquad \Delta \Vdash N^\circ : \#(\mathscr{M}) \qquad \Lambda; x : \mathscr{M} \Vdash P^\circ : \sigma}{\Delta \uplus \Lambda \vdash (\lambda z.z)(N^\circ, x.P^\circ) : \sigma}$$

If $I = \emptyset$, We conclude by building the following derivation (where τ is arbitrary).

$$\dfrac{\dfrac{\dfrac{\dfrac{}{z : [\tau] \vdash z : \tau}}{\emptyset \vdash \lambda z.z : [\tau] \to \tau}}{\emptyset \vdash \lambda z.z : [[\tau] \to \tau]} \qquad \Delta \Vdash N^\circ : \#(\mathscr{M}) \qquad \Lambda; x : \mathscr{M} \Vdash P^\circ : \sigma}{\Delta \uplus \Lambda \vdash (\lambda z.z)(N^\circ, x.P^\circ) : \sigma}$$

□

Lemma 4.52. *Let $t \in \mathtt{T}_J$. Then $\Gamma \Vdash_{\cap J} t^{\star\circ} : \sigma \implies \Gamma \Vdash_{\cap J} t : \sigma$.*

Proof. By induction on t. The cases where $t = x$ or $t = \lambda x.s$ are straightforward by the *i.h.* We reason by cases for the generalized application.

Case $t = s(u, x.r)$ where $x \in \mathrm{fv}(r)$. We have

$$t^{\star\circ} = (r^\star\{x/x^{\mathrm{l}}x^{\mathrm{r}}\}[x^{\mathrm{r}}/u^\star][x^{\mathrm{l}}/s^\star])^\circ = \mathtt{I}(s^{\star\circ}, x^{\mathrm{l}}.\mathtt{I}(u^{\star\circ}, x^{\mathrm{r}}.r^{\star\circ}\{x/x^{\mathrm{l}}(x^{\mathrm{r}}, z.z)\}))$$

By construction and also by the anti-substitution lemma 4.34 it is not difficult to see that $\Gamma = \Gamma_s \uplus \Gamma_u \uplus \Gamma_r$ and there exist derivations having the following conclusions, where $I \neq \emptyset$:

1. $\Gamma_r; x : [\tau_i]_{i \in I} \Vdash_{\cap J} r^{\star\circ} : \sigma$
2. $x^{\mathrm{l}} : [[\tau_i] \to \tau_i]_{i \in I} \Vdash_{\cap J} x^{\mathrm{l}} : [[\tau_i] \to \tau_i]_{i \in I}$
3. $x^{\mathrm{r}} : [\tau_i]_{i \in I} \Vdash_{\cap J} x^{\mathrm{r}} : [\tau_i]_{i \in I}$
4. $\emptyset \Vdash_{\cap J} \mathtt{I} : [[\tau_i] \to \tau_i]_{i \in I}$
5. $\Gamma_u \Vdash_{\cap J} u^{\star\circ} : [\tau_i]_{i \in I}$
6. $\emptyset \Vdash_{\cap J} \mathtt{I} : [[[\tau_i] \to \tau_i] \to [\tau_i] \to \tau_i]_{i \in I}$
7. $\Gamma_s \Vdash_{\cap J} s^{\star\circ} : [[\tau_i] \to \tau_i]_{i \in I}$

The *i.h.* on points 1, 5 and 7 give $\Gamma_r; x : [\tau_i]_{i \in I} \Vdash_{\cap J} r : \sigma$, $\Gamma_u \Vdash_{\cap J} u : [\tau_i]_{i \in I}$ and $\Gamma_s \Vdash_{\cap J} s : [[\tau_i] \to \tau_i]_{i \in I}$ resp., so that we conclude with the following derivation:

$$\frac{\Gamma_s \Vdash s : [[\tau_i] \to \tau_i]_{i\in I} \qquad \Gamma_u \Vdash u : [\tau_i]_{i\in I} \qquad \Gamma_r; x : [\tau_i]_{i\in I} \Vdash r : \sigma}{\Gamma \vdash s(u, x.r) : \sigma}$$

Case $t = s(u, x.r)$ where $x \notin \mathrm{fv}(r)$. Then we have

$$t^{\star\circ} = (r^\star[x^{\mathrm{r}}/u^\star][x^{\mathrm{l}}/s^\star])^\circ = \mathtt{I}(s^{\star\circ}, x^{\mathrm{l}}.\mathtt{I}(u^{\star\circ}, x^{\mathrm{r}}.r^{\star\circ}))$$

We have the following derivation, where $\Gamma = \Gamma_s \uplus \Gamma_r \uplus \Gamma_r$, $[\tau_1] \to \tau_1, [\tau_2] \to \tau_2, \rho$ and ρ' are witness types.

$$\frac{\dfrac{\vdots}{\emptyset \vdash \mathtt{I} : [[\tau_1] \to \tau_1]} \qquad \Gamma_s \Vdash s^{\star\circ} : [\rho] \qquad \Phi}{\Gamma_s \uplus \Gamma_u \uplus \Gamma_r \vdash \mathtt{I}(s^{\star\circ}, x^{\mathrm{l}}.\mathtt{I}(u^{\star\circ}, x^{\mathrm{r}}.r^{\star\circ})) : \sigma}$$

Where

$$\Phi = \frac{\dfrac{\vdots}{\emptyset \vdash \mathtt{I} : [[\tau_2] \to \tau_2]} \qquad \Gamma_u \Vdash u^{\star\circ} : [\rho'] \qquad \Gamma_r \Vdash r^{\star\circ} : \sigma}{\Gamma_u \uplus \Gamma_r \vdash \mathtt{I}(u^{\star\circ}, x^{\mathrm{r}}.r^{\star\circ}) : \sigma}$$

By the *i.h.* we have derivations $\Gamma_r \Vdash_{\cap J} r : \sigma$, $\Gamma_s \Vdash_{\cap J} s : [\rho]$ and $\Gamma_u \Vdash_{\cap J} u : [\rho']$. We then derive $\Gamma \Vdash_{\cap J} s(u, x.r) : \sigma$ by rule (APP). □

Putting everything together, we get this equivalence:

Corollary 4.53. *Let* $t \in \mathtt{T}_J$. *Then* $\Gamma \Vdash_{\cap J} t : \sigma \iff \Gamma \Vdash_{\cap ES} t^\star : \sigma$.

This corollary, together with the two characterization theorems 4.37 and 4.49, provides the main result of this section:

Theorem 4.54 (Faithfulness). *Let* $t \in \mathtt{T}_J$. *Then* $t \in \mathrm{SN}(\mathrm{dB}) \iff t^\star \in \mathrm{SN}(\mathrm{dB}, \mathrm{sub})$.

4.6 Equivalent Notions of Strong Normalization

In the previous section, we related strong dβ-normalization with strong normalization of ES. In this section we compare the various concepts of strong normalization that are induced on $\mathtt{T}_J$ by β, dβ, $(\beta, \mathrm{p2})$ and jn. This comparison makes use of several results obtained in the previous sections. From it, we obtain new results about the original calculus ΛJ.

4.6.1 β-Normalization is not Enough

We have discussed the unblocking property of π and p2 in section 4.1. From the point of view of normalization, this means that $\mathtt{T}_J[\beta]$ has *premature* normal forms and that $\mathrm{SN}(\beta) \subsetneq$

SN(dβ). To illustrate this purpose we give an example of a $\mathtt{T}_J$-term which normalizes when only using rule β, but diverges when adding permutation rules or distance. Let us take $t \coloneqq w(u, w'.\delta^\circ)(\delta^\circ, x.x)$. Although this term is normal in $\mathtt{T}_J[\beta]$, the second δ° is actually an argument for the first one, as we can see with a π permutation:

$$t \to_\pi w(u, w'.\delta^\circ(\delta^\circ, x.x)) = w(u, w'.\Omega) \coloneqq t'$$

Thus $t \to_\pi t' \to_\beta t'$ which implies $t \notin \mathrm{SN}(\mathrm{jn})$. We can also unblock the redex in t by a p2-permutation moving the inner λx up:

$$t \to_{\mathrm{p2}} (\lambda y.w(u, w'.y(y, z.z)))(\delta^\circ, x.x) \to_\beta t'$$

Thus $t \to_{\mathrm{p2}} \to_\beta t' \to_\beta t'$ and thus $t \notin \mathrm{SN}(\beta, \mathrm{p2})$. We get the same thing in a unique dβ-step: $t \to_{\mathrm{d}\beta} t'$.

In all the three cases, β-strong normalization is not preserved by the permutation rules, as there is a term $t \in \mathrm{SN}(\beta)$ such that $t \notin \mathrm{SN}(\mathrm{jn})$, $t \notin \mathrm{SN}(\beta, \mathrm{p2})$ and $t \notin \mathrm{SN}(\mathrm{d}\beta)$.

4.6.2 Comparison with β + p2

We now formalize the fact that our calculus $\mathtt{T}_J[\mathrm{d}\beta]$ is a version with distance of $\mathtt{T}_J[\beta, \mathrm{p2}]$, so that they are equivalent from a normalization point of view. For this, we will establish the equivalence between strong normalization w.r.t. dβ and $(\beta, \mathrm{p2})$, through a long chain of equivalences. One of them is theorem 4.54, that we have proved in the previous section; the other is a result about σ-rules in the λ-calculus – which is why we have to go through the λ-calculus again.

Definition 4.55 (Translation $(\cdot)^\downarrow$ from $\mathtt{T}_{ES}$ to $\mathtt{T}_\Lambda$).

$$x^\downarrow \coloneqq x \quad (\lambda x.M)^\downarrow \coloneqq \lambda x.M^\downarrow \quad (MN)^\downarrow \coloneqq M^\downarrow N^\downarrow \quad MxN^\downarrow \coloneqq (\lambda x.M^\downarrow)N^\downarrow$$

Lemma 4.56. *Let $M \in \mathtt{T}_{ES}$. Then $M \in \mathrm{SN}(\mathrm{dB}, \mathrm{sub}) \implies M^\downarrow \in \mathrm{SN}(\beta)$.*

Proof. For typability in the λ-calculus, we use the type system $\mathcal{S}'_\lambda$ with choice operators of Kesner and Vial [KV20]. It can be seen as a restriction of our system $\cap ES$ to λ-terms. Suppose $M \in \mathrm{SN}(\mathrm{dB}, \mathrm{sub})$. By theorem 4.49 M is typable in $\cap ES$, and it is straightforward to show that $M^\downarrow$ is typable in $\mathcal{S}'_\lambda$. Moreover, $M^\downarrow$ typable implies that $M^\downarrow \in \mathrm{SN}(\beta)$ [KV20], which is what we want. □

For $t \in \mathtt{T}_J$, let $t^\square \coloneqq (t^\downarrow)^\star$. So, we are just composing the alternative encoding of generalized application into ES with the map into λ-calculus just introduced. The translation $(\cdot)^\square$ may be given directly by recursion as follows:

$$x^\square = x \qquad (\lambda x.t)^\square = \lambda x.t^\square \qquad t(u, y.r)^\square = (\lambda y^{\mathrm{r}}.(\lambda y^{\mathrm{l}}.r^\square\{y/y^{\mathrm{l}}y^{\mathrm{r}}\})t^\square)u^\square$$

Lemma 4.57. $t^\square \in \mathrm{SN}(\beta, \sigma_2) \implies t \in \mathrm{SN}(\beta, \mathrm{p2})$.

Proof. Because $(\cdot)^{\Box}$ produces a strict simulation from $\mathtt{T}_J$ to $\mathtt{T}_\Lambda$. More precisely: (i) if $t_1 \to_\beta t_2$ then $t_1^{\Box} \to_\beta^+ t_2^{\Box}$; (ii) if $t_1 \to_{\mathrm{p2}} t_2$ then $t_1^{\Box} \to_{\sigma_2}^2 t_2^{\Box}$. □

Theorem 4.58. *Let $t \in \mathtt{T}_J$. Then $t \in \mathrm{SN}(\beta, \mathrm{p2})$ iff $t \in \mathrm{SN}(\mathrm{d}\beta)$.*

Proof. We prove that the following conditions are equivalent: 1) $t \in \mathrm{SN}(\beta, \mathrm{p2})$. 2) $t \in \mathrm{SN}(\mathrm{d}\beta)$. 3) $t^\star \in \mathrm{SN}(\mathrm{dB}, \mathrm{sub})$. 4) $t^{\Box} \in \mathrm{SN}(\beta)$. 5) $t^{\Box} \in \mathrm{SN}(\beta, \sigma_2)$. Now, 1) $\implies$ 2) is because $\to_{\mathrm{d}\beta} \subset \to_{\beta,\mathrm{p2}}^+$. 2) $\implies$ 3) is by theorem 4.54. 3) $\implies$ 4) is by lemma 4.56. 4) $\implies$ 5) is showed by Regnier [Reg94]. 5) $\implies$ 1) is by lemma 4.57. □

4.6.3 Comparison with $\beta + \pi$

We now prove the equivalence between strong normalization for $\mathrm{d}\beta$ and for jn. One of the implications already follows from the properties of the typing system.

Lemma 4.59. *Let $t \in \mathtt{T}_J$. If $t \in \mathrm{SN}(\mathrm{d}\beta)$ then $t \in \mathrm{SN}(\mathrm{jn})$.*

Proof. Follows from the completeness of the typing system (lemma 4.36) and soundness of $\cap J$ for jn (lemma 4.47). □

The proof of the other implication requires more work, organized in 4 parts: 1) A remark about ES. 2) A remark about translations of ES into the ΛJ-calculus. 3) Two new properties of strong normalization for jn in ΛJ. 4) Preservation of strong jn-normalization by a certain map from the set $\mathtt{T}_J$ into itself.

The remark about explicit substitutions is this:

Lemma 4.60. *For all $M \in \mathtt{T}_{ES}$, $M \in \mathrm{SN}(\mathrm{dB}, \mathrm{sub})$ iff $M \in \mathrm{SN}(\mathrm{B}, \mathrm{sub})$.*

As in section 3.6, we do not use the original translation $(\cdot)^\circ$ from $\mathtt{T}_{ES}$ to $\mathtt{T}_J$, but rather the new one $(\cdot)^\bullet$, which allows simulation of dB and sub reductions. In that translation (defined in section 3.6), the clause for applications changes:

$$(MN)^\bullet := \mathtt{I}(N^\bullet, y.M^\bullet(y, z.z))$$

This strict simulation gives immediately:

Lemma 4.61. *For all $M \in \mathtt{T}_{ES}$, if $M^\bullet \in \mathrm{SN}(\beta)$ then $M \in \mathrm{SN}(\mathrm{B}, \mathrm{sub})$.*

We now prove two properties of strong normalization for jn in ΛJ. Following Matthes [Mat00], SN(jn) admits an inductive characterization ISN(jn), given in figure 4.1, which uses the following inductive generation for $\mathtt{T}_J$-terms:

$$t, u, r ::= x\vec{S} \mid \lambda x.t \mid (\lambda x.t)S\vec{S} \qquad S ::= (u, y.r)$$

$$\frac{}{x \in \mathrm{ISN}(\mathrm{jn})}\ (\textsc{var}) \qquad \frac{u, r \in \mathrm{ISN}(\mathrm{jn})}{x(u, z.r) \in \mathrm{ISN}(\mathrm{jn})}\ (\textsc{hvar}) \qquad \frac{t \in \mathrm{ISN}(\mathrm{jn})}{\lambda x.t \in \mathrm{ISN}(\mathrm{jn})}\ (\textsc{lambda})$$

$$\frac{x(u, y.rS)\vec{S} \in \mathrm{ISN}(\mathrm{jn})}{x(u, y.r)S\vec{S} \in \mathrm{ISN}(\mathrm{jn})}\ (\textsc{pi}) \qquad \frac{r\{y/t\{x/u\}\}\vec{S} \in \mathrm{ISN}(\mathrm{jn}) \quad t, u \in \mathrm{ISN}(\mathrm{jn})}{(\lambda x.t)(u, y.r)\vec{S} \in \mathrm{ISN}(\mathrm{jn})}\ (\textsc{beta})$$

Figure 4.1: Inductive characterization of the strong jn-normalizing ΛJ-terms.

Hence S stands for a *generalized* argument, while $\vec{S}$ denotes a possibly empty list of S's. Notice that at most one rule applies to a given term, so the rules are deterministic (and thus invertible).

A preliminary fact is the following:

Lemma 4.62. *The set* $\mathrm{SN}(\mathrm{jn})$ *is closed under prefixing of arbitrary* π*-reduction steps:*

$$\frac{t \longrightarrow_\pi t' \text{ and } t' \in \mathrm{SN}(\mathrm{jn})}{t \in \mathrm{SN}(\mathrm{jn})}$$

Proof. We first consider the following three facts:

1. Every $t \in \mathrm{T}_J$ has a unique π-normal form $\pi(t)$.
2. The map $\pi(\cdot)$ preserves β-reduction steps, that is, $t_1 \longrightarrow_\beta t_2$ implies $\pi(t_1) \longrightarrow_\beta \pi(t_2)$ (lemma 4.41).
3. $\longrightarrow_\pi$ is terminating.

Now, suppose $t \notin \mathrm{SN}(\mathrm{jn})$, so that there is an infinite (jn)-reduction sequence starting at t. Then by the previous facts it is possible to construct an infinite β-reduction sequence starting at $\pi(t)$. But $\pi(t) = \pi(t')$ and $t' \longrightarrow_\pi^* \pi(t')$, so there is an infinite $\beta\pi$-reduction sequence starting at t', which leads to a contradiction. □

Given that $\mathrm{SN}(\mathrm{jn}) = \mathrm{ISN}(\mathrm{jn})$, the "rule" in lemma 4.62, when written with $\mathrm{ISN}(\mathrm{jn})$, is admissible for the predicate $\mathrm{ISN}(\mathrm{jn})$. Now, consider:

$$\frac{u, r \in \mathrm{ISN}(\mathrm{jn})}{r\{x/y(u, z.z)\} \in \mathrm{ISN}(\mathrm{jn})}\ (\mathrm{I})$$

$$\frac{r\{x/r\{z/t\{y/u\}\}\} \in \mathrm{ISN}(\mathrm{jn}) \qquad t, u \in \mathrm{ISN}(\mathrm{jn}) \qquad x \notin \mathrm{fv}(t, u, r)}{r\{x/(\lambda y.t)(u, z.r)\} \in \mathrm{ISN}(\mathrm{jn})}\ (\mathrm{II})$$

Notice rule (II) generalizes rule (BETA): just take $r = x\vec{S}$, with $x \notin \vec{S}$.

The two new properties of strong normalization for jn in ΛJ are contained in the following lemma.

Lemma 4.63. *Rules (I) and (II) are admissible rules for the predicate* ISN(jn).

Proof. Proof of (I). By induction on $t \in \mathrm{ISN(jn)}$, we prove that $t\{x/y(u, z.z)\} \in \mathrm{ISN(jn)}$. The most interesting case is (PI), which we spell out in detail. We will use a device to shorten the writing: if E is t, or S, or $\vec{S}$, then $\underline{E}$ denotes $E\{x/y(u, z.z)\}$. Suppose $t = y'(u', z'.t')S\vec{S} \in \mathrm{ISN(jn)}$ with $y'(u', z'.t'S)\vec{S} \in \mathrm{ISN(jn)}$. We want $\underline{t} \in \mathrm{ISN(jn)}$. If $y' \neq y$, then the thesis follows by the *i.h.* and one application of (PI). Otherwise, $\underline{t} = y(u, z.z)(\underline{u'}, z'.\underline{t'})\underline{S}\underline{\vec{S}}$. By the *i.h.*,

$$y(u, z.z)(\underline{u'}, z'.\underline{t'S})\underline{\vec{S}} \in \mathrm{ISN(jn)}.$$

By inversion of (PI), we get

$$y(u, z.z(\underline{u'}, z'.\underline{t'S}))\underline{\vec{S}} \in \mathrm{ISN(jn)}.$$

From this, lemma 4.62 gives

$$y(u, z.z(\underline{u'}, z'.\underline{t'})\underline{S})\underline{\vec{S}} \in \mathrm{ISN(jn)}.$$

Finally, two applications of (PI) yield $\underline{t} \in \mathrm{ISN(jn)}$.

Proof of (II). We prove the following: for all $t_1 \in \mathrm{ISN(jn)}$, for all $n \geq 0$, if t_1 has n occurrences of the sub-term $r\{z/t\{y/u\}\}$, then, for any choice of n such occurrences, $t_2 \in \mathrm{ISN(jn)}$, where t_2 is the term that results from t_1 by replacing each of those n occurrences by $(\lambda y.t)(u, z.r)$.

Notice the statement we are going to prove entails the admissibility of (II). Indeed, given s, let n be the number of free occurrences of x in s. The term $t_1 = s\{x/r\{z/t\{y/u\}\}\}$ has well determined n occurrences of the sub-term $r\{z/t\{y/u\}\}$ (it may have others), and $s\{x/(\lambda y.t)(u, z.r)\}$ is the term that results from t_1 by replacing each of those n occurrences by $(\lambda y.t)(u, z.r)$.

Suppose $t_1 \in \mathrm{ISN(jn)}$ and consider n occurrences of the sub-term $r\{z/t\{y/u\}\}$ in t_1. The proof is by induction on $t_1 \in \mathrm{ISN(jn)}$ and sub-induction on n. A term s is determined, with n free occurrences of x, such that $x \notin t, u, r$ and $t_1 = s\{x/r\{z/t\{y/u\}\}\}$. We want to prove that $s\{x/(\lambda y.t)(u, z.r)\} \in \mathrm{ISN(jn)}$. We will use a device to shorten the writing: if E is t, or S, or $\vec{S}$, then $\underline{E}$ denotes $E\{x/r\{z/t\{y/u\}\}\}$ and $\underline{\underline{E}}$ denotes $E\{x/(\lambda y.t)(u, z.r)\}$. The proof proceeds by case analysis on s.

We show the critical case $s = x\vec{S}$, where use is made of the sub-induction hypothesis. We are given $\underline{s} = r\{z/t\{y/u\}\}\underline{\vec{S}} \in \mathrm{ISN(jn)}$. We want to show $\underline{\underline{s}} = (\lambda y.t)(u, z.r)\underline{\underline{\vec{S}}} \in \mathrm{ISN(jn)}$.

Given that $t, u \in \mathrm{ISN}(\mathrm{jn})$, it suffices

$$r\{z/t\{y/u\}\}\underline{\vec{S}} \in \mathrm{ISN}(\mathrm{jn}) \tag{4.3}$$

due to invertibility of (BETA). Let $s' \coloneqq r\{z/t\{y/u\}\}\vec{S}$. Since $x \notin t, u, r$, we have $\underline{s'} = \underline{s}$ (whence $\underline{s'} \in \mathrm{ISN}(\mathrm{jn})$), and the number of free occurrences of x in s' is $n - 1$. By sub-induction hypothesis, $\underline{s'} \in \mathrm{ISN}(\mathrm{jn})$. But $\underline{s'} = r\{z/t\{y/u\}\}\underline{\vec{S}}$, again due to $x \notin t, u, r$. Therefore (4.3) holds. □

We now move to the fourth part of the ongoing reasoning. Consider the map from $\mathtt{T}_J$ to itself obtained by composing $(\cdot)^\star : \mathtt{T}_J \longrightarrow \mathtt{T}_{ES}$ with $(\cdot)^\bullet : \mathtt{T}_{ES} \longrightarrow \mathtt{T}_J$. Let us write $(\cdot)^\dagger$ this composition. A recursive definition is also possible, as follows:

$$x^\dagger = x \qquad \lambda x.t^\dagger = \lambda x.t^\dagger \qquad t(u, y.r)^\dagger = \mathtt{I}(t^\dagger, y_1.\mathtt{I}(u^\dagger, y_2.r^\dagger\{y/y_1(y_2, z.z)\}))$$

Lemma 4.64. *If* $t \in \mathrm{SN}(\mathrm{jn})$ *then* $t^\dagger \in \mathrm{SN}(\mathrm{jn})$.

Proof. For $t \in \mathrm{SN}(\mathrm{jn})$, $\|t\|_{\mathrm{jn}}$ denotes the length of the longest jn-reduction sequence starting at t. We prove $t^\dagger \in \mathrm{ISN}(\mathrm{jn})$ by induction on the longest jn reduction sequence starting at t ($\|t\|_{\mathrm{jn}}$), with sub-induction on the size of t. We proceed by case analysis of t.

Case $t = x$. We have $x^\dagger = x \in \mathrm{ISN}(\mathrm{jn})$.

Case $t = \lambda x.s$. We have $t^\dagger = \lambda x.s^\dagger$. The sub-inductive hypothesis gives $s^\dagger \in \mathrm{ISN}(\beta\pi)$. By rule (LAMBDA), $\lambda x.s^\dagger \in \mathrm{ISN}(\mathrm{jn})$.

Case $t = y(u, x.r)$. We have $t^\dagger = \mathtt{I}(y, x_1.\mathtt{I}(u^\dagger, x_2.r^\dagger\{x/x_1(x_2, z.z)\}))$. By the (sub)-*i.h.*, $u^\dagger, r^\dagger \in \mathrm{ISN}(\mathrm{jn})$. Rule (I) yields $r^\dagger\{x/y(u^\dagger, z.z)\} \in \mathrm{ISN}(\mathrm{jn})$. Applying rule (BETA) twice, we obtain $t^\dagger \in \mathrm{ISN}(\mathrm{jn})$.

Case $t = (\lambda y.s)(u, x.r)$. We have $t^\dagger = \mathtt{I}(\lambda y.s^\dagger, x_1.\mathtt{I}(u^\dagger, x_2.r^\dagger\{x/x_1(x_2, z.z)\}))$. Notice that $\|t\|_{\mathrm{jn}}$ is greater than $\|s\|_{\mathrm{jn}}$ and $\|u\|_{\mathrm{jn}}$. By the induction hypothesis, $s^\dagger, u^\dagger \in \mathrm{ISN}(\mathrm{jn})$. Also $\|t\|_{\mathrm{jn}} > \|r\{x/s\{y/u\}\}\|_{\mathrm{jn}}$. Hence $(r\{x/s\{y/u\}\})^\dagger \in \mathrm{ISN}(\mathrm{jn})$, again by the *i.h.* Since map $(\cdot)^\dagger$ commutes with substitution, $r^\dagger\{x/s^\dagger\{y/u^\dagger\}\} \in \mathrm{ISN}(\mathrm{jn})$. This, together with $s^\dagger, u^\dagger \in \mathrm{ISN}(\mathrm{jn})$, gives $r^\dagger\{x/(\lambda y.s^\dagger)(u^\dagger, z.z)\} \in \mathrm{ISN}(\mathrm{jn})$, due to rule (II). Applying rule (BETA) twice, we obtain $t^\dagger \in \mathrm{ISN}(\mathrm{jn})$.

Case $t = t_0(u_1, x.r_1)(u_2, y.r_2)$. Let $s \coloneqq t_0(u_1, x_1.r_1(u_2, y.r_2))$. Since $t \longrightarrow_\pi s$, the *i.h.* gives $s^\dagger \in \mathrm{ISN}(\mathrm{jn})$. The induction hypothesis also gives $t_0^\dagger, u_1^\dagger \in \mathrm{ISN}(\mathrm{jn})$. The term $s^\dagger$ is

$$\mathtt{I}(t_0^\dagger, x_1.\mathtt{I}(u_1^\dagger, x_2.\mathtt{I}(r_1^\dagger, y_1.\mathtt{I}(u_2^\dagger, y_2.r_2^\dagger\{y/y_1(y_2, z.z)\}))\{x/x_1(x_2, z.z)\}))$$

From $s^\dagger \in \mathrm{ISN}(\mathrm{jn})$, by four applications of (BETA) we obtain

$$r_2^\dagger\{y/(r_1^\dagger\{x/t_0^\dagger(u_1^\dagger, z.z)\})(u_2^\dagger, z.z)\} \in \mathrm{ISN}(\mathrm{jn}) \tag{4.4}$$

We want $t^\dagger \in \mathrm{ISN}(\mathrm{jn})$, where $t^\dagger$ is

$$\mathtt{I}(\mathtt{I}(t_0^\dagger, x_1.\mathtt{I}(u_1^\dagger, x_2.r_1^\dagger\{x/x_1(x_2, z.z)\})), y_1.\mathtt{I}(u_2^\dagger, y_2.r_2^\dagger\{y/y_1(y_2, z.z)\}))$$

From (4.4) and $u_1^\dagger \in \mathrm{ISN}(\mathrm{jn})$, rule (II) obtains

$$r_2^\dagger\{y/\mathtt{I}(u_1^\dagger, x_2.r_1^\dagger\{x/t_0^\dagger(x_2, z.z)\}(u_2^\dagger, z.z))\} \in \mathrm{ISN}(\mathrm{jn})$$

From this, lemma 4.62 (prefixing of π-reduction steps) obtains

$$r_2^\dagger\{y/\mathtt{I}(u_1^\dagger, x_2.r_1^\dagger\{x/t_0^\dagger(x_2, z.z)\})(u_2^\dagger, z.z)\} \in \mathrm{ISN}(\mathrm{jn})$$

From this and $t_0^\dagger \in \mathrm{ISN}(\mathrm{jn})$, rule (II) obtains

$$r_2^\dagger\{y/\mathtt{I}(t_0^\dagger, x_1.\mathtt{I}(u_1^\dagger, x_2.r_1^\dagger\{x/x_1(x_2, z.z)\})(u_2^\dagger, z.z))\} \in \mathrm{ISN}(\mathrm{jn})$$

From this, lemma 4.62 (prefixing of π-reduction steps) obtains

$$r_2^\dagger\{y/\mathtt{I}(t_0^\dagger, x_1.\mathtt{I}(u_1^\dagger, x_2.\{^\dagger x/x_1(x_2, z.z)\}r_1))(u_2^\dagger, z.z)\} \in \mathrm{ISN}(\mathrm{jn})$$

Finally, two applications of (BETA) obtain $t^\dagger \in \mathrm{ISN}(\mathrm{jn}) = \mathrm{SN}(\mathrm{jn})$. □

All is in place to obtain the desired result:

Theorem 4.65. *Let $t \in \mathtt{T}_J$. $t \in \mathrm{SN}(\mathrm{d}\beta)$ iff $t \in \mathrm{SN}(\mathrm{jn})$.*

Proof. The implication from left to right is lemma 4.59. For the converse, suppose $t \in \mathrm{SN}(\mathrm{jn})$. By lemma 4.64, $t^\dagger \in \mathrm{SN}(\mathrm{jn})$. Trivially, $t^\dagger \in \mathrm{SN}(\beta)$. Since $t^\dagger = (t^\star)^\bullet$, lemma 4.61 gives $t^\star \in \mathrm{SN}(\mathrm{B}, \mathrm{sub})$. By lemma 4.60, $t^\star \in \mathrm{SN}(\mathrm{dB}, \mathrm{sub})$. By an equivalence in the proof of theorem 4.58, $t \in \mathrm{SN}(\mathrm{d}\beta)$. □

4.6.4 Consequences for ΛJ

The comparison with λJ_n gives new results about the original ΛJ (a quantitative typing system characterizing strong normalization, and a faithful translation into ES) as immediate consequences of theorems 4.37, 4.54 and 4.65.

Theorem 4.66. *Let $t \in \mathtt{T}_J$.*

Characterization *$t \in \mathrm{SN}(\mathrm{jn})$ iff t is $\cap J$-typable.*

Faithfulness $t \in \mathrm{SN}(\mathrm{jn})$ *iff* $t^\star \in \mathrm{SN}(\mathrm{dB}, \mathrm{sub})$.

Beyond strong normalization, ΛJ gains a new normalizing strategy, which reuses the notion of left-right normal form introduced in section 4.3.2. We take the definitions of neutral terms, answer and left-right context $\mathtt{R}$ given there for λJ_n, in order to define a new left-right strategy and a new predicate ISNj for ΛJ. The strategy is defined as the closure under $\mathtt{R}$ of rule β and of the particular case of rule π where the redex has the form $\mathtt{n}(u, x.\mathtt{a})S$.[2]

Definition 4.67. Predicate ISNj is defined by the rules (SNVAR), (SNAPP), (SNABS) in definition 4.17, together with the following two rules (which replace rule (SNBETA)):

$$\frac{\mathtt{R}\langle \mathtt{n}(u, y.\mathtt{a}S)\rangle \in \mathrm{ISNj}}{\mathtt{R}\langle \mathtt{n}(u, y.\mathtt{a})S\rangle \in \mathrm{ISNj}}\ (\textsc{snredex1}) \qquad \frac{\mathtt{R}\langle r\{y/t\{x/u\}\}\rangle, t, u \in \mathrm{ISNj}}{\mathtt{R}\langle(\lambda x.t)(u, y.r)\rangle \in \mathrm{ISNj}}\ (\textsc{snredex2})$$

The corresponding normalization strategy is organized as usual: an initial phase obtains a left-right normal form, whose components are then reduced by internal reduction. Is this new strategy any good? Theorem 4.70 answers positively with the equivalence between ISNj and ISN(jn). Before proving it, we need a few intermediate lemmas.

Lemma 4.68. *The following rule is admissible for the predicate* ISNj*:*

$$\frac{u, r \in \mathrm{ISNj}}{x(u, y.r) \in \mathrm{ISNj}}$$

Proof. The proof is by induction on $r \in \mathrm{ISNj}$. If r is generated by rules (SNVAR), (SNAPP) or (SNABS), then r is a weak-head normal form and rule (SNAPP) applies. Otherwise $r = \mathtt{R}\langle \mathit{redex}\rangle$. By inversion of rules (SNREDEX1) and (SNREDEX2), one obtains $\mathtt{R}\langle \mathit{contractum}\rangle \in \mathrm{ISNj}$, plus two other subterms of the redex also in ISNj in case of (SNREDEX1). Let $\mathtt{R}' := x(u, y.\mathtt{R})$. By the *i.h.* $\mathtt{R}'\langle \mathit{contractum}\rangle \in \mathrm{ISNj}$. By one of the rules (SNREDEX1)/(SNREDEX2), $\mathtt{R}'\langle \mathit{redex}\rangle \in \mathrm{ISNj}$, that is $x(u, y.r) \in \mathrm{ISNj}$. □

Lemma 4.69. *The following rule is admissible for the predicate* ISNj*:*

$$\frac{\mathtt{n}(u, y.sS)\vec{S} \in \mathrm{ISNj}}{\mathtt{n}(u, y.s)S\vec{S} \in \mathrm{ISNj}}$$

Proof. We prove by induction on $r \in \mathrm{ISNj}$, that, if $r = \mathtt{n}(u, y.sS)\vec{S}$, then $\mathtt{n}(u, y.s)S\vec{S} \in \mathrm{ISNj}$. We do case analysis of s.

Case $s = \mathtt{a}$. Follows by rule (SNREDEX1) by taking $\mathtt{R} = \Diamond\vec{S}$.

Case $s = \mathtt{R}\langle \mathit{redex}\rangle$. Let $\mathtt{R}_1 := \mathtt{n}(u, y.\mathtt{R}S)\vec{S}$ and $\mathtt{R}_2 := \mathtt{n}(u, y.\mathtt{R})S\vec{S}$. Since $r = \mathtt{R}_1\langle \mathit{redex}\rangle$, inversion of rule (SNREDEX1)/(SNREDEX2) gives $\mathtt{R}_1\langle \mathit{contractum}\rangle \in \mathrm{ISNj}$, plus two other sub-

[2] Notice how a redex has the two possible forms $(\lambda x.t)S$ or $\mathtt{n}(u, x.\mathtt{a})S$, that can be written as $\mathtt{a}S$, that is, the form $\mathtt{D}_\mathtt{n}\langle\lambda x.t\rangle S$ of a left-right redex in λJ_n.

terms of the redex also in ISNj in case of (SNREDEX2). By *i.h.* $\mathtt{R}_2\langle \mathit{contractum}\rangle \in$ ISNj. A final application of (SNREDEX1)/(SNREDEX2) gives $\mathtt{R}_2\langle \mathit{redex}\rangle \in$ ISNj, as required.

Case $s = \mathtt{n}'$. First, notice there are exactly four sub-cases:

Subcase $\mathtt{n}'S$ is a weak-head normal form and $\vec{S}$ is empty. By inversion of (SNAPP), we take sS apart, obtain its components in ISNj and, using (SNAPP), we reconstruct the term $\mathtt{n}(u, y.\mathtt{n}')S$ in ISNj.

Subcase S has the form $(u', y'.\mathtt{R}\langle \mathit{redex}\rangle)$ and $\vec{S}$ is arbitrary. By inversion of the rule (SNREDEX1)/(SNREDEX2), we have $\mathtt{n}(u, y.\mathtt{n}'(u', y'.\mathtt{R}\langle \mathit{contractum}\rangle))\vec{S} \in$ ISNj, plus two other subterms of the redex also in ISNj in case of (SNREDEX2). By the *i.h.*, we have that $\mathtt{n}(u, y.\mathtt{n}')(u', y'.\mathtt{R}\langle \mathit{contractum}\rangle)\vec{S} \in$ ISNj. As required, we obtain $\mathtt{n}(u, y.\mathtt{n}')(u', y'.\mathtt{R}\langle \mathit{redex}\rangle)\vec{S} \in$ ISNj by rule (SNREDEX1)/(SNREDEX2).

Subcase S has the form $(u', y'.\mathtt{a})$ and $\vec{S}$ is non-empty. Let $\vec{S} = R\vec{R}$. By applying inversion of (SNREDEX1) twice, we obtain $\mathtt{n}(u, y.\mathtt{n}'(u', y'.\mathtt{a}R))\vec{R} \in$ ISNj. By the *i.h.*, $\mathtt{n}(u, y.\mathtt{n}')(u', y'.\mathtt{a}R)\vec{R} \in$ ISNj. By (SNREDEX1), $\mathtt{n}(u, y.\mathtt{n}')(u', y'.\mathtt{a})R\vec{R} \in$ ISNj, as required.

Subcase S has the form $(u', y'.\mathtt{n}'')$ and $\vec{S}$ is non-empty. We have to analyze $\vec{S}$. For that, we introduce some notation. R^{nl} (respectively R^{ans}, R^{whnf}, R^{rdx}) will denote a generalized argument of the form $(t, z.\mathtt{n})$ (resp. $(t, z.\mathtt{a})$, $(t, z.w)hnf$, $(t, z.\mathtt{R}\langle \mathit{redex}\rangle)$).

Let $\mathtt{n}_0 = \mathtt{n}(u, y.\mathtt{n}'(u', y'.\mathtt{n}''))$ and $\mathtt{n}_1 = \mathtt{n}(u, y.\mathtt{n}')(u', y'.\mathtt{n}'')$. The non-empty $\vec{S}$ has exactly 3 possible forms (in all cases $m \geq 0$).

Subsubcase $R_1^{nl} \cdots R_m^{nl} R_{m+1}^{whnf}$. We apply the same kind of reasoning as in subcase 1.

Subsubcase $R_1^{nl} \cdots R_m^{nl} R^{rdx} \vec{R}$. Let $R^{rdx} = (u'', y''.\mathtt{R}''\langle \mathit{redex}\rangle)$ and let

$$\begin{array}{rcl} \mathtt{R}_0 & = & \mathtt{n}_0 R_1^{nl} \cdots R_m^{nl}(u'', y''.\mathtt{R}'')\vec{R} \\ \mathtt{R}_1 & = & \mathtt{n}_1 R_1^{nl} \cdots R_m^{nl}(u'', y''.\mathtt{R}'')\vec{R} \end{array}$$

Inversion of rule (SNREDEX1)/(SNREDEX2) gives $\mathtt{R}_0\langle \mathit{contractum}\rangle \in$ ISNj, plus two other subterms of the redex also in ISNj in case of (SNREDEX2). By the *i.h.*, we have that $\mathtt{R}_1\langle \mathit{contractum}\rangle \in$ ISNj. We obtain $\mathtt{R}_1\langle \mathit{redex}\rangle \in$ ISNj by rule (SNREDEX1)/(SNREDEX2), as required.

Subsubcase $R_1^{nl} \cdots R_m^{nl} R_{m+1}^{ans} R_{m+2} \vec{R}$. Let $R_{m+1}^{ans} = (u'', y''.\mathtt{a})$ and let

$$\begin{array}{rcl} \mathtt{n}_2 & = & \mathtt{n}_0 R_1^{nl} \cdots R_m^{nl} \\ \mathtt{n}_3 & = & \mathtt{n}_1 R_1^{nl} \cdots R_m^{nl} \end{array}$$

By inversion of (SNREDEX1), we obtain $\mathtt{n}_2(u'', y''.\mathtt{a}R_{m+2})\vec{R} \in \text{ISNj}$. Next *i.h.* gives $\mathtt{n}_3(u'', y''.\mathtt{a}R_{m+2})\vec{R} \in \text{ISNj}$. By (SNREDEX1), $\mathtt{n}_3(u'', y''.\mathtt{a})R_{m+2}\vec{R} \in$ ISNj as required. □

Theorem 4.70. *Let* $t \in \mathtt{T}_J$. $t \in \text{ISNj}$ *iff* $t \in \text{ISN(jn)}$.

Proof. $\Longrightarrow$) We show that each rule defining ISNj is admissible for the predicate ISN(jn) defined in figure 4.1. Cases (SNVAR) and (SNABS) are straightforward. Case (SNREDEX1) is by the *i.h.* and lemma 4.62. Case (SNREDEX2) is by the *i.h.* and rule (II). Case (SNAPP) is proved by a straightforward induction on n.

$\Longleftarrow$) We show that each rule in figure 4.1 defining the predicate ISN(jn) is admissible for the predicate ISNj. Cases (VAR) and (LAMBDA) are straightforward. Case (BETA) is by rule (SNREDEX2) and the *i.h.*, by just taking $\mathtt{R} = \Diamond\vec{S}$. Case (HVAR) follows by lemma 4.68 and the *i.h.* Case (PI) is by lemma 4.69 and the *i.h.* □

4.6.5 Alternative Proof of Equivalence

The last theorem can also be shown as a corollary of ISNj = SN(jn) and the fact that SN(jn) = ISN(jn) proved by Joachimski and Matthes [JM03]. We will show the first equality ISNj = SN(jn) in a similar way as for dβ (theorem 4.21).

Lemma 4.71. *If* $t_0 \longrightarrow_{\text{jn}} t_1$, *then*

(i) $t_0\{x/u\} \longrightarrow_{\text{jn}} t_1\{x/u\}$, *and*

(ii) $u\{x/t_0\} \longrightarrow^*_{\text{jn}} u\{x/t_1\}$.

Proof. The first statement is proved by induction on $t_0 \longrightarrow_{\text{jn}} t_1$ using lemma 4.6. The second is proved by induction on u. □

Lemma 4.72. *The strategy introduced in section 4.6.4 is deterministic.*

Proof. For every term there is a unique decomposition in terms of a R context and a redex. Besides that, β and π redexes do not overlap. □

Lemma 4.73. *If* $t_0 = \mathtt{R}\langle r\{y/t\{x/u\}\}\rangle \in \text{SN(jn)}$ *and* $t, u \in \text{SN(jn)}$, *then* $t_0' = \mathtt{R}\langle(\lambda x.t)(u, y.r)\rangle \in$ SN(jn).

Proof. By hypothesis we also have $r \in \text{SN(jn)}$. We use the lexicographic order to reason by induction on $\langle ||t_0||_{\text{jn}}, ||t||_{\text{jn}}, ||u||_{\text{jn}}, \mathtt{R}\rangle$. To show $t_0' \in \text{SN(jn)}$ it is sufficient to show that all its reducts are in SN(jn). We analyze all possible cases.

Case $t_0' \longrightarrow_\beta t_0$. We conclude by the hypothesis.

Case $t_0' \to_{\mathrm{jn}} \mathtt{R}\langle(\lambda x.t')(u, y.r)\rangle = t_1'$, where $t \to_{\mathrm{jn}} t'$. We have $t', u \in \mathrm{SN(jn)}$ and by lemma 4.71(ii) $t_0 = \mathtt{R}\langle r\{y/t\{x/u\}\}\rangle \to_{\mathrm{jn}}^* \mathtt{R}\langle r\{y/t'\{x/u\}\}\rangle = t_1$, so that also $t_1 \in \mathrm{SN(jn)}$. We conclude $t_1' \in \mathrm{SN(jn)}$ by the *i.h.* since $\|t_1\|_{\mathrm{jn}} \leq \|t_0\|_{\mathrm{jn}}$ and $\|t'\|_{\mathrm{jn}} < \|t\|_{\mathrm{jn}}$.

Case $t_0' \to_{\mathrm{jn}} \mathtt{R}\langle(\lambda x.t)(u', y.r)\rangle = t_1'$, where $u \to_{\mathrm{jn}} u'$. We have $t, u' \in \mathrm{SN(jn)}$ and by lemma 4.71(ii) $t_0 = \mathtt{R}\langle r\{y/t\{x/u\}\}\rangle \to_{\mathrm{jn}}^* \mathtt{R}\langle r\{y/t\{x/u'\}\}\rangle = t_1$, so that also $t_1 \in \mathrm{SN(jn)}$. We conclude $t_1' \in \mathrm{SN(jn)}$ by the *i.h.* since $\|t_1\|_{\mathrm{jn}} \leq \|t_0\|_{\mathrm{jn}}$ and $\|u'\|_{\mathrm{jn}} < \|u\|_{\mathrm{jn}}$.

Case $t_0' \to_{\mathrm{jn}} \mathtt{R}\langle(\lambda x.t)(u, y.r')\rangle = t_1'$, where $r \to_{\mathrm{jn}} r'$. We have $t, u \in \mathrm{SN(jn)}$ and by lemma 4.71(i) $t_0 = \mathtt{R}\langle r\{y/t\{x/u\}\}\rangle \to_{\mathrm{jn}} \mathtt{R}\langle r'\{y/t\{x/u\}\}\rangle = t_1$, so that also $t_1 \in \mathrm{SN(jn)}$. We conclude $t_1' \in \mathrm{SN(jn)}$ by the *i.h.* since $\|t_1\|_{\mathrm{jn}} < \|t_0\|_{\mathrm{jn}}$.

Case $t_0' \to_{\mathrm{jn}} \mathtt{R}'\langle(\lambda x.t)(u, y.r)\rangle = t_1'$, where $\mathtt{R} \to_{\mathrm{jn}} \mathtt{R}'$. Thus we also have that $t_0 = \mathtt{R}\langle r\{y/t\{x/u\}\}\rangle \to_{\mathrm{jn}} \mathtt{R}'\langle r\{y/t\{x/u\}\}\rangle = t_1$. We have $t_1, t, u \in \mathrm{SN(jn)}$. We conclude that $t_1' \in \mathrm{SN(jn)}$ by the *i.h.* since $\|t_1\|_{\mathrm{jn}} < \|t_0\|_{\mathrm{jn}}$.

Case $\mathtt{R} = \mathtt{R}'\langle\Diamond S\rangle$ and $t_0' = \mathtt{R}'\langle(\lambda x.t)(u, y.r)S\rangle \to_\pi \mathtt{R}'\langle(\lambda x.t)(u, y.rS)\rangle = t_1'$. This is the only case left. We have $t_0 = \mathtt{R}'\langle r\{y/t\{x/u\}\}S\rangle = \mathtt{R}'\langle(rS)\{y/t\{x/u\}\}\rangle = t_1$. We also have $t, u \in \mathrm{SN(jn)}$. We conclude $t_1' \in \mathrm{SN(jn)}$ by the *i.h.* on $\mathtt{R}$ since $\langle\|t_1\|_{\mathrm{jn}}, \|t\|_{\mathrm{jn}}, \|u\|_{\mathrm{jn}}\rangle = \langle\|t_0\|_{\mathrm{jn}}, \|t\|_{\mathrm{jn}}, \|u\|_{\mathrm{jn}}\rangle$. Notice that when $\mathtt{R} = \Diamond$, then π-reduction can only take place in some subterm of t_0', already considered in the previous cases. □

Lemma 4.74. *If* $t_0 = \mathtt{R}\langle\mathtt{n}(u, y.\mathtt{a}S)\rangle \in \mathrm{SN(jn)}$, *then* $t_0' = \mathtt{R}\langle\mathtt{n}(u, y.\mathtt{a})S\rangle \in \mathrm{SN(jn)}$.

Proof. We use the lexicographic order to reason by induction on $\langle\|t_0\|_{\mathrm{jn}}, \mathtt{n}\rangle$. To show $t_0' \in \mathrm{SN(jn)}$ it is sufficient to show that all its reducts are in SN(jn). We analyze all possible cases.

Case $t_0' \to_\pi t_0$. We conclude by the hypothesis.

Case $t_0' \to_{\mathrm{jn}} \mathtt{R}\langle\mathtt{n}'(u, y.\mathtt{a})S\rangle = t_1'$, where $\mathtt{n} \to_{\mathrm{jn}} \mathtt{n}'$. We have $t_0 \to_{\mathrm{jn}} \mathtt{R}\langle\mathtt{n}'(u, y.\mathtt{a}S)\rangle = t_1$, so that also $t_1 \in \mathrm{SN(jn)}$. We conclude $t_1' \in \mathrm{SN(jn)}$ by the *i.h.* since $\|t_1\|_{\mathrm{jn}} < \|t_0\|_{\mathrm{jn}}$.

Case $t_0' \to_{\mathrm{jn}} \mathtt{R}\langle\mathtt{n}(u', y.\mathtt{a})S\rangle = t_1'$, where $u \to_{\mathrm{jn}} u'$. We have $t_0 \to_{\mathrm{jn}} \mathtt{R}\langle\mathtt{n}(u', y.\mathtt{a}S)\rangle = t_1$, so that also $t_1 \in \mathrm{SN(jn)}$. We conclude $t_1' \in \mathrm{SN(jn)}$ by the *i.h.* since $\|t_1\|_{\mathrm{jn}} < \|t_0\|_{\mathrm{jn}}$.

Case $t_0' \to_{\mathrm{jn}} \mathtt{R}\langle\mathtt{n}(u, y.a')S\rangle = t_1'$, where $\mathtt{a} \to_{\mathrm{jn}} \mathtt{a}'$. We have $t_0 \to_{\mathrm{jn}} \mathtt{R}\langle\mathtt{n}(u, y.\mathtt{a}'S)\rangle = t_1$, so that also $t_1 \in \mathrm{SN(jn)}$. We conclude $t_1' \in \mathrm{SN(jn)}$ by the *i.h.* since $\|t_1\|_{\mathrm{jn}} < \|t_0\|_{\mathrm{jn}}$.

Case $t_0' \to_{\mathrm{jn}} \mathtt{R}\langle\mathtt{n}(u, y.\mathtt{a})S'\rangle = t_1'$, where $S \to_{\mathrm{jn}} S'$. We have $t_0 \to_{\mathrm{jn}} \mathtt{R}\langle\mathtt{n}(u, y.\mathtt{a}S')\rangle = t_1$, so that also $t_1 \in \mathrm{SN(jn)}$. We conclude $t_1' \in \mathrm{SN(jn)}$ by the *i.h.* since $\|t_1\|_{\mathrm{jn}} < \|t_0\|_{\mathrm{jn}}$.

Case $\mathtt{R} = \mathtt{R}'\langle\Diamond S'\rangle$. Thus, $t_0' = \mathtt{R}'\langle\mathtt{n}(u, y.\mathtt{a})(u', z.r)S'\rangle \to_\pi \mathtt{R}'\langle\mathtt{n}(u, y.\mathtt{a})(u', z.rS')\rangle = t_1'$, where $S = (u', z.r)$. Then, $t_0 = \mathtt{R}'\langle\mathtt{n}(u, y.\mathtt{a}(u', z.r))S'\rangle \to_\pi^2 \mathtt{R}'\langle\mathtt{n}(u, y.\mathtt{a}(u', z.rS'))\rangle = t_1$, so that also $t_1 \in \mathrm{SN(jn)}$. We conclude $t_1' \in \mathrm{SN(jn)}$ by the *i.h.* since $\|t_1\|_{\mathrm{jn}} < \|t_0\|_{\mathrm{jn}}$.

Case $\mathtt{n} = \mathtt{n}''(u', z.\mathtt{n}')$. Thus $t_0' = \mathtt{R}\langle \mathtt{n}''(u', z.\mathtt{n}')(u, y.\mathtt{a})S\rangle \to_\pi^2 \mathtt{R}\langle \mathtt{n}''(u', z.\mathtt{n}'(u, y.\mathtt{a})S)\rangle = t_1'$. We do a case analysis on all the one-step reducts of t_0' so we need to consider t_1' with S outside. We have $t_0 \to_\pi \mathtt{R}\langle \mathtt{n}''(u', z.\mathtt{n}'(u, y.\mathtt{a}S))\rangle = t_1$, so that also $t_1 \in \mathrm{SN(jn)}$. Let $\mathtt{R}' = \mathtt{R}\langle \mathtt{n}''(u', z.\Diamond)\rangle$. We have $\|t_1\|_{\mathrm{jn}} < \|t_0\|_{\mathrm{jn}}$ so by the *i.h.*$\mathtt{R}'\langle \mathtt{n}'(u, y.\mathtt{a})S\rangle \in \mathrm{SN(jn)}$. Because $\mathtt{n}'(u, y.\mathtt{a})$ is an answer we can apply the *i.h.* on $\mathtt{n}''$ and we conclude $t_1' \in \mathrm{SN(jn)}$. □

Lemma 4.75. ISNj = SN(jn).

Proof. First, we show ISNj ⊆ SN(jn). We proceed by induction on $t \in \mathrm{ISNj}$.

Case $t = x$. Straightforward.

Case $t = \lambda x.s$, where $s \in \mathrm{ISNj}$. By the *i.h.* $s \in \mathrm{SN(jn)}$, so that $t \in \mathrm{SN(jn)}$ trivially holds.

Case $t = \mathtt{n}(u, x.r)$ where $\mathtt{n}, u, r \in \mathrm{ISNj}$ and $r \in \mathrm{NF}_{\mathrm{lr}}$. Since n is stable by reduction, n cannot in particular reduce to an answer. Therefore any kind of reduction starting at t only occurs in the subterms n, u and r. We conclude since $\mathtt{n}, u, r \in \mathrm{SN(jn)}$ hold by the *i.h.*

Case $t = \mathtt{R}\langle \mathtt{n}(u, y.\mathtt{a})S\rangle$, where $\mathtt{R}\langle \mathtt{n}(u, y.\mathtt{a}S)\rangle \in \mathrm{ISNj}$. The *i.h.* gives $\mathtt{R}\langle \mathtt{n}(u, y.\mathtt{a}S)\rangle \in \mathrm{SN(jn)}$, so that $t \in \mathrm{SN(jn)}$ holds by lemma 4.74.

Case $t = \mathtt{R}\langle(\lambda x.s)(u, y.r)\rangle$, where $\mathtt{R}\langle r\{y/s\{x/u\}\}\rangle, s, u \in \mathrm{ISNj}$. The *i.h.* gives $\mathtt{R}\langle r\{y/s\{x/u\}\}\rangle \in \mathrm{SN(jn)}$, $s \in \mathrm{SN(jn)}$ and $u \in \mathrm{SN(jn)}$ so that $t \in \mathrm{SN(jn)}$ holds by lemma 4.73.

Next, we show SN(jn) ⊆ ISNj. Let $t \in \mathrm{SN(jn)}$. We reason by induction on $\langle \|t\|_{\mathrm{jn}}, |t|\rangle$ w.r.t. the lexicographic order. If $\langle \|t\|_{\mathrm{jn}}, |t|\rangle$ is minimal, *i.e.* $\langle 0, 1\rangle$, then t is a variable and thus in ISNj by rule (SNVAR). Otherwise we proceed by case analysis.

Case $t = \lambda x.s$. Since $\|s\|_{\mathrm{jn}} = \|t\|_{\mathrm{jn}}$ and $|s| < |t|$, we conclude by the *i.h.* and rule (SNABS).

Case t is an application. There are three cases.

- **Subcase** $t \in \mathrm{NF}_{\mathrm{lr}}$. Then $t = \mathtt{n}(u, x.r)$ with $\mathtt{n}, u, r \in \mathrm{SN(jn)}$ and $r \in \mathrm{NF}_{\mathrm{lr}}$. We have $\|\mathtt{n}\|_\beta \le \|t\|_{\mathrm{jn}}$, $\|u\|_{\mathrm{jn}} \le \|t\|_{\mathrm{jn}}$, $\|r\|_{\mathrm{jn}} \le \|t\|_{\mathrm{jn}}$, $|\mathtt{n}| < |t|$, $|u| < |t|$ and $|r| < |t|$. By the *i.h.* $\mathtt{n}, u, r \in \mathrm{ISNj}$ and thus we conclude by rule (SNAPP).
- **Subcase** $t = \mathtt{R}\langle(\lambda x.s)(u, y.r)\rangle$. $t \in \mathrm{SN(jn)}$ implies in particular $\mathtt{R}\langle r\{y/s\{x/u\}\}\rangle, s, u \in \mathrm{SN(jn)}$, so that they are in ISNj by the *i.h.* We conclude that $t \in \mathrm{ISNj}$ by rule (SNREDEX2).
- **Subcase** $t = \mathtt{R}\langle \mathtt{n}(u, y.\mathtt{a})S\rangle$. $t \in \mathrm{SN(jn)}$ implies in particular $\mathtt{R}\langle \mathtt{n}(u, y.\mathtt{a}S)\rangle \in \mathrm{SN(jn)}$, so that this term is in ISNj by the *i.h.* We conclude $t \in \mathrm{ISNj}$ by rule (SNREDEX1).

□

4.7 Conclusion

Generalizing elimination rules of natural deduction is an old idea, occurring several times in the literature, most notably by Schroeder-Heister [Sch84a; Sch84b] or Tennant [Ten92; Ten02], before being coined in the version at the origin of ΛJ by von Plato [vPla01]. The generalization of implication elimination itself has come up independently along the years, as pointed out by Schroeder-Heister [Sch14].

Concerning ΛJ, some interesting results were given, motivated by a proof-theoretical approach. In parallel to his works with Joachimski [JM00; JM03] introducing the calculus, Matthes [Mat01] proves an interpolation theorem (with information on terms) for ΛJ extended with pairs and sum datatypes. In his PhD thesis, Barral [Bar08] defines a set of conversions for ΛJ beyond β and π. Some of these conversions where already given by Matthes [Mat01], another one is an undirected version of p2.

Espírito Santo and his coauthors have used ΛJ, and his multiary extension ΛJ_m [EP03] to compare the computational content of natural deduction and the sequent calculus [Esp09; EFP18]. Our results on the λ-calculus with generalized applications might be extended to ΛJ_m, a fragment of the sequent calculus, and give a new perspective on computational interpretations of the sequent calculus. Extending generalized applications to the classical case, in the spirit of the $\lambda\mu$-calculus could also be insightful.

When introducing operational semantics with distance, we have kept the homogeneity between CbN and CbV: we have distant rules that only differ by the notion of substitution. We would like to consider further unification between CbN and CbV with the help of generalized applications in the setting of the polarized lambda-calculus [Esp17] or call-by-push-value [Lev06]. Both formalisms subsume CbN and CbV, by allowing to express them within the same calculus.

An interesting line of works involving generalized applications is currently being developed, starting with Geuvers and Hurkens [GH16]. In these works, inference systems are derived from truth table, with elimination rules having a generalized shape, akin to von Plato's system. They give definition of proof terms for derived systems (only intuitionistic) [GH18], for which they prove strong normalization [GvdGH19; Abe21]. Interestingly, the standard implication introduction rule is replaced in their system by two rules. It would be interesting to look at the peculiarities of a λ-calculus using generalized applications and the two derived forms of abstractions.

Finally, Díaz-Caro and Dowek [DD22] use generalized elimination rules in a calculus for scalar addition and multiplication. However, they keep Gentzen's original rule for the application. We hope to have shown the interest of generalized applications in an abstract programming languages with our work.

The works cited above give a qualitative, but not quantitative analysis of (strong) normalization. Likewise, intersection type systems for ΛJ have been given by Matthes [Mat00], and by [EIL12] through an embedding in a more general calculus. However, their type systems are idempotent. Switching to non-idempotence reveals the quantitative failure of the permutative reduction π. That failure leads us to devise a calculus λJ_n compatible with quantitative models, and give one such model as a type system. Several other calculi have been adapted to enable quantitative analyzes: this is for instance the case of $\lambda\mu$ [KV20] or the Curry-Howard

interpretation of the intuitionistic sequent calculus $\bar{\lambda}$ [KV15].

It would be interesting to see if the techniques developed for tightness [AGK20; KV22] can be adapted to this framework. The precise measures on reduction length obtained would enable us to precisely measure the quantitative relationship between the CbN λ-calculus and λJ_n. Such techniques could also be adopted for CbV, to sharpen the relation between λJ_v and CbV calculi.

CHAPTER 5

Conclusion

Intermediate conclusions were given at the end of each chapter, as well as pointers to future directions of work. We now give a global overview of our contributions. Let us recall the question at the center of this work.

> What contributions do node replication and generalized applications, analyzed quantitatively, provide to the theory of programming languages?

Node replication. Node replication is an original implementation of substitution in the λ-calculus. We have abstracted it from other features of the original *atomic λ-calculus* of Gundersen, Heijltjes, and Parigot [GHP13b], and given an implementation in terms of explicit substitutions as a new calculus λR. The *essence* of node replication is put forward, and brought to the well-understood setting of calculi with explicit substitutions. The use of distance emphasizes the computational part of the calculus, each step either being an instance of B-rule or replicating one node of a term.

Node replication allows a *fine-grained* substitution of terms, necessary for optimality in weak and strong settings. Optimality relies on optimizations, such as *full laziness* in the weak case, which are possible because only parts of terms can be duplicated. We have shown concretely how node replication can be used for full laziness by giving an operational description of the splitting operation between a skeleton and the MFEs, and a fully lazy CbNeed strategy.

The obtained formalism is relatively simple. Besides fully lazy call-by-need, node replication can be used to implement different strategies: we have given a call-by-name strategy; a (fully lazy) call-by-value strategy could also be defined in this setting. Our splitting operation, crucial for CbNeed, can indeed be used modularly for different forms of evaluation. Substitution by node replication can be combined with other kinds of substitutions as well, as demonstrated by our CbNeed strategy, which also relies on linear substitution.

Generalized applications. Calculi with generalized applications add new conversions to the λ-calculus. In particular, permutation π puts the leftmost application on top of the term. This can be thought of as implementing a search for a redex. Therefore, using π simplifies evaluation contexts of the calculus, and provides very simple inductive definitions of normal form. This can be seen in our local versions of the strategies, in particular in a very natural *leftmost-outermost* call-by-value strategy. To go further, generalized applications can be converted, without affecting normalization, to $v_1(v_2, x.r)$, a shape which closely resembles *ANF (administrative normal form)* (see section 3.8).

Those conversions can alternatively be integrated directly into the computational rules, thus giving a simple framework defined by means of *distance*, fit for quantitative measures

and models. We have introduced two such distant calculi, for CbN and CbV. This formalism is closer to the λ-calculus, as it does not carry out the search for a redex. However, applications are still shared by the constructors in the grammar. Sharing only the applications, and all of them, simplifies the semantics and syntax of the calculus compared to calculi with explicit substitutions or let bindings. But, this is the most important: in CbV and CbNeed, values are substituted, while applications representing a pending computation are kept shared to avoid duplicating work.

A particular feature of calculi with generalized applications is indeed its elegant theory of CbV. It relies on a reduction rule differing from CbN only in the meta-level substitution (both for distant and local versions), keeping the same pattern of redexes.

Thus, CbV generalized applications are well-behaved and retain the good properties of CbN. We have demonstrated it by giving an operational characterization of *CbV solvability*. The characterization is given by a syntactical definition of normal forms, and of an operational reduction relation reducing terms to that kind of normal form. The characterization is somewhat more complex than in CbN, because of the different behavior of CbV concerning erasure. However, no ad-hoc techniques are necessary. Moreover, the difference between the two kinds of solvability are visible in the reduction. The good behavior of CbV is also demonstrated by the leftmost-outermost value reduction, which adopts the same inductive rules as a potential one for CbN.

Quantitative types. Our approach was guided by quantitative type systems. Quantitative types subsume idempotent intersection types, in that they offer the same qualitative characterizations plus quantitative measures. We have captured semantical properties of different systems, namely normalization, solvability and potential valuability. These characterizations enable simple proofs of otherwise involved theorems like the normalization property and observational equivalence.

We have indeed used the characterizations to relate normalization of different formalisms. In node replication, we have shown that fully lazy CbNeed is observationally equivalent to the usual CbN with full substitution and to the semantical notion of neededness. For generalized applications, we have shown that for strong normalization, solvability and potential valuability, local and distant versions correspond. This holds even in CbN, where the distant calculus does not rely on the same permutation rule as the original. This validates our choice of using distance, better for quantitative analysis, without changing the qualitative semantics of the calculus. We have extended these results to show equivalence of strong normalization, solvability and potential valuability, respectively, between generalized applications and the λ-calculus.

Quantitativity is a first step toward complexity analysis, giving in particular *upper bounds* on the length of reduction and on the size of normal forms automatically from the size of derivations. None of the calculi at the origin of our work had been previously analyzed quantitatively. The type systems and logical characterizations we provide are all new.

These type systems have influenced the operational semantics of our new calculi. They have lead us to use *distance* primarily. Also, the quantitative analysis of generalized applications has revealed that the behavior of π-conversions is not quantitatively appropriate for a

CbN calculus. This lead us to consider another reduction rule, in order to stay quantitatively coherent with the λ-calculus.

Non-idempotence also simplifies the proofs of soundness: a typable term is normalizing simply because the size of the type derivation decreases at each step. Only for the strong normalization did we have to complete the combinatorial proofs, which lead us to give an original *inductive definition* of strong normalization for CbN generalized applications.

Final words. Only a first step in going "from proof-terms to programs" has been accomplished. To go the full path into programs and implementation, a full semantics based on node replication or generalized applications should be devised.

The first step is to devise *abstract machines* for strategies using node replication (in particular fully lazy CbNeed) and generalized applications. Generalized applications seem to be a good starting point for an abstract machine, as they can be transformed to a kind of ANF, a representation giving access to optimizations in abstract machines [Acc+19].

Beyond abstract machines, ANFs are used in many concrete implementations as intermediate representations for compilers. We would like to investigate whether generalized applications and ANF differ substantially. The full syntax of generalized applications could serve as a good first intermediate language between a language based on the λ-calculus and an ANF representation.

Implementation should be guided by a complexity analysis. We aim at reasonable abstract machines, implementing constant or polynomial operations. For full laziness, it is unclear whether the splitting operation can be implemented in polynomial time. We conjecture that generalized applications are reasonable, because such a result is achieved in [Acc+19] with ANFs.

In parallel, the syntax of the calculus should be expanded with usual constructors and constants. This asks to expand our operational semantics for node replication and the splitting of a skeleton and MFEs to other constructors. Concerning the generalized applications, this means adopting generalized forms for the elimination constructors, and see how our results can be adapted. Some constructors such as the disjunction will have several continuations, and we would have to be cautious to devise permutations that do not duplicate subterms.

The complexity analysis can also be refined in the quantitative model. For this, we could adopt *tight* type systems, precise quantitative type systems from which we can extract exact bounds on the length of reduction and the size of normal forms. Tight types could also enable us to precisely compare our formalisms to the λ-calculus.

Finally, as the subtitle of the thesis suggests, we have been working in an intuitionistic setting. All of our work could be expanded to the classical case, thus integrating *control operators* to the calculi. For node replication, we could get inspiration from Fanny He's [He18] atomic $\lambda\mu$-calculus, which extends the atomic λ-calculus to the classical case. Integrating λ-calculi with generalized eliminations to a classical setting is interesting because of their links to proof theory, and to the sequent calculus, where classical logic is better understood than in natural deduction.

To summarize, we have provided formalisms for node replication and generalized applications, that enjoy the main advantage of the λ-calculus: the emphasis on core components

of computation. These systems can be used as a core for functional languages, or as a basis for more theoretical studies of substitution, optimality, conversions or call-by-value.

Bibliography

[Aba+91] Martin Abadi, Luca Cardelli, Pierre-Louis Curien, and Jean-Jacques Lévy. "Explicit Substitutions." In: *Journal of Functional Programming* 1.4 (October 1991), pages 375–416. DOI: 10.1017/s0956796800000186 (cited on page 121).

[Abe+20] Margaux Abello, Juliette Courson, Arnaud Maury, and Julian Renaud. "String Shooter's Overall Shape in Ambient Air." In: *Emergent Scientist* 4 (2020), page 1. DOI: 10.1051/emsci/2019007 (cited on page x).

[Abe21] Andreas Abel. "On Model-Theoretic Strong Normalization for Truth-Table Natural Deduction." In: *26th International Conference on Types for Proofs and Programs (TYPES 2020)*. Edited by Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch. Volume 188. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 1:1–1:21. ISBN: 9783959771825. DOI: 10.4230/LIPIcs.TYPES.2020.1. URL: https://drops.dagstuhl.de/opus/volltexte/2021/13880 (visited on June 16, 2021) (cited on page 239).

[ABM14] Beniamino Accattoli, Pablo Barenbaum, and Damiano Mazza. "Distilling Abstract Machines." In: *Proceedings of the 19th ACM SIGPLAN international conference on Functional programming - ICFP '14*. ACM Press, 2014. DOI: 10.1145/2628136.2628154 (cited on pages 7, 37, 121, 190).

[Acc+19] Beniamino Accattoli, Andrea Condoluci, Giulio Guerrieri, and Claudio Sacerdoti Coen. "Crumbling Abstract Machines." In: *Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages 2019*. ACM, October 2019. DOI: 10.1145/3354166.3354169 (cited on pages 190, 243).

[Acc12] Beniamino Accattoli. "An Abstract Factorization Theorem for Explicit Substitutions." In: *23rd International Conference on Rewriting Techniques and Applications (RTA'12)*. Edited by Ashish Tiwari. Volume 15. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012, pages 6–21. ISBN: 978-3-939897-38-5. DOI: 10.4230/LIPIcs.RTA.2012.6. URL: http://drops.dagstuhl.de/opus/volltexte/2012/3481 (cited on page 223).

[Acc15] Beniamino Accattoli. "Proof Nets and the Call-by-Value λ-calculus." In: *Theoretical Computer Science* 606 (November 2015), pages 2–24. DOI: 10.1016/j.tcs.2015.08.006 (cited on page 188).

[Acc18a] Beniamino Accattoli. "(In)Efficiency and Reasonable Cost Models." In: *Electronic Notes in Theoretical Computer Science* 338 (October 2018), pages 23–43. DOI: 10.1016/j.entcs.2018.10.003 (cited on pages 36, 122).

[Acc18b] Beniamino Accattoli. "Proof Nets and the Linear Substitution Calculus." In: *Theoretical Aspects of Computing – ICTAC 2018*. Springer International Publishing, 2018, pages 37–61. DOI: 10.1007/978-3-030-02508-3_3 (cited on pages 7, 37, 121).

[ACS21] Beniamino Accattoli, Andrea Condoluci, and Claudio Sacerdoti Coen. "Strong Call-by-Value is Reasonable, Implosively." In: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, June 2021. DOI: 10.1109/lics52264.2021.9470630 (cited on pages 44, 122, 182).

[AD16] Beniamino Accattoli and Ugo Dal Lago. "(Leftmost-Outermost) Beta Reduction is Invariant, Indeed." In: *Logical Methods in Computer Science* 12.1 (March 2016). Edited by Dale Miller. DOI: 10.2168/lmcs-12(1:4)2016 (cited on page 122).

[AF97] Zena M. Ariola and Matthias Felleisen. "The Call-by-Need Lambda Calculus." In: *Journal of functional programming* 7.3 (May 1997), pages 265–301. DOI: 10.1017/s0956796897002724 (cited on pages 8, 39, 91, 121–123).

[AG16] Beniamino Accattoli and Giulio Guerrieri. "Open Call-by-Value." In: *Programming Languages and Systems*. Volume abs/1609.00322. Springer International Publishing, 2016, pages 206–226. DOI: 10.1007/978-3-319-47958-3_12. arXiv: 1609.00322 (cited on pages 9, 39, 188).

[AG22] Beniamino Accattoli and Giulio Guerrieri. "The Theory of Call-by-Value Solvability." In: *Proceedings of the ACM on Programming Languages* 6.ICFP (August 2022). Submitted to ICFP, pages 855–885. DOI: 10.1145/3547652 (cited on pages 13, 43, 156, 165, 171, 173, 175, 178, 181, 182, 188, 189).

[AGK20] Beniamino Accattoli, Stéphane Graham-Lengrand, and Delia Kesner. "Tight Typings and Split Bounds, Fully Developed." In: *Journal of Functional Programming* 30.ICFP (2020). ISSN: 2475-1421. DOI: 10.1017/s095679682000012x (cited on pages 123, 184, 240).

[AGL19] Beniamino Accattoli, Giulio Guerrieri, and Maico Leberle. "Types by Need." In: *Programming Languages and Systems*. Springer International Publishing, February 15, 2019, pages 410–439. DOI: 10.1007/978-3-030-17184-1_15. arXiv: 1902.05945 [cs.LO] (cited on pages 36, 94, 123).

[AGL21] Beniamino Accattoli, Giulio Guerrieri, and Marco Leberle. "Semantic Bounds and Strong Call-by-Value Normalization." April 28, 2021. eprint: arXiv:2104.13979. URL: https://arxiv.org/abs/2104.13979v1 (cited on pages 182–184).

[AK10] Beniamino Accattoli and Delia Kesner. "The Structural λ-calculus." In: *Computer Science Logic*. working paper or preprint. Springer Berlin Heidelberg, October 2010, pages 381–395. DOI: 10.1007/978-3-642-15205-4_30. URL: https://hal.archives-ouvertes.fr/hal-00528228 (cited on pages 7, 37, 121).

[AKV20] Sandra Alves, Delia Kesner, and Daniel Ventura. "A Quantitative Understanding of Pattern Matching." In: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. DOI: 10.4230/LIPICS.TYPES.2019.3 (cited on pages 20, 36).

[AP12] Beniamino Accattoli and Luca Paolini. "Call-by-Value Solvability, Revisited." In: *Functional and Logic Programming*. Springer Berlin Heidelberg, 2012. DOI: 10.1007/978-3-642-29822-6_4 (cited on pages 12, 13, 42, 43, 188).

[App91] Andrew W. Appel. *Compiling with Continuations*. Cambridge University Press, November 1991. DOI: 10.1017/cbo9780511609619 (cited on pages 35, 191).

[Bal+17] Thibaut Balabonski, Pablo Barenbaum, Eduardo Bonelli, and Delia Kesner. "Foundations of Strong Call by Need." In: *Proceedings of the ACM on Programming Languages* 1.ICFP (August 2017), pages 1–29. DOI: 10.1145/3110264 (cited on page 122).

[Bal12a] Thibaut Balabonski. "A Unified Approach to Fully Lazy Sharing." In: *Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '12*. Volume 47. ACM Press, January 2012, pages 469–480. DOI: 10.1145/2103656.2103713 (cited on page 122).

[Bal12b] Thibaut Balabonski. "La pleine paresse, une certaine optimalité. Partage de sous-termes et stratégies de réduction en récriture d'ordre supérieur." PhD thesis. Université Paris Diderot, November 16, 2012 (cited on pages 5, 85, 122).

[Bal13] Thibaut Balabonski. "Weak Optimality, and the Meaning of Sharing." In: *ACM SIGPLAN Notices* 48.9 (November 2013). DOI: 10.1145/2544174.2500606 (cited on page 32).

[Bar08] Freiric Barral. "Decidability for Non-Standard Conversions in Typed Lambda-Calculi." PhD thesis. Université de Toulouse III - Paul Sabatier and Ludwig-Maximilian Universität München, 2008 (cited on page 239).

[Bar84] Henk Barendregt. *The Lambda Calculus - Its Syntax and Semantics*. Elsevier, 1984. DOI: 10.1016/c2009-0-14341-6 (cited on pages 12, 42, 46, 189, 198).

[BCD83] Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. "A Filter Lambda Model and the Completeness of Type Assignment." In: *Journal of Symbolic Logic* 48.4 (December 1983), pages 931–940. DOI: 10.2307/2273659 (cited on page 35).

[BDD95] Franco Barbanera, Mariangiola Dezani-Ciancaglini, and U. De'Liguoro. "Intersection and Union Types: Syntax and Semantics." In: *Information and Computation* 119.2 (June 1995), pages 202–230. DOI: 10.1006/inco.1995.1086 (cited on page 36).

[BDS09] Henk Barendregt, Wil Dekkers, and Richard Statman. *Lambda Calculus with Types*. Cambridge University Press, 2009. DOI: 10.1017/cbo9781139032636 (cited on pages 35, 94).

[BE01] Antonio Bucciarelli and Thomas Ehrhard. "On Phase Semantics and Denotational Semantics: The Exponentials." In: *Annals of Pure and Applied Logic* 109.3 (May 2001), pages 205–241. DOI: 10.1016/s0168-0072(00)00056-7 (cited on page 97).

[BEM07] Antonio Bucciarelli, Thomas Ehrhard, and Giulio Manzonetto. "Not Enough Points Is Enough." In: *Computer Science Logic*. Edited by Jacques Duparc and Thomas A. Henzinger. Volume 4646. Lecture Notes in Computer Science. Lausanne, Switzerland: Springer Berlin Heidelberg, September 2007, pages 298–312. DOI: 10.1007/978-3-540-74915-8_24. URL: https://hal.archives-ouvertes.fr/hal-00148820 (cited on page 53).

[BKR21] Antonio Bucciarelli, Delia Kesner, and Simona Ronchi Della Rocca. "Solvability = Typability + Inhabitation." In: *Logical Methods in Computer Science* 17 (1 2021). DOI: 10.23638/LMCS-17(1:7)2021 (cited on page 188).

[BKR98] Nick Benton, Andrew Kennedy, and George Russell. "Compiling Standard ML to Java Bytecodes." In: *Proceedings of the third ACM SIGPLAN international conference on Functional programming - ICFP '98*. ACM Press, 1998. DOI: 10.1145/289423.289435 (cited on page 190).

[BKV17] Antonio Bucciarelli, Delia Kesner, and Daniel Ventura. "Non-idempotent Intersection Types for the Lambda-calculus." In: *Logic Journal of the IGPL* 25.4 (July 2017), pages 431–464. DOI: 10.1093/jigpal/jzx018 (cited on page 36).

[BL13] Alexis Bernadet and Stéphane Lengrand. "Non-idempotent Intersection Types and Strong Normalisation." In: *Logical Methods in Computer Science* 9.4 (October 2013). Edited by Marc Bezem, pages 17–42. DOI: 10.2168/lmcs-9(4:3)2013. URL: https://hal.inria.fr/hal-00906778 (cited on page 54).

[BLM05] Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget. "Sharing in the Weak Lambda-Calculus." In: *Processes, Terms and Cycles: Steps on the Road to Infinity*. Springer Berlin Heidelberg, 2005, pages 70–87. DOI: 10.1007/11601548_7 (cited on page 122).

[BLM07] Tomasz Blanc, Jean-Jacques Lévy, and Luc Maranget. "Sharing in the Weak Lambda-calculus Revisited." In: *Reflections on type theory, λ-calculus, and the mind*. 2007 (cited on page 122).

[BLM21] Thibaut Balabonski, Antoine Lanco, and Guillaume Melquiond. "A Strong Call-By-Need Calculus." In: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. DOI: 10.4230/LIPIcs.FSCD.2021.9 (cited on page 122).

[BN98] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, March 1998. DOI: 10.1017/cbo9781139172752 (cited on page 68).

[BR13] Erika De Benedetti and Simona Ronchi Della Rocca. "Bounding Normalization Time through Intersection Types." In: *Electronic Proceedings in Theoretical Computer Science* 121 (July 2013), pages 48–57. DOI: 10.4204/eptcs.121.4 (cited on page 54).

[BR95] Roel Bloo and Kristoffer H. Rose. "Preservation of Strong Normalisation in Named Lambda Calculi with Explicit Substitution and Garbage Collection." In: *Computing Science in the Netherlands (CSN'95, Utrecht, The Netherlands, November 27-28, 1995)*. Centrum voor Wiskunde en Informatica, 1995. ISBN: 90-6196-460-1 (cited on page 121).

[Brü10] Kai Brünnler. "Nested Sequents." In: *arXiv:1004.1845 [cs, math]* (April 2010). arXiv: 1004.1845 version: 1. URL: http://arxiv.org/abs/1004.1845 (visited on March 29, 2022) (cited on page 33).

[Buc+20] Antonio Bucciarelli, Delia Kesner, Alejandro Ríos, and Andrés Viso. "The Bang Calculus Revisited." In: *Functional and Logic Programming*. Springer International Publishing, 2020, pages 13–32. DOI: 10.1007/978-3-030-59025-3_2 (cited on pages 36, 49, 94, 155, 170).

[Cas21] Giuseppe Castagna. *Programming with Union, Intersection, and Negation Types*. 2021. DOI: 10.48550/ARXIV.2111.03354 (cited on page 36).

[CD80] Mario Coppo and Mariangiola Dezani-Ciancaglini. "An Extension of the Basic Functionality Theory for the λ-calculus." In: *Notre Dame Journal of Formal Logic* 21.4 (October 1980), pages 685–693. DOI: 10.1305/ndjfl/1093883253 (cited on pages 6, 35).

[CDV81] Mario Coppo, Mariangiola Dezani-Ciancaglini, and Betti Venneri. "Functional Characters of Solvable Terms." In: *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* 27.2-6 (1981). DOI: 10.1002/malq.19810270205 (cited on pages 35, 49).

[CF12] Stephen Chang and Matthias Felleisen. "The Call-by-Need Lambda Calculus, Revisited." In: *Programming Languages and Systems*. Springer Berlin Heidelberg, 2012, pages 128–147. DOI: 10.1007/978-3-642-28869-2_7 (cited on page 5).

[CF14] Stephen Chang and Matthias Felleisen. "Profiling for Laziness." In: *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '14*. ACM Press, 2014. DOI: 10.1145/2535838.2535887 (cited on page 121).

[CG14] Alberto Carraro and Giulio Guerrieri. "A Semantical and Operational Account of Call-by-Value Solvability." In: *Lecture Notes in Computer Science*. Volume 8412. Springer Berlin Heidelberg, 2014, pages 103–118. DOI: 10.1007/978-3-642-54830-7_7 (cited on pages 12, 42, 94, 188).

[CH00] Pierre-Louis Curien and Hugo Herbelin. "The Duality of Computation." In: *ACM SIGPLAN Notices* 35.9 (September 2000), pages 233–243. DOI: 10.1145/357766.351262 (cited on page 31).

[CH09] Felice Cardone and J. Roger Hindley. "Lambda-Calculus and Combinators in the 20th Century." In: *Handbook of the History of Logic*. Edited by Elsevier. Volume 5. Elsevier, 2009, pages 723–817. DOI: 10.1016/s1874-5857(09)70018-4 (cited on page 19).

[CH88] Thierry Coquand and Gérard Huet. "The Calculus of Constructions." In: *Information and Computation* 76.2-3 (February 1988), pages 95–120. DOI: 10.1016/0890-5401(88)90005-3 (cited on page 26).

[Chu32] Alonzo Church. "A Set of Postulates for the Foundation of Logic." In: *Annals of Mathematics* 33.2 (April 1932), pages 346–366. ISSN: 0003-486X. DOI: 10.2307/1968337 (cited on pages 2, 19).

[Chu36] Alonzo Church. "An Unsolvable Problem of Elementary Number Theory." In: *American Journal of Mathematics* 58.2 (April 1936), page 345. DOI: 10.2307/2371045 (cited on page 28).

[Chu40] Alonzo Church. "A Formulation of the Simple Theory of Types." In: *Journal of Symbolic Logic* 5.2 (June 1940), pages 56–68. DOI: 10.2307/2266170 (cited on page 27).

[CKP00] Roberto Di Cosmo, Delia Kesner, and Emmanuel Polonovski. "Proof Nets and Explicit Substitutions." In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2000, pages 63–81. DOI: 10.1007/3-540-46432-8_5 (cited on page 121).

[Con+19] Youyou Cong, Leo Osvald, Grégory M. Essertel, and Tiark Rompf. "Compiling with Continuations, or Without? Whatever." In: *Proceedings of the ACM on Programming Languages* 3.ICFP (July 2019), pages 1–28. DOI: 10.1145/3341643 (cited on page 191).

[CPT14] Luís Caires, Frank Pfenning, and Bernardo Toninho. "Linear Logic Propositions as Session Types." In: *Mathematical Structures in Computer Science* 26.3 (November 2014), pages 367–423. DOI: 10.1017/s0960129514000218 (cited on page 31).

[Cur34] Haskell B. Curry. "Functionality in Combinatory Logic." In: *Proceedings of the National Academy of Sciences* 20.11 (November 1934), pages 584–590. DOI: 10.1073/pnas.20.11.584 (cited on page 27).

[Dal+19] Ugo Dal Lago, Marc de Visme, Damiano Mazza, and Akira Yoshimizu. "Intersection Types and Runtime Errors in the Pi-calculus." In: *Proceedings of the ACM on Programming Languages* 3.POPL (January 2019), pages 1–29. DOI: 10.1145/3290320 (cited on page 36).

[dCar07] Daniel de Carvalho. "Sémantiques de la logique linéaire et temps de calcul." PhD thesis. Université de la Méditerranée Aix-Marseille II, 2007 (cited on pages 36, 94).

[dCar17] Daniel de Carvalho. "Execution Time of Λ-terms Via Denotational Semantics and Intersection Types." In: *Mathematical Structures in Computer Science* 28.7 (January 2017), pages 1169–1203. DOI: 10.1017/s0960129516000396 (cited on pages 35, 36, 138).

[dCPT11] Daniel de Carvalho, Michele Pagani, and Lorenzo Tortora de Falco. "A Semantic Measure of the Execution Time in Linear Logic." In: *Theoretical Computer Science* 412.20 (April 2011), pages 1884–1902. DOI: 10.1016/j.tcs.2010.12.017 (cited on page 165).

[DD22] Alejandro Díaz-Caro and Gilles Dowek. "Linear Lambda-calculus Is Linear." In: *arXiv:2201.11221 [cs]* (February 2022). arXiv: 2201.11221. URL: http://arxiv.org/abs/2201.11221 (visited on March 22, 2022) (cited on page 239).

[DG01] Mariangiola Dezani-Ciancaglini and Elio Giovannetti. "From Böhm's Theorem to Observational Equivalences: an Informal Account." In: *Electronic Notes in Theoretical Computer Science*. BOTH 2001, Bohm's theorem: applications to Computer Science Theory (Satellite Workshop of ICALP 2001) 50.2 (August 2001), pages 83–116. ISSN: 1571-0661. DOI: 10.1016/S1571-0661(04)00167-7 (cited on page 189).

[EFP06] José Espírito Santo, Maria João Frade, and Luís Pinto. "Structural Proof Theory as Rewriting." In: *Term Rewriting and Applications*. Edited by Frank Pfenning. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pages 197–211. ISBN: 9783540368359. DOI: 10.1007/11805618_15 (cited on page 194).

[EFP18] José Espírito Santo, Maria João Frade, and Luís Pinto. "Permutability in Proof Terms for Intuitionistic Sequent Calculus with Cuts." In: (2018). DOI: 10.4230/LIPICS.TYPES.2016.10 (cited on pages 4, 11, 35, 41, 239).

[Ehr12] Thomas Ehrhard. "Collapsing Non-idempotent Intersection Types." In: *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL*. Edited by Patrick Cégielski and Arnaud Durand. Volume 16. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012, pages 259–273. ISBN: 978-3-939897-42-2. DOI: 10.4230/LIPICS.CSL.2012.259 (cited on pages 36, 94).

[EIL12] José Espírito Santo, Jelena Ivetić, and Silvia Likavec. "Characterising Strongly Normalising Intuitionistic Terms." In: *Fundamenta Informaticae* 121 (2012). ISSN: 01692968. DOI: 10.3233/FI-2012-772 (cited on page 239).

[EKP22] José Espírito Santo, Delia Kesner, and Loïc Peyrot. "A Faithful and Quantitative Notion of Distant Reduction for Generalized Applications." In: *Lecture Notes in Computer Science*. Springer International Publishing, 2022, pages 285–304. DOI: 10.1007/978-3-030-99253-8_15 (cited on pages 37, 263).

[EP03] José Espírito Santo and Luís Pinto. "Permutative Conversions in Intuitionistic Multiary Sequent Calculi with Cuts." In: *Typed Lambda Calculi and Applications*. Edited by Martin Hofmann. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pages 286–300. ISBN: 9783540449041. DOI: 10.1007/3-540-44904-3_20 (cited on pages 194, 239).

[Esp07] José Espírito Santo. "Delayed Substitutions." In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2007, pages 169–183. DOI: 10.1007/978-3-540-73449-9_14 (cited on pages 126, 223).

[Esp09] José Espírito Santo. "The λ-Calculus and the Unity of Structural Proof Theory." In: *Theory of Computing Systems* 45.4 (February 2009), pages 963–994. DOI: 10.1007/s00224-009-9183-9 (cited on pages 4, 11, 35, 41, 239).

[Esp13] José Espírito Santo. "Towards a Canonical Classical Natural Deduction System." In: *Annals of Pure and Applied Logic* 164.6 (June 2013), pages 618–650. DOI: 10.1016/j.apal.2012.05.008 (cited on page 35).

[Esp17] José Espírito Santo. "The Polarized λ-calculus." In: *Electronic Notes in Theoretical Computer Science*. LSFA 2016 - 11th Workshop on Logical and Semantic Frameworks with Applications (LSFA) 332 (June 2017), pages 149–168. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2017.04.010. URL: https://www.sciencedirect.com/science/article/pii/S1571066117300221 (visited on October 14, 2021) (cited on page 239).

[Esp20] José Espírito Santo. "The Call-By-Value Lambda-Calculus with Generalized Applications." In: *28th EASCL Annual Conference on Computer Science Logic (CSL 2020)*. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, 2020. DOI: 10.4230/LIPICS.CSL.2020.35 (cited on pages 3, 11, 34, 128, 197).

[Fla+93] Cormac Flanagan, Amr Sabry, Bruce F. Duba, and Matthias Felleisen. "The Essence of Compiling with Continuations." In: *Proceedings of the ACM SIGPLAN 1993 conference on Programming language design and implementation - PLDI '93*. ACM Press, 1993. DOI: 10.1145/155090.155113 (cited on pages 35, 127, 190).

[Gar94] Philippa Gardner. "Discovering Needed Reductions Using Type Theory." In: *Lecture Notes in Computer Science*. Edited by Masami Hagiya and John C. Mitchell. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pages 555–574. ISBN: 978-3-540-48383-0. DOI: 10.1007/3-540-57887-0_115 (cited on pages 6, 36, 49, 94, 138).

[Gen35a] Gerhard Gentzen. "Untersuchungen über das logische Schließen. I." In: *Mathematische Zeitschrift* 39.1 (December 1935), pages 176–210. DOI: 10.1007/bf01201353 (cited on pages 4, 29, 31).

[Gen35b] Gerhard Gentzen. "Untersuchungen über das logische Schließen. II." In: *Mathematische Zeitschrift* 39.1 (December 1935), pages 405–431. DOI: 10.1007/bf01201363 (cited on pages 4, 29, 31).

[GGP10] Alessio Guglielmi, Tom Gundersen, and Michel Parigot. "A Proof Calculus Which Reduces Syntactic Bureaucracy." In: *Proceedings of the 21st International Conference on Rewriting Techniques and Applications, RTA 2010* 6 (January 2010). DOI: 10.4230/LIPIcs.RTA.2010.135 (cited on pages 3, 33).

[GH16] Herman Geuvers and Tonny Hurkens. "Deriving Natural Deduction Rules from Truth Tables." In: *Logic and Its Applications*. Springer Berlin Heidelberg, December 2016, pages 123–138. DOI: 10.1007/978-3-662-54069-5_10 (cited on page 239).

[GH18] Herman Geuvers and Tonny Hurkens. "Proof Terms for Generalized Natural Deduction." In: *23rd International Conference on Types for Proofs and Programs (TYPES 2017)*. Edited by Andreas Abel, Fredrik Nordvall Forsberg, and Ambrus Kaposi. Volume 104. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 3:1–3:39. ISBN: 9783959770712. DOI: 10.4230/LIPIcs.TYPES.2017.3 (cited on page 239).

[GHP13a] Tom Gundersen, Willem Heijltjes, and Michel Parigot. "A Proof of Strong Normalisation of the Typed Atomic Lambda-Calculus." In: *Logic for Programming, Artificial Intelligence, and Reasoning*. Edited by Ken McMillan, Aart Middeldorp, and Andrei Voronkov. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pages 340–354. ISBN: 978-3-642-45221-5. DOI: 10.1007/978-3-642-45221-5_24 (cited on pages 91, 97, 121).

[GHP13b] Tom Gundersen, Willem Heijltjes, and Michel Parigot. "Atomic Lambda Calculus: A Typed Lambda-Calculus with Explicit Sharing." In: *2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE, June 2013. DOI: 10.1109/lics.2013.37 (cited on pages 3, 8, 32, 38, 39, 58, 60, 67, 76, 85, 86, 121, 241).

[GHP21] Giulio Guerrieri, Willem Heijltjes, and Joseph Paulus. "A Deep Quantitative Type System." In: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. DOI: 10.4230/LIPICS.CSL.2021.24 (cited on pages 94, 123).

[Gir00] Jean-Yves Girard. "Du pourquoi au comment: la théorie de la démonstration de 1950 à nos jours." In: *Development of Mathematics, 1950-2000*. Birkhäuser Basel, 2000, pages 515–545. DOI: 10.1007/978-3-0348-8968-1_17 (cited on page 4).

[Gir87] Jean-Yves Girard. "Linear Logic." In: *Theoretical Computer Science* 50.1 (1987), pages 1–101. DOI: 10.1016/0304-3975(87)90045-4 (cited on page 122).

[Gir89] Jean-Yves Girard. *Proofs and Types*. Cambridge University Press, 1989 (cited on page 30).

[GL02] Benjamin Grégoire and Xavier Leroy. "A Compiled Implementation of Strong Reduction." In: *Proceedings of the seventh ACM SIGPLAN international conference on Functional programming - ICFP '02*. ACM Press, 2002. DOI: 10.1145/581478.581501 (cited on pages 26, 122).

[GN16] Álvaro García-Pérez and Pablo Nogueira. "No Solvable Lambda-value Term Left Behind." In: *Logical Methods in Computer Science* 12.2 (June 2016). Edited by Neil Jones. DOI: 10.2168/lmcs-12(2:12)2016 (cited on pages 14, 44, 165, 189).

[GO21] Giulio Guerrieri and Federico Olimpieri. "Categorifying Non-Idempotent Intersection Types." In: *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*. Edited by Christel Baier and Jean Goubault-Larrecq. Volume 183. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021. ISBN: 978-3-95977-175-7. DOI: 10.4230/LIPIcs.CSL.2021.25 (cited on page 35).

[Göd31] Kurt Gödel. "Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I." In: *Monatshefte für Mathematik und Physik* 38-38.1 (December 1931), pages 173–198. DOI: 10.1007/bf01700692 (cited on page 28).

[GPD17] Giulio Guerrieri, Luca Paolini, and Simona Ronchi Della Rocca. "Standardization and Conservativity of a Refined Call-by-Value lambda-Calculus." In: *Logical Methods in Computer Science ; Volume 13* (2017). DOI: 10.23638/LMCS-13(4:29)2017 (cited on pages 170, 188).

[Gug07] Alessio Guglielmi. "A System of Interaction and Structure." In: *ACM Transactions on Computational Logic* 8.1 (January 2007), page 1. DOI: 10.1145/1182613.1182614 (cited on page 33).

[Gug15] Alessio Guglielmi. "Deep Inference." In: *All About Proofs, Proofs for All*. January 22, 2015 (cited on page 4).

[GvdGH19] Herman Geuvers, Iris van der Giessen, and Tonny Hurkens. "Strong Normalization for Truth Table Natural Deduction." In: *Fundamenta Informaticae* 170.1-3 (October 2019). Edited by Thorsten Altenkirch and Aleksy Schubert, pages 139–176. DOI: 10.3233/fi-2019-1858 (cited on page 239).

[He18] Fanny He. "The Atomic Lambda-Mu Calculus." PhD thesis. University of Bath, 2018 (cited on pages 122, 243).

[Hin84] J. Roger Hindley. "Coppo-Dezani Types Do Not Correspond to Propositional Logic." In: *Theoretical Computer Science* 28.1-2 (1984), pages 235–236. DOI: 10.1016/0304-3975(83)90074-9 (cited on page 35).

[Hol16] Simon Holywell. *Functional Programming in PHP*. php[architect], 2016. 176 pages. ISBN: 9781940111469 (cited on page 17).

[How80] William H. Howard. "The formulae-as-types notion of construction." In: *To H. B. Curry: Essays on Combinatory Logic, Lambda-Calculus and Formalism*. Edited by J. Seldin and J. Hindley. London Academic Press, 1980, pages 480–490 (cited on pages 4, 30).

[Hug83] John Hughes. *The Design And Implementation Of Programming Languages*. Technical report PRG40. OUCL, July 1983. 159 pages (cited on page 122).

[HvO03] Dimitri Hendriks and Vincent van Oostrom. "λ." In: *Lecture Notes in Computer Science* 2741 (2003). Proceedings title: Conference on Automated Deduction, pages 136–150. ISSN: 0302-9743 (cited on page 122).

[HZ09] Hugo Herbelin and Stéphane Zimmermann. "An Operational Account of Call-by-Value Minimal and Classical λ-Calculus in "Natural Deduction" Form." In: *Typed Lambda Calculi and Applications*. Edited by Pierre-Louis Curien. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pages 142–156. ISBN: 9783642022739. DOI: 10.1007/978-3-642-02273-9_12 (cited on page 188).

[JGS93] Neil D. Jones, Carsten K. Gomard, and Peter Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice Hall International, 1993, page 415. ISBN: 0-13-020249-5 (cited on page 26).

[JM00] Felix Joachimski and Ralph Matthes. "Standardization and Confluence for a Lambda Calculus with Generalized Applications." In: *Rewriting Techniques and Applications*. Springer Berlin Heidelberg, 2000, pages 141–155. DOI: 10.1007/10721975_10 (cited on pages 3, 11, 33, 128, 168, 196, 197, 239).

[JM03] Felix Joachimski and Ralph Matthes. "Short Proofs of Normalization for the Simply- Typed λ-calculus, Permutative Conversions and Gödel's T." In: *Archive for Mathematical Logic* 42.1 (January 2003). DOI: 10.1007/s00153-002-0156-9 (cited on pages 11, 33, 128, 223, 236, 239).

[Kah87] Gilles Kahn. "Natural Semantics." In: *STACS 87*. Springer-Verlag, 1987. DOI: 10.1007/bfb0039592 (cited on pages 8, 39).

[Ken07] Andrew Kennedy. "Compiling with Continuations, Continued." In: *ACM SIGPLAN Notices* 42.9 (October 2007), pages 177–190. DOI: 10.1145/1291220.1291179 (cited on page 191).

[Kes07] Delia Kesner. "The Theory of Calculi with Explicit Substitutions Revisited." In: *Computer Science Logic*. Springer Berlin Heidelberg, 2007, pages 238–252. DOI: 10.1007/978-3-540-74915-8_20 (cited on page 121).

[Kes09] Delia Kesner. "A Theory of Explicit Substitutions with Safe and Full Composition." In: *Logical Methods in Computer Science* Volume 5, Issue 3 (July 2009). DOI: 10.2168/LMCS-5(3:1)2009 (cited on page 20).

[Kes16] Delia Kesner. "Reasoning about Call-by-Need by Means of Types." In: *Lecture Notes in Computer Science*. Edited by Bart Jacobs and Christof Löding. Springer Berlin Heidelberg, 2016, pages 424–441. DOI: 10.1007/978-3-662-49630-5_25 (cited on pages 8, 23, 36, 39, 53, 94, 123).

[Kes22] Delia Kesner. "A Fine-Grained Computational Interpretation of Girard's Intuitionistic Proof-Nets." In: *Proceedings of the ACM on Programming Languages* 6.POPL (2022), pages 1–28. DOI: 10.1145/3498669 (cited on pages 7, 37).

[Kfo00] Assaf Kfoury. "A Linearization of the Lambda-calculus and Consequences." In: *Journal of Logic and Computation* 10.3 (June 2000), pages 411–436. DOI: 10.1093/logcom/10.3.411 (cited on page 36).

[KL07] Delia Kesner and Stéphane Lengrand. "Resource Operators for λ-calculus." In: *Information and Computation* 205.4 (April 2007), pages 419–473. DOI: 10.1016/j.ic.2006.08.008 (cited on pages 7, 37, 76).

[KMP20] Emma Kerinec, Giulio Manzonetto, and Michele Pagani. "Revisiting Call-by-value Böhm trees in light of their Taylor expansion." In: *Logical Methods in Computer Science* Volume 16, Issue 3 (July 2020). DOI: 10.23638/LMCS-16(3:6)2020 (cited on page 189).

[KN19] Steve Klabnik and Carol Nichols. *The Rust Programming Language*. No Starch Press, Incorporated, 2019, page 560. ISBN: 9781718500440. URL: https://doc.rust-lang.org/book/ (cited on page 17).

[KP22] Delia Kesner and Loïc Peyrot. "Solvability for Generalized Applications." In: *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*. Edited by Amy P. Felty. Volume 228. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, June 28, 2022, 18:1–18:22. ISBN: 978-3-95977-233-4. DOI: 10.4230/LIPIcs.FSCD.2022.18 (cited on pages 36, 37, 263).

[KPV21] Delia Kesner, Loïc Peyrot, and Daniel Ventura. "The Spirit of Node Replication." In: *Lecture Notes in Computer Science*. Springer International Publishing, 2021, pages 344–364. DOI: 10.1007/978-3-030-71995-1_18 (cited on pages 37, 263).

[KPV22] Delia Kesner, Loïc Peyrot, and Daniel Ventura. "Node replication: Theory and Practice." Submitted to LMCS. 2022 (cited on pages 37, 263).

[Kri02] Jean-Louis Krivine. *Lambda-Calculus Types and Models*. 2002 (cited on page 184).

[Kri07] Jean-Louis Krivine. "A Call-by-Name Lambda-calculus Machine." In: *Higher-Order and Symbolic Computation* 20.3 (October 2007), pages 199–207. DOI: 10.1007/s10990-007-9018-9 (cited on pages 10, 40).

[KRV18] Delia Kesner, Alejandro Ríos, and Andrés Viso. "Call-by-Need, Neededness and All That." In: *Lecture Notes in Computer Science*. Springer International Publishing, 2018, pages 241–257. DOI: 10.1007/978-3-319-89366-2_13 (cited on pages 9, 39, 123).

[Kup95] Jan Kuper. "Proving the Genericity Lemma by Leftmost Reduction is Simple." In: *Rewriting Techniques and Applications*. Springer Berlin Heidelberg, 1995, pages 271–278. DOI: 10.1007/3-540-59200-8_63 (cited on page 189).

[KV14] Delia Kesner and Daniel Ventura. "Quantitative Types for the Linear Substitution Calculus." In: *Lecture Notes in Computer Science*. Edited by Josep Diaz, Ivan Lanese, and Davide Sangiorgi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pages 296–310. ISBN: 978-3-662-44602-7. DOI: 10.1007/978-3-662-44602-7_23 (cited on pages 36, 49, 94, 170).

[KV15] Delia Kesner and Daniel Ventura. "A Resource Aware Computational Interpretation for Herbelin's Syntax." In: *Theoretical Aspects of Computing - ICTAC 2015*. Springer International Publishing, 2015, pages 388–403. DOI: 10.1007/978-3-319-25150-9_23 (cited on pages 36, 240).

[KV20] Delia Kesner and Pierre Vial. "Non-idempotent Types for Classical Calculi in Natural Deduction Style." In: *Logical Methods in Computer Science ; Volume 16* abs/1802.05494 (2020). DOI: 10.23638/LMCS-16(1:3)2020. arXiv: 1802.05494 (cited on pages 36, 54, 94, 228, 239).

[KV22] Delia Kesner and Andrés Viso. "Encoding Tight Typing in a Unified Framework." In: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. DOI: 10.4230/LIPICS.CSL.2022.27 (cited on page 240).

[KvOdV08] Jan Willem Klop, Vincent van Oostrom, and Roel de Vrijer. "Lambda Calculus With Patterns." In: *Theoretical Computer Science* 398.1-3 (May 2008), pages 16–31. DOI: 10.1016/j.tcs.2008.01.019 (cited on page 20).

[KvOdV96] Richard Kennaway, Vincent van Oostrom, and Fer-Jan de Vries. "Meaningless Terms in Rewriting." In: *Algebraic and Logic Programming*. Springer Berlin Heidelberg, 1996, pages 254–268. DOI: 10.1007/3-540-61735-3_17 (cited on page 189).

[Lam90] John Lamping. "An Algorithm for Optimal Lambda Calculus Reduction." In: *Proceedings of the 17th ACM SIGPLAN-SIGACT symposium on Principles of programming languages - POPL '90*. ACM Press, 1990. DOI: 10.1145/96709.96711 (cited on pages 3, 32).

[Leb21] Maico Carlos Leberle. "Dissecting Call-by-Need by Customizing Multi Type Systems." PhD thesis. Institut Polytechnique de Paris, May 7, 2021 (cited on page 14).

[Lev06] Paul Blain Levy. "Call-by-Push-Value: Decomposing Call-by-Value and Call-by-Name." In: *Higher-Order and Symbolic Computation* 19.4 (December 2006), pages 377–414. DOI: 10.1007/s10990-006-0480-6 (cited on page 239).

[Lév78] Jean-Jacques Lévy. "Réductions correctes et optimales dans le lambda-calcul." PhD thesis. Université Paris VII, 1978 (cited on page 122).

[Lév80] Jean-Jacques Lévy. "Optimal Reductions in the Lambda-Calculus." In: *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalisms*. Academic Press Inc, 1980, pages 159–191 (cited on pages 32, 122).

[LM99] Jean-Jacques Lévy and Luc Maranget. "Explicit Substitutions and Programming Languages." In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1999, pages 181–200. DOI: 10.1007/3-540-46691-6_14 (cited on page 32).

[Mar+99] John Maraist, Martin Odersky, David N. Turner, and Philip Wadler. "Call-by-Name, Call-by-Value, Call-by-Need, and the Linear Lambda Calculus." In: *Electronic Notes in Theoretical Computer Science* 1 (1999), pages 370–392. DOI: 10.1016/s1571-0661(04)00022-2 (cited on page 121).

[Mat00] Ralph Matthes. "Characterizing Strongly Normalizing Terms for a Lambda Calculus with Generalized Applications via Intersection Types." In: *Proc. ICALP 2000 (Geneva), volume 8 of Proceedings in Informatics*. 2000, pages 339–353 (cited on pages 14, 44, 229, 239).

[Mat01] Ralph Matthes. "Interpolation for Natural Deduction with Generalized Eliminations." In: *Proof Theory in Computer Science*. Springer Berlin Heidelberg, 2001, pages 153–169. DOI: 10.1007/3-540-45504-3_10 (cited on page 239).

[Mau+17] Luke Maurer, Paul Downen, Zena M. Ariola, and Simon Peyton Jones. "Compiling without Continuations." In: *ACM SIGPLAN Notices* 52.6 (September 2017), pages 482–494. DOI: 10.1145/3140587.3062380 (cited on page 191).

[Mic22] Microsoft. *The Typescript Handbook*. 2022. URL: https://www.typescriptlang.org/docs/handbook/2/objects.html#intersection-types (visited on July 4, 2022) (cited on page 36).

[Mil21] Dale Miller. *A Survey of the Proof-Theoretic Foundations of Logic Programming*. 2021. DOI: 10.48550/ARXIV.2109.01483 (cited on page 29).

[Mog91] Eugenio Moggi. "Notions of Computation and Monads." In: *Information and Computation* 93.1 (July 1991), pages 55–92. DOI: 10.1016/0890-5401(91)90052-4 (cited on page 20).

[MOW98] John Maraist, Martin Odersky, and Philip Wadler. "The Call-by-Need Lambda Calculus." In: *Journal of functional programming* 8.3 (May 1998), pages 275–317. DOI: 10.1017/s0956796898003037 (cited on pages 8, 39, 121).

[MPV18] Damiano Mazza, Luc Pellissier, and Pierre Vial. "Polyadic Approximations, Fibrations and Intersection Types." In: *Proceedings of the ACM on Programming Languages* 2.POPL (January 2018), pages 1–28. DOI: 10.1145/3158094 (cited on pages 35, 165).

[NM04] Peter Møller Neergaard and Harry G. Mairson. "Types, Potency, and Idempotency." In: *Proceedings of the ninth ACM SIGPLAN international conference on Functional programming - ICFP '04*. ACM Press, 2004. DOI: 10.1145/1016850.1016871 (cited on pages 28, 36).

[NvP01] Sara Negri and Jan von Plato. *Structural Proof Theory*. Cambridge University Press, June 2001. DOI: 10.1017/cbo9780511527340 (cited on page 34).

[Pao01] Luca Paolini. "Call-by-Value Separability and Computability." In: *Theoretical Computer Science*. Edited by Antonio Restivo, Simona Ronchi Della Rocca, and Luca Roversi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2001, pages 74–89. ISBN: 9783540454465. DOI: 10.1007/3-540-45446-2_5 (cited on page 189).

[Par92] Michel Parigot. "λμ-calculus: An Algorithmic Interpretation of Classical Natural Deduction." In: *Logic Programming and Automated Reasoning*. Edited by Andrei Voronkov. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1992. ISBN: 9783540472797. DOI: 10.1007/BFb0013061 (cited on page 31).

[Pey87] Simon Peyton Jones. *The Implementation of Functional Programming Languages*. Edited by C. A. R. Hoare. Prentice Hall, January 1987. URL: https://www.microsoft.com/en-us/research/publication/the-implementation-of-functional-programming-languages/ (cited on pages 85, 122).

[Plo75] Gordon D. Plotkin. "Call-by-Name, Call-by-Value and the Lambda-calculus." In: *Theoretical Computer Science* 1 (1975), pages 125–159. DOI: 10.1016/0304-3975(75)90017-1. (Visited on January 22, 2021) (cited on pages 5, 23).

[Plo77] Gordon D. Plotkin. "LCF Considered As a Programming Language." In: *Theoretical Computer Science* 5.3 (December 1977), pages 223–255. DOI: 10.1016/0304-3975(77)90044-5 (cited on page 20).

[PPR17] Luca Paolini, Mauro Piccolo, and Simona Ronchi Della Rocca. "Essential and Relational Models." In: *Mathematical Structures in computer science* 27.5 (2017), pages 626–650. DOI: 10.1017/S0960129515000316 (cited on page 35).

[PR10] Michele Pagani and Simona Ronchi della Rocca. "Solvability in Resource Lambda-Calculus." In: *Foundations of Software Science and Computational Structures*. Springer Berlin Heidelberg, 2010, pages 358–373. DOI: 10.1007/978-3-642-12032-9_25 (cited on page 36).

[PR99] Luca Paolini and Simona Ronchi Della Rocca. "Call-by-Value Solvability." In: *RAIRO - Theoretical Informatics and Applications* 33.6 (November 1999). DOI: 10.1051/ita:1999130 (cited on pages 12, 42, 149, 156, 171).

[Pra79] Dag Prawitz. "Proofs and the Meaning and Completeness of the Logical Constants." In: *Essays on Mathematical and Philosophical Logic*. Springer Netherlands, 1979, pages 25–40. DOI: 10.1007/978-94-009-9825-4_2 (cited on page 34).

[RDF20] Simona Ronchi Della Rocca, Ugo Dal Lago, and Claudia Faggian. "Solvability in a Probabilistic Setting." In: Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. DOI: 10.4230/LIPICS.FSCD.2020.1 (cited on page 36).

[Reg94] Laurent Regnier. "Une équivalence sur les lambda-termes." In: *Theor. Comput. Sci.* 126.2 (1994), pages 281–292. DOI: 10.1016/0304-3975(94)90012-4 (cited on pages 12, 31, 196, 197, 229).

[Rey74] John C. Reynolds. "Towards a Theory of Type Structure." In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1974, pages 408–425. DOI: 10.1007/3-540-06859-7_148 (cited on page 31).

[Sch14] Peter Schroeder-Heister. "Generalized Elimination Inferences, Higher-Level Rules, and the Implications-as-Rules Interpretation of the Sequent Calculus." In: *Trends in Logic*. Springer Netherlands, 2014, pages 1–29. DOI: 10.1007/978-94-007-7548-0_1 (cited on page 239).

[Sch84a] Peter Schroeder-Heister. "A Natural Extension of Natural Deduction." In: *Journal of Symbolic Logic* 49.4 (December 1984), pages 1284–1300. DOI: 10.2307/2274279 (cited on pages 34, 239).

[Sch84b] Peter Schroeder-Heister. "Generalized Rules for Quantifiers and the Completeness of the Intuitionistic Operators &, ∨, ⊃, f, ∀, ∃." In: *Computation and Proof Theory*. Springer Berlin Heidelberg, 1984, pages 399–426. DOI: 10.1007/bfb0099494 (cited on page 239).

[SF93] Amr Sabry and Matthias Felleisen. "Reasoning about Programs in Continuation-passing Style." In: *LISP and Symbolic Computation* 6.3-4 (November 1993). DOI: 10.1007/bf01019462 (cited on page 190).

[She+20] David Sherratt, Willem Heijltjes, Tom Gundersen, and Michel Parigot. "Spinal Atomic Lambda-Calculus." In: *Lecture Notes in Computer Science*. Springer International Publishing, 2020, pages 582–601. DOI: 10.1007/978-3-030-45231-5_30 (cited on page 122).

[She19] David Sherratt. "A Lambda-calculus That Achieves Full Laziness with Spine Duplication." PhD thesis. University of Bath, 2019 (cited on page 122).

[Str00] Christopher Strachey. "Fundamental Concepts in Programming Languages." In: *Higher-Order and Symbolic Computation* 13.1/2 (2000), pages 11–49. DOI: 10.1023/a:1010000313106 (cited on page 35).

[SU06] Morten Heine B. Sørensen and Paweł Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. Elsevier, 2006. DOI: 10.1016/s0049-237x(06)x8001-1 (cited on page 30).

[SW10] Olin Shivers and Mitchell Wand. "Bottom-up β-reduction: Uplinks and λ-DAGs." In: *Fundamenta Informaticae* 103.1-4 (2010), pages 247–287. DOI: 10.3233/fi-2010-328 (cited on page 122).

[Tai67] William W. Tait. "Intensional Interpretations of Functionals of Finite Type I." In: *Journal of Symbolic Logic* 32.2 (August 1967), pages 198–212. DOI: 10.2307/2271658 (cited on page 52).

[Tak94] Masako Takahashi. "A Simple Proof of the Genericity Lemma." In: *Logic, Language and Computation*. Springer-Verlag, 1994, pages 117–118. DOI: 10.1007/bfb0032397 (cited on page 189).

[Tak95] Masako Takahashi. "Parallel Reductions in λ-Calculus." In: *Information and Computation* 118.1 (April 1995), pages 120–127. DOI: 10.1006/inco.1995.1057 (cited on page 197).

[Ten02] Neil Tennant. "Ultimate Normal Forms for Parallelized Natural Deductions." In: *Logic Journal of IGPL* 10.3 (May 2002), pages 299–337. DOI: 10.1093/jigpal/10.3.299 (cited on page 239).

[Ten92] Neil Tennant. *Autologic*. Edinburgh: Edinburgh University Press, 1992. ISBN: 0748603581 (cited on pages 34, 239).

[Tiu06] Alwen Tiu. "A System of Interaction and Structure II: The Need for Deep Inference." In: *Logical Methods in Computer Science* 2.2 (April 2006). Edited by Prakash Panangaden. DOI: 10.2168/lmcs-2(2:4)2006 (cited on page 33).

[Tur37] Alan Turing. "On Computable Numbers, with an Application to the Entscheidungsproblem." In: *Proceedings of the London Mathematical Society* s2-42.1 (1937), pages 230–265. DOI: 10.1112/plms/s2-42.1.230 (cited on page 28).

[Urz99] Paweł Urzyczyn. "The Emptiness Problem for Intersection Types." In: *Journal of Symbolic Logic* 64.3 (September 1999), pages 1195–1215. DOI: 10.2307/2586625 (cited on pages 28, 36).

[vBak11] Steffen van Bakel. "Strict Intersection Types for the Lambda Calculus." In: *ACM Computing Surveys* 43.3 (April 2011), pages 1–49. DOI: 10.1145/1922649.1922657 (cited on page 35).

[vPla01] Jan von Plato. "Natural Deduction with General Elimination Rules." In: *Archive for mathematical logic* 40.7 (October 2001), pages 541–567. DOI: 10.1007/s001530100091 (cited on pages 4, 9, 34, 40, 239).

[vRaa96] Femke van Raamsdonk. "Confluence and Normalisation for Higher-order Rewriting." PhD thesis. Vrije Universiteit Amsterdam, May 13, 1996 (cited on page 200).

[vRos09] Guido van Rossum. *Origins of Python's "Functional" Features*. April 21, 2009. URL: https://python-history.blogspot.com/2009/04/origins-of-pythons-functional-features.html (visited on August 18, 2022) (cited on page 17).

[Wad71] Christopher P. Wadsworth. "Semantics and Pragmatics of the Lambda Calculus." PhD thesis. Oxford University, 1971 (cited on pages 5, 8, 23, 25, 38, 85, 121, 122).

[Wad76] Christopher P. Wadsworth. "The Relation between Computational and Denotational Properties for Scott's D_∞-Models of the Lambda-Calculus." In: 5.3 (September 1976), pages 488–521. DOI: 10.1137/0205036 (cited on pages 12, 42).

[WR10] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. 3 volumes. 1910 (cited on page 27).

[Zap22] Carlo Zapponi. *GitHut 2.0*. 2022. URL: https://madnight.github.io/githut/#/pull_requests/2022/1 (visited on June 24, 2022) (cited on page 27).

Articles liés à la thèse

Cette thèse reprend, revoit et étend les articles suivants, respectivement dans les chapitres 2 à 4 :

- Delia Kesner, Loïc Peyrot et Daniel Ventura. “Node replication : Theory and Practice”. Submitted to LMCS. 2022 Cet article a pour origine l’article de conférence suivant : Delia Kesner, Loïc Peyrot et Daniel Ventura. “The Spirit of Node Replication”. In : *Lecture Notes in Computer Science*. Springer International Publishing, 2021, pages 344-364. DOI : 10.1007/978-3-030-71995-1_18
- José Espírito Santo, Delia Kesner et Loïc Peyrot. “A Faithful and Quantitative Notion of Distant Reduction for Generalized Applications”. In : *Lecture Notes in Computer Science*. Springer International Publishing, 2022, pages 285-304. DOI : 10.1007/978-3-030-99253-8_15
- Delia Kesner et Loïc Peyrot. “Solvability for Generalized Applications”. In : *7th International Conference on Formal Structures for Computation and Deduction (FSCD 2022)*. Sous la direction d’Amy P. Felty. Tome 228. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany : Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 28 juin 2022, 18 :1-18 :22. ISBN : 978-3-95977-233-4. DOI : 10.4230/LIPIcs.FSCD.2022.18

Nomenclature

Abbreviations

ANF	administrative normal form
CbN	call-by-name
CbNeed	call-by-need
CbV	call-by-value
CPS	continuation-passing style
ES	explicit substitution
MFE	maximal free expression

Calculi

$\lambda\mu$ Classical calculus 31
λa Atomic λ-calculus 32
λES CbN calculus with explicit substitutions 48
λR Node replication calculus 57
ΛJ Original λ-calculus with generalized applications 128
ΛJ_v Original CbV λ-calculus with generalized applications 128
λJ_n Distant CbN λ-calculus with generalized applications 129
λJ_v Distant CbV λ-calculus with generalized applications 129
$\lambda_{\mathrm{v}}^{\sigma}$ Call-by-value λ-calculus with permutations 188
$\lambda_{\mathtt{vsub}}$ Call-by-value λ-calculus with explicit substitutions 188
$\mathtt{T}_J[\mathscr{R}]$ Calculus with generalized applications built on relation $\mathscr{R}$ 193

Terms and Contexts

$\mathscr{V}$ Variables 46
$\mathtt{T}_\Lambda$ λ-terms 45
$\mathtt{T}_{ES}$ λ-terms with explicit substitutions 46
$\mathtt{T}_J$ λ-terms with generalized applications 126
$\mathtt{T}_R$ λR-terms 58
$\mathtt{T}_P$ Pure subset of $\mathtt{T}_R$, isomorphic to $\mathtt{T}_\Lambda$ 58
$\mathtt{T}$ Linear cut values 78
$\mathtt{U}$ Restricted λR terms 78
$\mathrm{NF}_{\mathscr{R}}$ Normal forms of the reduction $\mathscr{R}$ 47

$\mathrm{NE}_{\mathscr{R}}$ Neutral normal forms of the reduction $\mathscr{R}$ 47
$\mathtt{n}$ Neutral normal forms of $\to_{\mathrm{lr}}$ 201
$\mathtt{a}$ Answers of $\to_{\mathrm{lr}}$ 201
$\mathrm{SN}(\mathscr{R})$ Set of strongly normalizing terms under $\to_{\mathscr{R}}$ 47
$\mathrm{ISN}(\mathscr{R})$ Inductively defined set of strongly normalizing terms under $\to_{\mathscr{R}}$ 200
ISNj Set of strongly normalizing terms in ΛJ 234

$t(u, x.r)$ Generalized application 126
$\overline{t(u, z.z)}^{n}$ $t(u, z.z) \dots (u, z.z)$ with n arguments 134
$[x/N]$ Explicit substitution 46
$[x/\!/\lambda y.u]$ Distributor 57
$[x \triangleleft u]$ Cut: explicit substitution or distributor 58
$\mathtt{I}$ Identity $\lambda x.x$ 19
δ $\lambda x.xx$ 19
Ω Looping term $(\lambda x.xx)(\lambda x.xx)$ 19
$\mathtt{o}^n$ $\lambda x_n \dots \lambda x_0.x_0$ 126

$\Diamond$ Empty context 47
$\mathtt{C}$ Full context for every grammar of terms 47
$\mathtt{L}$ List context ($\mathtt{T}_{ES}$, $\mathtt{T}_R$) 48
$\mathtt{LL}$ Linear list context ($\mathtt{T}_R$) 78
$\mathtt{D}$ Distant context ($\mathtt{T}_J$) 129
$\mathtt{D}_{\mathtt{n}}$ Distant neutral contexts ($\mathtt{T}_J$) 201
$\mathtt{H}$ Head contexts ($\mathtt{T}_\Lambda$, $\mathtt{T}_{ES}$, $\mathtt{T}_J$) 47, 49, 131
$\mathtt{G}$ Neutral head contexts ($\mathtt{T}_J$) 133
$\mathtt{W}$ Weak-head contexts ($\mathtt{T}_\Lambda$, $\mathtt{T}_{ES}$, $\mathtt{T}_J$) 47, 53, 201
$\mathtt{R}$ Left-right contexts ($\mathtt{T}_J$) 201
$\mathtt{N}$ Needed contexts ($\mathtt{T}_R$) 91
$\mathtt{C}\langle M\rangle$ Plugging of M in $\mathtt{C}$ 47
$\mathtt{C}\langle\!\langle M\rangle\!\rangle$ Plugging of M in $\mathtt{C}$ without capture 47

Functions on Terms

$M\{x/N\}$ Meta-level substitution of N for x in M 46
$t\{x\backslash\!\backslash u\}$ Left (meta-level) substitution 128
$\mathrm{fv}(M)$ Free variables of M 46
$\mathrm{bv}(M)$ Bound variables of M 46
$\mathrm{ndv}(t)$ Needed variables of t 92
$\mathrm{hv}(t)$ Head variables of t 131
$\mathrm{lv}_z(t)$ Level of variable z in t 61
$\mathrm{es}([x \triangleleft u])$ Equal to 1 if $[x \triangleleft u]$ is an explicit substitution, 0 if it is a distributor 58
$\{\!\{p\}\!\}^\theta$ θ-skeleton of p given by the inductive definition 85
$\{\!\{\!\{p\}\!\}\!\}^\theta$ θ-skeleton of p given by removing the MFEs 87

$|M|$ Size of the term M 46
$|M|_x$ Number of occurrences of x in M 46
$|t|_@$ Number of hereditary head variables of the $\mathtt{T}_J$-term t 134
$||M||_{\mathscr{R}}$ Maximum length of reduction $\mathscr{R}$ starting from t 47
dom(L) Domain of L 48
$(t)^{\mathrm{d}\beta}$ dβ-development of t 199
$(\cdot)^{\downarrow}$ Translation from $\mathtt{T}_{ES}$ and $\mathtt{T}_R$ to $\mathtt{T}_\Lambda$ (unfolding) 228
$(\cdot)^{\#}$ Translation from $\mathtt{T}_\Lambda$ to $\mathtt{T}_J$ 126
$(\cdot)^{\star}$ Translation from $\mathtt{T}_J$ to $\mathtt{T}_{ES}$ 126
$(\cdot)^{\star}$ New translation from $\mathtt{T}_J$ to $\mathtt{T}_{ES}$ 224
$(\cdot)^{\circ}$ Translation from $\mathtt{T}_{ES}$ to $\mathtt{T}_J$ 126
$(\cdot)^{\bullet}$ New translation from $\mathtt{T}_{ES}$ to $\mathtt{T}_J$ 176
$(\cdot)^{\square}$ New translation from $\mathtt{T}_J$ to $\mathtt{T}_\Lambda$ 228
$(\cdot)^{\dagger}$ Composition of $(\cdot)^{\star}$ with $(\cdot)^{\circ}$ 232

Reduction

$\mapsto_{\mathrm{r}}$ Root reduction rule r 47
$\to_{\mathscr{R}}$ Reduction step in relation $\mathscr{R}$ 47
$\to^{+}_{\mathscr{R}}$ Transitive closure of $\to_{\mathscr{R}}$ 47
$\to^{*}_{\mathscr{R}}$ Transitive and reflexive closure of $\to_{\mathscr{R}}$ 47
$\Downarrow^{\theta}$ Big-steps θ-skeleton extraction 85
$\Rightarrow$ Parallel reduction 197

$\mapsto_{\beta}$ Call-by-name rule $(\lambda, \Lambda J)$ 47, 127
$\mapsto_{\beta\mathrm{v}}$ Call-by-value rule $(\lambda^{\sigma}_{\mathrm{v}}, \Lambda J_v)$ 22, 128
$\mapsto_{\mathrm{B}}$ $(\lambda x.M)N \mapsto M[x/N]$ 20
$\mapsto_{\mathrm{dB}}$ Distant B-rule $(\lambda ES, \lambda R)$ 48, 59
$\mapsto_{\mathrm{sub}}$ $M[x/N] \mapsto M\{x/N\}$ 48
$\mapsto_{\mathrm{var}}$ Replication of a variable node 59
$\mapsto_{\mathrm{app}}$ Replication of an application node 59
$\mapsto_{\mathrm{dist}}$ Creation of a distributor 59
$\mapsto_{\mathrm{abs}}$ Replication of the content of a distributor 59
$\mapsto_{\rho}$ Permutation rules in λR 58
$\mapsto_{\mathrm{spl}}$ Skeleton extraction in the fully lazy strategy 91
$\mapsto_{\mathrm{sub}}$ Duplication of the skeleton in the fully lazy strategy 91
$\mapsto_{\pi}$ Original permutation for generalized applications 127
$\mapsto_{\mathrm{p2}}$ CbN permutation for generalized applications 130
$\mapsto_{\beta\mathrm{h}}$ Restriction of rule β for head reduction (ΛJ) 141
$\mapsto_{\pi\mathrm{h}}$ Restriction of rule π for head reduction (ΛJ) 141
$\mapsto_{\mathrm{d}\beta}$ Distant β-rule for generalized applications (λJ_n) 129
$\mapsto_{\mathrm{d}\beta\mathrm{h}}$ Restriction of rule dβ for head reduction (λJ_n) 132

$\mapsto_{d\beta_v}$ Distant βv-rule for generalized applications (λJ_v) 129

$\longrightarrow_{whr}$ Weak-head reduction (λ, λR) 47, 83
$\longrightarrow_{whes}$ Weak-head reduction (λES) 53
$\longrightarrow_{h}$ Head reduction (λ) 47
$\longrightarrow_{hes}$ Head reduction (λES) 49
$\longrightarrow_{es}$ Full reduction (λES) 54
$\longrightarrow_{R}$ Full reduction (λR) 59
$\longrightarrow_{sub}$ Substitution rules of the fully lazy reduction (λR) 60
$\longrightarrow_{name}$ Weak-head CbN reduction (λR) 79
$\longrightarrow_{ndB}$ Weak-head dB (λR) 79
$\longrightarrow_{nsub}$ Weak-head CbN substitutions (λR) 79
$\longrightarrow_{st}$ Small-steps skeleton extraction 87
$\longrightarrow_{flneed}$ Fully lazy CbNeed strategy 91
$\longrightarrow_{djn}$ Full distant CbN reduction (λJ_n) 129
$\longrightarrow_{jn}$ Full CbN reduction (ΛJ) 128
$\longrightarrow_{djv}$ Full distant CbV reduction (λJ_v) 129
$\longrightarrow_{jv}$ Full CbV reduction (ΛJ_v) 128
$\longrightarrow_{sn}$ Solving distant CbN reduction (λJ_n) 132
$\longrightarrow_{lsn}$ Solving CbN reduction (ΛJ) 166
$\longrightarrow_{sv}$ Solving distant CbV reduction (λJ_v) 152
$\longrightarrow_{lsv}$ Solving CbV reduction (ΛJ_v) 166
$\longrightarrow_{ev}$ Distant CbV evaluation (λJ_v) 149
$\longrightarrow_{lev}$ CbV evaluation (ΛJ_v) 166
$\longrightarrow_{lr}$ Left-right distant CbN reduction (λJ_n) 201
$\longrightarrow_{lov}$ Distant CbV leftmost-outermost reduction (λJ_v) 183
$\longrightarrow_{llov}$ CbV leftmost-outermost reduction (ΛJ_v) 187
$\longrightarrow_{o}$ CbV evaluation ($\lambda_{\mathtt{vsub}}$) 175
$\longrightarrow_{s}$ CbV solving reduction ($\lambda_{\mathtt{vsub}}$) 175
$\longrightarrow_{vsub}$ CbV reduction ($\lambda_{\mathtt{vsub}}$) 175

$=_\alpha$ Alpha-equivalence 46
$\sim_\pi$ $t_1(u_1, z_1.t_2)(u_2, z_2.r) \sim t_1(u_1, z_1.t_2(u_2, z_2.r))$ 178
$\sim_{arg}$ $t_2(t_1(u_1, z_1.u_2), z_2.r) \sim t_1(u_1, z_1.t_2(u_2, z_2.r))$ 178
$\sim_{com}$ $t_2(u_2, z_2.t_1(u_1, z_1.r)) \sim t_1(u_1, z_1.t_2(u_2, z_2.r))$ 178
$\equiv_{jv}$ Strong bisimulation on λJ_v 178
$\equiv^1_{jv}$ Contextual closure of the equivalence rules 178
$\longrightarrow_{djv/\equiv_{com}}$ Reduction in λJ_v modulo $\equiv_{com}$ 177
$\longrightarrow_{djv/\equiv_{jv}}$ Reduction in λJ_v modulo $\equiv_{jv}$ 177
$=_{jv}$ Equational theory of λJ_v with structural equivalence 181
$=_{vsub}$ Equational theory of $\lambda_{\mathtt{vsub}}$ with structural equivalence 181

Typing