Normalisation & Equivalence in Proof Theory & Type Theory

Stéphane Lengrand

Université Paris VII — Denis Diderot
University of St Andrews

Normalisation & Équivalence en Théorie de la Démonstration & Théorie des Types

Stéphane Lengrand

Thèse de Doctorat
Spécialité Informatique

Dirigée par :

Pr. Delia Kesner, Paris VII
Dr Roy Dyckhoff, St Andrews

Soutenue publiquement le 8 Décembre 2006 devant le jury composé de :

Dr Pierre-Louis Curien,	Président
Dr James McKinna,	Examinateur interne de St Andrews
Pr. Gilles Dowek,	Rapporteur pour Paris VII
Pr. Luke Ong,	Rapporteur pour St Andrews
Pr. Henk Barendregt	Examinateur
Pr. Dale Miller	Examinateur

& des directeurs de thèse

Résumé

Au coeur des liens entre Théorie de la Démonstration et Théorie des Types, la correspondance de Curry-Howard fournit des termes de preuves aux aspects calculatoires et équipés de théories équationnelles, i.e. des notions de normalisation et d'équivalence. Cette thèse contribue à étendre son cadre à des formalismes (comme le calcul des séquents) appropriés à des considérations d'ordre logique comme la recherche de preuve, à des systèmes expressifs dépassant la logique propositionnelle comme des théories des types, et aux raisonnements classiques plutôt qu'intuitionistes.

La première partie est intitulée **Termes de Preuve pour la Logique Intuitioniste Implicationnelle**, avec des contributions en déduction naturelle et calcul des séquents, normalisation et élimination des coupures, sémantiques en appel par nom et par valeur. En particulier elle introduit des calculs de termes de preuve pour le calcul des séquents depth-bounded G4 et la déduction naturelle multiplicative. Cette dernière donne lieu à un calcul de substitutions explicites avec affaiblissements et contractions, qui raffine la β-réduction.

La deuxième partie, intitulée **Théorie des Types en Calcul des Séquents**, développe une théorie des Pure Type Sequent Calculi, équivalents aux Systèmes de Types Purs mais mieux adaptés à la recherche de preuve.

La troisième partie, intitulée **Vers la Logique Classique**, étudie des approches à la Théorie des Types classique. Elle développe un calcul des séquents pour une version classique du Système F_ω. Une approche à la question de l'équivalence de preuves classiques consiste à calculer les représentants canoniques de preuves équivalentes dans le cadre du Calcul des Structures.

Synopsis

Au coeur des liens entre Théorie de la Démonstration et Théorie des Types se trouve sans aucun doute la *correspondance de Curry-Howard* [How80]. Quels avantages la Logique tire de ces liens n'est pas la question qui fait l'objet de la présente thèse, dans laquelle on considère comme acquis le fait qu'il soit intellectuelement désirable d'avoir en Logique:

- des objets mathématiques qui formalisent la notion de démonstration,
- des notions de *calcul* et de *normalisation* de ces objets,
- des théories équationnelles sur ces objets, c'est-à-dire des notions d'*équivalence*, en relation avec les notions de calcul.

La correspondance de Curry-Howard fournit de tels concepts à la Logique en connectant démonstrations et programmes d'une part, et proposition et types d'autre part. Alors que son cadre originel était la *déduction naturelle intuitionniste* [Gen35] (ou des systèmes à la Hilbert), les questions abordées par la présente thèse se positionnent dans un cadre plus large, mais en mettant toujours l'accent sur les notions d'équivalence et de normalisation. Plus précisément, elle contribue à étendre le cadre des concepts fournis par la correspondance de Curry-Howard:

- à des formalismes en théorie de la démonstration (tels que le *calcul des séquents* [Gen35]) qui sont appropriés à des considérations d'ordre logique comme la *recherche de démonstration* (aussi appelée *recherche de preuve*),
- à des systèmes expressifs dépassant la logique propositionnelle tels que des *théories des types*,
- aux raisonnements *classiques* plutôt qu'intuitionistes.

Les trois parties de cette thèse reflètent ces directions.

La première partie est intitulée **Termes de Preuve pour la Logique Intuitioniste Implicationnelle**; elle reste dans le cadre de la logique propositionnelle intuitioniste (avec l'implication comme seul connecteur logique) mais elle étudie les notions de la correspondance de Curry-Howard non seulement en déduction naturelle mais aussi en calcul des séquents. Elle présente ainsi les *termes* (aussi

appelés *termes de preuve*) avec lesquels sont représentés les démonstrations des formalismes sus-mentionnés, ainsi que les avantages de l'utilisation, en déduction naturelle, de règles issues du calcul des séquents, et enfin les notions de calcul dans ce dernier formalisme et leurs liens avec les sémantiques d'*appel par nom* et d'*appel par valeur* [Plo75].

La seconde partie est intitulée **Théorie des Types en Calcul des Séquents**; elle reste dans le cadre de la logique intuitioniste et construit une théorie qui est au calcul des séquents ce que les *Systèmes de Types Purs* [Bar92] sont à la déduction naturelle. Au delà des propriétés fondamentales de cette théorie sont aussi étudiées des questions telles que la recherche de preuve et l'*inférence de type*.

La troisième partie est intitulée **Vers la logique classique** et est motivée par l'objectif de construire en logique classique des théories similaires à celles dévelopées dans la seconde partie en logique intuitioniste. En particulier, cette partie dévelope un calcul des séquent correspondant au *Système* F_ω [Gir72] mais dont la couche des démonstrations est classique. Au delà d'une telle théorie des types, la notion d'*équivalence des démonstrations classiques* devient cruciale et la partie est conclue par une approche à ce problème.

La présente thèse aborde donc un grand nombre de questions, pour lesquelles un cadre et une terminologie unifiés, les couvrant toutes, sont à la fois importants et non-triviaux:

Chapitre 1

Ce chapitre présente les concepts et la terminologie utilisés dans le reste de la thèse. Il présente tout d'abord les notions et notations concernant relations et fonctions, comme les notions de *(relation de) réduction*, *forte et faible simulation*, *correspondance équationnelle*, *réflection*, *confluence*, *dérivation dans une structure d'inférence*...

Puis une partie importante est consacrée à la forte et la faible simulation ainsi qu'aux techniques fondamentales pour prouver ces propriétés. Une considération omniprésente dans cette partie est l'approche intuitioniste des résultats établis, qui évite les raisonnements classiques commençant par "Supposons qu'il existe une séquence de réduction infinie" et aboutissant à une contradiction. Par exemple la technique de simulation est établie de manière intuitioniste, ainsi que les résultats de forte normalisation des réductions *lexicographiques* et des réductions de *multi-ensembles*.

Une autre partie importante de ce chapitre est consacrée aux *calculs d'ordre supérieur* (HOC). En effet, les outils de la correspondance de Curry-Howard, dans son cadre originel ou dans ceux abordés dans cette thèse, sont fondés sur des syntaxes de termes dont les *constructeurs* impliquent des *liaisons de variables* et dont les relations de réduction sont définies par *systèmes de réécriture* [Ter03] (que nous présentons avec les notions associées de *redex*, *clôture contextuelle*, *variable libre et muette*, *substitution*,...). De tels outils nécessitent un prudent

traitement de l'α-*equivalence* (le fait que le choix d'une variable muette n'est pas d'importance), en utilisant habituellement des conditions qui évitent la *capture* et la *libération de variables*. Dans la présente thèse nous écrirons rarement ces conditions, précisément parce qu'elles peuvent être retrouvée mécaniquement d'après le contexte où elle sont nécéssitées. Ce chapitre décrit comment.

Partie I

Chapitre 2

Ce chapitre présente la déduction naturelle, le calcul des séquents [Gen35] avec (ou sans) sans règle de *coupure*, le λ-calcul [Chu41] avec sa notion de réduction appelée β-*reduction*, ainsi que la correspondance de Curry-Howard [How80]. Pour cela, ce chapitre prolonge le chapitre 1 avec les concepts de *sequent*, *système logique*, *système de typage*, *terme de preuve*, et la propriété de *subject reduction*, utilisés non seulement dans la partie I mais aussi dans le chapitre 10. Ce chapitre est conclu par un calcul d'ordre supérieur pour représenter les démonstrations du calcul des séquents intuitioniste $\mathsf{G3ii}$, utilisé par exemple dans [DP99b]. Est remarqué le fait que le constructeur typé par une coupure est de la forme d'une *substitution explicite* [ACCL91, BR95]. Les encodages de Gentzen et Prawitz [Gen35, Pra65] entre la déduction naturelle et le calcul des séquents sont exprimés ici comme des traductions de termes de preuves qui préservent le typage.

Chapitre 3

Ce chapitre étudie le λ-calculus en *appel par valeur* (CBV) et part du calcul λ_{V} [Plo75], qui restreint la β-reduction aux cas β_{V} où l'argument d'une fonction est déjà évalué comme *valeur*. On présente ensuite la sémantique CBV grâce à deux variantes de traduction en *Continuation Passing Style* (CPS) dans des fragments du λ-calcul : celle de Reynolds [Rey72, Rey98] et celle de Fischer [Fis72, Fis93]. On présente le calcul λ_{C} de Moggi qui étend λ_{V} [Mog88] d'une manière telle que les traductions CPS deviennent des correspondances équationnelles [SF93, SW97] : l'équivalence entre termes de λ_{C} générée par leurs réductions correspond à l'équivalence entre leurs encodages générée par la β-reduction. Dans [SW97], ceci est renforcé par le fait qu'un raffinement de la traduction de Reynolds forme même une réflection. On prouve ici que c'est aussi le cas pour la traduction de Fischer (si une modification mineure et naturelle de λ_{C} est faite), and l'on déduit de cette réflection la confluence de λ_{C} (modifié). La modification et la réflection aide aussi à établir un lien avec le calcul des séquents LJQ présenté au chapitre 6.

Chapitre 4

Dans ce chapitre sont présentées deux techniques (qui peuvent être combinées) pour prouver des propriétés de forte normalisation, en particulier des propriétés de calculs proches du λ-calcul telles que la *préservation de la forte normalisation* (PSN) [BBLRD96]. Ces deux techniques sont des raffinements de la technique de simulation du chapitre 1.

Le point de départ de la première technique, appelée *safeness and minimality technique*, est le fait que, pour prouver la forte normalisation, seules les réductions de rédexes dont les sous-termes sont fortement normalisables ont besoin d'être considérées (*minimality*). Ensuite, la notion de *safeness* fournit un critère utile (selon que le rédex à réduire est fortement normalisable ou non) pour séparer les réductions minimales en deux relations de réduction, dont la forte normalisation peut être alors prouvée par composition lexicographique. L'exemple du calcul de substitution explicites λx [BR95] illustre la technique, par des preuves courtes de PSN, et de forte normalisation des termes simplement typés ou typés avec des *types intersection* [CD78, LLD$^+$04].

La seconde technique fournit des outils pour prouver la forte normalisation d'un calcul lié au λ-calcul, par simulation dans le λI-calcul de [Klo80] (fondé sur les travaux antérieurs de [Chu41, Ned73]), si une simulation directe dans le λ-calcul échoue. On démontre ainsi la propriété PSN pour λI. Cette seconde technique est illustrée dans le chapitre 5.

Chapitre 5

Dans ce chapitre on présente un calcul d'ordre supérieur appelé λlxr dont la version typée correspond, par la correspondance de Curry-Howard, à une version *multiplicative* de la déduction naturelle intuitioniste. Celle-ci utilise des *affaiblissements*, des *contractions* and des *coupures*, habituels en calcul des séquents (par exemple dans la version original de [Gen35]). Les constructeurs typés par les règles sus-mentionnées peuvent être vus comme des constructeurs de ressources gérant l'*effacement* et la *duplication* de substitutions explicites.

On décrit le comportement opérationnel de λlxr et ses propriétés fondamentales, pour lesquels une notion d'équivalence sur les termes joue un rôle essentiel, en particulier dans les réductions du calcul. Cette équivalence rapproche λlxr des *réseaux de preuve* pour (le fragment intuitioniste de) la logique linéaire [Gir87] (en fait [KL05, KL06] révèle une correspondance), mais elle est aussi nécessaire pour que λlxr simule la β-réduction. Dans ce chapitre est établie une réflection du λ-calculus dans λlxr, qui inclut la forte simulation de la β-reduction et entraîne la confluence de λlxr. En utilisant la seconde technique dévelopée au chapitre 4, on prouve aussi PSN et la forte normalisation des termes typés. λlxr est un calcul de substitutions explicites qui a une notion de *full composition* et la propriété PSN.

Chapitre 6

Dans ce chapitre est étudiée la notion de calcul dans le calcul des séquents intuitioniste propositionnel —ici G3ii, fondée sur l'*élimination des coupures*. On passe en revue trois types de systèmes d'élimination des coupures, tous fortement normalisables sur les termes de preuve (typés), et l'on identifie une structure commune d'où sont définies les notions de réduction en *appel par nom* (CBN) et en *appel par valeur* (CBV). On rappelle la t-restriction et la q-restriction dans le calcul des séquents [DJS95, Her95] qui, dans le cas intuitioniste, mènent aux fragments LJT et LJQ. Ceux-ci sont respectivement *stables par* réductions CBN et CBV. On formalise par une réflection le lien entre LJT (et son calcul $\overline{\lambda}$ de termes de preuve) et la déduction naturelle (et le λ-calcul). On prouve aussi PSN pour $\overline{\lambda}$, illustrant à nouveau la méthode de *safeness and minimality* dévelopée au Chapitre 4. Le fragment LJQ est quant à lui connecté à la version du calcul CBV λ_{C} [Mog88] présentée au chapitre 3.

Chapitre 7

Dans ce chapitre est appliquée la méthodologie des chapitre 2 et 6 (pour G3ii) au calcul des séquents intuitioniste *depth-bounded* G4ii de [Hud89, Hud92, Dyc92]. On montre dans un premier temps comment l'on passe de LJQ à G4ii. Un calcul de termes de preuve est ensuite présenté, utilisant des constructeurs correspondant aux règles d'inférence admissibles dans le système (comme la règle de coupure). Alors que les démonstrations traditionnelle d'admissibilité par induction suggèrent des transformations de démonstration faiblement normalisables, on renforce ces approches en introduisant divers systèmes de réduction de terme de preuve qui sont fortement normalisables. Les diverses variantes correspondent à différentes optimisations, dont certaines sont *orthogonales* comme les sous-systèmes CBN et CBV similaires à ceux de G3ii. Nous remarquons toutefois que le sous-système CBV est plus naturel que celui CBN, ce qui est lié au fait que G4ii est fondé sur LJQ.

Partie II

Chapitre 8

Fondés sur la déduction naturelle, les *Systèmes de types purs* (PTS) [Bar92] peuvent exprimer de nombreuses théories des types. Pour exprimer la recherche de preuve dans de telles théories, on présente dans ce chapitre les *Pure Type Sequent Calculi* (PTSC) en enrichissant LJT et le $\overline{\lambda}$-calculus [Her95], presentés au chapitre 6, car ils sont particulièrement adaptés à la recherche de preuve et fortement liés à la déduction naturelle et au λ-calcul.

Les PTSC sont équippés d'une procédure de normalisation, adaptant celle de $\overline{\lambda}$ et donc définie par des règles de réécriture *locales* en utilisant des substitutions explicites. La propriété de *subject reduction* est démontrée et la réflection dans $\overline{\lambda}$ du λ-calcul est adaptée en une réflection dans les PTSC des PTS. De plus, le fait que la réflection préserve le typage montre qu'un PTSC est logiquement équivalent au PTS correspondant. On démontre aussi que le premier est fortement normalisable si et seulement si le deuxième l'est aussi.

Chapitre 9

Dans se chapitre sont étudiées des variantes des PTSC, essentiellement développées pour la recherche de preuve et l'inférence de type.

On montre comment les *règles de conversion* peuvent être incorporées aux autres règles pour que les tactiques basiques de recherche de preuve soient simplement l'application de bas en haut des règles d'inférence. Des *variables d'ordre supérieur* (qui peuvent être vues comme des *méta-variables*) sont alors ajoutées au formalisme ainsi que des *contraintes d'unification*, afin d'ajourner la résolution de *sous-buts* dans la recherche de preuve et d'exprimer des algorithmes d'énumération d'*habitants de types* similaire à ceux de [Dow93, Mun01].

On montre aussi comment les règles de conversion peuvent être incorporées aux autres règles pour que l'inférence de type soit l'application de bas en haut des règles d'inférence, d'un manière similaire au *Constructive engine* de la déduction naturelle [Hue89, vBJMP94]. Pour cette section il est nécessaire d'introduire une version des PTSC aux substitutions implicites, car l'inférence de type échoue avec nos règles de typage pour les substitutions explicites.

Cette version avec substitutions implicites peut être aussi plus facilement transformée en une version avec *indices de de Bruijn*, ce qui est fait dans la dernière partie du chapitre, dans le style de [KR02].

Partie III

Chapitre 10

Dans ce chapitre on présente un système appelé $F_\omega^{\mathcal{C}}$, une version du *Système* F_ω [Gir72] dans laquelle la couche des *constructeurs de types* est essentiellement la même mais la prouvabilité des types est classique. Le calcul des termes de preuve qui rend compte du raisonnement classique est une variante du λ-calcul symétrique de [BB96].

On prouve que l'ensemble du calcul est fortement normalisable. Pour la couche des constructeurs de types, on utilise la méthode de réductibilité de Tait et Girard, combinée à des techniques d'*orthogonalité*. Pour la couche (classique) des termes de preuve, on utilise la méthode de [BB96] fondée sur une *notion symétrique de*

candidats de réductibilité. Le système $F^{\mathcal{C}}_{\omega}$, avec ses deux couches de différente nature, est ainsi une opportunité de comparer les deux techniques et poser la conjecture que la deuxième technique ne peut pas être subsumée par la première.

Nous concluons avec une preuve de cohérence de $F^{\mathcal{C}}_{\omega}$, et un encodage du traditionel Système F_{ω} dans $F^{\mathcal{C}}_{\omega}$, même lorsque le premier utilise des axiomes supplémentaires de la logique classique.

Chapitre 11

Tenter d'introduire des raisonnements classiques plus loin dans les théories des types dévelopées en seconde partie (par exemple avec des *types dépendants*) se heurte au problème de la définition d'une notion appropriée d'équivalence de démonstrations classiques. Dans ce chapitre une approche originale est suggérée, dans le cadre de la logique propositionnelle classique, et fondée sur le *Calcul des Structures* [Gug, Gug02]. Les démonstrations sont des séquences de réécriture sur les formules, et nous utilisons la notion de *réduction parallèle* [Tak89] pour identifier différentes séquentialisations d'étapes de réécritures qui réduisent des rédexes qui ne se chevauchent pas. Le cas des rédexes parallèles et celui des rédexes imbriqués donnent lieu à deux notions d'équivalence sur les séquences de réécriture (dans un système de réécriture linéaire qui formalise la logique classique). A partir de ces notions sont présentés deux formalismes qui autorisent les réductions parallèles et ainsi fournissent un représentant canonique à deux démonstrations équivalentes. Ce représentant peut être obtenu d'une démonstration quelconque par une relation de réduction particulière, qui est confluente and qui termine (la confluence dans le cas des rédexes imbriqués n'est que conjecturée). Ces formalismes se révèlent être des calculs des séquents avec axiomes, et leurs termes de preuves utilisent des combinateurs comme pour les systèmes à la Hilbert. La procédure de normalisation qui produit les représentants canoniques se trouve être une procédure de réduction des coupures.

Dépendences entre chapitres

Pour faciliter la lecture de cette thèse et permettre certains chapitres d'être lus indépendemment, le graphe de dépendence des chapitres est présenté dans la Fig. 0, ainsi que les références bibliographiques dans lesquelles leur contenus, ou certaines parties de leurs contenus, sont parus.

Les flèches pleines représentent une réelle dépendence (mais parfois seulement parce qu'un chapitre utilise des définitions données dans un chapitre antérieur mais familières au lecteur) ; les flèches en pointillés ne représentent qu'un lien plus faible sans faire d'un chapitre le préalable d'un autre.

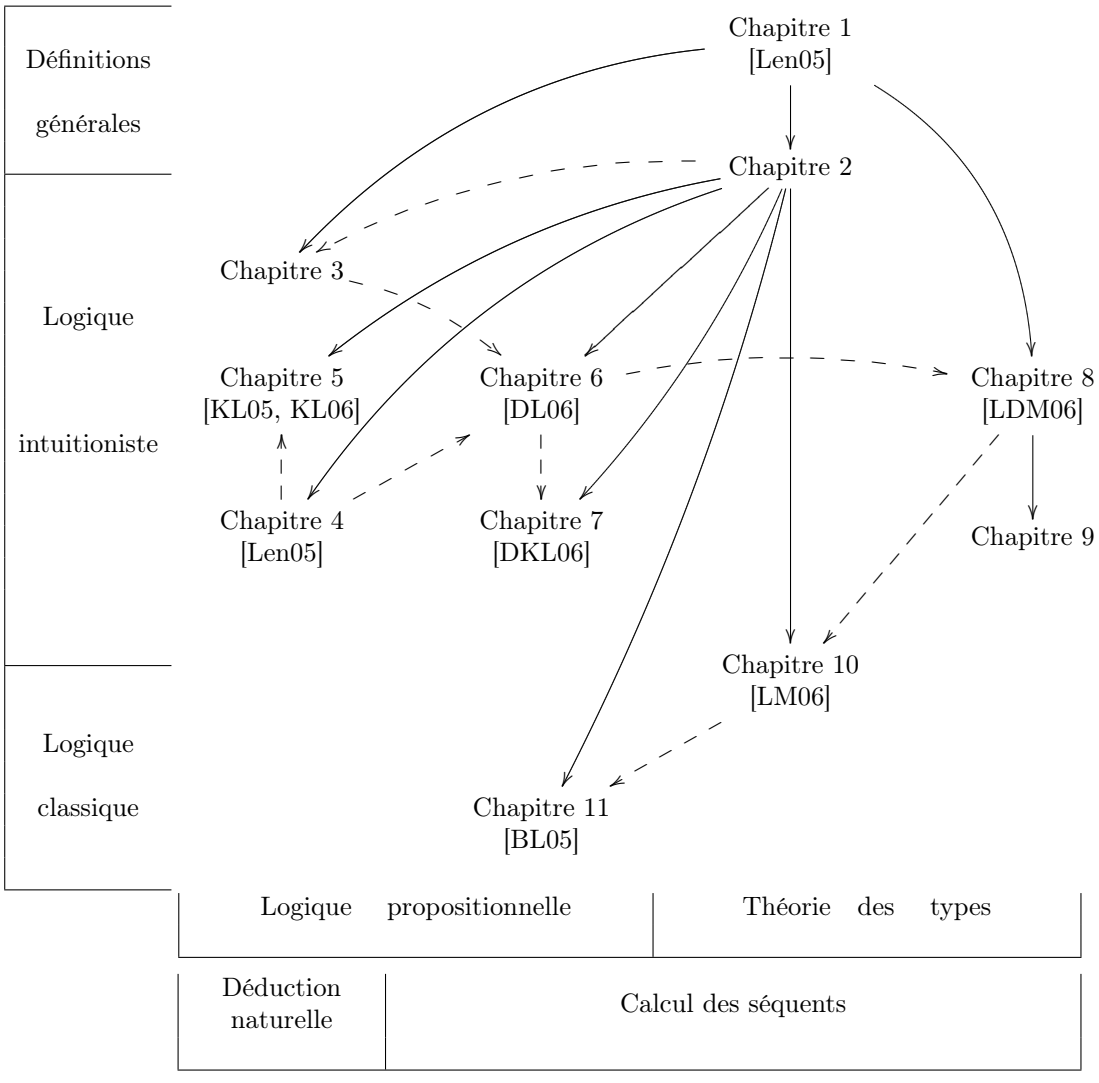


Fig. 0: Graphe de dépendence

Normalisation & Equivalence in Proof Theory & Type Theory

Stéphane Lengrand

Dissertation submitted towards the degree of

Doctor of Philosophy

Université Paris VII — Denis Diderot

University of St Andrews

Advisers:

Pr. Delia Kesner, Paris VII
Dr Roy Dyckhoff, St Andrews

Thesis publicly defended on Friday 8TH December 2006 before

Dr Pierre-Louis Curien, Chairman
Dr James McKinna, Internal examiner of St Andrews

Pr. Gilles Dowek, Referee for Paris VII
Pr. Luke Ong, Referee for St Andrews

Pr. Henk Barendregt Examiner
Pr. Dale Miller Examiner

& the advisers

Abstract

At the heart of the connections between Proof Theory and Type Theory, the Curry-Howard correspondence provides proof-terms with computational features and equational theories, i.e. notions of normalisation and equivalence. This dissertation contributes to extend its framework in the directions of proof-theoretic formalisms (such as sequent calculus) that are appealing for logical purposes like proof-search, powerful systems beyond propositional logic such as type theories, and classical (rather than intuitionistic) reasoning.

Part I is entitled **Proof-terms for Intuitionistic Implicational Logic**. Its contributions use rewriting techniques on proof-terms for natural deduction (λ-calculus) and sequent calculus, and investigate normalisation and cut-elimination, with call-by-name and call-by-value semantics. In particular, it introduces proof-term calculi for multiplicative natural deduction and for the depth-bounded sequent calculus G4. The former gives rise to the calculus λlxr with explicit substitutions, weakenings and contractions that refines the λ-calculus and β-reduction, and preserves strong normalisation with a full notion of composition of substitutions. The latter gives a new insight to cut-elimination in G4.

Part II, entitled **Type Theory in Sequent Calculus** develops a theory of Pure Type Sequent Calculi (PTSC), which are sequent calculi that are equivalent (with respect to provability and normalisation) to Pure Type Systems but better suited for proof-search, in connection with proof-assistant tactics and proof-term enumeration algorithms.

Part III, entitled **Towards Classical Logic**, presents some approaches to classical type theory. In particular it develops a sequent calculus for a classical version of System F_ω. Beyond such a type theory, the notion of equivalence of classical proofs becomes crucial and, with such a notion based on parallel rewriting in the Calculus of Structures, we compute canonical representatives of equivalent proofs.

Acknowledgements

It goes without saying that my greatest debt of gratitude is owed to my advisers Delia Kesner and Roy Dyckhoff for all the time and energy that they have invested in my thesis. Not only was I lucky to grow up in a biparental academic family, but both of them provided the support and patience that a Ph.D. student could hope for from a single adviser; the joint supervision was thus all the more fruitful, eventually leading to [KL05, KL06, DL06, DKL06] and this dissertation.

Whilst they had a central role in making this *cotutelle* rewarding, it would not have been possible without the initial efforts of Jacques Chevalier and the French Embassy in London, whose plans met my Auld Alliance project at the right time. With them I wish to thank Frank Riddell in St Andrews and other people from each university whom I have not met, for pushing the *cotutelle* agreement through many bureaucratic obstacles.

I am very grateful to Gilles Dowek and Luke Ong for agreeing to referee my thesis, particularly in light of the work and time requisite in reading the 349 pages of this unexpectedly lengthy dissertation. I also wish to thank Pierre-Louis Curien, Henk Barendregt, and Dale Miller for having accepted to sit on my examination panel, and thus honoured me with their interest and time.

I thank James McKinna for the above reasons, and also for his role during my time in St Andrews. Owing to his pioneering some of the ideas that Part II investigates, his unique insight consolidated the progress I made. Part II thus benefitted from numerous discussions with him, to the extent that his role therein was that of an additional adviser, eventually leading to [LDM06]. I was met with equal friendliness working with my other co-authors, such as Alexandre Miquel and Kai Brünnler with whom joint work [LM06, BL05] was the origin of Part III. They share a genuine and open-minded scientific curiosity that supported my work. Alexandre's grasp of Type Theory, pedagogical skills, patience, and frequent after-dark presence at PPS made him a point of reference in the field to which I regularly turn to. I appreciate in Kai his analytical mind and methodological approach to research, always having astute questions and concerns; a memorable experience of *skihock* during a delightful wintery week-end in Bern reminds me that his energy and enthusiasm to explore new and exotic ideas go beyond academia.

I am also extremely grateful to Hugo Herbelin —on the work of whom most of my thesis is based, Kentaro Kikuchi and José Espírito Santo for their challenging ideas and feedback, countless enlightening discussions and shared interests, which, I hope, will lead to future work together.

Among those many people with whom scientific discussions helped me make progress in my thesis are Alessio Guglielmi, Claude Kirchner, Alex Simpson, Christian Urban, Emmanuel Polonovski, and Zhaohui Luo, who were so kind as to invite me to give a talk and/or go for a pint.

To conclude the scientific acknowledgements, I thank Pierre Lescanne and his teachings that have ushered me to my field of research and its communities, and anonymous referees for their helpful feedback and constructive criticism.

I am particularly grateful to my mother, for having heavily edited the latex source of Chapter 10 with me dictating over the phone the day before a deadline, and my father; their experience as academics is and has consistently been of great value for me. Greatly appreciated was the friendly atmosphere of my offices in Paris and St Andrews, mostly due to their occupants. I wish to thank the secretaries of both departments for their patience and efforts, I know that I have been much of a complicated case at times.

Finally, I thank those who put up with me in times of hardship, and sincerely apologise for my absence or silence which these busy times of writing have caused. For the close person amongst them who endured this most I have a very special thought.

Thank you.

à Rémi,

Table of Contents

Introduction

At the heart of the connections between Proof Theory and Type Theory undoubtedly lies the *Curry-Howard correspondence* [How80]. What Logic gains from these connections is not the issue that we take as the purpose of this dissertation, in which we rather take for granted the fact that are intellectually appealing such concepts in Logic as:

- mathematical objects that can formalise the notion of proof,
- computational features of these objects with notions of *normalisation*,
- equational theories about these objects, that is to say, notions of *equivalence*, that are related to their computational features.

The Curry-Howard correspondence provides such concepts to Logic by relating proofs to programs and propositions to types, so that insight into one aspect helps the understanding of the other. While its original framework was *intuitionistic propositional natural deduction* [Gen35] (or *Hilbert-style systems*), this dissertation investigates some issues pertaining to a more general framework, with a particular emphasis on the notions of equivalence and normalisation. More precisely, it contributes to broaden the scope of the concepts provided by the Curry-Howard correspondence in three directions:

- proof-theoretic formalisms (such as *sequent calculus* [Gen35]) that are appealing for logical purposes such as *proof-search*,
- powerful systems beyond propositional logic such as *type theories*,
- *classical reasoning* rather than intuitionistic reasoning.

The three parts of this dissertation reflect these directions.

Part I is entitled **Proof-terms for Intuitionistic Implicational Logic**; it remains within the framework of propositional intuitionistic logic (with implication as the only connective) but investigates the notions given by the Curry-Howard correspondence in natural deduction as well as in sequent calculus. It addresses such topics as the *terms* (a.k.a. *proof-terms*) with which the proofs of these formalisms are represented, the benefits of using in natural deduction some

rules of sequent calculus, the notions of computation in sequent calculus and their relation with *call-by-name* and *call-by-value* semantics [Plo75].

Part II is entitled **Type Theory in Sequent Calculus**; it remains in the framework of intuitionistic logic and builds a theory that is to sequent calculus what *Pure Type Systems* [Bar92] are to natural deduction. Beyond the basic properties of the theory, further aspects are developed, such as proof-search and *type inference*.

Part III is entitled **Towards Classical Logic** and is motivated by the purpose of building theories in classical logic such as those of Part II in intuitionistic logic. In particular it develops a sequent calculus corresponding to *System* F_ω [Gir72] but whose layer of proofs is classical. Beyond such a type theory, the notion of *equivalence of classical proofs* becomes crucial and the part concludes with an approach to this issue.

This dissertation thus addresses a wide range of topics, for which a unified framework and terminology, covering all of them, are thus both important and non-trivial:

Chapter 1

This chapter introduces the concepts and the terminology that are used throughout the three parts of the dissertation. It first introduces all notions and notations about relations and functions, including *reduction relation*, *strong and weak simulation*, *equational correspondence*, *reflection*, *confluence*, *derivation in an inference structure*...

Then a major section is devoted to the notions of (weak and strong) normalisation and basic techniques to prove these properties. A particular concern of this section is to develop these ideas in a constructive setting, avoiding the usual reasonings starting with "Let us assume that there is an infinite reduction sequence,...", used especially when the strong normalisation of a reduction relation is inferred, by simulation, from that of another one that we already know to be strongly normalising. We thus prove results about the strong normalisation of the *lexicographic composition of relations* and the *multi-set reduction relation*.

Another major section of this chapter is devoted to *Higher-Order Calculi*. Indeed, the tools for the Curry-Howard correspondence, whether in its original setting or in the various ones tackled here, are based on syntaxes of terms involving *variable binding*, with reduction relations given by *rewrite systems* [Ter03] which we present together with the notions, traditional in rewriting, of *redex*, *contextual closure*, *free variables*, *substitution*,... Such tools require a careful treatment of α*-equivalence* (the fact that the choice of a variable that is bound is irrelevant), usually using conditions to avoid *variable capture* and *liberation*. In this dissertation we deliberately not write these conditions, precisely because they can be recovered mechanically from the context in which they are needed. This chapter explains how.

Part I

Chapter 2

This chapter introduces natural deduction, sequent calculus [Gen35] with (or without) its *cut*-rule, λ-calculus [Chu41] with its notion of reduction called β-*reduction*, and the Curry-Howard correspondence [How80]. For that it extends Chapter 1 with such general concepts as *sequents*, *logical systems*, *typing systems*, *proof-terms*, and *subject reduction property*, which will not only be used throughout Part I but also in Chapter 10. It concludes by presenting an higher-order calculus to represent proofs of the intuitionistic sequent calculus $\mathsf{G3ii}$, as used for instance in [DP99b]. It notes the fact that the constructor typed by a cut has the shape of an *explicit substitution* [ACCL91, BR95]. Gentzen's and Prawitz's encodings [Gen35, Pra65] between natural deduction and sequent calculus are expressed here as type-preserving translations of proof-terms.

Chapter 3

In this chapter we investigate the *call-by-value* (CBV) λ-calculus. We start from the calculus λ_{V} [Plo75], which merely restricts β-reduction to the case β_{V} where arguments of functions are already evaluated as *values*. We then present the CBV semantics given by *Continuation Passing Style* (CPS) translations (into fragments of λ-calculus), in two variants: Reynolds' [Rey72, Rey98] and Fischer's [Fis72, Fis93]. We present Moggi's λ_{C}-calculus [Mog88] that extends λ_{V} in a way such that the CPS-translations become equational correspondences [SF93, SW97]. In other words, the equivalence on λ_{C}-terms generated by their reductions matches the equivalence between their encodings given by β-reduction. In [SW97] the result is stronger in that (a refinement of) the Reynolds translation even forms a reflection. In this chapter we prove that it is also the case for the Fischer translation (if we make a minor and natural modification to λ_{C}), and we infer from our reflection the confluence of (this modified) λ_{C}. The modification and the reflection also help establishing a connection with a sequent calculus called LJQ and presented in Chapter 6.

Chapter 4

In this chapter we present two techniques, which can be combined, to prove strong normalisation properties, especially properties of calculi related to λ-calculus such as *Preservation of Strong Normalisation* (PSN) [BBLRD96]. They are both refinements of the simulation technique from Chapter 1.

The first technique, called the *safeness and minimality technique*, starts with the fact that, in order to prove strong normalisation, only the reduction of redexes whose sub-terms are strongly normalising need to be looked at (*minimal-*

ity). Then *safeness* provides a useful criterion (reducing redexes that are strongly normalising or that are not) in order to split the (minimal) reductions of a calculus into two reduction relations, which can then be proved strongly normalising by a lexicographic composition. The example of the explicit substitution calculus $\lambda\mathsf{x}$ [BR95] illustrates the technique, with short proofs of PSN, strong normalisation of the simply-typed version and that of its version with *intersection types* [CD78, LLD$^+$04].

The second technique provides tools to prove the strong normalisation of a calculus related to λ-calculus by simulation in the λI-calculus of [Klo80] (based on earlier work by [Chu41, Ned73]), when a direct simulation in λ-calculus fails. Such a tool is the PSN property for λI. This technique is illustrated in Chapter 5.

Chapter 5

In this chapter, we present a higher-order calculus called $\lambda\mathsf{lxr}$ whose typed version corresponds, via the Curry-Howard correspondence, to a *multiplicative* version of intuitionistic natural deduction. The latter uses *weakenings*, *contractions* and *cuts*, which are common in sequent calculus, e.g. in its original version [Gen35]. The constructors typed by the above rules can be seen as *resource constructors* handling *erasure* and *duplication* of explicit substitutions.

We describe the operational behaviour of $\lambda\mathsf{lxr}$ and its fundamental properties, for which a notion of equivalence on terms plays an essential role, in particular in the reduction relation of the calculus. This equivalence brings $\lambda\mathsf{lxr}$ close to the *proof nets* for (the intuitionistic fragment of) linear logic [Gir87] (in fact [KL05, KL06] reveals a sound and complete correspondence), but it is also necessary in order for $\lambda\mathsf{lxr}$ to simulate β-reduction. In this chapter we actually establish a reflection of λ-calculus in $\lambda\mathsf{lxr}$, which includes the strong simulation of β-reduction and entails confluence of $\lambda\mathsf{lxr}$. Using the second technique developed in Chapter 4, we also prove PSN, and strong normalisation of typed terms. $\lambda\mathsf{lxr}$ is an HOC with explicit substitutions having *full composition* and preserving strong normalisation.

Chapter 6

In this chapter we investigate the notions of computation in intuitionistic propositional sequent calculus —here G3ii, based on *cut-elimination*. We survey three kinds of cut-elimination system, proved or conjectured to be strongly normalising on typed terms, and identify a common structure in them, from which we generically define *call-by-name* (CBN) and *call-by-value* (CBV) reductions. We recall the t- and q-restrictions in sequent calculus [DJS95, Her95] which, in the intuitionistic case, lead to the fragments LJT and LJQ of G3ii. These are *stable under* CBN and CBV reductions, respectively. By means of a reflection we relate

LJT, and its higher-order calculus $\overline{\lambda}$ that provides its proof-terms, to natural deduction and λ-calculus. We also prove PSN of $\overline{\lambda}$ as another illustration of the safeness and minimality technique from Chapter 4. We then relate LJQ and its proof-terms to (the modified version of) the CBV calculus λ_{C} [Mog88] presented in Chapter 3.

Chapter 7

In this chapter we apply the methodology of Chapter 2 and Chapter 6 (for G3ii) to the *depth-bounded* intuitionistic sequent calculus G4ii of [Hud89, Hud92, Dyc92]. We first show how G4ii is obtained from LJQ. We then present a higher-order calculus for it —decorating proofs with proof-terms, which uses constructors corresponding to admissible rules such as the cut-rule. While existing inductive arguments for admissibility suggest weakly normalising proof transformations, we strengthen these approaches by introducing various term-reduction systems, all strongly normalising on typed terms, representing proof transformations. The variations correspond to different optimisations, some of them being *orthogonal* such as CBN and CBV sub-systems similar to those of G3ii. We note however that the CBV sub-system seems more natural than the CBN one, which is related to the fact that G4ii is based on LJQ.

Part II

Chapter 8

Based on natural deduction, *Pure Type Systems* (PTS) [Bar92] can express a wide range of type theories. In order to express proof-search in such theories, we introduce in this chapter the *Pure Type Sequent Calculi* (PTSC) by enriching LJT and the $\overline{\lambda}$-calculus [Her95], presented in Chapter 6, because they are adapted to proof-search and strongly related to natural deduction and λ-calculus.

PTSC are equipped with a normalisation procedure, adapted from that of $\overline{\lambda}$ and defined by local rewrite rules as in cut-elimination, using explicit substitutions. We prove that they satisfy subject reduction and turn the reflection in $\overline{\lambda}$ of λ-calculus into a reflection in PTSC of PTS. Moreover, the fact that the reflection is type-preserving shows that a PTSC is logically equivalent to its corresponding PTS. Then we prove that the former is strongly normalising if and only if the latter is.

Chapter 9

In this chapter we investigate variants of PTSC, mostly designed for proof-search and type inference.

We show how the *conversion rules* can be incorporated into the other rules so that basic proof-search tactics in type theory are merely the root-first application of the inference rules. We then add to this formalism *higher-order variables* (that can be seen as *meta-variables*) and *unification constraints*, in order to delay the resolution of *sub-goals* in proof-search and express type inhabitant enumeration algorithms such as those of [Dow93, Mun01].

We also show how the conversion rules can be incorporated into the other rules so that type inference becomes the root-first application of the inference rules, in a way similar to the *Constructive engine* in natural deduction [Hue89, vBJMP94]. For this section we also need to introduce a version of PTSC with implicit substitutions, since type inference fails on our typing rule for explicit substitutions.

This version with implicit substitutions is also easier to turn into a version with *de Bruijn indices*, which we do in the final part of the chapter, in the style of [KR02].

Part III

Chapter 10

In this chapter we present a system called $F_\omega^{\mathcal{C}}$, a version of *System* F_ω [Gir72] in which the layer of *type constructors* is essentially the same whereas provability of types is classical. The proof-term calculus accounting for the classical reasoning is a variant of Barbanera and Berardi's symmetric λ-calculus [BB96].

We prove that the whole calculus is strongly normalising. For the layer of type constructors, we use Tait and Girard's reducibility method combined with *orthogonality techniques*. For the (classical) layer of terms, we use Barbanera and Berardi's method based on a *symmetric notion of reducibility candidates*. System $F_\omega^{\mathcal{C}}$, with its two layers of different nature, is thus an opportunity to compare the two above techniques and raise the conjecture that the latter cannot be captured by the former.

We conclude with a proof of consistency for $F_\omega^{\mathcal{C}}$, and an encoding from the traditional System F_ω into $F_\omega^{\mathcal{C}}$, also when the former uses extra axioms to allow classical reasonings.

Chapter 11

Trying to introduce classical reasonings further inside the type theories developed in Part II, namely when these feature *dependent types*, runs into the problem of defining a suitable notion of equivalence for classical proofs. In this chapter we suggest an original approach, in the framework of classical propositional logic, based on the *Calculus of Structures* [Gug, Gug02]. Proofs are rewrite sequences

on formulae, and we use the notion of *parallel reduction* [Tak89] to collapse bureaucratic sequentialisations of rewrite steps that reduce non-overlapping redexes. The case of parallel redexes and that of nested redexes give rise to two notions of equivalence on rewrite sequences (according to a linear rewrite system that formalises classical logic). We thence introduce two formalisms that allow parallel rewrite steps, thus providing to equivalent proofs a single representative. This representative can be obtained from any proof in the equivalence class by a particular reduction relation, which is confluent and terminating (confluence in the case of nested redexes is only conjectured). These formalisms turn out to be sequent calculi with axioms and proof-terms for them use combinators as in Hilbert-style systems. The normalisation process that produce canonical representatives is then a cut-reduction process.

Dependencies between chapters

To facilitate the reading of this dissertation and allow particular chapters to be read independently, we show in Fig. 1 the dependency graph of chapters, together with the references where some or all of their contents already appeared.

Full arrows represent a real dependency (but sometimes only because a chapter uses definitions, given in another chapter, of notions that are familiar to the reader), while dashed arrows represent only a weak connection without making one chapter a prerequisite for the other.

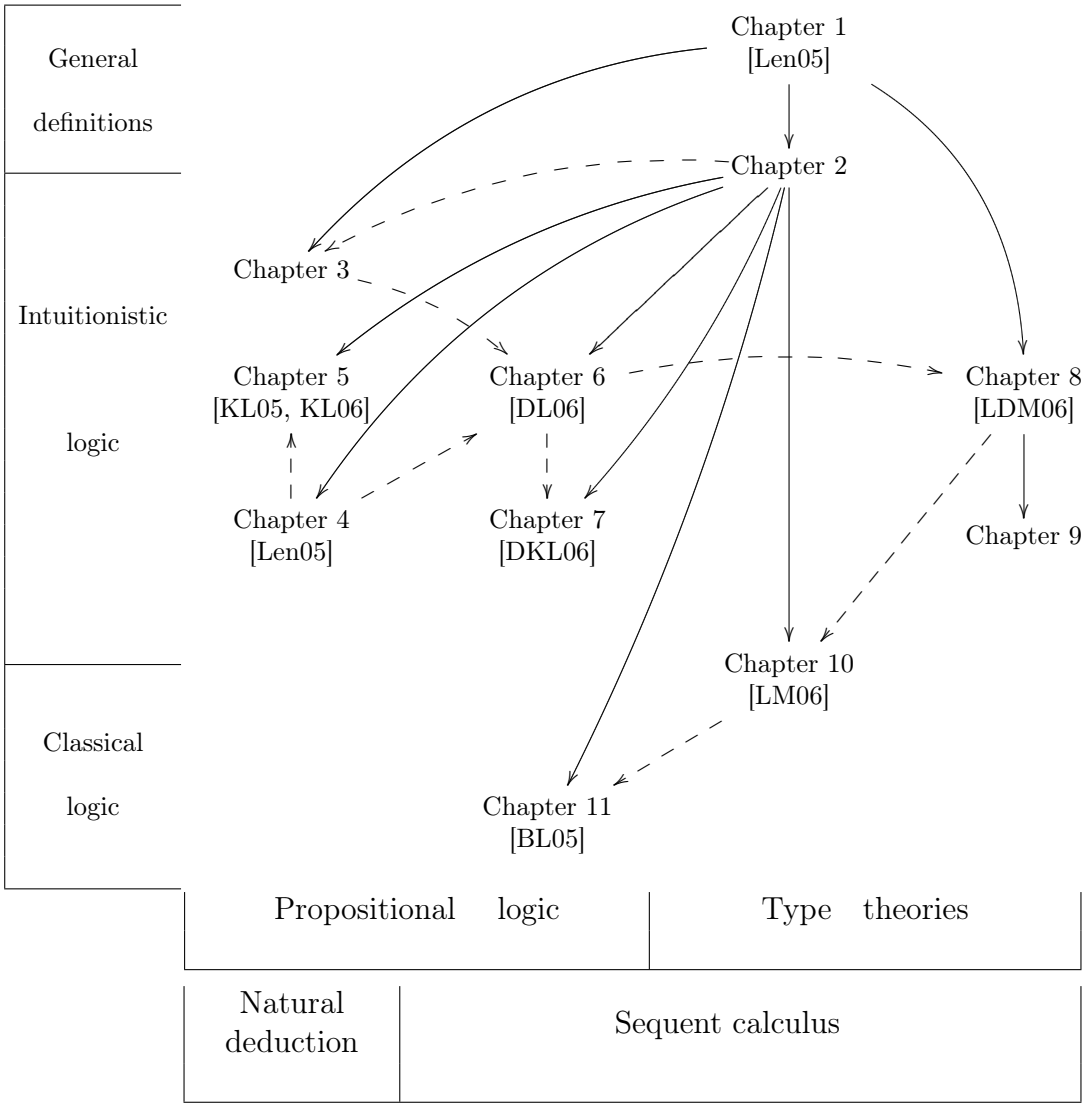


Figure 1: Dependency graph

Chapter 1

Concepts & terminology

This chapter defines coherent terminology and notations for the concepts used in this dissertation.

Section 1.1 introduces the concepts related to relations, simulation, confluence, inference and derivations.

Section 1.2 tackles notions of normalisation. Their definitions are inspired by a thread created by René Vestergaard on the TYPES mailing-list, gathering and comparing various definitions. Our first purpose here is defining and establishing a theory of normalisation that does not rely on classical logic and double negation.

Negation usually lies in the very definition of strong normalisation when it is expressed as "there is no infinite reduction sequence". The most striking example is the following use of the definition to prove that a reduction relation is strongly normalising, starting with "suppose an infinite reduction sequence" and ending with a contradiction.

A positive version such as "all reduction sequences are finite" subtly requires a general definition of reduction sequence that can be finite or not, and its careful formal treatment in a constructive theory of finiteness and infiniteness does not seem simpler than our approach, which we now present.

In our approach, the induction principle is no longer a property of strongly normalising relations, but is the basis of its very definition. In other words, instead of basing the notion of strong normalisation on the finiteness of reduction sequences, we base it on the notion of induction: by definition, a relation is strongly normalising if it satisfies the induction principle. The latter should hold for every predicate, so the notion of normalisation is based on second-order quantification rather than double-negation.

We express several induction principles in this setting, then we re-establish some traditional results, especially some techniques to prove strong normalisation. We constructively prove the correctness of the simulation technique and a few refinements, as well as the termination of the lexicographic reductions and the multi-set reductions. A constructive proof of the latter has already been given by Wilfried Buchholz and is a special case of Coquand's constructive treat-

ment [Coq94] of Ramsey theory.

Most of the material in this section can be found in various textbooks (e.g. [Ter03]), but perhaps not always with constructive proofs, and we intend to make this dissertation self-contained.

A first version of this section, together with the major part of chapter 4, appeared as the technical report [Len05].

Section 1.3 describes how the rest of the dissertation treats higher-order calculi, i.e. calculi involving variable binding. Again, a good overview of formalisms describing higher-order formalisms can be found in [Ter03].

Objects of the theory are α-equivalence classes of terms, and variable binding requires us to take care of such problems as variable capture and variable liberation, constantly juggling between α-equivalence classes and their representatives.

Reasonings about higher-order calculi are thus tricky: formalising them properly in first-order logic, where terms have no intrinsic notion of binding, might require a lot of side-conditions and lemmas (e.g. about renaming, name-indifference, etc.).

Whether or not such a heavy formalisation is in fact helpful for the reader's understanding might be questioned. It could be argued that a human mind is rather distracted from the main reasoning by such formalities, while it can naturally grasp reasonings modulo α-conversion, noting that mathematicians have been working with bound variables for a long time.

Stating Barendregt's convention at the beginning of a short paper is often the only option that space permits, with the implicit intention to convince the reader that, with some effort, he could formalise the forthcoming reasonings in first-order logic by recovering all necessary side-conditions.

Here we develop the ideas of Barendregt's convention, starting with the claim that solutions for dealing with variable binding do not concern the object-level (in other words, the *terms*), but the meta-level in the way we describe reasoning —in other words, the *expressions* that we use to denote terms.

We can *mechanically* infer the side-conditions avoiding variable capture and liberation by looking at expressions: for each meta-variable we look at the binders in the scope of which it occurs, and if a binder on some variable appears above one occurrence but not above another, then we forbid this variable to be free in the term represented by the meta-variable, otherwise it will be liberated or captured (depending on the way we see the two occurrences).

The concepts are thus very natural but their formalisation makes this section rather technical, as with most works tackling the formalisation of the meta-level.

Hence, reading section 1.3 is only of significant interest to the rest of the dissertation insofar as assuring that our treatment of α-equivalence is rigorous. We also recall the notions of terms, sub-terms, rewrite systems, but only standard knowledge of these concepts is required for the understanding of the following chapters.

1.1 Relations

We take for granted usual notions and results of set theory, such as the empty set and subsets, the union, intersection and difference of sets, relations, functions, injectivity, surjectivity, natural numbers... (see e.g. [Kri71]). Unless otherwise stated, relations are binary relations. We denote by $|\mathcal{S}|$ the cardinal of a set $\mathcal{S}$.

1.1.1 Definitions & notations

Definition 1 (Relations)

- We denote the composition of relations by $\cdot$, the identity relation by Id, and the inverse of a relation by $^{-1}$, all defined below:

  Let $\mathcal{R} : \mathcal{A} \longrightarrow \mathcal{B}$ and $\mathcal{R}' : \mathcal{B} \longrightarrow \mathcal{C}$.

  - Composition
    $\mathcal{R} \cdot \mathcal{R}' : \mathcal{A} \longrightarrow \mathcal{C}$ is defined as follows: given $M \in \mathcal{A}$ and $N \in \mathcal{C}$, $M(\mathcal{R} \cdot \mathcal{R}')N$ if there exists $P \in \mathcal{B}$ such that $M\mathcal{R}P$ and $P\mathcal{R}'N$. Sometimes we also use the notation $\mathcal{R}' \circ \mathcal{R}$ for $\mathcal{R} \cdot \mathcal{R}'$.
  - Identity
    $\mathsf{Id}[\mathcal{A}] : \mathcal{A} \longrightarrow \mathcal{A}$ is defined as follows:
    given $M \in \mathcal{A}$ and $N \in \mathcal{A}$, $M\mathsf{Id}_{\mathcal{A}}N$ if $M = N$.
  - Inverse
    $\mathcal{R}^{-1} : \mathcal{B} \longrightarrow \mathcal{A}$ is defined as follows:
    given $M \in \mathcal{B}$ and $N \in \mathcal{A}$, $M\mathcal{R}^{-1}N$ if $N\mathcal{R}M$.

- If $\mathcal{D} \subseteq \mathcal{A}$, we write $\mathcal{R}(\mathcal{D})$ for $\{M \in \mathcal{B} |\ \exists N \in \mathcal{D}, N\mathcal{R}M\}$, or equivalently $\bigcup_{N \in \mathcal{D}} \{M \in \mathcal{B} |\ N\mathcal{R}M\}$. When $\mathcal{D}$ is the singleton $\{M\}$, we write $\mathcal{R}(M)$ for $\mathcal{R}(\{M\})$.

- Now when $\mathcal{A} = \mathcal{B}$ we define the *relation induced by* $\mathcal{R}$ *through* $\mathcal{R}'$, written $\mathcal{R}'[\mathcal{R}]$, as $\mathcal{R}'^{-1} \cdot \mathcal{R} \cdot \mathcal{R}' : \mathcal{C} \longrightarrow \mathcal{C}$.

- We say that a relation $\mathcal{R} : \mathcal{A} \longrightarrow \mathcal{B}$ is *total* if $\mathcal{R}^{-1}(\mathcal{B}) = \mathcal{A}$.

- If $\mathcal{R} : \mathcal{A} \longrightarrow \mathcal{B}$ and $\mathcal{A}' \subseteq \mathcal{A}$ then $\mathcal{R}_{|\mathcal{A}'} : \mathcal{A}' \longrightarrow \mathcal{B}$ is the restriction of $\mathcal{R}$ to $\mathcal{A}'$, i.e. those pairs of $\mathcal{R}$ whose first components are in $\mathcal{A}'$.

- All those notions and notations can be used in the particular case when $\mathcal{R}$ is a function, that is, if $\forall M \in \mathcal{A}$, $\mathcal{R}(M)$ is of the form $\{N\}$ (which we simply write $\mathcal{R}(M) = N$).

- A total function is called a *mapping* (also called an *encoding*, a *translation* or an *interpretation*).

- An injective mapping is called an *embedding*.

Remark 1 Notice that composition is associative, and identity relations are neutral for the composition operation.

Computation in a calculus is described by the notion of reduction relation, defined as follows.

Definition 2 (Reduction relation)

- A *reduction relation* on $\mathcal{A}$ is a relation from $\mathcal{A}$ to $\mathcal{A}$ (i.e. a subset of $\mathcal{A}\times\mathcal{A}$), which we often write as $\rightarrow$.

- Given a reduction relation $\rightarrow$ on $\mathcal{A}$, we define the set of $\rightarrow$-*reducible forms* (or just *reducible forms* when the relation is clear) as $\mathsf{rf}^{\rightarrow} := \{M \in \mathcal{A} |\ \exists N \in \rightarrow(M)\}$. We define the set of *normal forms* as $\mathsf{nf}^{\rightarrow} := \{M \in \mathcal{A} |\ \rightarrow(M) = \emptyset\}$. In other words,

$$\begin{array}{rcl} \mathsf{rf}^{\rightarrow} & := & \{M \in \mathcal{A} |\ \exists N \in \mathcal{A}, M \rightarrow N\} \\ \mathsf{nf}^{\rightarrow} & := & \{M \in \mathcal{A} |\ \nexists N \in \mathcal{A}, M \rightarrow N\} \end{array}$$

- Given a reduction relation $\rightarrow$ on $\mathcal{A}$, we write $\leftarrow$ for $\rightarrow^{-1}$, and we define $\rightarrow^n$ by induction on the natural number n as follows:
$\rightarrow^0 := \mathsf{Id}$
$\rightarrow^{n+1} := \rightarrow \cdot \rightarrow^n (= \rightarrow^n \cdot \rightarrow)$
$\rightarrow^+$ denotes the transitive closure of $\rightarrow$ (i.e. $\rightarrow^+ := \bigcup_{n\geq 1} \rightarrow^n$).
$\rightarrow^*$ denotes the transitive and reflexive closure of $\rightarrow$ (i.e. $\rightarrow^* := \bigcup_{n\geq 0} \rightarrow^n$).
$\leftrightarrow$ denotes the symmetric closure of $\rightarrow$ (i.e. $\leftrightarrow := \leftarrow \cup \rightarrow$).
$\leftrightarrow^*$ denotes the transitive, reflexive and symmetric closure of $\rightarrow$.

- An *equivalence relation* on $\mathcal{A}$ is a transitive, reflexive and symmetric reduction relation on $\mathcal{A}$, i.e. a relation $\rightarrow = \leftrightarrow^*$, hence denoted more often by $\sim$, $\equiv \ldots$

- Given a reduction relation $\rightarrow$ on $\mathcal{A}$ and a subset $\mathcal{B} \subseteq \mathcal{A}$, the *closure of* $\mathcal{B}$ *under* $\rightarrow$ is $\rightarrow^*(\mathcal{B})$.

Definition 3 (Finitely branching relation) A reduction relation $\rightarrow$ on $\mathcal{A}$ is *finitely branching* if $\forall M \in \mathcal{A},\ \rightarrow(M)$ is finite.

Definition 4 (Stability) Given a reduction relation $\rightarrow$ on $\mathcal{A}$, we say that a subset $\mathcal{T}$ of $\mathcal{A}$ is $\rightarrow$-*stable* (or *stable under* $\rightarrow$) if $\rightarrow(\mathcal{T}) \subseteq \mathcal{T}$ (in other words, if $\mathcal{T}$ is equal to its closure under $\rightarrow$).

Definition 5 (Reduction modulo) Let $\sim$ be an equivalence relation on a set $\mathcal{A}$, let $\to$ be a reduction relation on $\mathcal{A}$. The *reduction relation* $\to$ *modulo* $\sim$ on $\mathcal{A}$, denoted $\to_\sim$, is $\sim \cdot \to \cdot \sim$. It provides a reduction relation on the $\sim$-equivalence classes of $\mathcal{A}$. If $\to'$ is a reduction relation $\to$ modulo $\sim$, $\to$ alone is called the *basic* reduction relation and denoted $\to'_b$.[1]

We now present the notion of simulation. We shall use it for two kinds of results: confluence (below) and strong normalisation (section 1.2). While simulation is often presented using an mapping from one calculus to another, we provide here a useful generalised version for an arbitrary relation between two calculi.

Definition 6 (Strong and weak simulation)
Let $\mathcal{R}$ be a relation between two sets $\mathcal{A}$ and $\mathcal{B}$, respectively equipped with the reduction relations $\to_\mathcal{A}$ and $\to_\mathcal{B}$.

- $\to_\mathcal{B}$ *strongly simulates* $\to_\mathcal{A}$ *through* $\mathcal{R}$ if $(\mathcal{R}^{-1} \cdot \to_\mathcal{A}) \subseteq (\to_\mathcal{B}^+ \cdot \mathcal{R}^{-1})$.

  In other words, for all $M, M' \in \mathcal{A}$ and for all $N \in \mathcal{B}$, if $M\mathcal{R}N$ and $M \to_\mathcal{A} M'$ then there is $N' \in \mathcal{B}$ such that $M'\mathcal{R}N'$ and $N \to_\mathcal{B}^+ N'$.

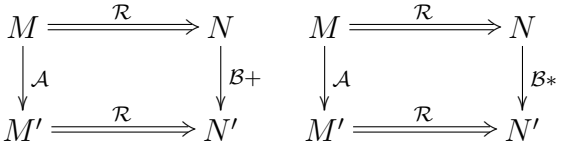
  Notice that when $\mathcal{R}$ is a function, this implies $\mathcal{R}[\to_\mathcal{A}] \subseteq \to_\mathcal{B}^+$.
  If it is a mapping, then $\to_\mathcal{A} \subseteq \mathcal{R}^{-1}[\to_\mathcal{B}^+]$.

- $\to_\mathcal{B}$ *weakly simulates* $\to_\mathcal{A}$ *through* $\mathcal{R}$ if $(\mathcal{R}^{-1} \cdot \to_\mathcal{A}) \subseteq (\to_\mathcal{B}^* \cdot \mathcal{R}^{-1})$.

  In other words, for all $M, M' \in \mathcal{A}$ and for all $N \in \mathcal{B}$, if $M\mathcal{R}N$ and $M \to_\mathcal{A} M'$ then there is $N' \in \mathcal{B}$ such that $M'\mathcal{R}N'$ and $N \to_\mathcal{B}^* N'$.

  Notice that when $\mathcal{R}$ is a function, this implies $\mathcal{R}[\to_\mathcal{A}] \subseteq \to_\mathcal{B}^*$.
  If it is a mapping, then $\to_\mathcal{A} \subseteq \mathcal{R}^{-1}[\to_\mathcal{B}^*]$.

The notions are illustrated in Fig. 1.1.



Strong simulation Weak simulation

Figure 1.1: Strong and weak simulation

[1] This is not a functional notation that only depends on a reduction relation $\to'$ on $\sim$-equivalence classes of $\mathcal{A}$, but a notation that depends on the construction of $\to'$ as a reduction relation modulo $\sim$.

Remark 2

1. If $\to_{\mathcal{B}}$ strongly (resp. weakly) simulates $\to_{\mathcal{A}}$ through $\mathcal{R}$, and if $\to_{\mathcal{B}} \subseteq \to'_{\mathcal{B}}$ and $\to'_{\mathcal{A}} \subseteq \to_{\mathcal{A}}$, then $\to'_{\mathcal{B}}$ strongly (resp. weakly) simulates $\to'_{\mathcal{A}}$ through $\mathcal{R}$.

2. If $\to_{\mathcal{B}}$ strongly (resp. weakly) simulates $\to_{\mathcal{A}}$ and $\to'_{\mathcal{A}}$ through $\mathcal{R}$, then it also strongly (resp. weakly) simulates $\to_{\mathcal{A}} \cdot \to'_{\mathcal{A}}$ through $\mathcal{R}$.

3. Hence, if $\to_{\mathcal{B}}$ strongly simulates $\to_{\mathcal{A}}$ through $\mathcal{R}$, then it also strongly simulates $\to^+_{\mathcal{A}}$ through $\mathcal{R}$.
   If $\to_{\mathcal{B}}$ strongly or weakly simulates $\to_{\mathcal{A}}$ through $\mathcal{R}$, then it also weakly simulates $\to^+_{\mathcal{A}}$ and $\to^*_{\mathcal{A}}$ through $\mathcal{R}$.

We now define some more elaborate notions based on simulation, such as equational correspondence [SF93], Galois connection and reflection [MSS86].

Definition 7 (Galois connection, reflection & related notions)
Let $\mathcal{A}$ and $\mathcal{B}$ be sets respectively equipped with the reduction relations $\to_{\mathcal{A}}$ and $\to_{\mathcal{B}}$. Consider two mappings $f : \mathcal{A} \longrightarrow \mathcal{B}$ and $g : \mathcal{B} \longrightarrow \mathcal{A}$.

- f and g form an *equational correspondence* between $\mathcal{A}$ and $\mathcal{B}$ if the following holds:
  - $f[\leftrightarrow_{\mathcal{A}}] \subseteq \leftrightarrow_{\mathcal{B}}$
  - $g[\leftrightarrow_{\mathcal{B}}] \subseteq \leftrightarrow_{\mathcal{A}}$
  - $f \cdot g \subseteq \leftrightarrow_{\mathcal{A}}$
  - $g \cdot f \subseteq \leftrightarrow_{\mathcal{B}}$
- f and g form a *Galois connection* from $\mathcal{A}$ to $\mathcal{B}$ if the following holds:
  - $\to_{\mathcal{B}}$ weakly simulates $\to_{\mathcal{A}}$ through f
  - $\to_{\mathcal{A}}$ weakly simulates $\to_{\mathcal{B}}$ through g
  - $f \cdot g \subseteq \to^*_{\mathcal{A}}$
  - $g \cdot f \subseteq \leftarrow^*_{\mathcal{B}}$
- f and g form a *pre-Galois connection* from $\mathcal{A}$ to $\mathcal{B}$ if in the four conditions above we remove the last one.
- f and g form a *reflection* in $\mathcal{A}$ of $\mathcal{B}$ if the following holds:
  - $\to_{\mathcal{B}}$ weakly simulates $\to_{\mathcal{A}}$ through f
  - $\to_{\mathcal{A}}$ weakly simulates $\to_{\mathcal{B}}$ through g
  - $f \cdot g \subseteq \to^*_{\mathcal{A}}$
  - $g \cdot f = \mathsf{Id}_{\mathcal{B}}$

Remark 3

1. Note that saying that f and g form an equational correspondence between $\mathcal{A}$ and $\mathcal{B}$ only means that f and g extend to a bijection between $\leftrightarrow_{\mathcal{A}}$-equivalence classes of $\mathcal{A}$ and $\leftrightarrow_{\mathcal{B}}$-equivalence classes of $\mathcal{B}$. If f and g form an equational correspondence, so do g and f; it is a symmetric relation, unlike (pre-)Galois connections and reflections.

2. A Galois connection forms both an equational correspondence and a pre-Galois connection. A reflection forms a Galois connection. Also note that if f and g form a reflection then g and f form a pre-Galois connection.

3. If f and g form an equational correspondence between $\mathcal{A}$ and $\mathcal{B}$ (resp. a pre-Galois connection from $\mathcal{A}$ to $\mathcal{B}$, a Galois connection from $\mathcal{A}$ to $\mathcal{B}$, a reflection in $\mathcal{A}$ of $\mathcal{B}$),
and f' and g' form an equational correspondence between $\mathcal{B}$ and $\mathcal{C}$ (resp. a pre-Galois connection from $\mathcal{B}$ and $\mathcal{C}$, a Galois connection from $\mathcal{B}$ and $\mathcal{C}$, a reflection in $\mathcal{B}$ of $\mathcal{C}$),
then $f \cdot f'$ and $g \cdot g'$ form an equational correspondence between $\mathcal{A}$ and $\mathcal{C}$ (resp. a pre-Galois connection from $\mathcal{A}$ and $\mathcal{C}$, a Galois connection from $\mathcal{A}$ and $\mathcal{C}$, a reflection in $\mathcal{A}$ of $\mathcal{C}$).

1.1.2 Confluence

Definition 8 (Confluence & Church-Rosser)

- A reduction relation $\rightarrow$ on $\mathcal{A}$ is *confluent* if $\leftarrow^* \cdot \rightarrow^* \subseteq \rightarrow^* \cdot \leftarrow^*$
- A reduction relation $\rightarrow$ on $\mathcal{A}$ is *Church-Rosser* if $\leftrightarrow^* \subseteq \rightarrow^* \cdot \leftarrow^*$

Theorem 4 (Confluence is equivalent to Church-Rosser)
A reduction relation $\rightarrow$ is confluent if and only if it is Church-Rosser.

Proof:

- *if*: it suffices to note that $\leftarrow^* \cdot \rightarrow^* \subseteq \leftrightarrow^*$.
- *only if*: we prove $\leftrightarrow^n \subseteq \rightarrow^* \cdot \leftarrow^*$ by induction on n. For $n = 0$ it trivially holds. Suppose it holds for $\leftrightarrow^n$.

$$\begin{array}{lcll} \leftrightarrow^{n+1} & = & \leftrightarrow^n \cdot (\leftarrow \cup \rightarrow) & \\ & \subseteq & \rightarrow^* \cdot \leftarrow^* \cdot (\leftarrow \cup \rightarrow) & \text{by i.h.} \\ & = & (\rightarrow^* \cdot \leftarrow^*) \cup (\rightarrow^* \cdot \leftarrow^* \cdot \rightarrow) & \\ & \subseteq & \rightarrow^* \cdot \leftarrow^* & \text{by assumption} \end{array}$$

We can illustrate in Fig. 1.2 the right-hand side case of the union $\cup$.

□

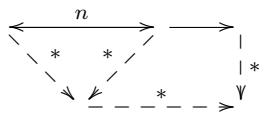


Figure 1.2: Confluence implies Church-Rosser

Theorem 5 (Confluence by simulation) *If f and g form a pre-Galois connection from $\mathcal{A}$ to $\mathcal{B}$ and $\rightarrow_{\mathcal{B}}$ is confluent, then $\rightarrow_{\mathcal{A}}$ is confluent.*

Proof:

$$
\begin{array}{lcll}
\leftarrow^*_{\mathcal{A}} \cdot \rightarrow^*_{\mathcal{A}} & \subseteq & f^{-1}[\leftarrow^*_{\mathcal{B}} \cdot \rightarrow^*_{\mathcal{B}}] & \text{weak simulation} \\
& \subseteq & f^{-1}[\rightarrow^*_{\mathcal{B}} \cdot \leftarrow^*_{\mathcal{B}}] & \text{confluence of } \rightarrow_{\mathcal{B}} \\
& = & f \cdot \rightarrow^*_{\mathcal{B}} \cdot \leftarrow^*_{\mathcal{B}} \cdot f^{-1} & \\
& \subseteq & f \cdot g^{-1}[\rightarrow^*_{\mathcal{A}} \cdot \leftarrow^*_{\mathcal{A}}] \cdot f^{-1} & \text{weak simulation} \\
& = & f \cdot g \cdot \rightarrow^*_{\mathcal{A}} \cdot \leftarrow^*_{\mathcal{A}} g^{-1} \cdot f^{-1} & \text{weak simulation} \\
& \subseteq & \rightarrow^*_{\mathcal{A}} \cdot \leftarrow^*_{\mathcal{A}} & \text{by assumption}
\end{array}
$$

This proof can be graphically represented in Fig. 1.3. □

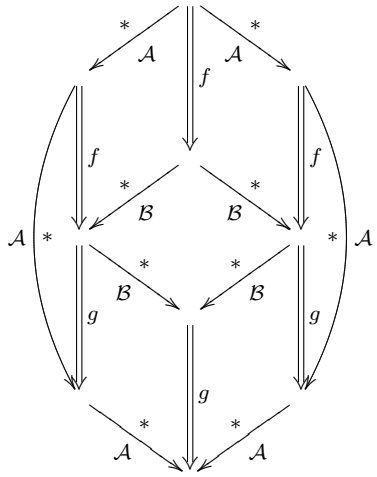


Figure 1.3: Confluence by simulation

1.1.3 Inference & derivations

We take for granted the notion of (labelled) tree, the notions of node, internal node and leaf, see e.g. [CDG+97]. In particular, the *height* of a tree is the length of its longest branch (e.g. the height of a tree with only one node is 1), and its size is its number of nodes.

We now introduce the notions of *inference structure* and *derivations*. The former are used to inductively define atomic predicates, which can be seen as

particular sets (for predicates with one argument), or as particular n-ary relations (for predicates with n-arguments). The definitions are more readable if we only consider sets (rather than arbitrary n-ary relations), but are not less general: indeed, a n-ary relation is but a set of n-tuples.

Definition 9 (Inference structure) Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be sets whose elements are called *judgements*. An *inference structure* is a set of non-empty tuples of judgements, usually denoted

$$\frac{M_1 \ldots M_n}{M}$$

instead of $(M, M_1, \ldots, M_n)$. M is called the *conclusion* of the tuple and $M_1 \ldots M_n$ are called the *premisses*.

Definition 10 (Derivations)

- A *derivation* in an inference structure (sometimes called *full derivation*[2]) is a tree whose nodes are labelled with judgements, and such that if a node is labelled with M and has n sons ($n \geq 0$) respectively labelled with $M_1, \ldots, M_n$ then $(M, M_1, \ldots, M_n)$ is in the inference structure.[3]

- A *partial derivation*[4] is a tree whose nodes are labelled with judgements, together with a subset of its leaves whose elements are called *open leaves*, and such that if a node is not an open leaf, is labelled with M and has n sons ($n \geq 0$) respectively labelled with $M_1, \ldots, M_n$ then $(M, M_1, \ldots, M_n)$ is in the inference structure.[5]

- The judgement at the root of a (partial or full) derivation is called the *conclusion of the derivation*. The latter is said to *conclude* this judgement.

- A *derivation of a judgement* is a (full) derivation concluding this judgement. The latter is said to be *derivable*.

- A *derivation from a set $\mathcal{A}$ to a judgement M* is a partial derivation concluding M and whose open leaves are labelled with judgements in $\mathcal{A}$.

- Derivations inherit from their tree structures a notion of *sub-derivation*, *height* and *size*. We sometimes say that we prove a statement "by induction on a derivation" when we mean "on the height of a derivation".

- A *derivation step* is a partial derivation of height 1, i.e. a node and its sons (i.e. an element of the inference structure).

[2]Some authors also call them *complete derivations* or *categorical derivations*
[3]Leaves of the tree are such that $n = 0$, with (M) belonging to the inference structure.
[4]Some authors also call them *complete derivations* or *hypothetical derivations*
[5]Note that no condition is imposed on open leaves.

- We write down derivations by composing with itself the notation with one horizontal bar that we use for inference steps, as shown in Example 1.

Example 1 (Inference structure & derivation) Consider the following inference structure:

$$\frac{d \quad b}{c} \qquad \frac{c \quad b}{a} \qquad \frac{}{d}$$

The following derivation is from $\{b\}$ to a, has height 3 and size 5.

$$\dfrac{\dfrac{\dfrac{}{d} \quad b}{c} \qquad b}{a}$$

Note the different status of the leaves labelled with b and d, the former being open and the latter being not.

Definition 11 (Derivability & admissibility)

- A tuple of judgements $\dfrac{M_1 \dots M_n}{M}$ is *derivable* in an inference system if there is a derivation from the set $\{M_1, \dots, M_n\}$ to M.
  In this case we write $\dfrac{M_1 \dots M_n}{M}$.[6]
- A tuple of judgements $\dfrac{M_1 \dots M_n}{M}$ is *admissible* in an inference system if for all derivations of $M_1 \dots M_n$ there is a derivation of M.
  In this case we write $\dfrac{M_1 \dots M_n}{M}$.
- A tuple of judgements $\dfrac{M_1 \dots M_n}{M}$ is *height-preserving admissible* in an inference system if for all derivations of $M_1 \dots M_n$ with heights at most $h \in \mathbb{N}$ there exists a derivation of M with height at most h.
  In this case we write $\dfrac{M_1 \dots M_n}{M}$.[7]

[6] Note that our notation for derivability, using a double line, is used by some authors for invertibility. Our notation is based on Kleene's [Kle52], with the rationale that the double line evokes several inference steps.

[7] The rationale of our notation for height-preserving admissibility is that we can bound the height of a derivation with fake steps of height-preserving admissibility just by not counting these steps, since the conclusion in such a step can be derived with a height no greater than that of some premiss.

- A tuple of judgements $\dfrac{M_1 \dots M_n}{M}$ is *invertible* in an inference system if it is derivable and if for all derivations of M there are derivations of $M_1 \dots M_n$. In this case we write $\overset{M_1 \dots M_n}{\wr\wr\wr\wr\wr\wr\wr\wr\wr\wr\wr}{\scriptstyle M}$.[8]

We shall use these notations within derivations: when writing a derivation we can use derivable and admissible tuples as *fake inference steps*. Proofs of derivability and admissibility then provide the real derivations that are denoted with fake steps: a fake step of derivability stands in fact for a sequence of real steps, while a fake step of admissibility requires its premisses to be derivable (i.e. with full derivations rather than partial ones).

Remark 6 If $\dfrac{M_1 \dots M_n}{M}$ is derivable then it is admissible (we can plug the derivations of $M_1,\dots,\ M_n$ into the derivation from $M_1,\dots,\ M_n$ to M). Note that the reverse is not true: in order to build a derivation of M knowing derivations of $M_1,\dots,\ M_n$ we could use another construction than the one above, potentially without the existence of a derivation from $M_1,\dots,\ M_n$ to M.

Remark 7 Note that a reduction relation is a particular inference structure made of pairs.

Definition 12 (Reduction sequence)

- A *reduction sequence* is a partial derivation in a reduction relation $\to$, and we often write $M \to \cdots \to N$ instead of $\begin{array}{c} M \\ \hline \vdots \\ \hline N \end{array}$. In that case we also say *reduction step* instead of *inference step*. Note that $M \to^* N$ is then the same as $\dfrac{M}{\overline{\overline{N}}}$.

- The height of a reduction sequence is also called its *length*.

Remark 8 Note that $M \to^n N$ if and only if there is a reduction sequence of length n from M to N.

[8] The rationale of our notation for invertibility is that derivability of the conclusion is *equivalent* to the derivability of the premisses.

1.2 A constructive theory of normalisation

1.2.1 Normalisation & induction

Proving a universally quantified property by induction consists of verifying that the set of elements having the property is stable, in some sense similar to —yet more subtle than— that of Definition 4. Leading to different induction principles, we define two such notions of stability property: being *patriarchal* and being *paternal*.

Definition 13 (Patriarchal, paternal) Given a reduction relation $\rightarrow$ on $\mathcal{A}$, we say that

- a subset $\mathcal{T}$ of $\mathcal{A}$ is $\rightarrow$*-patriarchal* (or just *patriarchal* when the relation is clear) if $\forall N \in \mathcal{A},\ \rightarrow(N) \subseteq \mathcal{T} \Rightarrow N \in \mathcal{T}$.
- a subset $\mathcal{T}$ of $\mathcal{A}$ is $\rightarrow$*-paternal* (or just *paternal* when the relation is clear) if it contains $\mathsf{nf}^{\rightarrow}$ and is stable under $\rightarrow^{-1}$.
- a predicate P on $\mathcal{A}$ is *patriarchal* (resp. *paternal*) if $\{M \in \mathcal{A} |\ P(M)\}$ is *patriarchal* (resp. *paternal*).

Lemma 9 *Suppose that for any N in $\mathcal{A}$, $N \in \mathsf{rf}^{\rightarrow}$ or $N \in \mathsf{nf}^{\rightarrow}$ and suppose $\mathcal{T} \subseteq \mathcal{A}$. If $\mathcal{T}$ is paternal, then it is patriarchal.*

Proof: In order to prove $\forall N \in \mathcal{A},\ \rightarrow(N) \subseteq \mathcal{T} \Rightarrow N \in \mathcal{T}$, a case analysis is needed: either $N \in \mathsf{rf}^{\rightarrow}$ or $N \in \mathsf{nf}^{\rightarrow}$. In both cases $N \in \mathcal{T}$ because $\mathcal{T}$ is paternal. □

Remark 10 Notice that we can obtain from classical logic the hypothesis for all N in $\mathcal{A}$, $N \in \mathsf{rf}^{\rightarrow}$ or $N \in \mathsf{nf}^{\rightarrow}$, because it is an instance of the Law of Excluded Middle. In intuitionistic logic, assuming this amounts to saying that being reducible is decidable, which might not always be true.

We would not require this hypothesis if we defined that $\mathcal{T}$ is paternal whenever $\forall N \in \mathcal{A}, N \in \mathcal{T} \vee (N \in \mathsf{rf}^{\rightarrow} \wedge (\rightarrow(N) \cap \mathcal{T} = \emptyset))$. This is classically equivalent to the definition above, but this definition also has some disadvantages as we shall see later.

Typically, if we want to prove that a predicate P on on some set $\mathcal{A}$ holds throughout $\mathcal{A}$, we actually prove that P is patriarchal or paternal, depending on the induction principle we use.

Hence, we define normalisation so that normalising elements are those captured by an induction principle, which should hold for every predicate satisfying the corresponding stability property. We thus get two notions of normalisation: the *strongly* (resp. *weakly*) *normalising* elements are those in every patriarchal (resp. paternal) set.

Definition 14 (Normalising elements) Given a reduction relation $\rightarrow$ on $\mathcal{A}$:

- The set of $\rightarrow$-*strongly normalising* elements is

$$\mathsf{SN}^{\rightarrow} := \bigcap_{\tau \text{ is patriarchal}} \mathcal{T}$$

- The set of $\rightarrow$-*weakly normalising* elements is

$$\mathsf{WN}^{\rightarrow} := \bigcap_{\tau \text{ is paternal}} \mathcal{T}$$

Remark 11 Interestingly enough, $\mathsf{WN}^{\rightarrow}$ can also be captured by an inductive definition:

$$\mathsf{WN}^{\rightarrow} = \bigcup_{n \geq 0} \mathsf{WN}^{\rightarrow}_n$$

where $\mathsf{WN}^{\rightarrow}_n$ is defined by induction on the natural number n as follows:

$$\begin{array}{lcl} \mathsf{WN}^{\rightarrow}_0 & := & \mathsf{nf}^{\rightarrow} \\ \mathsf{WN}^{\rightarrow}_{n+1} & := & \{M \in \mathcal{A} |\ \exists n' \leq n, M \in {\rightarrow}^{-1}(\mathsf{WN}^{\rightarrow}_{n'})\} \end{array}$$

With the alternative definition of paternal suggested in Remark 10, the inclusion $\mathsf{WN}^{\rightarrow} \subseteq \bigcup_n \mathsf{WN}^{\rightarrow}_n$ would require the assumption that being reducible by $\rightarrow$ is decidable. We therefore preferred the first definition because we can then extract from a term M in $\mathsf{WN}^{\rightarrow}$ a natural number n such that $M \in \mathsf{WN}^{\rightarrow}_n$, without the hypothesis of decidability.

Such a characterisation gives us the possibility to prove that all weakly normalising elements satisfy some property by induction on natural numbers. On the other hand, trying to do so with strong normalisation leads to a different notion, as we shall see below. Hence, we lack for $\mathsf{SN}^{\rightarrow}$ an induction principle based on natural numbers, which is the reason why we built a specific induction principle into the definition of $\mathsf{SN}^{\rightarrow}$.

Definition 15 (Bounded elements) The set of $\rightarrow$-*bounded* elements is defined as

$$\mathsf{BN}^{\rightarrow} := \bigcup_{n \geq 0} \mathsf{BN}^{\rightarrow}_n$$

where $\mathsf{BN}^{\rightarrow}_n$ is defined by induction on the natural number n as follows:

$$\begin{array}{lcl} \mathsf{BN}^{\rightarrow}_0 & := & \mathsf{nf}^{\rightarrow} \\ \mathsf{BN}^{\rightarrow}_{n+1} & := & \{M \in \mathcal{A} |\ \exists n' \leq n,\ {\rightarrow}(M) \subseteq \mathsf{BN}^{\rightarrow}_{n'}\} \end{array}$$

But we have the following fact:

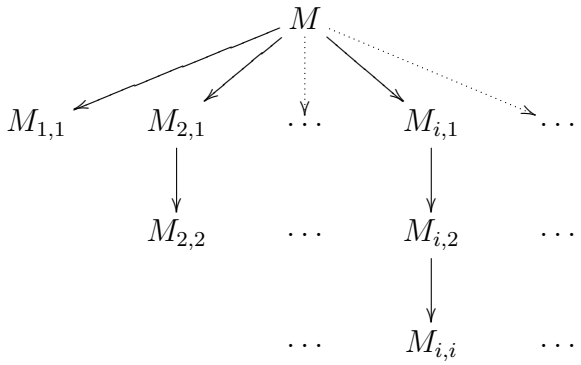


Figure 1.4: $M \in \mathsf{SN}^{\rightarrow}$ but $M \notin \mathsf{BN}^{\rightarrow}$

Remark 12 For some reduction relations $\rightarrow$, $\mathsf{SN}^{\rightarrow} \neq \mathsf{BN}^{\rightarrow}$. For instance, Fig. 1.4 shows a term M and relation $\rightarrow$ such that $M \in \mathsf{SN}^{\rightarrow}$ but $M \notin \mathsf{BN}^{\rightarrow}$.

Lemma 13 *However, if $\rightarrow$ is finitely branching, then $\mathsf{BN}^{\rightarrow}$ is patriarchal. As a consequence, $\mathsf{BN}^{\rightarrow} = \mathsf{SN}^{\rightarrow}$ (the counter-example above could not be finitely branching).*

Proof: Suppose $\rightarrow(M) \subseteq \mathsf{BN}^{\rightarrow}$. Because $\rightarrow$ is finitely branching, there exists a natural number n such that $\rightarrow(M) \subseteq \mathsf{BN}^{\rightarrow}_{n}$. Clearly, $M \in \mathsf{BN}^{\rightarrow}_{n+1} \subseteq \mathsf{BN}^{\rightarrow}$. $\square$

Remark 14 As a trivial example, all the natural numbers are $>$-bounded. Indeed, any natural number n is in $\mathsf{BN}^{>}_{n}$, which can be proved by induction.

A canonical way of proving a statement $\forall M \in \mathsf{BN}^{\rightarrow}, P(M)$ is to prove by induction on the natural number n that $\forall M \in \mathsf{BN}^{\rightarrow}_{n}, P(M)$. Although we can exhibit no such natural number on which a statement $\forall M \in \mathsf{SN}^{\rightarrow}, P(M)$ can be proved by induction, the following induction principles hold by definition of normalisation:

Remark 15 Given a predicate P on $\mathcal{A}$ and an element $M \in \mathcal{A}$,

1. If P is patriarchal and $M \in \mathsf{SN}^{\rightarrow}$, then $P(M)$.
2. If P is paternal and $M \in \mathsf{WN}^{\rightarrow}$, then $P(M)$.

When we use this remark to prove $\forall M \in \mathsf{SN}^{\rightarrow}, P(M)$ (resp. $\forall M \in \mathsf{WN}^{\rightarrow}, P(M)$), we say that we prove it *by raw induction in $\mathsf{SN}^{\rightarrow}$* (resp. *in $\mathsf{WN}^{\rightarrow}$*).

Definition 16 (Strongly & weakly normalising relations) Given a reduction relation $\rightarrow$ on $\mathcal{A}$ and a subset $\mathcal{T} \subseteq \mathcal{A}$, we say that the reduction relation is *strongly normalising* or *terminating* on $\mathcal{T}$ (or it *terminates* on $\mathcal{T}$) if $\mathcal{T} \subseteq \mathsf{SN}^{\rightarrow}$. We say that it is *weakly normalising* on $\mathcal{T}$ if $\mathcal{T} \subseteq \mathsf{WN}^{\rightarrow}$. If we do not specify $\mathcal{T}$, it means that we take $\mathcal{T} = \mathcal{A}$.

Lemma 16

1. *If $n < n'$ then $\mathit{BN}^{\to}_n \subseteq \mathit{BN}^{\to}_{n'} \subseteq \mathit{BN}^{\to}$. In particular, $\mathit{nf}^{\to} \subseteq \mathit{BN}^{\to}_n \subseteq \mathit{BN}^{\to}$.*
2. *$\mathit{BN}^{\to} \subseteq \mathit{SN}^{\to}$ and $\mathit{BN}^{\to} \subseteq \mathit{WN}^{\to}$.*
   Hence, all natural numbers are in $\mathit{SN}^{\rhd}$ and $\mathit{WN}^{\rhd}$.
3. *If being reducible is decidable (or if we work in classical logic), then $\mathit{SN}^{\to} \subseteq \mathit{WN}^{\to}$.*

Proof:

1. By definition.
2. Both facts can be proved for all $\mathsf{BN}^{\to}_n$ by induction on n.
3. This comes from Remark 9 and thus requires either classical logic or the particular instance of the Law of Excluded Middle stating that for all N,

$$N \in \mathsf{rf}^{\to} \vee N \in \mathsf{nf}^{\to}$$

□

Lemma 17

1. *$\mathit{SN}^{\to}$ is patriarchal, $\mathit{WN}^{\to}$ is paternal.*
2. *If $M \in \mathit{BN}^{\to}$ then $\to(M) \subseteq \mathit{BN}^{\to}$.*
   If $M \in \mathit{SN}^{\to}$ then $\to(M) \subseteq \mathit{SN}^{\to}$.
   If $M \in \mathit{WN}^{\to}$ then either $M \in \mathit{nf}^{\to}$ or $M \in \to^{-1}(\mathit{WN}^{\to})$
   (which implies $M \in \mathit{rf}^{\to} \Rightarrow M \in \to^{-1}(\mathit{WN}^{\to})$).

Proof:

1. For the first statement, let $M \in \mathcal{A}$ such that $\to(M) \subseteq \mathsf{SN}^{\to}$ and let $\mathcal{T}$ be patriarchal. We want to prove that $M \in \mathcal{T}$. It suffices to prove that $\to(M) \subseteq \mathcal{T}$. This is the case, because $\to(M) \subseteq \mathsf{SN}^{\to} \subseteq \mathcal{T}$.
   For the second statement, first notice that $\mathsf{nf}^{\to} \subseteq \mathsf{WN}^{\to}$. Now let $M, N \in \mathcal{A}$ such that $M \to N$ and $N \in \mathsf{WN}^{\to}$, and let $\mathcal{T}$ be paternal. We want to prove that $M \in \mathcal{T}$. This is the case because $N \in \mathcal{T}$ and $\mathcal{T}$ is paternal.
2. The first statement is straightforward.
   For the second, we show that $\mathcal{T} = \{P \in \mathcal{A} |\ \to(P) \subseteq \mathsf{SN}^{\to}\}$ is patriarchal:
   Let $P \in \mathcal{A}$ such that $\to(P) \subseteq \mathcal{T}$, that is, $\forall R \in \to(P)$, $\to(R) \subseteq \mathsf{SN}^{\to}$.
   Because $\mathsf{SN}^{\to}$ is patriarchal, $\forall R \in \to(P), R \in \mathsf{SN}^{\to}$.
   Hence, $\to(P) \subseteq \mathsf{SN}^{\to}$, that is, $P \in \mathcal{T}$ as required.
   Now by definition of $\mathsf{SN}^{\to}$, we get $M \in \mathcal{T}$.

For the third statement, we prove that $\mathcal{T} = \mathsf{nf}^{\to} \cup \ {\to}^{-1}(\mathsf{WN}^{\to})$ is paternal: Clearly, it suffices to prove that it is stable under ${\to}^{-1}$. Let $P, Q \in \mathcal{A}$ such that $P \to Q$ and $Q \in \mathcal{T}$. If $Q \in \mathsf{nf}^{\to} \subseteq \mathsf{WN}^{\to}$, then $P \in {\to}^{-1}(\mathsf{WN}^{\to}) \subseteq \mathcal{T}$. If $Q \in {\to}^{-1}(\mathsf{WN}^{\to})$, then, because $\mathsf{WN}^{\to}$ is paternal, we get $Q \in \mathsf{WN}^{\to}$, so that $P \in {\to}^{-1}(\mathsf{WN}^{\to}) \subseteq \mathcal{T}$ as required.
Now by definition of $M \in \mathsf{WN}^{\to}$, we get $M \in \mathcal{T}$.

□

Notice that this lemma gives the well-known characterisation of $\mathsf{SN}^{\to}$:
$M \in \mathsf{SN}^{\to}$ if and only if $\forall N \in {\to}(M), N \in \mathsf{SN}^{\to}$.

Now we refine the induction principle immediately contained in the definition of normalisation by relaxing the requirement that the predicate should be patriarchal or paternal:

Theorem 18 (Induction principle) *Given a predicate P on $\mathcal{A}$,*

1. *Suppose* $\forall M \in \mathsf{SN}^{\to}, (\forall N \in {\to}(M), P(N)) \Rightarrow P(M)$.
   Then $\forall M \in \mathsf{SN}^{\to}, P(M)$.

2. *Suppose* $\forall M \in \mathsf{WN}^{\to}, (M \in \mathsf{nf}^{\to} \vee \exists N \in {\to}(M), P(N)) \Rightarrow P(M)$.
   Then $\forall M \in \mathsf{WN}^{\to}, P(M)$.

When we use this theorem to prove a statement $P(M)$ for all M in $\mathsf{SN}^{\to}$ (resp. $\mathsf{WN}^{\to}$), we just add $(\forall N \in {\to}(M), P(N))$ (resp. $M \in \mathsf{nf}^{\to} \vee \exists N \in {\to}(M), P(N)$) to the assumptions, which we call the induction hypothesis.

We say that we prove the statement by induction in $\mathsf{SN}^{\to}$ *(resp.* in $\mathsf{WN}^{\to}$*).*

Proof:

1. We prove that $\mathcal{T} = \{M \in \mathcal{A} |\ M \in \mathsf{SN}^{\to} \Rightarrow P(M)\}$ is patriarchal.
   Let $N \in \mathcal{A}$ such that ${\to}(N) \subseteq \mathcal{T}$. We want to prove that $N \in \mathcal{T}$:
   Suppose that $N \in \mathsf{SN}^{\to}$. By Lemma 17 we get that $\forall R \in {\to}(N), R \in \mathsf{SN}^{\to}$.
   By definition of $\mathcal{T}$ we then get $\forall R \in {\to}(N), P(R)$. From the main hypothesis we get $P(N)$. Hence, we have shown $N \in \mathcal{T}$.
   Now by definition of $M \in \mathsf{SN}^{\to}$, we get $M \in \mathcal{T}$, which can be simplified as $P(M)$ as required.

2. We prove that $\mathcal{T} = \{M \in \mathcal{A} |\ M \in \mathsf{WN}^{\to} \wedge P(M)\}$ is paternal.
   Let $N \in \mathsf{nf}^{\to} \subseteq \mathsf{WN}^{\to}$. By the main hypothesis we get $P(N)$.
   Now let $N \in {\to}^{-1}(\mathcal{T})$, that is, there is $R \in \mathcal{T}$ such that $N \to R$.
   We want to prove that $N \in \mathcal{T}$:
   By definition of $\mathcal{T}$, we have $R \in \mathsf{WN}^{\to}$, so $N \in \mathsf{WN}^{\to}$ (because $\mathsf{WN}^{\to}$ is paternal). We also have $P(R)$, so we can apply the main hypothesis to get $P(N)$. Hence, we have shown $N \in \mathcal{T}$.
   Now by definition of $M \in \mathsf{WN}^{\to}$, we get $M \in \mathcal{T}$, which can be simplified as $P(M)$ as required.

□

As a first application of the induction principle, we prove the following results:

Lemma 19 *$M \in \mathit{SN}^{\rightarrow}$ if and only if there is no infinite reduction sequence starting from M (classically, with the countable axiom of choice).*

Proof:

- *only if*: Consider the predicate $P(M)$ "having no infinite reduction sequence starting from M". We prove it by induction in $\mathsf{SN}^{\rightarrow}$. If M starts an infinite reduction sequence, then there is a $N \in \rightarrow(M)$ that also starts an infinite reduction sequence, which contradicts the induction hypothesis.
- *if*: Suppose $M \notin \mathsf{SN}^{\rightarrow}$. There is a patriarchal set $\mathcal{T}$ in which M is not. Hence, there is a $N \in \rightarrow(M)$ that is not in $\mathcal{T}$, and we re-iterate on it, creating an infinite reduction sequence. This uses the countable axiom of choice.

□

Lemma 20

1. *If $\rightarrow_1 \subseteq \rightarrow_2$, then $\mathit{nf}^{\rightarrow_1} \supseteq \mathit{nf}^{\rightarrow_2}$, $\mathit{WN}^{\rightarrow_1} \supseteq \mathit{WN}^{\rightarrow_2}$, $\mathit{SN}^{\rightarrow_1} \supseteq \mathit{SN}^{\rightarrow_2}$, and for all n, $\mathit{BN}_n^{\rightarrow_1} \supseteq \mathit{BN}_n^{\rightarrow_2}$.*
2. *$\mathit{nf}^{\rightarrow} = \mathit{nf}^{\rightarrow^+}$, $\mathit{WN}^{\rightarrow} = \mathit{WN}^{\rightarrow^+}$, $\mathit{SN}^{\rightarrow} = \mathit{SN}^{\rightarrow^+}$, and for all n, $\mathit{BN}_n^{\rightarrow^+} = \mathit{BN}_n^{\rightarrow}$.*

Proof:

1. By expanding the definitions.
2. For each statement, the right-to-left inclusion is a corollary of point 1.
   For the first statement, it remains to prove that $\mathsf{nf}^{\rightarrow} \subseteq \mathsf{nf}^{\rightarrow^+}$.
   Let $M \in \mathsf{nf}^{\rightarrow}$. By definition, $\rightarrow(M) = \emptyset$, so clearly $\rightarrow^+(M) = \emptyset$ as well.
   For the second statement, it remains to prove that $\mathsf{WN}^{\rightarrow} \subseteq \mathsf{WN}^{\rightarrow^+}$ which we do by induction in $\mathsf{WN}^{\rightarrow}$:
   Assume $M \in \mathsf{WN}^{\rightarrow}$ and the induction hypothesis that either $M \in \mathsf{nf}^{\rightarrow}$ or there is $N \in \rightarrow(M)$ such that $N \in \mathsf{WN}^{\rightarrow^+}$. In the former case, we have $M \in \mathsf{nf}^{\rightarrow} = \mathsf{nf}^{\rightarrow^+}$ and $\mathsf{nf}^{\rightarrow^+} \subseteq \mathsf{WN}^{\rightarrow^+}$. In the latter case, we have $N \in \rightarrow^+(M)$. Because of Lemma 17, $\mathsf{WN}^{\rightarrow^+}$ is stable by $\mathsf{WN}^{\rightarrow^{+-1}}$, and hence $M \in \mathsf{WN}^{\rightarrow^+}$.
   For the third statement, it remains to prove that $\mathsf{SN}^{\rightarrow} \subseteq \mathsf{SN}^{\rightarrow^+}$. We prove the stronger statement that $\forall M \in \mathsf{SN}^{\rightarrow}$, $\rightarrow^*(M) \subseteq \mathsf{SN}^{\rightarrow^+}$ by induction in $\mathsf{SN}^{\rightarrow}$: assume $M \in \mathsf{SN}^{\rightarrow}$ and the induction hypothesis $\forall N \in \rightarrow(M)$, $\rightarrow^*(N) \subseteq \mathsf{SN}^{\rightarrow^+}$. Clearly, $\rightarrow^+(M) \subseteq \mathsf{SN}^{\rightarrow^+}$. Because of

Lemma 17, $\mathsf{SN}^{\rightarrow^+}$ is $\rightarrow^+$-patriarchal, so $M \in \mathsf{SN}^{\rightarrow^+}$, and hence $\rightarrow^*(M) \subseteq \mathsf{SN}^{\rightarrow^+}$.
The statement $\mathsf{BN}_n^{\rightarrow} \subseteq \mathsf{BN}_n^{\rightarrow^+}$ can easily be proved by induction on n.

□

Notice that this result enables us to use a stronger induction principle: in order to prove $\forall M \in \mathsf{SN}^{\rightarrow}, P(M)$, it now suffices to prove

$$\forall M \in \mathsf{SN}^{\rightarrow}, (\forall N \in \rightarrow^+(M), P(N)) \Rightarrow P(M)$$

This induction principle is called the *transitive induction in* $\mathsf{SN}^{\rightarrow}$.

Theorem 21 (Strong normalisation of disjoint union)
Suppose that $(\mathcal{A}_i)_{i\in I}$ is a family of disjoint sets on some index set I, each being equipped with a reduction relation $\rightarrow_i$, and consider the reduction relation $\rightarrow := \bigcup_{i\in I} \rightarrow_i$ on $\bigcup_{i\in I} \mathcal{A}_i$.
We have $\bigcup_{i\in I} \mathsf{SN}^{\rightarrow_i} \subseteq \mathsf{SN}^{\rightarrow}$.

Proof: It suffices to prove that for all $j \in I$, $\mathsf{SN}^{\rightarrow_j} \subseteq \mathsf{SN}^{\rightarrow}$, which we do by induction in $\mathsf{SN}^{\rightarrow_j}$. Assume $M \in \mathsf{SN}^{\rightarrow_j}$ and assume the induction hypothesis $\rightarrow_j(M) \subseteq \mathsf{SN}^{\rightarrow}$. We must prove $M \in \mathsf{SN}^{\rightarrow}$, so it suffices to prove that for all N such that $M \rightarrow N$ we have $N \in \mathsf{SN}^{\rightarrow}$. By definition of the disjoint union, since $M \in \mathcal{A}_i$, all such N are in $\rightarrow_j(M)$ so we can apply the induction hypothesis. □

1.2.2 Termination by simulation

Now that we have established an induction principle on strongly normalising elements, the question arises of how we can prove strong normalisation. In this subsection we re-establish in our framework the well-known technique of simulation, which can be found for instance in [BN98]. The first technique to prove that a reduction relation on the set $\mathcal{A}$ terminates consists in simulating it (in the sense of Definition 6) in another set $\mathcal{B}$ equipped with its own reduction relation known to be terminating.

The mapping from $\mathcal{A}$ to $\mathcal{B}$ is sometimes called the *measure function* or the *weight function*, but Definition 6 generalises the concept to an arbitrary relation between $\mathcal{A}$ and $\mathcal{B}$, not necessarily functional. Similar results are to be found in [Che04], with the notions of *prosimulation*, *insertion*, and *repercussion*. The main point here is that the simulation technique is the typical example where the proof usually starts with "suppose an infinite reduction sequence" and ends with a contradiction. We show how the use of classical logic is completely unnecessary, provided that we use a constructive definition of SN such as ours.

Theorem 22 (Strong normalisation by strong simulation) *Let $\mathcal{R}$ be a relation between $\mathcal{A}$ and $\mathcal{B}$, equipped with the reduction relations $\rightarrow_{\mathcal{A}}$ and $\rightarrow_{\mathcal{B}}$.*
If $\rightarrow_{\mathcal{B}}$ strongly simulates $\rightarrow_{\mathcal{A}}$ through $\mathcal{R}$, then $\mathcal{R}^{-1}(\mathsf{SN}^{\rightarrow_{\mathcal{B}}}) \subseteq \mathsf{SN}^{\rightarrow_{\mathcal{A}}}$.

Proof: $\mathcal{R}^{-1}(\mathsf{SN}^{\to_{\mathcal{B}}}) \subseteq \mathsf{SN}^{\to_{\mathcal{A}}}$ can be reformulated as

$$\forall N \in \mathsf{SN}^{\to_{\mathcal{B}}}, \forall M \in \mathcal{A}, M\mathcal{R}N \Rightarrow M \in \mathsf{SN}^{\to_{\mathcal{A}}}$$

which we prove by transitive induction in $\mathsf{SN}^{\to_{\mathcal{B}}}$. Assume $N \in \mathsf{SN}^{\to_{\mathcal{B}}}$ and assume the induction hypothesis $\forall N' \in {\to_{\mathcal{B}}^{+}}(N), \forall M' \in \mathcal{A}, M'\mathcal{R}N' \Rightarrow M' \in \mathsf{SN}^{\to_{\mathcal{A}}}$. Now let $M \in \mathcal{A}$ such that $M\mathcal{R}N$. We want to prove that $M \in \mathsf{SN}^{\to_{\mathcal{A}}}$. It suffices to prove that $\forall M' \in {\to}(M), M' \in \mathsf{SN}^{\to_{\mathcal{A}}}$. Let M' be such that $M \to_{\mathcal{A}} M'$. The simulation hypothesis provides $N' \in {\to_{\mathcal{B}}^{+}}(N)$ such that $M'\mathcal{R}N'$. We apply the induction hypothesis on N', M' and get $M' \in \mathsf{SN}^{\to_{\mathcal{A}}}$ as required. We illustrate the technique in Fig. 1.5. □

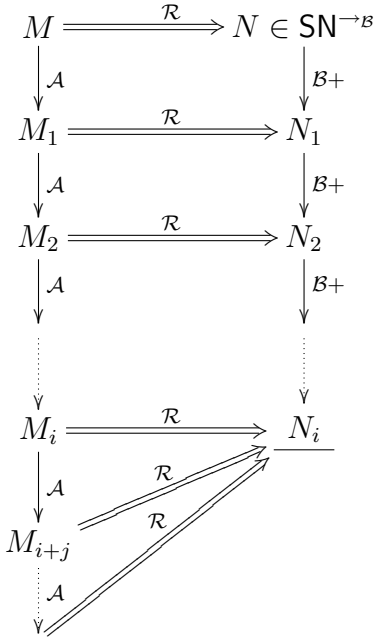


Figure 1.5: Deriving strong normalisation by simulation

1.2.3 Lexicographic termination

The simulation technique can be improved by another standard method. It consists of splitting the reduction relation into two parts, then proving that the first part is strongly simulated by a first auxiliary terminating relation, and then proving that the second part is weakly simulated by it and strongly simulated by a second auxiliary terminating relation. In some sense, the two auxiliary terminating relations act as measures that decrease lexicographically. We express this method in our constructive framework.

Lemma 23 *Given two reduction relations $\to_1$, $\to_2$, suppose that $\mathsf{SN}^{\to_1}$ is stable under $\to_2$. Then $\mathsf{SN}^{\to_1 \cup \to_2} = \mathsf{SN}^{\to_1^{*} \cdot \to_2} \cap \mathsf{SN}^{\to_1}$*

Proof: The left-to-right inclusion is an application of Theorem 22: $\to_1 \cup \to_2$ strongly simulates both $\to_1^* \cdot \to_2$ and $\to_1$ through Id.

Now we prove the right-to-left inclusion. We first prove the following lemma:

$$\forall M \in \mathsf{SN}^{\to_1}, (\to_1^* \cdot \to_2)(M) \subseteq \mathsf{SN}^{\to_1 \cup \to_2} \Rightarrow M \in \mathsf{SN}^{\to_1 \cup \to_2}$$

We do this by induction in $\mathsf{SN}^{\to_1}$, so not only assume $(\to_1^* \cdot \to_2)(M) \subseteq \mathsf{SN}^{\to_1 \cup \to_2}$, but also assume the induction hypothesis:
$\forall N \in \to_1(M),\ (\to_1^* \cdot \to_2)(N) \subseteq \mathsf{SN}^{\to_1 \cup \to_2} \Rightarrow N \in \mathsf{SN}^{\to_1 \cup \to_2}$.
We want to prove that $M \in \mathsf{SN}^{\to_1 \cup \to_2}$, so it suffices to prove that both $\forall N \in \to_2(M), N \in \mathsf{SN}^{\to_1 \cup \to_2}$ and $\forall N \in \to_1(M), N \in \mathsf{SN}^{\to_1 \cup \to_2}$. The former case is a particular case of the first hypothesis. The latter case would be provided by the second hypothesis (the induction hypothesis) if only we could prove that $(\to_1^* \cdot \to_2)(N) \subseteq \mathsf{SN}^{\to_1 \cup \to_2}$. But this is true because $(\to_1^* \cdot \to_2)(N) \subseteq (\to_1^* \cdot \to_2)(M)$ and the first hypothesis reapplies.

Now we prove

$$\forall M \in \mathsf{SN}^{\to_1^* \cdot \to_2}, M \in \mathsf{SN}^{\to_1} \Rightarrow M \in \mathsf{SN}^{\to_1 \cup \to_2}$$

We do this by induction in $\mathsf{SN}^{\to_1^* \cdot \to_2}$, so not only assume $M \in \mathsf{SN}^{\to_1}$, but also assume the induction hypothesis $\forall N \in (\to_1^* \cdot \to_2)(M), N \in \mathsf{SN}^{\to_1} \Rightarrow N \in \mathsf{SN}^{\to_1 \cup \to_2}$.
Now we can combine those two hypotheses, because we know that $\mathsf{SN}^{\to_1}$ is stable under $\to_2$: since $M \in \mathsf{SN}^{\to_1}$, we have $(\to_1^* \cdot \to_2)(M) \subseteq \mathsf{SN}^{\to_1}$, so that the induction hypothesis can be simplified in $\forall N \in (\to_1^* \cdot \to_2)(M), N \in \mathsf{SN}^{\to_1 \cup \to_2}$.
This gives us exactly the conditions to apply the above lemma to M. □

Definition 17 (Lexicographic reduction) Let $\mathcal{A}_1, \ldots, \mathcal{A}_n$ be sets, respectively equipped with the reduction relations $\to_{\mathcal{A}_1}, \ldots, \to_{\mathcal{A}_n}$.

For $1 \leq i \leq n$, let $\to_i$ be the reduction relation on $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ defined as follows:

$$(M_1, \ldots, M_n) \to_i (N_1, \ldots, N_n)$$

if $M_i \to_{\mathcal{A}_i} N_i$ and for all $1 \leq j < i$, $M_j = N_j$ and for all $i < j \leq n$, $N_j \in \mathsf{SN}^{\to_{\mathcal{A}_j}}$.
We define the *lexicographic reduction*

$$\to_{\mathsf{lex}} = \bigcup_{1 \leq i \leq n} \to_i$$

We sometimes write $\to_{\mathsf{lexx}}$ for $\to_{\mathsf{lex}}{}^+$, i.e. the transitive closure of $\to_{\mathsf{lex}}$.[9]

Corollary 24 (Lexicographic termination 1)

$$SN^{\to_{\mathcal{A}_1}} \times \cdots \times SN^{\to_{\mathcal{A}_n}} \subseteq SN^{\to_{lex}}$$

In particular, if $\to_{\mathcal{A}_i}$ is terminating on $\mathcal{A}_i$ for all $1 \leq i \leq n$, then $\to_{lex}$ is terminating on $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n$.

[9] This is the traditional lexicographic order, see e.g. [Ter03].

Proof: By induction on n: for $n = 1$, we conclude from $\rightarrow_{\mathcal{A}_1} = \rightarrow_1$.
Then notice that $\rightarrow_{\mathcal{A}_{n+1}}$ strongly simulates $\rightarrow_{n+1}$ through the $(n+1)^{th}$ projection. Hence, by Theorem 22, if $N_{n+1} \in \mathsf{SN}^{\rightarrow_{\mathcal{A}_{n+1}}}$ then $(N_1, \ldots, N_{n+1}) \in \mathsf{SN}^{\rightarrow_{n+1}}$, which we can also formulate as $\mathcal{A}_1 \times \cdots \times \mathcal{A}_n \times \mathsf{SN}^{\rightarrow_{\mathcal{A}_{n+1}}} \subseteq \mathsf{SN}^{\rightarrow_{n+1}}$.
A first consequence of this is $\mathsf{SN}^{\rightarrow_{\mathcal{A}_1}} \times \cdots \times \mathsf{SN}^{\rightarrow_{\mathcal{A}_{n+1}}} \subseteq \mathsf{SN}^{\rightarrow_{n+1}}$ (1). A second one is that $\mathsf{SN}^{\rightarrow_{n+1}}$ is stable under $\rightarrow_1 \cup \ldots \cup \rightarrow_n$ (2). Now notice that $\rightarrow_1 \cup \ldots \cup \rightarrow_n$ strongly simulates $\rightarrow_{n+1}^* \cdot (\rightarrow_1 \cup \ldots \cup \rightarrow_n)$ through the projection that drops the $(n+1)^{th}$ component. We can thus apply Theorem 22 to get $\mathsf{SN}^{\rightarrow_1 \cup \ldots \cup \rightarrow_n} \times \mathcal{A}_{n+1} \subseteq \mathsf{SN}^{\rightarrow_{n+1}^* \cdot (\rightarrow_1 \cup \ldots \cup \rightarrow_n)}$, which, combined with the induction hypothesis, gives $\mathsf{SN}^{\rightarrow_{\mathcal{A}_1}} \times \cdots \times \mathsf{SN}^{\rightarrow_{\mathcal{A}_{n+1}}} \subseteq \mathsf{SN}^{\rightarrow_{n+1}^* \cdot (\rightarrow_1 \cup \ldots \cup \rightarrow_n)}$ (3). From (1), (2), and (3) we can now conclude by using Lemma 23. □

Corollary 25 (Lexicographic termination 2) *Let $\mathcal{A}$ be a set equipped with a reduction relation $\rightarrow$.*

For each natural number n, let $\rightarrow_{lex n}$ be the lexicographic reduction on $\mathcal{A}^n$.

Consider the reduction relation $\rightarrow_{lex} = \bigcup_n \rightarrow_{lex n}$ on the disjoint union $\bigcup_n \mathcal{A}^n$.

$$\bigcup_n (SN^{\rightarrow})^n \subseteq SN^{\rightarrow_{lex}}$$

Proof: It suffices to combine Corollary 24 with Theorem 21. □

Corollary 26 (Lexicographic simulation technique) *Let $\rightarrow_{\mathcal{A}}$ and $\rightarrow'_{\mathcal{A}}$ be two reduction relations on $\mathcal{A}$, and $\rightarrow_{\mathcal{B}}$ be a reduction relation on $\mathcal{B}$. Suppose*

- *$\rightarrow'_{\mathcal{A}}$ is strongly simulated by $\rightarrow_{\mathcal{B}}$ through $\mathcal{R}$*
- *$\rightarrow_{\mathcal{A}}$ is weakly simulated by $\rightarrow_{\mathcal{B}}$ through $\mathcal{R}$*
- *$SN^{\rightarrow_{\mathcal{A}}} = \mathcal{A}$*

Then $\mathcal{R}^{-1}(SN^{\rightarrow_{\mathcal{B}}}) \subseteq SN^{\rightarrow_{\mathcal{A}} \cup \rightarrow'_{\mathcal{A}}}$.
(In other words, if $M \mathcal{R} N$ and $N \in SN^{\rightarrow_{\mathcal{B}}}$ then $M \in SN^{\rightarrow_{\mathcal{A}} \cup \rightarrow'_{\mathcal{A}}}$.)

Proof: Clearly, the reduction relation $\rightarrow_{\mathcal{A}}^* \cdot \rightarrow'_{\mathcal{A}}$ is strongly simulated by $\rightarrow_{\mathcal{B}}$ through $\mathcal{R}$, so that by Theorem 22 we get $\mathcal{R}^{-1}(\mathsf{SN}^{\rightarrow_{\mathcal{B}}}) \subseteq \mathsf{SN}^{\rightarrow_{\mathcal{A}}^* \cdot \rightarrow'_{\mathcal{A}}}$.
But $\mathsf{SN}^{\rightarrow_{\mathcal{A}}^* \cdot \rightarrow'_{\mathcal{A}}} = \mathsf{SN}^{\rightarrow_{\mathcal{A}}^* \cdot \rightarrow'_{\mathcal{A}}} \cap \mathsf{SN}^{\rightarrow_{\mathcal{A}}} = \mathsf{SN}^{\rightarrow_{\mathcal{A}} \cup \rightarrow'_{\mathcal{A}}}$, by the Lemma 23 (since $\mathsf{SN}^{\rightarrow_{\mathcal{A}}} = \mathcal{A}$ is obviously stable by $\rightarrow'_{\mathcal{A}}$). □

The intuitive idea behind this corollary is that after a certain number of $\rightarrow_{\mathcal{A}}$-steps and $\rightarrow'_{\mathcal{A}}$-steps, the only reductions in $\mathcal{A}$ that can take place are those that no longer modify the encoding in $\mathcal{B}$, that is, $\rightarrow_{\mathcal{A}}$-steps. Then it suffices to show that $\rightarrow_{\mathcal{A}}$ terminate, so that no infinite reduction sequence can start from M, as illustrated in Fig. 1.6.

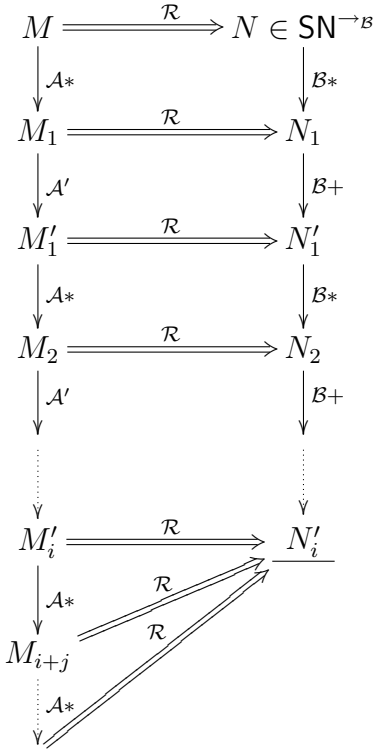


Figure 1.6: Deriving strong normalisation by lexicographic simulation

1.2.4 Multi-set termination

Now we define the notions of multi-sets their reductions [DM79, BN98]. We constructively prove their termination. Classical proofs of the result can also be found in [Ter03].

Definition 18 (Multi-Sets)

- Given a set $\mathcal{A}$, a *multi-set of* $\mathcal{A}$ is a total function from $\mathcal{A}$ to the natural numbers such that only a finite subset of elements are not mapped to 0.
- Note that for two multi-sets f and g, the function $f + g$ mapping any element M of $\mathcal{A}$ to $f(M) + g(M)$ is still a multi-set of $\mathcal{A}$ and is called the (multi-set) union of f and g. We also define the multi-set $f \setminus g$ as the function mapping each element $M \in \mathcal{A}$ to $\mathsf{max}(f(M) - g(M), 0)$.
- We define the multi-set $\{\!\{N_1, \ldots, N_n\}\!\}$ as $f_1 + \cdots + f_n$, where for all $1 \leq i \leq n$, f_i maps N_i to 1 and every other element to 0.
- We abusively write $M \in f$ if $f(M) \neq 0$.

Definition 19 (Multi-Set reduction relation) Given $\rightarrow$ is a reduction relation on $\mathcal{A}$, we define the multi-set reduction as follows:
if f and g are multi-sets of $\mathcal{A}$, we say that $f \rightarrow_{\mathsf{mul}} g$ if there is a M in $\mathcal{A}$ such that

$$\begin{cases} f(M) = g(M) + 1 \\ \forall N \in \mathcal{A}, f(N) < g(N) \Rightarrow M \rightarrow N \end{cases}$$

We sometimes write $\rightarrow_{\mathsf{mull}}$ for ${\rightarrow_{\mathsf{mul}}}^{+}$, i.e. the transitive closure of $\rightarrow_{\mathsf{mul}}$.[10]

Example 2 (Multi-set reduction) Considering multi-sets of natural numbers, for which the reduction relation is $>$, we have for instance $\{\!\{5,7,3,5,1,3\}\!\} >_{\mathsf{mul}} \{\!\{4,3,1,7,3,5,1,3\}\!\}$. In this case, the element M is 5, and an occurrence has been "replaced" by $4,3,1$, which are all smaller than 5.

In what follows we always assume that $\mathcal{A}$ is a set with a reduction relation $\rightarrow$.

Lemma 27 *If $f_1, \ldots, f_n, g$ are multi-sets of $\mathcal{A}$ and $f_1 + \cdots + f_n \rightarrow_{mul} g$ then there is $1 \leq i \leq n$ and a multi-set f_i' such that $f_i \rightarrow_{mul} f_i'$ and $f_1 + \cdots + f_{i-1} + f_i' + f_{i+1} + \cdots + f_n = g$.*

Proof: We know that there is a M in $\mathcal{A}$ such that

$$\begin{cases} f_1(M) + \cdots + f_n(M) = g(M) + 1 \\ \forall N \in \mathcal{A}, f_1(N) + \cdots + f_n(N) < g(N) \Rightarrow M \rightarrow N \end{cases}$$

An easy lexicographic induction on two natural numbers p and q shows that if $p + q > 0$ then $p > 0$ or $q > 0$. By induction on the natural number n, we extend this result: if $p_1 + \cdots + p_n > 0$ then $\exists i, p_i > 0$. We apply this result on $f_1(M) + \cdots + f_n(M)$ and get some $f_i(M) > 0$. Obviously there is a unique f_i' such that $f_1 + \cdots + f_{i-1} + f_i' + f_{i+1} + \cdots + f_n = g$, and we also get $f_i \rightarrow_{\mathsf{mul}} f_i'$. □

Definition 20 (Sets of multi-sets) Given two sets $\mathcal{N}$ and $\mathcal{N}'$ of multi-sets, we define $\mathcal{N} + \mathcal{N}'$ as $\{f + g \mid f \in \mathcal{N}, g \in \mathcal{N}'\}$.

We define for every M in $\mathcal{A}$ its *relative multi-sets* as all the multi-sets f of $\mathcal{A}$ such that if $N \in f$ then $M \rightarrow^* N$. We denote the set of relative multi-sets as $\mathcal{M}_M$.

Remark 28 Notice that for any $M \in \mathcal{A}$, $\mathcal{M}_M$ is stable under $\rightarrow_{\mathsf{mul}}$.

Lemma 29 *For all $M_1, \ldots, M_n$ in $\mathcal{A}$, if $\mathcal{M}_{M_1} \cup \ldots \cup \mathcal{M}_{M_n} \subseteq SN^{\rightarrow_{mul}}$ then $\mathcal{M}_{M_1} + \cdots + \mathcal{M}_{M_n} \subseteq SN^{\rightarrow_{mul}}$.*

Proof: Let $\mathcal{W}$ be the relation between $\mathcal{M}_{M_1} + \cdots + \mathcal{M}_{M_n}$ and $\mathcal{M}_{M_1} \times \cdots \times \mathcal{M}_{M_n}$ defined as: $f_1 + \cdots + f_n \mathcal{W} (f_1, \ldots, f_n)$ for all $f_1, \ldots, f_n$ in $\mathcal{M}_{M_1} \times \cdots \times \mathcal{M}_{M_n}$.

[10] This is the traditional multi-set order, see e.g. [Ter03].

We consider as a reduction relation on $\mathcal{M}_{M_1} \times \cdots \times \mathcal{M}_{M_n}$ the lexicographic composition of $\rightarrow_{\mathsf{mul}}$. We denote this reduction relation as $\rightarrow_{\mathsf{mullex}}$. By Corollary 24, we know that $\mathcal{M}_{M_1} \times \cdots \times \mathcal{M}_{M_n} \subseteq \mathsf{SN}^{\rightarrow_{\mathsf{mullex}}}$.
Hence, $\mathcal{W}^{-1}(\mathsf{SN}^{\rightarrow_{\mathsf{mullex}}}) = \mathcal{M}_{M_1} + \cdots + \mathcal{M}_{M_n}$.

Now we prove that $\mathcal{M}_{M_1} + \cdots + \mathcal{M}_{M_n}$ is stable by $\rightarrow_{\mathsf{mul}}$ and that $\rightarrow_{\mathsf{mullex}}$ strongly simulates $\rightarrow_{\mathsf{mul}}$ through $\mathcal{W}$. Suppose $f_1 + \cdots + f_n \rightarrow_{\mathsf{mul}} g$. By Lemma 27 we get a multi-set f_i' such that $f_1 + \cdots + f_{i-1} + f_i' + f_{i+1} + \cdots + f_n = g$ and $f_i \rightarrow_{\mathsf{mul}} f_i'$.
Hence, $f_i' \in \mathcal{M}_{M_i}$, so that $(f_1, \ldots, f_{i-1}, f_i', f_{i+1}, \cdots, f_n) \in \mathcal{M}_{M_1} \times \cdots \times \mathcal{M}_{M_n}$ and even $(f_1, \cdots, f_n) \rightarrow_{\mathsf{mullex}} (f_1, \ldots, f_{i-1}, f_i', f_{i+1}, \cdots, f_n)$.

By Theorem 22 we then get $\mathcal{W}^{-1}(\mathsf{SN}^{\rightarrow_{\mathsf{mullex}}}) \subseteq \mathsf{SN}^{\rightarrow_{\mathsf{mul}}}$, which concludes the proof because $\mathcal{W}^{-1}(\mathsf{SN}^{\rightarrow_{\mathsf{mullex}}}) = \mathcal{M}_{M_1} + \cdots + \mathcal{M}_{M_n}$. □

Lemma 30 $\forall M \in \mathsf{SN}^{\rightarrow}, \mathcal{M}_M \subseteq \mathsf{SN}^{\rightarrow_{mul}}$

Proof: By transitive induction in $\mathsf{SN}^{\rightarrow}$. Assume that $M \in \mathsf{SN}^{\rightarrow}$ and assume the induction hypothesis $\forall N \in \rightarrow^+(M), \mathcal{M}_N \subseteq \mathsf{SN}^{\rightarrow_{\mathsf{mul}}}$.

Let us split the reduction relation $\rightarrow_{\mathsf{mul}}$: if $f \rightarrow_{\mathsf{mul}} g$, let $f \rightarrow_{\mathsf{mul1}} g$ if $f(M) = g(M)$ and let $f \rightarrow_{\mathsf{mul2}} g$ if $f(M) > g(M)$. Clearly, if $f \rightarrow_{\mathsf{mul}} g$ then either $f \rightarrow_{\mathsf{mul1}} g$ or $f \rightarrow_{\mathsf{mul1}} g$. This is an intuitionistic implication since the equality of two natural numbers can be decided.

Now we prove that $\rightarrow_{\mathsf{mul1}}$ is terminating on $\mathcal{M}_M$.

Let $\mathcal{W}'$ be the following relation (actually, a function) between $\mathcal{M}_M$ to itself: for all f and g in $\mathcal{M}_M$, $f \mathcal{W} g$ if $g(M) = 0$ and for all $N \neq M$, $f(N) = g(N)$.

For a given $f \in \mathcal{M}_M$, let $N_1, \ldots, N_n$ be the elements of $\mathcal{A}$ that are not mapped to 0 by f and that are different from M. Since $f \in \mathcal{M}_M$, for all $1 \leq i \leq n$ we know $M \rightarrow^+ N_i$, and we also know that $\mathcal{W}'(f) \in \mathcal{M}_{N_1} + \cdots + \mathcal{M}_{N_n}$. Hence, we apply the induction hypothesis and Lemma 29 to get $\mathcal{M}_{N_1} + \cdots + \mathcal{M}_{N_n} \subseteq \mathsf{SN}^{\rightarrow_{\mathsf{mul}}}$. Hence, $\mathcal{W}'(f) \in \mathsf{SN}^{\rightarrow_{\mathsf{mul}}}$.

Now notice that $\rightarrow_{\mathsf{mul}}$ strongly simulates $\rightarrow_{\mathsf{mul1}}$ through $\mathcal{W}'$, so by Theorem 22, $f \in \mathsf{SN}^{\rightarrow_{\mathsf{mul1}}}$.

Now that we know that $\rightarrow'_{\mathsf{mul}}$ is terminating on $\mathcal{M}_M$, we notice that the decreasing order on natural numbers strongly simulates $\rightarrow_{\mathsf{mul2}}$ and weakly simulates $\rightarrow_{\mathsf{mul1}}$ through the function that maps every $f \in \mathcal{M}_M$ to the natural number $f(M)$.

Hence, we can apply Corollary 26 to get $\mathcal{M}_M \subseteq \mathsf{SN}^{\rightarrow_{\mathsf{mul}}}$. □

Corollary 31 (Multi-Set termination) *Let f be a multi-set of $\mathcal{A}$.*
If for every $M \in f$, $M \in \mathsf{SN}^{\rightarrow}$, then $f \in \mathsf{SN}^{\rightarrow_{mul}}$.

Proof: Let $M_1, \ldots, M_n$ be the elements of $\mathcal{A}$ that are not mapped to 0 by f. Clearly, $f \in \mathcal{M}_{M_1} + \cdots + \mathcal{M}_{M_n}$. By Lemma 30, $\mathcal{M}_{M_1} \cup \ldots \mathcal{M}_{M_n} \subseteq \mathsf{SN}^{\rightarrow_{\mathsf{mul}}}$, and by Lemma 29, $\mathcal{M}_{M_1} + \cdots + \mathcal{M}_{M_n} \subseteq \mathsf{SN}^{\rightarrow_{\mathsf{mul}}}$, so $f \in \mathsf{SN}^{\rightarrow_{\mathsf{mul}}}$. □

1.3 Higher-Order Calculi (HOC)

In this section we introduce *higher-order calculi*, in which the set $\mathcal{A}$ is recursively defined by a term syntax possibly involving *variable binding* and the reduction relation $\rightarrow$ is defined as a rewrite system. This section is intended to capture all the formalisms introduced in the rest of this thesis, which are quite numerous.

1.3.1 Introduction and literature

There are several ways to express higher-order calculi in a generic way, among which *Higher-Order Systems* (HRS) [Nip91], *Combinatory Reduction Systems* (CRS) [Klo80], *Expression Reduction Systems* (ERS) [Kha90], *Interaction Systems* (IS) [AL94], etc. These formalisms are presented in particular in [Ter03] with a comparative approach.

Here we choose a presentation of higher-order calculi of which traditional presentations of many calculi are direct instances. In this presentation, higher-order terms can be seen as those of HRS, i.e. the η-long β-normal forms of the simply-typed λ-calculus [Bar84] extended with constants (representing *term constructors*). Types are here called *syntactic categories* to avoid confusion with systems presented later.

The notion of reduction is given by rewriting. However, unlike the aforementioned formalisms, rewrite systems are considered not at the object-level but at the meta-level as a way to define a (reduction) relation (just like we can define a function by a set of equations treating different cases for its argument).

We thus use in rewrite systems the same meta-variables for terms as in the rest of the dissertation, i.e. those variables of the meta-level that we use to quantify over terms in statements and proofs. Meta-variables are convenient to describe higher-order grammars in BNF-format and allow the presentation of rewrite systems in a traditional way. They are similar to those of CRS and even more similar to those of ERS and IS (since they have no arity and are not applied). These formalisms internalise parts of the meta-level (such as rewrite systems with meta-variables) into the object-level.

The meta-level language for describing higher-order calculi can actually be considered on its own as an object, and this is for example what we do in section 1.3.3 to state our conventions for dropping the side-conditions that are needed in first-order logic to deal with α-conversion and avoid variable capture and liberation.

Particular definitions of reduction relations can thus be encoded at the object-level in the formalisms mentioned earlier, so that we can use established results such as confluence of orthogonal systems. Rewriting in IS is constrained by the notions of constructor and destructor (here we call term constructor any constant) and that in ERS is more general, but it is into HRS that we explicitly give an encoding (section 1.3.4), both because we want to use its intrinsic typing

system and because everything is expressed within the object-level (thus avoiding confusion between the meta-level and the extra elements added to the object-level to mimic the meta-level —such as meta-variables in the same syntax as variables).

1.3.2 Syntax of HOC

When representing labelled trees as strings, we shall use parentheses to remove ambiguities.

Definition 21 (Syntactic categories)

- Given a set $\mathcal{SC}$ of elements called *basic syntactic categories*, the set of *syntactic categories* is the set of binary trees whose internal nodes are labelled with the symbol $\hookrightarrow$ and whose leaves are labelled with basic syntactic categories.

  In other words, syntactic categories are given by the following syntax:

  $$\mathcal{C}, \mathcal{C}' ::= \mathcal{T} \in \mathcal{SC} \mid \mathcal{C} \hookrightarrow \mathcal{C}'$$

  We consider that $\hookrightarrow$ is associative to the right, i.e. we abbreviate $\mathcal{C}_1 \hookrightarrow (\mathcal{C}_2 \hookrightarrow \mathcal{C}_3)$ as $\mathcal{C}_1 \hookrightarrow \mathcal{C}_2 \hookrightarrow \mathcal{C}_3$.

- The *arity* of a syntactic category $\mathcal{C}$ is defined by induction on $\mathcal{C}$ as follows:

  $$\boxed{\begin{array}{lcll} \mathsf{arity}(\mathcal{T}) & := & 0 & \text{if } \mathcal{T} \in \mathcal{SC} \\ \mathsf{arity}(\mathcal{C}_1 \hookrightarrow \mathcal{C}_2) & := & 1 + \mathsf{arity}(\mathcal{C}_2) & \end{array}}$$

- The *order* of a syntactic category $\mathcal{C}$ is defined by induction on $\mathcal{C}$ as follows:

  $$\boxed{\begin{array}{lcll} \mathsf{order}(\mathcal{T}) & := & 0 & \text{if } \mathcal{T} \in \mathcal{SC} \\ \mathsf{order}(\mathcal{C}_1 \hookrightarrow \mathcal{C}_2) & := & \mathsf{max}(\mathsf{order}(\mathcal{C}_1) + 1, \mathsf{order}(\mathcal{C}_2)) & \end{array}}$$

A *Higher-Order Calculus* (*HOC*) is given by a grammar, as defined in Definition 22, and a reduction relation on terms, which are defined in Definition 28.

Definition 22 (Grammar of HOC) The *grammar of an HOC* is given by

- a finite or denumerable set of basic syntactic categories;
- for each syntactic category $\mathcal{C}$,
  - a set of elements called *variables* and ranged over by x (sometimes written $x \wr \mathcal{C}$ to indicate the category); $\mathcal{C}$ is said to be *variable-free* if this set is empty, otherwise it is *with variables*,

– a set of elements called *term constructors* (or just *constructors*), denoted $\mathsf{c} \wr \mathcal{C}$.

A basic syntactic category $\mathcal{T}$ with variables and such that for no $\mathcal{C}_1, \ldots, \mathcal{C}_n$ there is a term constructor $\mathsf{c} \wr \mathcal{C}_1 \hookrightarrow \cdots \hookrightarrow \mathcal{C}_n \hookrightarrow \mathcal{T}$ is called a *variable category*.

We sometimes write f for either a variable or a term constructor. The *arity* of $f \wr \mathcal{C}$ is $\mathsf{arity}(\mathcal{C})$.

The aforementioned sets of variables are assumed to be pairwise disjoint. We also fix a total order on the set of all variables.

Definition 23 (Syntactic terms of HOC)

- In such an HOC, the *syntactic terms* of a syntactic category are trees whose nodes are labelled with either variables, term constructors or a dot with a bracketed variable, and respecting the syntactic categories in the sense specified by the following two rules:

  For each variable or term constructor $f \wr \mathcal{C}_1 \hookrightarrow \cdots \hookrightarrow \mathcal{C}_n \hookrightarrow \mathcal{T}$ (with $n \geq 0$ and $\mathcal{T}$ being a basic syntactic category),

  $$\frac{(S_i \wr \mathcal{C}_i)_{1 \leq i \leq n}}{f(S_1, \ldots, S_n) \wr \mathcal{T}}$$

  For each variable $x \wr \mathcal{C}_1$,

  $$\frac{S \wr \mathcal{C}_2}{[x].S \wr \mathcal{C}_1 \hookrightarrow \mathcal{C}_2}$$

- The *height* and *size* of a syntactic term are its height and size as a tree. The notion of sub-tree (resp. strict sub-tree) provides the notion of *sub-syntactic term* (resp. *strict sub-syntactic term*). If S is a sub-syntactic term (resp. strict sub-syntactic term) of S' we write $S \sqsubseteq S'$ (resp. $S \sqsubset S'$), and we write $\sqsupseteq$ for $\sqsubseteq^{-1}$ (resp. $\sqsupset$ for $\sqsubset^{-1}$).

- The terminating relation $\sqsupset$ provides a notion of induction on syntactic terms.

- We abbreviate $f()$ as f for a variable or term constructor $f \wr \mathcal{T}$.

- The notion of equality between syntactic terms is sometimes called *syntactic equality*.

Example 3 (λ-calculus as an HOC) We can express the syntax of λ-calculus as an HOC. There is one basic syntactic category $\mathcal{T}$ with variables, and there are two term constructors: the abstraction $\lambda : (\mathcal{T} \hookrightarrow \mathcal{T}) \hookrightarrow \mathcal{T}$ and the application, of the syntactic category $\mathcal{T} \hookrightarrow \mathcal{T} \hookrightarrow \mathcal{T}$.

It will often be necessary to rename variables, and a elegant way of doing it is by swapping two variables, a notion borrowed from nominal logic [Pit03].

Definition 24 (Swapping) The *swapping* of two variables x and y in a syntactic term S is defined by induction on S, using the swapping of two variables x and y on a variable z:

$(x\ y)x$	$:=$	y	
$(x\ y)y$	$:=$	x	
$(x\ y)z$	$:=$	z	$z \neq x, z \neq y$
$(x\ y)(z(S_1, \ldots, S_n))$	$:=$	$((x\ y)z)((x\ y)S_1, \ldots, (x\ y)S_n)$	
$(x\ y)(\mathsf{c}(S_1, \ldots, S_n))$	$:=$	$\mathsf{c}((x\ y)S_1, \ldots, (x\ y)S_n)$	
$(x\ y)([z].S)$	$:=$	$[(x\ y)z].(x\ y)S$	

Remark 32 Swapping preserves the height and size of syntactic terms.

Definition 25 (Syntactic terms & relations) A relation $\mathcal{R}$ between syntactic terms *respects syntactic categories* if whenever $S\mathcal{R}S'$ then S and S' belong to the same syntactic category.

Definition 26 (Free variables) The set of *free $\mathcal{C}$-variables* of a syntactic term S, denoted $\mathsf{FV}_{\mathcal{C}}(S)$ is defined inductively as follows:

$\mathsf{FV}_{\mathcal{C}}(x(S_1, \ldots, S_n))$	$:=$	$\{x\} \cup \bigcup_{1 \leq i \leq n} \mathsf{FV}_{\mathcal{C}}(S_i)$	if $x \wr \mathcal{C}$
$\mathsf{FV}_{\mathcal{C}}(x(S_1, \ldots, S_n))$	$:=$	$\bigcup_{1 \leq i \leq n} \mathsf{FV}_{\mathcal{C}}(S_i)$	if not
$\mathsf{FV}_{\mathcal{C}}(\mathsf{c}(S_1, \ldots, S_n))$	$:=$	$\bigcup_{1 \leq i \leq n} \mathsf{FV}_{\mathcal{C}}(S_i)$	
$\mathsf{FV}_{\mathcal{C}}([x].S)$	$:=$	$\mathsf{FV}_{\mathcal{C}}(S) \setminus x$	

We also write $\mathsf{FV}(S)$ for $\bigcup_{\mathcal{C}} \mathsf{FV}_{\mathcal{C}}(S)$ ($\mathcal{C}$ ranging over categories with variables) or when $\mathcal{C}$ is unique or clear from context.

The construct $[x].S$ gives a mechanism for binding and induces a notion of α-equivalence, for which we follow Kahrs' definition [Kah92]. Again, this definition is an example of an inference structure.

Definition 27 (α-equivalence)

- Two syntactic terms S, S' are *α-equivalent*, denoted $S{=_\alpha}S'$, if the judgement $\vdash S =_\alpha S'$ is derivable in the following inference structure for judgements of the form $\Gamma \vdash S =_\alpha S'$ or $\Gamma \vdash x - y$ (where Γ is a list of pairs of variables written $x' - y'$):

$$\frac{}{\vdash x - x} \qquad \frac{}{\Gamma, x - y \vdash x - y} \qquad \frac{\Gamma \vdash x' - y'}{\Gamma, x - y \vdash x' - y'}\, x \neq x' \wedge y \neq y'$$

$$\frac{\Gamma \vdash x - y \quad (\Gamma \vdash S_i =_\alpha S'_i)_{1 \leq i \leq n}}{\Gamma \vdash x(S_1, \ldots, S_n) =_\alpha y(S'_1, \ldots, S'_n)} \qquad \frac{(\Gamma \vdash S_i =_\alpha S'_i)_{1 \leq i \leq n}}{\Gamma \vdash \mathsf{c}(S_1, \ldots, S_n) =_\alpha \mathsf{c}(S'_1, \ldots, S'_n)}$$

$$\frac{\Gamma, x - y \vdash S_1 =_\alpha S_2}{\Gamma \vdash [x].S_1 =_\alpha [y].S_2}\, x \wr \mathcal{C} \wedge y \wr \mathcal{C}$$

- A relation $\mathcal{R}$ between syntactic terms is *compatible with α-equivalence* if $(=_\alpha \cdot \mathcal{R} \cdot =_\alpha) \subseteq \mathcal{R}$.
- A function f from syntactic terms to a set $\mathcal{A}$ is *compatible with α-equivalence* if α-equivalent terms are mapped to the same element of $\mathcal{A}$.

Remark 33

1. If $x \notin \mathsf{FV}(S)$ then $[y].S =_\alpha [x].(x\ y)S$.
2. α-equivalence is an equivalence relation that respects syntactic categories.

Definition 28 (Terms of HOC)

- The *terms* of HOC are simply the α-equivalence classes of syntactic terms.[11]
- We now introduce notations whose meanings are defined inductively as terms:
  - $x.M$ denotes the α-equivalence class of $[x].S$ if M denotes the α-equivalence class of S,
  - $f(M_1, \ldots, M_n)$ denotes the α-equivalence class of $f(S_1, \ldots, S_n)$ if each M_i denotes the α-equivalence class of S_i and f is a variable or a term constructor. (Again we abbreviate $f()$ as f for a variable or term constructor $f \wr \mathcal{T}$.)

  A term $x.M$ is called *abstraction* or *binder* on x with *scope* M.

[11] The terms of HOC can be seen as the η-long β-normal forms of the simply-typed λ-calculus (extended with constants corresponding to term constructors), i.e. as the terms of HRS, in effect. (Note however in Definition 22 that we can prevent some syntactic categories from being inhabited by variables.)

Note that when there are no binders, a syntactic term and its α-equivalence class are denoted by the same expression, in other words there is an overloaded notation for a syntactic term and the singleton set that contains it.

- The syntactic category of a term is the syntactic category of any/all[12] of its representatives, and again we use the notation $M \wr \mathcal{C}$.[13] We identify a syntactic category $\mathcal{C}$ with the set of terms $\{M \mid M \wr \mathcal{C}\}$.

Remark 34

1. The function that maps a syntactic term S to $\mathsf{FV}_{\mathcal{C}}(S)$ is compatible with α-conversion, so we can now use the notations $\mathsf{FV}_{\mathcal{C}}(M)$ for a term M.[14] Again, we also write $\mathsf{FV}(M)$ for $\bigcup_{\mathcal{C}} \mathsf{FV}_{\mathcal{C}}(M)$ ($\mathcal{C}$ ranging over categories with variables) or when $\mathcal{C}$ is unique or clear from context.

2. The function that maps a syntactic term S to $\mathsf{FV}(S)$, to $(x\ y)S$, so we can now use the notation $(x\ y)M$ for a term M.[15] Also, if $x \notin \mathsf{FV}(M)$ then $y.M =_\alpha x.(x\ y)M$.

3. The height and size of a syntactic term are compatible with α-conversion, so we can now talk about the height and size of a term.

Definition 29 (Closed terms) Given a syntactic category $\mathcal{C}$ with variables, we say that a term M is *$\mathcal{C}$-closed* if $\mathsf{FV}_{\mathcal{C}}(M) = \emptyset$, and that it is *closed* if $\mathsf{FV}(M) = \emptyset$.

Definition 30 (Sub-terms) The relations $\sqsubseteq$ and $\sqsubset$ on syntactic terms are *not* compatible with α-conversion, however $\sqsubseteq \cdot =_\alpha$ and $\sqsubset \cdot =_\alpha$ are. This provides two relations on terms, which we call the *sub-term* (resp. *strict sub-term*) relation

[12] This is the same because of Remark 33

[13] Hence, $x.M \wr \mathcal{C}_1 \hookrightarrow \mathcal{C}_2$ if and only if $x \wr \mathcal{C}_1$ and $M \wr \mathcal{C}_2$, and for all variable or term constructor f, $f(S_1, \ldots, S_n) \wr \mathcal{T}$ if and only if $f \wr \mathcal{C}_1 \hookrightarrow \cdots \hookrightarrow \mathcal{C}_n \hookrightarrow \mathcal{T}$ and for all i, $S_i \wr \mathcal{C}_i$.

[14] Hence, the free variables of $\mathcal{C}$ of a term M, denoted $\mathsf{FV}_{\mathcal{C}}(M)$, satisfy the following equations:

$$\begin{array}{lcll} \mathsf{FV}_{\mathcal{C}}(x(M_1, \ldots, M_n)) & = & \{x\} \cup \bigcup_{1 \le i \le n} \mathsf{FV}_{\mathcal{C}}(M_i) & \text{if } x \wr \mathcal{C} \\ \mathsf{FV}_{\mathcal{C}}(x(M_1, \ldots, M_n)) & = & \bigcup_{1 \le i \le n} \mathsf{FV}_{\mathcal{C}}(M_i) & \text{if not} \\ \mathsf{FV}_{\mathcal{C}}(\mathsf{c}(M_1, \ldots, M_n)) & = & \bigcup_{1 \le i \le n} \mathsf{FV}_{\mathcal{C}}(M_i) & \\ \mathsf{FV}_{\mathcal{C}}(x.M) & = & \mathsf{FV}_{\mathcal{C}}(M) \setminus x & \end{array}$$

[15] Hence, the swapping of two variables x and y on a term M satisfies the following equalities:

$$\begin{array}{ll} (x\ y)(z(M_1, \ldots, M_n)) & = ((x\ y)z)((x\ y)M_1, \ldots, (x\ y)M_n) \\ (x\ y)(\mathsf{c}(M_1, \ldots, M_n)) & = \mathsf{c}((x\ y)M_1, \ldots, (x\ y)M_n) \\ (x\ y)(z.M) & = (x\ y)z.(x\ y)M \end{array}$$

and denote $\sqsubseteq$ (resp. $\sqsubset$) as well. Again we write $\sqsupseteq$ for $\sqsubseteq^{-1}$ (resp. $\sqsupset$ for $\sqsubset^{-1}$). For instance, y is a sub-term of $x.x$.

Remark 35 By definition of terms, $\mathsf{SN}^{\sqsupset}$ is equal to the set of all terms of an HOC, and $\sqsupset$ thus provides a notion of induction on terms, called *(structural) induction* (on terms).

Definition 31 (Terms & relations)

- If a relation between syntactic term (resp. a function on syntactic terms) is compatible with α-equivalence, then it provides a relation between terms (resp. function on terms), usually denoted the same way.
- Conversely, a relation between terms provides a relation between syntactic terms that is compatible with α-equivalence.
- A relation $\mathcal{R}$ between terms *respects syntactic categories* if the relation it provides on syntactic terms does (i.e. if whenever $M\mathcal{R}M'$ then M and M' belong to the same syntactic category).
- Let $\mathcal{R}$ be a relation between terms respecting syntactic categories. The *contextual closure* $\mathbf{cc}\mathcal{R}$ of $\mathcal{R}$ is the set of pairs derivable in the inference structure given by the following rules:

$$\frac{M\ \mathcal{R}\ M'}{M\ \mathbf{cc}\mathcal{R}\ M'} \qquad \frac{M\ \mathbf{cc}\mathcal{R}\ M'}{x.M\ \mathbf{cc}\mathcal{R}\ x.M'}$$

$$\frac{\exists i_0, M_{i_0}\ \mathbf{cc}\mathcal{R}\ M'_{i_0} \wedge \forall i \neq i_0, M_i = M'_i}{f(M_1, \ldots, M_n)\ \mathbf{cc}\mathcal{R}\ f(M'_1, \ldots, M'_n)}$$

if f is a variable or a term constructor.

- $\mathcal{R}$ is *context-closed* if $\mathcal{R} = \mathbf{cc}\mathcal{R}$. A *congruence* is an equivalence relation between terms that is context-closed, e.g. syntactic equality.

Remark 36 The contextual closure of the union is the union of the contextual closures. In other words if $\mathcal{R}$ and $\mathcal{R}'$ are relations on terms that respects syntactic categories then $\mathbf{cc}(\mathcal{R} \cup \mathcal{R}') = \mathbf{cc}\mathcal{R} \cup \mathbf{cc}\mathcal{R}'$.

It will later be useful to have a common notation to extract the variables of various structures:

Definition 32 (Support)

- The *support* of a variable is itself as a singleton: $\mathsf{Support}(x) := \{x\}$.

- The *support* of a term is its set of free variables: $\mathsf{Support}(M) := \mathsf{FV}(M)$.
- The *support* of a set of variables or terms is the union of the supports of its elements: $\mathsf{Support}(\mathcal{S}) := \bigcup_{h\in\mathcal{S}} \mathsf{Support}(h)$
- The *support* of a multi-set or a list of variables or terms is also the union of the supports of its elements.

We also generalise the notion of swapping to these structures:

Definition 33 (Generalised swapping) If $\mathcal{S}$ is a set, a multi-set or a list of variables, syntactic terms or terms, then $(x\ y)\mathcal{S}$ denotes the same set, multi-set or list but with each of its elements h changed to $(x\ y)h$. This can in fact be generalised to any structure whose elementary components are variables, syntactic terms or terms.

Generalised swapping provides the notion of equivariance:

Definition 34 (Equivariance) A set $\mathcal{S}$ of elements that can be subject to (generalised) swapping is *equivariant* if for any variables x, y we have $(x\ y)\mathcal{S} \subseteq \mathcal{S}$.

Example 4 (Equivariance) The relations $\{(x, M) \mid x \in \mathsf{FV}(M)\}$ and $\{(x, M) \mid x \notin \mathsf{FV}(M)\}$ are equivariant.

Remark 37 Saying that the function on syntactic terms $S \mapsto (x\ y)S$ is compatible with α-equivalence is the same as saying that α-equivalence (i.e. the set of pairs of syntactic terms that are α-equivalent) is equivariant.

Lemma 38 (Admissibility of swapping) *Suppose we have an inference structure whose judgements $h, h_1, h_2, \ldots$ can be subject to (generalised) swapping (variables, terms, sets, lists of them,...). Assume that the inference structure is equivariant.*

If there is a derivation from $\mathcal{A}$ to h there is one of same height from $(x\ y)\mathcal{A}$ to $(x\ y)h$.

In particular, $\dfrac{h}{(x\ y)h}$ is admissible.

Proof: By induction on the height of derivations. □

1.3.3 Meta-level & conventions about variable binding

The meta-language is based on multi-sorted first-order logic, and again, we replace the terminology of sorts by that of syntactic categories (of the meta-level). As in multi-sorted first-order logic, expressions are based on notational constants taking n arguments ($n \geq 0$), each being of some syntactic category.

As in multi-sorted first-order logic, the meta-level has no higher-order syntactic category; however some first-order notations *intend* some variable binding:

- to denote a binder of the object level,
- to denote a construction in which some variable name does not matter, as for the (implicit) substitution $\left\{{}^{N}\!/_{x}\right\}M$ that we define in this section.

Both kinds of bindings introduce the problem of variable capture and liberation. We might want to write for instance

- a rule like η-reduction in λ-calculus: $\lambda x.M\ x \to M$, or the propagation of an explicit substitution through an abstraction in the $\lambda\mathsf{x}$-calculus [BR95]: $\langle P/y\rangle\lambda x.M \to \lambda x.\langle P/y\rangle M$ (all bindings from the object-level)
- the substitution lemma $\left\{{}^{P}\!/_{y}\right\}\left\{{}^{N}\!/_{x}\right\}M = \left\{\left\{{}^{P}\!/_{y}\right\}{}^{N}\!/_{x}\right\}\left\{{}^{P}\!/_{y}\right\}M$ (all bindings from the meta-level)
- the case of the abstraction for the definition of (implicit) substitution: $\left\{{}^{P}\!/_{y}\right\}(x.M) = x.\left\{{}^{P}\!/_{y}\right\}M$ (interaction between object-level binding and meta-level binding)

The side-conditions avoiding variable capture and liberation are $x \notin \mathsf{FV}(M)$ for η-reduction, and $x \notin \mathsf{FV}(P)$ and $x \neq y$ for the rule of $\lambda\mathsf{x}$, the substitution lemma, and the definition of implicit substitution. In all cases we want a safe way to drop these side-conditions, because they can be mechanically recovered just by looking at the above expressions, and this is the main point of this section. But in order to define this mechanical process, information about the intended variable bindings must be available, so we slightly enrich the sorting of multi-sorted first-order logic so that it bears this information.

The grammar for syntactic categories of the meta-level must cover every kind of notation used in this dissertation. First-order notations are standard to deal with, but expressions of the meta-level might intend not only unary bindings (as in the example of the substitution) but also n-ary bindings, i.e. bindings of several variables in one construct. For example in Chapter 5 we use lists. Lists can be used as binders in a notation of λ-calculus like $\lambda\Pi.M$ that stands for $\lambda x_1.\ldots.\lambda x_n.M$ if Π is the list $x_1,\ldots,x_n$. In this case writing $\left\{{}^{N}\!/_{x}\right\}(\lambda\Pi.M) = \lambda\Pi.\left\{{}^{N}\!/_{x}\right\}M$ should generate the side-conditions $\mathsf{Dom}(\Pi) \cap \mathsf{FV}(N) = \emptyset$ and $x \notin \mathsf{Dom}(\Pi)$, which are more complex than the side-conditions of the previous examples.

Definition 35 (Syntactic categories of the meta-level)
The meta-level uses the basic syntactic categories of the object-level but new ones can also be used, given in a set $\mathcal{SC}^{\mathcal{M}}$. Syntactic categories of the meta-level are given by the following grammar:

$$\mathbb{S} ::= \mathbb{U} \in \mathcal{SC}^{\mathcal{M}} \mid \mathbb{B} \mid \mathbb{P}$$

where $\mathbb{P}$ ranges over the *syntactic categories of super-bound expressions* and $\mathbb{B}$ ranges over the *syntactic categories of binders*, given as follows:

$$\begin{array}{lcl} \mathbb{P} & ::= & \mathbb{T} \mid \mathbb{B} \times \mathbb{P} \\ \mathbb{T} & ::= & \mathcal{T} \in \mathcal{SC} \mid \mathsf{V}_{\mathcal{C}} \times \mathbb{T} \\ \mathbb{B} & ::= & \mathsf{V}_{\mathcal{C}} \mid \mathsf{Lists}_{\mathcal{C}} \mid \mathsf{Multisets}_{\mathcal{C}} \mid \mathsf{Sets}_{\mathcal{C}} \mid \ldots \end{array}$$

The syntactic categories ranged over by $\mathbb{T}$ are *syntactic categories of term-expressions*, those expressions denoting terms. Expressions of a syntactic category $\mathbb{P}$, called *super-bound expressions* are those involving the complex binders mentioned earlier such as lists of variables.

Expressions in $\mathbb{B} \times \mathbb{P}$ will be pairs, with the first component representing some binders of the object-level and the second component representing their scope. We must be able to apply the support extractor $\mathsf{Support}$ to expressions representing binders to produce the set of variables that they bind.

Hence, these expressions can be in $\mathsf{V}_{\mathcal{C}}, \mathsf{Lists}_{\mathcal{C}}, \mathsf{Multisets}_{\mathcal{C}}, \mathsf{Sets}_{\mathcal{C}}$, i.e. respectively the syntactic categories of variables, lists of variables, multi-sets and sets of variables of some syntactic category $\mathcal{C}$ of the object-level (with variables).

In fact in this dissertation the only categories of binders we use are variables, and lists of variables in Chapter 5. But we could imagine having other categories of binders whose expressions can be the argument of $\mathsf{Support}$, for instance terms themselves if we needed to express pattern matching.

Now we give an encoding † of syntactic category of the object-level as syntactic categories of term-expressions, ranged over by $\mathbb{T}$. The idea is that a term of a syntactic category $\mathcal{C}$ will be represented by an expression of $\mathcal{C}^\dagger$:

Definition 36 (Encoding of object-level syntactic categories)

$$\boxed{\begin{array}{lcll} \mathcal{T}^\dagger & := & \mathcal{T} & \text{if } \mathcal{T} \in \mathcal{SC} \\ (\mathcal{C}_1 \hookrightarrow \mathcal{C}_2)^\dagger & := & \mathsf{V}_{\mathcal{C}_1} \times {\mathcal{C}_2}^\dagger & \end{array}}$$

Note that is encoding is bijective: expressions of $\mathbb{T}$ will denote a term of some $\mathcal{C}$ with $\mathcal{C}^\dagger = \mathbb{T}$.

Definition 37 (Meta-grammar of HOC) The grammar of the meta-language to describe HOC is given by the following sets, for each syntactic category $\mathbb{S}$:

- a denumerable set of elements called *meta-variables* such as $M, N \ldots$, and ranged over by $\mathbb{M}, \mathbb{N} \ldots$ As at the object-level, we also write $\mathbb{M} : \mathbb{S}$ to indicate the category.

  If $\mathbb{M} : \mathbb{T}$ ($\mathbb{T}$ category of term-expressions) we say that $\mathbb{M}$ is a *meta-variable for terms*, and if $\mathbb{M} : \mathsf{V}_{\mathcal{C}}$ we say it is a *meta-variable for variables*, but rather use $\mathbb{X}, \mathbb{Y}, \ldots$ instead of $\mathbb{M}$.

- a set of elements called *constructions*, denoted like $\mathbf{d}$, that can take n arguments ($n \geq 0$). Its *signature* is a tuple of syntactic categories $(\mathbb{S}_1, \ldots, \mathbb{S}_n)$ which describes the expected categories of the arguments.

  We also write $\mathbf{c} \vdots \mathbb{S}_1 \rightharpoonup \cdots \rightharpoonup \mathbb{S}_n \rightharpoonup \mathbb{S}$.

Definition 38 (Expressions of HOC)

- In such an HOC, the *expressions*, also called *meta-terms*, of a syntactic category are given by the following five rules:

For each meta-variable $\mathbb{M} \vdots \mathbb{S}$,

$$\frac{}{\mathbb{M} \vdots \mathbb{S}}$$

For all construction $\mathbf{d} \vdots \mathbb{S}_1 \rightharpoonup \cdots \rightharpoonup \mathbb{S}_n \rightharpoonup \mathbb{S}$,

$$\frac{(\mathbb{E}_i \vdots \mathbb{S}_i)_{1 \leq i \leq n}}{\mathbf{d}(\mathbb{E}_1, \ldots, \mathbb{E}_n) \vdots \mathbb{S}}$$

For each meta-variable $\mathbb{X} \vdots \mathsf{V}_{\mathcal{C}_1 \hookrightarrow \cdots \hookrightarrow \mathcal{C}_n \hookrightarrow \mathcal{T}}$ ($\mathcal{T} \in \mathcal{SC}$),

$$\frac{(\mathbb{E}_i \vdots {\mathcal{C}_i}^{\dagger})_{1 \leq i \leq n}}{\mathbb{X}(\mathbb{E}_1, \ldots, \mathbb{E}_n) \vdots \mathcal{T}}$$

For each term constructor $\mathsf{c} \wr \mathcal{C}_1 \hookrightarrow \cdots \hookrightarrow \mathcal{C}_n \hookrightarrow \mathcal{T}$ ($\mathcal{T} \in \mathcal{SC}$),

$$\frac{(\mathbb{E}_i \vdots {\mathcal{C}_i}^{\dagger})_{1 \leq i \leq n}}{\mathsf{c}(\mathbb{E}_1, \ldots, \mathbb{E}_n) \vdots \mathcal{T}}$$

$$\frac{\mathbb{E} \vdots \mathbb{B} \quad \mathbb{E}' \vdots \mathbb{P}}{\mathbb{E}.\mathbb{E}' \vdots \mathbb{B} \times \mathbb{P}}$$

- As at the object-level, definitions and theorems about the meta-language are sometimes done by induction on expressions, i.e. on their sizes as trees.
- This inductive definition provides a notion of *sub-expression*.
- The expression $\mathbb{E}'.\mathbb{E}$ is called a *meta-binder* on $\mathbb{E}'$ with *scope* $\mathbb{E}$.

Example 5 (Constructions)

- In the meta-language we often use notions from set theory, for instance we have the constructions $\mathsf{FV}_{\mathcal{C}} \vdots \mathbb{T} \rightharpoonup \mathsf{Sets}_{\mathcal{C}}$ for each $\mathbb{T}$. We also have constructions $\emptyset \vdots \mathsf{Sets}_{\mathcal{C}}$ and $\cup, \cap \vdots \mathsf{Sets}_{\mathcal{C}} \rightharpoonup \mathsf{Sets}_{\mathcal{C}} \rightharpoonup \mathsf{Sets}_{\mathcal{C}}$ that we use as usual as an infix notation, as well as set difference denoted $\setminus$.

- We have also used $\mathsf{Support} \vdots \mathsf{V}_{\mathcal{C}} \rightharpoonup \mathsf{Sets}_{\mathcal{C}}$ and $\mathsf{Support} \vdots \mathsf{Lists}_{\mathcal{C}} \rightharpoonup \mathsf{Sets}_{\mathcal{C}}$ and the swapping $(\_\ \_)\_ \vdots \mathsf{V}_{\mathcal{C}} \rightharpoonup \mathsf{V}_{\mathcal{C}} \rightharpoonup \mathbb{T} \rightharpoonup \mathbb{T}$.

  Generalised swapping can apply to expressions of other syntactic categories: we general say that an expression $\mathbb{E}$ *can be subject to the swapping operator* if it makes sense to write $(\mathbb{X}\ \mathbb{Y})\mathbb{E}$.

- Note that the notation $\_.\_$ can itself be considered a construction of $\mathbb{B} \rightharpoonup \mathbb{T} \rightharpoonup (\mathbb{B} \times \mathbb{T})$ with a particular binding behaviour. On the contrary, the notation $[\_].\_$ for syntactic terms, that has no intrinsic notion of binding, can be seen as a construction of $\mathsf{V}_{\mathcal{C}} \rightharpoonup \mathbb{ST}_{\mathcal{C}'} \rightharpoonup \mathbb{ST}_{\mathcal{C} \hookrightarrow \mathcal{C}'}$ if $\mathbb{ST}_{\mathcal{C}}$ is the syntactic category in $\mathcal{SC}^{\mathcal{M}}$ of the expressions representing syntactic terms of $\mathcal{C}$. No side-condition avoiding variable capture and liberation will then be produced, since the represented objects are syntactic terms and not equivalence classes of them.

- For each syntactic category $\mathcal{C}$ of the object level and each syntactic category $\mathbb{P}$, we have an construction called *substitution* in $(\mathsf{V}_{\mathcal{C}} \times \mathbb{P}) \rightharpoonup \mathcal{C}^{\dagger} \rightharpoonup \mathbb{P}$.

  The construction of substitution, when applied to two arguments $\mathbb{X}.\mathbb{E}$ and $\mathbb{E}'$, is denoted $\left\{{}^{\mathbb{E}'}\!/_{\mathbb{X}}\right\}\mathbb{E}$.

- We have mentioned that we sometimes use lists, such as in $\Pi.M$, and we sometimes write $x_1.\ldots.x_n$ for $\Pi = (x_i)_{1 \leq i \leq n}$.

We now describe how we define HOC in BNF-format, noting that a connection between BNF-definitions and ERS has been studied in [Kha90].

Definition 39 (BNF-definitions) We shall often give the grammar of an HOC in *BNF-format*, by giving for each syntactic category $\mathcal{T}$ a structure like the following one:

$$\begin{array}{rcl} \mathbb{M}_{\mathcal{T}}, \mathbb{N}_{\mathcal{T}}, \ldots & ::= & \mathbb{X}_{\mathcal{T}}(\overrightarrow{\mathbb{X}_{1\mathcal{C}_1}}.\mathbb{M}_{1\mathcal{T}_1}, \ldots, \overrightarrow{\mathbb{X}_{m\mathcal{C}_m}}.\mathbb{M}_{m\mathcal{T}_m}) \mid \ldots \\ & \mid & \mathsf{c}_{\mathcal{T}}(\overrightarrow{\mathbb{X}'_{1\mathcal{C}'_1}}.\mathbb{M}'_{1\mathcal{T}'_1}, \ldots, \overrightarrow{\mathbb{X}'_{n\mathcal{C}'_n}}.\mathbb{M}'_{n\mathcal{T}'_n}) \mid \ldots \end{array}$$

where

- $\mathbb{M}_{\mathcal{T}}, \mathbb{N}_{\mathcal{T}}, \ldots$ is a scheme describing the meta-variables of $\mathcal{T}$,

- $\mathbb{X}_{\mathcal{T}}(\overrightarrow{\mathbb{X}_{1\mathcal{C}_1}}.\mathbb{M}_{1\mathcal{T}_1},\ldots,\overrightarrow{\mathbb{X}_{m\mathcal{C}_m}}.\mathbb{M}_{m\mathcal{T}_m}) \mid \ldots$ is a scheme describing the constructs with the meta-variables $\mathbb{X}_{\mathcal{T}} \vdots \mathsf{V}_{\mathcal{C}''_1 \hookrightarrow \cdots \hookrightarrow \mathcal{C}''_m \hookrightarrow \mathcal{T}}$ (with, for all $1 \leq i \leq m$, $\mathcal{C}''_i = \mathcal{C}_{i,1} \hookrightarrow \cdots \hookrightarrow \mathcal{C}_{i,p_i} \hookrightarrow \mathcal{T}_i$, and $\overrightarrow{\mathbb{X}_{i\mathcal{C}_i}}$. representing a series of bindings on $(\mathbb{X}_{i,j} \vdots \mathsf{V}_{\mathcal{C}_{i,j}})_{1\leq j\leq p_i}$, and $\mathbb{M}_{i\mathcal{T}_i}$ being a meta-variable of $\mathcal{T}_i$),
- $\mathsf{c}_{\mathcal{T}}(\overrightarrow{\mathbb{X}'_{1\mathcal{C}'_1}}.\mathbb{M}'_{1\mathcal{T}'_1},\ldots,\overrightarrow{\mathbb{X}'_{n\mathcal{C}'_n}}.\mathbb{M}'_{n\mathcal{T}'_n}) \mid \ldots$ is a scheme describing the constructs with the term constructors $\mathsf{c}_{\mathcal{T}} \wr \mathcal{C}''_1 \hookrightarrow \cdots \hookrightarrow \mathcal{C}''_n \hookrightarrow \mathcal{T}$ (with, for all $1 \leq i \leq n$, $\mathcal{C}''_i = \mathcal{C}'_{i,1} \hookrightarrow \cdots \hookrightarrow \mathcal{C}'_{i,p_i} \hookrightarrow \mathcal{T}'_i$, and $\overrightarrow{\mathbb{X}'_{i\mathcal{C}'_i}}$. representing a series of bindings on $(\mathbb{X}'_{i,j} \vdots \mathsf{V}_{\mathcal{C}'_{i,j}})_{1\leq j\leq p_i}$, and $\mathbb{M}'_{i\mathcal{T}'_i}$ being a meta-variable of $\mathcal{T}'_i$).

Either of the last two schemes can be absent when the sets of such variables or term constructors are empty (e.g. variable-free syntactic categories).

All isomorphic notations are acceptable in the definition of HOC, as long as the binders and their scopes are specified. Traditional notations will thus be allowed. For instance we can write a term construct as $\langle N/x\rangle M$ instead of $\mathsf{expsub}(x.M, N)$ for explicit substitutions such as those of $\lambda\mathsf{x}$ [BR95]. More generally, when a term constructor corresponds, in some sense, to a construction, we tend to use angled brackets for the term constructor and braces for the construction (as in the case of explicit and implicit substitutions).

Example 6 (BNF-definition of λ-calculus) We re-express in BNF-format the definition of the syntax of λ-calculus from Example 3:

$$M, N ::= x \mid \lambda x.M \mid M\ N$$

But we can also define the notation $\lambda\Pi.M$ if Π is a list of variables, where $\lambda x_1,\ldots,x_n.M$ abbreviates $\lambda x_1.\ldots.\lambda x_n.M$.

Note that for a variable category $\mathcal{T}$ of order 0, the meta-variables for terms in $\mathcal{T}$ are abusively taken to be the same as meta-variables for variables in $\mathsf{V}_{\mathcal{T}}$. In BNF-definitions we thus often omit lines such as

$$\mathbb{M} ::= \mathbb{X}$$

and we use $\mathbb{X}$ everywhere instead of $\mathbb{M}$, as we illustrate with the following example.

Example 7 (BNF-definition of $\lambda\mathsf{x}$) We can express in BNF-format the definition of the syntax of the calculus with explicit substitutions $\lambda\mathsf{x}$ by [BR95].[16] Instead of

$$\begin{array}{ll} U, V & ::= x \\ M, N & ::= \mathsf{var}(U) \mid \lambda x.M \mid M\ N \mid \langle N/x\rangle M \end{array}$$

[16] As discussed in Chapter 4, this presentation of $\lambda\mathsf{x}$ is one among others.

we can more simply write

$$M, N ::= \mathsf{var}(x) \mid \lambda x.M \mid M\ N \mid \langle N/x \rangle M$$

We also abusively write

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

if it is clear from context that variables form a syntactic category of their own.

As mentioned in the introduction of this chapter, we now develop the ideas of Barendregt's convention by giving a mechanical way to recover, just from the expressions we write to denote terms, the side-conditions that are needed to avoid variable capture and liberation. We shall then be able to safely drop those side-conditions throughout the rest of this dissertation.

Nominal logic [Pit03] might be a way of implementing the reasonings that use such conventions in first-order logic.

The idea is very close to the notion of *parameter path* of SERS [BKR00] (a particular notion of ERS): given a finite set of expressions $(\mathbb{E}_i)_{1 \leq i \leq n}$ with occurrences of a particular meta-variable for terms $\mathbb{M}$, [BKR00] forbids $\mathbb{M}$ to be instantiated with a term that contains a variable that is bound by the parameter path of one occurrence and not by that of another occurrence.

Here we bypass the notion of instance but instead produce the side-conditions, directly expressed in the meta-language, that avoid variable capture and liberation. For this we define a set-theoretic expression of the meta-language that represents the set of variables that are allowed to occur freely in the term represented by $\mathbb{M}$ but that are bound outside by abstractions having $\mathbb{M}$ in their scopes.

We first need to define what the meta-variables of an expression are. For that we use set theoretic notions (at the meta-meta-level), which we need to distinguish from the set-theoretic constructions of the meta-level. Hence we write $⫽, \sqcup, \sqcap, \mathsf{E}, [\ldots]$ for the former and $\emptyset, \cup, \cap, \in, \{\ldots\}$ for the latter. For instance, if $\mathcal{F}$ is the set $[\mathbb{E}_1, \ldots, \mathbb{E}_n]$ of expressions *denoting* sets, then $\bigcup \mathcal{F}$ stands for the expression $\mathbb{E}_1 \cup \ldots \cup \mathbb{E}_n$ (for any particular order) while $\bigsqcup \mathcal{F}$ does not make sense (the $(\mathbb{E}_i)_{1 \leq i \leq n}$ are not sets but expressions).

Definition 40 (Meta-variables of an expression) We define the *meta-*

variables for terms & variables $\mathsf{MV}(\mathbb{E})$ of an expression $\mathbb{E}$ by induction on $\mathbb{E}$:

$$\begin{array}{lll}
\mathsf{MV}(\mathbb{M}) & := [\mathbb{M}] & \text{if } \mathbb{M} \vdots \mathbb{T} \\
\mathsf{MV}(\mathbb{M}) & := \not[] & \text{if not} \\
\mathsf{MV}(\mathbb{E}'.\mathbb{E}) & := \mathsf{MV}(\mathbb{E}) & \\
\mathsf{MV}(\mathbb{X}(\mathbb{E}_1,\ldots,\mathbb{E}_n)) & := [\mathbb{X}] \sqcup \bigsqcup_{1\leq i\leq n} \mathsf{MV}(\mathbb{E}_i) & \\
\mathsf{MV}(\mathsf{c}(\mathbb{E}_1,\ldots,\mathbb{E}_n)) & := \bigsqcup_{1\leq i\leq n} \mathsf{MV}(\mathbb{E}_i) & \\
\mathsf{MV}(\mathbf{d}(\mathbb{E}_1,\ldots,\mathbb{E}_n)) & := \bigsqcup_{1\leq i\leq n} \mathsf{MV}(\mathbb{E}_i) & \\
\multicolumn{3}{l}{\text{where } \mathbb{X}, \mathsf{c} \text{ and } \mathbf{d} \text{ are respectively a meta-variable}} \\
\multicolumn{3}{l}{\text{of some } \mathsf{V}_{\mathcal{C}}, \text{ a term constructor and a construction}}
\end{array}$$

Definition 41 (Allowed variables)

- The *expression of allowed variables in a meta-variable* $\mathbb{M} \in \mathsf{MV}(\mathbb{E})$, denoted $\mathsf{AVe}_{\mathbb{M}}(\mathbb{E})$, is an expression defined by induction on $\mathbb{E}$:

$$\begin{array}{lll}
\mathsf{AVe}_{\mathbb{M}}(\mathbb{M}) & := \emptyset & \\
\mathsf{AVe}_{\mathbb{M}}(\mathbb{E}'.\mathbb{E}) & := \mathsf{AVe}_{\mathbb{M}}(\mathbb{E}) \cup \mathsf{Support}(\mathbb{E}') & \\
\mathsf{AVe}_{\mathbb{M}}(\mathbb{M}(\mathbb{E}_1,\ldots,\mathbb{E}_n)) & := \emptyset & \\
\mathsf{AVe}_{\mathbb{M}}(\mathbb{X}(\mathbb{E}_1,\ldots,\mathbb{E}_n)) & := \bigcap[\mathsf{AVe}_{\mathbb{M}}(\mathbb{E}_i) \mid \mathbb{M} \in \mathsf{MV}(\mathbb{E}_i)] & \text{if } \mathbb{X} \neq \mathbb{M} \\
\mathsf{AVe}_{\mathbb{M}}(\mathsf{c}(\mathbb{E}_1,\ldots,\mathbb{E}_n)) & := \bigcap[\mathsf{AVe}_{\mathbb{M}}(\mathbb{E}_i) \mid \mathbb{M} \in \mathsf{MV}(\mathbb{E}_i)] & \\
\mathsf{AVe}_{\mathbb{M}}(\mathbf{d}(\mathbb{E}_1,\ldots,\mathbb{E}_n)) & := \bigcap[\mathsf{AVe}_{\mathbb{M}}(\mathbb{E}_i) \mid \mathbb{M} \in \mathsf{MV}(\mathbb{E}_i)] & \\
\multicolumn{3}{l}{\text{where } \mathbb{X}, \mathsf{c} \text{ and } \mathbf{d} \text{ respectively stand for a meta-variable}} \\
\multicolumn{3}{l}{\text{of some } \mathsf{V}_{\mathcal{C}}, \text{ a term constructor and a construction}}
\end{array}$$

- Suppose every construct $\mathbb{E}_1.\mathbb{E}_2$ in an expression $\mathbb{E}$ are such that $\mathbb{E}_1 = \mathbb{X} \vdots \mathsf{V}_{\mathcal{C}}$. The *set of allowed meta-variables in a meta-variable* $\mathbb{M} \in \mathsf{MV}(\mathbb{E})$, denoted $\mathsf{AVs}_{\mathbb{M}}(\mathbb{E})$, is the set of meta-variables defined by induction on $\mathbb{E}$ as follows:

$$\begin{array}{lll}
\mathsf{AVs}_{\mathbb{M}}(\mathbb{M}) & := \not[] & \\
\mathsf{AVs}_{\mathbb{M}}(\mathbb{X}.\mathbb{E}) & := \mathsf{AVs}_{\mathbb{M}}(\mathbb{E}) \sqcup [\mathbb{X}] & \\
\mathsf{AVs}_{\mathbb{M}}(\mathbb{M}(\mathbb{E}_1,\ldots,\mathbb{E}_n)) & := \not[] & \\
\mathsf{AVs}_{\mathbb{M}}(\mathbb{X}(\mathbb{E}_1,\ldots,\mathbb{E}_n)) & := \bigsqcap[\mathsf{AVs}_{\mathbb{M}}(\mathbb{E}_i) \mid \mathbb{M} \in \mathsf{MV}(\mathbb{E}_i)] & \text{if } \mathbb{X} \neq \mathbb{M} \\
\mathsf{AVs}_{\mathbb{M}}(\mathsf{c}(\mathbb{E}_1,\ldots,\mathbb{E}_n)) & := \bigsqcap[\mathsf{AVs}_{\mathbb{M}}(\mathbb{E}_i) \mid \mathbb{M} \in \mathsf{MV}(\mathbb{E}_i)] & \\
\mathsf{AVs}_{\mathbb{M}}(\mathbf{d}(\mathbb{E}_1,\ldots,\mathbb{E}_n)) & := \bigsqcap[\mathsf{AVs}_{\mathbb{M}}(\mathbb{E}_i) \mid \mathbb{M} \in \mathsf{MV}(\mathbb{E}_i)] & \\
\multicolumn{3}{l}{\text{where } \mathbb{X}, \mathsf{c} \text{ and } \mathbf{d} \text{ respectively stand for a meta-variable}} \\
\multicolumn{3}{l}{\text{of some } \mathsf{V}_{\mathcal{C}}, \text{ a term constructor and a construction}}
\end{array}$$

Definition 42 (Generation of side-conditions) We define the *side-conditions against capture and liberation* of a finite set of expressions $[\mathbb{E}_1,\ldots,\mathbb{E}_n]$. These are expressed directly in the meta-language and defined by use of a similar notion for a single expression $\mathbb{E}$:

- For $\mathbb{E} = \mathbb{M}$, there is no side-condition.
- For $\mathbb{E} = \mathbb{E}''.\mathbb{E}'$, the side-conditions are those of $\mathbb{E}'$ and those of $\mathbb{E}''$, plus
$$\mathsf{Support}(\mathbb{E}'') \cap \mathsf{AVe}_{\mathbb{M}}(\mathbb{E}') = \emptyset$$
for each meta-variable $\mathbb{M} \in \mathsf{MV}(\mathbb{E}')$.
- For $\mathbb{E} = \mathsf{c}(\mathbb{E}_1, \ldots, \mathbb{E}_n)$ or $\mathbb{E} = \mathbf{d}(\mathbb{E}_1, \ldots, \mathbb{E}_n)$ (where c and $\mathbf{d}$ respectively stand for a term constructor and an construction), the side-conditions are those of $\{\mathbb{E}_1, \ldots, \mathbb{E}_n\}$.
- For $\mathbb{E} = \mathbb{X}(\mathbb{E}_1, \ldots, \mathbb{E}_n)$ (where $\mathbb{X} \vdots \mathsf{V}_{\mathcal{C}}$ for some $\mathcal{C}$), the side-conditions are those of $[\mathbb{E}_1, \ldots, \mathbb{E}_n]$, plus
$$\bigcup [\mathsf{AVe}_{\mathbb{X}}(\mathbb{E}_i) \mid \mathbb{X} \in \mathsf{MV}(\mathbb{E}_i)] = \emptyset$$
- The side-conditions of $[\mathbb{E}_1, \ldots, \mathbb{E}_n]$ are:
for each meta-variable $\mathbb{M} \in \bigsqcup_{1 \leq i \leq n} \mathsf{MV}(\mathbb{E}_i)$,
$$\left(\left(\bigcup \mathcal{F}\right) \setminus \left(\bigcap \mathcal{F}\right)\right) \cap \mathsf{FV}(\mathbb{M}) = \emptyset$$
writing $\mathcal{F}$ for the set of expressions $[\mathsf{AVe}_{\mathbb{M}}(\mathbb{E}_i) \mid \mathbb{M} \in \mathsf{MV}(\mathbb{E}_i)]$,
as well as the side-conditions produced by each $\mathbb{E}_i$.

Remark 39 Note that the meta-variables that appear in the side-conditions produced by the process above can all be subject to the swapping operator.

Example 8 (Side-conditions against variable capture and liberation)

- $\mathsf{AVe}_M(\mathsf{c}(x.y.M, \Pi.M))$ is the expression
$$(((\emptyset \cup \mathsf{Support}(y)) \cup \mathsf{Support}(x)) \cap \mathsf{Support}(\Pi))$$
and because the meta-level uses first-order logic with set theory, this is therein equal to $\{x, y\} \cap \mathsf{Support}(\Pi)$.

  The side-conditions are equal, after a similar set-theoretic simplification at the meta-level, to $((\{y, x\} \cup \mathsf{Support}(\Pi)) \setminus (\{y, x\} \cap \mathsf{Support}(\Pi))) \cap \mathsf{FV}(M) = \emptyset$ and $x \neq y$.
  If $\mathsf{Support}(\Pi) = \{y, z\}$ then $\mathsf{AVe}_M(\mathsf{c}(x.y.M, \Pi.M)) = \{y\}$ and the first side-condition becomes $\{x, z\} \cap \mathsf{FV}(M) = \emptyset$, i.e. $x \notin \mathsf{FV}(M)$ and $z \notin \mathsf{FV}(M)$.
- Note that the expression $\lambda x.\lambda x.M$ of λ-calculus, although we are allowed to write it, produces the unsatisfiable side-condition $x \neq x$. To denote the α-equivalence class of $\lambda[x].\lambda[x].M$, we can use $\lambda y.\lambda x.M$ with the side-condition $y \notin \mathsf{FV}(M)$.

We shall use the above automated generation of the side-conditions whenever we write a term and when we write several terms at the same level, e.g. on the left-hand side and right-hand side of an equation, of a reduction relation,etc. For instance, we show how this process of producing side-conditions applies by giving the definition of substitution, directly at the meta-level:

Definition 43 (Substitution) For each syntactic category $\mathcal{C}_1 \hookrightarrow \mathcal{C}_2$ we define a construction called *substitution*, a.k.a. *implicit substitution* or *meta-substitution*, taking two terms $x.M \wr \mathcal{C}_1 \hookrightarrow \mathcal{C}_2$ and $N \wr \mathcal{C}_1$ and constructing a term $\left\{{}^{N}\!/_{x}\right\} M \wr \mathcal{C}_2$.

For that we define, for each tuple of syntactic categories $(\mathcal{C}_i)_{1\leq i\leq n}$ and each basic syntactic category $\mathcal{T}$, an auxiliary construction that takes a term $M \wr \mathcal{C}_1 \hookrightarrow \cdots \hookrightarrow \mathcal{C}_n \hookrightarrow \mathcal{T}$ and n terms $(N_i \wr \mathcal{C}_i)_{1\leq i\leq n}$ and constructs a term $\mathbf{app}(M, N_1, \ldots, N_n) \wr \mathcal{T}$.

The definition is by mutual induction on $\mathcal{C}_1 \hookrightarrow \cdots \hookrightarrow \mathcal{C}_n \hookrightarrow \mathcal{T}$ for the auxiliary construct and on $\mathcal{C}_1$ for the substitution. Then for each $\mathcal{C}_1$ the definition of $\left\{{}^{N}\!/_{x}\right\} M$ is by induction on the size of M.[17]

$\left\{{}^{N}\!/_{x}\right\} x(M_1, \ldots, M_n)$	$:=$	$\mathbf{app}(N, \left\{{}^{N}\!/_{x}\right\} M_1, \ldots, \left\{{}^{N}\!/_{x}\right\} M_n)$	
$\left\{{}^{N}\!/_{x}\right\} y(M_1, \ldots, M_n)$	$:=$	$y(\left\{{}^{N}\!/_{x}\right\} M_1, \ldots, \left\{{}^{N}\!/_{x}\right\} M_n)$	
$\left\{{}^{N}\!/_{x}\right\} \mathsf{c}(M_1, \ldots, M_n)$	$:=$	$\mathsf{c}(\left\{{}^{N}\!/_{x}\right\} M_1, \ldots, \left\{{}^{N}\!/_{x}\right\} M_n)$	
$\left\{{}^{N}\!/_{x}\right\} (y.M)$	$:=$	$y.\left\{{}^{N}\!/_{x}\right\} M$	
$\mathbf{app}(M)$	$:=$	M	
$\mathbf{app}(x.M, M_1, \ldots, M_n)$	$:=$	$\mathbf{app}(\left\{{}^{M_1}\!/_{x}\right\} M, M_2, \ldots, M_n)$	if $n \geq 1$

The process described in Definition 42 directly gives the conditions:

- $x \neq y$ in the second line,
- $x \neq y$ and $y \notin \mathsf{FV}(N)$ in the last one.

Note that the auxiliary construct is only useful when there are higher-order variables. It also allows the abbreviation of $\mathbf{app}(x_1.\ldots.x_n.M, N_1, \ldots, N_n)$ as $\left\{{}^{N_1,\ldots,N_n}\!/_{x_1,\ldots,x_n}\right\} M$, when $x_1, \ldots, x_n$ is a list of variables of some $\mathcal{C}$ and $N_1, \ldots, N_n$ is a list of terms of $\mathcal{C}$.

Lemma 40 (Substitution lemma) $\left\{{}^{P}\!/_{y}\right\} \left\{{}^{N}\!/_{x}\right\} M = \left\{ {}^{\left\{{}^{P}\!/_{y}\right\} N}\!/_{x} \right\} \left\{{}^{P}\!/_{y}\right\} M$
(Note that we implicitly have the side-conditions $x \neq y$ and $x \notin \mathsf{FV}(P)$.)

Proof: Straightforward induction following that of Definition 43, together with the statement

$$\left\{{}^{N}\!/_{x}\right\} \mathbf{app}(M, M_1, \ldots, M_n) = \mathbf{app}(\left\{{}^{N}\!/_{x}\right\} M, \left\{{}^{N}\!/_{x}\right\} M_1, \ldots, \left\{{}^{N}\!/_{x}\right\} M_n)$$

□

[17] Note that this definition is but β-normalisation of η-long normal forms.

Remark 41 We have $\{{}^{y}\!/_{x}\}M = (x\ y)M$ if $y \notin \mathsf{FV}(M)$.
Hence, $\lambda x.M = \lambda y.\{{}^{y}\!/_{x}\}M$ if $y \notin \mathsf{FV}(M)$.

1.3.4 Rules, systems & encoding as HRS

In this dissertation we define an inference structure by giving an *inference system*: the (often infinitely many) tuples are given by finitely many *inference rules* that describe them schematically:

Definition 44 (Inference rule & system)

- An *inference rule* is a non-empty tuple of expressions. It is used at the meta-level to denote an inference structure.
- An *inference system* is a finite set of inference rules.
- Every meta-variable appearing in a rule is universally quantified just outside the rule. However restrictions might be imposed within the scope of these quantifiers and are therefore called *side-conditions* of the rule.

  Some of them might actually be dropped because they are inferred from the expressions, such as those mechanically produced by the process described in Definition 42.
- A rule is thus often represented as $\dfrac{\mathbb{E}_1 \dots \mathbb{E}_n\ P_1 \dots P_m}{\mathbb{E}}$ or $\dfrac{\mathbb{E}_1 \dots \mathbb{E}_n}{\mathbb{E}}\ P_1 \wedge \ldots \wedge P_m$, where $P_1, \ldots, P_m$ are the side-conditions.
- For two inference systems R and R' we write R, R' for $\mathsf{R} \cup \mathsf{R}'$, or even $\mathsf{R}\mathsf{R}'$ if it introduces no ambiguity.
- Judgements are often denoted by expressions of the form $\vdash (\mathbb{E}_1, \ldots, \mathbb{E}_n)$ (with variations such as an infix notation), with a specific construction $\vdash$. The expression $\vdash_{\mathcal{S}} (\mathbb{E}_1, \ldots, \mathbb{E}_n)$ then means that the judgement $\vdash (\mathbb{E}_1, \ldots, \mathbb{E}_n)$ is derivable in an inference structure $\mathcal{S}$.[18]
- A rule is *derivable*, *admissible*, *height-preserving admissible* or *invertible* in an inference system $\mathcal{S}$ when the tuples it denotes are respectively derivable, admissible, height-preserving admissible or invertible in the inference structure denoted by $\mathcal{S}$.

[18] Sometimes we write $\vdash_{\mathcal{S}}$ in the inference system when we see it as an inductively defined predicate.

Examples of inference systems are the definitions of contextual closure and α-equivalence from section 1.3.2.

At the meta-level we shall often use define inference structures and use admissibility of swapping given by Lemma 38. Since we do not want to prove each time that the hypotheses of the lemma are met, we give here a generic way to prove that they are:

Lemma 42 (Equivariance in inference rules) *Consider a rule*

$$\frac{\mathbb{E}_2(\mathbb{M}_1,\ldots,\mathbb{M}_n)\ldots\mathbb{E}_p(\mathbb{M}_1,\ldots,\mathbb{M}_n)\ P_1(\mathbb{M}_1,\ldots,\mathbb{M}_n)\ldots P_m(\mathbb{M}_1,\ldots,\mathbb{M}_n)}{\mathbb{E}_1(\mathbb{M}_1,\ldots,\mathbb{M}_n)}\ {}^{19}$$

Suppose that each $\mathbb{E}_i$ and each $\mathbb{M}_j$ can be subject to the swapping operator, and that for any $\mathbb{X},\mathbb{Y}$, $(\mathbb{X}\ \mathbb{Y})\mathbb{E}_i(\mathbb{M}_1,\ldots,\mathbb{M}_n)=\mathbb{E}_i((\mathbb{X}\ \mathbb{Y})\mathbb{M}_1,\ldots,(\mathbb{X}\ \mathbb{Y})\mathbb{M}_n)$.

The proposition
*"*If the set of tuples $(\mathbb{M}_1,\ldots,\mathbb{M}_n)$ satisfying $P_1(\mathbb{M}_1,\ldots,\mathbb{M}_n)\ldots P_m(\mathbb{M}_1,\ldots,\mathbb{M}_n)$ is equivariant, then the inference structure denoted by the inference rule is equivariant.*"*
is a theorem of the meta-level.

Proof: $(\mathbb{M}_1,\ldots,\mathbb{M}_n)$ ranges over all the objects satisfying the side-conditions, including the one denoted by $((\mathbb{X}\ \mathbb{Y})\mathbb{M}_1,\ldots,(\mathbb{X}\ \mathbb{Y})\mathbb{M}_n)$. □

Remark 43

1. If $P_1\ldots P_m$ are the the side-conditions produced by the process described in Definition 42, then
   "*The set of tuples satisfying $P_1\ldots P_m$ is equivariant.*"
   is a theorem of the meta-level.

   Hence we can forget about them when applying Lemma 42.

2. We shall often combine these results with Lemma 38 to use the admissibility of swapping in inference systems.

Definition 45 (No obvious free variables) An expression *has no obvious free variables* if every sub-expression of the form $\mathbb{X}(\mathbb{E}_1,\ldots,\mathbb{E}_n)$ (with $n\geq 0$) is in the scope of a meta-binder on $\mathbb{X}$.

[19] By $\mathbb{E}_i(\mathbb{M}_1,\ldots,\mathbb{M}_n)$ we mean that every meta-variable in $\mathbb{E}_i$ is amongst $\mathbb{M}_1,\ldots,\mathbb{M}_n$ (and similarly for the side-conditions).

Definition 46 (Rewrite rule & system)

- A *rewrite rule* is particular case of inference rule: a pair of expressions of the same syntactic category, often written $\mathbb{E}_1 \longrightarrow \mathbb{E}_2$, possibly with side-conditions.
  The first component is called the *left-hand side* of the rule and the second is called its *right-hand side*.
  A *rewrite system* is a finite set of rewrite rules.

- A *term rewrite rule* is a rewrite rule made of two expressions of $\mathcal{T} \in \mathcal{SC}$.
  A *term rewrite system* is a finite set of term rewrite rules.

- A *reductive rewrite rule* is a term rewrite rule whose left-hand side contains no constructions and is not a meta-variable, and it has no side-conditions other than the implicit ones produced by Definition 42.
  A *reductive rewrite system* is a finite set of reductive rewrite rules.

- An *atomic rewrite rule* is a reductive rewrite rule $\mathbb{E} \longrightarrow \mathbb{E}'$ in which
  - the only construction used in $\mathbb{E}'$ is the substitution,
  - $\mathbb{E}$ and $\mathbb{E}'$ have no obvious free variables,
  - if $\mathbb{M} \vdots \mathbb{T}$ is in $\mathsf{MV}(\mathbb{E}')$ then it is in $\mathsf{MV}(\mathbb{E})$.

  An *atomic rewrite system* is a finite set of atomic rewrite rules.

Remark 44 In an atomic rewrite system the only binders are of the form $\mathbb{X}.\mathbb{E}$ for some $\mathbb{X} \vdots \mathsf{V}_{\mathcal{C}}$.

Example 9 (Rewrite rules & systems)

- The rewrite rules of λ-calculus (expressed as the HOC of Example 3) are an example of an atomic rewrite system:

$$\begin{array}{ll} (\lambda x.M)\ N & \longrightarrow_\beta \ \left\{{}^{N}\!/_{x}\right\} M \\ \lambda x.M\ x & \longrightarrow_\eta \ M \end{array}$$

  Note that in the η-rule, the implicit side-conditions generated by Definition 42 impose $x \notin \mathsf{FV}(M)$. Sometimes we shall still write such condition, especially when they prevent variable liberation.

- As an example of a reductive (but not atomic) rewrite system we can give:

$$\mathsf{apply}(\mathsf{apply}(M, l), l') \longrightarrow \mathsf{apply}(M, \mathbf{concat}(l, l'))$$

  where M is a meta-variable of some category $\mathcal{T}$, l and l' are meta-variables of the syntactic category $\mathcal{L}_{\mathcal{T}}$ of lists of terms of $\mathcal{T}$, apply is a term constructor of $\mathcal{T} \hookrightarrow \mathcal{L}_{\mathcal{T}} \hookrightarrow \mathcal{T}$, and **concat** is a construction of $\mathcal{L}_{\mathcal{T}} \hookrightarrow \mathcal{L}_{\mathcal{T}} \hookrightarrow \mathcal{L}_{\mathcal{T}}$ formally defined somewhere else as the concatenation of two lists.

- As an example of a (non-reductive) term rewrite system we can give β-expansion:
$$\left\{{}^{N}\!/_{x}\right\} M \longrightarrow (\lambda x.M)\ N$$
Analysing how the rule applies to some term is not straightforward because of the implicit substitution.

- As an example of the most general notion of rewrite system we can give a rule that eliminates several occurrences of an element in a multi-set:
$$\Gamma + \{\!\{A, A\}\!\} \longrightarrow \Gamma + \{\!\{A\}\!\}$$
Note that $\Gamma + \{\!\{A, A\}\!\}$ is not a term, it could be for instance $\{\!\{B, A, C, D, A\}\!\}$.

Definition 47 (Root reduction & redex)

- The relation denoted by a term rewrite system R is written $\xrightarrow{\mathsf{root}}_{\mathsf{R}}$ at the meta-level, and its contextual closure is written $\longrightarrow_{\mathsf{R}}$ (for $\mathbf{cc}\xrightarrow{\mathsf{root}}_{\mathsf{R}}$). Assuming $M \longrightarrow_{\mathsf{R}} N$ implies a root reduction inside M and a derivation for the contextual closure, on which we can do inductions. We call such an induction *induction on (the derivation of) the reduction step*, with root reduction as the base case.

- A R-*redex* of a term (or simply *redex* when the term rewrite system R is clear from context) is a sub-term that is $\xrightarrow{\mathsf{root}}_{\mathsf{R}}$-reducible.

- We simply say R-*normal form* (resp. R-*reducible form*) for $\longrightarrow_{\mathsf{R}}$-normal form (resp. $\longrightarrow_{\mathsf{R}}$-reducible form).

Definition 48 (Strongly & weakly normalising systems) A term rewrite system R is *strongly normalising/terminating* or *weakly normalising* on a set of terms $\mathcal{T}$ (or it *terminates* on $\mathcal{T}$) if $\longrightarrow_{\mathsf{R}}$ is.

If we do not specify $\mathcal{T}$, it means that we take $\mathcal{T} = \mathcal{A}$.

As we have already said in the introduction of this section, particular reduction relations could be given by HRS (rather than by rewrite systems at the meta-level as defined above). Such an alternative definition is useful to use established theorems of HRS such as confluence of orthogonal systems. We refer the reader to [Ter03] for the definition of a HRS and the reduction relation induced by it. However the HRS have to be given in each case, unless the work is factorised by analysing the way that the meta-level defines the reduction relations. Indeed, in the same way as we could generically produce a theorem of the meta-level from assumptions about the meta-level (Lemma 42), we can also generically produce a definition of the meta-level from assumptions about the meta-level. These assumptions are that the reduction relation is given by an atomic rewrite systems.

We start by defining the η-long form of a variable as follows

Definition 49 (η-long form of a meta-variable) The η-long form of a meta-variable $\mathbb{X} \vdots \mathsf{V}_{\mathcal{C}}$ is the expression $\eta_{\mathcal{C}}(\mathbb{X}) \vdots \mathcal{C}^{\dagger}$ defined by induction on $\mathcal{C}$ as follows:

$$\eta_{\mathcal{C}_1 \hookrightarrow \cdots \hookrightarrow \mathcal{C}_n \hookrightarrow \mathcal{T}}(\mathbb{X}) := \mathbb{X}_1 \ldots \mathbb{X}_n.\mathbb{X}(\eta_{\mathcal{C}_1}(\mathbb{X}_1), \ldots, \eta_{\mathcal{C}_n}(\mathbb{X}_n))$$

for fresh meta-variables $(\mathbb{X}_i)_{1 \leq i \leq n}$.

Definition 50 (Generation of the HRS)

- Consider an atomic rewrite rule $\mathbb{E} \longrightarrow \mathbb{E}'$. We translate it as a rule of HRS (another expression of the meta-level) as follows:

  For each meta-variable $\mathbb{M} \vdots (\mathcal{C}_1 \hookrightarrow \cdots \hookrightarrow \mathcal{C}_p \hookrightarrow \mathcal{T})^{\dagger}$ of $\mathsf{MV}(\mathbb{E} \longrightarrow \mathbb{E}')$, order $\mathsf{AVs}_{\mathbb{M}}(\mathbb{E} \longrightarrow \mathbb{E}')$ as the list $\mathbb{X}_1, \ldots, \mathbb{X}_n$ with $(\mathbb{X}_i \vdots \mathsf{V}_{\mathcal{C}'_i})_{1 \leq i \leq n}$, take a fresh meta-variable $\mathbb{X}_{\mathbb{M}} \vdots \mathsf{V}_{\mathcal{C}'_1 \hookrightarrow \cdots \hookrightarrow \mathcal{C}'_n \hookrightarrow \mathcal{C}_1 \hookrightarrow \cdots \hookrightarrow \mathcal{C}_p \hookrightarrow \mathcal{T}}$ and p fresh meta-variables $(\mathbb{X}_{\mathbb{M}-i} \vdots \mathsf{V}_{\mathcal{C}_i})_{1 \leq i \leq n}$, and consider the expression:

  $$\mathbb{E}_{\mathbb{M}} := \mathbb{X}_{\mathbb{M}-1} \ldots \mathbb{X}_{\mathbb{M}-p}.\mathbb{X}_{\mathbb{M}}(\eta_{\mathcal{C}'_1}(\mathbb{X}_1), \ldots, \eta_{\mathcal{C}'_n}(\mathbb{X}_n), \eta_{\mathcal{C}_1}(\mathbb{X}_{\mathbb{M}-1}), \ldots, \eta_{\mathcal{C}_p}(\mathbb{X}_{\mathbb{M}-p}))$$

  Consider the mapping $\rho := \mathbb{M} \mapsto \mathbb{E}_{\mathbb{M}}$.

  The "*rule of HRS corresponding to* $\mathbb{E} \longrightarrow \mathbb{E}'$" is the expression

  $$\mathbb{X}_{\mathbb{M}_1}.\ldots.\mathbb{X}_{\mathbb{M}_m}.\theta_{\rho}(\mathbb{E}) \longrightarrow \mathbb{X}_{\mathbb{M}_1}.\ldots.\mathbb{X}_{\mathbb{M}_m}.\theta_{\rho}(\mathbb{E}')$$

  where $\mathbb{M}_1, \ldots, \mathbb{M}_m$ is a list obtained by ordering $\mathsf{MV}(\mathbb{E} \longrightarrow \mathbb{E}')$ and θ is inductively defined as follows:

  $$\begin{array}{|ll|}\hline \theta_{\rho}(\mathbb{M}) & := \rho\mathbb{M} \\ \theta_{\rho}(\mathbb{Y}.\mathbb{E}) & := \mathbb{Y}.\theta_{\rho}(\mathbb{E}) \\ \theta_{\rho}(f(\mathbb{E}_1, \ldots, \mathbb{E}_n)) & := f(\theta_{\rho}(\mathbb{E}_1), \ldots, \theta_{\rho}(\mathbb{E}_n)) \\ \{\theta_{\rho}(\{^{\mathbb{E}_2}/_{\mathbb{Y}}\}\mathbb{E}_1) & := \{^{\theta_{\rho}(\mathbb{E}_2)}/_{\mathbb{Y}}\}\theta_{\rho}(\mathbb{E}_1) \\ \multicolumn{2}{|r|}{\text{where } f \text{ stands for a meta-variable}} \\ \multicolumn{2}{|r|}{\text{of some } \mathsf{V}_{\mathcal{C}} \text{ or a term constructor}} \\ \hline \end{array}$$

- Given an atomic rewrite system R, "*a corresponding HRS*" is the expression $\{\mathbb{E}_1, \ldots, \mathbb{E}_n\}$ where the $\mathbb{E}_i$ are the rules of HRS corresponding to the rules of R.

Example 10 We apply the above transformation to the reductions rules of λ-calculus given in Example 9, and we respectively get for β and η:

$$\begin{array}{ll} x_M.x_N.(\lambda x.x_M\ x)\ x_N & \longrightarrow_{\beta}\ x_M.x_N.\{^{x_N}/_{x}\}(x_M\ x) \\ x_M.\lambda x.x_M\ x & \longrightarrow_{\eta}\ x_M.x_M \end{array}$$

Note in rule β that we get $\{^{x_N}/_{x}\}(x_M\ x) = x_M\ x_N$ from Definition 43.

The way we obtain the HRS rule for the η-rule is by noticing that x could not be in M, in fact $\mathsf{AVs}_M(\lambda x.M\ x \longrightarrow_{\eta}\ M) = \emptyset$.

Remark 45 Consider an atomic rewrite system R. The proposition
"*The reduction relation $\longrightarrow_R$ between terms is equal to the one generated by a corresponding HRS.*"
is a theorem of the meta-level.

Remark 46 We sometimes use the terminology "system" and "rule" directly at the meta-level, to which we now come back.

1.3.5 Induction principles with HOC

Lemma 47 *If R is a reduction system on terms of an HOC, then $SN^{\longrightarrow_R \cup \sqsupset} = SN^{\longrightarrow_R}$.*

Proof: This is a typical theorem that is usually proved classically (using for instance the postponing technique [Ter03]). We prove it constructively here. The left-to-right inclusion is trivial, by Lemma 20. Now for the other direction, first notice that $\mathsf{SN}^{\sqsupset} = \mathcal{A}$. Because of the definition of a contextual closure, $\longrightarrow_{\mathsf{R}}$ strongly simulates $\longrightarrow_{\mathsf{R}}$ through $\sqsubseteq$. Also, it weakly simulates $\sqsupset$ through $\sqsubseteq$, so we may apply Corollary 26 and get $\forall N \in \mathsf{SN}^{\rightarrow_{\mathsf{R}}}, \forall M \in \mathcal{A}, M \sqsubseteq N \Rightarrow M \in \mathsf{SN}^{\rightarrow_{\mathsf{R}} \cup \sqsupset}$. In particular, $\forall N \in \mathsf{SN}^{\rightarrow_{\mathsf{R}}}, M \in \mathsf{SN}^{\rightarrow_{\mathsf{R}} \cup \sqsupset}$. □

Notice that this result enables us to use a stronger induction principle: in order to prove $\forall M \in \mathsf{SN}^{\longrightarrow_{\mathsf{R}}}, P(M)$, it now suffices to prove

$$\forall M \in \mathsf{SN}^{\longrightarrow_{\mathsf{R}}}, (\forall N \in \mathcal{A}, (M \longrightarrow_{\mathsf{R}}^{+} N \vee N \sqsubset M) \Rightarrow P(N)) \Rightarrow P(M)$$

This induction principle is called the *transitive induction in SN^R with sub-terms* and is used in the following sections.

We briefly recall the various induction principles:
In order to prove $\forall M \in \mathsf{SN}^{\longrightarrow_{\mathsf{R}}}, P(M)$, it suffices to prove

- $\forall M \in \mathcal{A}, (\forall N \in \mathcal{A}, (M \longrightarrow_{\mathsf{R}} N) \Rightarrow P(N)) \Rightarrow P(M)$
  (raw induction in SN^{R}), or just
- $\forall M \in \mathsf{SN}^{\longrightarrow_{\mathsf{R}}}, (\forall N \in \mathcal{A}, (M \longrightarrow_{\mathsf{R}} N) \Rightarrow P(N)) \Rightarrow P(M)$
  (induction in SN^{R}), or just
- $\forall M \in \mathsf{SN}^{\longrightarrow_{\mathsf{R}}}, (\forall N \in \mathcal{A}, (M \longrightarrow_{\mathsf{R}}^{+} N) \Rightarrow P(N)) \Rightarrow P(M)$
  (transitive induction in SN^{R}), or even
- $\forall M \in \mathsf{SN}^{\longrightarrow_{\mathsf{R}}}, (\forall N \in \mathcal{A}, (M \longrightarrow_{\mathsf{R}}^{+} N \vee N \sqsubset M) \Rightarrow P(N)) \Rightarrow P(M)$
  (transitive induction in SN^{R} with sub-terms)

Definition 51 SN^{R} henceforth denotes $\mathsf{SN}^{\longrightarrow_{\mathsf{R}} \cup \sqsupset} = \mathsf{SN}^{\longrightarrow_{\mathsf{R}}}$.

1.4 Termination by Recursive Path Ordering

Having defined the notion of HOC, we now give another technique to prove termination: the *Recursive Path Orderings* (RPO) [Der82]. In this section we define the concepts for first-order terms only, i.e. those terms built with constructors whose syntactic categories are of order 1 (in other word, there is no variable binding). The technique of RPO will still be relevant for HOC in general, which can all be turned into first-order calculi by erasing all bindings and replacing every bound variable by a common constructor of arity 0, say $\star$ (pronounced *blob*).

Such an encoding loses the information about the higher-order features of the calculus but will work for our purposes. The RPO technique could equivalently be defined for HOC in general, by embedding the erasure of bound variables into the definitions,[20] but the literature usually makes the encoding explicit, as we shall do as well.

Definition 52 (Path Orderings) Consider a terminating and transitive reduction relation $\succ$ on term constructors of the HOC. In the context of path orderings this will be called the *precedence relation*.

- Suppose that each term constructor is labelled with a status lex or mul. The *Recursive Path Ordering* (*RPO*) [Der82], noted $\gg$, is the relation on terms defined inductively[21] by the following rules:

$$\frac{}{\mathsf{c}(M_1,\ldots,M_n)\gg M_i} \qquad \frac{M_i \gg M}{\mathsf{c}(M_1,\ldots,M_n)\gg M}$$

$$\frac{(\mathsf{c}(M_1,\ldots,M_n)\gg N_i)_{1\leq i\leq n}}{\mathsf{c}(M_1,\ldots,M_n)\gg \mathsf{d}(N_1,\ldots,N_m)}\ \mathsf{c}\succ\mathsf{d}$$

  if c has status lex

$$\frac{(M_1,\ldots,M_n)\gg_{\mathsf{lex}} (N_1,\ldots,N_n) \qquad (\mathsf{c}(M_1,\ldots,M_n)\gg N_i)_{1\leq i\leq n}}{\mathsf{c}(M_1,\ldots,M_n)\gg \mathsf{c}(N_1,\ldots,N_n)}$$

  if c has status mul

$$\frac{\{\!\{M_1,\ldots,M_n\}\!\}\gg_{\mathsf{mul}} \{\!\{N_1,\ldots,N_m\}\!\}}{\mathsf{c}(M_1,\ldots,M_n)\gg \mathsf{c}(N_1,\ldots,N_m)}$$

[20] This is clearly quite different from (and much weaker than) Jouannaud and Rubio's higher-order RPO [JR99], which takes the higher-order features into account, by including in the technique the termination of the simply-typed λ-calculus.

[21] Note that because of the reference to the multi-set reduction and the lexicographic reduction, the above rules do not form a proper inductive definition. However, we can label $\gg$ with integers and define $\gg_k$ by induction on k using the rules. Then it suffices to take $\gg = \bigcup_k \gg_k$. See [Ter03] for a discussion on this.

where d and c are term constructors with arities m and n, respectively, and M, the M_i, and the N_j are terms.

- The *Lexicographic Path Ordering* (*LPO*) [KL80] is the RPO obtained by giving the label lex to all term constructors.
- The *Multi-set Path Ordering* (*MPO*) is the RPO obtained by giving the label mul to all term constructors.

Remark 48

1. If $s \sqsupset t$ then $s \gg t$, which we call the *sub-term property of* $\gg$.
2. The relation $\gg$ is transitive and context-closed.

Theorem 49 (Termination of RPO) *If $\succ$ terminates on the set of term constructors, then $\gg$ terminates on the set of terms.*

Proof: See e.g. [Ter03] for a classical proof. □

Conclusion

In this chapter we have established the notations, the terminology and the basic concepts that are used in the rest of this dissertation. We have presented a constructive theory of normalisation and induction based on an approach that relies on second-order quantification rather than classical logic. We have re-established a few normalisation results in this framework, including the simulation technique and a few variants.

We have presented higher-order calculi (HOC), i.e. calculi involving variable binding. Variable capture an liberation is avoided by the use of side-conditions that we often not write explicitly, but instead we described how they can be recovered mechanically from the expressions that we use to denote terms, building on the principles behind Barendregt's convention. For that we needed to formalise the meta-level. This step back could be avoided by encoding parts of the meta-level into the object-level, such as introducing meta-variables in the syntax of higher-order calculi. This is the approach of CRS, ERS and IS. However, the extent of meta-level encoded in the object-level might not feature any notions of variable binding other than the object-level bindings (and possibly the bindings of implicit substitutions). Here we wanted to define a notion of expression that can feature any object-level and meta-level bindings.

Part I

Proof-terms for Intuitionistic Implicational Logic

Chapter 2

Natural deduction & sequent calculus

In this chapter we introduce the concepts common to all chapters of Part I, which investigates intuitionistic implicational logic, i.e. intuitionistic logic with implication as the only logical connective. We start by formalising, in a more generic framework, a generalisation of the aforementioned concepts (also used in Part III which tackles classical logic), such as the notions of logical systems and typing systems, proof-terms, etc.

The paradigm of the Curry-Howard correspondence, which relates logical systems and typing systems, is then illustrated not only by (intuitionistic implicational) natural deduction and the simply-typed λ-calculus [How80], but also by a typed HOC corresponding to the (intuitionistic implicational) sequent calculus G3ii [Kle52]. We conclude the chapter by recalling traditional encodings from one to the other, originating from works by Gentzen [Gen35] and Prawitz [Pra65] but here presented by type-preserving translations of proof-terms (as in e.g. [Zuc74, DP99b]).

The main purpose of this chapter is to make the dissertation self-contained, but most concepts formalised therein correspond, in each particular framework treated in this dissertation, to the standard ones, so the reader may safely skip them (note however some notions that are new, such as being *logically principal* —Definition 57, and *term-irrelevant admissibility* —Definition 65).

2.1 Logical systems & implicational intuitionistic logic

We first introduce general notions related to logical systems. The syntax of logical systems is based on HOC as described in Chapter 1.

Definition 53 (Logical sequent) Given two index sets $\mathcal{J} \subseteq \mathcal{I}$ and basic syntactic categories $(\mathcal{T}_i)_{i\in\mathcal{I}}$, a *logical sequent* is an object of the form

$$(\mathcal{M}_k)_{k\in\mathcal{J}} \vdash^p S$$

where

- $p \in \mathcal{I}$, allowing the distinction of different kinds of logical sequents,
- for all $k \in \mathcal{J}$, $\mathcal{M}_k$ is a multi-set of terms of $\mathcal{T}_k$, and
- $S \wr \mathcal{T}_p$.

Definition 54 (Logical rule, system & derivation)

- A *logical system* (resp. *logical rule*) for an HOC is an inference system (resp. inference rule) whose judgements are logical sequents.
- A *logical derivation* is a derivation in an inference structure given by a logical system.

We now consider an HOC with one basic syntactic category, namely that of *implicational formulae*:

Definition 55 (Implicational formulae and logical sequents)

- Let $\mathcal{Y}$ be a denumerable set, the element of which are called *atomic formulae*, and denoted $p, q, \ldots$

  The set of *implicational formulae*[1] is defined by the grammar:

  $$A, B ::= p \mid A{\rightarrow}B$$

  The constructor $\rightarrow$ is called the *implication*.

- An (implicational intuitionistic)[2] *logical sequent* is a logical sequent as defined in Definition 53 with index sets $\mathcal{J} = \mathcal{I}$ being a singleton, so it is simply of the form $\Gamma \vdash A$, where Γ is a multi-set of formulae. Γ is called the *antecedent* and the singleton multi-set $\{\!\{A\}\!\}$ is called the *succedent* of the logical sequent.
- Derivations of logical sequents in a particular inference system are called *proof-trees* or sometimes just *proofs*.

The intuitive meaning of such a logical sequent is "A can be inferred from the hypotheses Γ".

Notice 1 *For logical sequents we now use the notation Γ, Δ for the union of multi-sets $\Gamma + \Delta$. We sometimes also write A for $\{\!\{A\}\!\}$.*

[1] In the chapters of this part we sometimes say *formula* for implicational formula.

[2] Again in the chapters of this part we say *logical sequent* for implicational intuitionistic logical sequent.

Natural deduction is a logical system introduced by Gentzen [Gen35]. Its implicational fragment in intuitionistic logic, called NJi, is given in Fig. 2.1.

$$\frac{}{\Gamma, A \vdash A}\ \mathsf{ax}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A{\to}B}\ {\to}_{\mathsf{right}} \qquad \frac{\Gamma \vdash A{\to}B \quad \Gamma \vdash A}{\Gamma \vdash B}\ {\to}_{\mathsf{elim}}$$

Figure 2.1: Logical NJi

Of the sequent calculus LJ for intuitionistic logic, also introduced by Gentzen [Gen35], we present two versions (here for implication only): systems G1ii and G3ii (with i for intuitionistic and i for implicational), which are respectively presented in Fig. 2.2 and Fig. 2.3.

$$\frac{}{A \vdash A}\ \mathsf{ax}_m \qquad \frac{\Gamma \vdash A \quad \Delta, A \vdash B}{\Gamma, \Delta \vdash B}\ \mathsf{cut}_m$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A{\to}B}\ {\to}_{\mathsf{right}} \qquad \frac{\Gamma \vdash A \quad \Delta, B \vdash C}{\Gamma, \Delta, A{\to}B \vdash C}\ {\to}_{\mathsf{left}m}$$

$$\frac{\Gamma \vdash B}{\Gamma, A \vdash B}\ \mathsf{weak} \qquad \frac{\Gamma, A, A \vdash B}{\Gamma, A \vdash B}\ \mathsf{cont}$$

Figure 2.2: Logical G1ii

$$\frac{}{\Gamma, A \vdash A}\ \mathsf{ax} \qquad \frac{\Gamma \vdash A \quad \Gamma, A \vdash B}{\Gamma \vdash B}\ \mathsf{cut}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A{\to}B}\ {\to}_{\mathsf{right}} \qquad \frac{\Gamma, A{\to}B \vdash A \quad \Gamma, A{\to}B, B \vdash C}{\Gamma, A{\to}B \vdash C}\ {\to}_{\mathsf{left}}$$

Figure 2.3: Logical G3ii

Definition 56 (Derivability in NJi, G1ii, G3ii) We write $\Gamma \vdash_{\mathsf{NJ}} A$ if $\Gamma \vdash A$ is derivable in NJi, $\Gamma \vdash_{\mathsf{G1ii}} A$ if it is derivable in G1ii ($\Gamma \vdash_{\mathsf{G1iicf}} A$ if it is derivable without the cut_m-rule), and $\Gamma \vdash_{\mathsf{G3ii}} A$ if it is derivable in G3ii ($\Gamma \vdash_{\mathsf{G3iicf}} A$ if it is derivable without the cut-rule).

Rules ax and ax_m are called *axiom rules*, cut and cut_m are called *cut-rules*, weak and cont are *structural rules*, respectively called the *weakening rule* and *contraction rule*. Rules $\rightarrow_{\mathsf{left}}$ and $\rightarrow_{\mathsf{left}_m}$ are the *left-introduction rules for implication*, $\rightarrow_{\mathsf{right}}$ is the *right-introduction rule for implication*, and $\rightarrow_{\mathsf{elim}}$ is the *elimination rule for implication*. Axioms are considered both left- and right-introduction rules.[3] On the contrary, cut, cut_m are neither left- nor right-introduction rules.

On the one hand, sequent calculus has left- and right-introduction rules, cuts and possibly structural rules. On the other hand, natural deduction never modifies the antecedent; only the axiom is a left-introduction rule, otherwise the rules are either right-introduction rules or elimination rules.

Definition 57 (Principal formula)

- In some rules of the above three systems, there is a formula that we distinguish and call *principal formula* of the inference step: $A \rightarrow B$ in $\rightarrow_{\mathsf{left}}$, $\rightarrow_{\mathsf{left}_m}$ and $\rightarrow_{\mathsf{right}}$, and A in ax, ax_m, weak and cont,
- A formula A is *not used* in a derivation in $\mathsf{G3ii}$ if the tree obtained from this derivation by changing the label $\Gamma \vdash B$ of each node into $\Gamma \setminus \{\!\{A\}\!\} \vdash B$ is still a derivation in $\mathsf{G3ii}$.
- In $\mathsf{G3ii}$, a formula A is *logically principal* if it is either principal in the succedent or both principal in the antecedent and not used in strict subderivations.

The definitions of (logically) principal formula, left- and right-introduction rules can clearly be adapted to other rules dealing e.g. with connectives other than implication. However, while it is very easy and natural to adapt the definition on a case by case basis, it seems much more difficult to give an abstract definition whose genericity would apply to all cases, e.g. in the general framework of logical systems.

Definition 58 (Context) In all the above rules, the formulae of the antecedent that are not principal are called the *context*.[4]

Note that $\mathsf{G1ii}$ is made of *context-splitting rules*, in that the context of the conclusion is split into the contexts of the premisses (and this holds for the axiom, which has no premiss: only the empty context can be split between 0 premisses). System $\mathsf{G3ii}$ is made of *context-sharing rules*, in that the context of

[3] This could be inherited from variants of the axioms where the formula A is necessarily atomic, in which case they do introduce atomic formulae, just like $\rightarrow_{\mathsf{left}}$, $\rightarrow_{\mathsf{left}_m}$ and $\rightarrow_{\mathsf{right}}$ introduce the implication. But we shall see later that considering axioms as introduction rules has more profound reasons connected to the notions of *value* and *covalue*, as we shall see from Section 2.3 onwards.

[4] This has nothing to do with the notion of contextual closure introduced in Chapter 1.

the conclusion is shared between all the premisses, being duplicated in rules with at least two premisses, and being erased in rules with no premisses such as the axiom. For rules with exactly one premiss, the notions of context-splitting and context-sharing is the same.

Sometimes, context-splitting rules are said to be *multiplicative* (hence the subscript m in the name of the rules), while context-sharing rules are said to be *additive*. Note that NJi is a context-sharing/additive system, like G3ii.

System G1ii and G3ii already illustrate the diversity of systems that exist in the literature for sequent calculus (including G4ii, which we investigate in Chapter 7), with quite a bewildering nomenclature.

Our G1ii-system here matches the implicational fragment of both the G1i-system of [TS00] and the G0i-system of [NvP01]. These slightly differ from earlier work by Kleene [Kle52] whose intuitionistic G1-system sticks to Gentzen's original presentation of LJ where the antecedent is a list of formulae instead of a multi-set, with an inference rule called *exchange*:

$$\frac{\Gamma, A, B, \Delta \vdash C}{\Gamma, B, A, \Delta \vdash C}$$

The terminology G1, G2, G3 originates from the sequent calculi presented in [Kle52], which differ from each other in the way they treat *structural rules*, i.e. weakening, contraction and exchange. G1-systems make them explicit as rules, whilst G3-systems incorporate them into the other rules.

Our system G3ii is exactly the implicational fragment Kleene's intuitionistic G3, from which both [TS00] and [NvP01] differ (ax is restricted to A being an atomic formula, and the antecedent formula $A{\to}B$ is systematically dropped in the second premiss of $\to_{\mathsf{left}}$, building on Kleene's variant G3a where arbitrary omissions of formulae are allowed in the premisses of the rules).

Remark 50 In G3-systems such as G3ii, weakenings are hidden in the ax-rule and contractions are hidden in the context-sharing rules and in the fact that principal formulae of the antecedent are already in the premisses.

The following lemma holds:

Lemma 51 *Weakening and contraction are height-preserving admissible in G3ii and in NJi.*

Proof: Straightforward induction on derivations. □

This is used to establish the following equivalence:

Theorem 52 (Logical equivalence of G1ii, G3ii & NJi)
$\Gamma \vdash_{G1ii} A$ *if and only if* $\Gamma \vdash_{G3ii} A$ *if and only if* $\Gamma \vdash_{NJ} A$.

Proof:

- We obtain that $\Gamma \vdash_{\mathsf{G1ii}} A$ implies $\Gamma \vdash_{\mathsf{G3ii}} A$ by a straightforward induction on derivations, using Lemma 51. The converse is also obtained by an induction, which formally reveals the ideas of Remark 50.
- The second equivalence is proved in the rest of this chapter by using the proof-terms approach.

□

System G1ii illustrates how the notions of weakening and contraction (together with that of cut) traditionally come from sequent calculus. Our approach of using them in natural deduction in Chapter 5 is thus an example of how fruitful the connection between sequent calculus and natural deduction can be made.

2.2 Typing systems

In this section we introduce basic notions of typing for an HOC with no higher-order variables (if $x \wr \mathcal{C}$ then $\mathsf{order}(\mathcal{C}) = 0$). Issues connected to α-equivalence such as variable capture and liberation are treated as described in Chapter 1.

We consider a function that maps some basic syntactic categories to other basic syntactic categories. If this function maps $\mathcal{T}$ to $\mathsf{T}_{\mathcal{T}}$ (possibly the same), $\mathsf{T}_{\mathcal{T}}$ is called the *typing category of* $\mathcal{T}$, and $\mathcal{T}$ is called a *typable category*. Nothing prevents a typing category from being a typable one, including the case $\mathcal{T} = \mathsf{T}_{\mathcal{T}}$.

If $\mathcal{T}$ is typed by $\mathsf{T}_{\mathcal{T}}$ which is typed by $\mathsf{T}_{\mathsf{T}_{\mathcal{T}}}$, terms of $\mathsf{T}_{\mathsf{T}_{\mathcal{T}}}$ are often called *kinds* and terms of $\mathsf{T}_{\mathcal{T}}$ are often called *types*, while *terms* is then reserved to terms of $\mathcal{T}$. $\mathsf{T}_{\mathcal{T}}$ is called the *category of types of* $\mathcal{T}$ and $\mathsf{T}_{\mathsf{T}_{\mathcal{T}}}$ the *category of kinds of* $\mathcal{T}$.

Definition 59 (Environments) Suppose $\mathcal{T}$ is a typable category (with variables), with category of types $\mathsf{T}_{\mathcal{T}}$.

- An $\mathcal{T}$*-environment* (or just *environment* when $\mathcal{T}$ is clear) Γ is a consistent[5] finite set of *declarations*, i.e. expressions $x : S$ (where x is a variable of $\mathcal{T}$ and S is a type —i.e. a term of $\mathsf{T}_{\mathcal{T}}$) *declaring* x to be of type S. In other words, an environment is a finite function from variables to types, so we have the standard notion of *domain* of an environment Γ, written $\mathsf{Dom}(\Gamma)$. The environment Γ *declares* the variable x if its belongs to its domain, i.e. if there is a type A such that $(x : A) \in \Gamma$.
- Unless otherwise stated, $\Gamma, \Delta, \ldots$ will henceforth denote environments.
- We write Γ, Δ to denote the disjoint and consistent union of two $\mathcal{T}$-environments Γ and Δ.

[5]By *consistent* is meant that if $x : S_1$ and $x : S_2$ are in Γ, then $S_1 = S_2$.

- We say that Δ is a *sub-environment* of Γ if $\Delta \subseteq \Gamma$ (in the set-theoretic sense).
- We denote by $m(\Gamma)$ the *range* of an environment Γ, i.e. the multi-set *associated with* Γ, obtained from Γ by removing the variables but keeping the types. Formally, it is the multi-set f that maps every $A \wr \mathsf{T}_{\mathcal{T}}$ to the natural number $|\{x \mid (x{:}A) \in \Gamma\}|$.

Definition 60 (Sequent) Suppose $\mathcal{T}$ is a typable category with category of types $\mathsf{T}_{\mathcal{T}}$. A *$\mathcal{T}$-sequent* (or just *sequent* when $\mathcal{T}$ is clear) is an object of the form $\{\Gamma_{\mathcal{T}_1}, \ldots, \Gamma_{\mathcal{T}_n}\} \vdash^{\mathcal{T}} M{:}S$, where

- for all i, $\Gamma_{\mathcal{T}_i}$ is an $\mathcal{T}_i$-environment, for every typable category T_i of the HOC with variables (and $i \neq j$ implies $\mathcal{T}_i \neq \mathcal{T}_j$),
- $M \wr \mathcal{T}$, and
- $S \wr \mathsf{T}_{\mathcal{T}}$.

Definition 61 (Typing system & typing derivation)

- A *typing rule* for an HOC is an inference rule denoting an inference structure whose judgements are sequents and that is equivariant, i.e. if $\dfrac{\mathcal{J}_1 \quad \ldots \quad \mathcal{J}_m}{\mathcal{J}}$ is in the inference structure then so is $\dfrac{(x\ y)\mathcal{J}_1 \quad \ldots \quad (x\ y)\mathcal{J}_m}{(x\ y)\mathcal{J}}$ for all variables x, y of some syntactic category.
- A *typing system* for an HOC is an inference system whose inference rules are typing rules.
- A *typing derivation* is a derivation in an inference structure given by a typing system.

We now introduce a basic property that we shall require of reduction relations on typed HOC:

Definition 62 (Subject reduction property) A reduction relation $\rightarrow$ on a typed HOC *satisfies the subject reduction property* if the following holds:
If $M \rightarrow N$ and $\{\Gamma_{\mathcal{T}_1}, \ldots, \Gamma_{\mathcal{T}_n}\} \vdash^{\mathcal{T}} M{:}S$ then $\{\Gamma_{\mathcal{T}_1}, \ldots, \Gamma_{\mathcal{T}_n}\} \vdash^{\mathcal{T}} N{:}S$.

We now define a translation from typing systems to logical systems, by removing all variable and term annotations:

Definition 63 (Erasure of term annotations) We consider logical sequents built by taking the typable categories as the index set $\mathcal{I}$ and the typable categories with variables as the index set $\mathcal{J}$.

- We extend the notation m to sequents:

$$m(\{\Gamma_{\mathcal{T}_1}, \ldots, \Gamma_{\mathcal{T}_n}\} \vdash^{\mathcal{T}} M:S) = (m(\Gamma_{\mathcal{T}_i}))_{1\leq i\leq n} \vdash^{\mathcal{T}} S$$

- We extend the notation m to typing rules:

$$m\left(\frac{\mathcal{J}_1 \quad \ldots \quad \mathcal{J}_n}{\mathcal{J}}\right) = \frac{m(\mathcal{J}_1) \quad \ldots \quad m(\mathcal{J}_n)}{m(\mathcal{J})}$$

- The notation m extends naturally to typing systems, typing derivations, etc.

Sequents can thus be turned into logical sequents by simply erasing the variables and the term, and conversely logical sequents can be *decorated* by variables and terms. Hence, we shall define inference rules with sequents, so that a logical sequent is derivable if and only if it can be decorated into a derivable sequent. In fact in general we only have the following:

Remark 53 If there exists a term M such that $\{\Gamma_{\mathcal{T}_1}, \ldots, \Gamma_{\mathcal{T}_n}\} \vdash^{\mathcal{T}} M:S$ is derivable in R then the logical sequent $(m(\Gamma_{\mathcal{T}_i}))_{1\leq i\leq n} \vdash^{\mathcal{T}} S$ is derivable in $m(\mathsf{R})$.

The reverse in not true in general, however we can express a condition for the reverse to hold. This is based on the notion of *unconditionality*. In simple words, a rule is *unconditional* if whenever it applies on premisses that type the terms $(M_i)_{1\leq i\leq p}$, it should also apply on premisses that type any terms $(N_i)_{1\leq i\leq p}$ with the same types in the same environments. In other words, the rule does not analyse the shape of the terms being typed in the premisses.

Definition 64 (Unconditionality)

- An inference structure $\mathcal{E}$ whose judgements are sequents is *unconditional* w.r.t. another inference structure $\mathcal{E}'$ if the following property holds:
  For all $\dfrac{(\{\Gamma^i_{\mathcal{T}_1}, \ldots, \Gamma^i_{\mathcal{T}_n}\} \vdash^{\mathcal{T}^i} M^i:S^i)_{1\leq i\leq p}}{\{\Gamma_{\mathcal{T}_1}, \ldots, \Gamma_{\mathcal{T}_n}\} \vdash^{\mathcal{T}} M:S} \in \mathcal{E}$ and all terms $(N_i)_{1\leq i\leq p}$,
  if $(\{\Gamma^i_{\mathcal{T}_1}, \ldots, \Gamma^i_{\mathcal{T}_n}\} \vdash^{\mathcal{T}^i} N^i:S^i)_{1\leq i\leq p}$ are sequents derivable in $\mathcal{E}'$
  then there is a term N such that $\dfrac{(\{\Gamma^i_{\mathcal{T}_1}, \ldots, \Gamma^i_{\mathcal{T}_n}\} \vdash^{\mathcal{T}^i} N^i:S^i)_{1\leq i\leq p}}{\{\Gamma_{\mathcal{T}_1}, \ldots, \Gamma_{\mathcal{T}_n}\} \vdash^{\mathcal{T}} N:S} \in \mathcal{E}$.

- A rule is *unconditional* w.r.t. a typing system if the inference structure that the former denotes is unconditional w.r.t. the inference structure that the latter denotes.

- A typing system is *unconditional* if its rules are unconditional w.r.t. itself.

Now we can prove:

Remark 54 There exists a term M such that $\{\Gamma_{\mathcal{T}_1}, \ldots, \Gamma_{\mathcal{T}_n}\} \vdash^{\mathcal{T}} M : A$ is derivable in an unconditional typing system R if and only if the logical sequent $(m(\Gamma_{\mathcal{T}_i}))_{1 \leq i \leq n} \vdash^{\mathcal{T}} A$ is derivable in $m(\mathsf{R})$.

Typing systems are often unconditional when the typed terms reflect precisely the structure of their typing derivations, and thus that of the logical derivations obtained by erasing the variable and term annotations.

This is the basis of the *Curry-Howard* paradigm, according to which the terms/types/reduction of a typed HOC respectively correspond to the proofs/propositions/proof transformations of a logical system. Such a correspondence gives a double reading of proofs as programs and programs as proofs, so that insight into one aspect helps the understanding of the other. A good account can be found in e.g. [SU06].

In unconditional systems, such as those used in the Curry-Howard paradigm, we can define a notion that corresponds to the notion of admissibility in the logical system obtained by erasing the variable and term annotations:

Definition 65 (Term-irrelevant admissibility)
Suppose R is an unconditional typing system. A rule $\dfrac{\mathcal{J}_1 \quad \ldots \quad \mathcal{J}_p}{\{\Gamma_{\mathcal{T}_1}, \ldots, \Gamma_{\mathcal{T}_n}\} \vdash^{\mathcal{T}} M : S}$ [6] that is unconditional w.r.t. R is *term-irrelevantly admissible* in R if the following property holds:
If the premisses $\mathcal{J}_1, \ldots, \mathcal{J}_p$ are derivable in R,
then there exists a derivation in R of $\{\Gamma_{\mathcal{T}_1}, \ldots, \Gamma_{\mathcal{T}_n}\} \vdash^{\mathcal{T}} M' : S$ for some term M'.

Remark 55 The notion of term-irrelevant admissibility corresponds to the standard notion of admissibility (Definition 11) when term annotations are erased:

$\dfrac{\mathcal{J}_1 \quad \ldots \quad \mathcal{J}_n}{\mathcal{J}}$ is term-irrelevantly admissible in an unconditional typing system R,

if and only if

$m\left(\dfrac{\mathcal{J}_1 \quad \ldots \quad \mathcal{J}_n}{\mathcal{J}}\right)$ is admissible in $m(\mathsf{R})$.

We now give a canonical way to generate typing systems for the Curry-Howard paradigm, with one term construct for each derivation step. These typing systems are called *canonical typing systems*, and we give here the definition for only a particular kind of HOC:

[6] Note the use of the dotted line to indicate this notion of admissibility as well.

Definition 66 (Canonical typing system) We consider an HOC in which only one typable category, called $\mathcal{T}_0$, has variables (a set $\mathcal{X}$ of variables denoted $x, y, z, \ldots$). By "variables" we now mean these ones rather than those of non-typable categories (in particular, variables of typing categories are called *type variables*). Most typed HOC presented in the chapters of Part I are built in that framework. Types are denoted $A, B, C, D, \ldots$ regardless of their syntactic categories.

Assume the BNF-definition of such an HOC is a collection of lines such as the following one for $\mathcal{T}_0$

$$M_{\mathcal{T}_0}, N_{\mathcal{T}_0}, P_{\mathcal{T}_0}, \ldots ::= x \mid \mathsf{c}_{\mathcal{T}_0}(\overrightarrow{x_1}.M_{1\mathcal{T}_1}, \ldots, \overrightarrow{x_n}.M_{n\mathcal{T}_n}) \mid \ldots$$

and such as the following one for each other typable category $\mathcal{T}$

$$M_{\mathcal{T}}, N_{\mathcal{T}}, P_{\mathcal{T}}, \ldots ::= \mathsf{c}_{\mathcal{T}}(\overrightarrow{x_1}.M_{1\mathcal{T}_1}, \ldots, \overrightarrow{x_n}.M_{n\mathcal{T}_n}) \mid \ldots$$

A *canonical typing system* is a typing system of the form:

$$\frac{(x:A) \in \Gamma}{\Gamma \vdash^{\mathcal{T}_0} x:A} \qquad \text{for each term constructor } \mathsf{c}_{\mathcal{T}},\ \frac{(\Gamma_i, \overrightarrow{x_i:B_i} \vdash^{\mathcal{T}_i} M_i:A_i)_{1\leq i\leq n}}{\Gamma \vdash^{\mathcal{T}} \mathsf{c}_{\mathcal{T}}(\overrightarrow{x_1}.M_1, \ldots, \overrightarrow{x_n}.M_n):A}$$
for all $M_i \wr \mathcal{T}_i$,
with $\Gamma_i \subseteq \Gamma$ for all i and $\bigcup_{1\leq i\leq n} \Gamma_i = \Gamma$

Note that:

- the condition that the rules are equivariant must still be checked for each specification of the environments Γ, Γ_i, etc.
- the unconditionality of the system is ensured by forcing M_i to range over all terms of $\mathcal{T}_i$,
- the tree-structure of a term M reflects the tree-structure of a derivation of $\Gamma \vdash^{\mathcal{T}} M:A$, so the terms are often called *proof-terms*,
- if $\Gamma \vdash^{\mathcal{T}} M:A$ then $\mathsf{FV}(M) \subseteq \mathsf{Dom}(\Gamma)$.

Note that a canonical typing system constrains terms to satisfy a structural property that we call *being well-formed*:

Definition 67 (Being well-formed) A term M is *well-formed* if in every subterm of the form $\mathsf{c}(N_1, \ldots, N_n)$, $(\mathsf{Dom}(\Gamma) \setminus \mathsf{Dom}(\Gamma_i)) \cap \mathsf{FV}(N_i) = \emptyset$ for all i.

This notion is especially interesting when, for every term constructor c, $\mathsf{Dom}(\Gamma) \setminus \mathsf{Dom}(\Gamma_i)$ can be expressed independently from the typing system, in which case the constraint of being well-formed can be considered before and independently from the notion of typing. Such an example of constraint is given in Chapter 7.

We can identify two kinds of canonical typing systems.

Definition 68 (Additive & multiplicative typing system)
An *additive typing system* is one of the form:

$$\frac{(x:A) \in \Gamma}{\Gamma \vdash^{\mathcal{T}_0} x:A}$$

for each term constructor $\mathsf{c}_{\mathcal{T}}$,

$$\frac{(\Gamma, \overrightarrow{x_i:B_i} \vdash^{\mathcal{T}_i} M_i:A_i)_{1\leq i\leq n}}{\Gamma \vdash^{\mathcal{T}} \mathsf{c}_{\mathcal{T}}(\overrightarrow{x_1}.M_1, \ldots, \overrightarrow{x_n}.M_n):A}$$

for all $M_i \wr \mathcal{T}_i$ and
all environment Γ (also for the axiom)

Note that

- equivariance is ensured by forcing Γ to range over all environments,
- the left-hand side rule, often called *axiom*, allows an arbitrary environment that declares x, and
- the rules with several premisses are *environment-sharing*, in that their premisses all use the antecedent of the conclusion instead of splitting it.

A *multiplicative typing system* is one of the form:

$$\frac{}{x:A \vdash^{\mathcal{T}_0} x:A}$$

for each term constructor $\mathsf{c}_{\mathcal{T}}$:

$$\frac{(\Gamma_i, \overrightarrow{x_i:B_i} \vdash^{\mathcal{T}_i} M_i:A_i)_{1\leq i\leq n}}{\Gamma_1, \ldots, \Gamma_n \vdash^{\mathcal{T}} \mathsf{c}_{\mathcal{T}}(\overrightarrow{x_1}.M_1, \ldots, \overrightarrow{x_n}.M_n):A}$$

for all $M_i \wr \mathcal{T}_i$ and
all environments Γ_1,...,Γ_n

Note that

- equivariance is ensured by forcing Γ_1,...,Γ_n to range over all environments,
- only x is declared by the environment, and

- the rules with several premisses are *environment-splitting*.

Most chapters of this dissertation investigate additive typing systems, but an example of multiplicative system is the object of Chapter 5. Some other typing systems are mixtures of multiplicative and additive systems (e.g. in Chapter 7).

In the additive case, the constraint of being well-formed becomes empty: no restriction on terms is suggested by the typing system. In the multiplicative case, the constraint can be strengthened as the notion of *linearity* on terms, which, indeed, can be considered before and independently from the notion of typing:

Definition 69 (Linearity) A term M is *linear* if

- in every sub-term $x.N$ we have $x \in \mathsf{FV}(N)$, and
- in every sub-term $\mathsf{c}(N_1, \ldots, N_n)$ the sets $(\mathsf{FV}(N_i))_{1 \leq i \leq n}$ are pairwise disjoint.

Remark 56 Indeed, if $\{\Gamma_{\mathcal{T}_1}, \ldots, \Gamma_{\mathcal{T}_n}\} \vdash^{\mathcal{T}} M : A$ is derivable in a multiplicative system, then M is linear.

From now on, throughout the chapters of Part I and unless otherwise stated, types are the *implicational formulae*[7] of Definition 55. *Atomic types* are atomic formulae (a.k.a. type variables), still denoted $p, q, \ldots$ The intuitive meaning of the decorated sequent $\Gamma \vdash M : A$ is that each use of a free variable in M corresponds to the "logical use of an hypothesis of Γ to derive A".

In the next section we start with the traditional Curry-Howard correspondence between NJi and the simply-typed λ-calculus [Chu41] as described in [How80], and then we introduce an HOC for a similar correspondence for G3ii.

2.3 Natural deduction & λ-calculus

Definition 70 (Syntax of λ-calculus) The syntax of λ-calculus [Chu41] is defined as follows:

$$M, N ::= x \mid \lambda x.M \mid \ M\ N$$

where x ranges over a denumerable set of variables.

Terms of the form $\lambda x.M$ are called a *(λ)-abstractions*, and those of the form $M\ N$ *applications*.

Definition 71 (β-reduction & η-reduction) The notion of reduction is β-reduction and η-reduction defined by the following rules:

$$\begin{array}{lllll} \beta & (\lambda x.M)\ N & \longrightarrow & \left\{{}^{N}\!/_{x}\right\} M & \\ \eta & \lambda x.M\ x & \longrightarrow & M & \text{if } x \notin \mathsf{FV}(M) \end{array}$$

[7] Hence, by *principal type* we mean *principal formula* rather than the *principal type of a term* in implicitly or explicitly polymorphic systems and intersection type systems.

We say that a λ-term is *normal* if it is a β-normal form.

We now present a typing system for λ-calculus, called the simply-typed λ-calculus[8] [GTL89].

Definition 72 (Simply-typed λ-calculus) We write $\Gamma \vdash_\lambda M : A$ if the sequent $\Gamma \vdash M : A$ is derivable with the inference rules of Fig. 2.4.

$$\frac{}{\Gamma, x:A \vdash x:A}\ \mathsf{ax}$$

$$\frac{\Gamma, x:A \vdash M:B}{\Gamma \vdash \lambda x.M : A \to B}\ \to_{\mathsf{right}} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}\ \to_{\mathsf{elim}}$$

Figure 2.4: Typing rules for λ-calculus

Note how the typing rules match those of Fig. 2.1, in that the erasure of variable and term annotations (of Definition 63) of the typing system is exactly the logical system.

Remark 57 By Remark 54, $\Gamma \vdash_{\mathsf{NJ}} A$ if and only if Γ' and M can be found such that $\Gamma' \vdash_\lambda M : A$ with $\Gamma = m(\Gamma')$.

This is the basis of the Curry-Howard correspondence [How80].

Definition 73 (Value) Those terms of the form x or $\lambda x.M$ are called *values* and are ranged over by $V, V', \ldots$ These are the term constructs typed by (right-)introduction rules.

Remark 58 The fact that weakening and contraction are admissible in NJi (Lemma 51) can be seen with terms:

$$\frac{\Gamma \vdash M:B}{\Gamma, x:A \vdash M':B} \quad \text{and} \quad \frac{\Gamma, x:A, y:A \vdash M:B}{\Gamma, x:A \vdash M':B}$$

are term-irrelevantly admissible rules in the simply-typed system of λ-calculus. The fact that they are even height-preserving admissible can be seen if in each case we give the term M' that works:

$$\frac{\Gamma \vdash M:B}{\Gamma, x:A \vdash M:B} \quad \text{and} \quad \frac{\Gamma, x:A, y:A \vdash M:B}{\Gamma, x:A \vdash \{x/y\}M:B}$$

are height-preserving rules in the simply-typed system of λ-calculus.

[8] The adjective "simply" opposes this typing system to more complicated ones, as we shall see e.g. in Chapter 4.

Remark 59 The notion of being *free* in a proof-term expresses whether an antecedent type of the sequent is actually used in the proof. Indeed, the following rule is also height-preserving admissible:

$$\frac{\Gamma, x:A \vdash M:B \quad x \notin \mathsf{FV}(M)}{\Gamma \vdash M:B}$$

Finally, we give three main theorems of the λ-calculus that we use later:

Theorem 60 (Confluence of λ-calculus) *$\longrightarrow_\beta$ and $\longrightarrow_{\beta,\eta}$ are confluent.*

Proof: See e.g. [Bar84]. □

Theorem 61 (Typing of substitution) *The following rule is admissible*[9] *in the simply-typed system of λ-calculus:*

$$\frac{\Gamma \vdash_\lambda N:A \qquad \Gamma, x:A \vdash_\lambda M:B}{\Gamma \vdash_\lambda \left\{{}^{N}\!/_{x}\right\} M:B}$$

Proof: See e.g. [GTL89]. □

Theorem 62 (Strong Normalisation of the simply-typed λ-calculus) *If $\Gamma \vdash_\lambda M:A$ then $M \in \mathsf{SN}^{\beta\eta}$.*

Proof: See e.g. [GTL89]. □

2.4 An HOC for G3ii

Formalisms where structural rules are treated implicitly, such as LJ or G3ii, can be turned into typing systems of HOC without implying any condition on subterms and free variables such as linearity or similar notions. Just as the purely logical system NJi is a typing system of λ-calculus, G3ii can also be turned into the typing system of an HOC, which we call λG3. This syntax can be found in various textbooks (e.g. [TS00]) and papers (e.g. [DP99b]) with notational variants. It is defined as follows:

Definition 74 (λG3)

$$M, N, P ::= x \mid \lambda x.M \mid x[M, y.N] \mid \langle M \dagger x.N \rangle$$

where x ranges over a denumerable set of variables (which form a syntactic category of their own).

The last constructor of the syntax is called the *cut-constructor*, and we call $\lambda\mathsf{G3}^{\mathsf{cf}}$ the sub-syntax made without it.

[9] Note that in general it is *not* height-preserving admissible.

Definition 75 (Simply-typed λG3-calculus) We write $\Gamma \vdash_{\lambda\mathsf{G3}} M : A$ if the sequent is derivable with the inference rules of Fig. 2.5, and $\Gamma \vdash_{\lambda\mathsf{G3^{cf}}} M : A$ if it is derivable without the cut-rule (i.e. $M \in \lambda\mathsf{G3^{cf}}$).

$$\frac{}{\Gamma, x:A \vdash x:A}\,\mathsf{ax} \qquad \frac{\Gamma \vdash M:A \quad \Gamma, x:A \vdash N:B}{\Gamma \vdash \langle M \dagger x.N\rangle : B}\,\mathsf{cut}$$

$$\frac{\Gamma, x:A \vdash M:B}{\Gamma \vdash \lambda x.M : A{\to}B}\,{\to}_{\mathsf{right}} \qquad \frac{\Gamma, x:A{\to}B \vdash M:A \quad \Gamma, x:A{\to}B, y:B \vdash N:C}{\Gamma, x:A{\to}B \vdash x[M, y.N]:C}\,{\to}_{\mathsf{left}}$$

Figure 2.5: G3ii

Note how the typing rules match those of Fig. 2.3, in that the erasure of variable and term annotations (of Definition 63) of the typing system is exactly the logical system.

Remark 63 By Remark 54, $\Gamma \vdash_{\mathsf{G3ii}} A$ (resp. $\Gamma \vdash_{\mathsf{G3iicf}} A$) if and only if Γ' and M can be found such that $\Gamma' \vdash_{\lambda\mathsf{G3}} M : A$ (resp. $\Gamma' \vdash_{\lambda\mathsf{G3^{cf}}} M : A$) with $\Gamma = m(\Gamma')$.

Definition 76 (Value & covalue)

- Those terms of the form x or $\lambda x.M$ are called *values* and are ranged over by $V, V', \ldots$
- We say that M is an x-*covalue* if $M = x$ or $M = x[N, y.P]$ for some N, y, P with $x \notin FV(N) \cup FV(P)$.

At this point we can already notice that values are constructs typed by a derivable sequent whose succedent is (logically) principal. Similarly, typed y-covalues correspond to those proofs of a sequent in the last step of which the type of y is logically principal.

It is clear that the λ-abstraction is the same as in the λ-calculus, and it will be made clear that the cut-constructor is very similar to a $\mathsf{let}\ldots\mathsf{in}$ construct or an explicit substitution.

Remark 64 As in Remark 58, the fact that weakening and contraction are height-preserving admissible in LJ (Lemma 51) can be seen with terms:

$$\frac{\Gamma \vdash M:B}{\Gamma, x:A \vdash M:B} \quad \text{and} \quad \frac{\Gamma, x:A, y:A \vdash M:B}{\Gamma, x:A \vdash \{x/y\}M:B}$$

are height-preserving admissible rules.

Remark 65 Again, the notion of being *free* in a proof-term expresses whether an antecedent type of the sequent is actually used in the proof. Indeed, the following rule is also admissible:

$$\frac{\Gamma, x:A \vdash M:B \quad x \notin \mathsf{FV}(M)}{\Gamma \vdash M:B}$$

Note that various HOC, similar to these proof-terms for $\mathsf{G3ii}$, can be introduced for classical sequent calculus, see e.g. [CH00, Urb00, Len03].

2.5 Encodings to & from λ-calculus

In order to understand the semantics of such a term syntax for $\mathsf{G3ii}$, we can re-express Gentzen's translation of proofs in sequent calculus to natural deductions using $\lambda\mathsf{G3}$-terms and λ-terms.

$$\boxed{\begin{array}{lcl} \mathcal{G}^1(x) & := & x \\ \mathcal{G}^1(\lambda x.M) & := & \lambda x.\mathcal{G}^1(M) \\ \mathcal{G}^1(x[M, y.N]) & := & \left\{{}^{x\ \mathcal{G}^1(M)}\!/_{y}\right\}\mathcal{G}^1(N) \\ \mathcal{G}^1(\langle M \dagger x.N\rangle) & := & \left\{{}^{\mathcal{G}^1(M)}\!/_{x}\right\}\mathcal{G}^1(N) \end{array}}$$

Notice that terms of $\lambda\mathsf{G3}^{\mathsf{cf}}$ are always mapped to λ-terms in normal form.

We can also a give backward translation from natural deduction to sequent calculus:

$$\boxed{\begin{array}{lcl} \mathcal{G}^2(x) & := & x \\ \mathcal{G}^2(\lambda x.M) & := & \lambda x.\mathcal{G}^2(M) \\ \mathcal{G}^2(M\ N) & := & \langle \mathcal{G}^2(M) \dagger x.x[\mathcal{G}^2(N), y.y]\rangle \end{array}}$$

Interestingly enough, normal forms of λ-calculus are not necessarily mapped to $\lambda\mathsf{G3}^{\mathsf{cf}}$-terms. This is fixed by Prawitz's encoding [Pra65]:

$$\boxed{\begin{array}{lcl} \mathcal{P}r(x) & := & x \\ \mathcal{P}r(\lambda x.M) & := & \lambda x.\mathcal{P}r(M) \\ \mathcal{P}r(M\ N) & := & \mathcal{P}r_{x.x}(M\ N) \\ \hline \mathcal{P}r_{x.Q}(y\ M) & := & y[\mathcal{P}r(M), x.Q] \\ \mathcal{P}r_{x.Q}((\lambda y.N)\ M) & := & \langle \lambda y.\mathcal{P}r(N) \dagger z.z[\mathcal{P}r(M), x.Q]\rangle \\ \mathcal{P}r_{x.Q}(N_1\ N_2\ M) & := & \mathcal{P}r_{z.z[\mathcal{P}r(M),x.Q]}(N_1\ N_2) \end{array}}$$

requiring $n \geq 1$ in the last two lines.

Theorem 66 (Preservation of typing)

- *If* $\Gamma \vdash_{\lambda G3} M : A$ *then* $\Gamma \vdash_{\lambda} \mathcal{G}^1(M) : A$.
- *If* $\Gamma \vdash_{\lambda} M : A$ *then* $\Gamma \vdash_{\lambda G3} \mathcal{G}^2(M) : A$.
- *If* $\Gamma \vdash_{\lambda} M\ N : A$ *and* $\Gamma, x : A \vdash_{\lambda G3} \mathcal{P}r(M) : B$ *then* $\Gamma \vdash_{\lambda G3} \mathcal{P}r_{x.Q}(M\ N) : B$.
- *If* $\Gamma \vdash_{\lambda} M : A$ *then* $\Gamma \vdash_{\lambda G3} \mathcal{P}r(M) : A$.

Proof: Straightforward induction on derivations, using Theorem 61. □

The following theorems can be found in the literature, some of them in [DP99b].

Theorem 67 (Properties of the encodings)

- $\mathcal{G}^1$ *is surjective,* $\mathcal{G}^1{}_{|\lambda G3^{cf}}$ *is surjective on normal* λ*-terms, and neither is injective.*
- $\mathcal{G}^2$ *and* $\mathcal{P}r$ *are both injective but not surjective on* $\lambda G3$.
- $\mathcal{G}^1 \circ \mathcal{G}^2 = Id_{\lambda}$ *and* $\mathcal{G}^1 \circ \mathcal{P}r = Id_{\lambda}$
- *Neither* $\mathcal{G}^2 \circ \mathcal{G}^1 \neq Id_{\lambda G3}$ *nor* $\mathcal{P}r \circ \mathcal{G}^1 \neq Id_{\lambda G3}$.
- $\mathcal{G}^2 \circ \mathcal{G}^1$ *does not leave* $\lambda G3^{cf}$ *stable but* $\mathcal{P}r \circ \mathcal{G}^1$ *does.*

Proof: Straightforward induction on terms. □

Conclusion

The first part of this chapter gives some directions for further work. The first is a generalisation to every logical systems of the notions of additive and multiplicative rules, introduction and elimination rules, principal formula, logically principal formula, context, etc. The second is a generic mechanism that would produce a canonical typing system that corresponds, when variable and term annotations are erased, to a given logical system, so that they form a Curry-Howard correspondence.

Chapter 3

Call-by-value λ-calculus

This chapter presents a few new results about λ-calculus.

The *call-by-name* (*CBN*) and *call-by-value* (*CBV*) disciplines occur both as particular reduction strategies in functional programming (leaving no choice as for which redex is to be reduced), and also more simply (in various λ-calculi) as two sub-calculi that can be interpreted with particular denotational semantics. Theorems relating the two aspects can be found in [Plo75], and here we are interested in the second, i.e. with no particular constraint about the context in which we perform root reductions.

While the CBN λ-calculus is simply the full λ-calculus, with β-reduction and possibly η-reduction (as presented in Chapter 2), the notion of CBV λ-calculus seems less canonical:

In [Plo75], a calculus called λ_{V} is introduced, whose terms are exactly those of λ-calculus and whose reduction rule β_{V} is merely β-reduction restricted to the case where the argument is a *value* V, i.e. a variable or an abstraction.

$$\beta_{\mathsf{V}} \qquad (\lambda x.M)\, V \longrightarrow \left\{{}^{V}\!/_{x}\right\} M$$

Clearly, in the fragment $\lambda_{\mathsf{EvalArg}}$ of λ-calculus where all arguments are values, $\beta_{\mathsf{V}} = \beta$. Hence, applying CBN- (general β) or CBV- (β_{V}) reduction is the same, a property that is called *strategy indifference*.

Using the notion of *continuation*, of which a history can be found in [Rey93], the (CBN) λ-calculus can be encoded into $\lambda_{\mathsf{EvalArg}}$ with a *continuation-passing-style* (*CPS*) translation in a way such that two terms are β-equivalent if and only if their encodings are β/β_{V}-equivalent [Plo75]. Similarly, two CPS-translations into $\lambda_{\mathsf{EvalArg}}$ can be defined for the CBV calculus λ_{V}: Reynolds' [Rey72, Rey98] and Fischer's [Fis72, Fis93], which only differ in the order of two arguments appearing in these translations. Both of them are sound, in that in each case two β_{V}-equivalent λ-terms are mapped to two β/β_{V}-equivalent terms, but they are incomplete in that the β/β_{V}-equivalence in $\lambda_{\mathsf{EvalArg}}$ is bigger than needed (see e.g. [Plo75]).

Strongly related to the idea of monad and monadic λ-calculus [Mog91], Moggi's λ_{C}-calculus [Mog88] extends λ_{V} with a $\mathsf{let}\ \_ = \_\ \mathsf{in}\ \_$ constructor and new reduction rules, such as:

$$\mathsf{let_V} \qquad \mathsf{let}\ x = V\ \mathsf{in}\ M \longrightarrow \left\{{}^{V}\!/_{x}\right\} M$$

In particular, with the use of this constructor, an application cannot be a normal form if one of its sides is not a value.

Both aforementioned CPS-translations can be extended to λ_{C}, in a way such that they are both sound and complete. Alternatively, the new constructor can be avoided and represented as a β-redex, and the reduction rules can then be expressed as manipulations (essentially, permutations) of β-redexes, with the same properties (e.g. [SF93] for the case of Fischer).

More refined investigations about the CBV λ-calculus consist of analysing how *reduction*, rather than equivalence, is preserved by CPS-translations. In other words, the question arises of how CPS-translations could be made not only equational correspondences, but also Galois connections or reflections (see Definition 7). The two aforementioned translations, in their original forms, construct redexes that do not correspond to those redexes present in the term that they translate. Directly reducing some of them, which are called *administrative redexes*, produces refined versions of the Reynolds and Fischer translations.

In [SW97], a refined Reynolds translation (with a reverse translation) is proved to form a reflection in λ_{C} of its target calculus $\lambda^{\mathcal{R}}_{\mathsf{CPS}}$. It is also stated that a particular refined Fischer translation cannot be a reflection, nor can it even be a Galois connection from λ_{C}. We claim here that a different choice of which redexes are considered administrative, which seems more natural, leads to a different refined version of the Fischer translation that *does form a reflection* of its target calculus $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ in (a minor variation of) λ_{C}.

We leave the discussion about administrative redexes for section 3.1. The minor variation of λ_{C} consists of replacing rule β_{V} with the following:

$$\mathsf{B} \qquad (\lambda x.M)\ N \longrightarrow \mathsf{let}\ x = N\ \mathsf{in}\ M$$

This change is justified for four reasons:

- This variation makes our refined Fischer translation a reflection in λ_{C} of its target calculus.
- It is closer to a sequent calculus called LJQ and presented in Chapter 6 as the CBV-fragment of G3ii from Chapter 2.
- It does not change the equational theory of λ_{C}.
- Splitting β_{V} into B followed by $\mathsf{let_V}$ seems more atomic and natural, with only one rule controlling whether a particular sub-term is a value or not.

From the reflection result we can also prove (Theorem 5) that our version of λ_{C} is confluent, using the confluence of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$. The latter is a simple consequence of the confluence of β-reduction in λ-calculus (Theorem 60) and the fact that $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ is stable under β/β_{V}-reduction.

All these results (reflection, confluence, etc.) also hold with (CBV) η-reduction added, e.g. in λ_{V} with the rule:

$$\eta_{\mathsf{V}} \qquad \lambda x.V\, x \longrightarrow V \qquad \text{if } x \notin \mathsf{FV}(V)$$

Unfortunately, confluence of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ with its corresponding η-reduction rules $\eta_{\mathsf{V}}1$ and $\eta_{\mathsf{V}}2$ is not as direct as with β-reduction only, since $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ is not stable under general η-reduction[1] (so we cannot directly use Theorem 60). Since we could not find a proof of this result in the literature, we establish it as follows:

- we consider the closure $\lambda^{+}_{\mathsf{CPS}}$ of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ under general β,η-reduction, which we know to be confluent from Theorem 60,
- we establish a reflection in $\lambda^{+}_{\mathsf{CPS}}$ of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ and use Theorem 5.

Section 3.1 presents Plotkin's CBV λ_{V}-calculus [Plo75], the Reynolds and Fischer *continuation-passing style* (CPS) translations from λ_{V}. Section 3.2 identifies the target calculi $\lambda^{\mathcal{R}}_{\mathsf{CPS}}$ and $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$, and proves the confluence of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$. Section 3.3 presents Moggi's λ_{C}-calculus [Mog88] extending λ_{V}, in which the extended Reynolds translation and its reverse translation were proved to form a reflection of $\lambda^{\mathcal{R}}_{\mathsf{CPS}}$ [SW97]. In section 3.4 we prove that the extended Fischer translation and its reverse translation, which we here introduce, similarly form a reflection of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ in our slightly modified version of λ_{C}.

3.1 λ_{V} & CPS-translations

Now we present the λ_{V}-calculus:

Definition 77 (λ_{V}) The syntax of λ_{V} is the same as that of λ-calculus, although it is elegant to present it by letting values form a syntactic category of their own:

$$\begin{array}{ll} V, W, \ldots & ::= x \mid \lambda x.M \\ M, N, \ldots & ::= V \mid M\ N \end{array}$$

$V, W, \ldots$ stand for *values*, i.e. variables or abstractions.

We obtain the reduction rules of λ_{V} by restricting β-reduction to the cases where the argument is a value and restricting η-reduction to the cases where the function is a value:

$$\begin{array}{lllll} \beta_{\mathsf{V}} & (\lambda x.M)\, V & \longrightarrow & \{V/x\}\, M & \\ \eta_{\mathsf{V}} & \lambda x.V\, x & \longrightarrow & V & \text{if } x \notin \mathsf{FV}(V) \end{array}$$

[1] In fact, even $\lambda_{\mathsf{EvalArg}}$ is not stable under general η-reduction.

In presence of β_{V}, the following rule has the same effect as η_{V}:

$$\eta'_{\mathsf{V}} \quad \lambda x.y\, x \longrightarrow y \qquad \text{if } x \neq y$$

Definition 78 ($\lambda_{\mathsf{EvalArg}}$) $\lambda_{\mathsf{EvalArg}}$ is the fragment of λ-calculus where all arguments are values, hence given by the following syntax:

$$\begin{array}{ll} V, W, \ldots & ::= x \mid \lambda x.M \\ M, N, \ldots & ::= V \mid M\ V \end{array}$$

Remark 68 (Strategy indifference) In the $\lambda_{\mathsf{EvalArg}}$ fragment, $\beta_{\mathsf{V}} = \beta$, so applying CBN- (general β) or CBV- (β_{V}) reduction is the same.

Note that the situation is different for η-conversion in $\lambda_{\mathsf{EvalArg}}$, since $\eta_{\mathsf{V}} \neq \eta$. The fragment $\lambda_{\mathsf{EvalArg}}$ is stable under β_{V}/β-reduction and under η_{V}-reduction, but not under η-reduction.

We can encode λ_{V} into $\lambda_{\mathsf{EvalArg}}$ by using the notion of continuation and defining *Continuation Passing Style translations* (*CPS-translations*). There are in fact two variants of the CBV CPS-translation: Reynolds' [Rey72, Rey98] and Fischer's [Fis72, Fis93], presented in Fig. 3.1.

$\mathcal{R}(V)$	$:=$	$\lambda k.k\ \mathcal{R}_{\mathsf{V}}(V)$
$\mathcal{R}(M\ N)$	$:=$	$\lambda k.\mathcal{R}(M)\ (\lambda x.\mathcal{R}(N)\ (\lambda y.x\ y\ k))$
$\mathcal{R}_{\mathsf{V}}(x)$	$:=$	x
$\mathcal{R}_{\mathsf{V}}(\lambda x.M)$	$:=$	$\lambda x.\lambda k.\mathcal{R}(M)\ k$

Reynolds' translation

$\mathcal{F}(V)$	$:=$	$\lambda k.k\ \mathcal{F}_{\mathsf{V}}(V)$
$\mathcal{F}(M\ N)$	$:=$	$\lambda k.\mathcal{F}(M)\ (\lambda x.\mathcal{F}(N)\ (\lambda y.x\ k\ y))$
$\mathcal{F}_{\mathsf{V}}(x)$	$:=$	x
$\mathcal{F}_{\mathsf{V}}(\lambda x.M)$	$:=$	$\lambda k.\lambda x.\mathcal{F}(M)\ k$

Fischer's translation

Figure 3.1: CBV CPS-translations

Note that the two translations only differ in the order of arguments in $x\ y\ k$ / $x\ k\ y$ and $\lambda x.\lambda k.\mathcal{R}(M)\ k$ / $\lambda k.\lambda x.\mathcal{F}(M)\ k$.

As mentioned in the introduction, both translations map β_{V}-equivalent terms to β/β_{V}-equivalent terms (soundness), but the converse fails (incompleteness) (see e.g. [Plo75]).

A more refined analysis is given by looking at the reduction rather than just equivalence. Note that the two translations above introduce many fresh variables, but bind them, often leading to new redexes and potential redexes not corresponding to redexes in the original terms. Some of these get in the way of

simulating β-reduction of the original terms. However, they can be identified as *administrative*, so that the translations above can be refined by reducing these redexes. The precise definition of which redexes are administrative is crucial, since this choice might or might not make the refined Fischer translation a Galois connection or a reflection (Definition 7), as we shall see in section 3.4. In Fig. 3.2 we give the refinement for a particular choice of administrative redexes.

$$
\begin{array}{lcll}
V :_{\mathcal{R}} K & := & K\ V^{\mathcal{R}} & \\
M\ N :_{\mathcal{R}} K & := & M :_{\mathcal{R}} \lambda x.(x\ N :_{\mathcal{R}} K) & \text{if } M \text{ is not a value} \\
V\ N :_{\mathcal{R}} K & := & N :_{\mathcal{R}} \lambda y.(V\ y :_{\mathcal{R}} K) & \text{if } N \text{ is not a value} \\
V\ V' :_{\mathcal{R}} K & := & V^{\mathcal{R}}\ V'^{\mathcal{R}}\ K & \\
\hline
x^{\mathcal{R}} & := & x & \\
(\lambda x.M)^{\mathcal{R}} & := & \lambda x.\lambda k.(M :_{\mathcal{R}} k) &
\end{array}
$$

Reynolds

$$
\begin{array}{lcll}
V :_{\mathcal{F}} K & := & K\ V^{\mathcal{F}} & \\
M\ N :_{\mathcal{F}} K & := & M :_{\mathcal{F}} \lambda x.(x\ N :_{\mathcal{F}} K) & \text{if } M \text{ is not a value} \\
V\ N :_{\mathcal{F}} K & := & N :_{\mathcal{F}} \lambda y.(V\ y :_{\mathcal{F}} K) & \text{if } N \text{ is not a value} \\
V\ V' :_{\mathcal{F}} K & := & V^{\mathcal{F}}\ K\ V'^{\mathcal{F}} & \\
\hline
x^{\mathcal{F}} & := & x & \\
(\lambda x.M)^{\mathcal{F}} & := & \lambda k.\lambda x.(M :_{\mathcal{F}} k) &
\end{array}
$$

Fischer

Figure 3.2: Refined CBV CPS-translations

We first prove that the refined translations are indeed obtained by reduction of the original ones.

Lemma 69

1. $\mathcal{R}_{\mathsf{V}}(V) \longrightarrow_\beta^* V^{\mathcal{R}}$ *and* $\mathcal{R}(M)\ K \longrightarrow_\beta^* M :_{\mathcal{R}} K$
2. $\mathcal{F}_{\mathsf{V}}(V) \longrightarrow_\beta^* V^{\mathcal{F}}$ *and* $\mathcal{F}(M)\ K \longrightarrow_\beta^* M :_{\mathcal{F}} K$

Proof: For each point the two statements are proved by mutual induction on V and M. The interesting case is for $M = M_1\ M_2$, which we present with the Fischer translation (the case of Reynolds is very similar): $\mathcal{F}(M_1\ M_2)\ K = \mathcal{F}(M_1)\ (\lambda x.\mathcal{F}(M_2)\ (\lambda y.x\ K\ y)) \longrightarrow_\beta\ M_1 :_{\mathcal{F}} (\lambda x.N :_{\mathcal{F}} (\lambda y.x\ K\ y))$ by induction hypothesis.

- If neither M_1 nor M_2 are values, this is also $M_1\ M_2 :_{\mathcal{F}} K$.
- If M_1 is a value and M_2 is not, this is also $(\lambda x.M_2 :_{\mathcal{F}} (\lambda y.x\ K\ y))\ {M_1}^{\mathcal{F}} \longrightarrow_\beta\ M_2 :_{\mathcal{F}} (\lambda y.{M_1}^{\mathcal{F}}\ K\ y) = M_1\ M_2 :_{\mathcal{F}} K$.

- If M_1 is not a value but M_2 is, this is also
  $M_1 :_{\mathcal{F}} (\lambda x.(\lambda y.x\ K\ y)\ {M_2}^{\mathcal{F}}) \longrightarrow_\beta M_1 :_{\mathcal{F}} (\lambda x.x\ K\ {M_2}^{\mathcal{F}}) = M_1\ M_2 :_{\mathcal{F}} K$.

- If both M_1 and M_2 are values, this is also
  $(\lambda x.(\lambda y.x\ K\ y)\ {M_2}^{\mathcal{F}})\ {M_1}^{\mathcal{F}} \longrightarrow_\beta^* {M_1}^{\mathcal{F}}\ K\ {M_2}^{\mathcal{F}} = M_1\ M_2 :_{\mathcal{F}} K$.

□

Remark 70 Note that K is a sub-term of $M :_{\mathcal{R}} K$ and $M :_{\mathcal{F}} K$ with exactly one occurrence[2], so for instance if $x \in \mathsf{FV}(K) \setminus \mathsf{FV}(M)$ then x has exactly one free occurrence in $M :_{\mathcal{R}} K$ and $M :_{\mathcal{F}} K$. Hence, the variables introduced by the translations and denoted by k, which we call the *continuation variables*, are such that the set of those that are free in the scope of a binder $\lambda k.$ is exactly $\{k\}$, with exactly one occurrence (only one of them can be free at a time).

In other words, in a term (which is a α-equivalence class of syntactic terms, i.e. a set) there is always a representative that always uses the same variable k. Note that K does not need to range over all λ-terms for the definition of the refined translations to make sense, but only over constructs of the form k or $\lambda x.M$, with $x \neq k$.

In that case, note that if we call *continuation redex* any β-redex binding a continuation variable (i.e. of the form $(\lambda k.M)\ N$), then the refined Reynolds translation considers all continuation redexes administrative and has thus reduced all of them, while the refined Fischer translation leaves a continuation redex in the construct $(\lambda x.M)\ V :_{\mathcal{F}} K = (\lambda k.\lambda x.M :_{\mathcal{F}} k)\ K\ V^{\mathcal{F}}$, which is thus not administrative.

This choice is different from that of [SW97] which, as for the Reynolds translation, considers all continuation redexes administrative. With that choice they establish negative results about the refined Fischer translation as we shall discuss in section 3.4.

We can now identify the target calculi of the refined translations, i.e. the fragments of λ-calculus reached by them, and look at their associated notions of reduction.

3.2 The CPS calculi $\lambda^{\mathcal{R}}_{\mathsf{CPS}}$ & $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$

From the refined Reynolds and Fischer translations we get the target fragments of λ-calculus described in Fig. 3.3.

- $M, N, \ldots$ denote (CPS-)*programs*,
- $V, W, \ldots$ denote (CPS-)*values*,
- $K, K', \ldots$ denote *continuations*.

M, N	$::= K\ V \mid V\ W\ K$	
V, W	$::= x \mid \lambda x.\lambda k.M$	
	with $k \in \mathsf{FV}(M)$	
K	$::= k \mid \lambda x.M$	

Reynolds: $\lambda^{\mathcal{R}}_{\mathsf{CPS}}$

M, N	$::= K\ V \mid V\ K\ W$	
V, W	$::= x \mid \lambda k.\lambda x.M$	
	with $k \in \mathsf{FV}(\lambda x.M)$	
K	$::= k \mid \lambda x.M$	

Fischer: $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$

Figure 3.3: Target calculi

Note that values have no free occurrence of continuation variables while programs and continuations have exactly one. Also note that x ranges over an infinite set of variables, while for every term it is always possible to find a representative (i.e. a syntactic term) that uses a unique continuation variable k. In fact we could have a constructor with arity 0 to represent this variable and a constructor with arity 1 for $\lambda k.\_$, but treating k as a variable allows the use of the implicit substitution of k.

The fragments are stable under the reductions in Fig. 3.4 and are sufficient to simulate β_{V} and η_{V} through the CPS-translations, as we shall see in section 3.4. We write $\lambda^{\mathcal{R}}_{\mathsf{CPS}\beta}$ (resp. $\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta}$) for system $\beta_{\mathsf{V}}1, \beta_{\mathsf{V}}2$ and $\lambda^{\mathcal{R}}_{\mathsf{CPS}\beta\eta}$ (resp. $\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta\eta}$) for system $\beta_{\mathsf{V}}1, \beta_{\mathsf{V}}2, \eta_{\mathsf{V}}1, \eta_{\mathsf{V}}2$ in $\lambda^{\mathcal{R}}_{\mathsf{CPS}}$ (resp. $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$).

$\beta_{\mathsf{V}}1$	$(\lambda x.M)\ V$	$\longrightarrow$	$\left\{{}^{V}/_{x}\right\} M$	
$\beta_{\mathsf{V}}2$	$(\lambda x.\lambda k.M)\ V\ K$	$\longrightarrow$	$\left\{{}^{K}/_{k}\right\}\left\{{}^{V}/_{x}\right\} M$	
$\eta_{\mathsf{V}}1$	$\lambda x.\lambda k.V\ x\ k$	$\longrightarrow$	V	if $x \notin \mathsf{FV}(V)$
$\eta_{\mathsf{V}}2$	$\lambda x.K\ x$	$\longrightarrow$	K	if $x \notin \mathsf{FV}(K)$

Reynolds

$\beta_{\mathsf{V}}1$	$(\lambda x.M)\ V$	$\longrightarrow$	$\left\{{}^{V}/_{x}\right\} M$	
$\beta_{\mathsf{V}}2$	$(\lambda k.\lambda x.M)\ K\ V$	$\longrightarrow$	$(\lambda x.\left\{{}^{K}/_{k}\right\} M)\ V$	
$\eta_{\mathsf{V}}1$	$\lambda k.\lambda x.V\ k\ x$	$\longrightarrow$	V	if $x \notin \mathsf{FV}(V)$
$\eta_{\mathsf{V}}2$	$\lambda x.K\ x$	$\longrightarrow$	K	if $x \notin \mathsf{FV}(K)$

Fischer

Figure 3.4: Reduction rules for $\lambda^{\mathcal{R}}_{\mathsf{CPS}}$ & $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$

Note the difference between the case of Reynolds and that of Fischer in the rule $\beta_{\mathsf{V}}2$. Reynolds-$\beta_{\mathsf{V}}2$ must perform two β_{V}-reduction steps, since $(\lambda k.\left\{{}^{V}/_{x}\right\} M)\ K$ is not a program of the fragment. Fischer-$\beta_{\mathsf{V}}2$ performs only one β_{V}-reduction step, $(\lambda x.\left\{{}^{K}/_{k}\right\} M)\ V$ being a valid program. It could obviously reduce further to $\left\{{}^{V}/_{x}\right\}\left\{{}^{K}/_{k}\right\} M$ as for the case of Reynolds, but leaving this second step as a $\beta_{\mathsf{V}}1$-reduction has nice properties: this split of reduction into two atomic β_{V}-steps makes the refined Fischer translation (as defined here) a reflection.

[2] In some sense the construction $\_ :_{\mathcal{F}} \_$ is *linear* in its second argument.

A good account of the refined Reynolds translation as a reflection can be found in [SW97], so here we study similar properties for the refined Fischer translation, building on earlier work [SF93] that established results of equational correspondence. Moreover, Fischer's approach helps establishing connections between CBV-λ-calculus and a particular fragment of G3ii called LJQ and studied in Chapter 6.

We now establish the confluence of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$. The confluence of $\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta}$ is straightforward, since every case of β-reduction in $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ is covered by either $\beta_{\mathsf{V}}1$ or $\beta_{\mathsf{V}}2$, so it is a particular case of the confluence of β-reduction in λ-calculus (Theorem 60).

For $\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta\eta}$ we use the confluence of β,η-reduction in λ-calculus, but unfortunately the grammar of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ is not stable under β,η. Fig. 3.5 shows its closure $\lambda^{+}_{\mathsf{CPS}}$ under β,η.

$$\begin{array}{lll} M,N & ::= & K\ V \\ V,W & ::= & x \mid \lambda k.K \\ K & ::= & k \mid \lambda x.M \mid V\ K \end{array}$$

Figure 3.5: Grammar of $\lambda^{+}_{\mathsf{CPS}}$

First, note that we no longer have $\beta = \beta_{\mathsf{V}}$. Second, this grammar is indeed stable under β,η; all the cases are:

$$\begin{array}{lll} (\lambda x.M)\ V & \longrightarrow_{\beta} & \{{}^{V}/_{x}\}\,M \\ (\lambda k.K)\ K' & \longrightarrow_{\beta} & \{{}^{K'}/_{k}\}\,K \\ \lambda k.V\ k & \longrightarrow_{\eta} & V \\ \lambda x.K\ x & \longrightarrow_{\eta} & K \quad \text{if } x \notin \mathsf{FV}(K) \end{array}$$

We can then take β,η as the reductions of $\lambda^{+}_{\mathsf{CPS}}$, and derive from the confluence of λ-calculus (Theorem 60) that β and η are confluent in $\lambda^{+}_{\mathsf{CPS}}$.

Note that $\lambda^{+}_{\mathsf{CPS}}$ is the smallest grammar that includes that of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ and that is stable under β,η: Fig. 3.6 defines a mapping $\Uparrow$ from $\lambda^{+}_{\mathsf{CPS}}$ onto $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ such that $\Uparrow M \longrightarrow_{\eta} M$. Our convention for parentheses assumes that $\Uparrow$ applies to the smallest expression on its right-hand side.

Remark 71 Note that $\Uparrow M \longrightarrow_{\eta} M$, $\Uparrow V \longrightarrow_{\eta} V$, $\Uparrow K \longrightarrow_{\eta} K$ and if M, V, K are in $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ then $\Uparrow M = M$, $\Uparrow V = V$, $\Uparrow K = K$.

We can now prove the following:

Theorem 72 ($\Uparrow$ is a Galois connection) *The identity $\mathsf{Id}_{\lambda^{\mathcal{F}}_{CPS}}$ and the mapping $\Uparrow$ form a Galois connection from $\lambda^{\mathcal{F}}_{CPS}$, equipped with $\lambda^{\mathcal{F}}_{CPS\beta\eta}$, to λ^{+}_{CPS}, equipped with β_{V} (and also with only $\lambda^{\mathcal{F}}_{CPS\beta}$ and β).*

Proof: Given Remark 71, it suffices to check the simulations:

- For the simulation of η by $\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta\eta}$ through $\Uparrow$, we check all cases:

$$
\begin{array}{|ll|}
\hline
\Uparrow (k\ V) & := \ k\ \Uparrow V \\
\Uparrow ((\lambda x.M)\ V) & := \ (\lambda x.\Uparrow M)\ \Uparrow V \\
\Uparrow (W\ K\ V) & := \Uparrow W\ \Uparrow K\ \Uparrow V \\
\hline
\Uparrow x & := \ x \\
\Uparrow \lambda k.k & := \ \lambda k.\lambda x.k\ x \\
\Uparrow \lambda k.\lambda x.M & := \ \lambda k.\lambda x.\Uparrow M \\
\Uparrow \lambda k.V\ K & := \ \lambda k.\lambda x.\Uparrow V\ \Uparrow K\ x \\
\hline
\Uparrow k & := \ k \\
\Uparrow \lambda x.M & := \ \lambda x.\Uparrow M \\
\Uparrow (V\ K) & := \ \lambda x.\Uparrow V\ \Uparrow K\ x \\
\hline
\end{array}
$$

Figure 3.6: Projection of $\lambda^{+}_{\mathsf{CPS}}$ onto $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$

$$
\begin{array}{|llll|}
\hline
\Uparrow \lambda k.\lambda x.k\ x & = \lambda k.\lambda x.k\ x & = & \Uparrow \lambda k.k \\
\Uparrow \lambda k.\lambda x.(\lambda y.M)\ x & = \lambda k.\lambda x.(\lambda y.\Uparrow M)\ x & \longrightarrow_{\eta \mathsf{v} 2} & \lambda k.\lambda y.\Uparrow M & = \Uparrow \lambda k.\lambda y.M \\
\Uparrow \lambda k.\lambda x.V\ K\ x & = \lambda k.\lambda x.\Uparrow V\ \Uparrow K\ x & = & \Uparrow \lambda k.V\ K \\
\Uparrow \lambda k.V\ k & = \lambda k.\lambda x.\Uparrow V\ k\ x & \longrightarrow_{\eta \mathsf{v} 1} & \Uparrow V \\
\hline
\Uparrow \lambda x.k\ x & = \lambda x.k\ x & \longrightarrow_{\eta \mathsf{v} 2} & k & = \Uparrow k \\
\Uparrow \lambda x.(\lambda y.M)\ x & = \lambda x.(\lambda y.\Uparrow M)\ x & \longrightarrow_{\eta \mathsf{v} 2} & \lambda y.\Uparrow M & = \Uparrow \lambda y.M \\
\Uparrow \lambda x.V\ K\ x & = \lambda x.\Uparrow V\ \Uparrow K\ x & = & \Uparrow V\ K \\
\hline
\end{array}
$$

For the simulation of β by $\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta\eta}$ through $\Uparrow$, we must first check:

$$
\begin{array}{ll}
\left\{{}^{\Uparrow V}\!/_{x}\right\} \Uparrow M = \Uparrow \left\{{}^{V}\!/_{x}\right\} M & \left\{{}^{\Uparrow K}\!/_{k}\right\} \Uparrow M \longrightarrow^{*}_{\beta \mathsf{v} 1} \Uparrow \left\{{}^{K}\!/_{k}\right\} M \\
\left\{{}^{\Uparrow V}\!/_{x}\right\} \Uparrow W = \Uparrow \left\{{}^{V}\!/_{x}\right\} W & \left\{{}^{\Uparrow K}\!/_{k}\right\} \Uparrow W \longrightarrow^{*}_{\beta \mathsf{v} 1} \Uparrow \left\{{}^{K}\!/_{k}\right\} W \\
\left\{{}^{\Uparrow V}\!/_{x}\right\} \Uparrow K' = \Uparrow \left\{{}^{V}\!/_{x}\right\} K' & \left\{{}^{\Uparrow K}\!/_{k}\right\} \Uparrow K' \longrightarrow^{*}_{\beta \mathsf{v} 1} \Uparrow \left\{{}^{K}\!/_{k}\right\} K'
\end{array}
$$

The left-hand side equalities and the right-hand side equalities are respectively proved by mutual induction on terms, with the following interesting case:

$$
\begin{array}{rll}
\left\{{}^{\Uparrow K}\!/_{k}\right\} \Uparrow (k\ V) & = & \Uparrow K\ \left\{{}^{\Uparrow K}\!/_{k}\right\} \Uparrow V \\
\text{by i.h.} & \longrightarrow^{*}_{\beta \mathsf{v} 1} & \Uparrow K\ \Uparrow \left\{{}^{K}\!/_{k}\right\} V \\
& \longrightarrow^{*}_{\beta \mathsf{v} 1} & \Uparrow (K\ \left\{{}^{K}\!/_{k}\right\} V) \\
& = & \Uparrow \left\{{}^{K}\!/_{k}\right\} (k\ V)
\end{array}
$$

The penultimate step is justified by the fact that $\Uparrow K\ \Uparrow V \longrightarrow^{*}_{\beta \mathsf{v} 1} \Uparrow (K\ V)$ (it is an equality for $K = k$ or $K = \lambda x.M$ and one $\beta_{\mathsf{v}}1$-reduction step for $K = W\ K'$).

We now check all cases for β-reduction. The last step for the simulation of the β-reduction of a value is an equality if $\left\{{}^{K}\!/_{k}\right\} K' = k'$ and one $\beta_{\mathsf{V}}1$-step otherwise.

$\Uparrow ((\lambda x.M)\ V)$	$=$	$(\lambda x.\Uparrow M)\ \Uparrow V$	
	$\longrightarrow_{\beta_{\mathsf{V}}1}$	$\left\{{}^{\Uparrow V}\!/_{x}\right\} \Uparrow M$	
	$=$	$\Uparrow \left\{{}^{V}\!/_{x}\right\} M$	
$\Uparrow ((\lambda k.k)\ K\ V)$	$=$	$(\lambda k.\lambda x.k\ x)\ \Uparrow K\ \Uparrow V$	
	$\longrightarrow^{*}_{\beta_{\mathsf{V}}2,\beta_{\mathsf{V}}1}$	$\Uparrow K\ \Uparrow V$	
	$\longrightarrow^{*}_{\beta_{\mathsf{V}}1}$	$\Uparrow (K\ V)$	
$\Uparrow ((\lambda k.\lambda x.M)\ K\ V)$	$=$	$(\lambda k.\lambda x.\Uparrow M)\ \Uparrow K\ \Uparrow V$	
	$\longrightarrow_{\beta_{\mathsf{V}}2}$	$\left\{{}^{\Uparrow K}\!/_{k}\right\} (\lambda x.\Uparrow M)\ \Uparrow V$	
	$\longrightarrow^{*}_{\beta_{\mathsf{V}}1}$	$\Uparrow ((\left\{{}^{K}\!/_{k}\right\} \lambda x.M)\ V)$	
$\Uparrow ((\lambda k.W\ K')\ K\ V)$	$=$	$(\lambda k.\lambda x.\Uparrow W\ \Uparrow K'\ x)\ \Uparrow K\ \Uparrow V$	
	$\longrightarrow^{*}_{\beta_{\mathsf{V}}2,\beta_{\mathsf{V}}1}$	$\Uparrow W\ \left\{{}^{\Uparrow K}\!/_{k}\right\} \Uparrow K'\ \Uparrow V$	
	$\longrightarrow^{*}_{\beta_{\mathsf{V}}1}$	$\Uparrow (W\ (\left\{{}^{K}\!/_{k}\right\} K')\ V)$	
$\Uparrow \lambda k'.(\lambda k.K')\ K$	$=$	$\lambda k'.\lambda x.\Uparrow (\lambda k.K')\ \Uparrow K\ x$	
	$\longrightarrow^{*}_{\beta_{\mathsf{V}}2,\beta_{\mathsf{V}}1}$	$\lambda k'.\lambda x.\Uparrow (\left\{{}^{K}\!/_{k}\right\} K')\ x$	as above
	$\longrightarrow^{*}_{\beta_{\mathsf{V}}1}$	$\Uparrow \lambda k'.\left\{{}^{K}\!/_{k}\right\} K'$	
$\Uparrow ((\lambda k.K')\ K)$	$=$	$\lambda x.\Uparrow (\lambda k.K')\ \Uparrow K\ x$	
	$\longrightarrow^{*}_{\beta_{\mathsf{V}}2,\beta_{\mathsf{V}}1}$	$\lambda x.\Uparrow (\left\{{}^{K}\!/_{k}\right\} K')\ x$	as above
	$\longrightarrow_{\eta_{\mathsf{V}}2}$	$\Uparrow \left\{{}^{K}\!/_{k}\right\} K'$	

- The fact that β, η simulate $\beta_{\mathsf{V}}1, \beta_{\mathsf{V}}2, \eta_{\mathsf{V}}1, \eta_{\mathsf{V}}2$ through $\mathsf{Id}_{\lambda^{\mathcal{F}}_{\mathsf{CPS}}}$ is trivial, since the latter are particular cases of the former.

□

Corollary 73 (Confluence of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$) *$\lambda^{\mathcal{F}}_{CPS\beta}$ and $\lambda^{\mathcal{F}}_{CPS\beta\eta}$ are confluent.*

Proof: The first system is confluent as it is the entire β-reduction relation in $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$, the second system is confluent by Theorem 72 and Theorem 5. □

3.3 Moggi's λ_{C}-calculus

Both the refined Reynolds translation and the refined Fischer translations suggest to extend λ_{V} with a construct $\mathsf{let}\ \_ = \_\ \mathsf{in}\ \_$ (reminiscent of our cut-construct for G3ii) with the following semantics:

$$(\mathsf{let}\ x = M\ \mathsf{in}\ N) :_{\mathcal{R}} K = M :_{\mathcal{R}} \lambda x.(N :_{\mathcal{R}} K)$$
$$(\mathsf{let}\ x = M\ \mathsf{in}\ N) :_{\mathcal{F}} K = M :_{\mathcal{F}} \lambda x.(N :_{\mathcal{F}} K)$$

and with the following rules:

$$\begin{array}{lll} M\ N & \longrightarrow & \mathsf{let}\ x = M\ \mathsf{in}\ (x\ N) \quad \text{if } M \text{ is not a value} \\ V\ N & \longrightarrow & \mathsf{let}\ y = N\ \mathsf{in}\ (V\ y) \quad \text{if } N \text{ is not a value} \end{array}$$

Indeed, for both refined translations, a redex of these rules and its reduced form are mapped to the same term.

This extension is related to Moggi's monadic λ-calculus [Mog91], which suggests additional rules to form the CBV-calculus λ_{C} [Mog88][3] defined as follows:

Definition 79 (λ_{C}) The terms of λ_{C} are given by the following grammar:

$$\begin{array}{ll} V, W, \ldots & ::= x \mid \lambda x.M \\ M, N, P, \ldots & ::= V \mid M\ N \mid \mathsf{let}\ x = M\ \mathsf{in}\ N \end{array}$$

The reduction system of λ_{C} is given in Fig. 3.7.

β_{V}	$(\lambda x.M)\ V$	$\longrightarrow$	$\{^{V}/_{x}\}M$
$\mathsf{let}_{\mathsf{V}}$	$\mathsf{let}\ x = V\ \mathsf{in}\ M$	$\longrightarrow$	$\{^{V}/_{x}\}M$
let_1	$M\ N$	$\longrightarrow$	$\mathsf{let}\ x = M\ \mathsf{in}\ (x\ N)$ (M not a value)
let_2	$V\ N$	$\longrightarrow$	$\mathsf{let}\ y = N\ \mathsf{in}\ (V\ y)$ (N not a value)
assoc	$\mathsf{let}\ y = (\mathsf{let}\ x = M\ \mathsf{in}\ N)\ \mathsf{in}\ P$	$\longrightarrow$	$\mathsf{let}\ x = M\ \mathsf{in}\ (\mathsf{let}\ y = N\ \mathsf{in}\ P)$

Figure 3.7: Rules of λ_{C}

Again, η-reduction can be added:

$$\begin{array}{lllll} \eta_{\mathsf{let}} & \mathsf{let}\ x = M\ \mathsf{in}\ x & \longrightarrow & M & \\ \eta_{\mathsf{V}} & \lambda x.(V\ x) & \longrightarrow & V & \text{if } x \notin FV(V) \end{array}$$

And again, in presence of β_V, rule η_{V} has the same effect as the following one:

$$\eta'_{\mathsf{V}} \quad \lambda x.(y\ x) \longrightarrow y \qquad \text{if } x \neq y$$

For various purposes described in the introduction, here we also consider a slight variation of λ_{C}, in which reduction is refined by replacing the reduction rule β_{V} with the following:

$$\mathsf{B} \quad (\lambda x.M)\ N \longrightarrow \mathsf{let}\ x = N\ \mathsf{in}\ M$$

This allows the split of the rule β_{V} into two steps: B followed by $\mathsf{let}_{\mathsf{V}}$. Note that in B we do not require N to be a value. Such a restriction will only apply when reducing $\mathsf{let}\ x = N\ \mathsf{in}\ M$ by rule $\mathsf{let}_{\mathsf{V}}$.

[3] A detailed presentation of these ideas can also be found in [SW97].

System $\lambda_{\mathsf{C}\beta}$ is $\mathsf{B}, \mathsf{let_V}, \mathsf{let}_1, \mathsf{let}_2, \mathsf{assoc}$ and $\lambda_{\mathsf{C}\beta\eta}$ is $\lambda_{\mathsf{C}\beta}, \eta_{\mathsf{let}}, \eta_{\mathsf{V}}$.

In [SF93] it is shown that, in effect, Fischer's translation forms an equational correspondence between (Moggi's original) λ_{C} and $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$. In [SW97], Sabry and Wadler establish not only that Reynolds' translation form an equational correspondence between (Moggi's original) λ_{C} and $\lambda^{\mathcal{R}}_{\mathsf{CPS}}$, but the refined Reynolds translation actually forms a reflection.

3.4 The refined Fischer translation *is* a reflection

In [SW97], Sabry and Wadler also establish that for a particular definition of administrative redex (namely, every β-redex with a binder on the continuation variable k is administrative), the refined Fischer translation cannot be a reflection, and from from λ_{C} to $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ it cannot even be a Galois connection.

Here we show that our (different) choice of administrative redex for the Fischer translation (given in Fig. 3.2) makes it a reflection of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ in our version of λ_{C}, where the rule β_{V} is decomposed into two steps as described above. This will also bring λ_{C} closer to a particular fragment of $\lambda\mathsf{G3}$ called LJQ and studied in Chapter 6.

Lemma 74

1. $\left\{{}^{K'}\!/_{k}\right\}(M :_{\mathcal{F}} K) = M :_{\mathcal{F}} \left\{{}^{K'}\!/_{k}\right\} K$
2. $\left\{{}^{V^{\mathcal{F}}}\!/_{x}\right\}(M :_{\mathcal{F}} K) = \left\{{}^{V}\!/_{x}\right\} M :_{\mathcal{F}} \left\{{}^{V^{\mathcal{F}}}\!/_{x}\right\} K$ *and* $\left\{{}^{V^{\mathcal{F}}}\!/_{x}\right\} W^{\mathcal{F}} = \left(\left\{{}^{V}\!/_{x}\right\} W\right)^{\mathcal{F}}$.
3. *If* $K \longrightarrow_{\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta}} K'$ *then* $M :_{\mathcal{F}} K \longrightarrow_{\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta}} M :_{\mathcal{F}} K'$ *(and similarly for* $\longrightarrow_{\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta\eta}}$*).*

Proof: Straightforward induction on M for the first and third points and on M and W for the second. □

Theorem 75 (Simulation of λ_{C} in $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$) *The reduction relation* $\longrightarrow_{\lambda_{\mathsf{C}\beta}}$ *(resp.* $\longrightarrow_{\lambda_{\mathsf{C}\beta\eta}}$*) is (weakly) simulated by* $\longrightarrow_{\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta}}$ *(resp.* $\longrightarrow_{\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta\eta}}$*) through the refined Fischer translation.*

Proof: By induction on the size of the term being reduced: We check all the root reduction cases, relying on Lemma 74:

$$
\begin{array}{lrll}
((\lambda x.M)\ V){:}_{\mathcal{F}}K & & = & (\lambda k.\lambda x.(M{:}_{\mathcal{F}}k))\ K\ V^{\mathcal{F}} \\
& (\text{Lemma } 74.1) & \longrightarrow_{\beta_{\mathsf{V}}2} & (\lambda x.(M{:}_{\mathcal{F}}K))\ V^{\mathcal{F}} \\
& & = & (\mathsf{let}\ x = V\ \mathsf{in}\ M){:}_{\mathcal{F}}K \\
((\lambda x.M)\ N){:}_{\mathcal{F}}K & & = & N{:}_{\mathcal{F}}\lambda y.(\lambda k.\lambda x.(M{:}_{\mathcal{F}}k))\ K\ y \\
(N \text{ not a value}) & (\text{Lemma } 74) & \longrightarrow^{*}_{\beta_{\mathsf{V}}2,\beta_{\mathsf{V}}1} & N{:}_{\mathcal{F}}\lambda x.(M{:}_{\mathcal{F}}K) \\
& & = & (\mathsf{let}\ x = N\ \mathsf{in}\ M){:}_{\mathcal{F}}K \\
(\mathsf{let}\ x = V\ \mathsf{in}\ M){:}_{\mathcal{F}}K & & = & (\lambda x.M{:}_{\mathcal{F}}K)\ V^{\mathcal{F}} \\
& & \longrightarrow_{\beta_{\mathsf{V}}1} & \left\{{}^{V^{\mathcal{F}}}\!/_{x}\right\}(M{:}_{\mathcal{F}}K) \\
& (\text{Lemma } 74.2) & = & (\left\{{}^{V}\!/_{x}\right\}M){:}_{\mathcal{F}}K \\
(M\ N){:}_{\mathcal{F}}K & & = & M{:}_{\mathcal{F}}\lambda x.(x\ N{:}_{\mathcal{F}}K) \\
(M \text{ not a value}) & & = & (\mathsf{let}\ x = M\ \mathsf{in}\ x\ N){:}_{\mathcal{F}}K \\
(V\ N){:}_{\mathcal{F}}K & & = & N{:}_{\mathcal{F}}\lambda x.(V\ x{:}_{\mathcal{F}}K) \\
(N \text{ not a value}) & & = & (\mathsf{let}\ x = N\ \mathsf{in}\ V\ x){:}_{\mathcal{F}}K \\
(\mathsf{let}\ y = (\mathsf{let}\ x = M\ \mathsf{in}\ N)\ \mathsf{in}\ P){:}_{\mathcal{F}}K & & = & M{:}_{\mathcal{F}}\lambda x.(N{:}_{\mathcal{F}}\lambda y.(P{:}_{\mathcal{F}}K)) \\
& & = & (\mathsf{let}\ x = M\ \mathsf{in}\ (\mathsf{let}\ y = N\ \mathsf{in}\ P)){:}_{\mathcal{F}}K \\
\hline
(\mathsf{let}\ x = M\ \mathsf{in}\ x){:}_{\mathcal{F}}K & & = & M{:}_{\mathcal{F}}\lambda x.K\ x \\
& (\text{Lemma } 74.3) & \longrightarrow_{\eta_{\mathsf{V}}2} & M{:}_{\mathcal{F}}K \\
(\lambda x.V\ x)^{\mathcal{F}} & & = & \lambda k.\lambda x.V^{\mathcal{F}}\ k\ x \\
& & \longrightarrow_{\eta_{\mathsf{V}}1} & V^{\mathcal{F}}
\end{array}
$$

The contextual closure is straightforward as well: only the side-condition "N is not a value" can become false by reduction of N. In that case if $N \longrightarrow_{\lambda_{\mathsf{C}\beta}} V$ we have

$$
\begin{array}{rrl}
N\ M{:}_{\mathcal{F}}K & = & N{:}_{\mathcal{F}}\lambda x.(x\ M{:}_{\mathcal{F}}K) \\
\text{by i.h.} & \longrightarrow^{*}_{\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta}} & V{:}_{\mathcal{F}}\lambda x.(x\ M{:}_{\mathcal{F}}K) \\
& = & (\lambda x.(x\ M{:}_{\mathcal{F}}K))\ V^{\mathcal{F}} \\
& \longrightarrow_{\beta_{\mathsf{V}}1} & V\ M{:}_{\mathcal{F}}K
\end{array}
$$

as well as:

$$
\begin{array}{rrl}
W\ N{:}_{\mathcal{F}}K & = & N{:}_{\mathcal{F}}\lambda x.(W\ x{:}_{\mathcal{F}}K) \\
\text{by i.h.} & \longrightarrow^{*}_{\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta}} & V{:}_{\mathcal{F}}\lambda x.(W\ x{:}_{\mathcal{F}}K) \\
& = & (\lambda x.(W\ x{:}_{\mathcal{F}}K))\ V^{\mathcal{F}} \\
& \longrightarrow_{\beta_{\mathsf{V}}1} & W\ V{:}_{\mathcal{F}}K
\end{array}
$$

and also if M is not a value:

$$
\begin{array}{rrl}
M\ N{:}_{\mathcal{F}}K & = & M{:}_{\mathcal{F}}\lambda x.(x\ N{:}_{\mathcal{F}}K) \\
\text{by i.h.} & \longrightarrow^{*}_{\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta}} & M{:}_{\mathcal{F}}\lambda x.(x\ V{:}_{\mathcal{F}}K) \\
& = & M\ V{:}_{\mathcal{F}}K
\end{array}
$$

and similarly with $\longrightarrow_{\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta\eta}}$ instead of $\longrightarrow_{\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta}}$. □

Definition 80 (The Fischer reverse translation)
We define a translation from $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ to λ_{C}:

$(k\ V)^{\mathcal{F}\mathsf{back}}$	$:=$	$V^{\mathcal{F}\mathsf{back}}$
$((\lambda x.M)\ V)^{\mathcal{F}\mathsf{back}}$	$:=$	$\mathsf{let}\ x = V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ M^{\mathcal{F}\mathsf{back}}$
$(W\ k\ V)^{\mathcal{F}\mathsf{back}}$	$:=$	$W^{\mathcal{F}\mathsf{back}}\ V^{\mathcal{F}\mathsf{back}}$
$(W\ (\lambda x.M)\ V)^{\mathcal{F}\mathsf{back}}$	$:=$	$\mathsf{let}\ x = W^{\mathcal{F}\mathsf{back}}\ V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ M^{\mathcal{F}\mathsf{back}}$
$x^{\mathcal{F}\mathsf{back}}$	$:=$	x
$(\lambda k.\lambda x.M)^{\mathcal{F}\mathsf{back}}$	$:=$	$\lambda x.M^{\mathcal{F}\mathsf{back}}$

Lemma 76

1. $(\{V/x\}\,W)^{\mathcal{F}back} = \left\{V^{\mathcal{F}back}/x\right\}W^{\mathcal{F}back}$ *and*
   $(\{V/x\}\,M)^{\mathcal{F}back} = \left\{V^{\mathcal{F}back}/x\right\}M^{\mathcal{F}back}$.
2. $\mathsf{let}\ x = M^{\mathcal{F}back}\ \mathsf{in}\ N^{\mathcal{F}back} \longrightarrow^{*}_{\lambda_{C\beta}} (\{\lambda x.N/k\}\,M)^{\mathcal{F}back}$ *(if* $k \in \mathsf{FV}(M)$*).*

Proof: The first point is straightforward by induction on W, M. The second is proved by induction on M:

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathsf{let}\ x = (k\ V)^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ N^{\mathcal{F}\mathsf{back}}} \\
& = & \mathsf{let}\ x = V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ N^{\mathcal{F}\mathsf{back}} \\
& = & (\{\lambda x.N/k\}\,(k\ V))^{\mathcal{F}\mathsf{back}} \\
\multicolumn{3}{l}{\mathsf{let}\ x = ((\lambda y.P)\ V)^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ N^{\mathcal{F}\mathsf{back}}} \\
& = & \mathsf{let}\ x = (\mathsf{let}\ y = V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ P^{\mathcal{F}\mathsf{back}})\ \mathsf{in}\ N^{\mathcal{F}\mathsf{back}} \\
& \longrightarrow_{\mathsf{assoc}} & \mathsf{let}\ y = V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ \mathsf{let}\ x = P^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ N^{\mathcal{F}\mathsf{back}} \\
\text{by i.h.} & \longrightarrow^{*}_{\lambda_{C\beta}} & \mathsf{let}\ y = V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ (\{\lambda x.N/k\}\,P)^{\mathcal{F}\mathsf{back}} \\
& = & ((\lambda y.\,\{\lambda x.N/k\}\,P)\ V)^{\mathcal{F}\mathsf{back}} \\
\multicolumn{3}{l}{\mathsf{let}\ x = (W\ k\ V)^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ N^{\mathcal{F}\mathsf{back}}} \\
& = & \mathsf{let}\ x = W^{\mathcal{F}\mathsf{back}}\ V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ N^{\mathcal{F}\mathsf{back}} \\
& = & (W\ (\lambda x.N)\ V)^{\mathcal{F}\mathsf{back}} \\
& = & (\{\lambda x.N/k\}\,(W\ k\ V))^{\mathcal{F}\mathsf{back}} \\
\multicolumn{3}{l}{\mathsf{let}\ x = (W\ (\lambda y.P)\ V)^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ N^{\mathcal{F}\mathsf{back}}} \\
& = & \mathsf{let}\ x = (\mathsf{let}\ y = W^{\mathcal{F}\mathsf{back}}\ V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ P^{\mathcal{F}\mathsf{back}})\ \mathsf{in}\ N^{\mathcal{F}\mathsf{back}} \\
& \longrightarrow_{\mathsf{assoc}} & \mathsf{let}\ y = W^{\mathcal{F}\mathsf{back}}\ V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ \mathsf{let}\ x = P^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ N^{\mathcal{F}\mathsf{back}} \\
\text{by i.h.} & \longrightarrow^{*}_{\lambda_{C\beta}} & \mathsf{let}\ y = W^{\mathcal{F}\mathsf{back}}\ V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ (\{\lambda x.N/k\}\,P^{\mathcal{F}\mathsf{back}}) \\
& = & (W\ (\lambda y.\,\{\lambda x.N/k\}\,P^{\mathcal{F}\mathsf{back}})\ V)^{\mathcal{F}\mathsf{back}} \\
& = & (\{\lambda x.N/k\}\,(W\ (\lambda y.P)\ V))^{\mathcal{F}\mathsf{back}}
\end{array}
$$

□

Theorem 77 (Simulation of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ in λ_{C}) *The reduction relation $\longrightarrow_{\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta}}$ (resp. $\longrightarrow_{\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta\eta}}$) is (weakly) simulated by $\longrightarrow_{\lambda_{\mathsf{C}\beta}}$ (resp. $\longrightarrow_{\lambda_{\mathsf{C}\beta\eta}}$) through $\mathcal{F}\mathsf{back}$.*

Proof: By induction on the size of the term being reduced: We check all root reduction cases, relying on Lemma 76:

$$
\begin{array}{rcl}
((\lambda x.M)\ V)^{\mathcal{F}\mathsf{back}} & = & \mathsf{let}\ x = V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ M^{\mathcal{F}\mathsf{back}} \\
 & \longrightarrow_{\mathsf{let_V}} & \left\{ V^{\mathcal{F}\mathsf{back}} \middle/ x \right\} M^{\mathcal{F}\mathsf{back}} \\
\text{(Lemma 76)} & = & \left(\left\{ V \middle/ x \right\} M\right)^{\mathcal{F}\mathsf{back}} \\
((\lambda k.\lambda x.M)\ k'\ V)^{\mathcal{F}\mathsf{back}} & = & (\lambda x.M^{\mathcal{F}\mathsf{back}})\ V^{\mathcal{F}\mathsf{back}} \\
 & \longrightarrow_{\mathsf{B}} & \mathsf{let}\ x = V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ M^{\mathcal{F}\mathsf{back}} \\
 & = & \left(\left(\left\{ k' \middle/ k \right\} \lambda x.M\right)\ V\right)^{\mathcal{F}\mathsf{back}} \\
((\lambda k.\lambda x.M)\ (\lambda y.N)\ V)^{\mathcal{F}\mathsf{back}} & & \\
 & = & \mathsf{let}\ y = (\lambda x.M^{\mathcal{F}\mathsf{back}})\ V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ N^{\mathcal{F}\mathsf{back}} \\
 & \longrightarrow_{\mathsf{B}} & \mathsf{let}\ y = (\mathsf{let}\ x = V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ M^{\mathcal{F}\mathsf{back}})\ \mathsf{in}\ N^{\mathcal{F}\mathsf{back}} \\
 & = & \mathsf{let}\ y = ((\lambda x.M)\ V)^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ N^{\mathcal{F}\mathsf{back}} \\
\text{(Lemma 76)} & \longrightarrow_{\lambda_{\mathsf{C}\beta}} & \left(\left\{ \lambda y.N^{\mathcal{F}\mathsf{back}} \middle/ k \right\}((\lambda x.M)\ V)\right)^{\mathcal{F}\mathsf{back}}
\end{array}
$$

$$
\begin{array}{rcll}
(\lambda k.\lambda x.V\ k\ x)^{\mathcal{F}\mathsf{back}} & = & \lambda x.V^{\mathcal{F}\mathsf{back}}\ x & \\
 & \longrightarrow_{\eta_{\mathsf{V}}} & V^{\mathcal{F}\mathsf{back}} & \\
((\lambda x.K\ x)\ V)^{\mathcal{F}\mathsf{back}} & \longrightarrow_{\lambda_{\mathsf{C}\beta}} & (K\ V)^{\mathcal{F}\mathsf{back}} & \text{by simulation of } \beta_{\mathsf{V}}1 \\
(W\ (\lambda x.k\ x)\ V)^{\mathcal{F}\mathsf{back}} & = & \mathsf{let}\ x = W^{\mathcal{F}\mathsf{back}}\ V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ x & \\
 & \longrightarrow_{\eta_{\mathsf{let}}} & W^{\mathcal{F}\mathsf{back}}\ V^{\mathcal{F}\mathsf{back}} & \\
 & = & (W\ k\ V)^{\mathcal{F}\mathsf{back}} & \\
(W\ (\lambda x.(\lambda y.M)\ x)\ V)^{\mathcal{F}\mathsf{back}} & & & \\
 & = & \mathsf{let}\ x = W^{\mathcal{F}\mathsf{back}}\ V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ \mathsf{let}\ y = x\ \mathsf{in}\ M^{\mathcal{F}\mathsf{back}} & \\
 & \longrightarrow_{\mathsf{let_V}} & \mathsf{let}\ y = W^{\mathcal{F}\mathsf{back}}\ V^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ M^{\mathcal{F}\mathsf{back}} & \\
 & = & (W\ (\lambda y.M)\ V)^{\mathcal{F}\mathsf{back}} &
\end{array}
$$

□

Lemma 78 *$V \longrightarrow^*_{\lambda_{\mathsf{C}\beta}} V^{\mathcal{F}\mathcal{F}\mathsf{back}}$ and $M \longrightarrow^*_{\lambda_{\mathsf{C}\beta}} (M :_{\mathcal{F}} k)^{\mathcal{F}\mathsf{back}}$.*

Proof: By induction on the size of V, M. The cases for V ($V = x$ and $V = \lambda x.M$) are straightforward, as is the case of $M = V$. For $M = (\mathsf{let}\ x = N'\ \mathsf{in}\ N)$ we have:

$$
\begin{array}{rcll}
M & = & (\mathsf{let}\ x = N'\ \mathsf{in}\ N) & \\
 & \longrightarrow_{\mathsf{C}\beta} & (\mathsf{let}\ x = (N' :_{\mathcal{F}} k)^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ (N :_{\mathcal{F}} k)^{\mathcal{F}\mathsf{back}}) & \text{by i.h.} \\
 & \longrightarrow_{\lambda_{\mathsf{C}\beta}} & \left(\left\{ \lambda x.N :_{\mathcal{F}} k \middle/ k \right\}(N' :_{\mathcal{F}} k)\right)^{\mathcal{F}\mathsf{back}} & \text{(Lemma 76)} \\
 & = & (N' :_{\mathcal{F}} \lambda x.(N :_{\mathcal{F}} k))^{\mathcal{F}\mathsf{back}} & \text{(Lemma 74)} \\
 & = & ((\mathsf{let}\ x = N'\ \mathsf{in}\ N) :_{\mathcal{F}} k)^{\mathcal{F}\mathsf{back}} &
\end{array}
$$

The cases for $M = M_1\ M_2$ are as follows:

$$\begin{array}{lll}
M_1\ M_2 & \longrightarrow_{\lambda_{\mathsf{C}\beta}} & \mathsf{let}\ y = M_1\ \mathsf{in}\ y\ M_2 \\
(M_1 \text{ not a value}) & \longrightarrow^*_{\lambda_{\mathsf{C}\beta}} & \mathsf{let}\ y = (M_1 :_{\mathcal{F}} k)^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ (y\ M_2 :_{\mathcal{F}} k)^{\mathcal{F}\mathsf{back}} \quad \text{by i.h.} \\
 & \longrightarrow^*_{\lambda_{\mathsf{C}\beta}} & (M_1 :_{\mathcal{F}} \lambda y.(y\ M_2 :_{\mathcal{F}} k))^{\mathcal{F}\mathsf{back}} \quad \text{(Lemma 76)} \\
 & = & ((M_1\ M_2) :_{\mathcal{F}} k)^{\mathcal{F}\mathsf{back}} \\
V\ M_2 & \longrightarrow_{\lambda_{\mathsf{C}\beta}} & \mathsf{let}\ y = M_2\ \mathsf{in}\ V\ y \\
(M_2 \text{ not a value}) & \longrightarrow^*_{\lambda_{\mathsf{C}\beta}} & \mathsf{let}\ y = (M_2 :_{\mathcal{F}} k)^{\mathcal{F}\mathsf{back}}\ \mathsf{in}\ (V\ y :_{\mathcal{F}} k)^{\mathcal{F}\mathsf{back}} \quad \text{by i.h.} \\
 & \longrightarrow^*_{\lambda_{\mathsf{C}\beta}} & (M_2 :_{\mathcal{F}} \lambda y.(V\ y :_{\mathcal{F}} k))^{\mathcal{F}\mathsf{back}} \quad \text{(Lemma 76)} \\
 & = & ((V\ M_2) :_{\mathcal{F}} k)^{\mathcal{F}\mathsf{back}} \\
V\ W & \longrightarrow^*_{\lambda_{\mathsf{C}\beta}} & V^{\mathcal{F}\mathcal{F}\mathsf{back}}\ W^{\mathcal{F}\mathcal{F}\mathsf{back}} \quad \text{by i.h.} \\
 & = & (V^{\mathcal{F}}\ k\ W^{\mathcal{F}})^{\mathcal{F}\mathsf{back}} \\
 & = & (V\ W :_{\mathcal{F}} k)^{\mathcal{F}\mathsf{back}}
\end{array}$$

□

Lemma 79 *$V = V^{\mathcal{F}\mathsf{back}\mathcal{F}}$ and $M = M^{\mathcal{F}\mathsf{back}} :_{\mathcal{F}} k$.*

Proof: Straightforward induction on V, M. □

Now we can prove the following:

Theorem 80 (The refined Fischer translation is a reflection)
The refined Fischer translation and $\mathcal{F}\mathsf{back}$ form a reflection in λ_{C} of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$.

Proof: This theorem is just the conjunction of Theorem 75, Theorem 77, Lemma 78 and Lemma 79. □

Corollary 81 (Confluence of λ_{C}) *$\lambda_{\mathsf{C}\beta}$ and $\lambda_{\mathsf{C}\beta\eta}$ are confluent.*

Proof: By Theorem 80 and Theorem 5. □

Conclusion

In this chapter we have investigated the call-by-value λ-calculus, and defined continuation-passing-style translations (and their refinements) in the style of Reynolds and Fischer. We have identified the target calculi and proved confluence of that of Fischer. We then presented Moggi's λ_{C}-calculus and proved that a decomposition of its main rule into two steps allowed the refined Fischer translation to form a reflection. Such a decomposition brings λ_{C} closer to the sequent calculus LJQ as described in Chapter 6.

Chapter 4

Two refinements of the simulation technique

In this chapter, whose contents appeared in [Len05], we develop two refinements of the simulation technique (Corollary 26) that were originally designed for deriving strong normalisation results from the strong normalisation of typed λ-calculus (Theorem 62).

The first technique, presented in section 4.1 and called the *Safeness & Minimality technique*, turns out to be more general and can be applied to any HOC. In contrast to other results, our proof of the main theorem about this technique is only valid in classical logic.

As an example, we show how this technique can be used for the explicit substitution calculus $\lambda\mathsf{x}$ [BR95], yielding a short proof of Preservation of Strong Normalisation (PSN) [BBLRD96]. The technique also allows us to easily derive the strong normalisation of typed terms from that of typed λ-terms. Unfortunately, since the technique is fundamentally classical, it cannot draw advantage of the constructive proofs of strong normalisation such as the one in [JM03] for the simply-typed λ-calculus.

The second technique, presented in section 4.2, is more specifically designed for proving normalisation results that are related to normalisation of β-reduction in λ-calculus. It consists in simulating an HOC in the calculus λI of [Klo80], when a simple attempt of simulation in λ-calculus fails. Its applicability holds in intuitionistic logic, apart maybe from one external result, whose provability in intuitionistic logic remains to be checked.

An example of how this technique can be applied is given in Chapter 5 to prove the PSN property of an explicit substitution calculus called $\lambda\mathsf{lxr}$ with full composition of substitutions, for which standard techniques that we tried all failed. This is a new result. Note that a presentation of $\lambda\mathsf{lxr}$ has been published by Kesner and Lengrand in [KL05, KL06].

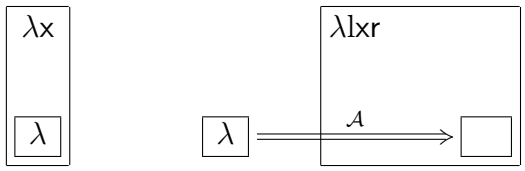


Figure 4.1: Standard and generalised situations for stating PSN

Preservation of Strong Normalisation

An important normalisation property related to λ-calculus is the *Preservation of Strong Normalisation* (*PSN*) [BBLRD96]. It concerns syntactic extensions of λ-calculus with their own reduction relations and states that if a λ-term is strongly normalising for β-reduction, then it is still strongly normalising when considered as a term of the extended calculus, subject to the reductions of the latter. In other words, the reduction relation should not be too big, although it is often required to be big enough to simulate β-reduction.

The definition of the PSN property can be slightly generalised for calculi in which λ-calculus can be *embedded* (by a one-to-one translation, say $\mathcal{A}$) rather than just included. In that case PSN states that if a λ-term is strongly normalising, then its encoding is also strongly normalising.

Fig. 4.1 illustrates the two situations with the examples of the calculus λx [BR95] and the calculus λlxr, introduced in Chapter 5: the former is a syntactic extension of λ-calculus, while the latter can *encode* the λ-calculus.

As discussed below, λx and λlxr are calculi *with explicit substitutions*.

Calculi with explicit substitutions

Indeed, examples of cases where the PSN property is relevant are the calculi with *explicit substitutions*. Such constructs, which use specific substitution constructors and which can be reduced, thus implement the notion of substitution (Definition 43), by means of which β-reduction can usually be decomposed.

In a wider sense we can call *explicit substitution* a construct that can be interpreted as (i.e. mapped to) an (implicit) substitution, with rules manipulating substitution constructors that are valid with respect to this interpretation, and whether or not, among these manipulations, some propagation rules actually implement substitution. In that sense will constructs such as $\langle \_ \dagger \_ . \_ \rangle$ (of section 2.4) or let _ = _ in _ (of section 3.3) be seen as explicit substitutions as well.

Among the possible manipulations of explicit substitutions is the notion of *composition*, in which, for instance, one explicit substitution can be permuted into another. In the literature about explicit substitutions, which has been especially abundant in the last 15 years (e.g. [ACCL91, BBLRD96, BR95, BBLRD96]), an unexpected result was given by Melliès [Mel95] who gave a counter-example

to the PSN property that applies to many calculi with explicit substitutions with a (rather weak, but present) notion of composition, such as for example $\lambda\sigma$ [ACCL91] or $\lambda\sigma_{\Uparrow}$ [HL89].

This phenomenon shows a flaw in the design of these calculi with explicit substitutions in that they are supposed to implement their underlying calculus without losing its good properties such as strong normalisation of simply-typed terms, as Melliès' counter-example implies. PSN is in some sense a *test* property when a calculus with explicit substitutions is introduced.

However, there are many ways to avoid Melliès' counter-example in order to recover the PSN property. One of them is simply to forbid the substitution constructors to cross λ-abstractions [LM99, For02]; another imposes a simple strategy on the calculus with explicit substitutions to mimic exactly the calculus without explicit substitutions [GL98]; another simply consists of avoiding composition of substitutions [BBLRD96]. The first solution leads to *weak* λ-calculi, not able to express *strong* β-equality, which is used for example in implementations of proof-assistants [Coq, HOL]. The second solution exploits very little of the notion of explicit substitutions because they can be neither composed nor even delayed. The last one is simple but composition of substitutions is useful in implementations of higher-order unification [DHK95, DHKP96] or functional abstract machines [HMP96].

The solution [BBLRD96] for the calculus with explicit substitutions λx [BR95], whose syntax extends that of λ-calculus, is investigated in section 4.1.1, with a proof of PSN. Its explicit substitutions are of the form $\langle M/x\rangle N$, in which the variable x is bound in N and the sub-term M is called the *body*. Most explicit substitutions in this dissertation (except from section 9.5 which uses de Bruijn indices [dB72]) are of this form as well. Of course, the techniques of this chapter are likely to be adaptable to other frameworks, e.g. with de Bruijn indices [dB72] or additional parameters.

Another example of calculus with explicit substitutions is λlxr, introduced in Chapter 5 and also presented in [KL05, KL06]. It uses the same form of explicit substitutions as above but requires terms to be linear and hence is not a syntactic extension of λ-calculus. However the latter can be embedded in the former so the generalised notion of PSN makes sense.

Strong normalisation of typed terms & sequent calculus

Apart from PSN, other normalisation results are desirable in calculi with explicit substitutions, such as strong normalisation of typed terms, which can sometimes be inferred from PSN.

Among the calculi with explicit substitutions for which this is extremely relevant are the intuitionistic sequent calculi [Gen35] (with term annotations), e.g. G3ii from Chapter 2. Indeed, the most natural typing rule for an explicit substitution as expressed above is precisely a cut-rule. The notion of computation

in sequent calculi is cut-elimination: the proof of a sequent may be simplified by eliminating the applications of the cut-rule, so that a sequent which is provable with the cut-rule is provable without. Many techniques aimed at proving normalisation results about calculi with explicit substitutions are in fact relevant for cut-elimination in sequent calculus. In other words, termination of cut-elimination processes can often be derived from termination of calculi with explicit substitutions. Of course, in the case of sequent calculi, termination of cut-elimination relies only on the strong normalisation of typed terms.

Failure of the simple simulation technique

The basic idea in proving that a term M of a calculus with explicit substitutions is SN is to use Corollary 26, that is, simulating the reduction steps from M by β-reduction steps from a strongly normalising λ-term $\mathsf{H}(M)$.

For PSN, if $M = \mathcal{A}(t)$ where t is the λ-term known to be SN^β by hypothesis, then we would take $\mathsf{H}(M) = t$.

For sequent calculus, it would be a typed (and hence strongly normalising) λ-term that denotes a proof in natural deduction of the same sequent (using the Curry-Howard correspondence). The idea of simulating cut-elimination by β-reductions has been investigated in [Zuc74].

There is one problem in doing so: an encoding into λ-calculus that allows the simulation needs to interpret explicit substitutions by implicit substitutions such as $\{{}^{u}\!/_{x}\}t$. But should x not be free in t, all reduction steps taking place within the term of which u is the encoding would not induce any β-reduction in $\{{}^{u}\!/_{x}\}t$.

Therefore, the reduction relation that is only weakly simulated, i.e. the one consisting of all the reductions that are not necessarily simulated by at least one β-reduction, is too big to be proved terminating (and very often it is not).

The two techniques developed hereafter are designed to overcome this problem, in a somewhat general setting. The two aforementioned calculi with explicit substitutions $\lambda\mathsf{x}$ and $\lambda\mathsf{lxr}$ respectively illustrate how each can be applied and can provide in particular a proof of the PSN property. The example of $\lambda\mathsf{x}$ is presented just after the first technique, while $\lambda\mathsf{lxr}$ is the object of Chapter 5.

4.1 The safeness & minimality technique

Given a rewrite system R on a set of terms $\mathcal{A}$, the *safeness and minimality technique* presents two subsystems minR and safeR satisfying $\longrightarrow_{\mathsf{safeR}} \subseteq \longrightarrow_{\mathsf{minR}} \subseteq \longrightarrow_{\mathsf{R}}$ and $\mathsf{SN}^{\mathsf{minR}} = \mathsf{SN}^{\mathsf{R}}$.

The intuitive idea is that a reduction step is *minimal* if all the (strict) subterms of the redex are in SN^{R}. Theorem 83 says that in order to prove that $\longrightarrow_{\mathsf{R}}$ is terminating, we can restrict our attention to minimal reductions only, without loss of generality.

Similarly, a reduction step is *safe* if the redex itself is in SN^{R}, which is a stronger requirement than minimality. Theorem 84 says that, whatever R, safe reductions always terminate.

Those ideas are made precise in the following definition:

Definition 81 (Safe & minimal reduction) Given two term rewrite systems h and R satisfying $\longrightarrow_{\mathsf{h}} \subset \longrightarrow_{\mathsf{R}}$,

- the (R-)*minimal* h-system is given by the following rule:

  minh $\quad M \longrightarrow N \quad$ if $M \longrightarrow_{\mathsf{h}} N$ such that for all $P \sqsubset M$, $P \in \mathsf{SN}^{\mathsf{R}}$

- the (R-)*safe* h-system is given by the following rule:

  safeh $\quad M \longrightarrow N \quad$ if $M \longrightarrow_{\mathsf{h}} N$ such that $M \in \mathsf{SN}^{\mathsf{R}}$

In both rules we could require $M \longrightarrow_{\mathsf{h}} N$ to be a root reduction so that M *is* the redex, but although the rules above seem stronger than that, they have the same contextual closure, so we consider the definition above which is the simplest.

Notice that being safe is stronger than being minimal as we have:

$$\longrightarrow_{\mathsf{safeh}} \subseteq \longrightarrow_{\mathsf{minh}} \subseteq \longrightarrow_{\mathsf{h}} \subseteq \longrightarrow_{\mathsf{R}}$$

We also say that a reduction step $M \longrightarrow_{\mathsf{h}} N$ is safe (resp. minimal) if $M \longrightarrow_{\mathsf{safeh}} N$ (resp. $M \longrightarrow_{\mathsf{minh}} N$) and that it is unsafe if not.

Obviously if $\longrightarrow_{\mathsf{h}}$ is finitely branching, then so are $\longrightarrow_{\mathsf{safeh}}$ and $\longrightarrow_{\mathsf{minh}}$.

Whether or not a reduction is safe or minimal is in general not decidable, so proofs that rely on such a case distinction will use classical logic.

Remark 82 We shall constantly use the following facts:

1. $\longrightarrow_{\mathsf{min(safeh)}} = \longrightarrow_{\mathsf{safe(minh)}} = \longrightarrow_{\mathsf{safeh}}$
2. $\longrightarrow_{\mathsf{safe(h,h')}} = \longrightarrow_{\mathsf{safeh,safeh'}}$
3. $\longrightarrow_{\mathsf{min(h,h')}} = \longrightarrow_{\mathsf{minh,minh'}}$

Theorem 83 (Sufficiency of minimal reduction) $SN^{minR} = SN^{R}$
In other words, in order to prove that a term is strongly normalising, it suffices to prove that it is strongly normalising for minimal reductions only.

Proof: The right-to-left inclusion is trivial. We now prove that $\mathsf{SN}^{\mathsf{minR}} \subseteq \mathsf{SN}^{\mathsf{R}}$, by transitive induction in $\mathsf{SN}^{\mathsf{minR}}$ with sub-terms.

Let $M \in \mathsf{SN}^{\mathsf{minR}}$, we have the induction hypothesis that
$\forall N, (M \longrightarrow^{+}_{\mathsf{minR}} N \vee N \sqsubset M) \Rightarrow N \in \mathsf{SN}^{\mathsf{R}}$.

We want to prove that $M \in \mathsf{SN}^{\mathsf{R}}$, so it suffices to check that if $M \longrightarrow_{\mathsf{R}} N$, then $N \in \mathsf{SN}^{\mathsf{R}}$.

We first show that in that case $M \longrightarrow_{\mathsf{minR}} N$. Let Q be the R-redex in M, and let $P \sqsubset Q$. We have $P \sqsubset M$. By the induction hypothesis we get $P \in \mathsf{SN}^{\mathsf{R}}$, so Q is a minR-redex. By contextual closure of minimal reduction, $M \longrightarrow_{\mathsf{minR}} N$.

Again by the induction hypothesis, we get $N \in \mathsf{SN}^{\mathsf{R}}$ as required. □

This proof is valid in intuitionistic logic.

Theorem 84 (Safe reduction terminates) $SN^{safeR} = \mathcal{A}$

Proof: Consider the multi-sets of (R)-strongly normalising terms, and consider the multi-set reductions induced by the reductions $(\longrightarrow_{\mathsf{R}} \cup \sqsupset)^{+}$ on strongly normalising terms. By Corollary 31, these multi-set reductions are terminating.

Considering the mapping ϕ of every term to the multi-set of its R-strongly normalising sub-terms, we can check that the multi-set reductions strongly simulate the safe reductions through ϕ. Hence, from Theorem 22, we get that safe reductions are terminating. □

This proof is valid in intuitionistic logic.

Now the aim of the safeness and minimality technique is to prove the strong normalisation of a system R.

We obtain this by the following theorem, which in general only holds in classical logic. Indeed, it relies on the fact that for the rewrite system R, for all term M we have either $M \in \mathsf{SN}^{\mathsf{R}}$ or $M \notin \mathsf{SN}^{\mathsf{R}}$.

Theorem 85 (Safeness & minimality theorem) *Given a system R and a subsystem R' satisfying $\longrightarrow_{safeR} \subseteq \longrightarrow_{R'} \subseteq \longrightarrow_{minR}$, suppose that we have:*

- *the strong simulation of $\longrightarrow_{minR} \setminus \longrightarrow_{R'}$ in a strongly normalising calculus, through a total relation $\mathcal{Q}$*
- *the weak simulation of $\longrightarrow_{R'}$ through $\mathcal{Q}$*
- *the strong normalisation of $\longrightarrow_{R'}$.*

Then R is strongly normalising.

Proof: This is a direct corollary of Corollary 26, using that $(\longrightarrow_{\mathsf{minR}} \setminus \longrightarrow_{\mathsf{R}'}) \cup \longrightarrow_{\mathsf{R}'} = \longrightarrow_{\mathsf{minR}}$. □

$$\mathsf{x}: \begin{cases} \mathsf{B} & (\lambda x.M)\ N \longrightarrow \langle N/x\rangle M \\ \mathsf{Abs} & \langle N/x\rangle \lambda y.M \longrightarrow \lambda y.\langle N/x\rangle M \\ \mathsf{App} & \langle N/x\rangle M_1\ M_2 \longrightarrow \langle N/x\rangle M_1\ \langle N/x\rangle M_2 \\ \mathsf{VarK} & \langle N/x\rangle y \longrightarrow y \\ \mathsf{VarI} & \langle N/x\rangle x \longrightarrow N \end{cases}$$

Figure 4.2: Reduction rules for $\lambda\mathsf{x}$

Now notice the particular case of the technique when we take $\mathsf{R}' = \mathsf{safeR}$. By Theorem 84 we would directly have its strong normalisation. Unfortunately, this situation is often too coarse, that is to say, the relation $\longrightarrow_{\mathsf{R}'}$ is too small, so that $\longrightarrow_{\mathsf{minR}} \setminus \longrightarrow_{\mathsf{R}'}$ is often too big to be strongly simulated.

Hence, in order to define R', we use the safeness criterion, but the precise definition depends on the calculus that is being treated. In the following section we give the example of the $\lambda\mathsf{x}$-calculus [BR95] and prove a few normalisation results. The technique will also be used in Chapter 6 to prove similar results about the calculus $\overline{\lambda}$ [Her95], the proof being shorter than the existing ones in [DU03] and [Kik04a].

This technique seems close to the notion of *dependency pairs* (see e.g. [AG00]). Formal connections with it should be studied and is left as further work.

4.1.1 A case study: PSN of $\lambda\mathsf{x}$

Definition 82 (Syntax of $\lambda\mathsf{x}$) $\lambda\mathsf{x}$ [BR95] is the syntactic extension of λ-calculus with the aforementioned explicit substitutions:

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

Definition 83 (Reduction in $\lambda\mathsf{x}$) The reduction relation of $\lambda\mathsf{x}$ reduces β-redexes into explicit substitutions which are thence evaluated, as shown in Fig. 4.2.

Note that for this system to be an atomic one (and hence be expressible as an HRS), there should be no obvious free variable in the rules (Definitions 46 and 45). Hence, variables should form a syntactic category of their own: in that case, rule VarK can be read with y standing for a meta-variable of that variable category (otherwise it could stand for a term and then we have the rule of *garbage collection* $\langle N/x\rangle M \longrightarrow M$). If variables form a syntactic category of their own, then the only notion of substitution that is provided by Definition 43 is the substitution of a variable for another variable and nothing else, but now we have a substitution constructor to deal with other substitutions explicitly. Alternatively, one could simply not require the system to be atomic, but then we lose the possibility to express it as a HRS and the advantages thereof.

In this example we take $\mathsf{R}' = \mathsf{safeB}, \mathsf{minx}$.

Lemma 86 $\longrightarrow_{\mathsf{safeB},\mathsf{x}}$ *is terminating.*

Proof: We use for that the LPO based on the following infinite first-order signature and its precedence relation:

$$\mathsf{sub}(\_,\_) \succ \mathsf{ii}(\_,\_) \succ \mathsf{i}(\_) \succ \mathsf{c}^M$$

where for every $M \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$ there is a constant c^M. Those constants are all below $\mathsf{i}(\_)$, and the precedence between them is given by $\mathsf{c}^M \succ \mathsf{c}^N$ if and only if $M \longrightarrow^{+}_{\mathsf{B},\mathsf{x}} N$ or $M \sqsupset N$. By Lemma 47, the precedence relation is terminating.

Encode $\lambda\mathsf{x}$ as follows:

$\overline{M}$	$=$	c^M	if $M \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$
otherwise			
$\overline{\lambda x.M}$	$=$	$\mathsf{i}(\overline{M})$	
$\overline{M\ N}$	$=$	$\mathsf{ii}(\overline{M},\overline{N})$	
$\overline{\langle N/x\rangle M}$	$=$	$\mathsf{sub}(\overline{N},\overline{M})$	

It is quite easy to check that $(\mathsf{safeB},\mathsf{x})$-reduction is simulated by the decreasing LPO through $\overline{(\ )}$, so it is terminating. □

Now consider the following encoding in λ:

$\mathsf{H}(x)$	$=$	x	
$\mathsf{H}(\lambda x.M)$	$=$	$\lambda x.\mathsf{H}(M)$	
$\mathsf{H}(M\ N)$	$=$	$\mathsf{H}(M)\ \mathsf{H}(N)$	
$\mathsf{H}(\langle N/x\rangle M)$	$=$	$\{^{\mathsf{H}(N)}/_{x}\}\mathsf{H}(M)$	if $N \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$
	$=$	$(\lambda x.\mathsf{H}(M))\ \mathsf{H}(N)$	if $N \notin \mathsf{SN}^{\mathsf{B},\mathsf{x}}$

Lemma 87

1. *If* $M \longrightarrow_{\mathsf{minB}} N$ *is unsafe then* $\mathsf{H}(M) \longrightarrow_{\beta} \mathsf{H}(N)$
2. *If* $M \longrightarrow_{\mathsf{minB}} N$ *is safe then* $\mathsf{H}(M) \longrightarrow^{*}_{\beta} \mathsf{H}(N)$
3. *If* $M \longrightarrow_{\mathsf{minx}} N$ *then* $\mathsf{H}(M) = \mathsf{H}(N)$

Proof: Straightforward induction on the derivation of the reduction step, with root reduction as the base case. □

Corollary 88 *If* $\mathsf{H}(M) \in \mathsf{SN}^{\beta}$ *then* $M \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$.

Proof: Direct application of Theorem 85. □

Considering that on pure terms (that is, substitution-free terms), the encoding into λ-calculus is the identity, this gives directly the PSN property for $\lambda\mathsf{x}$.

Corollary 89 (Preservation of Strong Normalisation)
If $t \in \mathcal{SN}^{\beta}$ then $t \in \mathcal{SN}^{\mathsf{B},\mathsf{x}}$.

Notice the subtlety of the definition for the encoding of an explicit substitution:

1. As we have already said, always encoding explicit substitutions as implicit substitutions leads to the weak simulation of too many B-steps, so that the system that is only weakly simulated is too big to be proved terminating.

2. On the other hand, always raising $\langle N/x\rangle M$ into a β-redex would be too strong, because the substitution $\langle N/x\rangle$ can be propagated into the subterms of M but the β-redex cannot be moved around, so the simulation theorem would not hold.

3. Hence, we needed to define an encoding that is a compromise of those two, and the side-condition $N \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$ is precisely the criterion we need:

   - First, the satisfiability of the condition may only evolve in one direction, as it may only become satisfied by some reduction within N, and not the other way around. If it does so, we can simulate this step by reducing the β-redex.
   - Now if $N \notin \mathsf{SN}^{\mathsf{B},\mathsf{x}}$, then the substitution is lifted into a β-redex and for the same reason as in point 2 we cannot simulate the propagation of $\langle N/x\rangle$. So we need to prove that we need not consider reduction steps that propagate a substitution of which the body is not strongly normalising. This is *precisely* the point of minimal reduction: Theorem 83 says that in order to prove a strong normalisation result, we may assume that all sub-terms of the redex are strongly normalising.
   - If on the contrary $N \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$, then we can indeed simulate its propagation, but for the same reason as in point 1, reduction steps within N might only be weakly simulated, but these are precisely what we call safe reductions and we have proved above that they (together with x-reduction) terminate.

4.1.2 A case study: strong normalisation of typed $\lambda\mathsf{x}$

With the Safeness and Minimality technique we can also prove strong normalisation of the typed versions of $\lambda\mathsf{x}$. The rules of Fig. 4.3 define the derivable judgements of the simply-typed $\lambda\mathsf{x}$, which we note as $\Gamma \vdash_{\lambda\mathsf{x}} M : A$.

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma \vdash P : A \qquad \Gamma, (x : A) \vdash M : C}{\Gamma \vdash \langle P/x \rangle M : C}$$

$$\frac{\Gamma, (x : A) \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$$

Figure 4.3: Typing rules for $\lambda\mathsf{x}$

In [Bon01, DL03], it is proved that typed terms are strongly normalising by a reducibility technique. We show here that one application of the Safeness and Minimality technique, apart from PSN, is to derive this result from the strong normalisation of λ-calculus (Theorem 62). Indeed, it turns out that the aforementioned encoding preserves typing:

Theorem 90 (Preservation of simple typing)
If $\Gamma \vdash_{\lambda\mathsf{x}} M : A$ *then* $\Gamma \vdash_{\lambda} \mathsf{H}(M) : A$.

Proof: Straightforward induction on the typing tree. □

Corollary 91 (Strong normalisation of the simply-typed $\lambda\mathsf{x}$)
If $\Gamma \vdash_{\lambda\mathsf{x}} M : A$ *then* $M \in \mathsf{SN}^{B,x}$.

Proof: By combining Theorem 62, Theorem 90 and Corollary 88. □

Moreover, this technique is quite modular, since it only relies on the preservation of typing by the translation H. Hence, any typing system on $\lambda\mathsf{x}$ entails strong normalisation if typed terms are mapped by H to strongly normalising λ-terms (maybe because these can be typed in a corresponding typing system on λ-calculus that entails strong normalisation). We illustrate this by presenting systems with *intersection types* [CD78].

Definition 84 (Types with intersections) For the purpose of this section only, we extend the syntax of types (which, until now, were just implicational formulae —Definition 55). The set of *types with intersections* is defined by the grammar:

$$A, B ::= p \mid A \to B \mid A \cap B$$

The constructor $\cap$ is called the *intersection*.

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash M : B}{\Gamma \vdash M : A \cap B} \qquad \frac{\Gamma \vdash M : A_1 \cap A_2}{\Gamma \vdash M : A_i}\ i \in \{1, 2\}$$

Figure 4.4: Intersection types for λ-calculus

Then we can add the two typing rules of Fig. 4.4 to those of the simply-typed λ-calculus (presented in Fig. 2.4).

Remark 92 Note that the left-hand side rule of Fig. 4.4 is not unconditional w.r.t. the general system, since the two premisses require the *same* term M. This has long prevented the understanding of intersection types via the Curry-Howard paradigm. Investigations into some logical counterpart to intersection types can be found in [PRDRR05].

We write $\Gamma \vdash_{\lambda\cap} M : A$ when the sequent is derivable in the typing system thus obtained, which was introduced in [CD78] and characterises SN^{β}:

Theorem 93 (Strong Normalisation of λ-calculus with intersections)
$\Gamma \vdash_{\lambda\cap} M : A$ *if and only if* $M \in \mathsf{SN}^{\beta}$.

Proof: See e.g. [Pot80]. □

Similarly, when the three inference rules of Fig. 4.5 are added to those of Fig. 4.3, we obtain a typing system whose derivable sequents we denote like $\Gamma \vdash_{\lambda\mathsf{x}\cap} M : A$.

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash M : B}{\Gamma \vdash M : A \cap B} \qquad \frac{\Gamma \vdash M : A_1 \cap A_2}{\Gamma \vdash M : A_i}\ i \in \{1,2\}$$

$$\frac{\Gamma \vdash M : A \qquad \Delta \vdash N : B \qquad x \notin \mathsf{Dom}(\Gamma)}{\Gamma \vdash \langle N/x \rangle M : A}$$

Figure 4.5: Intersection types for $\lambda\mathsf{x}$

This typing system echoes that of intersection types for λ-calculus, since it has the interesting property of characterising $\mathsf{SN}^{\mathsf{B},\mathsf{x}}$ [LLD+04]:

Theorem 94 (Capturing strongly normalising terms)
If $M \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$ *then there exist* Γ *and* A *such that* $\Gamma \vdash_{\lambda\mathsf{x}\cap} M : A$.

In [LLD+04], the converse (typed terms are strongly normalising) is also proved by a reducibility technique. Again we show that the Safeness and Minimality technique applies here to derive this result from the strong normalisation of λ-calculus with intersection types (Theorem 93). Indeed, the aforementioned encoding also preserves typing with intersections:

Theorem 95 (Preservation of typing with intersections)
If $\Gamma \vdash_{\lambda\mathsf{x}\cap} M : A$ *then* $\Gamma \vdash_{\lambda\cap} \mathsf{H}(M) : A$.

Proof: Straightforward induction on the typing tree. □

Hence, we also get:

Corollary 96 (Strong normalisation of $\lambda\mathsf{x}$ typed with intersections) *If* $\Gamma \vdash_{\lambda\mathsf{x}\cap} M : A$ *then* $M \in \mathsf{SN}^{\mathcal{B},x}$.

Proof: By combining Theorem 93, Theorem 95 and Corollary 88. □

Often, this kind of strong normalisation result is derived from the PSN property by lifting the explicit substitutions into β-redexes [Her95], but this is precisely what the encoding does in the necessary places, so that Corollary 88 is a shortcut of Herbelin's technique.

4.2 The simulation technique using a memory operator

In this section we present another refinement of the simulation technique to prove normalisation results of a calculus, henceforth called *the calculus*, that is related to λ-calculus, for instance a calculus with explicit substitutions. In case a simple simulation in λ-calculus fails, as described in the introduction, we suggest to use instead the λI-calculus of [Klo80], based on earlier work by [Chu41, Ned73]. We refer the reader to [Sør97, Xi97] for a survey on different techniques based on the λI-calculus to infer normalisation properties.

On the one hand, λI extends the syntax of λ-calculus with a "memory operator" so that, instead of being thrown away, a term N can be retained and carried along in a construct $[-, N]$. With this operator, those bodies of substitutions are encoded that would otherwise disappear, as described in the introduction. On the other hand, λI restricts λ-abstractions to variables that have at least one free occurrence, so that β-reduction never erases its argument.

Performing a simulation in λI requires the encoding to be non-deterministic, i.e. we define a relation $\mathcal{H}$ between *the calculus* and λI, and the reason for this is that, since the reductions in λI are non-erasing reductions, we need to add this memory operator at random places in the encoding, using such a rule:

$$\frac{M \;\mathcal{H}\; T}{M \;\mathcal{H}\; [T, U]}\, U \in \Lambda I$$

where ΛI is the set of λI-terms.

For instance, if *the calculus* is λ-calculus itself, we would have $\lambda x.x \;\mathcal{H}\; \lambda x.[x, x]$ but also $\lambda x.x \;\mathcal{H}\; [\lambda x.x, \lambda z.z]$, so that both $\lambda x.[x, x]$ and $[\lambda x.x, \lambda z.z]$ (and also $\lambda x.x$) are encodings of $\lambda x.x$.

The reduction relation of *the calculus* must then satisfy the hypotheses of Corollary 26. Namely, it should be the union of a reduction relation $\longrightarrow_Y$ that is strongly simulated by $\longrightarrow_{\beta,\pi}$ through $\mathcal{H}$ and a terminating reduction relation $\longrightarrow_Z$ that is weakly simulated by $\longrightarrow_{\beta,\pi}$ through $\mathcal{H}$. We then need the fact

that every term M of *the calculus* can be encoded into a strongly normalising term of λI, to start off the simulations. This depends on *the calculus*, but the following method generally works:

1. Encode the term M as a strongly normalising λ-term t, such that no subterm is lost, i.e. *not* using implicit substitutions. For PSN, the original λ-term would do, because it is strongly normalising by hypothesis; for a proof-term of sequent calculus, t would be a λ-term typed in an appropriate typing system, the typing tree of which is derived from the proof-tree of the sequent (we would get $t \in \mathsf{SN}^{\beta}$ using a theorem stating that typed terms are SN^{β}).

2. Using a translation i from λ-calculus to λI, introduced in this section, prove that $\mathsf{i}(t)$ reduces to one of the non-deterministic encodings of M in λI, that is, that there is a term T such that $M \mathcal{H} T$ and $\mathsf{i}(t) \longrightarrow^{*}_{\beta,\pi} T$.

In this section we prove that if a λ-term t is strongly normalising for β-reductions, then $\mathsf{i}(t)$ is weakly normalising in λI. The proof simply consists of simulating an adequate reduction sequence that starts from t and ends with a normal form, the encoding of which is a normal form of λI. What makes this simulation work is the fact that the reduction sequence is provided by a *perpetual strategy*, i.e. a strategy that terminates on a term only if it is strongly normalising. Also, weak normalisation implies strong normalisation in λI [Ned73], so $\mathsf{i}(t)$ is strongly normalising, as well as the above λI-term T.

The technique is summarised in Fig. 4.6.

As we shall see, this technique works for proving PSN of the explicit substitution calculus $\lambda\mathsf{lxr}$ of chapter 5. Furthermore, it can be combined with the safeness and minimality technique which provides proofs of strong normalisation for various sequent calculi, and it is, we believe, likely to be applicable to many other calculi.

4.2.1 The λI-calculus

Definition 85 (Grammar of λI) The set ΛI of terms of the λI-calculus of [Klo80] is defined by the following grammar:

$$T, U ::= x \mid \lambda x.T \mid T\ U \mid [T, U]$$

with the additional restriction that every abstraction $\lambda x.T$ satisfies $x \in \mathsf{FV}(T)$.

We denote lists of λI-terms using vectors, and if $\overrightarrow{T} = T_1, \ldots, T_n$, then $U\ \overrightarrow{T}$ denotes $U\ T_1 \ldots T_n$ and $[U, \overrightarrow{T}]$ denotes $[\ldots[U, T_1], \ldots, T_n]$, assuming that these expressions denote U when $n = 0$.

The following property is straightforward by induction on terms.

the calculus	λ	λI

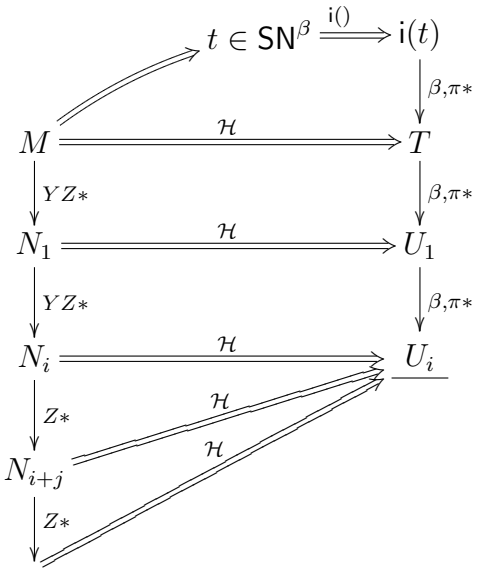


Figure 4.6: The general technique to prove that $M \in \mathsf{SN}$

Lemma 97 (Stability under substitution [Klo80])
If $T, U \in \Lambda I$, then $\{^U\!/_x\}T \in \Lambda I$.

Proof: By induction on T. □

Definition 86 (Reduction system of λI) The reduction rules are:

$$\begin{array}{lll} (\beta) & (\lambda x.T)\ U & \to \{^U\!/_x\}T \\ (\pi) & [T,U]\ T' & \to [T\ T', U] \end{array}$$

The following remark is straightforward [Klo80]:

Remark 98 If $T \longrightarrow_{\beta,\pi} T'$ then $\mathsf{FV}(T) = \mathsf{FV}(T')$ and $\{^T\!/_x\}U \longrightarrow^+_{\beta,\pi} \{^{T'}\!/_x\}U$ provided that $x \in \mathsf{FV}(U)$.

4.2.2 Simulating the perpetual strategy

We may want to use the technique of simulation in λI with some calculi that annotate λ-abstractions with types, and with others that do not. Indeed, one of the applications is the normalisation of systems in type theory (possibly with dependent types), so we also consider Π-types. In order to express the technique in its most general form, we present it with a mixed syntax called $\lambda^?$-calculus (to suggest that λ may or may not be annotated with types):

Definition 87 ($\lambda^?$-calculus) The *annotated?*-λ-calculus, denoted $\lambda^?$-calculus, uses the following syntax:

$$M, N, A, B ::= x \mid s \mid \Pi x^A.B \mid \lambda x^A.M \mid \lambda x.M \mid M\ N$$

where x ranges over a denumerable set of variables, and s ranges over a set of constants.

The reduction rules are

$$\begin{array}{lll} \beta^t & (\lambda x^A.M)\ N & \longrightarrow \left\{{}^{N}\!/_{x}\right\} M \\ \beta & (\lambda x.M)\ N & \longrightarrow \left\{{}^{N}\!/_{x}\right\} M \end{array}$$

Again, we denote lists of $\lambda^?$-terms using vectors, and if $\overrightarrow{t} = t_1, \ldots, t_n$, then $u\ \overrightarrow{t}$ denotes $u\ t_1 \ldots t_n$.

Definition 88 (Annotations)

- *Fully annotated terms* are those terms that have no construct $\lambda x.M$. The fragment of fully annotated terms is stable under β^t-reductions, so that β-reductions never apply and hence $\mathsf{SN}^{\beta^t} = \mathsf{SN}^{\beta^t,\beta}$ for that fragment.
- We define the notion of *type-annotation* as the smallest transitive, reflexive, context-closed relation $\lhd$ such that $\lambda x.M \lhd \lambda x^A.M$.

Note that, for a fully annotated term N, $N \lhd P$ implies $N = P$.

Lemma 99 *If $M \lhd M'$ and $M \longrightarrow_{\beta^t,\beta} N$ then there is a N' such that $N \lhd N'$ and $M' \longrightarrow_{\beta^t,\beta} N'$.*

Proof: By induction on the derivation of $M \lhd M'$. □

Corollary 100 (Strong normalisation with fewer type annotations) *If $M \lhd M'$ and $M' \in \mathcal{SN}^{\beta^t,\beta}$ then $M \in \mathcal{SN}^{\beta^t,\beta}$.*

Proof: By Theorem 22 ($\longrightarrow_{\beta^t,\beta}$ strongly simulates itself through $\lhd$). □

We now proceed with the connections between the $\lambda^?$-calculus and λI, with some results based on simulation again.

Definition 89 (Encoding of $\lambda^?$-calculus into λI) We encode the $\lambda^?$-calculus into λI as follows:

$$\begin{array}{lll} \mathsf{i}(x) & = x & \\ \mathsf{i}(\lambda x.t) & = \lambda x.\mathsf{i}(t) & x \in \mathsf{FV}(t) \\ \mathsf{i}(\lambda x.t) & = \lambda x.[\mathsf{i}(t), x] & x \notin \mathsf{FV}(t) \\ \mathsf{i}(\lambda x^A.t) & = [\mathsf{i}(\lambda x.t), \mathsf{i}(A)] & \\ \mathsf{i}(t\ u) & = \mathsf{i}(t)\ \mathsf{i}(u) & \\ \mathsf{i}(s) & = \wp & \\ \mathsf{i}(\Pi x^A.B) & = \wp\ [\mathsf{i}(\lambda x.t), \mathsf{i}(A)] & \end{array}$$

where $\wp$ is a dummy variable that does not appear in the term that is encoded.

Lemma 101 *For any $\lambda^?$-terms t and u,*

1. $\mathsf{FV}(\mathsf{i}(t)) = \mathsf{FV}(t)$
2. $\left\{{}^{\mathsf{i}(u)}/_{x}\right\} \mathsf{i}(t) = \mathsf{i}(\{{}^{u}/_{x}\} t)$

Proof: Straightforward induction on t. □

Definition 90 (Relation between $\lambda^?$ & λI) The relation $\mathcal{G}$ between $\lambda^?$-terms and λI-terms is given by the rules of Fig. 4.7.

$$\frac{A\ \mathcal{G}\ T \quad B\ \mathcal{G}\ U \quad x \in \mathsf{FV}(U)}{\Pi x^A.B\ \mathcal{G}\ \wp\ [\lambda x.U, T]}\ \mathcal{G}\Pi \qquad \frac{\forall j \qquad t_j\ \mathcal{G}\ T_j}{(x\ \overrightarrow{t_j})\ \mathcal{G}\ (x\ \overrightarrow{T_j})}\ \mathcal{G}\mathsf{var}$$

$$\frac{t\ \mathcal{G}\ T \quad x \in \mathsf{FV}(T)}{\lambda x.t\ \mathcal{G}\ \lambda x.T}\ \mathcal{G}\lambda \qquad \frac{}{((\lambda x.t)\ t'\ \overrightarrow{t_j})\ \mathcal{G}\ \mathsf{i}((\lambda x.t)\ t'\ \overrightarrow{t_j})}\ \mathcal{G}\beta_1$$

$$\frac{t\ \mathcal{G}\ T \quad A\ \mathcal{G}\ U \quad x \in \mathsf{FV}(T)}{\lambda x^A.t\ \mathcal{G}\ [\lambda x.T, U]}\ \mathcal{G}\lambda^t \qquad \frac{t'\ \mathcal{G}\ T' \qquad x \notin \mathsf{FV}(t)}{((\lambda x.t)\ t'\ \overrightarrow{t_j})\ \mathcal{G}\ (\mathsf{i}(\lambda x.t)\ T'\ \overrightarrow{\mathsf{i}(t_j)})}\ \mathcal{G}\beta_2$$

$$\frac{}{s\ \mathcal{G}\ \wp}\ \mathcal{G}\mathsf{c} \qquad \frac{}{((\lambda x^A.t)\ t'\ \overrightarrow{t_j})\ \mathcal{G}\ \mathsf{i}((\lambda x^A.t)\ t'\ \overrightarrow{t_j})}\ \mathcal{G}{\beta^t}_1$$

$$\frac{t\ \mathcal{G}\ T \qquad N \in \mathsf{nf}^{\beta,\pi}}{t\ \mathcal{G}\ [T, N]}\ \mathcal{G}\mathsf{weak} \qquad \frac{t'\ \mathcal{G}\ T' \quad A\ \mathcal{G}\ U \quad x \notin \mathsf{FV}(t)}{((\lambda x^A.t)\ t'\ \overrightarrow{t_j})\ \mathcal{G}\ ([\mathsf{i}(\lambda x.t), U]\ T'\ \overrightarrow{\mathsf{i}(t_j)})}\ \mathcal{G}{\beta^t}_2$$

Figure 4.7: Relation between $\lambda^?$ & λI

Lemma 102

1. *If $t \in \mathsf{nf}^{\beta^t}$ and $t\ \mathcal{G}\ T$, then $T \in \mathsf{nf}^{\beta,\pi}$.*
2. *For any $\lambda^?$-term t, $t\ \mathcal{G}\ \mathsf{i}(t)$.*

Proof:

1. By induction on the proof tree associated to $t\ \mathcal{G}\ T$, one can check that no β and no π-redex is introduced, since rules $\mathcal{G}\beta_1$, $\mathcal{G}\beta_2$, $\mathcal{G}{\beta^t}_1$ and $\mathcal{G}{\beta^t}_2$ are forbidden by the hypothesis that t is a β-normal form.
2. By induction on t:

- If $t = x\ \overrightarrow{t_j}$, then by induction hypothesis $t_j\ \mathcal{G}\ \mathsf{i}(t_j)$ for all j and then we can apply $\mathcal{G}\mathsf{var}$.
- If $t = (\lambda x.t')\ u\ \overrightarrow{t_j}$, then it suffices to use rules $\mathcal{G}\beta_1$.
- If $t = (\lambda x^A.t')\ u\ \overrightarrow{t_j}$, then it suffices to use rules $\mathcal{G}{\beta^t}_1$.
- If $t = \lambda x.u$ then by induction hypothesis $u\ \mathcal{G}\ \mathsf{i}(u)$. If $x \in \mathsf{FV}(u)$, then $\mathsf{i}(t) = \lambda x.\mathsf{i}(u)$ and $t\ \mathcal{G}\ \mathsf{i}(t)$ by rule $\mathcal{G}\lambda$. If $x \notin \mathsf{FV}(u)$, then $\mathsf{i}(t) = \lambda x.[\mathsf{i}(u), x]$, and thus $u\ \mathcal{G}\ [\mathsf{i}(u), x]$ by rule $\mathcal{G}\mathsf{weak}$ and $t\ \mathcal{G}\ \mathsf{i}(t)$ by rule $\mathcal{G}\lambda$.
- If $t = \lambda x^A.u$ then by induction hypothesis $u\ \mathcal{G}\ \mathsf{i}(u)$ and $A\ \mathcal{G}\ \mathsf{i}(A)$. If $x \in \mathsf{FV}(u)$, then $\mathsf{i}(t) = [\lambda x.\mathsf{i}(u), \mathsf{i}(A)]$ and $t\ \mathcal{G}\ \mathsf{i}(t)$ by rule $\mathcal{G}\lambda^t$. If $x \notin \mathsf{FV}(u)$, then $\mathsf{i}(t) = [\lambda x.[\mathsf{i}(u), x], \mathsf{i}(A)]$, and thus $u\ \mathcal{G}\ [\mathsf{i}(u), x]$ by rule $\mathcal{G}\mathsf{weak}$ and $t\ \mathcal{G}\ \mathsf{i}(t)$ by rule $\mathcal{G}\lambda^t$.
- If $t = s$, then clearly $s\ \mathcal{G}\ \wp$.
- If $t = \Pi x^A.B$, then by induction hypothesis $A\ \mathcal{G}\ \mathsf{i}(A)$ and $B\ \mathcal{G}\ \mathsf{i}(B)$. If $x \in \mathsf{FV}(B)$ then $\mathsf{i}(\Pi x^A.B) = \wp\ [\lambda x.\mathsf{i}(B), \mathsf{i}(A)]$ and $t\ \mathcal{G}\ \mathsf{i}(t)$ by rule $\mathcal{G}\Pi$. If $x \in \mathsf{FV}(B)$ then $\mathsf{i}(\Pi x^A.B) = \wp\ [\lambda x.[\mathsf{i}(B), x], \mathsf{i}(A)]$, and thus $B\ \mathcal{G}\ [\mathsf{i}(B), x]$ by rule $\mathcal{G}\mathsf{weak}$ and $t\ \mathcal{G}\ \mathsf{i}(t)$ by rule $\mathcal{G}\Pi$.

□

Definition 91 (A reduction strategy for $\lambda^?$) We define a reduction relation $\rightsquigarrow$ for $\lambda^?$-terms by the rules of Fig. 4.8.

Remark 103 $\rightsquigarrow \subseteq \longrightarrow_{\beta^t\beta}$
If t is not a $\beta^t\beta$-normal form, then there is a $\lambda^?$-term t' such that $t \rightsquigarrow t'$.

Remark 104 Although we do not need it in the rest of the proof, it is worth mentioning that, at least in the fragment of the untyped λ-calculus, the relation $\rightsquigarrow$ defines a *perpetual strategy* w.r.t. β-reduction, i.e. if M is not β-strongly normalising and $M \rightsquigarrow M'$, then neither is M' [vRSSX99].

Theorem 105 (Strong simulation of $\rightsquigarrow$ in λI)
$\longrightarrow_{\beta,\pi}$ *strongly simulates* $\rightsquigarrow$ *through* $\mathcal{G}$.

Proof:

$\mathsf{perp}\beta_1)$ $(\lambda x.t)\ t'\ \overrightarrow{t_j} \rightsquigarrow \{{}^{t'}/_{x}\}t\ \overrightarrow{t_j}$

- $x \in \mathsf{FV}(t)$:
  The last rule used to prove $u\ \mathcal{G}\ U$ must be $\mathcal{G}\beta_1$ (possibly followed by several steps of $\mathcal{G}\mathsf{weak}$), so

$$\begin{array}{rcl} U & = & [\lambda x.\mathsf{i}(t)\ \mathsf{i}(t')\ \overrightarrow{\mathsf{i}(t_j)}, \overrightarrow{N}] \\ & \longrightarrow_\beta & [\{{}^{\mathsf{i}(t')}/_{x}\}\mathsf{i}(t)\ \overrightarrow{\mathsf{i}(t_j)}, \overrightarrow{N}] \\ \text{(Lemma 101.2)} & = & [\mathsf{i}(\{{}^{t'}/_{x}\}t\ \overrightarrow{t_j}), \overrightarrow{N}] \end{array}$$

$$\frac{t \rightsquigarrow t'}{x\ \overrightarrow{t_j}\ t\ \overrightarrow{p_j} \rightsquigarrow x\ \overrightarrow{t_j}\ t'\ \overrightarrow{p_j}}\ \mathsf{perp\text{-}var} \qquad \frac{t \rightsquigarrow t'}{\lambda x.t \rightsquigarrow \lambda x.t'}\ \mathsf{perp}\lambda$$

$$\frac{t \rightsquigarrow t'}{\lambda x^A.t \rightsquigarrow \lambda x^A.t'}\ \mathsf{perp}\lambda^t_1 \qquad \frac{A \rightsquigarrow A'}{\lambda x^A.t \rightsquigarrow \lambda x^{A'}.t}\ \mathsf{perp}\lambda^t_2$$

$$\frac{x \in \mathsf{FV}(t) \vee t' \in \mathsf{nf}^{\beta^t\beta}}{(\lambda x.t)\ t'\ \overrightarrow{t_j} \rightsquigarrow \left\{{}^{t'}\!/_x\right\} t\ \overrightarrow{t_j}}\ \mathsf{perp}\beta_1$$

$$\frac{t' \rightsquigarrow t'' \qquad x \notin \mathsf{FV}(t)}{(\lambda x.t)\ t'\ \overrightarrow{t_j} \rightsquigarrow (\lambda x.t)\ t''\ \overrightarrow{t_j}}\ \mathsf{perp}\beta_2$$

$$\frac{x \in \mathsf{FV}(t) \vee t', A \in \mathsf{nf}^{\beta^t\beta}}{(\lambda x^A.t)\ t'\ \overrightarrow{t_j} \rightsquigarrow \left\{{}^{t'}\!/_x\right\} t\ \overrightarrow{t_j}}\ \mathsf{perp}\beta^t{}_1$$

$$\frac{t' \rightsquigarrow t'' \qquad x \notin \mathsf{FV}(t)}{(\lambda x^A.t)\ t'\ \overrightarrow{t_j} \rightsquigarrow (\lambda x^A.t)\ t''\ \overrightarrow{t_j}}\ \mathsf{perp}\beta^t{}_2$$

$$\frac{A \rightsquigarrow A' \qquad x \notin \mathsf{FV}(t)}{(\lambda x^A.t)\ t'\ \overrightarrow{t_j} \rightsquigarrow (\lambda x^{A'}.t)\ t'\ \overrightarrow{t_j}}\ \mathsf{perp}\beta^t{}_3$$

$$\frac{A \rightsquigarrow A'}{\Pi x^A.B \rightsquigarrow \Pi x^{A'}.B}\ \mathsf{perp}\Pi_1 \qquad \frac{B \rightsquigarrow B'}{\Pi x^A.B \rightsquigarrow \Pi x^A.B'}\ \mathsf{perp}\Pi_2$$

Figure 4.8: A reduction strategy for $\lambda^?$

Then by Lemma 102.2, $\left\{{}^{t'}\!/_x\right\} t\ \overrightarrow{t_j}\ \mathcal{G}\ \mathsf{i}(\left\{{}^{t'}\!/_x\right\} t\ \overrightarrow{t_j})$ and by rule $\mathcal{G}\mathsf{weak}$, $\left\{{}^{t'}\!/_x\right\} t\ \overrightarrow{t_j}\ \mathcal{G}\ [\mathsf{i}(\left\{{}^{t'}\!/_x\right\} t\ \overrightarrow{t_j}), \overrightarrow{N}]$.

– $x \notin \mathsf{FV}(t)$:
It means that t' is a β-normal form and $\left\{{}^{t'}\!/_x\right\} t\ \overrightarrow{t_j} = t\ \overrightarrow{t_j}$. The last rule used to prove $u\ \mathcal{G}\ U$ must be $\mathcal{G}\beta_1$ or $\mathcal{G}\beta_2$ (possibly followed by several steps of $\mathcal{G}\mathsf{weak}$), so in both cases we have $U = [\lambda x.[\mathsf{i}(t), x]\ T'\ \overrightarrow{\mathsf{i}(t_j)}, \overrightarrow{N}]$ with $t'\ \mathcal{G}\ T'$ (using Lemma 102.2 in the former case where $T' = \mathsf{i}(t')$). By Lemma 102.1, T' is a β,π-normal form. Now $U \longrightarrow_\beta [[\left\{{}^{T'}\!/_x\right\} \mathsf{i}(t), T']\ \overrightarrow{\mathsf{i}(t_j)}, \overrightarrow{N}]$. But by Lemma 101.1,

$x \notin \mathsf{FV}(\mathsf{i}(t))$ so the above term is $[[\mathsf{i}(t), T']\ \overrightarrow{\mathsf{i}(t_j)}, \overrightarrow{N}]$, which reduces by π to $[\mathsf{i}(t)\ \overrightarrow{\mathsf{i}(t_j)}, T', \overrightarrow{N}] = [\mathsf{i}(t\ \overrightarrow{t_j}), T', \overrightarrow{N}]$. By Lemma 102.2 and rule $\mathcal{G}\mathsf{weak}$, we get $t\ \overrightarrow{t_j}\ \mathcal{G}\ [\mathsf{i}(t\ \overrightarrow{t_j}), T', \overrightarrow{N}]$.

$\mathsf{perp}\beta_2$) $(\lambda x.t)\ t'\ \overrightarrow{t_j} \rightsquigarrow (\lambda x.t)\ t''\ \overrightarrow{t_j}$ with $t' \rightsquigarrow t''$ and $x \notin \mathsf{FV}(t)$.

The last rule used to prove $u\ \mathcal{G}\ U$ must be $\mathcal{G}\beta_1$ or $\mathcal{G}\beta_2$ (possibly followed by several steps of $\mathcal{G}\mathsf{weak}$), so in both cases $U = [\lambda x.[\mathsf{i}(t), x]\ T'\ \overrightarrow{\mathsf{i}(t_j)}, \overrightarrow{N}]$ with $t'\ \mathcal{G}\ T'$ (using Lemma 102.2 in the former case where $T' = \mathsf{i}(t')$). By induction hypothesis, there is a term T'' such that $T' \longrightarrow^+_{\beta,\pi} T''$ and $t''\ \mathcal{G}\ T''$.
Hence, $U \longrightarrow^+_{\beta,\pi} [\lambda x.[\mathsf{i}(t), x]\ T''\ \overrightarrow{\mathsf{i}(t_j)}, \overrightarrow{N}]$. By application of the rule $\mathcal{G}\beta_2$, $(\lambda x.t)\ t''\ \overrightarrow{t_j}\ \mathcal{G}\ \lambda x.[\mathsf{i}(t), x]\ T''\ \overrightarrow{\mathsf{i}(t_j)}$, and we use rule $\mathcal{G}\mathsf{weak}$ to conclude.

$\mathsf{perp}\beta^t{}_1$) $(\lambda x^A.t)\ t'\ \overrightarrow{t_j} \rightsquigarrow \{t'/x\}t\ \overrightarrow{t_j}$

– $x \in \mathsf{FV}(t)$:
The last rule used to prove $u\ \mathcal{G}\ U$ must be $\mathcal{G}\beta^t{}_1$ (possibly followed by several steps of $\mathcal{G}\mathsf{weak}$), so

$$\begin{array}{rcl} U & = & [[\lambda x.\mathsf{i}(t), \mathsf{i}(A)]\ \mathsf{i}(t')\ \overrightarrow{\mathsf{i}(t_j)}, \overrightarrow{N}] \\ & \longrightarrow^+_{\pi} & [\lambda x.\mathsf{i}(t)\ \mathsf{i}(t')\ \overrightarrow{\mathsf{i}(t_j)}, \mathsf{i}(A), \overrightarrow{N}] \\ & \longrightarrow_{\beta} & [\{\mathsf{i}(t')/x\}\mathsf{i}(t)\ \overrightarrow{\mathsf{i}(t_j)}, \mathsf{i}(A), \overrightarrow{N}] \\ \text{(Lemma 101.2)} & = & [\mathsf{i}(\{t'/x\}t\ \overrightarrow{t_j}), \mathsf{i}(A), \overrightarrow{N}] \end{array}$$

Then by Lemma 102.2, $\{t'/x\}t\ \overrightarrow{t_j}\ \mathcal{G}\ \mathsf{i}(\{t'/x\}t\ \overrightarrow{t_j})$ and by rule $\mathcal{G}\mathsf{weak}$, $\{t'/x\}t\ \overrightarrow{t_j}\ \mathcal{G}\ [\mathsf{i}(\{t'/x\}t\ \overrightarrow{t_j}), \mathsf{i}(A), \overrightarrow{N}]$.

– $x \notin \mathsf{FV}(t)$:
It means that t' and A are β-normal forms and $\{t'/x\}t\ \overrightarrow{t_j} = t\ \overrightarrow{t_j}$. The last rule used to prove $u\ \mathcal{G}\ U$ must be $\mathcal{G}\beta^t{}_1$ or $\mathcal{G}\beta^t{}_2$ (possibly followed by several steps of $\mathcal{G}\mathsf{weak}$), so in both cases we have $U = [[\lambda x.[\mathsf{i}(t), x], U']\ T'\ \overrightarrow{\mathsf{i}(t_j)}, \overrightarrow{N}]$ with $A\ \mathcal{G}\ U'$ and $t'\ \mathcal{G}\ T'$ (using Lemma 102.2 in the former case where $U' = \mathsf{i}(A)$ and $T' = \mathsf{i}(t')$). By Lemma 102.1, U' and T' are β,π-normal forms. Now $U \longrightarrow_{\pi} [\lambda x.[\mathsf{i}(t), x]\ T'\ \overrightarrow{\mathsf{i}(t_j)}, U', \overrightarrow{N}] \longrightarrow_{\beta} [[\{T'/x\}\mathsf{i}(t), T']\ \overrightarrow{\mathsf{i}(t_j)}, U', \overrightarrow{N}]$. But by Lemma 101.1, $x \notin \mathsf{FV}(\mathsf{i}(t))$ so the above term is $[[\mathsf{i}(t), T']\ \overrightarrow{\mathsf{i}(t_j)}, U', \overrightarrow{N}]$. This term reduces by π to $[\mathsf{i}(t)\ \overrightarrow{\mathsf{i}(t_j)}, T', U', \overrightarrow{N}] = [\mathsf{i}(t\ \overrightarrow{t_j}), T', U', \overrightarrow{N}]$. By Lemma 102.2 and rule $\mathcal{G}\mathsf{weak}$, we get $t\ \overrightarrow{t_j}\ \mathcal{G}\ [\mathsf{i}(t\ \overrightarrow{t_j}), T', U', \overrightarrow{N}]$.

perp$\beta^t{}_2$) $(\lambda x^A.t)\ t'\ \overrightarrow{t_j} \rightsquigarrow (\lambda x^A.t)\ t''\ \overrightarrow{t_j}$ with $t' \rightsquigarrow t''$ and $x \notin \mathsf{FV}(t)$.

The last rule used to prove $u\ \mathcal{G}\ U$ must be $\mathcal{G}\beta^t{}_1$ or $\mathcal{G}\beta^t{}_2$ (possibly followed by several steps of $\mathcal{G}\mathsf{weak}$), so in both cases $U = [[\lambda x.[\mathsf{i}(t), x], U']\ T'\ \overrightarrow{\mathsf{i}(t_j)}, \overrightarrow{N}]$ with $A\ \mathcal{G}\ U'$ and $t'\ \mathcal{G}\ T'$ (using Lemma 102.2 in the former case where $U' = \mathsf{i}(A)$ and $T' = \mathsf{i}(t')$). By induction hypothesis, there is a term T'' such that $T' \longrightarrow^+_{\beta,\pi} T''$ and $t''\ \mathcal{G}\ T''$.

Hence, $U \longrightarrow^+_{\beta,\pi} [[\lambda x.[\mathsf{i}(t), x], U']\ T''\ \overrightarrow{\mathsf{i}(t_j)}, \overrightarrow{N}]$. By application of the rule $\mathcal{G}\beta^t{}_2$, $(\lambda x^A.t)\ t''\ \overrightarrow{t_j}\ \mathcal{G}\ [\lambda x.[\mathsf{i}(t), x], U']\ T''\ \overrightarrow{\mathsf{i}(t_j)}$, and we use rule $\mathcal{G}\mathsf{weak}$ to conclude.

perp$\beta^t{}_3$) $(\lambda x^A.t)\ t'\ \overrightarrow{t_j} \rightsquigarrow (\lambda x^{A'}.t)\ t'\ \overrightarrow{t_j}$ with $A \rightsquigarrow A'$ and $x \notin \mathsf{FV}(t)$.

The last rule used to prove $u\ \mathcal{G}\ U$ must be $\mathcal{G}\beta^t{}_1$ or $\mathcal{G}\beta^t{}_2$ (possibly followed by several steps of $\mathcal{G}\mathsf{weak}$), so in both cases $U = [[\lambda x.[\mathsf{i}(t), x], U']\ T'\ \overrightarrow{\mathsf{i}(t_j)}, \overrightarrow{N}]$ with $A\ \mathcal{G}\ U'$ and $t'\ \mathcal{G}\ T'$ (using Lemma 102.2 in the former case where $U' = \mathsf{i}(A)$ and $T' = \mathsf{i}(t')$). By induction hypothesis, there is a term U'' such that $U' \longrightarrow^+_{\beta,\pi} U''$ and $A'\ \mathcal{G}\ U''$.

Hence, $U \longrightarrow^+_{\beta,\pi} [[\lambda x.[\mathsf{i}(t), x], U'']\ T'\ \overrightarrow{\mathsf{i}(t_j)}, \overrightarrow{N}]$. By application of the rule $\mathcal{G}\beta^t{}_2$, $(\lambda x^{A'}.t)\ t'\ \overrightarrow{t_j}\ \mathcal{G}\ [\lambda x.[\mathsf{i}(t), x], U'']\ T'\ \overrightarrow{\mathsf{i}(t_j)}$, and we use rule $\mathcal{G}\mathsf{weak}$ to conclude.

perpλ) $\lambda x.t \rightsquigarrow \lambda x.t'$ with $t \rightsquigarrow t'$.

The last rule used to prove $u\ \mathcal{G}\ U$ must be $\mathcal{G}\lambda$, so $U = [\lambda x.T, \overrightarrow{N}]$ with $t\ \mathcal{G}\ T$. By induction hypothesis, there is a term T' such that $T \longrightarrow^+_{\beta,\pi} T'$ and $t'\ \mathcal{G}\ T'$. Hence, $U \longrightarrow^+_{\beta,\pi} [\lambda x.T', \overrightarrow{N}]$ (with $x \in \mathsf{FV}(T')$), and we obtain by application of rules $\mathcal{G}\lambda$ and $\mathcal{G}\mathsf{weak}$ that $\lambda x.t'\ \mathcal{G}\ [\lambda x.T', \overrightarrow{N}]$.

perpλ^t_1) $\lambda x^A.t \rightsquigarrow \lambda x^A.t'$ with $t \rightsquigarrow t'$.

The last rule used to prove $u\ \mathcal{G}\ U$ must be $\mathcal{G}\lambda^t$, so $U = [\lambda x.T, U', \overrightarrow{N}]$ with $A\ \mathcal{G}\ U'$ and $t\ \mathcal{G}\ T$. By induction hypothesis, there is a term T' such that $T \longrightarrow^+_{\beta,\pi} T'$ and $t'\ \mathcal{G}\ T'$. Hence, $U \longrightarrow^+_{\beta,\pi} [\lambda x.T', U', \overrightarrow{N}]$ (with $x \in \mathsf{FV}(T')$), and we obtain by application of rules $\mathcal{G}\lambda^t$ and $\mathcal{G}\mathsf{weak}$ that $\lambda x^A.t'\ \mathcal{G}\ [\lambda x.T', U', \overrightarrow{N}]$.

perpλ^t_2) $\lambda x^A.t \rightsquigarrow \lambda x^{A'}.t$ with $A \rightsquigarrow A'$.

The last rule used to prove $u\ \mathcal{G}\ U$ must be $\mathcal{G}\lambda^t$, so $U = [\lambda x.T, U', \overrightarrow{N}]$ with $A\ \mathcal{G}\ U'$ and $t\ \mathcal{G}\ T$. By induction hypothesis, there is a term U'' such that $U' \longrightarrow^+_{\beta,\pi} U''$ and $A'\ \mathcal{G}\ U''$. Hence, $U \longrightarrow^+_{\beta,\pi} [\lambda x.T, U'', \overrightarrow{N}]$ (with $x \in \mathsf{FV}(T')$), and we obtain by application of rules $\mathcal{G}\lambda^t$ and $\mathcal{G}\mathsf{weak}$ that $\lambda x^A.t'\ \mathcal{G}\ [\lambda x.T, U'', \overrightarrow{N}]$.

perp-var) $x\ \overrightarrow{t_j}\ t\ \overrightarrow{p_j} \rightsquigarrow x\ \overrightarrow{t_j}\ t'\ \overrightarrow{p_j}$ with $t \rightsquigarrow t'$.

The last rule used to prove $u\ \mathcal{G}\ U$ must be $\mathcal{G}\mathsf{var}$, so $U = [x\ \overrightarrow{Q_j}\ T\ \overrightarrow{U_j}, \overrightarrow{N}]$ with $t\ \mathcal{G}\ T$, $t_j\ \mathcal{G}\ Q_j$ and $p_j\ \mathcal{G}\ U_j$. By induction hypothesis, there is a term T' such that $T \longrightarrow^{+}_{\beta,\pi} T'$ and $t'\ \mathcal{G}\ T'$. As a consequence we get $U \longrightarrow^{+}_{\beta,\pi} [x\ \overrightarrow{Q_j}\ T'\ \overrightarrow{U_j}, \overrightarrow{N}]$ and by rules $\mathcal{G}\mathsf{var}$ and $\mathcal{G}\mathsf{weak}$ we obtain $x\ \overrightarrow{t_j}\ t'\ \overrightarrow{p_j}\ \mathcal{G}\ [x\ \overrightarrow{Q_j}\ T'\ \overrightarrow{U_j}, \overrightarrow{N}]$.

perpΠ_1) $\Pi x^A.B \rightsquigarrow \Pi x^{A'}.B$ with $A \rightsquigarrow A'$.

The last rule used to prove $u\ \mathcal{G}\ U$ must be $\mathcal{G}\Pi$, so $U = [\wp\ [\lambda x.T, V], \overrightarrow{N}]$ with $B\ \mathcal{G}\ T$ and $A\ \mathcal{G}\ V$. By induction hypothesis, there is a term V' such that $V \longrightarrow^{+}_{\beta,\pi} V'$ and $A'\ \mathcal{G}\ V'$.

As a consequence we get $U \longrightarrow^{+}_{\beta,\pi} [\wp\ [\lambda x.T, V'], \overrightarrow{N}]$ and by application of rules $\mathcal{G}\Pi$ and $\mathcal{G}\mathsf{weak}$ we obtain $\Pi x^{A'}.B\ \mathcal{G}\ [\wp\ [\lambda x.T, V'], \overrightarrow{N}]$.

perpΠ_2) $\Pi x^A.B \rightsquigarrow \Pi x^A.B'$ with $B \rightsquigarrow B'$.

The last rule used to prove $u\ \mathcal{G}\ U$ must be $\mathcal{G}\Pi$, so $U = [\wp\ [\lambda x.T, V], \overrightarrow{N}]$ with $B\ \mathcal{G}\ T$ and $A\ \mathcal{G}\ V$. By induction hypothesis, there is a term T' such that $T \longrightarrow^{+}_{\beta,\pi} T'$ and $B'\ \mathcal{G}\ T'$.

As a consequence we get $U \longrightarrow^{+}_{\beta,\pi} [\wp\ [\lambda x.T', V], \overrightarrow{N}]$ and by application of rules $\mathcal{G}\Pi$ and $\mathcal{G}\mathsf{weak}$ we obtain $\Pi x^A.B'\ \mathcal{G}\ [\wp\ [\lambda x.T', V], \overrightarrow{N}]$.

□

Corollary 106 (Reaching normal forms)
If $t \in \mathsf{WN}^{\rightsquigarrow}$ and $t\ \mathcal{G}\ T$ then $T \in \mathsf{WN}^{\beta,\pi}$.

Proof: By induction in $\mathsf{WN}^{\rightsquigarrow}$, the induction hypothesis is:
$t \in \mathsf{nf}^{\rightsquigarrow} \vee (\exists u \in \rightsquigarrow(t), \forall U, u\ \mathcal{G}\ U \Rightarrow U \in \mathsf{WN}^{\beta,\pi})$.

If $t \in \mathsf{nf}^{\rightsquigarrow}$, then Lemma 102.1 gives $T \in \mathsf{nf}^{\beta,\pi} \subseteq \mathsf{WN}^{\beta,\pi}$.

If $\exists u \in \rightsquigarrow(t), \forall U, u\ \mathcal{G}\ U \Rightarrow U \in \mathsf{WN}^{\beta,\pi}$, then by Theorem 105 we get a specific T' such that $u\ \mathcal{G}\ T'$ and $T \longrightarrow^{+}_{\beta,\pi} T'$. We can apply the induction hypothesis by taking $U = T'$ and get $T' \in \mathsf{WN}^{\beta,\pi}$. But because $\mathsf{WN}^{\beta,\pi}$ is patriarchal, $T \in \mathsf{WN}^{\beta,\pi}$ as required. □

Corollary 107 (SN in $\lambda^?$ implies WN in λI) *$i(\mathsf{SN}^{\beta^t,\beta}) \subseteq \mathsf{WN}^{\beta,\pi}$*

Proof: Notice that $\mathsf{SN}^{\beta^t,\beta} \subseteq \mathsf{SN}^{\rightsquigarrow} \subseteq \mathsf{WN}^{\rightsquigarrow}$. Then Lemma 102.2 gives $\forall t \in \mathsf{SN}^{\beta^t,\beta}, t\ \mathcal{G}\ \mathsf{i}(t)$, and thus, by Theorem 105, $\mathsf{i}(t) \in \mathsf{WN}^{\beta,\pi}$. □

We then use the following theorem about λI:

Theorem 108 (Nederpelt [Ned73]) $\mathit{WN}^{\beta,\pi} \subseteq \mathit{SN}^{\beta,\pi}$

From this we conclude that the property of being strongly normalising is preserved by i:

Corollary 109 (Preservation of strong normalisation)
For any $\lambda^?$-term t, if $t \in \mathit{SN}^{\beta^t,\beta}$, then $\mathsf{i}(t) \in \mathit{SN}^{\beta,\pi}$.

Proof: By Corollary 107 and Theorem 108. If $t \in \mathsf{SN}^{\beta}$, then every strategy terminates for t. We have in particular that $\rightsquigarrow$ terminates for t so that $\mathsf{i}(t) \in \mathsf{WN}^{\beta\pi}$ by Corollary 106 and hence $\mathsf{i}(t) \in \mathsf{SN}^{\beta,\pi}$ by Theorem 108. □

Conclusion

In this chapter we introduced two new extensions of the simulation technique. The first one, called the Safeness & Minimality technique, can be applied to any HOC. The second one concerns more specifically systems that can be related to λ-calculus, and uses the λI-calculus of [Klo80].

The first technique has been illustrated by the example of the calculus $\lambda\mathsf{x}$ [BR95], and the second will be illustrated by the example of $\lambda\mathsf{lxr}$ [KL05, KL06] in the next chapter.

Further work includes checking that Nederpelt's result that weak normalisation in λI implies strong normalisation (Theorem 108) can be proved constructively, so that the whole technique of simulation in λI is constructive.

Also, the examples for the Safeness & Minimality technique rely on a few external results such as the termination of LPO [KL80], which has been proved in a framework with traditional definitions of normalisation. The latter are classically equivalent to ours, so that we can classically use them.

However, although the Safeness & Minimality technique is classical, it would be interesting to prove the LPO technique in our constructive framework, which is left as future work. This technique seems close to the notion of *dependency pairs* (see e.g. [AG00]). Formal connections with it should be studied and is left as further work.

Chapter 5

Weakening, contraction & cut in λ-calculus

As we have seen in Chapter 2, a typical example of the Curry-Howard correspondence is obtained between the simply-typed λ-calculus [Chu41] and the logical system NJi. Both formalisms can be decomposed in the following sense:

On the one hand, β-reduction in λ-calculus can be decomposed into more elementary operations by implementing the implicit substitution as the interaction between (and the propagation of) erasure, duplication and substitution constructors.

On the other hand, the additive rules of NJi can be decomposed into multiplicative rules and the structural rules of weakening and contraction, just like G3ii can be decomposed into G1ii in the same manner (as described in Chapter 2).

In this chapter, we show the connection between these two elementary decompositions by introducing a calculus called λlxr with erasure, duplication and substitution constructors, which can be seen as a λ-calculus with *explicit substitutions*. Its simply-typed version corresponds, via the Curry-Howard paradigm, to a multiplicative version of NJi, with cuts.

Note that λlxr and most of the theory presented hereafter already appeared in [KL05, KL06].

Explicit substitutions & new constructors

The λlxr-calculus is the product of a line of research tackling the problem described in Chapter 4 of designing an HOC, with explicit substitutions and a notion of composition as strong as possible, but satisfying the PSN property.

In order to tackle this problem, [DG01] defined a calculus with *labels*, called λ_{ws}, which allows a *controlled* composition of explicit substitutions without losing PSN. The typing rule for these labels is precisely a *weakening* rule such as that of G1ii (in Chapter 2). But the λ_{ws}-calculus has a complicated syntax and its

named version [DCKP00] is even less readable. On the positive side, we should mention that λ_{ws}-calculus has nice properties as it is confluent and satisfies PSN.

Also, [DCKP03] establishes a translation from the simply-typed λ_{ws} to *proof-nets*, a graph formalism originally introduced for proofs in [Gir87]. However, a clear fragment of linear logic's proof-nets accounts for intuitionistic logic, and is the part of the formalism concerned by the aforementioned translation. Moreover, the translation reveals a natural semantics for composition of explicit substitutions, and also suggests that erasure and duplication constructors can be naturally added to the calculus, respectively corresponding to *weakening* and *contraction*, while the substitution constructor corresponds to *cut*.

This is the essence of λlxr in its typed version. The connection between λlxr and proof-nets is formalised in [KL05, KL06], by means of a translation, from the former to the latter, that is not only *sound* but also *complete* (in contrast to the translation from λ_{ws} to proof-nets, which is only sound). This is achieved by equipping λlxr with a congruence on terms that corresponds, via a translation, to equalities between proof-nets, when these are considered *modulo* two equations that allow the simulation of β-reduction in λ-calculus by cut-elimination in proof-nets [DCG99]. To this notion of reduction of *proof-nets modulo* corresponds the reduction of λlxr-terms modulo the aforementioned congruence.

In this chapter we do not tackle the connection with proof-nets from which λlxr originates, but concentrate on some intrinsic properties of the calculus, of which most features make sense even in an untyped framework (such as the aforementioned congruence), as well as its connection with λ-calculus. Namely, we establish a reflection in λlxr of λ-calculus, from which we get confluence, and we prove the PSN property as well as the strong normalisation of typed terms.

Composition

From a rewriting point of view this calculus is the first HOC that is confluent and strongly normalising on typed terms, strongly simulates β-reduction, satisfies PSN as well as having *full composition*. By full composition is meant that we can compute the application of a substitution constructor to a term, no matter which substitution remains non-evaluated within that term. In particular, in a term $\langle v/x\rangle\langle u/y\rangle t$, the external substitution is not blocked by the internal one and can be further evaluated without ever requiring any preliminary evaluation of $\langle u/y\rangle t$. In other words, the application of the substitution $\langle v/x\rangle$ to the term t can be evaluated independently from that of $\langle u/y\rangle$. A more technical explanation of the concept of full composition appears in section 5.1.

Weakening and Garbage Collection

The *erasure/weakening constructor* has an interesting computational behaviour in calculi such as λ_{ws} and λlxr that we illustrate via an example. Let us denote

by $\mathcal{W}\_(\_)$ the weakening constructor, so that a λlxr-term whose variable x is used to weaken the term t is written $\mathcal{W}_x(t)$, that is, we explicitly annotate that the variable x does not appear free in the term t. Then, when evaluating the application of a term $\lambda x.\mathcal{W}_x(t)$ to another term u, a substitution constructor $\langle u/x\rangle$ is created and the computation will continue with $\langle u/x\rangle\mathcal{W}_x(t)$. Then, the weakening constructor will be used to prevent the substitution $\langle u/x\rangle$ from going into the term t, thus making more efficient the propagation of a substitution with respect to the original term.

Another interesting feature of our system is that weakening constructors are always *pulled out* to the top-level during λlxr-reduction. Moreover, free variables are *never* lost during computation because they get marked as weakening constructors. Indeed, if t β-reduces to t', then its λlxr-interpretation reduces to that of t' where weakening constructors are added at the top level to keep track of the variables that are lost during the β-reduction step. Thus for example, when simulating the β-reduction steps $(\lambda x.\lambda y.x)\ u\ z \longrightarrow_\beta^* u$, the lost variable z will appear in the result of the computation by means of a weakening constructor at the top level, i.e. as $\mathcal{W}_z(\overline{u})$ (where $\overline{u}$ is the interpretation of u in λlxr), thus preparing the situation for an efficient garbage collection on z.

The weakening constructor can thus be seen as a tool for handling *garbage collection*. For instance, it is worth noticing that the labels of the λ_{ws}-calculus cannot be pulled out to the top-level as in λlxr. Also, free variables may be lost during λ_{ws}-computation. Thus, garbage collection within λ_{ws} does not offer the advantages existing in λlxr.

Related work

The literature is rich in correspondences between logical systems with cuts and typed HOC with explicit substitutions.

For intuitionistic logic, we can mention for instance [VW01], and the next chapters tackle G3ii and λG3, with some fragments such as LJT [Her94], as well as some variants such as G4ii (in Chapter 7). In a very different spirit, [CK99] relates the *pattern matching constructor* in functional programming to cut-elimination in sequent calculus for intuitionistic logic.

For linear logic, Abramsky [Abr93] gives computational interpretations which are based on sequents rather than proof-nets (i.e. no equations between terms reflect the irrelevance of some syntactic details appearing in sequent proofs). Many other term calculi based on sequents rather than proof-nets have been proposed for linear logic, as for example [GdR00, BBdH93, RR97, Wad93, OH06].

An axiomatisation of (acyclic and cyclic) sharing graphs by means of higher-order term syntax is proposed by Hasegawa [Has99] who investigates categorical models of the term calculi thus obtained. While we exploit the relation between particular graphs (proof-nets) and λ-calculus, he mainly focuses on Ariola and Klop's cyclic lambda calculi and Milner's action calculi.

A related approach was independently developed by V. van Oostrom (available in course notes written in Dutch [vO01]), where constructors for contraction and weakening are added to the λ-calculus to define a fine control of duplication and erasing. We show here how the same constructors allow a fine control of composition when using substitution constructors, although the proofs of some fundamental properties, such as PSN and confluence, become harder. An overview on optimal sharing in functional programming languages, and its connection with linear logic can be found in [AG98].

Finally, a revised version of the calculus λ_{ws} with names has been developed simultaneously and independently in [Pol04b], satisfying similar properties such as full composition and PSN, but with a more complicated substitution constructor due to the absence of duplication constructors.

Structure of the chapter

Section 5.1 presents the syntax and operational semantics of the λlxr-calculus. Section 5.2 shows the relation between λ-calculus and λlxr-calculus by establishing the reflection. In section 5.3 we establish PSN and strong normalisation of typed terms.

5.1 The calculus λlxr

5.1.1 The linear syntax

We present in this section the syntax of the untyped λlxr-calculus as well as the notions of congruence and reduction between terms.

The syntax for raw terms, given by the following grammar, is very simple[1] and can be just viewed as an extension of that of λx [BR95], presented in Chapter 4. We recall that the denumerable set of *variables*, denoted $x, y, z, \ldots$, is equipped with a *total order* (Definition 22).

$$t ::= x \mid \lambda x.t \mid t\ t \mid \langle t/x\rangle t \mid \mathcal{W}_x(t) \mid \mathcal{C}_x^{y\,|\,z}(t)$$

The term $\lambda x.t$ is called an *abstraction*, $t\ u$ an *application*, and $\langle u/x\rangle t$ an *explicit substitution*. The term constructors $\mathcal{W}\_(\_)$, $\mathcal{C}\_^{\,|\,}\_(\_)$ and $\langle \_/\_\rangle\_$ are respectively called *weakening*, *contraction* and *substitution constructors*. The constructs $\lambda x.t$ and $\langle u/x\rangle t$ bind x in t. The construct $\mathcal{C}_x^{y\,|\,z}(t)$ binds x and y in t.

We now consider linear terms in the syntax of λlxr, in the sense of Definition 69. For instance, the terms $\mathcal{W}_x(x)$ and $\lambda x.xx$ are not linear. However, the latter can be represented in the λlxr-calculus by the linear term $\lambda x.\mathcal{C}_x^{y\,|\,z}(y\ z)$.

[1] in contrast to λ_{ws} with names [DCKP00, DCKP03], where terms affected by substitutions have a complex format $t[x, u, \Gamma, \Delta]$

More generally, every λ-term can be represented by a linear $\lambda\mathsf{lxr}$-term (cf. Section 5.2). Note that being linear is a property of α-equivalent classes, i.e. given two α-equivalent terms, either both are linear or both are not.

5.1.2 Congruence & notations

As mentioned in the introduction of this chapter, $\lambda\mathsf{lxr}$ inherits a congruence on terms from the connection with proof-nets modulo. Not only is the congruence an essential part of this connection, as described in [KL05, KL06], but the notion of reduction in $\lambda\mathsf{lxr}$, presented in the next section, hardly makes sense without considering it modulo the congruence, in particular for simulating β-reduction. The equations defining the congruence of $\lambda\mathsf{lxr}$ are presented in Fig. 5.1.

$\mathcal{C}_w^{x\,\vert\,v}(\mathcal{C}_x^{z\,\vert\,y}(t))$	$\equiv_{\mathsf{A_c}}$	$\mathcal{C}_w^{z\,\vert\,x}(\mathcal{C}_x^{y\,\vert\,v}(t))$	
$\mathcal{C}_x^{y\,\vert\,z}(t)$	$\equiv_{\mathsf{C_c}}$	$\mathcal{C}_x^{z\,\vert\,y}(t)$	
$\mathcal{C}_{x'}^{y'\,\vert\,z'}(\mathcal{C}_x^{y\,\vert\,z}(t))$	$\equiv_{\mathsf{P_c}}$	$\mathcal{C}_x^{y\,\vert\,z}(\mathcal{C}_{x'}^{y'\,\vert\,z'}(t))$	if $x \neq y', z'$ & $x' \neq y, z$
$\mathcal{W}_x(\mathcal{W}_y(t))$	$\equiv_{\mathsf{P_w}}$	$\mathcal{W}_y(\mathcal{W}_x(t))$	
$\langle v/y\rangle\langle u/x\rangle t$	$\equiv_{\mathsf{P_s}}$	$\langle u/x\rangle\langle v/y\rangle t$	if $y \notin \mathsf{FV}(u)$ & $x \notin \mathsf{FV}(v)$
$\langle u/x\rangle\mathcal{C}_w^{y\,\vert\,z}(t)$	$\equiv_{\mathsf{P_{cs}}}$	$\mathcal{C}_w^{y\,\vert\,z}(\langle u/x\rangle t)$	if $x \neq w$ & $y, z \notin \mathsf{FV}(u)$

Figure 5.1: Congruence equations for $\lambda\mathsf{lxr}$-terms

The equations $\mathsf{A_c}$ and $\mathsf{C_c}$ express the internal associativity and commutativity of contraction, when seen as a binary operation merging two "wires" labelled with its two bound variables into one labelled by its free variable. The equations $\mathsf{P_c}, \mathsf{P_w}, \mathsf{P_s}$ express the permutability of independent contractions, weakenings, and substitutions, respectively. The point of the equation $\mathsf{P_{cs}}$, expressing the permutability between independent contraction and substitution, is discussed in the next sub-section.

We define the relation $\equiv$ as the smallest congruence on terms that contains the equations of Fig. 5.1. It can easily be proved that $\equiv$ preserves free variables and linearity. Since we shall deal with rewriting modulo the congruence $\equiv$, it is worth noticing that $\equiv$ is decidable. More than that, each congruence class contains finitely many terms. Indeed, two congruent terms have clearly the same size, so it is easy to see by induction on this size that the congruence rules generate finitely many possibilities to pick up a representative of the class.

We use Φ, Υ, Σ, Ψ, Ξ, $\Omega, \ldots$ to denote finite *lists* of variables (with no repetition). The notation Φ, Ψ denote the concatenation of Φ and Ψ, and we always suppose in that case that no variable appears in both Φ and Ψ.

Since variables form a syntactic category of their own, we have the standard notion of substitution for that category (see Definition 43), written $\left\{{}^{\Psi}\!/_{\Phi}\right\}t$ for two lists of variables Φ and Ψ of the same length. Thus for instance $\left\{{}^{x',y'}\!/_{x,y}\right\}\mathcal{C}_w^{y\,\vert\,z}(x\ (y\ z)) = \mathcal{C}_w^{y\,\vert\,z}(x'\ (y\ z))$ (y is not replaced since the occur-

rence is bound). We remark that for any permutation π of $1 \dots n$, we have $\left\{{}^{\Psi}\!/_{\Phi}\right\} t = \left\{{}^{\pi(\Psi)}\!/_{\pi(\Phi)}\right\} t$.

We use the notation $\mathcal{W}_{x_1,\dots,x_n}(t)$ for $\mathcal{W}_{x_1}(\dots \mathcal{W}_{x_n}(t))$ and $\mathcal{C}^{y_1,\dots,y_n \,|\, z_1,\dots,z_n}_{x_1,\dots,x_n}(t)$ for $\mathcal{C}^{y_1 \,|\, z_1}_{x_1}(\dots \mathcal{C}^{y_n \,|\, z_n}_{x_n}(t))$, where $x_1, \dots, x_n, y_1, \dots, y_n, z_1, \dots, z_n$ are all distinct variables. In the case of the empty list, we define $\mathcal{W}_{\emptyset}(t) = t$ and $\mathcal{C}^{\emptyset \,|\, \emptyset}_{\emptyset}(t) = t$.

As in the case of substitution, for any permutation π of $1 \dots n$, we have $\mathcal{W}_{\Psi}(t) \equiv \mathcal{W}_{\pi(\Psi)}(t)$ and $\mathcal{C}^{\Psi \,|\, \Upsilon}_{\Phi}(t) \equiv \mathcal{C}^{\pi(\Psi) \,|\, \pi(\Upsilon)}_{\pi(\Phi)}(t)$. Moreover, we have $\mathcal{C}^{\Psi \,|\, \Upsilon}_{\Phi}(t) \equiv \mathcal{C}^{\Upsilon \,|\, \Psi}_{\Phi}(t)$ and $\mathcal{C}^{\Psi \,|\, \Upsilon}_{\Phi}(\mathcal{C}^{\Sigma \,|\, \Psi}_{\Psi}(t)) \equiv \mathcal{C}^{\Sigma \,|\, \Psi}_{\Phi}(\mathcal{C}^{\Psi \,|\, \Upsilon}_{\Psi}(t))$.

Notice 2 *Sometimes we use a set of variables, e.g. $\mathcal{S}$, in places where lists are expected, as in $\mathcal{W}_{\mathcal{S}}(u)$, $\mathcal{C}^{\Phi \,|\, \Psi}_{\mathcal{S}}(t)$, $\left\{{}^{\Phi}\!/_{\mathcal{S}}\right\} t$ or $\Phi := \mathcal{S}$. The intended list is obtained by ordering $\mathcal{S}$ according to the total order that we have on the set of variables. These notations introduce no ambiguity and are much more legible.*

5.1.3 Reduction

Rules & relation

The reduction relation of the calculus is the relation generated by the reduction rules in Fig. 5.2 modulo the congruence relation in Fig. 5.1.

We will use xr to denote the set of rules $\mathsf{x} \cup \mathsf{r}$ and $B\mathsf{xr}$ to denote the set $\{B\} \cup \mathsf{xr}$. Hence, the most general reduction relation of our calculus is $\longrightarrow_{B\mathsf{xr}}$ (i.e. generated by the rules of $B\mathsf{xr}$), often written $\longrightarrow_{\lambda\mathsf{lxr}}$ (i.e. pertaining to the calculus $\lambda\mathsf{lxr}$).

General properties

The rules should be understood in the prospect of applying them to linear terms. Indeed, linearity is preserved by the reduction relation, which satisfies the following properties:

Lemma 110 (Preservation Properties)
Let t be a linear term and $t \longrightarrow_{\lambda\mathsf{lxr}} t'$.

1. *The set of free variables is preserved, i.e. $\mathsf{FV}(t) = \mathsf{FV}(t')$.*
2. *Linearity is preserved, i.e. t' is linear.*

Proof: By using the fact that the congruence preserves free variables and linearity, the two properties have to be satisfied by the basic reduction relation. This can be checked by a straightforward simultaneous induction on the reduction step and case analysis. □

B	$(\lambda x.t)\ u$	$\longrightarrow$	$\langle u/x\rangle t$	
System x				
Abs	$\langle u/x\rangle(\lambda y.t)$	$\longrightarrow$	$\lambda y.\langle u/x\rangle t$	
App1	$\langle u/x\rangle(t\ v)$	$\longrightarrow$	$\langle u/x\rangle t\ v$	$x \notin \mathsf{FV}(v)$
App2	$\langle u/x\rangle(t\ v)$	$\longrightarrow$	$t\ \langle u/x\rangle v$	$x \notin \mathsf{FV}(t)$
Var	$\langle u/x\rangle x$	$\longrightarrow$	u	
Weak1	$\langle u/x\rangle \mathcal{W}_x(t)$	$\longrightarrow$	$\mathcal{W}_{\mathsf{FV}(u)}(t)$	
Weak2	$\langle u/x\rangle \mathcal{W}_y(t)$	$\longrightarrow$	$\mathcal{W}_y(\langle u/x\rangle t)$	$x \neq y$
Cont	$\langle u/x\rangle \mathcal{C}_x^{y\,\vert\,z}(t)$	$\longrightarrow$	$\mathcal{C}_{\mathsf{FV}(u)}^{\Upsilon\,\vert\,\Psi}(\langle \{{}^{\Upsilon}\!/_{\mathsf{FV}(u)}\}\, u/z\rangle \langle \{{}^{\Psi}\!/_{\mathsf{FV}(u)}\}\, u/y\rangle t)$	
Comp	$\langle u/x\rangle\langle v/y\rangle t$	$\longrightarrow$	$\langle\langle u/x\rangle v/y\rangle t$	$x \notin \mathsf{FV}(t) \setminus \{y\}$
System r				
WAbs	$\lambda x.\mathcal{W}_y(t)$	$\longrightarrow$	$\mathcal{W}_y(\lambda x.t)$	$x \neq y$
WApp1	$\mathcal{W}_y(u)\ v$	$\longrightarrow$	$\mathcal{W}_y(u\ v)$	
WApp2	$u\ \mathcal{W}_y(v)$	$\longrightarrow$	$\mathcal{W}_y(u\ v)$	
WSubs	$\langle \mathcal{W}_y(u)/x\rangle t$	$\longrightarrow$	$\mathcal{W}_y(\langle u/x\rangle t)$	
Merge	$\mathcal{C}_w^{y\,\vert\,z}(\mathcal{W}_y(t))$	$\longrightarrow$	$\{{}^{w}\!/_{z}\}\, t$	
Cross	$\mathcal{C}_w^{y\,\vert\,z}(\mathcal{W}_x(t))$	$\longrightarrow$	$\mathcal{W}_x(\mathcal{C}_w^{y\,\vert\,z}(t))$	$x \neq y,\ \ x \neq z$
CAbs	$\mathcal{C}_w^{y\,\vert\,z}(\lambda x.t)$	$\longrightarrow$	$\lambda x.\mathcal{C}_w^{y\,\vert\,z}(t)$	
CApp1	$\mathcal{C}_w^{y\,\vert\,z}(t\ u)$	$\longrightarrow$	$\mathcal{C}_w^{y\,\vert\,z}(t)\ u$	$y, z \notin \mathsf{FV}(u)$
CApp2	$\mathcal{C}_w^{y\,\vert\,z}(t\ u)$	$\longrightarrow$	$t\ \mathcal{C}_w^{y\,\vert\,z}(u)$	$y, z \notin \mathsf{FV}(t)$
CSubs	$\mathcal{C}_w^{y\,\vert\,z}(\langle u/x\rangle t)$	$\longrightarrow$	$\langle \mathcal{C}_w^{y\,\vert\,z}(u)/x\rangle t$	$y, z \notin \mathsf{FV}(t) \setminus \{x\}$

Figure 5.2: Reduction rules for λlxr-terms

For instance in rule Cont, it is the introduction of the lists of fresh variables Ψ and Υ that ensures the linearity of terms.

In contrast to λ-calculus where the set of free variables may decrease during reduction, preservation of free variables (Lemma 110.1) holds in λlxr thanks to the weakening constructor. This is similar to the property called "interface preserving" [Laf90] in interaction nets. It is also worth noticing that the set of bound variables of a term may either increase (cf. rule Cont) or decrease (cf. rules Var, Merge, Weak1, ...).

The fact that linearity is preserved by congruence and reduction (Lemma 110.2) is a minimal requirement of the system.

Notice 3 *From now on we only consider linear terms.*

Role of the rules

The B-rule is a key rule of $\lambda\mathsf{lxr}$ in that it reduces what is considered in the λ-calculus as a β-redex, and creates a substitution constructor, as in $\lambda\mathsf{x}$ [BR95] but respecting the linearity constraints.

System x propagates and eliminates substitution constructors, and duplication and erasure are controlled by the presence of contraction and weakening (rules Cont and $\mathsf{Weak1}$ respectively). Contraction and weakening can thus be seen as *resource constructors* also called respectively *duplication* and *erasure* constructors. Note that this only makes sense if the linearity constraints are satisfied; in this case a construct such as $\langle t/x\rangle y$ is forbidden.

Lemma 111 *t is a x-normal form if and only if t has no explicit substitution.*

Proof: We first remark that if t has no explicit substitution, then clearly no x-rule can be applied. Conversely, for each substitution constructor applied term that is not an explicit substitution there is a reduction rule. □

Thus for instance, the term $\langle\lambda z.z/y\rangle(\lambda x.x\ y)$ reduces to $\lambda x.(x\ \lambda z.z)$. Reducing terms by system x implements a notion of implicit substitution (on $\lambda\mathsf{lxr}$-terms), but a major difference with $\lambda\mathsf{x}$ is that the notion of substitution thus implemented applies here to all terms, not only to those that have no explicit substitutions. For example, $\langle\lambda z.z/y\rangle\langle y\ \lambda z.z/x'\rangle x$ does not reduces to $\langle(\lambda z.z)\ \lambda z.z/x'\rangle x$ in $\lambda\mathsf{x}$ but it does in $\lambda\mathsf{lxr}$ thanks to the notion of composition.

Note that when linearity constraints are not considered, four cases may occur when composing two substitutions as in $\langle u/x\rangle\langle v/y\rangle t$: either (1) $x \in \mathsf{FV}(t)\cap\mathsf{FV}(v)$, or (2) $x \in \mathsf{FV}(t) \setminus \mathsf{FV}(v)$, or (3) $x \in \mathsf{FV}(v) \setminus \mathsf{FV}(t)$, or (4) $x \notin \mathsf{FV}(t) \cup \mathsf{FV}(v)$.

Composition is said to be *partial* in calculi like λ_{ws} [DG01] because only cases (1) and (3) are considered by the reduction rules. Because of the linearity constraints of $\lambda\mathsf{lxr}$, cases (1) and (4) have to be dealt with by the introduction of a contraction for case (1) and a weakening for case (4). Those constructors will interact with external substitutions by the use of rules ($\mathsf{Weak1}$) and (Cont), respectively. Case (3) is treated by rule (Comp), and case (2) by the congruence rule $\mathsf{P_s}$. More precisely, the congruence rule can be applied to swap the substitutions, thus allowing the evaluation of the external substitution $\langle u/x\rangle$ without forcing the internal one to be evaluated first. We say in this case that composition is *full* as all cases (1)-(4) are treated.

The linearity constraints are *essential* for composition: if they are not taken into account, the composition rule Comp causes failure of the PSN and strong

normalisation properties [BG99]. Hence, it is because of the presence of weakenings and contractions, combined with the linearity constraints, that the notion of composition in λlxr is full. Thus, λlxr turns out to be the first term calculus with substitution constructors having full composition and preserving β-strong normalisation (Corollary 137).

Respectively viewed as duplication and erasure constructors, contraction and weakening play a very special role with respect to optimisation issues. In a term, the further down a contraction $\mathcal{C}_x^{y\,|\,z}(\_)$ lies, the later a substitution constructor on x will be duplicated in its propagation process by system x. Symmetrically, the further up a weakening $\mathcal{W}_x(\_)$ lies, the sooner a substitution on x, called a *void* substitution, will be erased. For instance, if $y, z \in \mathsf{FV}(t_2)$, we have $\langle\lambda x'.x'/x\rangle\mathcal{C}_x^{y\,|\,z}(t_1\ t_2\ t_3) \longrightarrow_{\mathsf{x}}^{5} t_1\ \langle\lambda x'.x'/z\rangle\langle\lambda x'.x'/y\rangle t_2\ t_3$ but $\langle\lambda x'.x'/x\rangle(t_1\ \mathcal{C}_x^{y\,|\,z}(t_2)\ t_3) \longrightarrow_{\mathsf{x}}^{3} t_1\ \langle\lambda x'.x'/z\rangle\langle\lambda x'.x'/y\rangle t_2\ t_3$, so $t_1\ \mathcal{C}_x^{y\,|\,z}(t_2)\ t_3$ is in a sense more optimised than $\mathcal{C}_x^{y\,|\,z}(t_1\ \ t_2\ \ t_3)$. Symmetrically, we have $\langle\lambda x'.x'/x\rangle(t_1\ \mathcal{W}_x(t_2)\ t_3) \longrightarrow_{\mathsf{x}}^{3} t_1\ t_2\ t_3$ but $\langle\lambda x'.x'/x\rangle\mathcal{W}_x(t_1\ t_2\ t_3) \longrightarrow_{\mathsf{x}}^{1} t_1\ \ t_2\ \ t_3$, so $\mathcal{W}_x(t_1\ t_2\ t_3)$ is in a sense more optimised than $t_1\ \mathcal{W}_x(t_2)\ t_3$.

System r optimises terms by pushing down contractions and pulling up weakenings, so that they reach *canonical places* in λlxr-terms (also using Weak2 and the left to right direction of the equation $\mathsf{P_{cs}}$). Such a place for a contraction $\mathcal{C}_x^{y\,|\,z}(\_)$ is just above an application or an explicit substitution, with y and z in distinct sides (i.e. $\mathcal{C}_x^{y\,|\,z}(t\ u)$ or $\mathcal{C}_x^{y\,|\,z}(\langle u/x'\rangle t)$ with $y \in \mathsf{FV}(t)$ and $z \in \mathsf{FV}(u)$ or *vice versa*). The canonical place for a weakening $\mathcal{W}_x(\_)$ is either at the top-level of a term or just below a binder on x (i.e. $\lambda x.\mathcal{W}_x(t)$ or $\langle u/x\rangle\mathcal{W}_x(t)$).

For terms without explicit substitutions, these constructs are just $\mathcal{C}_x^{y\,|\,z}(t\ u)$ (with $y \in \mathsf{FV}(t)$ and $z \in \mathsf{FV}(u)$ or *vice versa*) and either $\lambda x.\mathcal{W}_x(t)$ or $\mathcal{W}_x(t)$ at the top-level. In that case, the rules CSubs and WSubs and the right to left direction of the equation $\mathsf{P_{cs}}$ are not needed to place contractions and weakenings in canonical places. Removing these rules and orienting $\mathsf{P_{cs}}$ from left to right as a rule of system x would yield a system for which most of the results of this chapter would hold (but not optimising as much terms with explicit substitutions); in particular, x would still eliminate substitution constructors and implement the same notion of implicit substitution and β-reduction could still be simulated (cf. Theorem 121).

5.1.4 Termination of xr

It is clear that rule B will be used to simulate β-reduction. The rules of system xr handle the constructors that we have introduced, and a minimal requirement for those rules is to induce a terminating system. We shall also see in Section 5.3 that xr is confluent.

The use of resource constructors allows us to derive information about the number of times that a substitution can be duplicated along a sequence of xr-

reductions. Indeed, this will happen when a substitution meets a contraction that concerns the substituted variable. This idea inspires the notion of *multiplicity* of the substituted variable:

Definition 92 (Multiplicity) Given a free variable x in a (linear) term t, the *multiplicity of x in t*, written $\mathcal{M}_x(t)$, is defined by induction on terms as presented in Fig. 5.3.
Supposing that $x \neq y, x \neq z, x \neq w$,

$\mathcal{M}_x(x) := 1$		$\mathcal{M}_x(\langle u/y\rangle t) :=$	$\mathcal{M}_x(t)$ if $x \in \mathsf{FV}(t) \setminus \{y\}$
$\mathcal{M}_x(\lambda y.t) := \mathcal{M}_x(t)$		$\mathcal{M}_x(\langle u/y\rangle t) :=$	$\mathcal{M}_y(t) \cdot (\mathcal{M}_x(u) + 1)$ if $x \in \mathsf{FV}(u)$
$\mathcal{M}_x(\mathcal{W}_x(t)) := 1$		$\mathcal{M}_x((t\ u)) :=$	$\mathcal{M}_x(t)$ if $x \in \mathsf{FV}(t)$
$\mathcal{M}_x(\mathcal{W}_y(t)) := \mathcal{M}_x(t)$		$\mathcal{M}_x((t\ u)) :=$	$\mathcal{M}_x(u)$ if $x \in \mathsf{FV}(u)$
		$\mathcal{M}_x(\mathcal{C}_x^{z\,\vert\,w}(t)) :=$	$\mathcal{M}_z(t) + \mathcal{M}_w(t) + 1$
		$\mathcal{M}_x(\mathcal{C}_y^{z\,\vert\,w}(t)) :=$	$\mathcal{M}_x(t)$

Figure 5.3: Multiplicity

Roughly, this notion corresponds to the number of occurrences of a variable in a λlxr-term when translated to its corresponding λ-term free from linearity constraints and resource constructors (see Section 5.2 for details), but we add a twist to this concept (+1 in the second case for explicit substitution and the first case for contraction in the definition above), so that the following notion of *term complexity*, which weighs the complexity of a sub-term in a substitution with the multiplicity of the substituted variable, is decreased by reductions (Lemma 2).

Definition 93 (Term complexity) We define the notion of *term complexity* by induction on terms as presented in Fig. 5.4.

$\mathcal{T}\mathsf{x}(x) := 1$		$\mathcal{T}\mathsf{x}(\langle u/x\rangle t) :=$	$\mathcal{T}\mathsf{x}(t) + \mathcal{M}_x(t) \cdot \mathcal{T}\mathsf{x}(u)$
$\mathcal{T}\mathsf{x}(\lambda x.t) := \mathcal{T}\mathsf{x}(t)$		$\mathcal{T}\mathsf{x}(t\ u) :=$	$\mathcal{T}\mathsf{x}(t) + \mathcal{T}\mathsf{x}(u)$
$\mathcal{T}\mathsf{x}(\mathcal{W}_x(t)) := \mathcal{T}\mathsf{x}(t)$		$\mathcal{T}\mathsf{x}(\mathcal{C}_x^{y\,\vert\,z}(t)) :=$	$\mathcal{T}\mathsf{x}(t)$

Figure 5.4: Term complexity

Lemma 112 *The notions of multiplicity and term complexity are invariant under conversion by $\equiv$.*

Proof: Indeed, as two non-trivial cases, let us consider the case $\mathcal{C}_w^{x\,\vert\,v}(\mathcal{C}_x^{y\,\vert\,z}(t)) \equiv \mathcal{C}_w^{x\,\vert\,y}(\mathcal{C}_x^{z\,\vert\,v}(t))$ for which we have:

$$\mathcal{M}_w(\mathcal{C}_w^{x\,\vert\,v}(\mathcal{C}_x^{y\,\vert\,z}(t))) = \mathcal{M}_y(t) + \mathcal{M}_z(t) + \mathcal{M}_v(t) + 2 = \mathcal{M}_w(\mathcal{C}_w^{x\,\vert\,y}(\mathcal{C}_x^{z\,\vert\,v}(t)))$$

and let us consider the case $\langle v/y\rangle\langle u/x\rangle t \equiv \langle u/x\rangle\langle v/y\rangle t$, where $y \notin \mathsf{FV}(u)$ and $x \notin \mathsf{FV}(v)$, for which we have:

- if $w \in \mathsf{FV}(t) \setminus \{x, y\}$, then
  $\mathcal{M}_w(\langle v/y\rangle\langle u/x\rangle t) = \mathcal{M}_w(t) = \mathcal{M}_w(\langle u/x\rangle\langle v/y\rangle t)$;

- if $w \in \mathsf{FV}(u)$, then
  $\mathcal{M}_w(\langle v/y\rangle\langle u/x\rangle t) = \mathcal{M}_x(t) \cdot (\mathcal{M}_w(u) + 1) = \mathcal{M}_w(\langle u/x\rangle\langle v/y\rangle t)$;

- if $w \in \mathsf{FV}(v)$, then
  $\mathcal{M}_w(\langle v/y\rangle\langle u/x\rangle t) = \mathcal{M}_y(t) \cdot (\mathcal{M}_w(v) + 1) = \mathcal{M}_w(\langle u/x\rangle\langle v/y\rangle t)$.

We then obtain

$$\mathcal{T}\mathsf{x}(\langle v/y\rangle\langle u/x\rangle t) = \mathcal{T}\mathsf{x}(t) + \mathcal{M}_x(t) \cdot \mathcal{T}\mathsf{x}(u) + \mathcal{M}_y(t) \cdot \mathcal{T}\mathsf{x}(v) = \mathcal{T}\mathsf{x}(\langle u/x\rangle\langle v/y\rangle t)$$

□

We have now to show that the term complexity does not increase during xr-reduction. In particular, the term complexity strictly decreases for some rules and it remains equal for others. This relies on the fact that the multiplicities cannot increase.

Lemma 113 (Decrease of multiplicities and term complexities)

1. *If* $t \longrightarrow_{\mathsf{xr}} u$, *then for all* $w \in \mathsf{FV}(t)$, $\mathcal{M}_w(t) \geq \mathcal{M}_w(u)$.
2. *If* $t \longrightarrow_{\mathsf{xr}} u$, *then* $\mathcal{T}\mathsf{x}(t) \geq \mathcal{T}\mathsf{x}(u)$. *Moreover, if* $t \longrightarrow_{\mathsf{Var,Weak1,Cont,Comp}} u$, *then* $\mathcal{T}\mathsf{x}(t) > \mathcal{T}\mathsf{x}(u)$.

Proof:

1. Since the congruence steps preserve the multiplicity, we only have to consider the basic reduction relation. This is done by induction on the reduction step, the base cases being shown in Fig. 5.5. Note that we use the fact that $\mathcal{M}_x(t) > 0$ (provided $x \in \mathsf{FV}(t)$) and $\mathcal{T}\mathsf{x}(t) > 0$.

2. Since the congruence steps preserve the term complexity, we only have to consider the basic reduction relation. The proof can be done by structural induction on terms. The inductive cases are straightforward by using by the first point. We show in Fig. 5.6 the root reductions.

   The last line holds because the term complexity measure forgets weakenings, contractions, abstractions and applications.

□

	Left-hand side Right-hand side
(Var)	$\langle u/x\rangle x \longrightarrow u$ $\mathcal{M}_w(u)+1 > \mathcal{M}_w(u)$
(Weak1) $w \in \mathsf{FV}(u)$ $w \in \mathsf{FV}(t) \setminus \{x\}$	$\langle u/x\rangle \mathcal{W}_x(t) \longrightarrow \mathcal{W}_{\mathsf{FV}(u)}(t)$ $\mathcal{M}_w(u)+1 > 1$ $\mathcal{M}_w(t) = \mathcal{M}_w(t)$
(Cont) $w \in \mathsf{FV}(u) = \Phi$ $w \in \mathsf{FV}(t) \setminus \{x,y,z\}$	$\langle u/x\rangle \mathcal{C}_x^{y\,\vert\,z}(t) \longrightarrow \mathcal{C}_\Phi^{\Psi\,\vert\,\Upsilon}(\langle \{\Upsilon/\Phi\} u/z\rangle \langle \{\Psi/\Phi\} u/y\rangle t)$ $(\mathcal{M}_y(t)+\mathcal{M}_z(t)+1)\cdot(\mathcal{M}_w(u)+1) > \mathcal{M}_y(t)\cdot(\mathcal{M}_w(u)+1)+\mathcal{M}_z(t)\cdot(\mathcal{M}_w(u)+1)+1$ $\mathcal{M}_w(t) = \mathcal{M}_w(t)$
(Comp) $w \in \mathsf{FV}(t) \setminus \{y\}$ $w \in \mathsf{FV}(v) \setminus \{x\}$ $w \in \mathsf{FV}(u)$	$\langle u/x\rangle\langle v/y\rangle t \longrightarrow \langle\langle u/x\rangle v/y\rangle t$ $\mathcal{M}_w(t) = \mathcal{M}_w(t)$ $\mathcal{M}_y(t)\cdot(\mathcal{M}_w(v)+1) = \mathcal{M}_y(t)\cdot(\mathcal{M}_w(v)+1)$ $\mathcal{M}_y(t)\cdot(\mathcal{M}_x(v)+1)\cdot(\mathcal{M}_w(u)+1) > \mathcal{M}_y(t)\cdot(\mathcal{M}_x(v)\cdot(\mathcal{M}_w(u)+1)+1)$
Other x $w \in \mathsf{FV}(t) \setminus \{x\}$ $w \in \mathsf{FV}(u)$	$\langle u/x\rangle t \longrightarrow t'$ $\mathcal{M}_w(t) = \mathcal{M}_w(t)$ $\mathcal{M}_w(u) = \mathcal{M}_w(u)$
(Merge) $w = w'$ $w \neq w'$	$\mathcal{C}_{w'}^{y\,\vert\,z}(\mathcal{W}_y(t)) \longrightarrow \left\{w'/z\right\} t$ $\mathcal{M}_z(t)+2 > \mathcal{M}_z(t)$ $\mathcal{M}_w(t) = \mathcal{M}_w(t)$
(WSubs) $w \in \mathsf{FV}(t) \setminus \{x\}$ $w = y$ $w \in \mathsf{FV}(u)$	$\langle \mathcal{W}_y(u)/x\rangle t \longrightarrow \mathcal{W}_y(\langle u/x\rangle t)$ $\mathcal{M}_w(t) = \mathcal{M}_w(t)$ $\mathcal{M}_x(t)\cdot(1+1) > 1$ $\mathcal{M}_x(t)\cdot(\mathcal{M}_w(u)+1) = \mathcal{M}_x(t)\cdot(\mathcal{M}_w(u)+1)$
(CSubs) $w \in \mathsf{FV}(t) \setminus \{y'\}$ $w = z$ $w \in \mathsf{FV}(u)$	$\mathcal{C}_z^{x\,\vert\,y}(\langle u/y'\rangle t) \longrightarrow \langle \mathcal{C}_z^{x\,\vert\,y}(u)/y'\rangle t$ $\mathcal{M}_w(t) = \mathcal{M}_w(t)$ $\mathcal{M}_{y'}(t)\cdot(\mathcal{M}_x(u)+\mathcal{M}_y(u)+2)+1 > \mathcal{M}_{y'}(t)\cdot(\mathcal{M}_x(u)+\mathcal{M}_y(u)+1+1)$ $\mathcal{M}_{y'}(t)\cdot(\mathcal{M}_w(u)+1) = \mathcal{M}_{y'}(t)\cdot(\mathcal{M}_w(u)+1)$
Other r	$t \longrightarrow t'$ $\mathcal{M}_w(t) = \mathcal{M}_w(t')$

Figure 5.5: Decrease of multiplicities

Note that this does not hold for rule B. For instance, $t = (\lambda x.\mathcal{C}_x^{x_1\,\vert\,x_2}(x_1\ x_2))\ \lambda y.\mathcal{C}_y^{y_1\,\vert\,y_2}(y_1\ y_2) \longrightarrow_B \langle \lambda y.\mathcal{C}_y^{y_1\,\vert\,y_2}(y_1\ y_2)/x\rangle \mathcal{C}_x^{x_1\,\vert\,x_2}(x_1\ x_2) = u$ but $\mathcal{T}\mathsf{x}(t) = 4$ and $\mathcal{T}\mathsf{x}(u) = 8$.

We now use another measure to show the termination of the subsystem of xr containing only the rules that might not decrease the term complexity.

Left-hand side		Right-hand side
$\langle u/x\rangle x$	$\longrightarrow_{\mathsf{Var}}$	u
$1 + \mathcal{T}\mathsf{x}(u)$	$>$	$\mathcal{T}\mathsf{x}(u)$
$\langle u/x\rangle \mathcal{W}_x(t)$	$\longrightarrow_{\mathsf{Weak1}}$	$\mathcal{W}_{\mathsf{FV}(u)}(t)$
$\mathcal{T}\mathsf{x}(t) + \mathcal{T}\mathsf{x}(u)$	$>$	$\mathcal{T}\mathsf{x}(t)$
$\langle u/x\rangle \mathcal{C}_x^{y\,\vert\,z}(t)$	$\longrightarrow_{\mathsf{Cont}}$	$\mathcal{C}_\Phi^{\Psi\,\vert\,\Upsilon}(\langle\{\Upsilon/\!\!/_\Phi\}u/z\rangle\langle\{\Psi/\!\!/_\Phi\}u/y\rangle t)$
$\mathcal{T}\mathsf{x}(t) + \mathcal{T}\mathsf{x}(u)\cdot(\mathcal{M}_y(t) + \mathcal{M}_z(t) + 1)$	$>$	$\mathcal{T}\mathsf{x}(t) + \mathcal{T}\mathsf{x}(u)\cdot\mathcal{M}_y(t) + \mathcal{T}\mathsf{x}(u)\cdot\mathcal{M}_z(t)$
$\langle u/x\rangle\langle v/y\rangle t$	$\longrightarrow_{\mathsf{Comp}}$	$\langle\langle u/x\rangle v/y\rangle t$
$\mathcal{T}\mathsf{x}(t) + \mathcal{M}_y(t)\cdot(\mathcal{T}\mathsf{x}(v) + (\mathcal{M}_x(v)+1)\cdot\mathcal{T}\mathsf{x}(u))$	$>$	$\mathcal{T}\mathsf{x}(t) + \mathcal{M}_y(t)\cdot(\mathcal{T}\mathsf{x}(v) + \mathcal{M}_x(v)\cdot\mathcal{T}\mathsf{x}(u))$
t	$\longrightarrow_{\text{Other }\mathsf{xr}}$	t'
$\mathcal{T}\mathsf{x}(t)$	$=$	$\mathcal{T}\mathsf{x}(t')$

Figure 5.6: Decrease of term complexity

$\mathcal{P}(x)$	$:=$	2	$\mathcal{P}(\langle u/x\rangle t)$	$:=$	$\mathcal{P}(t)\cdot(\mathcal{P}(u)+1)$
$\mathcal{P}(\lambda x.t)$	$:=$	$2\cdot\mathcal{P}(t)+2$	$\mathcal{P}(t\ u)$	$:=$	$2\cdot(\mathcal{P}(t)+\mathcal{P}(u))+2$
$\mathcal{P}(\mathcal{W}_x(t))$	$:=$	$\mathcal{P}(t)+1$	$\mathcal{P}(\mathcal{C}_x^{y\,\vert\,z}(t))$	$:=$	$2\cdot\mathcal{P}(t)$

Figure 5.7: Mapping $\mathcal{P}$ to natural numbers

Definition 94 We define an mapping $\mathcal{P}$ from λlxr-terms to natural numbers as resented in Fig. 5.7.

Lemma 114 *The mapping $\mathcal{P}$ is invariant under conversion by $\equiv$.*

Proof: The polynomial interpretation is blind to the variables' names, so it is trivially sound with respect to α-conversion, and rules $\mathsf{A_c}$, $\mathsf{C_c}$, $\mathsf{P_c}$ and $\mathsf{P_w}$. For the equivalence rule $\mathsf{P_s}$ we have by commutativity of multiplication the following equality:

$$\mathcal{P}(\langle v/y\rangle\langle u/x\rangle t) = \mathcal{P}(t)\cdot\mathcal{P}(u)\cdot\mathcal{P}(v) = \mathcal{P}(\langle u/x\rangle\langle v/y\rangle t)$$

For the equivalence rule $\mathsf{P_{cs}}$ we have:

$$\mathcal{P}(\langle u/x\rangle\mathcal{C}_w^{y\,\vert\,z}(t)) = 2\cdot\mathcal{P}(t)\cdot\mathcal{P}(u) + 2\cdot\mathcal{P}(t) = \mathcal{P}(\mathcal{C}_w^{y\,\vert\,z}(\langle u/x\rangle t))$$

□

Lemma 115 (Simulation through $\mathcal{P}$) *If $t \longrightarrow_{\mathsf{xr}} u$ and the reduction is neither Var, Weak1, Cont nor Comp, then $\mathcal{P}(t) > \mathcal{P}(u)$.*

Proof: Since the mapping is invariant under the congruence, we only have to consider the basic reduction relation. The proof can be done by structural induction on terms. The cases of root reductions are given in Fig. 5.8:

Rule	Left-hand side		Right-hand side
(Abs)	$(2 \cdot \mathcal{P}(t) + 2) \cdot (\mathcal{P}(u) + 1)$	$>$	$2 \cdot \mathcal{P}(t) \cdot (\mathcal{P}(u) + 1) + 2$
(App1)	$(2 \cdot (\mathcal{P}(t) + \mathcal{P}(v)) + 2) \cdot (\mathcal{P}(u) + 1)$	$>$	$2 \cdot (\mathcal{P}(t) \cdot (\mathcal{P}(u) + 1) + \mathcal{P}(v)) + 2$
(App2)	$(2 \cdot (\mathcal{P}(t) + \mathcal{P}(v)) + 2) \cdot (\mathcal{P}(u) + 1)$	$>$	$2 \cdot (\mathcal{P}(t) + \mathcal{P}(v) \cdot (\mathcal{P}(u) + 1)) + 2$
(Weak2)	$(\mathcal{P}(t) + 1) \cdot (\mathcal{P}(u) + 1)$	$>$	$\mathcal{P}(t) \cdot (\mathcal{P}(u) + 1) + 1$
(WAbs)	$2 \cdot (\mathcal{P}(t) + 1) + 2$	$>$	$2 \cdot \mathcal{P}(t) + 2 + 1$
(WApp1)	$2 \cdot (\mathcal{P}(u) + 1 + \mathcal{P}(v)) + 2$	$>$	$2 \cdot (\mathcal{P}(u) + \mathcal{P}(v)) + 2 + 1$
(WApp2)	$2 \cdot (\mathcal{P}(u) + \mathcal{P}(v) + 1) + 2$	$>$	$2 \cdot (\mathcal{P}(u) + \mathcal{P}(v)) + 2 + 1$
(WSubs)	$\mathcal{P}(t) \cdot (\mathcal{P}(u) + 1 + 1)$	$>$	$\mathcal{P}(t) \cdot (\mathcal{P}(u) + 1) + 1$
(Merge)	$2 \cdot (\mathcal{P}(t) + 1)$	$>$	$\mathcal{P}(t)$
(Cross)	$2 \cdot (\mathcal{P}(t) + 1)$	$>$	$2 \cdot \mathcal{P}(t) + 1$
(CAbs)	$2 \cdot (2 \cdot \mathcal{P}(t) + 2)$	$>$	$2 \cdot (2 \cdot \mathcal{P}(t)) + 2$
(CApp1)	$2 \cdot (2 \cdot (\mathcal{P}(t) + \mathcal{P}(u)) + 2)$	$>$	$2 \cdot (2 \cdot \mathcal{P}(t) + \mathcal{P}(u)) + 2$
(CApp2)	$2 \cdot (2 \cdot (\mathcal{P}(t) + \mathcal{P}(u)) + 2)$	$>$	$2 \cdot (\mathcal{P}(t) + 2 \cdot \mathcal{P}(u)) + 2$
(CSubs)	$2 \cdot \mathcal{P}(t) \cdot (\mathcal{P}(u) + 1)$	$>$	$\mathcal{P}(t) \cdot (2 \cdot \mathcal{P}(u) + 1)$

Figure 5.8: Simulation through $\mathcal{P}$

□

We can conclude this section with the following property:

Theorem 116 *The system* xr *is terminating.*

Proof: By Corollary 26 (xr-reduction decreases the pair of integers $(\mathcal{T}\mathsf{x}(t), \mathcal{P}(t))$ w.r.t. the lexicographical order). □

5.1.5 Typing

In this section we present the *simply-typed* λlxr-calculus. The typing system ensures strong normalisation (as in the λ-calculus) and also linearity.

The typing rules of the simply-typed λlxr-calculus are shown in Fig. 5.9. The derivability of a sequent $\Gamma \vdash t : A$ in this system is denoted $\Gamma \vdash_{\lambda\mathsf{lxr}} t : A$.

Remark that $\Gamma \vdash_{\lambda\mathsf{lxr}} t : A$ implies that $\mathsf{Dom}(\Gamma) = \mathsf{FV}(t)$.

Lemma 117 *The following rule is height-preserving admissible in the typing system of* λlxr*:*

$$\frac{\Gamma, \Delta \vdash t : A}{\{\Phi/\_\}\Gamma, \Delta \vdash \{\Phi/\mathsf{Dom}(\Gamma)\}t : A}$$

$$\frac{}{x : A \vdash x : A}\ \mathsf{ax}_m \qquad \frac{\Delta \vdash u : B \quad \Gamma, x : B \vdash t : A}{\Gamma, \Delta \vdash \langle u/x \rangle t : A}\ \mathsf{cut}_m$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \to B}\ \to_{\mathsf{right}} \qquad \frac{\Gamma \vdash t : A \to B \quad \Delta \vdash v : A}{\Gamma, \Delta \vdash t\ v : B}\ \to_{\mathsf{elim}m}$$

$$\frac{\Gamma \vdash t : A}{\Gamma, x : B \vdash \mathcal{W}_x(t) : A}\ \mathsf{weak} \qquad \frac{\Gamma, x : A, y : A \vdash M : B}{\Gamma, z : A \vdash \mathcal{C}_z^{x \mid y}(M) : B}\ \mathsf{cont}$$

Figure 5.9: Typing Rules for λlxr-terms

where $\left\{{}^{y_1,\ldots,y_n}\!/_{\_}\right\}\{x_1 \colon A_1, \ldots, x_n \colon A_n\} := \{y_1 \colon A_1, \ldots, y_n \colon A_n\}$ *(provided* $x_1, \ldots, x_n$ *are ordered according to the total order on the set of variables).*

The following rules are derivable in the typing system of λlxr:

$$\frac{\Delta \vdash t \colon A}{\Gamma, \Delta \vdash \mathcal{W}_{\mathsf{Dom}(\Gamma)}(t) \colon A} \qquad \frac{\left\{{}^{\Phi}\!/_{\_}\right\}\Gamma, \left\{{}^{\Pi}\!/_{\_}\right\}\Gamma, \Delta \vdash t \colon A}{\Gamma, \Delta \vdash \mathcal{C}^{\Phi \mid \Pi}_{\mathsf{Dom}(\Gamma)}(t) \colon A}$$

Proof: The admissibility of the first rule is proved by a routine induction on (the size of) t. For the next two rules an induction on the cardinal of $\mathsf{Dom}(\Gamma)$ suffices. □

As expected, the following holds:

Theorem 118 (Subject reduction)

- *If* $\Gamma \vdash_{\lambda lxr} s : A$ *and* $s \equiv s'$*, then* $\Gamma \vdash_{\lambda lxr} s' : A$*.*
- *If* $\Gamma \vdash_{\lambda lxr} s : A$ *and* $s \longrightarrow_{\lambda lxr} s'$*, then* $\Gamma \vdash_{\lambda lxr} s' : A$*.*

Proof: The proof of the first point is straightforward, by an induction on the derivation of the reduction step and by case analysis. We can easily re-compose from the hypothesis $\Gamma \vdash_{\lambda\mathsf{lxr}} s : A$ the last steps of its derivation and rearrange the sub-derivations to conclude $\Gamma \vdash_{\lambda\mathsf{lxr}} s' : A$ as follows:

- For $\mathcal{C}_w^{x \mid v}(\mathcal{C}_x^{y \mid z}(t)) \equiv \mathcal{C}_w^{y \mid x}(\mathcal{C}_x^{z \mid v}(t))$ we have

$$\frac{\dfrac{\Gamma, y : B, z : B, v : B \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, x : B, v : B \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_x^{y \mid z}(t) : A}}{\Gamma, w : B \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_w^{x \mid v}(\mathcal{C}_x^{y \mid z}(t)) : A} \qquad \frac{\dfrac{\Gamma, y : B, z : B, v : B \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, y : B, x : B \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_x^{z \mid v}(t) : A}}{\Gamma, w : B \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_w^{y \mid x}(\mathcal{C}_x^{z \mid v}(t)) : A}$$

- For $\mathcal{C}_x^{y \mid z}(t) \equiv \mathcal{C}_x^{z \mid y}(t)$ we have

$$\frac{\Gamma, y : B, z : B \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, x : B \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_x^{y \mid z}(t) : A} \qquad \frac{\Gamma, y : B, z : B \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, x : B \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_x^{z \mid y}(t) : A}$$

- For $\mathcal{C}_{x'}^{y'\,|\,z'}(\mathcal{C}_{x}^{y\,|\,z}(t)) \equiv \mathcal{C}_{x'}^{y'\,|\,z'}(\mathcal{C}_{x}^{y\,|\,z}(t))$ we have on the one hand

$$\dfrac{\dfrac{\Gamma, y : B, z : B, y' : C, z' : C \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, x : B, y' : C, z' : C \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_{x}^{y\,|\,z}(t) : A}}{\Gamma, x : B, x' : C \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_{x'}^{y'\,|\,z'}(\mathcal{C}_{x}^{y\,|\,z}(t)) : A}$$

  and on the other hand

$$\dfrac{\dfrac{\Gamma, y : B, z : B, y' : C, z' : C \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, y : B, z : B, x' : C \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_{x'}^{y'\,|\,z'}(t) : A}}{\Gamma, x : B, x' : C \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_{x}^{y\,|\,z}(\mathcal{C}_{x'}^{y'\,|\,z'}(t)) : A}$$

- For $\mathcal{W}_x(\mathcal{W}_y(t)) \equiv \mathcal{W}_y(\mathcal{W}_x(t))$

$$\dfrac{\dfrac{\Gamma \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, y : B \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_y(t) : A}}{\Gamma, y : B, x : C \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_x(\mathcal{W}_y(t)) : A} \qquad \dfrac{\dfrac{\Gamma \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, x : C \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_x(t) : A}}{\Gamma, y : B, x : C \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_y(\mathcal{W}_x(t)) : A}$$

- For $\langle v/y\rangle\langle u/x\rangle t \equiv \langle u/x\rangle\langle v/y\rangle t$ we have on the one hand

$$\dfrac{\Pi \vdash_{\lambda\mathsf{lxr}} v : B \qquad \dfrac{\Delta \vdash_{\lambda\mathsf{lxr}} u : C \quad \Gamma, y : B, x : C \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, y : B, \Delta \vdash_{\lambda\mathsf{lxr}} \langle u/x\rangle t : A}}{\Gamma, \Delta, \Pi \vdash_{\lambda\mathsf{lxr}} \langle v/y\rangle\langle u/x\rangle t : A}$$

  and on the other hand,

$$\dfrac{\Delta \vdash_{\lambda\mathsf{lxr}} u : C \qquad \dfrac{\Pi \vdash_{\lambda\mathsf{lxr}} v : B \quad \Gamma, y : B, x : C \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, x : C, \Pi \vdash_{\lambda\mathsf{lxr}} \langle v/y\rangle t : A}}{\Gamma, \Pi, \Delta \vdash_{\lambda\mathsf{lxr}} \langle u/x\rangle\langle v/y\rangle t : A}$$

- For $\langle v/x\rangle\mathcal{C}_{w}^{y\,|\,z}(t) \equiv \mathcal{C}_{w}^{y\,|\,z}(\langle v/x\rangle t)$ we have on the one hand

$$\dfrac{\Pi \vdash_{\lambda\mathsf{lxr}} v : B \qquad \dfrac{\Gamma, y : C, z : C, x : B \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, w : C, x : B \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_{w}^{y\,|\,z}(t) : A}}{\Gamma, w : C, \Pi \vdash_{\lambda\mathsf{lxr}} \langle v/x\rangle\mathcal{C}_{w}^{y\,|\,z}(t) : A}$$

  and on the other hand

$$\dfrac{\dfrac{\Pi \vdash_{\lambda\mathsf{lxr}} v : B \quad \Gamma, y : C, z : C, x : B \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, y : C, z : C, \Pi \vdash_{\lambda\mathsf{lxr}} \langle v/x\rangle t : A}}{\Gamma, w : C, \Pi \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_{w}^{y\,|\,z}(\langle v/x\rangle t) : A}$$

Using the first point leaves only the basic reduction to be checked in the second point. This is also straightforward and proved again by an induction on the reduction step and by case analysis.

- (B): We have $s = (\lambda x.t)\ u$ and $s' = \langle u/x\rangle t$.

$$\dfrac{\dfrac{\Gamma, x : B \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma \vdash_{\lambda\mathsf{lxr}} \lambda x.t : B{\to}A} \quad \Delta \vdash_{\lambda\mathsf{lxr}} u : B}{\Gamma, \Delta \vdash_{\lambda\mathsf{lxr}} (\lambda x.t)\ u : A} \qquad \dfrac{\Delta \vdash_{\lambda\mathsf{lxr}} u : B \quad \Gamma, x : B \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, \Delta \vdash_{\lambda\mathsf{lxr}} \langle u/x\rangle t : A}$$

- (Abs): We have $s = \langle u/x\rangle(\lambda y.t)$, $s' = \lambda y.\langle u/x\rangle t$ and $A = B{\to}C$.

$$\dfrac{\Delta \vdash_{\lambda\mathsf{lxr}} u : D \quad \dfrac{\Gamma, x : D, y : B \vdash_{\lambda\mathsf{lxr}} t : C}{\Gamma, x : D \vdash_{\lambda\mathsf{lxr}} \lambda y.t : B{\to}C}}{\Gamma, \Delta \vdash_{\lambda\mathsf{lxr}} \langle u/x\rangle(\lambda y.t) : B{\to}C}$$

$$\dfrac{\dfrac{\Delta \vdash_{\lambda\mathsf{lxr}} u : D \quad \Gamma, x : D, y : B \vdash_{\lambda\mathsf{lxr}} t : C}{\Gamma, y : B, \Delta \vdash_{\lambda\mathsf{lxr}} \langle u/x\rangle t : C}}{\Gamma, \Delta \vdash_{\lambda\mathsf{lxr}} \lambda y.\langle u/x\rangle t : B{\to}C}$$

- $(\mathsf{App1})$: We have $s = \langle u/x\rangle(t\ v)$ and $s' = \langle u/x\rangle t\ v$.

$$\dfrac{\Pi \vdash_{\lambda\mathsf{lxr}} u : B \quad \dfrac{\Gamma, x : B \vdash_{\lambda\mathsf{lxr}} t : C{\to}A \quad \Delta \vdash_{\lambda\mathsf{lxr}} v : C}{\Gamma, \Delta, x : B \vdash_{\lambda\mathsf{lxr}} t\ v : A}}{\Gamma, \Delta, \Pi \vdash_{\lambda\mathsf{lxr}} \langle u/x\rangle(t\ v) : A}$$

$$\dfrac{\dfrac{\Pi \vdash_{\lambda\mathsf{lxr}} u : B \quad \Gamma, x : B \vdash_{\lambda\mathsf{lxr}} t : C{\to}A}{\Gamma, \Pi \vdash_{\lambda\mathsf{lxr}} \langle u/x\rangle t : C{\to}A} \qquad \Delta \vdash_{\lambda\mathsf{lxr}} v : C}{\Gamma, \Pi, \Delta \vdash_{\lambda\mathsf{lxr}} \langle u/x\rangle t\ v : A}$$

- $(\mathsf{App2})$: Similar to the previous case.
- (Var): We have $s = \langle u/x\rangle x$ and $s' = u$.

$$\dfrac{\dfrac{}{x : A \vdash_{\lambda\mathsf{lxr}} x : A} \quad \Gamma \vdash_{\lambda\mathsf{lxr}} u : A}{\Gamma \vdash_{\lambda\mathsf{lxr}} \langle u/x\rangle x : A} \qquad \Gamma \vdash_{\lambda\mathsf{lxr}} u : A$$

- (Weak1): We have $s = \langle u/x\rangle \mathcal{W}_x(t)$ and $s' = \mathcal{W}_{\mathsf{FV}(u)}(t)$.

$$\dfrac{\dfrac{\Gamma \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, x : B \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_x(t) : A} \quad \Delta \vdash_{\lambda\mathsf{lxr}} u : B}{\Gamma, \Delta \vdash_{\lambda\mathsf{lxr}} \langle u/x\rangle \mathcal{W}_x(t) : A} \qquad \dfrac{\Gamma \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, \Delta \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_{\mathsf{FV}(u)}(t) : A}$$

  since $\mathsf{Dom}(\Delta) = \mathsf{FV}(u)$.

- (Weak2): We have $s = \langle u/x\rangle \mathcal{W}_y(t)$ and $s' = \mathcal{W}_y(\langle u/x\rangle t)$ with $x \neq y$.

$$\dfrac{\dfrac{\Gamma, x : B \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, y : C, x : B \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_y(t) : A} \quad \Delta \vdash_{\lambda\mathsf{lxr}} u : B}{\Gamma, y : C, \Delta \vdash_{\lambda\mathsf{lxr}} \langle u/x\rangle \mathcal{W}_y(t) : A}$$

$$\dfrac{\dfrac{\Gamma, x : B \vdash_{\lambda\mathsf{lxr}} t : A \quad \Delta \vdash_{\lambda\mathsf{lxr}} u : B}{\Gamma, \Delta \vdash_{\lambda\mathsf{lxr}} \langle u/x\rangle t : A}}{\Gamma, y : C, \Delta \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_y(\langle u/x\rangle t) : A}$$

- (Cont): $s = \langle v/x\rangle \mathcal{C}_x^{y\,|\,z}(t)$ and $s' = \mathcal{C}^{\Phi\,|\,\Sigma}_{\mathsf{FV}(v)}(\langle \{{}^{\Sigma}\!/_{\mathsf{FV}(v)}\} v/z\rangle \langle \{{}^{\Phi}\!/_{\mathsf{FV}(v)}\} v/y\rangle t)$.

$$\dfrac{\dfrac{\Gamma, y : B, z : B \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, x : B \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_x^{y\,|\,z}(t) : A} \quad \Delta \vdash_{\lambda\mathsf{lxr}} v : B}{\Gamma, \Delta \vdash_{\lambda\mathsf{lxr}} \langle v/x\rangle \mathcal{C}_x^{y\,|\,z}(t) : A}$$

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma, z : B, \{{}^{\Phi}\!/_{\_}\}\Delta \vdash_{\lambda\mathsf{lxr}} \langle \{{}^{\Phi}\!/_{\mathsf{FV}(v)}\} v/y\rangle t : A} \quad \dfrac{\Delta \vdash_{\lambda\mathsf{lxr}} v : B}{\{{}^{\Sigma}\!/_{\_}\}\Delta \vdash_{\lambda\mathsf{lxr}} \{{}^{\Sigma}\!/_{\mathsf{FV}(v)}\} v : B}}{\dfrac{\Gamma, \{{}^{\Phi}\!/_{\_}\}\Delta, \{{}^{\Sigma}\!/_{\_}\}\Delta \vdash_{\lambda\mathsf{lxr}} \langle \{{}^{\Sigma}\!/_{\mathsf{FV}(v)}\} v/z\rangle \langle \{{}^{\Phi}\!/_{\mathsf{FV}(v)}\} v/y\rangle t : A}{\Gamma, \Delta \vdash_{\lambda\mathsf{lxr}} \mathcal{C}^{\Phi\,|\,\Sigma}_{\mathsf{FV}(v)}(\langle \{{}^{\Sigma}\!/_{\mathsf{FV}(v)}\} v/z\rangle \langle \{{}^{\Phi}\!/_{\mathsf{FV}(v)}\} v/y\rangle t) : A}}$$

  since $\mathsf{Dom}(\Delta) = \mathsf{FV}(v)$, with $\mathcal{D}$ being the following derivation:

$$\dfrac{\Gamma, y : B, z : B \vdash_{\lambda\mathsf{lxr}} t : A \quad \dfrac{\Delta \vdash_{\lambda\mathsf{lxr}} v : B}{\{{}^{\Phi}\!/_{\_}\}\Delta \vdash_{\lambda\mathsf{lxr}} \{{}^{\Phi}\!/_{\mathsf{FV}(v)}\} v : B}}{\Gamma, z : B, \{{}^{\Phi}\!/_{\_}\}\Delta \vdash_{\lambda\mathsf{lxr}} \langle \{{}^{\Phi}\!/_{\mathsf{FV}(v)}\} v/y\rangle t : A}$$

- (Comp): $s = \langle v/x\rangle\langle u/y\rangle t$ and $s' = \langle\langle v/x\rangle u/y\rangle t$.

$$\dfrac{\Pi \vdash_{\lambda\mathsf{lxr}} v : B \qquad \dfrac{\Delta, x : B \vdash_{\lambda\mathsf{lxr}} u : C \quad \Gamma, y : C \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, \Delta, x : B \vdash_{\lambda\mathsf{lxr}} \langle u/y\rangle t : A}}{\Gamma, \Delta, \Pi \vdash_{\lambda\mathsf{lxr}} \langle v/x\rangle\langle u/y\rangle t : A}$$

$$\dfrac{\dfrac{\Pi \vdash_{\lambda\mathsf{lxr}} v : B \quad \Delta, x : B \vdash_{\lambda\mathsf{lxr}} u : C}{\Delta, \Pi \vdash_{\lambda\mathsf{lxr}} \langle v/x\rangle u : C} \qquad \Gamma, y : C \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, \Delta, \Pi \vdash_{\lambda\mathsf{lxr}} \langle\langle v/x\rangle u/y\rangle t : A}$$

- (WAbs): We have $s = \mathcal{W}_y(\lambda x.t)$ and $s' = \lambda x.\mathcal{W}_y(t)$.

$$\dfrac{\dfrac{\Gamma, x : B \vdash_{\lambda\mathsf{lxr}} t : C}{\Gamma \vdash_{\lambda\mathsf{lxr}} \lambda x.t : B{\to}C}}{y : D, \Gamma \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_y(\lambda x.t) : B{\to}C} \qquad \dfrac{\dfrac{\Gamma, x : B \vdash_{\lambda\mathsf{lxr}} t : C}{y : D, \Gamma, x : B \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_y(t) : C}}{y : D, \Gamma \vdash_{\lambda\mathsf{lxr}} \lambda x.\mathcal{W}_y(t) : B{\to}C}$$

- (WApp1): We have $s = \mathcal{W}_y(u)\ v$ and $s' = \mathcal{W}_y(u\ v)$.

$$\dfrac{\dfrac{\Gamma \vdash_{\lambda\mathsf{lxr}} u : B{\to}C}{\Gamma, y : D \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_y(u) : B{\to}C} \quad \Delta \vdash_{\lambda\mathsf{lxr}} v : B}{\Gamma, y : D, \Delta \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_y(u)\ v : C}$$

$$\dfrac{\dfrac{\Gamma \vdash_{\lambda\mathsf{lxr}} u : B{\to}C \quad \Delta \vdash_{\lambda\mathsf{lxr}} v : B}{\Gamma, \Delta \vdash_{\lambda\mathsf{lxr}} u\ v : C}}{\Gamma, y : D, \Delta \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_y(u\ v) : C}$$

- (WApp2): Similar to the previous case.

- (WSubs): We have $s = \langle\mathcal{W}_y(u)/x\rangle t$ and $s' = \mathcal{W}_y(\langle u/x\rangle t)$.

$$\dfrac{\dfrac{\Gamma \vdash_{\lambda\mathsf{lxr}} u : B}{\Gamma, y : C \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_y(u) : B} \quad \Delta, x : B \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, y : C, \Delta \vdash_{\lambda\mathsf{lxr}} \langle\mathcal{W}_y(u)/x\rangle t : A}$$

$$\dfrac{\dfrac{\Gamma \vdash_{\lambda\mathsf{lxr}} u : B \quad \Delta, x : B \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, \Delta \vdash_{\lambda\mathsf{lxr}} \langle u/x\rangle t : A}}{\Gamma, y : C, \Delta \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_y(\langle u/x\rangle t) : A}$$

- (Merge): $s = \mathcal{C}_w^{y\,|\,z}(\mathcal{W}_y(t))$ and $s' = t$.

$$\dfrac{\dfrac{\Gamma, z : C \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, y : C, z : C \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_y(t) : A}}{\Gamma, w : C \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_w^{y\,|\,z}(\mathcal{W}_y(t)) : A} \qquad \dfrac{\Gamma, z : C \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, w : C \vdash_{\lambda\mathsf{lxr}} \{w/z\}t : A}$$

- (Cross): $s = \mathcal{C}_w^{y\,|\,z}(\mathcal{W}_x(t))$ and $s' = \mathcal{W}_x(\mathcal{C}_w^{y\,|\,z}(t))$.

$$\dfrac{\dfrac{\Gamma, y : C, z : C \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, y : C, z : C, x : B \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_x(t) : A}}{\Gamma, w : C, x : B \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_w^{y\,|\,z}(\mathcal{W}_x(t)) : A} \qquad \dfrac{\dfrac{\Gamma, y : C, z : C \vdash_{\lambda\mathsf{lxr}} t : A}{\Gamma, w : C \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_w^{y\,|\,z}(t) : A}}{\Gamma, w : C, x : B \vdash_{\lambda\mathsf{lxr}} \mathcal{W}_x(\mathcal{C}_w^{y\,|\,z}(t)) : A}$$

- (CAbs): $s = \mathcal{C}_w^{y\,|\,z}(\lambda x.t)$ and $s' = \lambda x.\mathcal{C}_w^{y\,|\,z}(t)$.

$$\dfrac{\dfrac{\Gamma, y : D, z : D, x : B \vdash_{\lambda\mathsf{lxr}} t : C}{\Gamma, y : D, z : D \vdash_{\lambda\mathsf{lxr}} \lambda x.t : B{\to}C}}{\Gamma, w : D \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_w^{y\,|\,z}(\lambda x.t) : B{\to}C} \qquad \dfrac{\dfrac{\Gamma, y : D, z : D, x : B \vdash_{\lambda\mathsf{lxr}} t : C}{\Gamma, w : D, x : B \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_w^{y\,|\,z}(t) : C}}{\Gamma, w : D \vdash_{\lambda\mathsf{lxr}} \lambda x.\mathcal{C}_w^{y\,|\,z}(t) : B{\to}C}$$

- (CApp1): $s = \mathcal{C}_w^{y\,|\,z}(t\ u)$ and $s' = \mathcal{C}_w^{y\,|\,z}(t)\ u$.

$$\dfrac{\dfrac{\Gamma, y : C, z : C \vdash_{\lambda\mathsf{lxr}} t : A{\to}B \quad \Delta \vdash_{\lambda\mathsf{lxr}} u : A}{\Gamma, y : C, z : C, \Delta \vdash_{\lambda\mathsf{lxr}} (t\ u) : B}}{\Gamma, w : C, \Delta \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_w^{y\,|\,z}(t\ u) : B}$$

$$\dfrac{\dfrac{\Gamma, y : C, z : C \vdash_{\lambda\mathsf{lxr}} t : A{\to}B}{\Gamma, w : C \vdash_{\lambda\mathsf{lxr}} \mathcal{C}_w^{y\,|\,z}(t) : A{\to}B} \quad \Delta \vdash_{\lambda\mathsf{lxr}} u : A}{\Gamma, w : C, \Delta \vdash_{\lambda\mathsf{lxr}} (\mathcal{C}_w^{y\,|\,z}(t)\ u) : B}$$

- (CApp2): Similar to the previous case.

- (CSubs): $s = \mathcal{C}_w^{y|z}(\langle u/x\rangle t)$ and $s' = \langle \mathcal{C}_w^{y|z}(u)/x\rangle t$.

$$\dfrac{\dfrac{\Gamma, y : B, z : B \vdash_{\lambda \mathsf{lxr}} u : C \quad \Delta, x : C \vdash_{\lambda \mathsf{lxr}} t : A}{\Gamma, \Delta, y : B, z : B \vdash_{\lambda \mathsf{lxr}} \langle u/x\rangle t : A}}{\Gamma, \Delta, w : B \vdash_{\lambda \mathsf{lxr}} \mathcal{C}_w^{y|z}(\langle u/x\rangle t) : A}$$

$$\dfrac{\dfrac{\Gamma, y : B, z : B \vdash_{\lambda \mathsf{lxr}} u : C}{\Gamma, w : B \vdash_{\lambda \mathsf{lxr}} \mathcal{C}_w^{y|z}(u) : C} \quad \Delta, x : C \vdash_{\lambda \mathsf{lxr}} t : A}{\Gamma, \Delta, w : B \vdash_{\lambda \mathsf{lxr}} \langle \mathcal{C}_w^{y|z}(u)/x\rangle t}$$

□

5.2 A reflection in λlxr of λ-calculus

We show in this section the relation between λlxr-terms and λ-terms. We consider λ-terms as an independent syntax rather than particular λlxr-terms, since they might not be linear.

We define two translations $\mathcal{B}$ and $\mathcal{A}$ and establish that they form a reflection in λlxrof λ-calculus. A corollary of the reflection is the confluence of λlxr. We will also show in this section that the two translations $\mathcal{A}$ and $\mathcal{B}$ preserve typing.

In particular, the reflection includes the property that the reduction relation in λlxr simulates (in fact, strongly simulates) β-reduction through $\mathcal{A}$: we show that the linearity constraints and the use of the resource constructors in λlxr decompose the β-reduction step into smaller steps.

5.2.1 From λ-calculus to λlxr-calculus

In this section we investigate an encoding $\mathcal{A}$ from λ-calculus to λlxr, with the strong simulation result (Theorem 121).

Definition 95 (Translation from λ-calculus to λlxr) The encoding of λ-terms is defined by induction as shown in Fig. 5.10.

Using the fact that $\mathcal{W}_\emptyset(t) = t$, we can write the translation of an abstraction, with only one case, as $\mathcal{A}(\lambda x.t) = \lambda x.\mathcal{W}_{\{x\}\setminus \mathsf{FV}(t)}(\mathcal{A}(t))$. Note that $\mathcal{A}(t\ u) = \mathcal{A}(t)\mathcal{A}(u)$ in the particular case $\mathsf{FV}(t) \cap \mathsf{FV}(u) = \emptyset$. More generally, a λ-term which, viewed as a particular λlxr-term, is linear, is translated by $\mathcal{A}$ to itself. Note also that the weakenings and contractions introduced by this translations are already in their canonical places, i.e. $\mathcal{A}(t)$ is an xr-normal form for every λ-term t.

In most of the following proofs, we shall use the following results:

$$
\begin{array}{lcll}
\mathcal{A}(x) & := & x & \\
\mathcal{A}(\lambda x.t) & := & \lambda x.\mathcal{A}(t) & \text{if } x \in \mathsf{FV}(t) \\
\mathcal{A}(\lambda x.t) & := & \lambda x.\mathcal{W}_x(\mathcal{A}(t)) & \text{if } x \notin \mathsf{FV}(t) \\
\mathcal{A}(t\ u) & := & \mathcal{C}_\Phi^{\Upsilon\,|\,\Omega}(\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(t)\ \{{}^{\Omega}\!/_{\Phi}\}\mathcal{A}(u)) & (\text{where } \Phi := \mathsf{FV}(t) \cap \mathsf{FV}(u) \\
 & & & \text{and } \Upsilon, \Omega \text{ are fresh})
\end{array}
$$

Figure 5.10: From λ-calculus to λlxr

Lemma 119 (Properties of $\mathcal{A}$)

1. $FV(t) = FV(\mathcal{A}(t))$.
2. $\mathcal{A}(\{{}^{\Upsilon}\!/_{\Phi}\}t) = \{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(t)$

As a consequence, the encoding of a λ-term is a linear λlxr-term.

Example 11 If $t = \lambda x.\lambda y.y(zz)$, then $\mathcal{A}(t) = \lambda x.\mathcal{W}_x(\lambda y.(y\ \mathcal{C}_z^{z_1\,|\,z_2}(z_1\ z_2)))$.

We now want to simulate a β-reduction step in λlxr, so we start by proving that the interaction between (and the propagation of) the three constructors of λlxr by means of the system xr do implement the notion of substitution. More precisely, given two λ-terms t_1 and t_2, we identify a λlxr-term, built from the translations by $\mathcal{A}$ of t_1 and t_2 and using a substitution constructor, that reduces to the translation of $\{{}^{t_2}\!/_{x}\}t_1$, as shown by the following lemma:

Lemma 120 *For all λ-terms t_1 and t_2 such that $x \in FV(t_1)$,*

$$\mathcal{C}_\Phi^{\Upsilon\,|\,\Omega}(\langle\{{}^{\Omega}\!/_{\Phi}\}\mathcal{A}(t_2)/x\rangle\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(t_1)) \longrightarrow_{xr}^{*} \mathcal{A}(\{{}^{t_2}\!/_{x}\}t_1)$$

where $\Phi := (FV(t_1) \setminus \{x\}) \cap FV(t_2)$, provided that the former term is linear.

In the simple case where $\Phi = \emptyset$, the statement reads:

$$\langle\mathcal{A}(t_2)/x\rangle\mathcal{A}(t_1) \longrightarrow_{xr}^{*} \mathcal{A}(\{{}^{t_2}\!/_{x}\}t_1)$$

Proof: By induction on the size of t_1, by propagating the substitution constructor, pulling out weakenings and pushing in contractions. Note that whenever we use the induction hypothesis throughout the proof, it will be applied to a term which is linear (Lemma 110, Property 1).

1. If t_1 is a variable, then it must be x, so there is no contraction and

$$\langle\mathcal{A}(t_2)/x\rangle x \longrightarrow_{\mathsf{Var}} \mathcal{A}(t_2) = \mathcal{A}(\{{}^{t_2}\!/_{x}\}x)$$

2. If $t_1 = \lambda y.v$ then by α-conversion we can suppose $y \neq x$ and $y \notin \mathsf{FV}(t_2)$.

   (a) If $y \in \mathsf{FV}(v)$ then

$$\begin{array}{ll} & \mathcal{C}_{\Phi}^{\Upsilon\,|\,\Omega}(\langle\{{}^{\Omega}\!/_{\Phi}\}\mathcal{A}(t_2)/x\rangle\{{}^{\Upsilon}\!/_{\Phi}\}\lambda y.\mathcal{A}(v)) \\ = & \mathcal{C}_{\Phi}^{\Upsilon\,|\,\Omega}(\langle\{{}^{\Omega}\!/_{\Phi}\}\mathcal{A}(t_2)/x\rangle(\lambda y.\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(v))) \\ \longrightarrow_{\mathsf{Abs}} & \mathcal{C}_{\Phi}^{\Upsilon\,|\,\Omega}(\lambda y.(\langle\{{}^{\Omega}\!/_{\Phi}\}\mathcal{A}(t_2)/x\rangle\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(v))) \\ \longrightarrow_{\mathsf{CAbs}} & \lambda y.\mathcal{C}_{\Phi}^{\Upsilon\,|\,\Omega}(\langle\{{}^{\Omega}\!/_{\Phi}\}\mathcal{A}(t_2)/x\rangle\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(v)) \end{array}$$

   and we get the result by the induction hypothesis.

   (b) If $y \notin \mathsf{FV}(v)$ then

$$\begin{array}{ll} & \mathcal{C}_{\Phi}^{\Upsilon\,|\,\Omega}(\langle\{{}^{\Omega}\!/_{\Phi}\}\mathcal{A}(t_2)/x\rangle\{{}^{\Upsilon}\!/_{\Phi}\}\lambda y.\mathcal{W}_y(\mathcal{A}(v))) \\ = & \mathcal{C}_{\Phi}^{\Upsilon\,|\,\Omega}(\langle\{{}^{\Omega}\!/_{\Phi}\}\mathcal{A}(t_2)/x\rangle(\lambda y.\mathcal{W}_y(\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(v)))) \\ \longrightarrow_{\mathsf{Abs}} & \mathcal{C}_{\Phi}^{\Upsilon\,|\,\Omega}(\langle\{{}^{\Omega}\!/_{\Phi}\}\mathcal{A}(t_2)/x\rangle\lambda y.(\mathcal{W}_y(\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(v)))) \\ \longrightarrow_{\mathsf{Weak2}} & \mathcal{C}_{\Phi}^{\Upsilon\,|\,\Omega}(\lambda y.\mathcal{W}_y(\langle\{{}^{\Omega}\!/_{\Phi}\}\mathcal{A}(t_2)/x\rangle\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(v))) \\ \longrightarrow_{\mathsf{CAbs}} & \lambda y.\mathcal{C}_{\Phi}^{\Upsilon\,|\,\Omega}(\mathcal{W}_y(\langle\{{}^{\Omega}\!/_{\Phi}\}\mathcal{A}(t_2)/x\rangle\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(v))) \\ \longrightarrow^{*}_{\mathsf{Cross}} & \lambda y.\mathcal{W}_y(\mathcal{C}_{\Phi}^{\Upsilon\,|\,\Omega}(\langle\{{}^{\Omega}\!/_{\Phi}\}\mathcal{A}(t_2)/x\rangle\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(v))) \end{array}$$

   and we get the result by the induction hypothesis.

3. If $t_1 = (t\ u)$, then by α-conversion we can suppose $x \notin \mathsf{FV}(t_2)$, and let
   $\Sigma := \mathsf{FV}(t_2) \cap \mathsf{FV}(t) \cap \mathsf{FV}(u)$
   $\Lambda := (\mathsf{FV}(t_2) \cap \mathsf{FV}(t)) \setminus (\mathsf{FV}(t_2) \cap \mathsf{FV}(t) \cap \mathsf{FV}(u))$
   $\Psi := (\mathsf{FV}(t_2) \cap \mathsf{FV}(u)) \setminus (\mathsf{FV}(t_2) \cap \mathsf{FV}(t) \cap \mathsf{FV}(u))$
   $\Xi := (\mathsf{FV}(t) \cap \mathsf{FV}(u)) \setminus (\mathsf{FV}(t_2) \cap \mathsf{FV}(t) \cap \mathsf{FV}(u))$
   $\Theta := \mathsf{FV}(t_2) \setminus (\mathsf{FV}(t) \cup \mathsf{FV}(u))$

   Note that $\Phi = \mathsf{FV}(t_1) \cap \mathsf{FV}(t_2)$ is a permutation of Σ, Λ, Ψ.
   Also note that $\mathsf{FV}(t) \cap \mathsf{FV}(u)$ is a permutation of Σ, Ξ and hence

$$\mathcal{A}(t_1) \equiv \mathcal{C}_{\Sigma,\Xi}^{\Sigma_3,\Xi_3\,|\,\Sigma_4,\Xi_4}(\{{}^{\Sigma_3,\Xi_3}\!/_{\Sigma,\Xi}\}\mathcal{A}(t)\ \ \{{}^{\Sigma_4,\Xi_4}\!/_{\Sigma,\Xi}\}\mathcal{A}(u))$$

   Hence the term h that we have to reduce is:

$$\begin{array}{rl} h := & \mathcal{C}_{\Sigma,\Lambda,\Psi}^{\Sigma_1,\Lambda_1,\Psi_1\,|\,\Sigma_2,\Lambda_2,\Psi_2}(\langle\{{}^{\Sigma_2,\Lambda_2,\Psi_2}\!/_{\Sigma,\Lambda,\Psi}\}\mathcal{A}(t_2)/x\rangle\{{}^{\Sigma_1,\Lambda_1,\Psi_1}\!/_{\Sigma,\Lambda,\Psi}\}\mathcal{A}(t_1)) \\ = & \mathcal{C}_{\Sigma,\Lambda,\Psi}^{\Sigma_1,\Lambda_1,\Psi_1\,|\,\Sigma_2,\Lambda_2,\Psi_2}(\langle\{{}^{\Sigma_2,\Lambda_2,\Psi_2}\!/_{\Sigma,\Lambda,\Psi}\}\mathcal{A}(t_2)/x\rangle\mathcal{C}_{\Sigma_1,\Xi}^{\Sigma_3,\Xi_3\,|\,\Sigma_4,\Xi_4}(t'\ u')) \end{array}$$

   where $t' = \{{}^{\Lambda_1,\Sigma_3,\Xi_3}\!/_{\Lambda,\Sigma,\Xi}\}\mathcal{A}(t)$ and $u' = \{{}^{\Psi_1,\Sigma_4,\Xi_4}\!/_{\Psi,\Sigma,\Xi}\}\mathcal{A}(u)$.

   (a) If $x \in \mathsf{FV}(t) \cap \mathsf{FV}(u)$, then x is necessarily in Ξ (since $x \notin \mathsf{FV}(t_2)$), so Ξ is a permutation of Ξ', x for some list Ξ'. Hence, the contractions $\mathcal{C}_{\Sigma_1,\Xi}^{\Sigma_3,\Xi_3\,|\,\Sigma_4,\Xi_4}(\_)$ are equivalent by $\equiv$ to $\mathcal{C}_{\Sigma_1,\Xi'}^{\Sigma_3,\Xi'_3\,|\,\Sigma_4,\Xi'_4}(\mathcal{C}_x^{x_3\,|\,x_4}(\_))$ (where

Ξ'_3, x_3 and Ξ'_4, x_4 are the corresponding permutations of Ξ_3 and Ξ_4, respectively). Hence:

$$
\begin{array}{rcl}
h & \equiv & \mathcal{C}^{\Sigma_1,\Lambda_1,\Psi_1\,|\,\Sigma_2,\Lambda_2,\Psi_2}_{\Sigma,\Lambda,\Psi}(\mathcal{C}^{\Sigma_3,\Xi'_3\,|\,\Sigma_4,\Xi'_4}_{\Sigma_1,\Xi'}(\langle v/x\rangle \mathcal{C}^{x_3\,|\,x_4}_{x}(t'\ u'))) \\
 & & \qquad\qquad \text{where } v := \left\{^{\Sigma_2,\Lambda_2,\Psi_2}\!\!\diagup_{\Sigma,\Lambda,\Psi}\right\}\mathcal{A}(t_2) \\
\longrightarrow_{\mathsf{Cont}} & & \mathcal{C}^{\Sigma_1,\Lambda_1,\Psi_1\,|\,\Sigma_2,\Lambda_2,\Psi_2}_{\Sigma,\Lambda,\Psi}(\mathcal{C}^{\Sigma_3,\Xi'_3\,|\,\Sigma_4,\Xi'_4}_{\Sigma_1,\Xi'}(\mathcal{C}^{\Theta_5,\Sigma_5,\Lambda_5,\Psi_5\,|\,\Theta_6,\Sigma_6,\Lambda_6,\Psi_6}_{\Theta,\Sigma_2,\Lambda_2,\Psi_2}(v_1))) \\
 & & \qquad\qquad \text{where } v_1 := \langle v''/x_4\rangle\langle v'/x_3\rangle(t'\ u') \\
 & & \qquad\qquad \text{with } v' := \left\{^{\Theta_5,\Sigma_5,\Lambda_5,\Psi_5}\!\!\diagup_{\Theta,\Sigma,\Lambda,\Psi}\right\}\mathcal{A}(t_2) \\
 & & \qquad\qquad \text{and } v'' := \left\{^{\Theta_6,\Sigma_6,\Lambda_6,\Psi_6}\!\!\diagup_{\Theta,\Sigma,\Lambda,\Psi}\right\}\mathcal{A}(t_2) \\
 & \equiv & \mathcal{C}^{\Theta_5\,|\,\Theta_6}_{\Theta}(\mathcal{C}^{\Xi'_3\,|\,\Xi'_4}_{\Xi'}(\mathcal{C}^{\Lambda_1\,|\,\Lambda_2}_{\Lambda}(\mathcal{C}^{\Lambda_5\,|\,\Lambda_6}_{\Lambda_2}(\mathcal{C}^{\Psi_1\,|\,\Psi_2}_{\Psi}(\mathcal{C}^{\Psi_5\,|\,\Psi_6}_{\Psi_2}(v_2)))))) \\
 & & \qquad\qquad \text{where } v_2 := \mathcal{C}^{\Sigma_1\,|\,\Sigma_2}_{\Sigma}(\mathcal{C}^{\Sigma_3\,|\,\Sigma_4}_{\Sigma_1}(\mathcal{C}^{\Sigma_5\,|\,\Sigma_6}_{\Sigma_2}(v_1))) \\
 & \equiv & \mathcal{C}^{\Theta_5\,|\,\Theta_6}_{\Theta}(\mathcal{C}^{\Xi'_3\,|\,\Xi'_4}_{\Xi'}(\mathcal{C}^{\Lambda_2\,|\,\Lambda_6}_{\Lambda}(\mathcal{C}^{\Lambda_1\,|\,\Lambda_5}_{\Lambda_2}(\mathcal{C}^{\Psi_5\,|\,\Psi_2}_{\Psi}(\mathcal{C}^{\Psi_1\,|\,\Psi_6}_{\Psi_2}(v'_2)))))) \\
 & & \qquad\qquad \text{where } v'_2 := \mathcal{C}^{\Sigma_1\,|\,\Sigma_2}_{\Sigma}(\mathcal{C}^{\Sigma_3\,|\,\Sigma_5}_{\Sigma_1}(\mathcal{C}^{\Sigma_4\,|\,\Sigma_6}_{\Sigma_2}(v_1))) \\
 & \equiv & \mathcal{C}^{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1\,|\,\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}_{\Theta,\Xi',\Lambda,\Psi,\Sigma}(\mathcal{C}^{\Lambda_1,\Sigma_3\,|\,\Lambda_5,\Sigma_5}_{\Lambda_2,\Sigma_1}(\mathcal{C}^{\Psi_1,\Sigma_4\,|\,\Psi_6,\Sigma_6}_{\Psi_2,\Sigma_2}(v_1))) \\
\longrightarrow^2_{\mathsf{xr}} & & \mathcal{C}^{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1\,|\,\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}_{\Theta,\Xi',\Lambda,\Psi,\Sigma}(\mathcal{C}^{\Lambda_1,\Sigma_3\,|\,\Lambda_5,\Sigma_5}_{\Lambda_2,\Sigma_1}(\mathcal{C}^{\Psi_1,\Sigma_4\,|\,\Psi_6,\Sigma_6}_{\Psi_2,\Sigma_2}(p\ q))) \\
 & & \qquad\qquad \text{with } p := \langle t'_2/x_3\rangle t' \text{ and } q := \langle t''_2/x_4\rangle u' \\
\longrightarrow_{\mathsf{CApp2}} & & \mathcal{C}^{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1\,|\,\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}_{\Theta,\Xi',\Lambda,\Psi,\Sigma}(\mathcal{C}^{\Lambda_1,\Sigma_3\,|\,\Lambda_5,\Sigma_5}_{\Lambda_2,\Sigma_1}(p\ \mathcal{C}^{\Psi_1,\Sigma_4\,|\,\Psi_6,\Sigma_6}_{\Psi_2,\Sigma_2}(q))) \\
\longrightarrow_{\mathsf{CApp1}} & & \mathcal{C}^{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1\,|\,\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}_{\Theta,\Xi',\Lambda,\Psi,\Sigma}(\mathcal{C}^{\Lambda_1,\Sigma_3\,|\,\Lambda_5,\Sigma_5}_{\Lambda_2,\Sigma_1}(p)\ \mathcal{C}^{\Psi_1,\Sigma_4\,|\,\Psi_6,\Sigma_6}_{\Psi_2,\Sigma_2}(q))
\end{array}
$$

The Cont-reduction step is justified by noticing that $\mathsf{FV}(t_2)$ is a permutation of $\Theta, \Sigma, \Lambda, \Psi$, and the reduction sequence $v_1 \longrightarrow^2_{\mathsf{xr}} p\ q$ is:

$$
\begin{array}{rcl}
v_1 & = & \langle t''_2/x_4\rangle\langle t'_2/x_3\rangle(t'\ u') \\
 & \longrightarrow_{\mathsf{App1}} & \langle t''_2/x_4\rangle(\langle t'_2/x_3\rangle t'\ u') \\
 & \longrightarrow_{\mathsf{App2}} & \langle t'_2/x_3\rangle t'\ \langle t''_2/x_4\rangle u'
\end{array}
$$

Then note that

$$\mathcal{C}^{\Lambda_1,\Sigma_3\,|\,\Lambda_5,\Sigma_5}_{\Lambda_2,\Sigma_1}(p) = \left\{{}^{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1}\!/_{\Theta,\Xi',\Lambda,\Psi,\Sigma}\right\} p'$$
$$\mathcal{C}^{\Psi_1,\Sigma_4\,|\,\Psi_6,\Sigma_6}_{\Psi_2,\Sigma_2}(q) = \left\{{}^{\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}\!/_{\Theta,\Xi',\Lambda,\Psi,\Sigma}\right\} q'$$

$$\text{with } p' := \mathcal{C}^{\Lambda_1,\Sigma_3\,|\,\Lambda_5,\Sigma_5}_{\Lambda,\Sigma}(\langle\left\{{}^{\Sigma_5,\Lambda_5}\!/_{\Sigma,\Lambda}\right\}\mathcal{A}(t_2)/x_3\rangle\left\{{}^{\Lambda_1,\Sigma_3}\!/_{\Lambda,\Sigma}\right\}\{{}^{x_3}\!/_{x}\}\mathcal{A}(t))$$
$$\text{and } q' := \mathcal{C}^{\Psi_1,\Sigma_4\,|\,\Psi_6,\Sigma_6}_{\Psi,\Sigma}(\langle\left\{{}^{\Sigma_6,\Psi_6}\!/_{\Sigma,\Psi}\right\}\mathcal{A}(t_2)/x_4\rangle\left\{{}^{\Psi_1,\Sigma_4}\!/_{\Psi,\Sigma}\right\}\{{}^{x_4}\!/_{x}\}\mathcal{A}(u))$$

We can now apply the induction hypothesis to both p' and q' and we get:

$$p' \longrightarrow^*_{\mathsf{xr}} \mathcal{A}(\{{}^{t_2}\!/_{x}\}t)$$
$$q' \longrightarrow^*_{\mathsf{xr}} \mathcal{A}(\{{}^{t_2}\!/_{x}\}u)$$

Hence,

$$\mathcal{C}^{\Lambda_1,\Sigma_3\,|\,\Lambda_5,\Sigma_5}_{\Lambda_2,\Sigma_1}(p) \longrightarrow^*_{\mathsf{xr}} p'' := \left\{{}^{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1}\!/_{\Theta,\Xi',\Lambda,\Psi,\Sigma}\right\}\mathcal{A}(\{{}^{t_2}\!/_{x}\}t)$$
$$\mathcal{C}^{\Psi_1,\Sigma_4\,|\,\Psi_6,\Sigma_6}_{\Psi_2,\Sigma_2}(q) \longrightarrow^*_{\mathsf{xr}} q'' := \left\{{}^{\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}\!/_{\Theta,\Xi',\Lambda,\Psi,\Sigma}\right\}\mathcal{A}(\{{}^{t_2}\!/_{x}\}u)$$

And h finally reduces to

$$\mathcal{C}^{\Theta_5,\Xi'_3,\Lambda_2,\Psi_5,\Sigma_1\,|\,\Theta_6,\Xi'_4,\Lambda_6,\Psi_2,\Sigma_2}_{\Theta,\Xi',\Lambda,\Psi,\Sigma}(p''\ q'')$$

which is $\mathcal{A}(\{{}^{t_2}\!/_{x}\}t\ \ \{{}^{t_2}\!/_{x}\}u) = \mathcal{A}(\{{}^{t_2}\!/_{x}\}(t\ u))$.

(b) If $x \in \mathsf{FV}(t)$ et $x \notin \mathsf{FV}(u)$, the term h can be transformed by $\mathsf{P_{cs}}$ to:

$$\begin{array}{ll} & \mathcal{C}^{\Sigma_1,\Lambda_1,\Psi_1\,|\,\Sigma_2,\Lambda_2,\Psi_2}_{\Sigma,\Lambda,\Psi}(\mathcal{C}^{\Sigma_3,\Xi_3\,|\,\Sigma_4,\Xi_4}_{\Sigma_1,\Xi}(\langle\left\{{}^{\Sigma_2,\Lambda_2,\Psi_2}\!/_{\Sigma,\Lambda,\Psi}\right\}\mathcal{A}(t_2)/x\rangle(t'\ u'))) \\ \longrightarrow_{\mathsf{App1}} & \mathcal{C}^{\Sigma_1,\Lambda_1,\Psi_1\,|\,\Sigma_2,\Lambda_2,\Psi_2}_{\Sigma,\Lambda,\Psi}(\mathcal{C}^{\Sigma_3,\Xi_3\,|\,\Sigma_4,\Xi_4}_{\Sigma_1,\Xi}(\langle\left\{{}^{\Sigma_2,\Lambda_2,\Psi_2}\!/_{\Sigma,\Lambda,\Psi}\right\}\mathcal{A}(t_2)/x\rangle t'\ u')) \\ \equiv & \mathcal{C}^{\Sigma_1,\Psi_2,\Xi_3\,|\,\Sigma_4,\Psi_1,\Xi_4}_{\Sigma,\Psi,\Xi}(\mathcal{C}^{\Sigma_3,\Lambda_1\,|\,\Sigma_2,\Lambda_2}_{\Sigma_1,\Lambda}(\langle\left\{{}^{\Sigma_2,\Lambda_2,\Psi_2}\!/_{\Sigma,\Lambda,\Psi}\right\}\mathcal{A}(t_2)/x\rangle t'\ u')) \\ \longrightarrow_{\mathsf{CApp1}} & \mathcal{C}^{\Sigma_1,\Psi_2,\Xi_3\,|\,\Sigma_4,\Psi_1,\Xi_4}_{\Sigma,\Psi,\Xi}(\mathcal{C}^{\Sigma_3,\Lambda_1\,|\,\Sigma_2,\Lambda_2}_{\Sigma_1,\Lambda}(\langle\left\{{}^{\Sigma_2,\Lambda_2,\Psi_2}\!/_{\Sigma,\Lambda,\Psi}\right\}\mathcal{A}(t_2)/x\rangle t')\ u') \\ = & \mathcal{C}^{\Sigma_1,\Psi_2,\Xi_3\,|\,\Sigma_4,\Psi_1,\Xi_4}_{\Sigma,\Psi,\Xi}(\left\{{}^{\Sigma_1,\Psi_2,\Xi_3}\!/_{\Sigma,\Psi,\Xi}\right\}v\ \ \left\{{}^{\Sigma_4,\Psi_1,\Xi_4}\!/_{\Sigma,\Psi,\Xi}\right\}u) \end{array}$$

where $v := \mathcal{C}^{\Sigma_3,\Lambda_1\,|\,\Sigma_2,\Lambda_2}_{\Sigma,\Lambda}(\langle\left\{{}^{\Sigma_2,\Lambda_2}\!/_{\Sigma,\Lambda}\right\}\mathcal{A}(t_2)/x\rangle\left\{{}^{\Lambda_1,\Sigma_3}\!/_{\Lambda,\Sigma}\right\}\mathcal{A}(t))$, which reduces, by induction hypothesis, to $\mathcal{A}(\{{}^{t_2}\!/_{x}\}t)$. Hence,

$$h \longrightarrow^*_{\mathsf{xr}} \mathcal{C}^{\Sigma_1,\Psi_2,\Xi_3\,|\,\Sigma_4,\Psi_1,\Xi_4}_{\Sigma,\Psi,\Xi}(\left\{{}^{\Sigma_1,\Psi_2,\Xi_3}\!/_{\Sigma,\Psi,\Xi}\right\}\mathcal{A}(\left\{{}^{t_2}\!/_{x}\right\}t)\ \ \left\{{}^{\Sigma_4,\Psi_1,\Xi_4}\!/_{\Sigma,\Psi,\Xi}\right\}u)$$

which is exactly $\mathcal{A}(\{{}^{t_2}\!/_{x}\}t\ \ u) = \mathcal{A}(\{{}^{t_2}\!/_{x}\}(t\ u))$.

(c) If $x \in \mathsf{FV}(t)$ et $x \notin \mathsf{FV}(u)$ the proof is exactly the same.

(d) The case $x \notin \mathsf{FV}(t)$ and $x \notin \mathsf{FV}(u)$ cannot happen since we assumed $x \in \mathsf{FV}(t_1)$.

□

The *correctness* result concerning substitution constructors obtained in the previous lemma enables us to prove a more general property concerning simulation of β-reduction in $\lambda\mathsf{lxr}$. Notice that a β-reduction step may not preserve the set of free variables whereas any reduction in $\lambda\mathsf{lxr}$ does. Indeed, we have $t = (\lambda x.y)\ z \longrightarrow_\beta\ y$, but

$$\mathcal{A}(t) = (\lambda x.\mathcal{W}_x(y))\ z \longrightarrow^*_{\lambda\mathsf{lxr}} \mathcal{W}_z(y) = \mathcal{W}_z(\mathcal{A}(y))$$

As a consequence, the simulation property has to be stated by taking into account the operational behaviour of system xr given by Lemma 120.

Theorem 121 (Simulating β-reduction)
Let t be a λ-term such that $t \longrightarrow_\beta\ t'$. Then $\mathcal{A}(t) \longrightarrow^+_{\lambda lxr} \mathcal{W}_{\mathsf{FV}(t)\setminus\mathsf{FV}(t')}(\mathcal{A}(t'))$.

Proof: We prove this by induction on the derivation of reduction step. We only show here the root reduction cases.

1. The root case is the reduction $(\lambda x.t_1)t_2 \longrightarrow_\beta\ \{^{t_2}/_x\}t_1$. By α-conversion, $x \notin \mathsf{FV}(t_2)$.

   (a) If $x \notin \mathsf{FV}(t_1)$, let $\Phi := \mathsf{FV}(t_1) \cap \mathsf{FV}(t_2)$ and $\Xi := \mathsf{FV}(t_2) \setminus \mathsf{FV}(t_1)$.

$$\begin{array}{lll}
\mathcal{A}((\lambda x.t_1)t_2) & = & \mathcal{C}^{\Upsilon\,|\,\Omega}_{\Phi}(\lambda x.\mathcal{W}_x(\{^{\Upsilon}/_{\Phi}\}\mathcal{A}(t_1))\ \{^{\Omega}/_{\Phi}\}\mathcal{A}(t_2)) \\
& \longrightarrow_B & \mathcal{C}^{\Upsilon\,|\,\Omega}_{\Phi}(\langle\{^{\Omega}/_{\Phi}\}\mathcal{A}(t_2)/x\rangle\mathcal{W}_x(\{^{\Upsilon}/_{\Phi}\}\mathcal{A}(t_1))) \\
& \longrightarrow_{\mathsf{Weak1}} & \mathcal{C}^{\Upsilon\,|\,\Omega}_{\Phi}(\mathcal{W}_{\mathsf{FV}(\{^{\Omega}/_{\Phi}\}t_2)}(\{^{\Upsilon}/_{\Phi}\}\mathcal{A}(t_1))) \\
& \equiv & \mathcal{C}^{\Upsilon\,|\,\Omega}_{\Phi}(\mathcal{W}_{\Omega,\Xi}(\{^{\Upsilon}/_{\Phi}\}\mathcal{A}(t_1))) \\
& = & \mathcal{C}^{\Upsilon\,|\,\Omega}_{\Phi}(\mathcal{W}_{\Omega}(\mathcal{W}_{\Xi}(\{^{\Upsilon}/_{\Phi}\}\mathcal{A}(t_1)))) \\
& \longrightarrow^*_{\mathsf{Merge}} & \{^{\Phi}/_{\Upsilon}\}\mathcal{W}_{\Xi}(\{^{\Upsilon}/_{\Phi}\}\mathcal{A}(t_1)) \\
& = & \mathcal{W}_{\Xi}(\mathcal{A}(t_1))
\end{array}$$

   Now it suffices to notice that $\Xi := \mathsf{FV}((\lambda x.t_1)\ t_2) \setminus \mathsf{FV}(\{^{t_2}/_x\}t_1)$ using $\mathsf{FV}(\{^{t_2}/_x\}t_1) = \mathsf{FV}(t_1)$ since $x \notin \mathsf{FV}(t_1)$.

   (b) If $x \in \mathsf{FV}(t_1)$, let $\Phi := (\mathsf{FV}(t_1) \setminus \{x\}) \cap \mathsf{FV}(t_2)$.

$$\begin{array}{lll}
\mathcal{A}((\lambda x.t_1)t_2) & = & \mathcal{C}^{\Upsilon\,|\,\Omega}_{\Phi}(\lambda x.\{^{\Upsilon}/_{\Phi}\}\mathcal{A}(t_1)\ \{^{\Omega}/_{\Phi}\}\mathcal{A}(t_2)) \\
& \longrightarrow_B & \mathcal{C}^{\Upsilon\,|\,\Omega}_{\Phi}(\langle\mathcal{A}(\{^{\Omega}/_{\Phi}\}t_2)/x\rangle\mathcal{A}(\{^{\Upsilon}/_{\Phi}\}t_1)) \\
& \longrightarrow^*_{\mathsf{xr}} & \mathcal{A}(\{^{t_2}/_x\}t_1) \qquad \text{by Lemma 120}
\end{array}$$

2. Now suppose $\lambda x.u \longrightarrow_\beta\ \lambda x.u'$ with $u \longrightarrow_\beta\ u'$,

   (a) If $x \notin \mathsf{FV}(u)$

$$\begin{array}{lll}
\mathcal{A}(\lambda x.u) & = & \lambda x.\mathcal{W}_x(\mathcal{A}(u)) \\
& \longrightarrow^+_{\lambda\mathsf{lxr}} & \lambda x.\mathcal{W}_x(\mathcal{W}_{\mathsf{FV}(u)\setminus\mathsf{FV}(u')}(\mathcal{A}(u'))) \qquad \text{by the i.h.} \\
& = & \lambda x.\mathcal{W}_x(\mathcal{W}_{\mathsf{FV}(\lambda x.u)\setminus\mathsf{FV}(\lambda x.u')}(\mathcal{A}(u'))) \\
& \longrightarrow^*_{\mathsf{WAbs}} & \mathcal{W}_{\mathsf{FV}(\lambda x.u)\setminus\mathsf{FV}(\lambda x.u')}(\lambda x.\mathcal{W}_x(\mathcal{A}(u')))
\end{array}$$

(b) If $x \in \mathsf{FV}(u)$

$$\begin{array}{rcll}
\mathcal{A}(\lambda x.u) & = & \lambda x.\mathcal{A}(u) & \\
& \longrightarrow^{+}_{\lambda\mathsf{lxr}} & \lambda x.\mathcal{W}_{\mathsf{FV}(u)\setminus\mathsf{FV}(u')}(\mathcal{A}(u')) & \text{by the i.h.} \\
& = & \lambda x.\mathcal{W}_{\mathsf{FV}(\lambda x.u)\setminus\mathsf{FV}(u')}(\mathcal{W}_{\{x\}\setminus\mathsf{FV}(u')}(\mathcal{A}(u'))) & \\
& = & \lambda x.\mathcal{W}_{\mathsf{FV}(\lambda x.u)\setminus\mathsf{FV}(\lambda x.u')}(\mathcal{W}_{\{x\}\setminus\mathsf{FV}(u')}(\mathcal{A}(u'))) & \\
& \longrightarrow^{*}_{\mathsf{WAbs}} & \mathcal{W}_{\mathsf{FV}(\lambda x.u)\setminus\mathsf{FV}(\lambda x.u')}(\lambda x.\mathcal{W}_{\{x\}\setminus\mathsf{FV}(u')}(\mathcal{A}(u'))) &
\end{array}$$

3. Now suppose $t_1\ t_2 \longrightarrow_\beta\ t_1'\ t_2$ with $t_1 \longrightarrow_\beta\ t_1'$, let
$\Sigma := \mathsf{FV}(t_1') \cap \mathsf{FV}(t_2)$
$\Lambda := \mathsf{FV}(t_1') \setminus (\mathsf{FV}(t_1') \cap \mathsf{FV}(t_2))$
$\Psi := (\mathsf{FV}(t_1) \cap \mathsf{FV}(t_2)) \setminus \mathsf{FV}(t_1')$
$\Xi := \mathsf{FV}(t_1) \setminus (\mathsf{FV}(t_1') \cap \mathsf{FV}(t_2))$
Note in particular that $\mathsf{FV}(t_1) \cap \mathsf{FV}(t_2)$ is a permutation of Σ, Ψ. Correspondingly, let Σ_l, Ψ_l and Σ_r, Ψ_r be fresh variables.

We have:

$$\begin{array}{rcll}
\mathcal{A}(t_1\ t_2) \equiv & \mathcal{C}^{\Sigma_l,\Psi_l\,|\,\Sigma_r,\Psi_r}_{\Sigma,\Psi}(\left\{{}^{\Sigma_l,\Psi_l}\!/_{\Sigma,\Psi}\right\}\mathcal{A}(t_1)\ \left\{{}^{\Sigma_r,\Psi_r}\!/_{\Sigma,\Psi}\right\}\mathcal{A}(t_2)) & \\
\longrightarrow^{+}_{\lambda\mathsf{lxr}} & \mathcal{C}^{\Sigma_l,\Psi_l\,|\,\Sigma_r,\Psi_r}_{\Sigma,\Psi}(\left\{{}^{\Sigma_l,\Psi_l}\!/_{\Sigma,\Psi}\right\}\mathcal{W}_{\mathsf{FV}(t_1)\setminus\mathsf{FV}(t_1')}(\mathcal{A}(t_1'))\ \left\{{}^{\Sigma_r,\Psi_r}\!/_{\Sigma,\Psi}\right\}\mathcal{A}(t_2)) & \\
& & & \text{by the i.h.} \\
\equiv & \mathcal{C}^{\Sigma_l,\Psi_l\,|\,\Sigma_r,\Psi_r}_{\Sigma,\Psi}(\left\{{}^{\Sigma_l,\Psi_l}\!/_{\Sigma,\Psi}\right\}\mathcal{W}_{\Xi,\Psi}(\mathcal{A}(t_1'))\ \left\{{}^{\Sigma_r,\Psi_r}\!/_{\Sigma,\Psi}\right\}\mathcal{A}(t_2)) & \\
= & \mathcal{C}^{\Sigma_l,\Psi_l\,|\,\Sigma_r,\Psi_r}_{\Sigma,\Psi}(\mathcal{W}_{\Xi}(\mathcal{W}_{\Psi_l}(\left\{{}^{\Sigma_l}\!/_{\Sigma}\right\}\mathcal{A}(t_1')))\ \left\{{}^{\Sigma_r,\Psi_r}\!/_{\Sigma,\Psi}\right\}\mathcal{A}(t_2)) & \\
\longrightarrow^{*}_{\mathsf{WApp1}} & \mathcal{C}^{\Sigma_l,\Psi_l\,|\,\Sigma_r,\Psi_r}_{\Sigma,\Psi}(\mathcal{W}_{\Xi}(\mathcal{W}_{\Psi_l}(\left\{{}^{\Sigma_l}\!/_{\Sigma}\right\}\mathcal{A}(t_1')\ \left\{{}^{\Sigma_r,\Psi_r}\!/_{\Sigma,\Psi}\right\}\mathcal{A}(t_2)))) & \\
\longrightarrow^{*}_{\mathsf{Cross}} & \mathcal{W}_{\Xi}(\mathcal{C}^{\Sigma_l,\Psi_l\,|\,\Sigma_r,\Psi_r}_{\Sigma,\Psi}(\mathcal{W}_{\Psi_l}(\left\{{}^{\Sigma_l}\!/_{\Sigma}\right\}\mathcal{A}(t_1')\ \left\{{}^{\Sigma_r,\Psi_r}\!/_{\Sigma,\Psi}\right\}\mathcal{A}(t_2)))) & \\
\longrightarrow^{*}_{\mathsf{Merge}} & \mathcal{W}_{\Xi}(\mathcal{C}^{\Sigma_l\,|\,\Sigma_r}_{\Sigma}(\left\{{}^{\Psi}\!/_{\Psi_r}\right\}\left\{{}^{\Sigma_l}\!/_{\Sigma}\right\}\mathcal{A}(t_1')\ \left\{{}^{\Sigma_r,\Psi_r}\!/_{\Sigma,\Psi}\right\}\mathcal{A}(t_2))) & \\
= & \mathcal{W}_{\Xi}(\mathcal{C}^{\Sigma_l\,|\,\Sigma_r}_{\Sigma}(\left\{{}^{\Sigma_l}\!/_{\Sigma}\right\}\mathcal{A}(t_1')\ \left\{{}^{\Sigma_r}\!/_{\Sigma}\right\}\mathcal{A}(t_2))) &
\end{array}$$

Then it suffices to notice that $\Xi = \mathsf{FV}(t_1\ t_2) \setminus \mathsf{FV}(t_1'\ t_2)$.

4. The case $t_1\ t_2 \longrightarrow_\beta\ t_1\ t_2'$ is similar to the previous one.

□

As for the types, a straightforward induction on typing derivations allows us to show soundness of the translation $\mathcal{A}$:

Lemma 122 (Encoding $\mathcal{A}$ preserves types) *If t is a λ-term s.t. $\Gamma \vdash_\lambda t : A$, then $\Gamma \vdash_{\lambda lxr} \mathcal{W}_{\Gamma\setminus FV(t)}(\mathcal{A}(t)) : A$.*

5.2.2 From λlxr-calculus to λ-calculus

In this section we investigate an encoding $\mathcal{B}$ from λlxr to λ-calculus, with the (weak) simulation result (Theorem 121).

Definition 96 (Translation from λlxr to λ-calculus) We define the function $\mathcal{B}(t)$ by induction on the structure of the λlxr-t as shown in Fig. 5.11.

$$\begin{array}{lcl}
\mathcal{B}(x) & := & x \\
\mathcal{B}(\lambda x.t) & := & \lambda x.\mathcal{B}(t) \\
\mathcal{B}(\mathcal{W}_x(t)) & := & \mathcal{B}(t) \\
\mathcal{B}(\mathcal{C}_x^{y\,|\,z}(t)) & := & \{{}^{x}\!/_{z}\}\{{}^{x}\!/_{y}\}\mathcal{B}(t) \\
\mathcal{B}(t\ u) & := & \mathcal{B}(t)\ \mathcal{B}(u) \\
\mathcal{B}(\langle u/x\rangle t) & := & \{{}^{\mathcal{B}(u)}\!/_{x}\}\mathcal{B}(t)
\end{array}$$

Figure 5.11: From λlxr to λ-calculus

Remark that $\mathcal{B}(t)$ is *not* the xr-normal form of t since weakenings and contractions disappear and thus the linearity constraints need not hold anymore.

Lemma 123 (Properties of $\mathcal{B}$) *The translation $\mathcal{B}$ satisfies the following properties.*

- $\mathcal{B}(\{{}^{\Upsilon}\!/_{\Phi}\}t) = \{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{B}(t)$
- $FV(\mathcal{B}(t)) \subseteq FV(t)$

The following result will allow us to project the λlxr-calculus onto the λ-calculus, as usually done for calculi with explicit substitutions [Ros96].

Lemma 124 (Simulating λlxr-reduction)

1. *If $t_1 \equiv t_2$, then $\mathcal{B}(t_1) = \mathcal{B}(t_2)$.*
2. *If $t_1 \longrightarrow_B t_2$, then $\mathcal{B}(t_1) \longrightarrow_\beta^* \mathcal{B}(t_2)$.*
3. *If $t_1 \longrightarrow_{xr} t_2$, then $\mathcal{B}(t_1) = \mathcal{B}(t_2)$.*

Proof:

1. This is obvious for the equivalence rule $\mathsf{P_w}$. For the other ones we have to use the substitution lemma (Lemma 40).
2. A B-reduction step at the root of t_1 corresponds exactly to a β-reduction step at the root of $\mathcal{B}(t_1)$. For the contextual closure, all cases are trivial except for:

- the contraction, for which we use the fact that if $\mathcal{B}(t) \longrightarrow_\beta^* \mathcal{B}(t')$ then $\{{}^{x}\!/_{z}\}\{{}^{x}\!/_{y}\}\mathcal{B}(t) \longrightarrow_\beta^* \{{}^{x}\!/_{z}\}\{{}^{x}\!/_{y}\}\mathcal{B}(t')$.
- the substitution constructor, for which we use the two following facts: If $\mathcal{B}(t) \longrightarrow_\beta^* \mathcal{B}(t')$ then $\left\{{}^{\mathcal{B}(u)}\!/_{x}\right\}\mathcal{B}(t) \longrightarrow_\beta^* \left\{{}^{\mathcal{B}(u)}\!/_{x}\right\}\mathcal{B}(t')$.
  If $\mathcal{B}(u) \longrightarrow_\beta^* \mathcal{B}(u')$ then $\left\{{}^{\mathcal{B}(u)}\!/_{x}\right\}\mathcal{B}(t) \longrightarrow_\beta^* \left\{{}^{\mathcal{B}(u')}\!/_{x}\right\}\mathcal{B}(t)$.

3. We only discuss the cases where the reduction takes place at the root, all the other ones being trivial.

   - If the rule applied is WAbs, WApp1, WApp2, WSubs, Cross, Weak2, then the property is trivial.
   - If the rule applied is Abs, App1, App2, Var, CAbs, CApp1, CApp2, then the property follows from the definition of substitution.
   - If the rule applied is Comp, then x is not free in t since the left-hand side is linear, so by Remark 123 x is neither free in $\mathcal{B}(t)$. It suffices to use the substitution lemma as before.
   - If the rule applied is Weak1, then x is not free in t since the left-hand side is linear, so by Remark 123 x is neither free in $\mathcal{B}(t)$. Hence, we get on the left-hand side $\left\{{}^{\mathcal{B}(u)}\!/_{x}\right\}\mathcal{B}(t)$ which is exactly $\mathcal{B}(t)$.
   - If the rule applied is Merge, then, as before, y is not free in $\mathcal{B}(t)$ so that it suffices to notice that $\{{}^{w}\!/_{z}\}\mathcal{B}(t) = \mathcal{B}(\{{}^{w}\!/_{z}\}t)$ by Remark 123.
   - If the rule applies is CSubs, then it is sufficient to apply the substitution lemma of λ-calculus.
   - If the rule applied is Cont, then, as before, x is not free in $\mathcal{B}(t)$ so that $\mathcal{B}(t_1) = \left\{{}^{\mathcal{B}(u)}\!/_{z}\right\}\left\{{}^{\mathcal{B}(u)}\!/_{y}\right\}\mathcal{B}(t)$ by the substitution lemma. For the right-hand side we have

     $$\mathcal{B}(t_2) = \left\{{}^{\Phi}\!/_{\Xi}\right\}\left\{{}^{\Phi}\!/_{\Psi}\right\}\left\{{}^{\mathcal{B}(\{{}^{\Xi}\!/_{\Phi}\}u)}\!/_{z}\right\}\left\{{}^{\mathcal{B}(\{{}^{\Psi}\!/_{\Phi}\}u)}\!/_{y}\right\}\mathcal{B}(t)$$

     which, using Remark 123, is equal to

     $$\left\{{}^{\Phi}\!/_{\Xi}\right\}\left\{{}^{\Phi}\!/_{\Psi}\right\}\left\{{}^{\{{}^{\Xi}\!/_{\Phi}\}\mathcal{B}(u)}\!/_{z}\right\}\left\{{}^{\{{}^{\Psi}\!/_{\Phi}\}\mathcal{B}(u)}\!/_{y}\right\}\mathcal{B}(t)$$

     which is equal to the left-hand side.

□

Corollary 125 *If* $t_1 \longrightarrow_{\lambda lxr} t_2$, *then* $\mathcal{B}(t_1) \longrightarrow_\beta^* \mathcal{B}(t_2)$.

A straightforward induction on typing derivations allows us to show:

Lemma 126 ($\mathcal{B}$ preserves types) *If* t *is a* λlxr*-term such that* $\Gamma \vdash_{\lambda lxr} t : A$, *then* $\Gamma \vdash_\lambda \mathcal{B}(t) : A$.

5.2.3 Reflection & confluence

We now proceed to prove the remaining conditions for $\mathcal{B}$ and $\mathcal{A}$ to form a reflection in $\lambda\mathsf{lxr}$ of λ-calculus, namely, we look at their compositions.

In one direction, the composition is easy:

Remark 127 (Composition 1) Since congruent terms are mapped to the same λ-term, it makes sense to consider $\mathcal{B} \circ \mathcal{A}$, which is in fact the identity: $t = \mathcal{B}(\mathcal{A}(t))$ (straightforward induction on t).

For the other direction we shall get $t \longrightarrow^*_{\mathsf{xr}} \mathcal{W}_{\mathsf{FV}(t)\setminus\mathsf{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$, which is a xr-normal form. We start with a result that relates xr-normal forms to the composition of encodings $\mathcal{A}$ and $\mathcal{B}$.

Theorem 128 (Composition 2)
If t is an xr-normal form, then $t \equiv \mathcal{W}_{FV(t)\setminus FV(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$.

Proof: The proof may proceed by induction since a sub-term of an xr-normal form is an xr-normal form:

- If $t = x$, then $x = \mathcal{A}(\mathcal{B}(x))$ and $\mathsf{FV}(t) \setminus \mathsf{FV}(\mathcal{B}(t)) = \emptyset$
- If $t = \lambda x.u$, then we know $u \equiv \mathcal{W}_{\mathsf{FV}(u)\setminus\mathsf{FV}(\mathcal{B}(u))}(\mathcal{A}(\mathcal{B}(u)))$ by the i.h. But t is an xr-normal form, so $\mathsf{FV}(u) \setminus \mathsf{FV}(\mathcal{B}(u)) \subseteq \{x\}$, otherwise it can be reduced by WAbs. Now, if $\mathsf{FV}(u) \setminus \mathsf{FV}(\mathcal{B}(u)) = \emptyset$, then also $\mathsf{FV}(t) \setminus \mathsf{FV}(\mathcal{B}(t)) = \emptyset$ and the claim $t \equiv \mathcal{A}(\mathcal{B}(\lambda x.u))$ immediately holds. Otherwise, $\mathsf{FV}(u) \setminus \mathsf{FV}(\mathcal{B}(u)) = \{x\}$ and $t \equiv \lambda x.\mathcal{W}_x(\mathcal{A}(\mathcal{B}(u))) = \mathcal{A}(\mathcal{B}(t))$.
- If $t = u\ v$, $t \equiv \mathcal{W}_{\mathsf{FV}(u)\setminus\mathsf{FV}(\mathcal{B}(u))}(\mathcal{A}(\mathcal{B}(u)))\ \mathcal{W}_{\mathsf{FV}(v)\setminus\mathsf{FV}(\mathcal{B}(v))}(\mathcal{A}(\mathcal{B}(v)))$ by the i.h. But t is a xr-normal form, so

  $$\mathsf{FV}(u) \setminus \mathsf{FV}(\mathcal{B}(u)) = \mathsf{FV}(v) \setminus \mathsf{FV}(\mathcal{B}(v)) = \emptyset$$

  (otherwise it could be reduced by $\mathsf{WApp1}$ or $\mathsf{WApp1}$). Hence, $\mathsf{FV}(t) = \mathsf{FV}(\mathcal{B}(t))$ and $t \equiv \mathcal{A}(\mathcal{B}(u))\ \mathcal{A}(\mathcal{B}(v)) \equiv \mathcal{A}(\mathcal{B}(t))$ since u and v have no variable in common.
- The case $t = \langle v/x\rangle u$ is not possible by Lemma 111.
- If $t = \mathcal{W}_x(u)$, $t \equiv \mathcal{W}_x(\mathcal{W}_{\mathsf{FV}(u)\setminus\mathsf{FV}(\mathcal{B}(u))}(\mathcal{A}(\mathcal{B}(u))))$ by the i.h. This last term is equal to $\mathcal{W}_{\mathsf{FV}(t)\setminus\mathsf{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$ since $x \in \mathsf{FV}(t)$ but $x \notin \mathsf{FV}(\mathcal{B}(t))$.
- If $t = \mathcal{C}_x^{y\,|\,z}(u)$, $t \equiv \mathcal{C}_x^{y\,|\,z}(\mathcal{W}_{\mathsf{FV}(u)\setminus\mathsf{FV}(\mathcal{B}(u))}(\mathcal{A}(\mathcal{B}(u))))$ by the i.h.

  First, we remark that y and z are free in u since t is linear, and also x is not free in u, hence neither is it free in $\mathcal{B}(u)$.

Second, since t is an xr-normal form, we have $\mathsf{FV}(u) \setminus \mathsf{FV}(\mathcal{B}(u)) = \emptyset$ (otherwise t could be reduced by Cross or Merge). Hence, y and z are free in $\mathcal{B}(u)$ and $t \equiv \mathcal{C}_x^{y\,|\,z}(\mathcal{A}(\mathcal{B}(u)))$.
But $\mathcal{B}(t) = \{{}^{x}\!/_{z}\}\{{}^{x}\!/_{y}\}\mathcal{B}(u)$, so x is free in $\mathcal{B}(t)$. We conclude $\mathsf{FV}(t) = \mathsf{FV}(\mathcal{B}(t))$.

Third, notice that $\mathcal{B}(u)$ can be neither a variable (otherwise t would not be linear) nor an abstraction (otherwise t could be reduced by CAbs), so $\mathcal{B}(u) = w\ v$,

and $\mathcal{A}(\mathcal{B}(u)) = \mathcal{C}_\Phi^{\Upsilon\,|\,\Psi}(\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(w)\ \ \{{}^{\Psi}\!/_{\Phi}\}\mathcal{A}(v))$ with $\Phi = \mathsf{FV}(w) \cap \mathsf{FV}(v)$.
Hence, $t \equiv \mathcal{C}_x^{y\,|\,z}(\mathcal{C}_\Phi^{\Upsilon\,|\,\Psi}(\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(w)\ \ \{{}^{\Psi}\!/_{\Phi}\}\mathcal{A}(v)))$.

Now it would suffice that $y \in \mathsf{FV}(w) \setminus \mathsf{FV}(v)$ and $z \in \mathsf{FV}(v) \setminus \mathsf{FV}(w)$ to prove that this term is in fact

$$\mathcal{A}(\{{}^{x}\!/_{y}\}w\ \ \{{}^{x}\!/_{z}\}v) = \mathcal{A}(\{{}^{x}\!/_{z}\}\{{}^{x}\!/_{y}\}\mathcal{B}(u)) = \mathcal{A}(\mathcal{B}(t))$$

We are going to prove that this is the case (or the symmetrical case when y and z are swapped): we know that they are free in $w\ v$.

Suppose that one of them, say y, is both in w and in v. Then $y \in \Phi$, so

$$t \equiv \mathcal{C}_x^{y\,|\,z}(\mathcal{C}_{\Phi',y}^{(\Upsilon',y')\,|\,(\Psi',y'')}(\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(w)\ \ \{{}^{\Psi}\!/_{\Phi}\}\mathcal{A}(v)))$$

which we can rearrange into

$$t \equiv \mathcal{C}_x^{y\,|\,y''}(\mathcal{C}_{\Phi',y}^{(\Upsilon',y')\,|\,(\Psi',z)}(\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(w)\ \ \{{}^{\Psi}\!/_{\Phi}\}\mathcal{A}(v)))$$

if $z \in \mathsf{FV}(w)$, or $t \equiv \mathcal{C}_x^{y\,|\,y'}(\mathcal{C}_{\Phi',y}^{(\Upsilon',z)\,|\,(\Psi',y'')}(\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(w)\ \ \{{}^{\Psi}\!/_{\Phi}\}\mathcal{A}(v)))$ if $z \in \mathsf{FV}(v)$.

In the first case, t can be reduced by $\mathsf{CApp1}$ (on $\mathcal{C}_y^{y'\,|\,z}(\_)$), and in the second by $\mathsf{CApp2}$ (on $\mathcal{C}_y^{z\,|\,y''}(\_)$). In both cases, it contradicts the fact that t is a xr-normal form. Hence, $y \notin \Phi$ (and similarly $z \notin \Phi$).

Now suppose that both y and z are on the same side, say in w. Then t can be reduced by $\mathsf{CApp1}$ on $\mathcal{C}_x^{y\,|\,z}(\_)$. Similarly, they cannot be both in v (t could be reduced by $\mathsf{CApp2}$). Hence one of them is only in w, and the other is only in v, as required.

□

Lemma 129 *The system* xr *is confluent (and we already know that it is terminating), and the* xr*-normal form of* t *is* $\mathcal{W}_{\mathsf{FV}(t)\setminus\mathsf{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$.

Proof: By Theorem 116 the system xr is terminating so we can take any xr-normal form t' of t such that $t \longrightarrow^*_{\mathsf{xr}} t'$. We then have $\mathsf{FV}(t) = \mathsf{FV}(t')$ by Lemma 1 and $\mathcal{B}(t) = \mathcal{B}(t')$ by Lemma 124. Since t' is an xr-normal form, $t' \equiv \mathcal{W}_{\mathsf{FV}(t')\setminus\mathsf{FV}(\mathcal{B}(t'))}(\mathcal{A}(\mathcal{B}(t')))$ by Theorem 128, so $t' \equiv \mathcal{W}_{\mathsf{FV}(t)\setminus\mathsf{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$.

To show confluence let us suppose $t \longrightarrow^*_{\mathsf{xr}} t_1$ and $t \longrightarrow^*_{\mathsf{xr}} t_2$. Let us take xr-normal forms t'_1 and t'_2 such that $t_i \longrightarrow^*_{\mathsf{xr}} t'_i$. By the previous remark both t'_1 and t'_2 are congruent to $\mathcal{W}_{\mathsf{FV}(t)\setminus\mathsf{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$ which concludes the proof. □

We now establish the reflection. Unfortunately, the shape of the simulation of β-reduction is not *exactly* the standard one for a simulation, owing to the weakenings placed at the top-level that record the free variables that were lost in the β-reduction. Hence, we use a trick given by O'Conchúir [O'C06] that consists in generalising the encoding $\mathcal{A}$ with a parameter as follows:

Definition 97 (Generalised $\mathcal{A}$) For all finite set $\mathcal{S}$ of variables and all λ-term t such that $\mathsf{FV}(t) \subseteq \mathcal{S}$,

$$\mathcal{A}_{\mathcal{S}}(t) := \mathcal{W}_{\mathcal{S}\setminus\mathsf{FV}(t)}(\mathcal{A}(t))$$

Remark 130 Note that $\mathcal{A}_{\mathsf{FV}(t)}(t) = \mathcal{A}(t)$.

Theorem 131 (Reflection in λlxr of λ-calculus) *$\mathcal{B}$ and $\mathcal{A}_{\mathcal{S}}$ form a reflection in* λlxr *of the* λ*-calculus (more precisely, a reflection, in the terms of* λlxr *whose free variables are $\mathcal{S}$, of the* λ*-terms whose free variables are among $\mathcal{S}$).*

Proof:

- For the simulation through $\mathcal{A}_{\mathcal{S}}$, consider a finite set of variables $\mathcal{S}$. From Theorem 121, we get that for all λ-term t and u such that $\mathsf{FV}(t) \subseteq \mathcal{S}$ and $t \longrightarrow_\beta u$ the following holds:

$$\begin{array}{rcl} \mathcal{A}_{\mathcal{S}}(t) & = & \mathcal{W}_{\mathcal{S}\setminus\mathsf{FV}(t)}(\mathcal{A}(t)) \\ & \longrightarrow^+_{\lambda\mathsf{lxr}} & \mathcal{W}_{\mathcal{S}\setminus\mathsf{FV}(t)}(\mathcal{W}_{\mathsf{FV}(t)\setminus\mathsf{FV}(u)}(\mathcal{A}(u))) \\ & = & \mathcal{W}_{\mathcal{S}\setminus\mathsf{FV}(u)}(\mathcal{A}(u)) \\ & = & \mathcal{A}_{\mathcal{S}}(u) \end{array}$$

- The simulation through $\mathcal{B}$ is Corollary 125.
- From Remark 127 we get $t = \mathcal{B}(\mathcal{A}_{\mathcal{S}}(t))$.
- From Lemma 129 we get $t \longrightarrow^*_{\mathsf{xr}} \mathcal{A}_{\mathcal{S}}(\mathcal{B}(t))$, since $t \longrightarrow^*_{\mathsf{xr}} \mathcal{W}_{\mathsf{FV}(t)\setminus\mathsf{FV}(\mathcal{B}(t))}(\mathcal{A}(\mathcal{B}(t)))$ and $\mathcal{S} = \mathsf{FV}(t)$.

□

We can now derive the confluence property for system $\lambda\mathsf{lxr}$:

Theorem 132 *The system $\lambda\mathsf{lxr}$ is confluent.*

Proof: From Theorems 5 and 131. □

5.3 Normalisation results

In sections 5.1 and 5.2 we have already established the property of subject reduction, and the reflection in $\lambda\mathsf{lxr}$ of λ-calculus. But a calculus which is defined in order to implement λ-calculus is also expected to satisfy preservation of strong normalisation (PSN), which we prove in this section.

5.3.1 Preservation of Strong Normalisation

We establish PSN of $\lambda\mathsf{lxr}$ by using the simulation technique with a memory operator as presented in Section 4.2. The application of the technique to $\lambda\mathsf{lxr}$ can be summarised as follows:

1. Define a relation $\mathcal{H}$ between linear $\lambda\mathsf{lxr}$-terms and λI-terms (Definition 98).
2. Show that $\longrightarrow_{\beta\pi}$ strongly simulates $\longrightarrow_B$ and weakly simulates $\longrightarrow_{\mathsf{xr}}$ through $\mathcal{H}$ (Theorem 134).
3. Deduce by Corollary 26 that if $t\ \mathcal{H}\ T$ and $T \in \mathsf{SN}^{\beta\pi}$, then $t \in \mathsf{SN}^{B\mathsf{xr}}$.
4. Show that $\mathcal{A}(t)\ \mathcal{H}\ i(t)$ (Theorem 136) and conclude by Theorem 109 the PSN property (Corollary 137).

We now proceed to develop the above points needed to conclude PSN as explained above.

Definition 98 The relation $\mathcal{H}$ between linear $\lambda\mathsf{lxr}$-terms and λI-terms is inductively defined in Fig. 5.12.

$$\frac{}{x\ \mathcal{H}\ x} \qquad \frac{t\ \mathcal{H}\ T}{\lambda x.t\ \mathcal{H}\ \lambda x.T} \qquad \frac{t\ \mathcal{H}\ T \quad u\ \mathcal{H}\ U}{tu\ \mathcal{H}\ TU} \qquad \frac{t\ \mathcal{H}\ T}{t\ \mathcal{H}\ [T,N]}\ N \in \Lambda I$$

$$\frac{t\ \mathcal{H}\ T \quad u\ \mathcal{H}\ U}{\langle u/x\rangle t\ \mathcal{H}\ \{{}^{U}\!/_{x}\}T} \qquad \frac{t\ \mathcal{H}\ T}{\mathcal{C}_x^{y\,|\,z}(t)\ \mathcal{H}\ \{{}^{x}\!/_{z}\}\{{}^{x}\!/_{y}\}T} \qquad \frac{t\ \mathcal{H}\ T}{\mathcal{W}_x(t)\ \mathcal{H}\ T}\ x \in \mathsf{FV}(T)$$

Figure 5.12: Relation between $\lambda\mathsf{lxr}$ & λI

The relation $\mathcal{H}$ satisfies the following properties.

Lemma 133 *If $t\ \mathcal{H}\ M$, then*

1. $\mathsf{FV}(t) \subseteq \mathsf{FV}(M)$
2. $M \in \Lambda I$
3. $x \notin \mathsf{FV}(t)$ *and* $N \in \Lambda I$ *implies* $t\ \mathcal{H}\ \{{}^{N}\!/_{x}\}M$
4. $t \equiv t'$ *implies* $t'\ \mathcal{H}\ M$
5. $\{{}^{\Psi}\!/_{\Phi}\}t\ \mathcal{H}\ \{{}^{\Psi}\!/_{\Phi}\}M$

Proof: Property (1) is a straightforward induction on the proof tree as well as Property (2). Properties (3) and (5) are also proved by induction on the tree, using Lemma 40. For Property (4):

- If $\mathcal{W}_x(\mathcal{W}_y(t))\ \mathcal{H}\ M$ then $M = [[T, \overrightarrow{T_i}], \overrightarrow{U_i}]$ with $t\ \mathcal{H}\ T$, $y \in \mathsf{FV}(T)$ and $x \in \mathsf{FV}([T, \overrightarrow{T_i}])$. Then $\mathcal{W}_y(\mathcal{W}_x(t))\ \mathcal{H}\ M$.
- If $\langle v/y\rangle\langle u/x\rangle t\ \mathcal{H}\ M$ with $y \notin \mathsf{FV}(u)$, then $M = [\{{}^{V}\!/_{y}\}[\{{}^{U}\!/_{x}\}T, \overrightarrow{T_i}], \overrightarrow{U_i}]$ with $t\ \mathcal{H}\ T$, $u\ \mathcal{H}\ U$ and $v\ \mathcal{H}\ V$.
  By α-conversion we can assume that $x \notin \mathsf{FV}(T_1) \cup \ldots \cup \mathsf{FV}(T_m) \cup \mathsf{FV}(V)$, so that $M = [\{{}^{V}\!/_{y}\}\{{}^{U}\!/_{x}\}[T, \overrightarrow{T_i}], \overrightarrow{U_i}] = [\left\{{}^{\{{}^{V}\!/_{y}\}U}\!/_{x}\right\}\{{}^{V}\!/_{y}\}[T, \overrightarrow{T_i}], \overrightarrow{U_i}]$. As a consequence $\langle u/x\rangle\langle v/y\rangle t\ \mathcal{H}\ M$, since by (3) we get $u\ \mathcal{H}\ \{{}^{V}\!/_{y}\}U$.
- Associativity and commutativity of contraction are very similar to the previous case.
- If $\langle u/x\rangle\mathcal{C}_w^{y\,|\,z}(p)\ \mathcal{H}\ M$, then $M = [\{{}^{U}\!/_{x}\}[\{{}^{w}\!/_{z}\}\{{}^{w}\!/_{y}\}P, \overrightarrow{P_i}], \overrightarrow{U_i}]$, with $p\ \mathcal{H}\ P$ and $u\ \mathcal{H}\ U$. We then conclude that $\mathcal{C}_w^{y\,|\,z}(\langle u/x\rangle p)\ \mathcal{H}\ M$ where $M = [\{{}^{w}\!/_{z}\}\{{}^{w}\!/_{y}\}\{{}^{U}\!/_{x}\}P, \overrightarrow{\{{}^{U}\!/_{x}\}P_i}, \overrightarrow{U_i}]$.

□

Theorem 134 (Simulation in ΛI)

1. *If $t\ \mathcal{H}\ T$ and $t \longrightarrow_{\mathsf{xr}} t'$, then $t'\ \mathcal{H}\ T$.*
2. *If $t\ \mathcal{H}\ T$ and $t \longrightarrow_{B} t'$, then there is $T' \in \Lambda I$ such that $t'\ \mathcal{H}\ T'$ and $T \longrightarrow^{+}_{\beta\pi} T'$.*

Proof: By induction on the reduction step. Remark that the case $t \cong t'$ is already considered by Lemma 133.4 so that we restrict the proof here to basic reduction steps.

- B: $(\lambda x.p)\ u \longrightarrow \langle u/x\rangle p$.
  Then $T = [[\lambda x.P, \overrightarrow{P_i}]U, \overrightarrow{U_i}]$ with $p \mathcal{H} P$ and $u \mathcal{H} U$. We then obtain the reduction sequence $T \longrightarrow_\pi^* [(\lambda x.P)U, \overrightarrow{P_i}, \overrightarrow{U_i}] \longrightarrow_\beta [\{^U/_x\} P, \overrightarrow{P_i}, \overrightarrow{U_i}] = T'$.

- Abs: $\langle u/x\rangle(\lambda y.p) \longrightarrow \lambda y.\langle u/x\rangle p$. Then $T = [\{^U/_x\}[\lambda y.P, \overrightarrow{P_i}], \overrightarrow{U_i}]$ with $p \mathcal{H} P$ and $u \mathcal{H} U$. We have $T = [\lambda y.(\{^U/_x\} P), \overrightarrow{\{^U/_x\} P_i}, \overrightarrow{U_i}]$.

- App1,App2: Similar to the previous case.

- Var: $\langle u/x\rangle x \longrightarrow u$. Then $T = [\{^U/_x\}[x, \overrightarrow{P_i}], \overrightarrow{U_i}]$ with $u \mathcal{H} U$. We have $T = [U, \overrightarrow{\{^U/_x\} P_i}, \overrightarrow{U_i}]$.

- Weak1: $\langle u/x\rangle \mathcal{W}_x(p) \longrightarrow \mathcal{W}_{\mathsf{FV}(u)}(p)$.
  Then $T = [\{^U/_x\}[P, \overrightarrow{P_i}], \overrightarrow{U_i}]$ with $p \mathcal{H} P$, $u \mathcal{H} U$, and $x \in \mathsf{FV}(P)$. We have $T = [\{^U/_x\} P, \overrightarrow{\{^U/_x\} P_i}, \overrightarrow{U_i}]$. Since $x \notin \mathsf{FV}(p)$, then by Lemma 133.3 $p \mathcal{H} \{^U/_x\} P$, and since $x \in \mathsf{FV}(P)$, $\mathsf{FV}(U) \subseteq \mathsf{FV}(\{^U/_x\} P)$. By Lemma 133.1 $\mathsf{FV}(u) \subseteq \mathsf{FV}(U)$ so $\mathsf{FV}(u) \subseteq \mathsf{FV}(\{^U/_x\} P)$ concludes the proof.

- Weak2: $\langle u/x\rangle \mathcal{W}_y(p) \longrightarrow \mathcal{W}_y(\langle u/x\rangle p)$.
  Then $T = [\{^U/_x\}[P, \overrightarrow{P_i}], \overrightarrow{U_i}]$ with $p \mathcal{H} P$, $u \mathcal{H} U$, and $y \in \mathsf{FV}(P)$. We have $T = [\{^U/_x\} P, \overrightarrow{\{^U/_x\} P_i}, \overrightarrow{U_i}]$ and we still have $y \in \mathsf{FV}(\{^U/_x\} P)$.

- Cont: $\langle u/x\rangle \mathcal{C}_x^{y\,|\,z}(p) \longrightarrow \mathcal{C}_\Phi^{\Psi\,|\,\Upsilon}(\langle \{^\Upsilon/_\Phi\} u/z\rangle \langle \{^\Psi/_\Phi\} u/y\rangle p)$.
  Then $T = [\{^U/_x\}[\{^x/_z\}\{^x/_y\} P, \overrightarrow{P_i}], \overrightarrow{U_i}]$ with $p \mathcal{H} P$ and $u \mathcal{H} U$. We obtain the following equality $T = [\{^U/_z\}\{^U/_y\} P, \overrightarrow{\{^U/_x\} P_i}, \overrightarrow{U_i}]$ which can be expressed as

  $$T = [\{^\Phi/_\Upsilon\}\{^\Phi/_\Psi\}\left\{^{U''}/_z\right\}\left\{^{U'}/_y\right\} P, \overrightarrow{\{^U/_x\} P_i}, \overrightarrow{U_i}]$$

  where $U' = \{^\Psi/_\Phi\} U$ and $U'' = \{^\Upsilon/_\Phi\} U$. We obtain $\{^\Psi/_\Phi\} u \mathcal{H} U'$ and $\{^\Upsilon/_\Phi\} u \mathcal{H} U''$ by Lemma 133.5.

- Comp: $\langle u/x\rangle\langle v/y\rangle p \longrightarrow \langle\langle u/x\rangle v/y\rangle p$ where $x \in \mathsf{FV}(v)$.
  Then $T = [\{^U/_x\}[\{^Q/_y\} P, \overrightarrow{P_i}], \overrightarrow{U_i}]$ with $t \mathcal{H} P$, $v \mathcal{H} Q$, and $u \mathcal{H} U$.
  We have $T = [\left\{^{\{^{U}/_x\}Q}/_y\right\}\{^U/_x\} P, \overrightarrow{\{^U/_x\} P_i}, \overrightarrow{U_i}]$. Notice that we obtain $t \mathcal{H} \{^U/_x\} P$ by Lemma 133.3.

- WAbs, WApp1, WApp2, WSubs, Cross are straightforward because the condition $x \in \mathsf{FV}(P)$ that is checked by $\mathcal{W}_x(\_)$ is just changed into a side-condition $x \in \mathsf{FV}(Q)$ (checked one step later), where $x \in \mathsf{FV}(P)$ implies $x \in \mathsf{FV}(Q)$.

- Merge: $\mathcal{C}^{y\,|\,z}_{w}(\mathcal{W}_y(p)) \longrightarrow \{{}^{w}\!/_{z}\}p$.
  Then $T = [\{{}^{w}\!/_{z}\}\{{}^{w}\!/_{y}\}[P, \overrightarrow{P_i}], \overrightarrow{U_i}]$ with $t\ \mathcal{H}\ P$ and $y \in \mathsf{FV}(P)$. We then have the equality $T = [\{{}^{w}\!/_{y}\}[\{{}^{w}\!/_{z}\}P, \overrightarrow{\{{}^{w}\!/_{z}\}P_i}], \overrightarrow{U_i}]$ and we conclude by Lemma 133.3.

- CAbs: $\mathcal{C}^{y\,|\,z}_{w}(\lambda x.t) \longrightarrow \lambda x.\mathcal{C}^{y\,|\,z}_{w}(p)$.
  Then $T = [\{{}^{w}\!/_{z}\}\{{}^{w}\!/_{y}\}[\lambda x.P, \overrightarrow{P_i}], \overrightarrow{U_i}]$ with $t\ \mathcal{H}\ P$.
  We have $T = [\lambda x.(\{{}^{w}\!/_{z}\}\{{}^{w}\!/_{y}\}P), \overrightarrow{\{{}^{w}\!/_{z}\}\{{}^{w}\!/_{y}\}P_i}, \overrightarrow{U_i}]$.

- CApp1, CApp2: Similar to the previous case.

- CSubs: We have $\mathcal{C}^{y\,|\,z}_{w}(\langle u/x\rangle p)\ \mathcal{H}\ [\{{}^{w}\!/_{z}\}\{{}^{w}\!/_{y}\}[\{{}^{U}\!/_{x}\}P, \overrightarrow{P_i}], \overrightarrow{U_i}]$ which is equal to $T = [[\{{}^{\{{}^{w}\!/_{z}\}\{{}^{w}\!/_{y}\}U}\!/_{x}\}\{{}^{w}\!/_{z}\}\{{}^{w}\!/_{y}\}P, \overrightarrow{\{{}^{w}\!/_{z}\}\{{}^{w}\!/_{y}\}P_i}], \overrightarrow{U_i}]$ by Lemma 40. We have $\langle \mathcal{C}^{y\,|\,z}_{w}(u)/x\rangle p\ \mathcal{H}\ T$ by Lemma 133.3, which concludes this case.

Now for the contextual closure, we use the fact that if $P \longrightarrow_{\beta\pi} P'$ then $\{{}^{U}\!/_{x}\}P \longrightarrow_{\beta\pi} \{{}^{U}\!/_{x}\}P'$, and if moreover $x \in \mathsf{FV}(P)$ and $U \longrightarrow_{\beta\pi} U'$ then $\{{}^{U}\!/_{x}\}P \longrightarrow^{+}_{\beta\pi} \{{}^{U'}\!/_{x}\}P$. The latter is useful for explicit substitutions: if $\langle t/x\rangle p\ \mathcal{H}\ Q$ and $t \longrightarrow_B t'$, then $Q = [\{{}^{T}\!/_{x}\}P, \overrightarrow{U_i}]$ with $p\ \mathcal{H}\ P$, $t\ \mathcal{H}\ T$ and by the by i.h.we get $T \longrightarrow^{+}_{\beta\pi} T'$ such that $t'\ \mathcal{H}\ T'$. Since $x \in \mathsf{FV}(p)$, $x \in \mathsf{FV}(P)$ by Lemma 133.1, and hence $Q \longrightarrow^{+}_{\beta\pi} [\{{}^{T'}\!/_{x}\}P, \overrightarrow{U_i}]$. □

Corollary 135 *If $t\ \mathcal{H}\ T$ and $T \in \mathcal{SN}^{\beta\pi}$, then $t \in \mathcal{SN}^{\lambda\mathsf{lxr}}$.*

Proof: Given that xr is terminating (Lemma 110), it suffices to apply Corollary 26. □

Theorem 136 *For any λ-term u, $\mathcal{A}(u)\ \mathcal{H}\ i(u)$.*

Proof: By induction on u:

- $x\ \mathcal{H}\ x$ trivially holds.

- If $u = \lambda x.t$, then $\mathcal{A}(t)\ \mathcal{H}\ \mathsf{i}(t)$ holds by the by i.h. Therefore, we obtain $\lambda x.\mathcal{A}(t)\ \mathcal{H}\ \lambda x.\mathsf{i}(t)$ and $\lambda x.\mathcal{W}_x(\mathcal{A}(t))\ \mathcal{H}\ \lambda x.[\mathsf{i}(t), x]$.

- If $u = (t\ u)$, then $\mathcal{A}(t)\ \mathcal{H}\ \mathsf{i}(t)$ and $\mathcal{A}(u)\ \mathcal{H}\ \mathsf{i}(u)$ hold by the i.h. and $\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(t)\ \mathcal{H}\ \{{}^{\Upsilon}\!/_{\Phi}\}\mathsf{i}(t)$ and $\{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(u)\ \mathcal{H}\ \{{}^{\Upsilon}\!/_{\Phi}\}\mathsf{i}(u)$ by Lemma 133-5. Since $\{{}^{\Phi}\!/_{\Upsilon}\}\{{}^{\Upsilon}\!/_{\Phi}\}\mathsf{i}(t) = \mathsf{i}(t)$ (and the same for $\mathsf{i}(u)$), we can then conclude $\mathcal{C}^{\Psi\,|\,\Upsilon}_{\Phi}(\{{}^{\Psi}\!/_{\Phi}\}\mathcal{A}(t)\ \{{}^{\Upsilon}\!/_{\Phi}\}\mathcal{A}(u))\ \mathcal{H}\ \mathsf{i}(t)\ \mathsf{i}(u)$. □

Corollary 137 (PSN) *For any λ-term t, if $t \in \mathcal{SN}^{\beta}$, then $\mathcal{A}(t) \in \mathcal{SN}^{\lambda\mathsf{lxr}}$.*

Proof: If $t \in \mathsf{SN}^{\beta}$, then $\mathsf{i}(t) \in \mathsf{SN}^{\beta\pi}$ by Theorems 109 and 108. As $\mathcal{A}(t)\ \mathcal{H}\ \mathsf{i}(t)$ by Theorem 136, then we conclude $\mathcal{A}(t) \in \mathsf{SN}^{\lambda\mathsf{lxr}}$ by Corollary 135. □

5.3.2 Strong normalisation of typed terms

We slightly refine the translation $\mathcal{B}$ by lifting all explicit substitutions into B-redexes (as suggested in [Her95]):

Definition 99 (Refined translation from λlxr to λ-calculus) The function $\mathsf{H}(t)$ is defined by induction in Fig. 5.13.

$$\begin{array}{lcl}
\mathsf{H}(x) & := & x \\
\mathsf{H}(\lambda x.t) & := & \lambda x.\mathsf{H}(t) \\
\mathsf{H}(\mathcal{W}_x(t)) & := & (\lambda y.\mathsf{H}(t))\ x \\
\mathsf{H}(\mathcal{C}_x^{y\,|\,z}(t)) & := & (\lambda y.\lambda z.\mathsf{H}(t))\ x\ x \\
\mathsf{H}(t\ u) & := & \mathsf{H}(t)\ \mathsf{H}(u) \\
\mathsf{H}(\langle u/x\rangle t) & := & (\lambda x.\mathsf{H}(t))\ \mathsf{H}(u)
\end{array}$$

Figure 5.13: From λlxr to λ-calculus

We easily get:

Lemma 138 *For all λlxr-term t, $\mathcal{A}(H(t)) \longrightarrow^*_{\lambda lxr} t$.*

Proof: Straightforward induction on t. □

A straightforward induction on typing derivations allows us to show:

Lemma 139 (H preserves types) *If t is a λlxr-term such that $\Gamma \vdash_{\lambda lxr} t : A$, then $\Gamma \vdash_{\lambda} H(t) : A$.*

Theorem 140 (Strong normalisation of typed terms) *If $\Gamma \vdash_{\lambda lxr} t : A$ then $t \in SN^{B,xr}$.*

Proof: If $\Gamma \vdash_{\lambda\mathsf{lxr}} t : A$ then $\Gamma \vdash_{\lambda} \mathsf{H}(t) : A$ (Lemma 139), so $\mathsf{H}(t) \in \mathsf{SN}^{\beta}$ (Theorem 62). By PSN (Corollary 137) we get $\mathcal{A}(\mathsf{H}(t)) \in \mathsf{SN}^{B,\mathsf{xr}}$. Since $\mathcal{A}(\mathsf{H}(t)) \longrightarrow^*_{\lambda\mathsf{lxr}} t$ (Lemma 138), we also have $t \in \mathsf{SN}^{B,\mathsf{xr}}$. □

Conclusion

The calculus λlxr extends the explicit substitution paradigm, in that it features new constructors in a simple syntax equipped with a natural operational semantics, given by the notion of reduction modulo a set of equations, and further decomposing β-reduction into more atomic steps.

These constructors represent a tool to analyse and control when sub-terms are duplicated or erased during computation, thus providing an elegant framework for studying resource usage or control. This relates to contractions and weakenings of proof-nets for linear logic [Gir87] to which a sound and complete interpretation can be defined [KL05, KL06]. From a computational point of view, weakening constructors are a useful tool to handle garbage collection. Indeed, free variables are never lost and weakening constructors are pulled out to the top-level during computation.

In contrast to other HOC in which there is a reflection of λ-calculus, λlxr has full composition of substitutions and satisfies PSN. It also satisfies confluence and strong normalisation of simply-typed terms.

It is worth mentioning the calculus obtained by turning the equation P_{cs} into a reduction rule (from left to right) and by eliminating reduction rules WSubs and CSubs satisfies exactly the same properties as the calculus presented in this chapter, namely Theorems 118,140,121,125,132, and Corollary 137. However, these rules seem to be necessary for the confluence on open terms (ongoing work).

Many points raised in this work deserve further development. The first one concerns the study of reduction strategies well-adapted to handle the constructors for substitution, erasure and duplication. This may take into account the notion of weak reduction used to implement functional programming [LM99].

Proof techniques used in the literature to show PSN of calculi with explicit substitutions (zoom-in [ABR00], minimality [BBLRD96], labelled RPO [BG99], PSN by standardisation [KOvO01], or intersection types) are not all easy to adapt/extend to reduction modulo and other formalisms. The proof technique used here seems really flexible.

Using the PSN result, we believe that we can characterise very neatly the strongly normalising terms of λlxr as the terms typable with intersection types, as it it the case in λ-calculus as well as in the explicit substitution calculus λx [LLD$^+$04].

First-order term syntax for λlxr via de Bruijn indices [dB72], or other special notation to avoid α-conversion as for example explicit scoping [HvO03] or also director strings [SFM03], would make implementation easier.

Connections with similar approaches relating graph formalisms to term calculi, as for example that of Hasegawa [Has99] also merits further investigations.

Chapter 6

Cut-elimination in G3ii & stable fragments

This chapter tackles the notion of computation in sequent calculus based on cut-elimination. In a way similar to a survey, it presents traditional ideas in a unified framework, using the traditional sequent calculus G3ii and its corresponding HOC of proof-terms called λG3 and presented in Chapter 2.

Starting from the admissibility of the cut-rule in G3ii, we use our framework with terms called λG3 to relate inductive proofs of term-irrelevant admissibility to rewrite systems that eliminate the cut-constructor. These systems in fact make sense even without the notion of typing for λG3, although we do need typing to prove their strong normalisation. We identify the structure of such rewrite systems that perform cut-elimination in a typed framework, in that they are made of a kernel that reduces *principal cuts/cut constructors* and propagation systems which may vary. In this generic framework we show the critical pairs of these systems, which can be solved in two canonical ways leading to the introduction of a generic notion of CBN and CBV sub-systems. We present three kinds of propagation system with a comparative approach, essentially by investigating their ability to simulate β-reduction through Gentzen's or Prawitz's encodings described in Chapter 2. We also compare the CBN and CBV equational theories that these propagation systems produce.

We then show two restrictions on λG3-terms and their typing rules that correspond to the sequent calculi LJT and LJQ [Her95], and show that these restrictions are respectively stable under the CBN and CBV sub-systems, which become their natural cut-elimination procedures. Two simple purification rules, reducing arbitrary terms of λG3 to terms of these restrictions, show the completeness of the two fragments.

We recall the strong connection between LJT, the $\overline{\lambda}$-calculus and the λ-calculus, by means of a reflection based on Prawitz's encoding. We also give a new proof of the PSN property for $\overline{\lambda}$, as another illustrative example of the safeness and minimality technique.

We then investigate LJQ, described as the typing system of a term syntax, which we then use to establish a connection with the CBV calculus λ_{C} of Moggi [Mog88]. A preliminary version of this work has been published in [DL06].

6.1 Cut-elimination

6.1.1 Aims

The main property of G3ii is that the cut-rule is admissible in the cut-free system, so for any derivation using cuts there exists a derivation of the same logical sequent. As described in Chapter 2, this corresponds, in the framework of λG3 with terms, to the term-irrelevant admissibility of the cut-rule in the rest of the system.

Usually, admissibility of a rule in sequent calculus is proved by induction, for instance on derivations of the premisses. The very same argument can be used to prove term-irrelevant admissibility, in a canonical typed HOC that corresponds to the sequent calculus. Moreover, it defines a process that transforms a term M that uses the constructor typed by the rule into another term M', with the same type in the same environment, that does not use this constructors.

The process given by the induction is in fact a notion of reduction given by an innermost strategy in a rewrite system that specifies how to eliminate the constructor in the following sense:

Property 141

141.1 A term containing the constructor is reducible (all cases are covered by the induction).

141.2 The rewrite system satisfies the subject reduction property.

141.3 The innermost strategy given by the inductive argument terminates, using the induction measure.

This gives a notion of reduction in the typed HOC that is weakly normalising (from point 3). This suffices to prove term-irrelevant admissibility, but for a general notion of computation we often want a strong normalisation property. And in fact it is often the case that the induction measure that proves termination of the innermost strategy also proves the strong normalisation of general reduction.[1]

This applies to G3ii and λG3, in that the admissibility of cut gives rise to a cut-elimination process. In the framework of λG3, this can be done by means of a rewrite system, cut-free proofs being thus denoted by terms in normal form.

[1]Strong normalisation can actually be directly inferred from weak innermost normalisation in the particular case of orthogonal first-order systems [O'D77]

Various such reduction systems are given in the literature, but their diversity is striking. Whilst this might be explained by the diverse requirements that the systems fulfil on top of the above properties, choices of design are often given little justification. They might aim at simplicity, strong normalisation, confluence, simulation of β-reduction...

Here we try to cover various reduction systems achieving cut-elimination, in the prospect of showing connections between proof theory and computer science, especially rewriting and (functional) programming. We thus express these systems with two (diverging) concerns:

- formulating them as general, unrestricted, and simple as we can without breaking strong normalisation,
- partitioning and restricting them only to give them semantical meaning, in effect relating them to CBV and CBN semantics.

Proving strong normalisation of the cut-reduction system inferred from inductive proofs of cut-admissibility such as that of [Gen35] is often simpler than proving strong normalisation of typed λ-calculus (this was in fact the motivation of [Gen35] for introducing sequent calculus). This is true until cut-reduction is able to strongly simulate β-reduction. Indeed, a criterion on which we compare the various systems expressed with the above concerns is their computational strength, how they simulate, or more generally how they relate to, β-reduction.

Finally, we choose to base those systems on $\lambda\mathsf{G3}$ as it is the most natural framework without, for instance, adding to the syntax extra constructors -and typing rules- as in [Her94, EFP06], until Chapter 7 where the $\mathsf{G4ii}$-calculus requires a particular treatment.

6.1.2 Kernel & propagation system

Despite their diversity, all the cut-elimination systems have to deal with those basic cases when the cut-type is principal in both premisses. In these cases the inference rules provide canonical ways of reducing the cut, possibly creating cuts on sub-formulae. These cuts are sometimes called *logical cuts* [Urb00], or *principal cuts*. At the level of terms, they correspond to the constructs $\langle N \dagger x.M \rangle$ where N is a value and M is an x-covalue, which we call *logical cut-constructor* or *principal cut-constructor*. Fig. 6.1 shows the standard rewrite rules to reduce them. We denotes the reduction relation $\longrightarrow_{\mathsf{princ_s}}$.

What is less standard is the way to deal with those cases in which the cut-type is not principal in at least one premiss. What process will reduce the problem to the case of principal cuts? It clearly depends on the proof of the premiss in which the cut-formula is not principal, pushing the cut thereinto, and can disregard the proof of the other premiss. But then if in both premisses the cut-type is not principal, a choice has to be made, leading to non-confluence of the

B	$\langle \lambda x.M \dagger y.y[N, z.P] \rangle$	$\longrightarrow$	$\langle \langle N \dagger x.M \rangle \dagger z.P \rangle$	if $y \notin \mathsf{FV}(P) \cup \mathsf{FV}(N)$
	$\langle x \dagger y.y[N, z.P] \rangle$	$\longrightarrow$	$x[N, z.P]$	if $y \notin \mathsf{FV}(P) \cup \mathsf{FV}(N)$
	$\langle \lambda x.M \dagger y.y \rangle$	$\longrightarrow$	$\lambda x.M$	
	$\langle x \dagger y.y \rangle$	$\longrightarrow$	x	

Figure 6.1: Principal cut-reductions

process unless it is restricted by a general preference (or strategy) that determines each of these choices. It is interesting to see that this non-confluence can even occur in the *intuitionistic* sequent calculus, whilst it is often thought to be a specificity of classical logic.

Cut-elimination systems thus reduce non-principal cuts with rules based on two kinds of behaviour: *left-propagation*, denoted $\longrightarrow_{\mathsf{left}}$, reduces a cut by pushing it to the left, depending on the proof of its first premiss (in which the cut-type is not principal), while *right-propagation*, denoted $\longrightarrow_{\mathsf{right}}$, reduces a cut by pushing it to the right depending on the proof of its second premiss (in which the cut-type is not principal). The former reduces $\langle N \dagger x.M \rangle$ depending on N that is not a value (in other words, it alleviates the body N of the cut-constructor, regardless of M), and the latter reduces it depending on M that is not an x-covalue, regardless of N.

Clearly, the two situations overlap when neither N is a value nor M is an x-covalue, or, in a typed framework, when in neither premisses of the cut the cut-type is principal. This generates critical pairs and non-confluence, and an interesting point is that techniques to avoid this situation (namely, deciding which kind of rule will apply with priority) reveal connections with the CBV and CBN semantics of functional programming, as introduced in Chapter 3. Thus, always giving preference to left-propagation corresponds to CBV, while preference to the right-propagation corresponds to CBN, which we define as follows:

Definition 100 (CBV & CBN sub-systems)

- The CBN-sub-system restricts the left-propagation system by requiring it to reduce $\langle N \dagger x.M \rangle$ only when M is an x-covalue (and thus the cut-constructor cannot be right-propagated).

  We write $\longrightarrow_{\mathsf{CBN_s}}$ for the reduction relation generated by $\longrightarrow_{\mathsf{princ_s}}$, $\longrightarrow_{\mathsf{right}}$, and this restricted $\longrightarrow_{\mathsf{left}}$.

- The CBV-sub-system restricts the right-propagation system by requiring it to reduce $\langle N \dagger x.M \rangle$ only when N is a value (and thus the cut-constructor cannot be left-propagated).

  We write $\longrightarrow_{\mathsf{CBV_s}}$ for the reduction relation generated by $\longrightarrow_{\mathsf{princ_s}}$, $\longrightarrow_{\mathsf{left}}$, and this restricted $\longrightarrow_{\mathsf{right}}$.

Now, obtaining the completeness of these restrictions for cut-elimination also justifies the need for a general strong normalisation result of the whole system rather than weak normalisation.

Finally, variations of the principal rules are of interest. When in one of the premisses of a principal cut, the cut-type is introduced by an axiom, the standard way to eliminate the cut is to only keep the proof of the other premiss (possibly using a contraction if the axiom proves the first premiss). But this works in fact whether or not the cut-type is principal in this other premiss, and it thus applies to cuts that might not be logical as well. This generalises the scope of principal rules, the reduction relation of which we denote $\longrightarrow_{\mathsf{princ}}$, but it also simplifies them as shown in Fig 6.2.

B	$\langle \lambda x.M \dagger y.y[N, z.P] \rangle$	$\longrightarrow$	$\langle \langle N \dagger x.M \rangle \dagger z.P \rangle$	if $y \notin \mathsf{FV}(P) \cup \mathsf{FV}(N)$
B_1	$\langle x \dagger y.N \rangle$	$\longrightarrow$	$\{^{x}/_{y}\} N$	
B_2	$\langle M \dagger y.y \rangle$	$\longrightarrow$	M	

Figure 6.2: Generalised principal reductions

We define $\longrightarrow_{\mathsf{CBV}}$ and $\longrightarrow_{\mathsf{CBN}}$ just like $\longrightarrow_{\mathsf{CBV_s}}$ and $\longrightarrow_{\mathsf{CBN_s}}$, but considering $\longrightarrow_{\mathsf{princ}}$ instead of $\longrightarrow_{\mathsf{princ_s}}$.

Convenient and simplifying though this extension is, it creates new critical pairs with the left and right propagation (making confluence of $\longrightarrow_{\mathsf{CBV}}$ and $\longrightarrow_{\mathsf{CBN}}$ more difficult to prove).

In the next section we present three kinds of propagation rules, respectively generating systems SI, KK and JC, for which we have

$$(\longleftrightarrow^*_{\mathsf{CBV_{KKs}}}) = (\longleftrightarrow^*_{\mathsf{CBV_{KK}}}) = (\longleftrightarrow^*_{\mathsf{CBV_{JCs}}}) = (\longleftrightarrow^*_{\mathsf{CBV_{JC}}})$$
$$(\longleftrightarrow^*_{\mathsf{CBN_{KKs}}}) = (\longleftrightarrow^*_{\mathsf{CBN_{KK}}}) = (\longleftrightarrow^*_{\mathsf{CBN_{JCs}}}) = (\longleftrightarrow^*_{\mathsf{CBN_{JC}}})$$

We call those equational theories $\equiv_{\mathsf{CBV}}$ and $\equiv_{\mathsf{CBN}}$. In system SI, we only have in general

$$(\longleftrightarrow^*_{\mathsf{CBV_{SIs}}}) \subseteq (\longleftrightarrow^*_{\mathsf{CBV_{SI}}}) \subseteq (\equiv_{\mathsf{CBV}})$$
$$(\longleftrightarrow^*_{\mathsf{CBN_{SIs}}}) \subseteq (\longleftrightarrow^*_{\mathsf{CBN_{SI}}}) \subseteq (\equiv_{\mathsf{CBN}})$$

However, on weakly normalising terms we also have $(\longleftrightarrow^*_{\mathsf{CBV_{SIs}}}) = (\longleftrightarrow^*_{\mathsf{CBV_{SI}}})$ and $(\longleftrightarrow^*_{\mathsf{CBN_{SIs}}}) = (\longleftrightarrow^*_{\mathsf{CBN_{SI}}})$. On strongly normalising terms, in particular for typed terms, we even have $(\longleftrightarrow^*_{\mathsf{CBV_{SIs}}}) = (\longleftrightarrow^*_{\mathsf{CBV_{SI}}}) = (\equiv_{\mathsf{CBV}})$ and $(\longleftrightarrow^*_{\mathsf{CBN_{SIs}}}) = (\longleftrightarrow^*_{\mathsf{CBN_{SI}}}) = (\equiv_{\mathsf{CBN}})$.

6.1.3 Instances of propagation systems

Simple propagation - System SI

We present in Figure 6.3 perhaps the simplest rewrite systems for left and right propagation, respectively denoted $\longrightarrow_{\mathsf{left_{SI}}}$ and $\longrightarrow_{\mathsf{right_{SI}}}$. We call SI (resp. SIs)

the system with these rules and those of princ (resp. of $\mathsf{princ_s}$).

$$
\begin{array}{lll}
\mathsf{left}_1 & \langle z[N, y.P] \dagger x.M\rangle & \longrightarrow\ z[N, y.\langle P \dagger x.M\rangle] \\
 & & \\
\mathsf{right}_1 & \langle N \dagger x.y\rangle & \longrightarrow\ y \\
\mathsf{right}_2 & \langle N \dagger x.(\lambda y.M)\rangle & \longrightarrow\ \lambda y.\langle N \dagger x.M\rangle \\
\mathsf{right}_3 & \langle N \dagger x.x[M, z.P]\rangle & \longrightarrow\ \langle N \dagger x.x[\langle N \dagger x.M\rangle, z.\langle N \dagger x.P\rangle]\rangle \\
 & & \qquad\qquad \text{if } x \in \mathsf{FV}(M) \cup \mathsf{FV}(P) \\
\mathsf{right}_4 & \langle N \dagger x.x'[M, z.P]\rangle & \longrightarrow\ x'[\langle N \dagger x.M\rangle, z.\langle N \dagger x.P\rangle]
\end{array}
$$

Figure 6.3: SI-propagation

Notice that in rule right_3, the side-condition $x \in \mathsf{FV}(M) \cup \mathsf{FV}(P)$ comes from our requiring the term, to which the cut-constructor is applied, to not be an x-covalue. Otherwise, the rule could be re-applied indefinitely, although termination could be alternatively recovered by applying the rule only before rule B as follows:

$$\langle \lambda x.M \dagger y.y[N, z.P]\rangle \longrightarrow\ \langle\langle\langle \lambda x.M \dagger y.N\rangle \dagger x.M\rangle \dagger z.\langle \lambda x.M \dagger y.P\rangle\rangle$$

The whole system remains complete for cut-elimination, but it intermingles the three rewrite systems in a way that obscures the connection with CBV and CBN, which we want to reveal formally. We therefore keep the side-condition.

The reduction relation $\longrightarrow_{\mathsf{SI}}$ can easily be proved satisfying Properties 141.1 and 141.2 (subject reduction). Now we consider normalisation:

Theorem 142 (Strong Normalisation)

1. $\longrightarrow_{\mathsf{SI}\setminus\mathsf{B}}$ *is strongly normalising.*
2. *If* $\Gamma \vdash_{\lambda\mathsf{G3}} M : A$ *then* $M \in \mathsf{SN}^{\mathsf{SI}}$.

Proof: Both results can be proved using a LPO based on the following infinite first-order signature and its precedence relation:

$$\mathsf{sub}(\_,\_) \succ \mathsf{cut}(\_,\_) \succ \mathsf{ii}(\_,\_) \succ \mathsf{i}(\_) \succ \star$$

We define the following encoding:

$$
\begin{array}{lll}
\overline{x} & := & \star \\
\overline{\lambda x.M} & := & \mathsf{i}(\overline{M}) \\
\overline{x[N, y.M]} & := & \mathsf{ii}(\overline{N}, \overline{M}) \\
\overline{\langle N \dagger y.M\rangle} & := & \mathsf{cut}(\overline{N}, \overline{M}) \quad \text{if } M \text{ is a } y\text{-covalue} \\
\overline{\langle N \dagger y.M\rangle} & := & \mathsf{sub}(\overline{N}, \overline{M}) \quad \text{otherwise}
\end{array}
$$

All the rules but B decrease the encoding. In the typed case, we refine the above precedence relation by

$$\mathsf{sub}^B(\_,\_) \succ \mathsf{cut}^B(\_,\_) \succ \cdots \succ \mathsf{sub}^A(\_,\_) \succ \mathsf{cut}^A(\_,\_) \succ \mathsf{ii}(\_,\_) \succ \mathsf{i}(\_) \succ \star$$

if type $B \sqsupset A$. We now refine the above encoding: technically, it is now defined on the typing trees (and the proof of subject reduction shows how the rules transform the trees), although we abusively express the encoding from the proof-term:

$$\begin{array}{lll}
\overline{x} & := & \star \\
\overline{\lambda x.M} & := & \mathsf{i}(\overline{M}) \\
\overline{x[N, y.M]} & := & \mathsf{ii}(\overline{N}, \overline{M}) \\
\overline{\langle N \dagger y.M \rangle} & := & \mathsf{cut}^A(\overline{N}, \overline{M}) \quad \text{if } M \text{ is a } y\text{-covalue} \\
\overline{\langle N \dagger y.M \rangle} & := & \mathsf{sub}^A(\overline{N}, \overline{M}) \quad \text{otherwise}
\end{array}$$

where A is the cut-formula. Now rule B decreases the encoding as well as the other rules. □

Now we state a few properties about the CBV and CBN relations :

Theorem 143 (Confluence)

1. $\longrightarrow_{\mathsf{CBN_{SIs}}}$ *and* $\longrightarrow_{\mathsf{CBV_{SIs}}}$ *are confluent.*
2. *We conjecture that* $\longrightarrow_{\mathsf{CBN_{SI}}}$ *and* $\longrightarrow_{\mathsf{CBV_{SI}}}$ *are confluent.*

Proof:

1. The left and right propagation systems are orthogonal higher-order rewrite systems, and hence, so are $\longrightarrow_{\mathsf{CBN_{SIs}}}$ and $\longrightarrow_{\mathsf{CBV_{SIs}}}$, which entails confluence (see e.g. [Ter03]).
2. We could either try the method of parallel reduction (see e.g. [Tak89]) or establish that a CPS-translation forms a pre-Galois connection with a confluent fragment of λ-calculus (as in Chapter 3).

□

Lemma 144 *Provided the terms are weakly normalising,*

1. $\langle x \dagger y.M \rangle \longleftrightarrow^*_{\mathsf{CBV_{SIs}}} \{^x/_y\} M$ *and* $\langle x \dagger y.M \rangle \longleftrightarrow^*_{\mathsf{CBN_{SIs}}} \{^x/_y\} M$,
2. $\langle M \dagger y.y \rangle \longleftrightarrow^*_{\mathsf{CBV_{SIs}}} M$ *and* $\langle M \dagger y.y \rangle \longleftrightarrow^*_{\mathsf{CBN_{SIs}}} M$.

Proof: We first prove it for $M \in \lambda\mathsf{G3^{cf}}$ by structural induction on M. Then for an arbitrary M we first reduce it using Property 141.1, to a $M' \in \lambda\mathsf{G3^{cf}}$ for which the statement holds. □

Theorem 145 (Equational theories) *On weakly normalising terms,* $(\longleftrightarrow^*_{\mathsf{CBV_{SIs}}}) = (\longleftrightarrow^*_{\mathsf{CBV_{SI}}})$ *and* $(\longleftrightarrow^*_{\mathsf{CBN_{SIs}}}) = (\longleftrightarrow^*_{\mathsf{CBN_{SI}}})$.

Proof: This is a direct corollary. □

Now we prove a lemma about commutation of cuts, that will later be used to relate the equational theories of system SI to those of richer propagation systems.

Lemma 146 *Provided $\langle N \dagger x.M\rangle$ is strongly normalising, we have*

$$\begin{array}{lll}
\langle P \dagger y.\langle N \dagger x.M\rangle\rangle & \longleftrightarrow^*_{CBN_{SI}} & \langle\langle P \dagger y.N\rangle \dagger x.\langle P \dagger y.M\rangle\rangle \\
\langle P \dagger y.\langle N \dagger x.M\rangle\rangle & \longleftrightarrow^*_{CBV_{SI}} & \langle\langle P \dagger y.N\rangle \dagger x.\langle P \dagger y.M\rangle\rangle \quad \textit{if } P \textit{ is a value} \\
\langle\langle N \dagger x.M\rangle \dagger y.P\rangle & \longleftrightarrow^*_{CBN_{SI}} & \langle N \dagger x.\langle M \dagger y.P\rangle\rangle \quad \textit{if } P \textit{ is a } y\textit{-covalue} \\
\langle\langle N \dagger x.M\rangle \dagger y.P\rangle & \longleftrightarrow^*_{CBN_{SI}} & \langle N \dagger x.\langle M \dagger y.P\rangle\rangle
\end{array}$$

Proof: By induction on the length of the longest reduction sequence reducing $\langle N \dagger x.M\rangle$. If $N \notin \lambda\mathsf{G3}^{\mathsf{cf}}$ or $M \notin \lambda\mathsf{G3}^{\mathsf{cf}}$ we can reduce it and the induction hypothesis applies. If both are in $\lambda\mathsf{G3}^{\mathsf{cf}}$ then $\langle N \dagger x.M\rangle$ is the redex of a rewrite rule and then it is a case analysis on the rule. □

A richer propagation - System KK

Simple though it is, the above propagation system does not make cut-elimination powerful enough to simulate β-reduction. Consider the reduction $M = (\lambda x.(\lambda x_1.x)\ x_2)\ \lambda y.y \longrightarrow_\beta (\lambda x_1.\lambda y.y)\ x_2 = N$. We have

$$\begin{array}{rcl}
\mathcal{G}^2(M) = \mathcal{P}r(M) & = & \langle \lambda x.\ \langle \lambda x_1.x \dagger z_3.z_3[x_2, z_4.z_4]\rangle \dagger z_1.z_1[\lambda y.y, z_2.z_2]\rangle \\
& \longrightarrow^*_{\mathsf{SI}} & \langle \lambda y.y \dagger x.\langle \lambda x_1.x \dagger z_3.z_3[x_2, z_4.z_4]\rangle\rangle
\end{array}$$

but then we are stuck, because all we could do before propagating the cut we want is reduce the inner cut first, which encodes the β-redex that remains in N and which we still therefore need.

Whether λ-calculus is encoded via Gentzen's or Prawitz's translation, a β-redex is encoded using a cut, and a substitution is implementing by reducing and propagating a cut. Hence, since substitutions can instantiate variables through a β-redex, cut should be propagated through cuts. However, a permutation of cuts such as $\langle N \dagger x.\langle M \dagger y.P\rangle\rangle \longrightarrow \langle\langle N \dagger x.M\rangle \dagger y.\langle N \dagger x.P\rangle\rangle$ would fail to be terminating. Variations on that problem can be found in works tackling the notion of composition in explicit substitution calculi such as $\lambda\mathsf{x}$ [BR95], already mentioned in Chapters 4 and 5.

However, [Kik04b, Kik06] noticed that arbitrary permutations were not necessary for the simulation, but only specific ones. Following his ideas, the permutation rules of Fig 6.4 enable the simulation of β-reduction.

We call KK (resp. KKs) the system with these rules and those of princ (resp. with $\mathsf{princ_s}$).

Remark 147 If all cut-constructors in N are logical and $x \notin \mathsf{FV}(N)$ then $\langle N' \dagger x.N\rangle \longrightarrow_{\mathsf{CBN_{KK}}} N$ and $\langle V \dagger x.N\rangle \longrightarrow_{\mathsf{CBV_{KK}}} N$.

$$\mathsf{left}_2 \quad \langle\langle \lambda z.M \dagger y.y[P, z'.Q]\rangle \dagger x.N\rangle \longrightarrow \langle \lambda z.M \dagger y.y[P, z'.\langle Q \dagger x.N\rangle]\rangle \quad \text{if } y \notin \mathsf{FV}(P) \cup \mathsf{FV}(Q)$$

$$\mathsf{right}_5 \quad \langle N \dagger x.\langle \lambda z.M \dagger y.y[P, z'.Q]\rangle\rangle \longrightarrow \langle \lambda z.\langle N \dagger x.M\rangle \dagger y.y[\langle N \dagger x.P\rangle, z'.\langle N \dagger x.Q\rangle]\rangle \quad \text{if } y \notin \mathsf{FV}(P) \cup \mathsf{FV}(Q)$$

Figure 6.4: Additional rules for KK-propagation

Lemma 148

1. *All cuts in $\mathcal{P}r(M)$ and $\mathcal{P}r_{x.N}(M_1\ M_2)$ are logical (provided all cuts in N are). This is a major difference with Gentzen's encoding.*

2. *If $N \longrightarrow_{\mathsf{CBN}_{\mathsf{KKs}}} N'$ then $\mathcal{P}r_{x.N}(M_1\ M_2) \longrightarrow_{\mathsf{CBN}_{\mathsf{KKs}}} \mathcal{P}r_{x.N'}(M_1\ M_2)$ and if $N \longrightarrow_{\mathsf{CBV}_{\mathsf{KKs}}} N'$ then $\mathcal{P}r_{x.N}(M_1\ M_2) \longrightarrow_{\mathsf{CBV}_{\mathsf{KKs}}} \mathcal{P}r_{x.N'}(M_1\ M_2)$.*

3. *If N is a y-covalue, then $\langle \mathcal{P}r_{x.N'}(M_1\ M_2) \dagger y.N\rangle \longrightarrow^*_{\mathsf{CBN}_{\mathsf{KKs}}} \mathcal{P}r_{x.\langle N' \dagger y.N\rangle}(M_1\ M_2)$ and $\langle \mathcal{P}r(M_1\ M_2) \dagger y.N\rangle \longrightarrow^*_{\mathsf{CBN}_{\mathsf{KKs}}} \mathcal{P}r_{y.N}(M_1\ M_2)$.*

4. *We then have $\langle \mathcal{P}r(M') \dagger x.\mathcal{P}r(M)\rangle \longrightarrow^*_{\mathsf{CBN}_{\mathsf{KKs}}} \mathcal{P}r(\{^{M'}/_x\}M)$ and $\langle \mathcal{P}r(V) \dagger x.\mathcal{P}r(M)\rangle \longrightarrow^*_{\mathsf{CBV}_{\mathsf{KKs}}} \mathcal{P}r(\{^{V}/_x\}M)$.*

Proof: Each of the above points is obtained by straightforward inductions on M and $M_1\ M_2$. For the last point the induction also requires the auxiliary property that if all cut-constructors in N are logical, the following holds:
if $x \in \mathsf{FV}(N)$, then
$\langle \mathcal{P}r(M') \dagger x.\mathcal{P}r_{y.N}(M_1\ M_2)\rangle \longrightarrow^*_{\mathsf{CBN}_{\mathsf{KKs}}} \mathcal{P}r_{y.\langle \mathcal{P}r(M') \dagger x.N\rangle}(\{^{M'}/_x\}(M_1\ M_2))$ and
$\langle \mathcal{P}r(V) \dagger x.\mathcal{P}r_{y.N}(M_1\ M_2)\rangle \longrightarrow^*_{\mathsf{CBV}_{\mathsf{KKs}}} \mathcal{P}r_{y.\langle \mathcal{P}r(V) \dagger x.N\rangle}(\{^{V}/_x\}(M_1\ M_2))$
otherwise, $\langle \mathcal{P}r(M') \dagger x.\mathcal{P}r_{y.N}(M_1\ M_2)\rangle \longrightarrow^*_{\mathsf{CBN}_{\mathsf{KKs}}} \mathcal{P}r_{y.N}(\{^{M'}/_x\}(M_1\ M_2))$ and
$\langle \mathcal{P}r(V) \dagger x.\mathcal{P}r_{y.N}(M_1\ M_2)\rangle \longrightarrow^*_{\mathsf{CBV}_{\mathsf{KKs}}} \mathcal{P}r_{y.N}(\{^{V}/_x\}(M_1\ M_2))$. □

Theorem 149 (Simulation of β-reduction)
$\longrightarrow_{\mathsf{KKs}}$ strongly simulates $\longrightarrow_\beta$ through Prawitz's translation. More precisely,

1. *If $M \longrightarrow_\beta M'$ then $\mathcal{P}r(M) \longrightarrow^+_{\mathsf{CBN}_{\mathsf{KKs}}} \mathcal{P}r(M')$.*

2. *If $M \longrightarrow_{\beta_{\mathsf{V}}} M'$ then $\mathcal{P}r(M) \longrightarrow^+_{\mathsf{CBV}_{\mathsf{KKs}}} \mathcal{P}r(M')$.*

where β_{V} is the reduction rule of the λ_{V}-calculus (see Chapter 3).

Proof: By induction on the derivation of the reduction step, using the lemma above. If $M \longrightarrow_\beta M'$ then $\mathcal{P}r(M) \longrightarrow^+_{\mathsf{CBN}_{\mathsf{KKs}}} \mathcal{P}r(M')$ and if $M_1\ M_2 \longrightarrow_\beta M_1'\ M_2'$ $\mathcal{P}r_{y.N}(M_1\ M_2) \longrightarrow^+_{\mathsf{CBN}_{\mathsf{KKs}}} \mathcal{P}r_{y.N}(M_1'\ M_2')$, which are proved by induction on the derivation step, using the above lemma. This is a minor variant of the proof in [Kik06]. Point 2 is proved similarly. □

The reduction relation $\longrightarrow_{\mathsf{KK}}$ still satisfies Property 141.1 because it extends $\longrightarrow_{\mathsf{SI}}$. For Property 141.2 (subject reduction), it suffices to check the two new rules, which is straightforward.

As for Property 141.3, we only conjecture the strong normalisation of typed terms:

Conjecture 150 (Strong Normalisation) *If* $\Gamma \vdash_{\lambda G3} M : A$ *then* $M \in \mathcal{SN}^{KK}$.

Since the calculus simulates β-reduction, proving this conjecture is at least as hard as the strong normalisation of the simply-typed λ-calculus. However what we *can* do now is prove the strong normalisation of the system without rule B: it suffices to refine the first encoding from the proof of Theorem 142 as shown in Fig. 6.5.

$\overline{x}$	$:=$	$\star$	
$\overline{\lambda x.M}$	$:=$	$\mathsf{i}(\overline{M})$	
$\overline{x[N, y.M]}$	$:=$	$\mathsf{ii}(\overline{N}, \overline{M})$	
$\overline{\langle N \dagger y.M\rangle}$	$:=$	$\mathsf{cut}(\overline{N}, \overline{M})$	if M is a y-covalue
$\overline{\langle N \dagger y.M\rangle}$	$:=$	$\mathsf{sub}(\overline{N}, \overline{M})$	otherwise

Figure 6.5: Encoding of $\lambda\mathsf{G3}$ into a first-order syntax

Lemma 151 *If* $M \longrightarrow_{KK\setminus B} N$ *then* $\overline{M} \gg \overline{N}$. *Hence,* $\longrightarrow_{KK\setminus B}$ *is terminating.*

Proof: It suffices to check all the rules. □

The conjecture above is motivated by the fact that in rules left_2 and right_5, the outer cut, which is not principal, is pushed through a principal cut. In other words, the property for a cut of being principal is preserved by reduction and can be used as a flag (as in Fig. 6.5) whose state can only evolve in one direction.

Note that the simulation works with Prawitz's encoding because cuts are only used to encode β-redexes and are thus principal. In Gentzen's encoding where a potentially non-principal cut-constructor encodes each application, the simulation fails. In fact, the original system of [Kik06] is rather like the rules below (although the first one is restricted to the case when N is an x-covalue

different from x):

$$\begin{array}{|ll|}\hline \langle\langle M \dagger y.P\rangle \dagger x.N\rangle & \longrightarrow \langle M \dagger y.\langle P \dagger x.N\rangle\rangle \\ & \text{if } \langle M \dagger y.P\rangle \text{ is a principal cut-constructor} \\ & \\ \langle N \dagger x.\langle M \dagger y.P\rangle\rangle & \longrightarrow \langle\langle N \dagger x.M\rangle \dagger y.\langle N \dagger x.P\rangle\rangle \\ & \text{if } \langle M \dagger y.P\rangle \text{ is a principal cut-constructor} \\ \hline \end{array}$$

Those rules are simpler, and in case we consider other connectives than implication, they suffice, otherwise we would need as many rules left_2 and right_5 as connectives, i.e. one for each pair of dual constructors for the left and right introduction.

However, the cut-constructor at the root of the right-hand side of the rules above is no longer a principal cut-constructor. It could become one again if the cut-constructor that has come in-between were pushed one step further (as in left_2 and right_5). Hence, in presence of non-determinism, a change of strategy can occur precisely after applying the above rules, so this version seems more difficult to prove strongly normalising than the previous one (however, the restriction of [Kik06] about N in the first rule above might reduce the difficulty of proving strong normalisation).

Again, we conjecture the confluence of the CBV- and CBN-reduction.

Conjecture 152 (Confluence) *Both* $\longrightarrow_{CBN_{KK}}$ *and* $\longrightarrow_{CBV_{KK}}$ *are confluent.*

And again we could either try the method of parallel reduction (see e.g. [Tak89]) or establish that a CPS-translation forms a pre-Galois connection with a confluent fragment of λ-calculus (as in Chapter 3).

Lemma 153
$\langle x \dagger y.M\rangle \longleftrightarrow^*_{CBV_{KKs}} \{^x\!/_y\}M$ *(resp.* $\langle x \dagger y.M\rangle \longleftrightarrow^*_{CBN_{KKs}} \{^x\!/_y\}M$*) and*
$\langle M \dagger y.y\rangle \longleftrightarrow^*_{CBV_{KKs}} M$ *(resp.* $\langle M \dagger y.y\rangle \longleftrightarrow^*_{CBN_{KKs}} M$*).*

Proof: We first prove it in the case when M is a normal form for system $\longrightarrow_{\mathsf{KK}\setminus B}$, that is to say, when all its cut-constructors are logical. With the two new rules left_2 and right_5 of system KK, the above terms are all redexes of $\mathsf{KK}\setminus\mathsf{B}$ (this is why this theorem might not hold for system SI), hence we can prove this by induction on M.

Then for an arbitrary M we first reduce it, using Lemma 151, to a term M' that is a normal form for $\longrightarrow_{\mathsf{KK}\setminus\mathsf{B}}$, and for which the statement holds. □

Corollary 154 (Equational theories)

1. $(\longleftrightarrow^*_{CBV_{SI}}) \subseteq (\longleftrightarrow^*_{CBV_{KK}})$ *and* $(\longleftrightarrow^*_{CBN_{SI}}) \subseteq (\longleftrightarrow^*_{CBN_{KK}})$.
2. $(\longleftrightarrow^*_{CBV_{KKs}}) = (\longleftrightarrow^*_{CBV_{KK}})$ *and* $(\longleftrightarrow^*_{CBN_{KKs}}) = (\longleftrightarrow^*_{CBN_{KK}})$.

Proof: The first point is straightforward. The second is a consequence of the above lemma. □

Remark 155 Note that on terms that are in $\mathsf{SN}^{\mathsf{SI}}$, we can infer from Lemma 146 that $(\longleftarrow^*_{\mathsf{CBV}_{\mathsf{KK}}}) = (\longleftarrow^*_{\mathsf{CBV}_{\mathsf{SI}}})$ and $(\longleftarrow^*_{\mathsf{CBN}_{\mathsf{KK}}}) = (\longleftarrow^*_{\mathsf{CBN}_{\mathsf{SI}}})$.

The simulation of λ-calculus only works with Prawitz's encoding because only logical cuts lie in the encoding. We shall also see that Prawitz's encoding will need to be modified to be adapted to the call-by-value discipline, in a way that creates non-principal cuts. Hence, we shall need propagation systems more powerful than KK that allow propagation of cuts through *any* kind of cut.

Urban's jumping cuts - System JC

One of the purposes of this chapter being to relate cut-elimination to normalisation in λ-calculus, we present an alternative (more powerful) propagation system. This idea comes from Christian Urban [Urb00]: a cut "jumps" to the places where the cut-formula is principal, as shown in the right-propagation system of Fig 6.6.

left	$\langle N \dagger x.M\rangle$	$\longrightarrow$	$\{N \not\nearrow x.M\}$	if N is not a value
right	$\langle N \dagger x.M\rangle$	$\longrightarrow$	$\left\{N \nwarrow\!\!\!\not\; x.M\right\}$	if M is not an x-covalue

where $\{N \not\nearrow x.M\}$ and $\left\{N \nwarrow\!\!\!\not\; x.M\right\}$ are constructions defined as follows:

$\{y \not\nearrow x.N\}$	:=	$\{{}^{y}\!/_{x}\}N$
$\{\lambda y.M \not\nearrow x.N\}$	:=	$\langle \lambda y.M \dagger x.N\rangle$
$\{z[M, y.P] \not\nearrow x.N\}$	:=	$z[M, y.\{P \not\nearrow x.N\}]$
$\{\langle M \dagger y.P\rangle \not\nearrow x.N\}$	:=	$\langle M \dagger y.\{P \not\nearrow x.N\}\rangle$
$\left\{N \nwarrow\!\!\!\not\; x.x\right\}$	:=	N
$\left\{N \nwarrow\!\!\!\not\; x.y\right\}$	:=	y
$\left\{N \nwarrow\!\!\!\not\; x.(\lambda y.M)\right\}$	:=	$\lambda y.\left\{N \nwarrow\!\!\!\not\; x.M\right\}$
$\left\{N \nwarrow\!\!\!\not\; x.x[M, z.P]\right\}$	:=	$\left\langle N \dagger x.x[\left\{N \nwarrow\!\!\!\not\; x.M\right\}, z.\left\{N \nwarrow\!\!\!\not\; x.P\right\}]\right\rangle$
$\left\{N \nwarrow\!\!\!\not\; x.x'[M, z.P]\right\}$	:=	$x'[\left\{N \nwarrow\!\!\!\not\; x.M\right\}, z.\left\{N \nwarrow\!\!\!\not\; x.P\right\}]$
$\left\{N \nwarrow\!\!\!\not\; x.\langle M \dagger y.P\rangle\right\}$	:=	$\left\langle\left\{N \nwarrow\!\!\!\not\; x.M\right\} \dagger y.\left\{N \nwarrow\!\!\!\not\; x.P\right\}\right\rangle$

Figure 6.6: JC-propagation

Alternatively, one could introduce $\{\_ \not\nearrow \_.\_\}$ and $\left\{\_ \nwarrow\!\!\!\not\; \_.\_\right\}$ as constructors of the syntax of proof-terms (with the same typing rule as that of $\langle\_ \dagger \_.\_\rangle$), and turn the above definitions into sets of rewrite rules, oriented from left to right, that eliminate the two constructors.

The rewrite rule corresponding to the last line of each definition would then not be needed for completeness of the cut-elimination process, as the inner cut could be eliminated first. However, this commutation of cuts is precisely what

makes the simulation of β-reduction possible (as in system KK), as does the version written above, with $\{\_ \nearrow \_.\_\}$ and $\{\_ \nwarrow \_.\_\}$ considered as constructions, rather than constructors. Indeed, both versions can reduce the external substitution in $\langle N \dagger x.\langle P \dagger y.R\rangle\rangle$ while the simple system cannot.

Although the version with the explicit operator is also proved terminating on typed terms, we shall stick to the version presented above, as it avoids extending the syntax and more closely relates to λ-calculus, which uses the construction of substitution.

Theorem 156 (Strong Normalisation)

1. $\longrightarrow_{\mathsf{JC}\setminus\mathsf{B}}$ *is strongly normalising.*
2. *If* $\Gamma \vdash_{\lambda G3} M : A$ *then* $M \in \mathsf{SN}^{\mathsf{JC}}$.

Proof: This is simply the intuitionistic restriction of the system of [Urb00], which is proved strongly normalising. □

Again, we conjecture the confluence of the CBV- and CBN-reduction.

Conjecture 157 (Confluence) *Both* $\longrightarrow_{\mathsf{CBN}_{\mathsf{JC}}}$ *and* $\longrightarrow_{\mathsf{CBV}_{\mathsf{JC}}}$ *are confluent.*

And again we could either try the method of parallel reduction (see e.g. [Tak89]) or establish that a CPS-translation forms a pre-Galois connection with a confluent fragment of λ-calculus (as in Chapter 3).

Again, we show that the enhanced cut-elimination is consistent with the simple one in the following sense:

Theorem 158 (Equational theories 1)
$(\longleftrightarrow^*_{\mathsf{CBV}_{\mathsf{JCs}}}) = (\longleftrightarrow^*_{\mathsf{CBV}_{\mathsf{JC}}})$ *and* $(\longleftrightarrow^*_{\mathsf{CBN}_{\mathsf{JCs}}}) = (\longleftrightarrow^*_{\mathsf{CBN}_{\mathsf{JC}}})$

Proof: System JCs simulates system JC. □

Theorem 159 (Equational theories 2)

1. *We have*
$$(\longleftrightarrow^*_{\mathsf{CBV}_{\mathsf{KK}}}) = (\longleftrightarrow^*_{\mathsf{CBV}_{\mathsf{JC}}})$$
$$(\longleftrightarrow^*_{\mathsf{CBN}_{\mathsf{KK}}}) = (\longleftrightarrow^*_{\mathsf{CBN}_{\mathsf{JC}}})$$
2. *Note that on terms that are in* $\mathsf{SN}^{\mathsf{SI}}$,
$$(\longleftrightarrow^*_{\mathsf{CBV}_{\mathsf{SI}}}) = (\longleftrightarrow^*_{\mathsf{CBV}_{\mathsf{JC}}})$$
$$(\longleftrightarrow^*_{\mathsf{CBN}_{\mathsf{SI}}}) = (\longleftrightarrow^*_{\mathsf{CBN}_{\mathsf{JC}}})$$

Proof: System $\mathsf{KK} \setminus \mathsf{B}$ can reduce any term to a term in which all cut-constructors are principal. Now for terms that satisfy this property, rules left_2 and right_5 of system KK are just as powerful as the reduction in JC that use $\{\_ \nearrow \_.\_\}$ and $\{\_ \nwarrow \_.\_\}$. The second point is then a corollary of Remark 155. □

6.2 T-restriction & LJT

6.2.1 A fragment of λG3

We call t-restriction of λG3 the fragment of CBN-pure terms defined as follows:

Definition 101 (CBN-purity) A term is *CBN-pure* if in any sub-term of the form $x[M, y.N]$, N is a y-co-value.

Theorem 160 (Preservation of CBN-purity) *CBN-reduction preserve CBN-purity.*

Proof: Easy check on the rules. □

Now we prove that the t-restriction is logically complete. We show that any proof can be transformed into a CBN-pure term, and we use for that the following purification rule:

$$(\mathsf{CBN} - pur) \quad x[M, y.N] \longrightarrow \langle x[M, z.z] \dagger y.N \rangle \qquad \text{if } N \text{ is not a } y\text{-covalue}$$

Note that this rule satisfies the subject reduction property. It also terminates, simply because every application of this rule deceases the number of sub-terms of the form $x[M, y.N]$ with N not a y-covalue.

Theorem 161 (Prawitz's encoding produces CBN-pure terms) *Prawitz's encoding only produces CBN-pure terms of λG3, that is, for every λ-term t, $\mathcal{P}r(t)$ is CBN-pure.*

Proof: This is proved by induction on t, together with the fact that if N is an x-covalue that is CBN-pure, then $\mathcal{P}r_{x.N}(t_1\ t_2)$ is CBN-pure. □

This is an important remark since it already suggests a strong connection between the t-fragment and λ-calculus.

From a logical point of view, the t-restriction corresponds to the sequent calculus LJT, as defined for instance in [Her95]. The t-restriction can also be expressed by separating explicitly the covalues from the other terms, with a syntactic category for covalues, that is to say, for x-covalues, abstracting on x:

$$\begin{array}{ll} M, N, P & ::= \lambda x.M \mid x\ l \mid \langle M \dagger x.N \rangle \\ l & ::= [] \mid M \cdot l \end{array}$$

The constructor [] can be seen as representing the higher-order term $x.x$ and $M \cdot l$ can be seen as representing $x.x[M, y.N]$ if l represents $y.N$ with $x \notin \mathsf{FV}(M) \cup \mathsf{FV}(N)$.

The reduction rules are simply inherited from the CBN-reductions of λG3. The typing rules are also inherited from λG3, as shown in Figure 6.7.

$$\frac{}{\Gamma; A \vdash [] : A} \qquad \frac{\Gamma \vdash M : A \quad \Gamma, x : A \vdash N : B}{\Gamma \vdash \langle M \dagger x.N \rangle : B}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma; B \vdash l : C}{\Gamma; A \to B \vdash M \cdot l : C} \qquad \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}$$

$$\frac{\Gamma, x : A; A \vdash l : B}{\Gamma, x : A \vdash x\ l : B}$$

Figure 6.7: LJT

6.2.2 The $\overline{\lambda}$-calculus

This calculus is (almost) that of [Her95] called $\overline{\lambda}$, whose syntax turns the constructions $\{\_ \nearrow \_ . \_\}$ and $\{\_ \nwarrow \_ . \_\}$ into constructors $\_@\_$ and $\langle \_ / \_ \rangle \_$, respectively. The syntax of $\overline{\lambda}$ is thus:

Definition 102 (Syntax of $\overline{\lambda}$)

$$\begin{array}{rcl} M, N & ::= & \lambda x.M \mid x\ l \mid M\ l \mid \langle M/x \rangle N \\ l, l' & ::= & [] \mid M \cdot l \mid l@l' \mid \langle M/x \rangle l \end{array}$$

$\lambda x.M$ and $\langle N/x \rangle M$ bind x in M, and $\langle M/x \rangle l$ binds x in l.

The reduction rules of $\overline{\lambda}$ are defined in Figure 6.8, the typing rules are defined in Figure 6.9. They inductively define the derivability of three kinds of sequents: some of the form $\Gamma \vdash M : A$ and some of the form $\Gamma; B \vdash l : A$. In the latter case, B is said to be in the *stoup* of the sequent, according to a terminology due to Girard. Derivability in $\overline{\lambda}$ of the two kinds of sequents is denoted $\Gamma \vdash_{\overline{\lambda}} M : A$, and $\Gamma; B \vdash_{\overline{\lambda}} l : A$, respectively.

Many variants of $\overline{\lambda}$ can be defined (such as the one in Fig. 6.7), depending on whether we prefer constructors or constructions, in particular whether we consider explicit or implicit substitutions. A comprehensive study of the variants for the t-restriction can be found in [Esp02].

6.2.3 A reflection of (CBN) λ-calculus

In this section we establish a strong connection between $\overline{\lambda}$ (or the t-fragment of λG3) and λ-calculus, whose unrestricted notion of computation can be considered as the CBN-λ-calculus.

Indeed, the example of $\overline{\lambda}$ is a typical case where the syntax does not include that of λ-calculus, but the latter can be encoded in it, since Prawitz's encoding

$$\mathsf{B}\quad (\lambda x.M)\,(N\cdot l) \longrightarrow (\langle N/x\rangle M)\,l$$

System x:

$$\begin{array}{lll}
\mathsf{B1} & M\,[] & \longrightarrow M\\
\mathsf{B2} & (x\,l)\,l' & \longrightarrow x\,(l@l')\\
\mathsf{B3} & (M\,l)\,l' & \longrightarrow M\,(l@l')\\
\\
\mathsf{A1} & (M\cdot l')@l & \longrightarrow M\cdot(l'@l)\\
\mathsf{A2} & []@l & \longrightarrow l\\
\mathsf{A3} & (l@l')@l'' & \longrightarrow l@(l'@l'')\\
\\
\mathsf{C1} & \langle P/y\rangle\lambda x.M & \longrightarrow \lambda x.\langle P/y\rangle M\\
\mathsf{C2} & \langle P/y\rangle(y\,l) & \longrightarrow P\,\langle P/y\rangle l\\
\mathsf{C3} & \langle P/y\rangle(x\,l) & \longrightarrow x\,\langle P/y\rangle l\\
\mathsf{C4} & \langle P/y\rangle(M\,l) & \longrightarrow \langle P/y\rangle M\,\langle P/y\rangle l\\
\\
\mathsf{D1} & \langle P/y\rangle[] & \longrightarrow []\\
\mathsf{D2} & \langle P/y\rangle(M\cdot l) & \longrightarrow (\langle P/y\rangle M)\cdot(\langle P/y\rangle l)\\
\mathsf{D3} & \langle P/y\rangle(l@l') & \longrightarrow (\langle P/y\rangle l)@(\langle P/y\rangle l')
\end{array}$$

Figure 6.8: Reduction rules for $\overline{\lambda}$

$$\frac{\Gamma;A\vdash l:B\quad (x:A)\in\Gamma}{\Gamma\vdash x\,l:B}\,\mathsf{select}_x \qquad \frac{}{\Gamma;A\vdash []:A}\,\mathsf{ax}$$

$$\frac{\Gamma,x:A\vdash M:B}{\Gamma\vdash\lambda x.M:A\to B}\to\mathsf{r} \qquad \frac{\Gamma\vdash M:A\qquad \Gamma;B\vdash l:C}{\Gamma;A\to B\vdash M\cdot l:C}\to\mathsf{l}$$

$$\frac{\Gamma\vdash M:A\qquad \Gamma;A\vdash l:B}{\Gamma\vdash M\,l:B}\,\mathsf{cut}_3 \qquad \frac{\Gamma;C\vdash l':A\qquad \Gamma;A\vdash l:B}{\Gamma;C\vdash l'@l:B}\,\mathsf{cut}_1$$

$$\frac{\Gamma\vdash P:A\qquad \Gamma,x:A\vdash M:C}{\Gamma\vdash\langle P/x\rangle M:C}\,\mathsf{cut}_4 \qquad \frac{\Gamma\vdash P:A\qquad \Gamma,x:A;B\vdash l:C}{\Gamma;B\vdash\langle P/x\rangle l:C}\,\mathsf{cut}_2$$

Figure 6.9: Typing rules for $\overline{\lambda}$

only produces CBN-pure terms of λG3 (Theorem 161). Hence we can reformulate the latter with the syntax of $\overline{\lambda}$, we do in Figure 6.10.

Fig. 6.11 reformulates, in the case of $\overline{\lambda}$, Gentzen's encoding from λG3 to λ-calculus (see Chapter 2).

Now we show that $\mathcal{B}$ and $\mathcal{A}$ form a reflection in $\overline{\lambda}$ of λ-calculus. We first prove the following results:

$\mathcal{A}(\lambda x^T.t)$	$:=$	$\lambda x^{\mathcal{A}(T)}.\mathcal{A}(t)$	
$\mathcal{A}(t)$	$:=$	$\mathcal{A}_{[]}(t)$	otherwise
$\mathcal{A}_l(t\ u)$	$:=$	$\mathcal{A}_{\mathcal{A}(u)\cdot l}(t)$	
$\mathcal{A}_l(x)$	$:=$	$x\ l$	
$\mathcal{A}_l(t)$	$:=$	$\mathcal{A}(t)\ l$	otherwise

Figure 6.10: From λ-calculus to $\overline{\lambda}$

$\mathcal{B}(\lambda x^A.M)$	$:=$	$\lambda x^{\mathcal{B}(A)}.\mathcal{B}(M)$
$\mathcal{B}(x\ l)$	$:=$	$\{x/z\}\mathcal{B}^z(l)$
$\mathcal{B}(M\ l)$	$:=$	$\{\mathcal{B}(M)/z\}\mathcal{B}^z(l)$
$\mathcal{B}(\langle P/x\rangle M)$	$:=$	$\{\mathcal{B}(P)/x\}\mathcal{B}(M)$
$\mathcal{B}^y([])$	$:=$	y
$\mathcal{B}^y(M\cdot l)$	$:=$	$\{y\ \mathcal{B}(M)/z\}\mathcal{B}^z(l)$
$\mathcal{B}^y(l@l')$	$:=$	$\{\mathcal{B}^y(l)/z\}\mathcal{B}^z(l')$
$\mathcal{B}^y(\langle P/x\rangle l)$	$:=$	$\{\mathcal{B}(P)/x\}\mathcal{B}^y(l)$

Figure 6.11: From $\overline{\lambda}$ to λ-calculus

Lemma 162

1. *$\mathcal{A}(t)$ and $\mathcal{A}_l(t)$ are always x-normal forms (provided l is).*
2. *If $l \longrightarrow_{\mathsf{Bx}} l'$ then $\mathcal{A}_l(t) \longrightarrow_{\mathsf{Bx}} \mathcal{A}_{l'}(t)$.*
3. *$\mathcal{A}_{l'}(t)\ l \longrightarrow_{\mathsf{x}}^* \mathcal{A}_{l'@l}(t)$ and $\mathcal{A}(t)\ l \longrightarrow_{\mathsf{x}}^* \mathcal{A}_l(t)$.*
4. *$\langle\mathcal{A}(u)/x\rangle\mathcal{A}(t) \longrightarrow_{\mathsf{x}}^* \mathcal{A}(\{u/x\}t)$ and $\langle\mathcal{A}(u)/x\rangle\mathcal{A}_l(t) \longrightarrow_{\mathsf{x}}^* \mathcal{A}_{\langle\mathcal{A}(u)/x\rangle l}(\{u/x\}t)$.*

Proof: Each of the above points is obtained by straightforward inductions on t. □

Now we study the composition of the two encodings:

Lemma 163 *Suppose M and l are x-normal forms.*

1. *If $t = x$ or $t = t_1\ t_2$ or $l \neq []$, then $\mathcal{A}_l(t) = \mathcal{A}(\{t/x\}\mathcal{B}^x(l))$ if $x \notin \mathsf{FV}(l)$.*
2. *$M = \mathcal{A}(\mathcal{B}(M))$.*

Proof: By simultaneous induction on l and M. □

Theorem 164 (A reflection of λ-calculus in $\overline{\lambda}$)

1. $\longrightarrow_{Bx}$ *strongly simulates* $\longrightarrow_{\beta}$ *through* $\mathcal{A}$.
2. $\mathcal{B}$ *and* $\mathcal{A}$ *form a reflection in* $\overline{\lambda}$ *of* λ*-calculus.*

Proof:

1. If $t \longrightarrow_{\beta} u$ then $\mathcal{A}(t) \longrightarrow^{+}_{\mathsf{Bx}} \mathcal{A}(u)$ and $\mathcal{A}_l(t) \longrightarrow^{+}_{\mathsf{Bx}} \mathcal{A}_l(u)$, which are proved by induction on the derivation step, using Lemma 162.4 for the base case and Lemma 162.3.
2.
   - The first simulation is given by point 1.
   - If $M \longrightarrow_{\mathsf{B}} N$ then $\mathcal{B}(M) \longrightarrow^{*}_{\beta} \mathcal{B}(N)$, if $l \longrightarrow_{\mathsf{B}} l'$ then $\mathcal{B}^y(l) \longrightarrow^{*}_{\beta} \mathcal{B}^y(l')$, if $M \longrightarrow_{\mathsf{x}} N$ then $\mathcal{B}(M) = \mathcal{B}(N)$ and if $l \longrightarrow_{\mathsf{x}} l'$ then $\mathcal{B}^y(l) = \mathcal{B}^y(l')$, which are proved by simultaneous induction on the derivation step and case analysis.
   - $M \longrightarrow^{*}_{\mathsf{x}} \mathcal{A}(\mathcal{B}(M))$ holds by induction in SN^{x} (because x is terminating): by Lemma 163.2 it holds if M is an x-normal form, and if $M \longrightarrow_{\mathsf{x}} N$ then we can apply the induction hypothesis on N and by point 2 we have $\mathcal{B}(M) = \mathcal{B}(N)$.
   - $\mathcal{B}(\mathcal{A}(t)) = t$ and $\mathcal{B}(\mathcal{A}_l(t)) = \{t/x\}\mathcal{B}^x(l)$ (with $x \neq \mathsf{FV}(l)$) are obtained by simultaneous induction on t.

□

Now we use Theorem 164 to prove the confluence of $\overline{\lambda}$and the equivalence of the equational theories.

Corollary 165 (Confluence) $\longrightarrow_{x}$ *and* $\longrightarrow_{Bx}$ *are confluent.*

Proof: From Theorems 5 and 164. □

Corollary 166 (Equational theories)

1. $t \longleftrightarrow^{*}_{\beta} u$ *if and only if* $\mathcal{A}(t) \longleftrightarrow^{*}_{Bx} \mathcal{A}(u)$.
2. $M \longleftrightarrow^{*}_{Bx} N$ *if and only if* $\mathcal{B}(M) \longleftrightarrow^{*}_{\beta} \mathcal{B}(N)$.

From the reflection of λ-calculus we also get an intuitive and functional interpretation of the $\overline{\lambda}$-calculus:

Notice that Prawitz's encoding, producing only CBN-pure terms, is based on a mechanism close to that of stack-based abstract machines such as in [Kri]: arguments of functions (e.g. values) are recursively stored in a *stack* or *list*, represented by the parameter of the encoding. Expressing Prawitz's encoding with $\overline{\lambda}$ clarifies the notion of list as follows: lists are those terms of the second syntactic

category of Definition 102. They are used to represent series of arguments of a function, the terms $x\ l$ (resp. $M\ l$) representing the application of x (resp. M) to the list of arguments l. Note that a variable alone is not a term, it has to be applied to a list, possibly the empty list, denoted []. The list with head M and tail l is denoted $M \cdot l$, with a typing rule corresponding to the left-introduction of implication. Successive applications give rise to the concatenation of lists, denoted $l@l'$, and $\langle M/x\rangle N$ and $\langle M/x\rangle l$ are explicit substitution operators on terms and lists, respectively. They are used to describe explicitly the interaction between the constructors in the normalisation process, adapted from [Her95, DU03]. More intuition about $\overline{\lambda}$, its syntax and operational semantics is given in [Her95].

6.2.4 Normalisation results in $\overline{\lambda}$

The $\overline{\lambda}$-calculus is also an example of example of how the safeness and minimality technique applies to prove PSN, with a proof shorter than those of [DU03, Kik04a].

Since $\overline{\lambda}$ can be typed by a version called LJT of the intuitionistic sequent calculus and the technique provides again a type-preserving encoding of $\overline{\lambda}$ into the simply-typed λ-calculus, we thus prove the strong normalisation of cut-elimination in LJT.

Now we prove PSN (and strong normalisation of typed terms) for $\overline{\lambda}$ with the safeness and minimality technique. Again, we consider a first-order syntax equipped with a LPO based on the following precedence:

$$\mathsf{sub}(\_,\_) \succ \mathsf{ii}(\_,\_) \succ \mathsf{i}(\_) \succ \mathsf{c}^M$$

where for every $M \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$ (resp. $l \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$) there is a constant c^M (resp. c^l). Those constants are all below $\mathsf{i}()$, and the precedence between them is given by $\mathsf{c}^M \succ \mathsf{c}^N$ if $M \longrightarrow^{+}_{B,\mathsf{x}} N$ or $M \sqsupset N$ (and similarly for lists). The precedence relation is thus terminating.

The encoding is presented in Fig. 6.12.

Lemma 167

1. *If* $M \longrightarrow_{\mathsf{safeB},\mathsf{x}} N$ *then* $\overline{M} \gg \overline{N}$.
2. *If* $l \longrightarrow_{\mathsf{safeB},\mathsf{x}} l'$ *then* $\overline{l} \gg \overline{l'}$.

Proof: We first check root reductions.
Clearly, if $M, l \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$ the Lemma holds, and this covers the case of safe reductions.
Also, when $N, l' \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$ the Lemma holds as well.
The remaining cases are when $\overline{M}, \overline{l}$ and $\overline{N}, \overline{l'}$ are not constants.

For B1, A2, the term $\overline{N}$ (resp. $\overline{l'}$) is a sub-term of $\overline{M}$ (resp. $\overline{l}$).

For B2, B3, A1, the arguments of $\mathsf{ii}(,)$ decrease in the lexicographic order.

$\overline{M}$	$=$	c^M	if $M \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$
otherwise			
$\overline{\lambda x.M}$	$=$	$\mathsf{ii}(\overline{A}, \overline{M})$	
$\overline{x\ l}$	$=$	$\mathsf{i}(\bar{l})$	
$\overline{M\ l}$	$=$	$\mathsf{ii}(\bar{l}, \overline{M})$	
$\overline{\langle M/x\rangle N}$	$=$	$\mathsf{sub}(\overline{M}, \overline{N})$	
$\bar{l}$	$=$	c^l	if $l \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$
otherwise			
$\overline{M \cdot l}$	$=$	$\mathsf{ii}(\overline{M}, \bar{l})$	
$\overline{l@l'}$	$=$	$\mathsf{ii}(\bar{l}, \overline{l'})$	
$\overline{\langle M/x\rangle l}$	$=$	$\mathsf{sub}(\overline{M}, \bar{l})$	

Figure 6.12: Encoding of $\overline{\lambda}$ into a first-order syntax

For Cs, Ds, the symbol at the root of $\overline{N}$ (resp. $\overline{l'}$) is strictly inferior to that of $\overline{M}$ (resp. $\bar{l}$), so we only have to check that the direct sub-terms of $\overline{N}$ (resp. $\overline{l'}$) are smaller than $\overline{M}$ (resp. $\bar{l}$). Clearly, it is the case for all sub-terms that are constants (namely, those encodings of strongly normalising sub-terms of N or l'). For those that are not, it is a routine check on every rule.

The contextual closure is a straightforward induction on M, l:
Again, if $M, l \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$ or $N, l' \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$, the Lemma holds;
otherwise, if the reduction is a $\mathsf{safe}B, \mathsf{x}$-reduction in a direct sub-term of M or l, it suffices to use the induction hypothesis on that sub-term. □

Corollary 168 *The reduction relation* $\longrightarrow_{\mathsf{safe}B,\mathsf{x}}$ *is terminating.*

Now we slightly modify the encoding of $\overline{\lambda}$ into λ-calculus as presented in Fig. 6.13.

$\mathcal{B}(\lambda x.M)$	$=$	$\lambda x.\mathcal{B}(M)$	
$\mathcal{B}(x\ l)$	$=$	$\{x/z\}\mathcal{B}^z(l)$	
$\mathcal{B}(M\ l)$	$=$	$\{\mathcal{B}(M)/z\}\mathcal{B}^z(l)$	
$\mathcal{B}(\langle M/x\rangle N)$	$=$	$\{\mathcal{B}(M)/x\}\mathcal{B}(N)$	if $M \in \mathsf{SN}^{\mathsf{B},\mathsf{x}}$
$\mathcal{B}(\langle M/x\rangle N)$	$=$	$(\lambda x.\mathcal{B}(N))\ \mathcal{B}(M)$	if $M \notin \mathsf{SN}^{\mathsf{B},\mathsf{x}}$
$\mathcal{B}^y([])$	$=$	y	
$\mathcal{B}^y(M \cdot l)$	$=$	$\{y\ \mathcal{B}(M)/z\}\mathcal{B}^z(l)$	
$\mathcal{B}^y(l@l')$	$=$	$\{\mathcal{B}^y(l)/z\}\mathcal{B}^z(l')$	
$\mathcal{B}^y(\langle M/x\rangle l)$	$=$	$\{\mathcal{B}(M)/x\}\mathcal{B}^y(l)$	if $M \in \mathsf{SN}^{B,\mathsf{x}}$
$\mathcal{B}^y(\langle M/x\rangle l)$	$=$	$(\lambda x.\mathcal{B}^y(l))\ \mathcal{B}(M)$	if $M \notin \mathsf{SN}^{B,\mathsf{x}}$

Figure 6.13: Modified encoding of $\overline{\lambda}$ into λ-calculus

Remark 169 For all y and l, $y \in FV(\mathcal{B}^y(l))$

Lemma 170

1. *If* $M \longrightarrow_{minB} N$ *is unsafe then* $\mathcal{B}(M) \longrightarrow_\beta \mathcal{B}(N)$
   If $l \longrightarrow_{minB} l'$ *is unsafe then* $\mathcal{B}^y(l) \longrightarrow_\beta \mathcal{B}^y(l')$
2. *If* $M \longrightarrow_{minB} N$ *is safe then* $\mathcal{B}(M) \longrightarrow_\beta^* \mathcal{B}(N)$
   If $l \longrightarrow_{minB} l'$ *is safe then* $\mathcal{B}^y(l) \longrightarrow_\beta^* \mathcal{B}^y(l')$
3. *If* $M \longrightarrow_{minx} N$ *then* $\mathcal{B}(M) = \mathcal{B}(N)$
   If $l \longrightarrow_{minx} l'$ *then* $\mathcal{B}^y(l) = \mathcal{B}^y(l')$

Corollary 171 *If* $\mathcal{B}(M) \in \mathcal{SN}^\beta$ *(resp.* $\mathcal{B}^y(l) \in \mathcal{SN}^\beta$*) then* $M \in \mathcal{SN}^{B,x}$ *(resp.* $l \in \mathcal{SN}^{B,x}$*).*

Proof: Direct application of Theorem 85. □

Now notice that $\mathcal{B} \cdot \mathcal{A} = \mathsf{Id}$, so that we conclude the following:

Corollary 172 (Preservation of Strong Normalisation)
If $t \in \mathcal{SN}^\beta$ *then* $\mathcal{A}(t) \in \mathcal{SN}^{B,x}$*.*

Note that the modified encoding still preserves types:

Remark 173

1. If $\Gamma \vdash_{\overline{\lambda}} M : A$ then $\Gamma \vdash_\lambda \mathcal{B}(M) : A$
2. If $\Gamma; B \vdash_{\overline{\lambda}} l : A$ then $\Gamma, y : B \vdash_\lambda \mathcal{B}^y(l) : A$ if y is fresh

And now by using the fact that typed λ-terms are in SN^β, we directly get:

Corollary 174 (Strong Normalisation of typed terms)

1. *If* $\Gamma \vdash_{\overline{\lambda}} M : A$ *then* $M \in \mathcal{SN}^{B,x}$*.*
2. *If* $\Gamma; B \vdash_{\overline{\lambda}} l : A$ *then* $l \in \mathcal{SN}^{B,x}$*.*

Again, this could also be done with any typing system such that the encodings of typed terms by $\mathcal{B}$ are typable in a typing system of λ-calculus that entails strong normalisation.

This is again the case with intersection types. Kentaro Kikuchi is working on a characterisation of $\mathsf{SN}^{B,x}$ in $\overline{\lambda}$ by such a typing system, the rules of which differ from those of Figure 4.4 in that the elimination rules of the intersection are replaced by rules for left-introduction, in the spirit of sequent calculus. Again, we expect the safeness and minimality technique to prove that typable terms are strongly normalising (using again Theorem 62), but this remains to be checked.

6.3 Q-restriction & LJQ

6.3.1 A fragment of λG3

We call q-restriction of λG3 the fragment of CBV-pure terms defined as follows:

Definition 103 (CBV-purity) A term is *CBV-pure* if in any sub-term of the form $x[M, y.N]$, M is a value.

Theorem 175 (Preservation of CBV-purity) *CBV-reduction preserve CBV-purity.*

Proof: Easy check on the rules. □

Now we prove that the q-restriction is logically complete. We show that any proof can be transformed into a CBV-pure term, and we use for that the following purification rule:

$$(\mathsf{CBV} - pur) \quad x[M, y.N] \longrightarrow \langle M \dagger z.x[z, y.N]\rangle \qquad \text{if } M \text{ is not a value}$$

Note that this rule satisfies the subject reduction property. It also terminates, simply because every application of this rule deceases the number of sub-terms of the form $x[M, y.N]$ with M not a value.

From a logical point of view, the q-restriction corresponds to the sequent calculus LJQ, as defined for instance in [Her95]. The q-restriction can also be expressed by separating explicitly the values from the other terms, which leads to a calculus that we call λLJQ. It appeared in [DL06] which contains a summary of this chapter.

Definition 104 (λLJQ)

$$\begin{array}{lcl} V, V' & ::= & x \mid \lambda x.M \mid \left\langle V \nwarrow x.V' \right\rangle \\ M, N, P & ::= & [V] \mid x[V, y.N] \mid \left\langle V \nwarrow x.N \right\rangle \mid \langle M \dagger x.N\rangle \end{array}$$

The typing rules, shown in Fig. 6.14, are inherited from those of λG3. Derivability, in the typing system of λLJQ, of the sequents $\Gamma \vdash^{\mathsf{V}} V : A$ and $\Gamma \vdash M : A$, is denoted $\Gamma \vdash^{\mathsf{V}}_{\lambda\mathsf{LJQ}} V : A$ and $\Gamma \vdash_{\lambda\mathsf{LJQ}} M : A$, respectively.

The reduction system, also called λLJQ, is inherited from CBV-reduction in λG3 as well, as shown in Fig. 6.15.

6.3.2 The CPS-semantics of λLJQ

From λLJQ to λ_{CPS}

We can adapt Fischer's translation to λLJQ so that reductions in λLJQ can be simulated. The (refined) Fischer CPS-translation of the terms of λLJQ is presented in Fig. 6.16.

$$\frac{}{\Gamma, x{:}A \vdash^{\mathsf{V}} x{:}A} \qquad \frac{\Gamma \vdash^{\mathsf{V}} V{:}A}{\Gamma \vdash [V]{:}A}$$

$$\frac{\Gamma, x{:}A \vdash V{:}B}{\Gamma \vdash^{\mathsf{V}} \lambda x.M{:}A{\to}B} \qquad \frac{\Gamma, x{:}A{\to}B \vdash^{\mathsf{V}} M{:}A \quad \Gamma, x{:}A{\to}B, y{:}B \vdash N{:}C}{\Gamma, x{:}A{\to}B \vdash x[V, y.N]{:}C}$$

$$\frac{\Gamma \vdash^{\mathsf{V}} V{:}A \quad \Gamma, x{:}A \vdash^{\mathsf{V}} V'{:}B}{\Gamma \vdash^{\mathsf{V}} \langle V \nwarrow x.V' \rangle{:}B} \qquad \frac{\Gamma \vdash V{:}A \quad \Gamma, x{:}A \vdash N{:}B}{\Gamma \vdash \langle V \nwarrow x.N \rangle{:}B}$$

$$\frac{\Gamma \vdash M{:}A \quad \Gamma, x{:}A \vdash N{:}B}{\Gamma \vdash \langle M \dagger x.N \rangle{:}B}$$

Figure 6.14: Typing system of $\lambda\mathsf{LJQ}$

$$\begin{array}{lcl}
\langle [\lambda x.M] \dagger y.y[V, z.P] \rangle & \longrightarrow & \langle \langle [V] \dagger x.M \rangle \dagger z.P \rangle \quad \text{if } y \notin \mathsf{FV}(V) \cup \mathsf{FV}(P) \\
\langle [x] \dagger y.N \rangle & \longrightarrow & \{x/y\} N \\
\langle M \dagger y.[y] \rangle & \longrightarrow & M \\
 & & \\
\langle z[V, y.P] \dagger x.N \rangle & \longrightarrow & z[V, y.\, \langle P \dagger x.N \rangle] \\
\langle \langle [V]' \dagger y.y[V, z.P] \rangle \dagger x.N \rangle & \longrightarrow & \langle [V'] \dagger y.y[V, z.\, \langle P \dagger x.N \rangle] \rangle \\
 & & \qquad \text{if } y \notin \mathsf{FV}(V) \cup \mathsf{FV}(P) \\
\langle \langle M \dagger y.P \rangle \dagger x.N \rangle & \longrightarrow & \langle M \dagger y.\langle P \dagger x.N \rangle \rangle \\
 & & \quad \text{if the redex is not one of the previous rule} \\
\langle [\lambda y.M] \dagger x.N \rangle & \longrightarrow & \langle \lambda y.M \nwarrow x.N \rangle \text{ if } N \text{ is not an } x\text{-covalue} \\
 & & \\
\langle V \nwarrow x.x \rangle & \longrightarrow & V \\
\langle V \nwarrow x.y \rangle & \longrightarrow & y \\
\langle V \nwarrow x.\lambda y.M \rangle & \longrightarrow & \lambda y.\, \langle V \nwarrow x.M \rangle \\
 & & \\
\langle V \nwarrow x.[V'] \rangle & \longrightarrow & [\langle V \nwarrow x.V' \rangle] \\
\langle V \nwarrow x.x[V', z.P] \rangle & \longrightarrow & \langle [V] \dagger x.x[\langle V \nwarrow x.V' \rangle, z.\, \langle V \nwarrow x.P \rangle] \rangle \\
\langle V \nwarrow x.x'[V', z.P] \rangle & \longrightarrow & x'[\langle V \nwarrow x.V' \rangle, z.\, \langle V \nwarrow x.P \rangle] \\
\langle V \nwarrow x.\langle M \dagger y.P \rangle \rangle & \longrightarrow & \langle \langle V \nwarrow x.M \rangle \dagger y.\langle V \nwarrow x.P \rangle \rangle
\end{array}$$

Figure 6.15: Reduction rules of $\lambda\mathsf{LJQ}$

Now we prove the simulation of $\lambda\mathsf{LJQ}$ by $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$, and for that we need the following remark and lemma.

$[V]\colon K$	$:=$	$K\ V^\dagger$	
$(x[V, y.M])\colon K$	$:=$	$x\ (\lambda y.(M\colon K))\ V^\dagger$	
$\langle W \dagger x.x[V, y.M]\rangle\colon K$	$:=$	$W^\dagger\ (\lambda y.(M\colon K))\ V^\dagger$	if $x \notin \mathsf{FV}(V) \cup \mathsf{FV}(M)$
$\langle N \dagger x.M\rangle\colon K$	$:=$	$N\colon \lambda x.(M\colon K)$	otherwise
$\langle V \nwarrow x.M\rangle\colon K$	$:=$	$\left\{{}^{V^\dagger}\!/_{x}\right\}(M\colon K)$	
$x^\dagger$	$:=$	x	
$(\lambda x.M)^\dagger$	$:=$	$\lambda k.\lambda x.(M\colon k)$	
$\langle V \nwarrow x.W\rangle^\dagger$	$:=$	$\left\{{}^{V^\dagger}\!/_{x}\right\} W^\dagger$	

Figure 6.16: The (refined) Fischer translation from LJQ

Remark 176 $\mathsf{FV}(M\colon K) \subseteq \mathsf{FV}(M) \cup \mathsf{FV}(K)$ and $\mathsf{FV}(V^\dagger) \subseteq \mathsf{FV}(V)$.

Lemma 177

1. $\left\{{}^{K'}\!/_{k}\right\}(M\colon K) = M\colon \left\{{}^{K'}\!/_{k}\right\} K$.
2. $M\colon \lambda x.(P\colon K) \longrightarrow^*_{\beta \mathsf{v} 1} \langle M \dagger x.P\rangle\colon K$ *if* $x \notin \mathsf{FV}(K)$.
3. $(\{{}^{y}\!/_{x}\} V)^\dagger = \{{}^{y}\!/_{x}\} V^\dagger$, *and*
   $\{{}^{y}\!/_{x}\} M\colon K = \{{}^{y}\!/_{x}\}(M\colon K)$, *provided* $x \notin \mathsf{FV}(K)$.
4. *If* $x \notin \mathsf{FV}(K)$, *then* $M\colon \lambda x.K\ x \longrightarrow_{\beta \mathsf{v} 1} M\colon K$.
5. $\left\{{}^{V^\dagger}\!/_{x}\right\}(M\colon K) = \left\{{}^{V^\dagger}\!/_{x}\right\}(M\colon \left\{{}^{V^\dagger}\!/_{x}\right\} K)$

Proof:

1. By induction on M.
2. The interesting case is the following:

$$\begin{array}{rcl} W\colon \lambda x.(x[V, y.M]\colon K) & = & (\lambda x.x\ (\lambda y.(M\colon K))\ V^\dagger)\ W^\dagger \\ & \longrightarrow_{\beta \mathsf{v} 1} & W^\dagger\ (\lambda y.(M\colon K))\ V^\dagger \\ & = & (\langle W \dagger x.x[V, y.M]\rangle)\colon K \end{array}$$

   when $x \notin \mathsf{FV}(V) \cup \mathsf{FV}(M)$.

3. By structural induction on V, M.
4. By induction on M. The translation propagates the continuation $\lambda x.K\ x$ into the sub-terms of M until it reaches a value, for which $[V]\colon \lambda x.K\ x = (\lambda x.K\ x)\ V^\dagger \longrightarrow_{\beta \mathsf{v} 1} K\ V = [V]\colon K$.

5. By induction on M. The term $M\colon K$ only depends on K in that K is a sub-term of $M\colon K$, affected by the substitution.

□

Theorem 178 (Simulation of λLJQ)

1. *If* $M \longrightarrow_{\lambda LJQ} M'$ *then for all* K *we have* $M\colon K \longrightarrow^*_{\lambda^{\mathcal{F}}_{CPS\beta}} M'\colon K$.
2. *If* $W \longrightarrow_{\lambda LJQ} W'$ *then we have* $W^\dagger \longrightarrow^*_{\lambda^{\mathcal{F}}_{CPS\beta}} W'^\dagger$.

Proof: By simultaneous induction on the derivation of the reduction step, using Lemma 177. □

A restriction on $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$: $\lambda^{f}_{\mathsf{CPS}}$

The (refined) Fischer translation of λLJQ is not surjective on the terms of λ_{CPS}, indeed we only need the terms of Fig. 6.17, which we call $\lambda^{f}_{\mathsf{CPS}}$.

$$\begin{array}{lll} M,N & ::= K\ V \mid V\ (\lambda x.M)\ W & \\ V,W & ::= x \mid \lambda k.\lambda x.M & k \in \mathsf{FV}(M) \\ K & ::= k \mid \lambda x.M & \end{array}$$

Figure 6.17: $\lambda^{f}_{\mathsf{CPS}}$

Note that $\lambda^{f}_{\mathsf{CPS}}$ is stable under $\beta_{\mathsf{V}}1, \beta_{\mathsf{V}}2$, but not under $\eta_{\mathsf{V}}1$ and $\eta_{\mathsf{V}}2$. However we can equip $\lambda^{f}_{\mathsf{CPS}}$ with the reduction system of Fig. 6.18, where the rules $\beta_{\mathsf{V}}1, \beta_{\mathsf{V}}3$. Note that $\beta_{\mathsf{V}}1$ is the same as for $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ and $\beta_{\mathsf{V}}3$ is merely rule $\beta_{\mathsf{V}}2$ with the assumption that the redex is in $\lambda^{f}_{\mathsf{CPS}}$. We write $\lambda^{f}_{\mathsf{CPS}_\beta}$ for system $\beta_{\mathsf{V}}1, \beta_{\mathsf{V}}2$ and $\lambda^{f}_{\mathsf{CPS}_{\beta\eta}}$ for system $\beta_{\mathsf{V}}1, \beta_{\mathsf{V}}2, \eta_{\mathsf{V}}1, \eta_{\mathsf{V}}2$ in $\lambda^{f}_{\mathsf{CPS}}$.

$$\begin{array}{lll} (\lambda x.M)\ V & \longrightarrow_{\beta_{\mathsf{V}}1} & \left\{{}^{V}\!/_{x}\right\} M \\ (\lambda k.\lambda x.M)\ (\lambda y.N)\ V & \longrightarrow_{\beta_{\mathsf{V}}3} & (\lambda x.\left\{{}^{\lambda y.N}\!/_{k}\right\} M)\ V \\ \hline \lambda k.\lambda x.V\ (\lambda z.k\ z)\ x & \longrightarrow_{\eta_{\mathsf{V}}3} & V \qquad \text{if } x \notin \mathsf{FV}(V) \end{array}$$

Figure 6.18: Reduction rules of $\lambda^{f}_{\mathsf{CPS}}$

We can now project $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ onto $\lambda^{f}_{\mathsf{CPS}}$, as shown in Fig. 6.19.

Remark 179 Note that, in $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$, $\uparrow M \longrightarrow_{\eta_{\mathsf{V}}2} M$, $\uparrow V \longrightarrow_{\eta_{\mathsf{V}}2} V$, $\uparrow K \longrightarrow_{\eta_{\mathsf{V}}2} K$ and if M, V, K are in $\lambda^{f}_{\mathsf{CPS}}$ then $\uparrow M = M$, $\uparrow V = V$, $\uparrow K = K$.

Theorem 180 (Galois connection from $\lambda^{f}_{\mathsf{CPS}}$ to $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$) *The identity* $\mathsf{Id}_{\lambda^{f}_{CPS}}$ *and the mapping* $\uparrow$ *form a Galois connection from* $\lambda^{\mathcal{F}}_{CPS}$, *equipped with* $\lambda^{f}_{CPS\beta\eta}$, *to* $\lambda^{\mathcal{F}}_{CPS}$, *equipped with* $\lambda^{\mathcal{F}}_{CPS\beta\eta}$, *(and also with only* $\lambda^{f}_{CPS\beta}$ *and* $\lambda^{\mathcal{F}}_{CPS\beta}$*).*

$\uparrow (K\ V)$	$:= \uparrow K\ \uparrow V$
$\uparrow (W\ k\ V)$	$:= \uparrow W\ (\lambda x.k\ x)\ \uparrow V$
$\uparrow (W\ (\lambda x.M)\ V)$	$:= \uparrow W\ \uparrow (\lambda x.\uparrow M)\ \uparrow V$
$\uparrow x$	$:= x$
$\uparrow \lambda k.\lambda x.M$	$:= \lambda k.\lambda x.\uparrow M$
$\uparrow k$	$:= k$
$\uparrow \lambda x.M$	$:= \lambda x.\uparrow M$

Figure 6.19: Projection of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ onto $\lambda^{f}_{\mathsf{CPS}}$

Proof: Given Remark 179, it suffices to check the simulations.

- For the simulation of $\lambda^{\mathcal{F}}_{\mathsf{CPS}\beta\eta}$ by $\lambda^{f}_{\mathsf{CPS}\beta\eta}$ through $\uparrow$, we use a straightforward induction on the derivation of the reduction step, using the following fact:

$$\begin{array}{ll} \left\{{}^{\uparrow V}/_{x}\right\} \uparrow M =\uparrow \left\{{}^{V}/_{x}\right\} M & \left\{{}^{\uparrow K}/_{k}\right\} \uparrow M \longrightarrow^{*}_{\beta\mathsf{v}1} \uparrow \left\{{}^{K}/_{k}\right\} M \\ \left\{{}^{\uparrow V}/_{x}\right\} \uparrow W =\uparrow \left\{{}^{V}/_{x}\right\} W & \left\{{}^{\uparrow K}/_{k}\right\} \uparrow W \longrightarrow^{*}_{\beta\mathsf{v}1} \uparrow \left\{{}^{K}/_{k}\right\} W \\ \left\{{}^{\uparrow V}/_{x}\right\} \uparrow K' =\uparrow \left\{{}^{V}/_{x}\right\} K' & \left\{{}^{\uparrow K}/_{k}\right\} \uparrow K' \longrightarrow^{*}_{\beta\mathsf{v}1} \uparrow \left\{{}^{K}/_{k}\right\} K' \end{array}$$

$$\begin{array}{l} \left\{{}^{\uparrow \lambda x.k\ x}/_{k}\right\} \uparrow M \longrightarrow^{*}_{\beta\mathsf{v}1} \uparrow M \\ \left\{{}^{\uparrow \lambda x.k\ x}/_{k}\right\} \uparrow W \longrightarrow^{*}_{\beta\mathsf{v}1} \uparrow W \\ \left\{{}^{\uparrow \lambda x.k\ x}/_{k}\right\} \uparrow K' \longrightarrow^{*}_{\beta\mathsf{v}1} \uparrow K' \end{array}$$

- The fact that $\beta\mathsf{v}1, \beta\mathsf{v}2, \eta\mathsf{v}1, \eta\mathsf{v}2$ simulate $\beta\mathsf{v}1, \beta\mathsf{v}2, \eta\mathsf{v}3$ through $\mathsf{Id}_{\lambda^{f}_{\mathsf{CPS}}}$ is straightforward.

□

Corollary 181 (Confluence of $\lambda^{f}_{\mathsf{CPS}}$) *$\lambda^{f}_{CPS\beta}$ and $\lambda^{f}_{CPS\beta\eta}$ are confluent.*

Proof: By Theorem 180 and Theorem 5. □

From $\lambda^{f}_{\mathsf{CPS}}$ to $\lambda\mathsf{LJQ}$

Definition 105 (The Fischer reverse translation)
We now encode $\lambda^{f}_{\mathsf{CPS}}$ into $\lambda\mathsf{LJQ}$.

$(k\ V)^{\mathsf{back}}$	$:= [V^{\mathsf{back}}]$
$((\lambda x.M)\ V)^{\mathsf{back}}$	$:= \langle V^{\mathsf{back}} \dagger x.M^{\mathsf{back}} \rangle$
$(y\ (\lambda x.M)\ V)^{\mathsf{back}}$	$:= y[V^{\mathsf{back}}, x.M^{\mathsf{back}}]$
$((\lambda k.\lambda z.N)\ (\lambda x.M)\ V)^{\mathsf{back}}$	$:= \langle \lambda z.N^{\mathsf{back}} \dagger y.y[V^{\mathsf{back}}, x.M^{\mathsf{back}}] \rangle$
x^{back}	$:= x$
$(\lambda k.\lambda x.M)^{\mathsf{back}}$	$:= \lambda x.M^{\mathsf{back}}$

Lemma 182

1. $\left\langle V^{\mathsf{back}} \nwarrow x.W^{\mathsf{back}} \right\rangle \longrightarrow^*_{\lambda LJQ} \left(\left\{{}^{V}\!/_{x}\right\} W\right)^{\mathsf{back}}$ *and*
   $\left\langle V^{\mathsf{back}} \nwarrow x.M^{\mathsf{back}} \right\rangle \longrightarrow^*_{\lambda LJQ} \left(\left\{{}^{V}\!/_{x}\right\} M\right)^{\mathsf{back}}$.

2. $\left\langle M^{\mathsf{back}} \dagger x.N^{\mathsf{back}} \right\rangle \longrightarrow^*_{\lambda LJQ} \left(\left\{{}^{\lambda x.N}\!/_{k}\right\} M\right)^{\mathsf{back}}$ *(if* $k \in \mathsf{FV}(M)$*).*

Proof: By induction on W, M. □

Theorem 183 (Simulation of λ^f_{CPS} in λLJQ)
The reduction relation $\longrightarrow_{\lambda^f_{CPS\beta}}$ *is (weakly) simulated by* $\longrightarrow_{\lambda LJQ}$ *through* $(\_)^{\mathsf{back}}$.

Proof: By induction on the derivation of the reduction step, using Lemma 182. □

Lemma 184 (Composition of the encodings)

1. $V \longrightarrow^*_{\lambda_{C\beta}} V^{\dagger^{\mathsf{back}}}$ *and* $M \longrightarrow^*_{\lambda_{C\beta}} (M\colon k)^{\mathsf{back}}$.

2. $V = V^{\mathsf{back}^\dagger}$ *and* $M = M^{\mathsf{back}}\colon k$.

Proof: By structural induction, using Lemma 182 for the first point. □

Now we can prove the following:

Theorem 185 (The refined Fischer translation is a reflection)
The refined Fischer translation and $(\_)^{\mathsf{back}}$ *form a reflection in* λLJQ *of* λ^f_{CPS} *(equipped with* $\lambda^f_{CPS\beta}$*).*

Proof: This theorem is just the conjunction of Theorem 178, Theorem 183, and Lemma 184. □

Corollary 186 (Confluence of λLJQ-reductions) λLJQ *is confluent.*

Proof: By Theorem 185 and Theorem 5. □

6.3.3 Connection with Call-by-Value λ-calculus

We have established three connections:

- a reflection in $\lambda\mathsf{LJQ}$ of λ^f_{CPS},
- a Galois connection from λ^f_{CPS} of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$,
- a reflection in λ_{C} of $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$ (in Chapter 3).

By composing the first two connections, we have a Galois connection from $\lambda\mathsf{LJQ}$ to $\lambda^{\mathcal{F}}_{\mathsf{CPS}}$, and together with last one, we have a a pre-Galois connection from $\lambda\mathsf{LJQ}$ to λ_{C}. The compositions of these connections also form an equational correspondence between $\lambda\mathsf{LJQ}$ to λ_{C}. These facts imply the following theorem:

Theorem 187 (Connections between $\lambda\mathsf{LJQ}$ and λ_{C}) *Let us write $V^\natural$ for $(\uparrow (V^{\mathcal{F}}))^{\mathsf{back}}$, $M^\sharp$ for $(\uparrow (M :_{\mathcal{F}} k))^{\mathsf{back}}$, $V^\flat$ for $(V^\dagger)^{\mathcal{F}\mathsf{back}}$ and $M^\flat$ for $(M : k)^{\mathcal{F}\mathsf{back}}$.*

1. *For any terms M and N of λ_{C}, if $M \longrightarrow_{\lambda_{\mathsf{C}\beta}} N$ then $M^\sharp \longrightarrow^*_{\lambda\mathsf{LJQ}} N^\sharp$.*
2. *For any terms M and N of $\lambda\mathsf{LJQ}$, if $M \longrightarrow_{\lambda\mathsf{LJQ}} N$ then $M^\flat \longrightarrow^*_{\lambda_{\mathsf{C}\beta}} N^\flat$.*
3. *For any term M of λ_{C}, $M \longleftrightarrow^*_{\lambda_{\mathsf{C}\beta}} M^{\sharp\flat}$.*
4. *For any term M of $\lambda\mathsf{LJQ}$, $M \longrightarrow^*_{\lambda\mathsf{LJQ}} M^{\flat\sharp}$.*

Proof: By composition of the reflections and Galois connection. □

Corollary 188 (Equational correspondence between $\lambda\mathsf{LJQ}$ and λ_{C})

1. *For any terms M and N of $\lambda\mathsf{LJQ}$, $M \longleftrightarrow^*_{\lambda\mathsf{LJQ}} N$ if and only if $M^\flat \longleftrightarrow^*_{\lambda_{\mathsf{C}\beta}} N^\flat$.*
2. *For any terms M and N of λ_{C}, $M \longleftrightarrow^*_{\lambda_{\mathsf{C}\beta}} N$ if and only if $M^\sharp \longleftrightarrow^*_{\lambda\mathsf{LJQ}} N^\sharp$.*

We can give the composition of encodings explicitly. We have for instance the following theorem:

Theorem 189 (From λ_{C} to $\lambda\mathsf{LJQ}$) *The following equations hold:*

$x^\natural$	$= x$
$(\lambda x.M)^\natural$	$= \lambda x.M^\sharp$
$V^\sharp$	$= [V^\natural]$
$(\mathsf{let}\ y = x\ V\ \mathsf{in}\ P)^\sharp$	$= x[V^\natural, y.P^\sharp]$
$(\mathsf{let}\ y = (\lambda x.M)\ V\ \mathsf{in}\ P)^\sharp$	$= \langle \lambda x.M^\sharp \dagger z.z[V^\natural, y.P^\sharp] \rangle$
$(\mathsf{let}\ z = V\ N\ \mathsf{in}\ P)^\sharp$	$= (\mathsf{let}\ y = N\ \mathsf{in}\ (\mathsf{let}\ z = V\ y\ \mathsf{in}\ P))^\sharp$ *if N is not a value*
$(\mathsf{let}\ z = M\ N\ \mathsf{in}\ P)^\sharp$	$= (\mathsf{let}\ x = M\ \mathsf{in}\ (\mathsf{let}\ z = x\ N\ \mathsf{in}\ P))^\sharp$ *if M is not a value*
$(\mathsf{let}\ z = (\mathsf{let}\ x = M\ \mathsf{in}\ N)\ \mathsf{in}\ P)^\sharp$	$= (\mathsf{let}\ x = M\ \mathsf{in}\ (\mathsf{let}\ z = N\ \mathsf{in}\ P))^\sharp$
$(\mathsf{let}\ y = V\ \mathsf{in}\ P)^\sharp$	$= \langle V^\sharp \dagger y.P^\sharp \rangle$
$(M\ N)^\sharp$	$= (\mathsf{let}\ y = M\ N\ \mathsf{in}\ y)^\sharp$

Proof: By structural induction, unfolding the definition of the encodings on both sides of each equation. □

In fact, we could take this set of equalities as the definition of the direct encoding of λ_{C} into $\lambda\mathsf{LJQ}$. For that it suffices to check that there is a measure that makes this set of equations a well-founded definition. We now give this measure, given by an encoding of the terms of λ_{C} into first-order terms equipped with an LPO.

Definition 106 (An LPO for λ_{C}) We encode λ_{C} into the first-order syntax given by the following term constructors and their precedence relation:

$$\mathsf{ap}(\_,\_) \succ \mathsf{let}(\_,\_) \succ \mathsf{ii}(\_,\_) \succ \mathsf{i}(\_) \succ \star$$

The precedence relation generates a terminating LPO $\gg$ as presented in Definition 52. The encoding is given in Fig. 6.20. We can now consider the (terminating) relation induced by $\gg$ through the reverse relation of the encoding as a measure for λ_{C}.

$\overline{x}$	$:=$	$\star$	
$\overline{\lambda x.M}$	$:=$	$\mathsf{i}(\overline{M})$	
$\overline{\mathsf{let}\ x = M_1\ M_2\ \mathsf{in}\ N}$	$:=$	$\mathsf{let}(\mathsf{ii}(\overline{M_1}, \overline{M_2}), \overline{N})$	
$\overline{\mathsf{let}\ x = M\ \mathsf{in}\ N}$	$:=$	$\mathsf{let}(\overline{M}, \overline{N})$	otherwise
$\overline{M\ N}$	$:=$	$\mathsf{ap}(\overline{M}, \overline{N})$	

Figure 6.20: Encoding of λ_{C} into the first-order syntax

Remark 190 $\mathsf{let}(\overline{M}, \overline{N}) \gg \overline{\mathsf{let}\ x = M\ \mathsf{in}\ N}$ or $\mathsf{let}(\overline{M}, \overline{N}) = \overline{\mathsf{let}\ x = M\ \mathsf{in}\ N}$.

In the other direction, we have:

Theorem 191 (From $\lambda\mathsf{LJQ}$ to λ_{C})

$x^\flat$	$=$	x
$(\lambda x.M)^\flat$	$=$	$\lambda x.M^\flat$
$\langle V \nwarrow x.V' \rangle^\flat$	$=$	$\left\{ {}^{V^\flat}\!/_{x} \right\} V'^\flat$
$[V]^\flat$	$=$	$V^\flat$
$(x[V, y.M])^\flat$	$=$	$\mathit{let}\ y = x\ V^\flat\ \mathit{in}\ M^\flat$
$\langle N \dagger x.M \rangle^\flat$	$\lambda_{\mathsf{C}\beta}{}^* \longleftarrow$	$\mathit{let}\ x = N^\flat\ \mathit{in}\ M^\flat$
$\langle V \nwarrow x.M \rangle^\flat$	$=$	$\left\{ {}^{V^\flat}\!/_{x} \right\} M^\flat$

Proof: By structural induction, unfolding the definition of the encodings on each side and using Lemma 76. Also note that we do *not* have an equality in the penultimate line but only a reduction. This prevents the use of this theorem as a definition for the encoding from $\lambda\mathsf{LJQ}$ to λ_{C}, although refining this case (with different sub-cases) could lead to a situation like that of Theorem 189 (with a measure to find in order for the set of equations to form a well-formed definition). □

Remark 192 Note from Theorem 189 and Theorem 191 that if M is a cut-free term of $\lambda\mathsf{LJQ}$, $M^{\flat\sharp} = M$.

The connection between $\lambda\mathsf{LJQ}$ and λ_{C} suggests to restricts λ_{C} by always requiring (a series of) applications to be explicitly given with a contination, i.e. to refuse a term of λ_{C} such as $\lambda x.M_1\ M_2\ M_3$ but only accept $\lambda x.\mathsf{let}\ y = M_1\ M_2\ M_3\ \mathsf{in}\ y$. The refined Fischer translation from Chapter 3, on that particular fragment of λ_{C}, directly has $\lambda^{f}_{\mathsf{CPS}}$ as its target calculus, and the equational correspondence between λ_{C} d $\lambda\mathsf{LJQ}$ would then also become a pre-Galois connection from the former to the latter. The fragment is not stable under rule η_{let}, and corresponds to the terms of λ_{C} in some notion of η_{let}-*long form*.

This restriction can be formalised with the syntax of a calculus given in [Esp05]:

$$\begin{array}{lll} M, N, P & ::= & x \mid \lambda x.M \mid \mathsf{let}\ x = E\ \mathsf{in}\ M \\ E & ::= & M \mid E\ M \end{array}$$

In fact, this calculus is introduced in [Esp05] as a counterpart, in natural deduction, of a sequent calculus. Hence, it seems that it is the adequate framework for formalising in natural deduction the concepts developed in this chapter about sequent calculus, in particular the CBN- and CBV-reductions, and the t- and q-restrictions. This calculus would thus capture both the traditional λ-calculus and (the η_{let}-long forms of) λ_{C}. Further work includes investigating the relation between this calculus and $\lambda\mathsf{G3}$.

Conclusion

This chapter surveyed some computational notions in $\lambda\mathsf{G3}$ based on cut-elimination in the sequent calculus G3ii. It presented various propagation systems with a comparative approach and a general framework with proof-terms, in which CBV and CBN sub-systems were generically defined. In each case, confluence of these sub-systems is only conjectured, which gives a direction for further work.

This chapter then presented the t- and q-restrictions of $\lambda\mathsf{G3}$/G3ii, both in the proof-terms and in the logic, corresponding to the sequent calculi LJT and LJQ. The two fragments are respectively stable under CBN- and CBV-reduction.

We recalled the strong connection between the t-fragment and the (CBN) λ-calculus by means of a reflection in the calculus $\overline{\lambda}$, and derived from this connection some properties of $\overline{\lambda}$ such as confluence, PSN and strong normalisation of typed terms (illustrating the use of the safeness and minimality technique from Chapter 2).

We established some new results about the q-fragment and the CBV λ-calculus, expressed via Moggi's λ_{C}-calculus presented in Chapter 3. Further development of the material in this section about LJQ and λ_{C} is ongoing work, including refining the encodings and the simulations to have a Galois connection or a reflection.

Once this is done, a promising direction for further work is given by the calculus from [Esp05] in natural deduction that encompasses both the traditional (CBN) λ-calculus and (a minor variant of) the CBV λ_{C}-calculus, with a very strong connection with sequent calculus as a whole (i.e. not just with the t-fragment or the q-fragment).

Chapter 7

A higher-order calculus for G4ii

In this chapter, whose contents appeared in [DKL06], we apply the same technique as in Chapter 6 to the sequent calculus G4ii (as it is called in [TS00]) for intuitionistic propositional logic.

Independently developed in [Hud89, Hud92] and [Dyc92] (see also [LSS91]), it has the strong property of being *depth-bounded*, in that proofs are of bounded depth and thus (for root-first proof search) no loop-checking is required. This contrasts with other calculi for this logic such as G3ii, where proofs can be of unbounded depth. Its essential ingredients appeared already in 1952 work of Vorob'ev, published in detail in [Vor70].

Its completeness can be shown by various means, either indirectly, using the completeness of another calculus and a permutation argument [Dyc92], or directly, such as in [DN00] where cut-admissibility is proved without reference to the completeness of any other sequent calculus.

As described in Chapter 6, such an admissibility proof can be seen, via the Curry-Howard correspondence, as a *weakly normalising* proof-reduction system. Again we present a formulation of implicational G4ii with derivations represented by terms; strong (instead of weak) normalisation is proved by the use of a MPO. Several variations, *all of them being strongly normalising*, are considered, depending on whether we want to have a system as general as possible or a system more restricted (but simpler) implementing some reduction strategy.

The merits of G4ii for proof-search and automated reasoning have been discussed in many papers (see [ORK05] for some recent pointers; note its use of an old name LJT for G4ii). However, a question that has been less investigated is the following: what are the proofs expressed in G4ii and what is their semantics? Here we investigate an operational, rather than denotational, semantics because it is more directly related to inductive proofs of cut-admissibility (such as in [DN00]). Further work will investigate denotational semantics, by relating these proofs and their reductions to the simply-typed λ-calculus.

In contrast to previous work, we present G4ii with a term syntax, and our approach to cut-elimination differs from that in [DN00], which showed (using

logical sequents) first the admissibility of contraction and then the admissibility of context-splitting (a.k.a. multiplicative) cut. By rather using a context-sharing (a.k.a. additive) cut (which is easier to handle with proof-terms since no linearity constraint restricts the cut-constructor), admissibility of contraction follows as a special case of that of cut.

To some extent, Matthes [Mat02] also investigated terms and reductions corresponding to cut-elimination in G4ii, with a variety of motivations, such as that of understanding better Pitts' algorithm [Pit92] for uniform interpolation. His work is similar to ours in using terms to represent derivations; but it differs conceptually from ours by considering not the use of explicit operators for the cut-rule but the closure of the syntax under (implicit) substitution, as in λ-calculus, where the general syntax of λ-terms may be regarded as the extension of the normal λ-terms by such a closure. His reduction rules are global (using implicit substitutions) rather than local (using explicit operators); strong normalisation is shown for a subset of the reductions, but unfortunately not for all that are required.

The chapter is organised as follows. Section 7.1 describes the steps that change G3ii into G4ii. Section 7.2 presents the term syntax and typing rules of our HOC for G4ii and its auxiliary (admissible) rules. Section 7.3 studies proof transformations and reduction rules of the calculus. Section 7.4 shows a translation from the derivations of the calculus to a first-order syntax and proves that every reduction step satisfies subject reduction and decreases first-order terms with respect to an MPO, thus proving strong normalisation. In Section 7.5 we discuss variants for the reduction system introduced in Section 7.3, some of them being confluent. Finally we conclude and give some ideas for further work.

7.1 From G3ii to G4ii

In this section we describe the steps that transform G3ii into G4ii.

Definition 107 (Standard terms of λG3) A term is *standard* if in any subterm of the form $y[N, x.M]$, $y \notin \mathsf{FV}(M)$.

In a typed framework, this corresponds to changing the left-introduction to the following one:

$$\frac{\Gamma, x\!:\!A{\to}B \vdash M\!:\!A \quad \Gamma, y\!:\!B \vdash N\!:\!C}{\Gamma, x\!:\!A{\to}B \vdash x[M, y.N]\!:\!C}$$

We call this sequent calculus G3ii$'$, and the following purification rule standardises terms, but using cuts:

$$x[M, y.N] \longrightarrow_{\mathsf{purG3ii'}} x[M, y.\,\langle \lambda z.y \dagger x.N\rangle]$$

Clearly this rule is non-terminating unless the side-condition $y \in \mathsf{FV}(M)$ ensures that the application of the rule is actually needed.

Unfortunately, G3ii′ is not stable under either CBN or CBV-reductions, so eliminating the cut introduced by purG3ii′ can produce non-standard terms so that purG3ii′ is needed again. Hence, it would be interesting to prove that adding this rule to the reductions of λG3 does not break strong normalisation on typed terms. Alternatively we could also look for an internal reduction system for G3ii′, by dropping the side-condition purG3ii′ but rather integrating the rule to each rewrite rule that might produce a non-standard term.

Now this rule is interesting in that it is the first rule we introduce that makes a semantical jump: so far, all the rules reducing terms of λG3 leave the image by $\mathcal{G}^1$ in the same $\beta\eta$-class. This one, on the contrary, jumps from one to another, as noticed by Vestergaard [Ves99].

$$f[x, y.f[y, z.z]] \longrightarrow_{\mathsf{purG3ii'}} f[x, y.\langle \lambda y'.y \dagger f.f[y, z.z]\rangle] \longrightarrow^* f[x, z.z]$$

while $\mathcal{G}^1(f[x, y.f[y, z.z]]) = f\ (f\ x)$ and
$\mathcal{G}^1(f[x, y.\langle \lambda y'.y \dagger f.f[y, z.z]\rangle]) = \mathcal{G}^1(f[x, z.z]) = f\ x$.

More generally for $n \geq 2$ we have by induction

$$\begin{array}{lll} f[x, x_n. \dots f[x_2, x_1.x_1]] & \longrightarrow^*_{i.h.} & f[x, x_n.f[x_n, x_1.x_1]] \\ & \longrightarrow^* & f[x, x_1.x_1] \end{array}$$

The G4ii-calculus is built from the combination of G3ii′ with the q-restriction. Indeed, such a combination replaces rule $\rightarrow_{\mathsf{left}}$ of G3ii by the following ones:

$$\frac{\Gamma, y\colon A, z\colon B \vdash N\colon E}{\Gamma, x\colon A{\rightarrow}B, y\colon A \vdash x[y, z.N]\colon E}$$

$$\frac{\Gamma, y\colon C, x\colon (C{\rightarrow}D){\rightarrow}B \vdash M\colon D \qquad \Gamma, z\colon B \vdash N\colon E}{\Gamma, x\colon (C{\rightarrow}D){\rightarrow}B \vdash x[\lambda y.M, z.N]\colon E}$$

with $x \notin \mathsf{FV}(N)$ in both cases (the restriction of being standard).

We obtain these two rules only by case analysis on the proof-term of the first premiss in the left-introduction rule of LJQ: it must be a value, so it is either a variable y or an abstraction $\lambda y.M$.

The restriction of G3ii′ forbids x to appear in N because it can use z instead, which has a logically stronger type, and this already drastically reduces the set of proofs (without losing completeness, see e.g. [TS00]).

For instance, all Church numerals (but zero) collapse to 1: Let $\overline{n}$ be the Church numeral n in λ-calculus, i.e. $\lambda x.\lambda f.f\ (\dots f\ x)$ (with n occurrences of f), we encode them in the q-fragment. Suppose $n \geq 2$.

$$\begin{array}{lll} \mathcal{P}r(\overline{n}) = \mathcal{G}^2(\overline{n}) & = & \lambda x.\lambda f.f[\dots f[x, x_n.x_n], x_1.x_1] \\ & \longrightarrow^*_{\mathsf{pur\text{-}q,CBV}} & \lambda x.\lambda f.f[x, x_n. \dots f[x_2, x_1.x_1]] \\ & \longrightarrow^*_{\mathsf{purG3ii',CBV}} & \lambda x.\lambda f.f[x, x_1.x_1] \\ & \longrightarrow^* & \mathcal{P}r(\overline{1}) \end{array}$$

The G4ii sequent calculus takes the even more drastic step of forbidding x (of type $(C \to D) \to B$) in M but allowing a variable v with a type $D \to B$ which, in presence of C (the type of y) is intuitionistically equivalent to $(C \to D) \to B$. Hence, we get the rule:

$$\frac{\Gamma, z{:}C, v{:}D{\to}B \vdash M{:}B \quad \Gamma, y{:}B \vdash N{:}C}{\Gamma, x{:}(C{\to}D){\to}B \vdash x[z.v.M, y.N]{:}C}$$

with a new constructor for the rule: $x[z.v.M, y.N]$.

An interpretation of this constructor in λG3 could be:

$$x[\lambda z.\, \langle \lambda w.x[\lambda u.w, y'.y'] \dagger v.M \rangle\, , y.N]$$

As we shall see, the move from the q-restriction of G3ii' to G4ii is what provides the property of being *depth-bounded*. As we shall see, logical completeness is not lost, but there is clearly some computational completeness that is lost (compared to, say, λG3 or λ-calculus): it was already the case for the q-restriction of G3ii', but even more proofs are lost in G4ii. On the other hand, depth-boundedness is very convenient for proof-search, since they ensure finiteness of search space.

7.2 An HOC for G4ii

7.2.1 Syntax

Definition 108 (Grammar of Terms)

$$\begin{array}{rcl} M, N & ::= & x \mid \lambda x.M \mid x[y, z.M] \mid x[u.v.M, z.N] \mid \\ & & \mathtt{inv}(x, y.M) \mid \mathtt{dec}(x, y, z.M) \mid \langle M \dagger x.N \rangle \end{array}$$

In this definition, the first line defines the syntax for *normal* terms (corresponding to primitive derivations) and the second gives the extra syntax for *auxiliary* terms, which may be built up using also the "auxiliary constructors" that appear in bold teletype font, such as `cut`. Six of the eight term constructors use variable binding: in $\lambda x.M$, x binds in M; in $x[y, z.M]$, z binds in M; in $x[u.v.M, z.N]$, u and v bind in M and z binds in N; in $\mathtt{inv}(x, y.M)$, y binds in M; in $\mathtt{dec}(x, y, z.M)$, z binds in M; and in $\langle M \dagger x.N \rangle$, x binds in N.

Certain constraints on the use of the term syntax will be evident once we present the typing rules; these constraints are captured by the notion of *well-formed term*, mentioned in Chapter 2 and defined in the case of G4ii as follows:

Definition 109 (Well-formed term) A term L is *well-formed* if in any subterm of the form

- $x[y, z.M]$, we have $x \neq y$, with x not free in M;

- $x[u.v.M, z.N]$, we have $u \neq v$, with x not free in M and not free in N;
- $\mathtt{inv}(x, y.M)$, we have x not free in M;
- $\mathtt{dec}(x, y, z.M)$, we have $x \neq y$, with both of them not free in M.

Definition 110 (Ordering on multi-sets of types) We compare multi-sets of types with the traditional multi-set ordering, denoted $<_{\mathsf{mul}}$, while types are compared according to their sizes (as trees).

For every rule of the logical sequent calculus G4ii, the multi-set of types appearing in the conclusion is greater than that of each premiss. Hence, we say that G4ii is *depth-bounded*. See [Dyc92] or [TS00] for details, and see the next section for the corresponding property in our version of G4ii with terms.

7.2.2 Typing

The next definition adds term notation to the rules for implication of G4ii; another view is that it shows how the untyped normal terms of the above grammar may be typed.

Definition 111 (Typing Rules for Normal Terms)

$$\frac{}{\Gamma, x{:}\, A \vdash x{:}\, A}\, Ax \qquad \frac{\Gamma, x{:}\, A \vdash M{:}\, B}{\Gamma \vdash \lambda x.M{:}\, A{\to}B}\, R{\to}$$

$$\frac{\Gamma, y{:}\, A, z{:}\, B \vdash M{:}\, E}{\Gamma, x{:}\, A{\to}B, y{:}\, A \vdash x[y, z.M]{:}\, E}\, L0{\to}$$

$$\frac{\Gamma, u{:}\, C, v{:}\, D{\to}B \vdash M{:}\, D \qquad \Gamma, z{:}\, B \vdash N{:}\, E}{\Gamma, x{:}\, (C{\to}D){\to}B \vdash x[u.v.M, z.N]{:}\, E}\, L{\to\to}$$

These rules only construct well-formed terms; e.g. the notation $\Gamma, x{:}\, A{\to}B, y{:}\, A$ in the conclusion of $L0{\to}$ forces $x \neq y$ and x to be not already declared in Γ (hence not free in M).

These rules are the extensions with terms of the logical rules of G4ii in [TS00] (note a slight difference of the $L{\to\to}$rule from that of [Dyc92]), with the variation that both in Ax and in $L0{\to}$the type A need not be atomic. In the rules $R{\to}$, $L0{\to}$and $L{\to\to}$the types $A{\to}B$, $A{\to}B$ and $(C{\to}D){\to}B$ respectively are *principal*; in $L0{\to}$ the type A is *auxiliary*. (This use of "auxiliary" is not to be confused with its use in Definition 108 to describe certain kinds of term.)

Remark 193 Note that, in both cases, terms reflect the structure of their typing derivation: for each derivable sequent $\Gamma \vdash M{:}\, A$ there is a unique derivation tree

(up to renaming, in sub-derivations, of the variables bound in M), which can be reconstructed using the structure of the term M that *represents* the proof. M is therefore called a *proof-term*.

Note that in every instance of a rule in Definition 111 with conclusion $\Gamma \vdash M\colon A$, each premiss $\Gamma' \vdash N\colon B$ is such that $m(\Gamma) \cup A >_{\mathsf{mul}} m(\Gamma') \cup B$, where $\cup$ denotes the union of multi-sets. As a consequence, given Γ and A, there are finitely many derivations concluding, for some (normal) term M, the sequent $\Gamma \vdash M\colon A$.

Definition 112 (Typing Rules for Auxiliary Terms)

$$\frac{\Gamma, y\colon C{\to}D \vdash M\colon E}{\Gamma, x\colon D \vdash \mathtt{inv}(x, y.M)\colon E}\, Inv \qquad \frac{\Gamma, z\colon (C{\to}D){\to}B \vdash M\colon A}{\Gamma, x\colon C, y\colon D{\to}B \vdash \mathtt{dec}(x, y, z.M)\colon A}\, Dec$$

$$\frac{\Gamma \vdash M\colon A \quad x\colon A, \Gamma \vdash N\colon B}{\Gamma \vdash \langle M \dagger x.N\rangle\colon B}\, Cut$$

These rules only construct well-formed terms; e.g. the notation $\Gamma, x\colon A$ in the conclusion of Inv forces x to be not declared in Γ and hence not free in M.

In the Cut-rule, we say that A is the *cut-type*. A derivation is *normal* if it uses only the primitive rules, i.e. those of Definition 111.

We will occasionally find it necessary to rename free variables. We can use for that the notion of substitution from Definition 43, but because of the well-formedness constraint we shall only use $(y\ x)M$ when y is not free in M.

As in λG3 and its derived systems, we have admissibility of weakening:

Lemma 194 *The following rule is height-preserving admissible both in the system of normal derivations and in the full system with auxiliary terms.*

$$\frac{\Gamma \vdash M\colon A}{\Gamma, y\colon B \vdash M\colon A}\, (\mathit{weak})$$

Proof: Routine induction on the height of the derivation of the premiss. Note that the notation $\Gamma, y\colon B$ forces y to be not declared in Γ and hence not free in M. □

7.3 Proof transformations & reduction rules

The starting point of this section is the admissibility in the (cut-free) logical sequent calculus G4ii of the following inference rules (i.e. the logical counter-part of the typing rules for auxiliary terms given in Definition 112):

$$\frac{\Gamma, C{\to}D \vdash E}{\Gamma, D \vdash E}\,Inv \qquad \frac{\Gamma, (C{\to}D){\to}B \vdash A}{\Gamma, C, D{\to}B \vdash A}\,Dec$$

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

The admissibility in G4ii of Inv alone can be proved by induction on the height of the derivation of the premiss. For the admissibility of Dec and Cut we can use a simultaneous induction, the admissibility of one rule being recursively used for the admissibility of the other. The measure now uses the multi-set of types appearing in the unique premiss for Dec and in the second premiss for Cut. In other words, the induction can be done on $\{\!\{\Gamma, (C{\to}D){\to}B, A\}\!\}$ for Dec and on $\{\!\{\Gamma, A, B\}\!\}$ for Cut.

We do not include here the detail of these proofs of admissibility, because the property turns out to be a consequence (Corollary 199) of our strong normalisation result for our calculus with terms.

Indeed, the admissibility property means, in our framework with terms, that a term M with auxiliary constructors `inv`, `dec` or `cut` can be transformed into another term M' with the same type in the same context that does not use these constructors. In other words, it means that the auxiliary typing rules are term-irrelevant admissible in the system of normal derivations. As described in Chapter 6 we extract from these proofs a rewrite system, such that the measures for induction mentioned above can be used (as part of a MPO —see section 7.4) to conclude their strong normalisation as well. We give in Fig. 7.1 and Fig. 7.2 the reduction systems that eliminate the auxiliary constructors `inv` and `dec`. All these rules, which we call system ivdc, will be part of the different variants that we are going to introduce.

$\mathtt{inv}(x, y.z)$	$\longrightarrow_{\mathsf{i}1}$	z
$\mathtt{inv}(x, y.y)$	$\longrightarrow_{\mathsf{i}2}$	$\lambda z.x$
$\mathtt{inv}(x, y.\lambda z.M)$	$\longrightarrow_{\mathsf{i}3}$	$\lambda z.\mathtt{inv}(x, y.M)$
$\mathtt{inv}(x, y.y[w, z.N])$	$\longrightarrow_{\mathsf{i}4}$	$(x\ z)N$
$\mathtt{inv}(x, y.y[u.v.M, z.N])$	$\longrightarrow_{\mathsf{i}5}$	$(x\ z)N$
$\mathtt{inv}(x, y.w[y, z.N])$	$\longrightarrow_{\mathsf{i}6}$	$w[u.v.x, z.\mathtt{inv}(x, y.N)]$
$\mathtt{inv}(x, y.y'[w, z.N])$	$\longrightarrow_{\mathsf{i}7}$	$y'[w, z.\mathtt{inv}(x, y.N)]$
$\mathtt{inv}(x, y.y'[u.v.M, z.N])$	$\longrightarrow_{\mathsf{i}8}$	$y'[u.v.\mathtt{inv}(x, y.M), z.\mathtt{inv}(x, y.N)]$

Figure 7.1: Reduction rules for `inv`-terms

The structure cut-reduction system follows the general pattern of cut-reduction systems described in Chapter 6: the rules can be split into three kinds ($\mathsf{Kind}_1, \mathsf{Kind}_2, \mathsf{Kind}_3$), according to whether they push cuts to the right, to the left, or they reduce logical cuts, breaking them into cuts on smaller types.

Here, owing to the particular inference rules of G4ii and the well-formedness

$\mathtt{dec}(x, y, z.w)$	$\longrightarrow_{\mathtt{d1}}$	w
$\mathtt{dec}(x, y, z.z)$	$\longrightarrow_{\mathtt{d2}}$	$\lambda v.v[x, w.y[w, u.u]]$
$\mathtt{dec}(x, y, z.\lambda w.M)$	$\longrightarrow_{\mathtt{d3}}$	$\lambda w.\mathtt{dec}(x, y, z.M)$
$\mathtt{dec}(x, y, z.w[u.v.M, w'.N])$	$\longrightarrow_{\mathtt{d4}}$	$w[u.v.\mathtt{dec}(x, y, z.M), w'.\mathtt{dec}(x, y, z.N)]$
$\mathtt{dec}(x, y, z.w[y', z'.M])$	$\longrightarrow_{\mathtt{d5}}$	$w[y', z'.\mathtt{dec}(x, y, z.M)]$
$\mathtt{dec}(x, y, z.z[y', z'.M])$	$\longrightarrow_{\mathtt{d6}}$	$y'[x, z''.y[z'', z'.\mathtt{inv}(z'', y'.M)]]$
$\mathtt{dec}(x, y, z.x'[z, z'.M])$	$\longrightarrow_{\mathtt{d7}}$	$x[u.v.v[x, z''.y[z'', w.w]], z'.\mathtt{dec}(x, y, z.M)]$
$\mathtt{dec}(x, y, z.z[u.v.M, z'.N])$	$\longrightarrow_{\mathtt{d8}}$	$\langle (x\ u)(y\ v)M \dagger y'.y[y', z'.N]\rangle$

Figure 7.2: Reduction rules for dec-terms

Kind_1

$\langle M \dagger x.x\rangle$	$\longrightarrow_{\mathtt{c1}}$	M
$\langle M \dagger x.y\rangle$	$\longrightarrow_{\mathtt{c2}}$	y
$\langle M \dagger x.\lambda y.N\rangle$	$\longrightarrow_{\mathtt{c3}}$	$\lambda y.\langle M \dagger x.N\rangle$
$\langle M \dagger x.y[z, w.N]\rangle$	$\longrightarrow_{\mathtt{c4}}$	$y[z, w.\langle \mathtt{inv}(w, y.M) \dagger x.N\rangle]$
$\langle M \dagger x.y[u.v.N', w.N]\rangle$	$\longrightarrow_{\mathtt{c5}}$	$y[u.v.P, w.\langle \mathtt{inv}(w, y.M) \dagger x.N\rangle]$ where $P = \langle \mathtt{dec}(u, v, y.M) \dagger x.N'\rangle$
$\langle \lambda z.M \dagger x.y[x, w.N]\rangle$	$\longrightarrow_{\mathtt{c6}}$	$y[u.v.P, w.\langle \mathtt{inv}(w, y.\lambda z.M) \dagger x.N\rangle]$ where $P = \langle u \dagger z.\mathtt{dec}(u, v, y.M)\rangle$
$\langle z \dagger x.y[x, w.N]\rangle$	$\longrightarrow_{\mathtt{c7}}$	$y[z, w.\langle z \dagger x.N\rangle]$

Kind_2

$\langle y[z, w.M] \dagger x.N\rangle$	$\longrightarrow_{\mathtt{c8}}$	$y[z, w.\langle M \dagger x.\mathtt{inv}(w, y.N)\rangle]$
$\langle y[u.v.M', w.M] \dagger x.N\rangle$	$\longrightarrow_{\mathtt{c9}}$	$y[u.v.M', w.\langle M \dagger x.\mathtt{inv}(w, y.N)\rangle]$

Figure 7.3: Cut-elimination rules cers/cears (Kind_1 and Kind_2)

Kind_3

$\langle \lambda y.M \dagger x.x[z, w.N]\rangle$	$\longrightarrow_{\mathsf{C}}$	$\langle\langle z \dagger y.M\rangle \dagger w.N\rangle$
$\langle \lambda y.M \dagger x.x[u.v.N', w.N]\rangle$	$\longrightarrow_{\mathsf{D}}$	$\langle\langle \lambda u.\langle \lambda z.\mathtt{inv}(z, y.M) \dagger v.N'\rangle \dagger y.M\rangle \dagger w.N\rangle$
$\langle y \dagger x.x[z, w.N]\rangle$	$\longrightarrow_{\mathsf{E}}$	$y[z, w'.\langle w' \dagger w.\mathtt{inv}(w', y.N)\rangle]$
$\langle y \dagger x.x[u.v.N', w.N]\rangle$	$\longrightarrow_{\mathsf{F}}$	$y[u'.v'.\langle u' \dagger u.P\rangle, w'.\langle w' \dagger w.\mathtt{inv}(w', y.N)\rangle]$
	where	$P := \mathtt{dec}(u', v', y.\langle \lambda y''.y[u.v.y'', z.z] \dagger v.N'\rangle)$

Figure 7.4: Cut-elimination rules cers (Kind_3)

Kind_3		
$\langle \lambda y.M \dagger x.x[z, w.N] \rangle$	$\longrightarrow_{\mathsf{C}}$	$\langle \langle z \dagger y.M \rangle \dagger w.N \rangle$
$\langle \lambda y.M \dagger x.x[u.v.N', w.N] \rangle$	$\longrightarrow_{\mathsf{D}}$	$\langle \langle \lambda u. \langle \lambda z. \mathtt{inv}(z, y.M) \dagger v.N' \rangle \dagger y.M \rangle \dagger w.N \rangle$
$\langle y \dagger x.x[z, w.N] \rangle$	$\longrightarrow_{\mathsf{E}}$	$y[z, w'. \langle w' \dagger w.\mathtt{inv}(w', y.N) \rangle]$
$\langle y \dagger x.x[u.v.N', w.N] \rangle$	$\longrightarrow_{\mathsf{G}}$	$y[u'.v'. \langle u' \dagger u.P' \rangle, w'. \langle w' \dagger w.\mathtt{inv}(w', y.N) \rangle]$ where $P' := \langle v' \dagger v.\mathtt{dec}(u', v', y.N') \rangle$

Figure 7.5: Cut-elimination rules cears (Kind_3)

constraints they impose on terms, the rewrite rules must use the auxiliary constructs `inv` and `dec`, rather than just propagate the cuts.

For the third kind of cut-reduction rules, reducing logical cuts, we have an alternative between rule $\longrightarrow_{\mathsf{F}}$ by $\longrightarrow_{\mathsf{G}}$, discussed in section 7.5, and leading to two different systems called rs (with $\longrightarrow_{\mathsf{F}}$) or ars (with $\longrightarrow_{\mathsf{G}}$).

Summing up :

Name of the System	Reduction Rules
ivdc	Fig. 7.1 and Fig. 7.2
$\mathsf{cers}/\mathsf{cears}$	Fig. 7.3, and Fig. 7.4/7.5
rs/ars	$\mathsf{ivdc} \cup \mathsf{cers}/\mathsf{ivdc} \cup \mathsf{cears}$

Lemma 195 *All rules of system* rs *and* ars *are such that well-formed terms reduce to well-formed terms.*

Proof: Routine. □

7.4 Subject reduction & strong normalisation

In this section we prove two fundamental properties of systems rs and ars. The first one is subject reduction and it guarantees that types are preserved by the reduction system. The second one is strong normalisation on typed terms of the rewrite systems of section 7.3, which guarantees that no infinite reduction sequence starts from a typed term.

The latter is proved using a simulation by MPO-reduction in a first-order syntax that encodes typing derivations. It is convenient to prove the simulation together with subject reduction since the proofs are based on the same case analysis.

The first-order syntax is given by the following infinite signature and its precedence relation:

$$\mathsf{C}^n(\_,\_) \succ \mathsf{D}^n(\_) \succ \cdots \succ \mathsf{C}^m(\_,\_) \succ \mathsf{D}^m(\_) \succ \mathsf{J}(\_) \succ \mathsf{K}(\_,\_) \succ \mathsf{I}(\_) \succ \star$$

for all multi-sets of types $n >_{\mathsf{mul}} m$.

The precedence relation on symbols provides an MPO $\gg$ on first-order terms.

Remark 196

1. The order on types (Definition 110) is terminating, so $>_{\mathsf{mul}}$ is terminating [DM79].
2. The order $>_{\mathsf{mul}}$ is terminating, so $\succ$ is also terminating.
3. The order $\succ$ is terminating, so the MPO $\gg$ is also terminating (Theorem 49).

Definition 113 (Encoding into the first-order syntax) Derivations are mapped to the first-order syntax according to Fig. 7.6. Note that since each sequent $\Gamma \vdash M\colon A$ has at most one derivation, we write $\phi(\Gamma \vdash M\colon A)$ for such a translation, and even $\phi(M)$ when the environment and the type are clear from context.

$\phi(\Gamma, x\colon A \vdash x\colon A)$	$:=$	$\star$
$\phi(\Gamma \vdash \lambda x.M\colon A{\to}B)$	$:=$	$\mathsf{I}(\phi(M))$
$\phi(\Gamma, x\colon A{\to}B, y\colon A \vdash x[y, z.M]\colon E)$	$:=$	$\mathsf{I}(\phi(M))$
$\phi(\Gamma, x\colon (C{\to}D){\to}B \vdash x[u.v.M, z.N]\colon E)$	$:=$	$\mathsf{K}(\phi(M), \phi(N))$
$\phi(\Gamma, x\colon D \vdash \mathtt{inv}(x, y.M)\colon E)$	$:=$	$\mathsf{J}(\phi(M))$
$\phi(\Gamma, x\colon C, y\colon D{\to}B \vdash \mathtt{dec}(x, y, z.M)\colon A)$	$:=$	$\mathsf{D}^k(\phi(\Gamma, z\colon (C{\to}D){\to}B \vdash M\colon A))$ where $k := \{\!\{\Gamma, (C{\to}D){\to}B, A\}\!\}$
$\phi(\Gamma \vdash \langle M \dagger x.N\rangle\colon B)$	$:=$	$\mathsf{C}^k(\phi(\Gamma \vdash M\colon A), \phi(x\colon A, \Gamma \vdash N\colon B))$ where $k := \{\!\{\Gamma, A, B\}\!\}$

Figure 7.6: Encoding into the first-order syntax

Observe that $\phi(M) = \phi((x\ y)M)$ for any renaming of M.

Theorem 197 *If* $\Gamma \vdash L\colon E$ *and* $L \longrightarrow_{\mathsf{rs,ars}} L'$, *then* $\Gamma \vdash L'\colon E$ *and* $\phi(L) \gg \phi(L')$.

Proof: By induction on the derivation of $\Gamma \vdash L\colon E$. We show only the cases of root-reduction.

i1 $\mathtt{inv}(x, y.z) \longrightarrow_{\mathsf{i1}} z$

The derivation

$$\dfrac{\dfrac{}{\Gamma', z{:}\, E, y{:}\, A{\to}B \vdash z{:}\, E}\, Ax}{\Gamma', z{:}\, E, x{:}\, B \vdash \mathtt{inv}(x, y.z){:}\, E}\, Inv$$

rewrites to

$$\dfrac{}{\Gamma', z{:}\, E, x{:}\, B \vdash z{:}\, E}\, Ax$$

Also, $\phi(L) = \mathsf{J}(\star) \gg \star = \phi(L')$ holds by the sub-term property of $\gg$.

i2 $\mathtt{inv}(x, y.y) \longrightarrow_{\mathsf{i2}} \lambda z.x$

The derivation

$$\dfrac{\dfrac{}{\Gamma', y{:}\, A{\to}B \vdash y{:}\, A{\to}B}\, Ax}{\Gamma', x{:}\, B \vdash \mathtt{inv}(x, y.y){:}\, A{\to}B}\, Inv$$

rewrites to

$$\dfrac{\dfrac{}{\Gamma', x{:}\, B, z{:}\, A \vdash x{:}\, B}\, Ax}{\Gamma', x{:}\, B \vdash \lambda z.x{:}\, A{\to}B}\, R{\to}$$

Also, $\phi(L) = \mathsf{J}(\star) \succ \mathsf{I}(\star) = \phi(L')$ holds by $\mathsf{J} \succ \mathsf{I}$.

i3 $\mathtt{inv}(x, y.\lambda z.M) \longrightarrow_{\mathsf{i3}} \lambda z.\mathtt{inv}(x, y.M)$ with $E = C{\to}D$

The derivation

$$\dfrac{\dfrac{\Gamma', y{:}\, A{\to}B, z{:}\, C \vdash M{:}\, D}{\Gamma', y{:}\, A{\to}B \vdash \lambda z.M{:}\, C{\to}D}\, R{\to}}{\Gamma', x{:}\, B \vdash \mathtt{inv}(x, y.\lambda z.M){:}\, C{\to}D}\, Inv$$

rewrites to

$$\dfrac{\dfrac{\Gamma', y{:}\, A{\to}B, z{:}\, C \vdash M{:}\, D}{\Gamma', x{:}\, B, z{:}\, C \vdash \mathtt{inv}(x, y.M){:}\, D}\, Inv}{\Gamma', x{:}\, B \vdash \lambda z.\mathtt{inv}(x, y.M){:}\, C{\to}D}\, R{\to}$$

Also, $\phi(L) = \mathsf{J}(\mathsf{I}(\phi(M))) \gg \mathsf{I}(\mathsf{J}(\phi(M))) = \phi(L')$ by $\mathsf{J} \succ \mathsf{I}$.

i4 $\mathtt{inv}(x, y.y[w, z.N]) \longrightarrow_{\mathsf{i4}} (x\ z)N$

The derivation

$$\dfrac{\dfrac{\Gamma', w{:}\, A, z{:}\, B \vdash N{:}\, E}{\Gamma', w{:}\, A, y{:}\, A{\to}B \vdash y[w, z.N]{:}\, E}\, L0{\to}}{\Gamma', w{:}\, A, x{:}\, B \vdash \mathtt{inv}(x, y.y[w, z.N]){:}\, E}\, Inv$$

rewrites to

$$\frac{\Gamma', w\colon A, z\colon B \vdash N\colon E}{\Gamma', w\colon A, x\colon B \vdash (x\ z)N\colon E}$$

by equivariance of typing systems.

Also, $\phi(M) = \mathsf{J}(\mathsf{I}(\phi(N)))\gg\phi(N) = \phi(M')$ holds by the sub-term property of $\gg$.

i5 $\mathtt{inv}(x, y.y[u.v.M, z.N]) \longrightarrow_{\mathsf{i5}} (x\ z)N$ with $A = C{\to}D$

The derivation

$$\dfrac{\dfrac{\Gamma', u\colon C, v\colon D{\to}B \vdash M\colon D \quad \Gamma', z\colon B \vdash N\colon E}{\Gamma', y\colon A{\to}B \vdash y[u.v.M, z.N]\colon E}\, L{\to\!\!\to}}{\Gamma', x\colon B \vdash \mathtt{inv}(x, y.y[u.v.M, z.N])\colon E}\, Inv$$

rewrites to

$$\frac{\Gamma', z\colon B \vdash N\colon E}{\Gamma', x\colon B \vdash (x\ z)N\colon E}$$

by equivariance of typing systems.

Also, $\phi(L) = \mathsf{J}(\mathsf{K}(\phi(M), \phi(N)))\gg\phi(N) = \phi(L')$ holds by the sub-term property of $\gg$.

i6 $\mathtt{inv}(x, y.w[y, z.N]) \longrightarrow_{\mathsf{i6}} w[u.v.x, z.\mathtt{inv}(x, y.N)]$

The derivation

$$\dfrac{\dfrac{\Gamma', y\colon A{\to}B, z\colon C \vdash N\colon E}{\Gamma', w\colon (A{\to}B){\to}C, y\colon A{\to}B \vdash w[y, z.N]\colon E}\, L0{\to}}{\Gamma', w\colon (A{\to}B){\to}C, x\colon B \vdash \mathtt{inv}(x, y.w[y, z.N])\colon E}\, Inv$$

rewrites to

$$\dfrac{\dfrac{}{\Gamma', x\colon B, u\colon A, v\colon B{\to}C \vdash x\colon B}\, Ax \qquad \dfrac{\Gamma', y\colon A{\to}B, z\colon C \vdash N\colon E}{\Gamma', x\colon B, z\colon C \vdash \mathtt{inv}(x, y.N)\colon E}\, Inv}{\Gamma', w\colon (A{\to}B){\to}C, x\colon B \vdash w[u.v.x, z.\mathtt{inv}(x, y.N)]\colon E}\, L{\to\!\!\to}$$

Also, $\phi(L) = \mathsf{J}(\mathsf{I}(\phi(N)))\gg\mathsf{K}(\star, \mathsf{J}(\phi(N))) = \phi(L')$ by $\mathsf{J} \succ \mathsf{K}, \star$.

i7 $\mathtt{inv}(x, y.y'[w, z.N]) \longrightarrow_{\mathsf{i7}} y'[w, z.\mathtt{inv}(x, y.N)]$

The derivation

$$\dfrac{\dfrac{\Gamma', w\colon C, z\colon D, y\colon A{\to}B \vdash N\colon E}{\Gamma', w\colon C, y'\colon C{\to}D, y\colon A{\to}B \vdash y'[w, z.N]\colon E}\, L0{\to}}{\Gamma', w\colon C, y'\colon C{\to}D, x\colon B \vdash \mathtt{inv}(x, y.y'[w, z.N])\colon E}\, Inv$$

rewrites to

$$\dfrac{\dfrac{\Gamma', w{:}\, C, z{:}\, D, y{:}\, A{\to}B \vdash N{:}\, E}{\Gamma', w{:}\, C, z{:}\, D, x{:}\, B \vdash \mathtt{inv}(x, y.N){:}\, E}\, Inv}{\Gamma', w{:}\, C, y'{:}\, C{\to}D, x{:}\, B \vdash y'[w, z.\mathtt{inv}(x, y.N)]{:}\, E}\, L0{\to}$$

Also, $\phi(L) = \mathsf{J}(\mathsf{I}(\phi(N))) \gg \mathsf{I}(\mathsf{J}(\phi(N))) = \phi(L')$ by $\mathsf{J} \succ \mathsf{I}$.

i8 $\mathtt{inv}(x, y.y'[u.v.M, z.N]) \longrightarrow_{\mathsf{i8}} y'[u.v.\mathtt{inv}(x, y.M), z.\mathtt{inv}(x, y.N)]$

The derivation

$$\dfrac{\dfrac{\Gamma', y{:}\, A{\to}B, u{:}\, C, v{:}\, D{\to}B' \vdash M{:}\, D \qquad \Gamma', y{:}\, A{\to}B, z{:}\, B' \vdash N{:}\, E}{\Gamma', y'{:}\, (C{\to}D){\to}B', y{:}\, A{\to}B \vdash y'[u.v.M, z.N]{:}\, E}\, L{\to\to}}{\Gamma', y'{:}\, (C{\to}D){\to}B', x{:}\, B \vdash \mathtt{inv}(x, y.y'[u.v.M, z.N]){:}\, E}\, Inv$$

rewrites to

$$\dfrac{\dfrac{\Gamma', y{:}\, A{\to}B, u{:}\, C, v{:}\, D{\to}B' \vdash M{:}\, D}{\Gamma', x{:}\, B, u{:}\, C, v{:}\, D{\to}B' \vdash \mathtt{inv}(x, y.M){:}\, D}\, Inv \qquad \dfrac{\Gamma', y{:}\, A{\to}B, z{:}\, B' \vdash N{:}\, E}{\Gamma', x{:}\, B, z{:}\, B' \vdash \mathtt{inv}(x, y.N){:}\, E}\, Inv}{\Gamma', y'{:}\, (C{\to}D){\to}B', x{:}\, B \vdash y'[u.v.\mathtt{inv}(x, y.M), z.\mathtt{inv}(x, y.N)]{:}\, E}\, L{\to\to}$$

Also, $\phi(L) = \mathsf{J}(\mathsf{K}(\phi(M), \phi(N))) \gg \mathsf{K}(\mathsf{J}(\phi(M)), \mathsf{J}(\phi(N))) = \phi(L')$ by $\mathsf{J} \succ \mathsf{K}$.

d1 $\mathtt{dec}(x, y, z.w) \longrightarrow_{\mathtt{d1}} w$

The derivation

$$\dfrac{\dfrac{}{\Gamma', w{:}\, E, z{:}\, (C{\to}D){\to}B \vdash w{:}\, E}\, Ax}{\Gamma', w{:}\, E, x{:}\, C, y{:}\, D{\to}B \vdash \mathtt{dec}(x, y, z.w){:}\, E}\, Dec$$

rewrites to

$$\dfrac{}{\Gamma', w{:}\, E, x{:}\, C, y{:}\, D{\to}B \vdash w{:}\, E}\, Ax$$

Also, $\phi(L) = \mathsf{D}^m(\star) \gg \star = \phi(L')$, where $m := \{\!\{\Gamma', E, (C{\to}D){\to}B, E\}\!\}$.

d2 $\mathtt{dec}(x, y, z.z) \longrightarrow_{\mathtt{d2}} \lambda v.v[x, w.y[w, u.u]]$.

The derivation

$$\dfrac{\dfrac{}{\Gamma', z{:}\, (C{\to}D){\to}B \vdash z{:}\, (C{\to}D){\to}B}\, Ax}{\Gamma', x{:}\, C, y{:}\, D{\to}B \vdash \mathtt{dec}(x, y, z.z){:}\, E}\, Dec$$

rewrites to

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma', x{:}\,C, w{:}\,D, u{:}\,B \vdash u{:}\,B}\,Ax}{\Gamma', x{:}\,C, w{:}\,D, y{:}\,D{\to}B \vdash y[w, u.u]{:}\,B}\,L0{\to}}{\Gamma', x{:}\,C, y{:}\,D{\to}B, v{:}\,C{\to}D \vdash v[x, w.y[w, u.u]]{:}\,B}\,L0{\to}}{\Gamma', x{:}\,C, y{:}\,D{\to}B \vdash \lambda v.v[x, w.y[w, u.u]]{:}\,(C{\to}D){\to}B}\,R{\to}$$

Also, $\phi(L) = \mathsf{D}^m(\star){\gg}\mathsf{I}(\mathsf{I}(\star))) = \phi(L')$,
where $m :=\ \{\!\{\Gamma', (C{\to}D){\to}B, (C{\to}D){\to}B\}\!\}$, by $\mathsf{D}^m \succ \mathsf{I}$.

d3 $\mathtt{dec}(x, y, z.\lambda w.M) \longrightarrow_{\mathtt{d3}} \lambda w.\mathtt{dec}(x, y, z.M)$.

The derivation

$$\dfrac{\dfrac{\Gamma', z{:}\,(C{\to}D){\to}B, w{:}\,E_1 \vdash M{:}\,E_2}{\Gamma', z{:}\,(C{\to}D){\to}B \vdash \lambda w.M{:}\,E_1{\to}E_2}\,R{\to}}{\Gamma', x{:}\,C, y{:}\,D{\to}B \vdash \mathtt{dec}(x, y, z.\lambda w.M){:}\,E_1{\to}E_2}\,Dec$$

rewrites to

$$\dfrac{\dfrac{\Gamma', z{:}\,(C{\to}D){\to}B, w{:}\,E_1 \vdash M{:}\,E_2}{\Gamma', x{:}\,C, y{:}\,D{\to}B, w{:}\,E_1 \vdash \mathtt{dec}(x, y, z.M){:}\,E_2}\,Dec}{\Gamma', x{:}\,C, y{:}\,D{\to}B \vdash \lambda w.\mathtt{dec}(x, y, z.M){:}\,E_1{\to}E_2}\,R{\to}$$

Let $m :=\ \{\!\{\Gamma', (C{\to}D){\to}B, E_1{\to}E_2\}\!\}$ and $n :=\ \{\!\{\Gamma', (C{\to}D){\to}B, E_1, E_2\}\!\}$.
We have

$$\phi(L) = \mathsf{D}^m(\mathsf{I}(\phi(M))){\gg}\mathsf{I}(\mathsf{D}^n(\phi(M))) = \phi(L')$$

since $\mathsf{D}^m \succ \mathsf{I}, \mathsf{D}^n$ because $m{>}_{\mathsf{mul}}n$.

d4 $\mathtt{dec}(x, y, z.w[u.v.M, w'.N]) \longrightarrow_{\mathtt{d4}} w[u.v.\mathtt{dec}(x, y, z.M), w'.\mathtt{dec}(x, y, z.N)]$.

The derivation

$$\dfrac{\dfrac{\Gamma', v{:}\,F, u{:}\,G{\to}H, z{:}\,(C{\to}D){\to}B \vdash M{:}\,G \quad \Gamma', w'{:}\,H, z{:}\,(C{\to}D){\to}B \vdash N{:}\,E}{\Gamma', w{:}\,(F{\to}G){\to}H, z{:}\,(C{\to}D){\to}B \vdash w[u.v.M, w'.N]{:}\,E}\,L{\to\to}}{\Gamma', w{:}\,(F{\to}G){\to}H, x{:}\,C, y{:}\,D{\to}B \vdash \mathtt{dec}(x, y, z.w[u.v.M, w'.N]){:}\,E}\,Dec$$

rewrites to

$$\dfrac{\dfrac{\Gamma', v{:}\,F, u{:}\,G{\to}H, z{:}\,(C{\to}D){\to}B \vdash M{:}\,G}{\Gamma', v{:}\,F, u{:}\,G{\to}H, x{:}\,C, y{:}\,D{\to}B \vdash M'{:}\,G}\,Dec \qquad \dfrac{\Gamma', w'{:}\,H, z{:}\,(C{\to}D){\to}B \vdash N{:}\,E}{\Gamma', w'{:}\,H, x{:}\,C, y{:}\,D{\to}B \vdash N'{:}\,E}\,Dec}{\Gamma', w{:}\,(F{\to}G){\to}H, x{:}\,C, y{:}\,D{\to}B \vdash w[u.v.M', w'.N']{:}\,G}\,L{\to\to}$$

with $M' :=\ \mathtt{dec}(x, y, z.M)$ and $N' :=\ \mathtt{dec}(x, y, z.N)$.

Let $k := \{\!\{\Gamma', (F{\to}G){\to}H, (C{\to}D){\to}B, E\}\!\}$ and
$m := \{\!\{\Gamma', F, G{\to}H, (C{\to}D){\to}B, G\}\!\}$ and $n := \{\!\{\Gamma', H, (C{\to}D){\to}B, E\}\!\}$.
We have

$$\phi(L) = \mathsf{D}^k(\mathsf{K}(\phi(M), \phi(N))) \gg \mathsf{K}(\mathsf{D}^m(\phi(M), \mathsf{D}^n(\phi(N))) = \phi(L')$$

since $\mathsf{D}^k \succ \mathsf{K}, \mathsf{D}^m, \mathsf{D}^n$ because $k >_{\mathsf{mul}} m, n$.

d5 $\mathtt{dec}(x, y, z.w[y', z'.M]) \longrightarrow_{\mathsf{d5}} w[y', z'.\mathtt{dec}(x, y, z.M)]$.

The derivation

$$\dfrac{\dfrac{\Gamma', y'\!: F, z'\!: G, z\!: (C{\to}D){\to}B \vdash M\!: E}{\Gamma', y'\!: F, w\!: F{\to}G, z\!: (C{\to}D){\to}B \vdash w[y', z'.M]\!: E}\, L0{\to}}{\Gamma', y'\!: F, w\!: F{\to}G, x\!: C, y\!: D{\to}B \vdash \mathtt{dec}(x, y, z.w[y', z'.M])\!: E}\, Dec$$

rewrites to

$$\dfrac{\dfrac{\Gamma', y'\!: F, z'\!: G, z\!: (C{\to}D){\to}B \vdash M\!: E}{\Gamma', y'\!: F, z'\!: G, x\!: C, y\!: D{\to}B \vdash \mathtt{dec}(x, y, z.M)\!: E}\, Dec}{\Gamma', y'\!: F, w\!: F{\to}G, x\!: C, y\!: D{\to}B \vdash w[y', z'.\mathtt{dec}(x, y, z.M)]\!: E}\, L0{\to}$$

Let $k := \{\!\{\Gamma', F, F{\to}G, (C{\to}D){\to}B, E\}\!\}$ and
$m := \{\!\{\Gamma', F, G, (C{\to}D){\to}B, E\}\!\}$. We have

$$\phi(L) = \mathsf{D}^k(\mathsf{I}(\phi(M))) \gg \mathsf{I}(\mathsf{D}^m(\phi(M))) = \phi(L')$$

since $\mathsf{D}^k \succ \mathsf{I}, \mathsf{D}^m$ because $k >_{\mathsf{mul}} m$.

d6 $\mathtt{dec}(x, y, z.z[y', z'.M]) \longrightarrow_{\mathsf{d6}} y'[x, z''.y[z'', z'.\mathtt{inv}(z'', y'.M)]]$.

The derivation

$$\dfrac{\dfrac{\Gamma', y'\!: C{\to}D, z'\!: B \vdash M\!: E}{\Gamma', z\!: (C{\to}D){\to}B, y'\!: C{\to}D \vdash z[y', z'.M]\!: E}\, L0{\to}}{\Gamma', x\!: C, y\!: D{\to}B, y'\!: C{\to}D \vdash \mathtt{dec}(x, y, z.z[y', z'.M])\!: E}\, Dec$$

rewrites to

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma', y'\!: C{\to}D, z'\!: B \vdash M\!: E}{\Gamma', z''\!: D, z'\!: B \vdash \mathtt{inv}(z'', y'.M)\!: E}\, Inv}{\Gamma', z''\!: D, y\!: D{\to}B \vdash y[z'', z'.\mathtt{inv}(z'', y'.M)]\!: E}\, L0{\to}}{\Gamma', x\!: C, z''\!: D, y\!: D{\to}B \vdash y[z'', z'.\mathtt{inv}(z'', y'.M)]\!: E}\, \mathsf{weak}}{\Gamma', y'\!: C{\to}D, x\!: C, y\!: D{\to}B \vdash y'[x, z''.y[z'', z'.\mathtt{inv}(z'', y'.M)]]\!: E}\, L0{\to}$$

Also, $\phi(L) = \mathsf{D}^k(\mathsf{I}(\phi(M))) \gg \mathsf{I}(\mathsf{I}(\mathsf{J}(\phi(M)))) = \phi(L')$ since $\mathsf{D}^k \succ \mathsf{I}, \mathsf{J}$, where $k := \{\!\{\Gamma', (C{\to}D){\to}B, C{\to}D, E\}\!\}$.

d7 $\mathtt{dec}(x,y,z.x'[z,z'.M]) \longrightarrow_{\mathtt{d7}} x[u.v.v[x,z''.y[z'',w.w]],z'.\mathtt{dec}(x,y,z.M)]$.
The derivation

$$\dfrac{\dfrac{\Gamma', z\colon (C\to D)\to B, z'\colon A \vdash M\colon E}{\Gamma', x'\colon ((C\to D)\to B)\to A, z\colon (C\to D)\to B \vdash x'[z,z'.M]\colon E}\,L0\to}{\Gamma', x'\colon ((C\to D)\to B)\to A, x\colon C, y\colon D\to B \vdash \mathtt{dec}(x,y,z.x'[z,z'.M])\colon E}\,Dec$$

rewrites to

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma''\vdash v[x,z''.y[z'',w.w]]\colon B} \qquad \dfrac{\Gamma', z\colon (C\to D)\to B, z'\colon A \vdash M\colon E}{\Gamma', x\colon C, y\colon D\to B, z'\colon A \vdash \mathtt{dec}(x,y,z.M)\colon E}\,Dec}{\Gamma', x\colon ((C\to D)\to B)\to A, y\colon C, z\colon D\to B \vdash x[u.v.v[x,z''.y[z'',w.w]],z'.\mathtt{dec}(x,y,z.M)]\colon E}\,L{\to}{\to}$$

where $\Gamma'' := \Gamma', x\colon C, y\colon D\to B, u\colon B\to A, v\colon C\to D$ and $\mathcal{D}$ is the following derivation:

$$\dfrac{\dfrac{\dfrac{}{\Gamma', x\colon C, w\colon B, u\colon B\to A, z''\colon D \vdash w\colon B}\,Ax}{\Gamma', x\colon C, y\colon D\to B, u\colon B\to A, z''\colon D \vdash y[z'',w.w]\colon B}\,L0\to}{\Gamma', x\colon C, y\colon D\to B, u\colon B\to A, v\colon C\to D \vdash v[x,z''.y[z'',w.w]]\colon B}\,L0\to$$

Let $k := \{\!\{\Gamma', (C\to D)\to B, ((C\to D)\to B)\to A, E\}\!\}$ and
$m := \{\!\{\Gamma', (C\to D)\to B, A, E\}\!\}$. We have

$$\phi(L) = \mathsf{D}^k(\mathsf{I}(\phi(M)))\gg\mathsf{K}(\mathsf{I}(\mathsf{I}(\star)),\mathsf{D}^m(\phi(M))) = \phi(L')$$

since $\mathsf{D}^k \succ \mathsf{K},\mathsf{I},\star,\mathsf{D}^m$ because $k>_{\mathsf{mul}} m$.

d8 $\mathtt{dec}(x,y,z.z[u.v.M,z'.N]) \longrightarrow_{\mathtt{d8}} \langle (y\ u)(x\ v)M \dagger y'.y[y',z'.N]\rangle$.
The derivation

$$\dfrac{\dfrac{\Gamma', v\colon C, u\colon D\to B \vdash M\colon D \qquad \Gamma', z'\colon B \vdash N\colon E}{\Gamma', z\colon (C\to D)\to B \vdash z[u.v.M,z'.N]\colon E}\,L{\to}{\to}}{\Gamma', x\colon C, y\colon D\to B \vdash \mathtt{dec}(x,y,z.z[u.v.M,z'.N])\colon E}\,Dec$$

rewrites to

$$\dfrac{\dfrac{\Gamma', v\colon C, u\colon D\to B \vdash M\colon D}{\Gamma', x\colon C, y\colon D\to B \vdash (y\ u)(x\ v)M\colon D} \qquad \dfrac{\dfrac{\Gamma', z'\colon B \vdash N\colon E}{y'\colon D, \Gamma', x\colon C, z'\colon B \vdash N\colon E}\,\mathsf{weak}}{y'\colon D, \Gamma', x\colon C, y\colon D\to B \vdash y[y',z'.N]\colon E}\,L0\to}{\Gamma', x\colon C, y\colon D\to B \vdash \langle (y\ u)(x\ v)M \dagger y'.y[y',z'.N]\rangle\colon E}\,Cut$$

Let $k := \{\!\{\Gamma', (C\to D)\to B, E\}\!\}$ and $j := \{\!\{\Gamma', D, C, D\to B, E\}\!\}$. We have

$$\phi(L) = \mathsf{D}^k(\mathsf{K}(\phi(M),\phi(N)))\gg\mathsf{C}^j(\phi(M),\mathsf{I}(\phi(N))) = \phi(L')$$

since $\mathsf{D}^k \succ \mathsf{C}^j,\mathsf{I}$ because $k>_{\mathsf{mul}} j$.

$\mathsf{c1}$ $\langle M \dagger x.x\rangle \longrightarrow_{\mathsf{c1}} M$.

The derivation

$$\dfrac{\Gamma \vdash M\colon A \qquad \dfrac{}{\Gamma, x\colon A \vdash x\colon A}\,Ax}{\Gamma \vdash \langle M \dagger x.x\rangle\colon A}\,Cut$$

rewrites to

$$\Gamma \vdash M\colon A$$

Also, $\phi(L) = \mathsf{C}^m(\phi(M), \star) \gg \phi(M) = \phi(L')$, where $m = \{\!\{\Gamma, A, A\}\!\}$.

$\mathsf{c2}$ $\langle M \dagger x.y\rangle \longrightarrow_{\mathsf{c2}} y$.

The derivation

$$\dfrac{\Gamma', y\colon E \vdash M\colon A \qquad \dfrac{}{\Gamma', y\colon E, x\colon A \vdash y\colon E}\,Ax}{\Gamma', y\colon E \vdash \langle M \dagger x.y\rangle\colon E}\,Cut$$

rewrites to

$$\Gamma', y\colon E \vdash y\colon E$$

Also, $\phi(L) = \mathsf{C}^m(\phi(M), \star) \gg \star = \phi(L')$, where $m := \{\!\{\Gamma', E, A, E\}\!\}$.

$\mathsf{c3}$ $\langle M \dagger x.\lambda y.N\rangle \longrightarrow_{\mathsf{c3}} \lambda y.\langle M \dagger x.N\rangle$.

The derivation

$$\dfrac{\Gamma \vdash M\colon A \qquad \dfrac{x\colon A, \Gamma, y\colon C \vdash N\colon D}{x\colon A, \Gamma \vdash \lambda y.N\colon C{\to}D}\,R{\to}}{\Gamma \vdash \langle M \dagger x.\lambda y.N\rangle\colon C{\to}D}\,Cut$$

rewrites to

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash M\colon A}{\Gamma, y\colon C \vdash M\colon A}\,(\mathsf{weak}) \qquad x\colon A, \Gamma, y\colon C \vdash N\colon D}{\Gamma, y\colon C \vdash \langle M \dagger x.N\rangle\colon D}\,Cut}{\Gamma \vdash \lambda y.\langle M \dagger x.N\rangle\colon C{\to}D}\,R{\to}$$

Let $k := \{\!\{A, \Gamma, C{\to}D\}\!\}$ and $j := \{\!\{A, \Gamma, C, D\}\!\}$. We have

$$\phi(L) = \mathsf{C}^k(\phi(M), \mathsf{I}(\phi(N))) \gg \mathsf{I}(\mathsf{C}^j(\phi(M), \phi(N))) = \phi(L')$$

since $\mathsf{C}^k \succ \mathsf{I}, \mathsf{C}^j$ because $k >_{\mathsf{mul}} j$.

c4 $\langle M \dagger x.z[y,w.N]\rangle \longrightarrow_{\mathsf{c4}} z[y,w.\langle \mathtt{inv}(w,z.M)\dagger x.N\rangle]$.

The derivation

$$\dfrac{\Gamma', y{:}\,C, z{:}\,C{\to}B \vdash M{:}\,A \qquad \dfrac{x{:}\,A, \Gamma', y{:}\,C, w{:}\,B \vdash N{:}\,E}{x{:}\,A, \Gamma', y{:}\,C, z{:}\,C{\to}B \vdash z[y,w.N]{:}\,E}\,L0{\to}}{\Gamma', y{:}\,C, z{:}\,C{\to}B \vdash \langle M \dagger x.z[y,w.N]\rangle{:}\,E}\,Cut$$

rewrites to

$$\dfrac{\dfrac{\Gamma', y{:}\,C, z{:}\,C{\to}B \vdash M{:}\,A}{\Gamma', y{:}\,C, w{:}\,B \vdash \mathtt{inv}(w,z.M){:}\,A}\,Inv \qquad x{:}\,A, \Gamma', y{:}\,C, w{:}\,B \vdash N{:}\,E}{\dfrac{\Gamma', y{:}\,C, w{:}\,B \vdash \langle \mathtt{inv}(w,z.M)\dagger x.N\rangle{:}\,E}{\Gamma', y{:}\,C, z{:}\,C{\to}B \vdash z[y,w.\langle \mathtt{inv}(w,z.M)\dagger x.N\rangle]{:}\,E}\,L0{\to}}\,Cut$$

Let $k := \{\!\{A, \Gamma', C, C{\to}B, E\}\!\}$ and $j := \{\!\{A, \Gamma', C, B, E\}\!\}$. We have

$$\phi(L) = \mathsf{C}^k(\phi(M), \mathsf{I}(\phi(N))) \gg \mathsf{I}(\mathsf{C}^j(\mathsf{J}(\phi(M)), \phi(N)) = \phi(L')$$

since $\mathsf{C}^k \succ \mathsf{I}, \mathsf{C}^j, \mathsf{J}$ because $k >_{\mathsf{mul}} j$.

c5 $\langle M \dagger x.y[u.v.N, z.N']\rangle$
$\longrightarrow_{\mathsf{c5}} y[u.v.\langle \mathtt{dec}(v,u,y.M)\dagger x.N\rangle, z.\langle \mathtt{inv}(z,y.M)\dagger x.N'\rangle]$.

The derivation

$$\dfrac{\Gamma', y{:}\,(C{\to}D){\to}B \vdash M{:}\,A \qquad \dfrac{x{:}\,A, \Gamma', v{:}\,C, u{:}\,D{\to}B \vdash N{:}\,D \quad x{:}\,A, \Gamma', z{:}\,B \vdash N'{:}\,E}{x{:}\,A, \Gamma', y{:}\,(C{\to}D){\to}B \vdash y[u.v.N, z.N']{:}\,E}\,L{\to\to}}{\Gamma', y{:}\,(C{\to}D){\to}B \vdash \langle M \dagger x.y[u.v.N, z.N']\rangle{:}\,E}\,Cut$$

rewrites to

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma', v{:}\,C, u{:}\,D{\to}B \vdash \langle \mathtt{dec}(v,u,y.M)\dagger x.N\rangle{:}\,D} \qquad \dfrac{\mathcal{D}'}{\Gamma', z{:}\,B \vdash \langle \mathtt{inv}(z,y.M)\dagger x.N'\rangle{:}\,E}}{\Gamma', y{:}\,(C{\to}D){\to}B \vdash y[u.v.\langle \mathtt{dec}(v,u,y.M)\dagger x.N\rangle, z.\langle \mathtt{inv}(z,y.M)\dagger x.N'\rangle]{:}\,E}\,L{\to\to}$$

where $\mathcal{D}$ is the following derivation:

$$\dfrac{\dfrac{\Gamma', y{:}\,(C{\to}D){\to}B \vdash M{:}\,A}{\Gamma', v{:}\,C, u{:}\,D{\to}B \vdash \mathtt{dec}(v,u,y.M){:}\,A}\,Dec \qquad x{:}\,A, \Gamma', v{:}\,C, u{:}\,D{\to}B \vdash N{:}\,D}{\Gamma', v{:}\,C, u{:}\,D{\to}B \vdash \langle \mathtt{dec}(v,u,y.M)\dagger x.N\rangle{:}\,D}\,Cut$$

and $\mathcal{D}'$ is the following derivation:

$$\dfrac{\dfrac{\Gamma', y{:}\,(C{\to}D){\to}B \vdash M{:}\,A}{\Gamma', z{:}\,B \vdash \mathtt{inv}(z,y.M){:}\,A}\,Inv \qquad x{:}\,A, \Gamma', z{:}\,B \vdash N'{:}\,E}{\Gamma', z{:}\,B \vdash \langle \mathtt{inv}(z,y.M)\dagger x.N'\rangle{:}\,E}\,Cut$$

Let $k := \{\!\{A, \Gamma', (C \to D) \to B, E\}\!\}$ and $j := \{\!\{A, \Gamma', C, D \to B, D\}\!\}$ and $i := \{\!\{A, \Gamma', B, E\}\!\}$ and $h := \{\!\{\Gamma', (C \to D) \to B, A\}\!\}$. We have

$$\begin{aligned}\phi(L) &= \mathsf{C}^k(\phi(M), \mathsf{K}(\phi(N'), \phi(N))) \\ &\gg \mathsf{K}(\mathsf{C}^j(\mathsf{D}^h(\phi(M)), \phi(N')), \mathsf{C}^i(\mathsf{J}(\phi(M)), \phi(N)) = \phi(L')\end{aligned}$$

since $\mathsf{C}^k \succ \mathsf{K}, \mathsf{J}, \mathsf{C}^j, \mathsf{C}^i, \mathsf{D}^h$ because $k >_{\mathsf{mul}} j, h, i$.

c6 $\langle \lambda z.M \dagger x.y[x, w.N] \rangle$
$\longrightarrow_{\mathsf{c6}} y[u.v.\, \langle u \dagger w.\mathtt{dec}(w, v, y.M) \rangle, w.\, \langle \mathtt{inv}(w, y.\lambda z.M) \dagger x.N \rangle]$.

The derivation

$$\dfrac{\dfrac{z{:}\,C, \Gamma', y{:}\,(C{\to}D){\to}B \vdash M{:}\,D}{\Gamma', y{:}\,(C{\to}D){\to}B \vdash \lambda z.M{:}\,C{\to}D}\, R{\to} \qquad \dfrac{x{:}\,C{\to}D, \Gamma', w{:}\,B \vdash N{:}\,E}{x{:}\,C{\to}D, \Gamma', y{:}\,(C{\to}D){\to}B \vdash y[x, w.N]{:}\,E}\, L0{\to}}{\Gamma', y{:}\,(C{\to}D){\to}B \vdash \langle \lambda z.M \dagger x.y[x, w.N] \rangle{:}\,E}\, Cut$$

rewrites to

$$\dfrac{\dfrac{\mathcal{D}}{\Gamma', u{:}\,C, v{:}\,D{\to}B \vdash M'{:}\,D} \qquad \dfrac{\mathcal{D}'}{\Gamma', w{:}\,B \vdash N'{:}\,E}}{\Gamma', y{:}\,(C{\to}D){\to}B \vdash y[u.v.M', w.N']{:}\,E}\, L{\to\to}$$

where $M' := \langle u \dagger w.\mathtt{dec}(w, v, y.M) \rangle$, $N' := \langle \mathtt{inv}(w, y.\lambda z.M) \dagger x.N \rangle$, $\mathcal{D}$ is the following derivation:

$$\dfrac{\dfrac{}{\Gamma', u{:}\,C, v{:}\,D{\to}B \vdash u{:}\,C}\, Ax \qquad \dfrac{u{:}\,C, \Gamma', y{:}\,(C{\to}D){\to}B \vdash M{:}\,D}{\Gamma', u{:}\,C, w{:}\,C, v{:}\,D{\to}B \vdash \mathtt{dec}(w, v, y.M){:}\,D}\, Dec}{\Gamma', u{:}\,C, v{:}\,D{\to}B \vdash \langle u \dagger w.\mathtt{dec}(w, v, y.M) \rangle{:}\,D}\, Cut$$

and $\mathcal{D}'$ is the following derivation:

$$\dfrac{\dfrac{\Gamma', y{:}\,A{\to}B \vdash \lambda z.M{:}\,C{\to}D}{\Gamma', w{:}\,B \vdash \mathtt{inv}(w, y.\lambda z.M){:}\,C{\to}D}\, Inv \qquad x{:}\,C{\to}D, \Gamma', w{:}\,B \vdash N{:}\,E}{\Gamma', w{:}\,B \vdash \langle \mathtt{inv}(w, y.\lambda z.M) \dagger x.N \rangle{:}\,E}\, Cut$$

Let $k := \{\!\{C{\to}D, \Gamma', (C{\to}D){\to}B, E\}\!\}$ and $j := \{\!\{\Gamma', C, C, D{\to}B, D\}\!\}$ and $h := \{\!\{C, \Gamma', (C{\to}D){\to}B, D\}\!\}$ and $i := \{\!\{C{\to}D, \Gamma', B, E\}\!\}$. We have

$$\begin{aligned}\phi(L) &= \mathsf{C}^k(\mathsf{I}(\phi(M)), \mathsf{I}(\phi(N))) \\ &\gg \mathsf{K}(\mathsf{C}^j(\star, \mathsf{D}^h(\phi(M))), \mathsf{C}^i(\mathsf{J}(\mathsf{I}(\phi(M))), \phi(N))) = \phi(L')\end{aligned}$$

since $\mathsf{C}^k \succ \mathsf{K}, \star, \mathsf{J}, \mathsf{I}, \mathsf{C}^j, \mathsf{C}^i, \mathsf{D}^h$ because $k >_{\mathsf{mul}} j, i, h$.

c7 $\langle x \dagger y.z[y,w.M]\rangle \longrightarrow_{\mathsf{c7}} z[y,w.\langle x \dagger y.M\rangle]$.

The derivation

$$\dfrac{\dfrac{}{\Gamma', x\colon A, z\colon A{\to}B \vdash x\colon A}\,Ax \qquad \dfrac{x\colon A, y\colon A, w\colon B, \Gamma' \vdash M\colon E}{x\colon A, y\colon A, z\colon A{\to}B, \Gamma' \vdash z[y,w.M]\colon E}\,L0{\to}}{\Gamma', x\colon A, z\colon A{\to}B \vdash \langle x \dagger y.z[y,w.M]\rangle\colon E}\,Cut$$

rewrites to

$$\dfrac{\dfrac{}{\Gamma', x\colon A, w\colon B \vdash x\colon A}\,Ax \qquad x\colon A, y\colon A, w\colon B, \Gamma' \vdash M\colon E}{\dfrac{\Gamma', x\colon A, w\colon B \vdash \langle x \dagger y.M\rangle\colon E}{\Gamma', x\colon A, z\colon A{\to}B \vdash z[y,w.\langle x \dagger y.M\rangle]\colon E}\,L0{\to}}\,Cut$$

Let $k := \{\!\{A, A, A{\to}B, \Gamma', E\}\!\}$ and $j := \{\!\{A, A, B, \Gamma', E\}\!\}$. We have

$$\phi(L) = \mathsf{C}^k(\star, \mathsf{I}(\phi(M))) \gg \mathsf{I}(\mathsf{C}^j(\star, \phi(M))) = \phi(L')$$

since $\mathsf{C}^k \succ \mathsf{I}, \mathsf{C}^j$ because $k >_{\mathsf{mul}} j$.

c8 $\langle z[y,w.M] \dagger x.N\rangle \longrightarrow_{\mathsf{c8}} z[y,w.\langle M \dagger x.\mathtt{inv}(w,z.N)\rangle]$.

The derivation

$$\dfrac{\dfrac{\Gamma', y\colon C, w\colon B \vdash M\colon A}{\Gamma', y\colon C, z\colon C{\to}B \vdash z[y,w.M]\colon A}\,L0{\to} \qquad x\colon A, \Gamma', y\colon C, z\colon C{\to}B \vdash N\colon E}{\Gamma', y\colon C, z\colon C{\to}B \vdash \langle z[y,w.M] \dagger x.N\rangle\colon E}\,Cut$$

rewrites to

$$\dfrac{\dfrac{\Gamma', y\colon C, w\colon B \vdash M\colon A \qquad \dfrac{x\colon A, \Gamma', y\colon C, z\colon C{\to}B \vdash N\colon E}{x\colon A, \Gamma', y\colon C, w\colon B \vdash \mathtt{inv}(w,z.N)\colon E}\,Inv}{\Gamma', y\colon C, w\colon B \vdash \langle M \dagger x.\mathtt{inv}(w,z.N)\rangle\colon E}\,Cut}{\Gamma', y\colon C, z\colon C{\to}B \vdash z[y,w.\langle M \dagger x.\mathtt{inv}(w,z.N)\rangle]\colon E}\,L0{\to}$$

Let $k := \{\!\{A, \Gamma', C, C{\to}B, E\}\!\}$ and $j := \{\!\{A, \Gamma', C, B, E\}\!\}$. We have

$$\phi(L) = \mathsf{C}^k(\mathsf{I}(\phi(M)), \phi(N)) \gg \mathsf{I}(\mathsf{C}^j(\phi(M), \mathsf{J}(\phi(N)))) = \phi(L')$$

since $\mathsf{C}^k \succ \mathsf{I}, \mathsf{C}^j, \mathsf{J}$ because $k >_{\mathsf{mul}} j$.

c9 $\langle y[u.v.M, z.M'] \dagger x.N\rangle \longrightarrow_{\mathsf{c9}} y[u.v.M, z.\langle M' \dagger x.\mathtt{inv}(z,y.N)\rangle]$.

The derivation

$$\dfrac{\dfrac{\Gamma', v{:}\, C, u{:}\, D{\to}B \vdash M{:}\, D \quad \Gamma', z{:}\, B \vdash M'{:}\, A}{\Gamma', y{:}\, (C{\to}D){\to}B \vdash y[u.v.M, z.M']{:}\, A}\, L{\to\!\!\to} \qquad x{:}\, A, \Gamma', y{:}\, (C{\to}D){\to}B \vdash N{:}\, E}{\Gamma', y{:}\, (C{\to}D){\to}B \vdash \langle y[u.v.M, z.M'] \dagger x.N\rangle{:}\, E}\, Cut$$

rewrites to

$$\dfrac{\Gamma', v{:}\, C, u{:}\, D{\to}B \vdash M{:}\, D \qquad \dfrac{\Gamma', z{:}\, B \vdash M'{:}\, A \qquad \dfrac{x{:}\, A, \Gamma', y{:}\, (C{\to}D){\to}B \vdash N{:}\, E}{x{:}\, A, \Gamma', z{:}\, B \vdash \mathtt{inv}(z, y.N){:}\, E}\, Inv}{\Gamma', z{:}\, B \vdash \langle M' \dagger x.\mathtt{inv}(z, y.N)\rangle{:}\, E}\, Cut}{\Gamma', y{:}\, (C{\to}D){\to}B \vdash y[u.v.M, z.\langle M' \dagger x.\mathtt{inv}(z, y.N)\rangle]{:}\, E}\, L{\to\!\!\to}$$

Let $k := \{\!\{A, \Gamma', (C{\to}D){\to}B, E\}\!\}$ and $j := \{\!\{A, \Gamma', B, E\}\!\}$. We have

$$\begin{aligned}\phi(L) &= \mathsf{C}^k(\mathsf{K}(\phi(M), \phi(M')), \phi(N))\\ &\gg \mathsf{K}(\phi(M), \mathsf{C}^j(\phi(M'), \mathsf{J}(\phi(N)))) = \phi(L')\end{aligned}$$

since $\mathsf{C}^k \succ \mathsf{K}, \mathsf{C}^j, \mathsf{J}$ because $k >_{\mathsf{mul}} j$.

A $\langle \lambda y.M \dagger x.x[z, w.N]\rangle \longrightarrow_{\mathsf{A}} \langle\langle z \dagger y.M\rangle \dagger w.N\rangle$.

The derivation

$$\dfrac{\dfrac{\Gamma', z{:}\, C, y{:}\, C \vdash M{:}\, B}{\Gamma', z{:}\, C \vdash \lambda y.M{:}\, C{\to}B}\, R{\to} \qquad \dfrac{\Gamma', z{:}\, C, w{:}\, B \vdash N{:}\, E}{\Gamma', z{:}\, C, x{:}\, C{\to}B \vdash x[z, w.N]{:}\, E}\, L0{\to}}{\Gamma', z{:}\, C \vdash \langle \lambda y.M \dagger x.x[z, w.N]\rangle{:}\, E}\, Cut$$

rewrites to

$$\dfrac{\dfrac{}{\Gamma', z{:}\, C \vdash z{:}\, C}\, Ax \qquad \Gamma', z{:}\, C, y{:}\, C \vdash M{:}\, B}{\Gamma', z{:}\, C \vdash \langle z \dagger y.M\rangle{:}\, B}\, Cut \qquad \Gamma', z{:}\, C, w{:}\, B \vdash N{:}\, E}{\Gamma', z{:}\, C \vdash \langle\langle z \dagger y.M\rangle \dagger w.N\rangle{:}\, E}\, Cut$$

Let $k := \{\!\{\Gamma', C, C{\to}B, E\}\!\}$ and $j := \{\!\{\Gamma', C, B, E\}\!\}$ and $i := \{\!\{\Gamma', C, C, B\}\!\}$. We have

$$\phi(L) = \mathsf{C}^k(\mathsf{I}(\phi(M)), \mathsf{I}(\phi(N))) \gg \mathsf{C}^j(\mathsf{C}^i(\star, \phi(M)), \phi(N)) = \phi(L')$$

since $\mathsf{C}^k \succ \star, \mathsf{J}, \mathsf{C}^j, \mathsf{C}^i$ because $k >_{\mathsf{mul}} j, i$.

B $\langle \lambda y.M \dagger x.x[u.v.N, z.N']\rangle \longrightarrow_{\mathsf{B}} \langle\langle \lambda u.\langle \lambda y'.\mathtt{inv}(y', y.M) \dagger v.N\rangle \dagger y.M\rangle \dagger z.N'\rangle$.

The derivation

$$\dfrac{\dfrac{\Gamma, y{:}\, C{\to}D \vdash M{:}\, B}{\Gamma \vdash \lambda y.M{:}\, (C{\to}D){\to}B} \qquad \dfrac{u{:}\, C, v{:}\, D{\to}B, \Gamma \vdash N{:}\, D \quad z{:}\, B, \Gamma \vdash N'{:}\, E}{x{:}\, (C{\to}D){\to}B, \Gamma \vdash x[u.v.N, z.N']{:}\, E}\, L{\to\to}}{\Gamma \vdash \langle \lambda y.M \dagger x.x[u.v.N, z.N'] \rangle{:}\, E}\, Cut$$

rewrites to

$$\dfrac{\dfrac{\dfrac{\mathcal{D}}{\Gamma \vdash M'{:}\, C{\to}D} \qquad \Gamma, y{:}\, C{\to}D \vdash M{:}\, B}{\Gamma \vdash \langle M' \dagger y.M \rangle{:}\, B}\, Cut \qquad z{:}\, B, \Gamma \vdash N'{:}\, E}{\Gamma \vdash \langle \langle M' \dagger y.M \rangle \dagger z.N' \rangle{:}\, E}\, Cut$$

where $M' = \lambda u.\langle \lambda y'.\mathtt{inv}(y', y.M) \dagger v.N \rangle$ and $\mathcal{D}$ is the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, y{:}\, C{\to}D \vdash M{:}\, B}{\Gamma, u{:}\, C, y{:}\, C{\to}D \vdash M{:}\, B}\, (\mathsf{weak})}{\Gamma, u{:}\, C, y'{:}\, D \vdash \mathtt{inv}(y', y.M){:}\, B}\, Inv}{\Gamma, u{:}\, C \vdash \lambda y'.\mathtt{inv}(y', y.M){:}\, D{\to}B}\, R{\to} \qquad u{:}\, C, v{:}\, D{\to}B, \Gamma \vdash N{:}\, D}{\Gamma, u{:}\, C \vdash \langle \lambda y'.\mathtt{inv}(y', y.M) \dagger v.N \rangle{:}\, D}\, Cut}{\Gamma \vdash \lambda u.\langle \lambda y'.\mathtt{inv}(y', y.M) \dagger v.N \rangle{:}\, C{\to}D}\, R{\to}$$

Let $k := \{\!\{(C{\to}D){\to}B, \Gamma, E\}\!\}$ and $j := \{\!\{B, \Gamma, E\}\!\}$ and $i := \{\!\{\Gamma, C{\to}D, B\}\!\}$ and $h := \{\!\{C, D{\to}B, \Gamma, D\}\!\}$. We have

$$\begin{aligned}\phi(L) = \;& \mathsf{C}^k(\mathsf{I}(\phi(M)), \mathsf{K}(\phi(N), \phi(N'))) \\ & \gg\!\mathsf{C}^j(\mathsf{C}^i(\mathsf{I}(\mathsf{C}^h(\mathsf{I}(\mathsf{J}(\phi(M))), \phi(N))), \phi(M)), \phi(N')) = \phi(L')\end{aligned}$$

since $\mathsf{C}^k \succ \mathsf{I}, \mathsf{J}, \mathsf{C}^j, \mathsf{C}^i, \mathsf{C}^h$ because $k >_{\mathsf{mul}} j, i, h$.

E $\langle y \dagger x.x[z, w.N] \rangle \longrightarrow_{\mathsf{E}} y[z, w'.\, \langle w' \dagger w.\mathtt{inv}(w', y.N) \rangle]$.

The derivation

$$\dfrac{\dfrac{}{\Gamma', z{:}\, C, y{:}\, C{\to}B \vdash y{:}\, C{\to}B}\, Ax \qquad \dfrac{\Gamma', z{:}\, C, y{:}\, C{\to}B, w{:}\, B \vdash N{:}\, E}{\Gamma', z{:}\, C, y{:}\, C{\to}B, x{:}\, C{\to}B \vdash x[z, w.N]{:}\, E}\, L0{\to}}{\Gamma', z{:}\, C, y{:}\, C{\to}B \vdash \langle y \dagger x.x[z, w.N] \rangle{:}\, E}\, Cut$$

rewrites to

$$\dfrac{\dfrac{\dfrac{}{\Gamma', z{:}\, C, w'{:}\, B \vdash w'{:}\, B}\, Ax \qquad \dfrac{\Gamma', z{:}\, C, y{:}\, C{\to}B, w{:}\, B \vdash N{:}\, E}{\Gamma', z{:}\, C, w'{:}\, B, w{:}\, B \vdash \mathtt{inv}(w', y.N){:}\, E}\, Inv}{\Gamma', z{:}\, C, w'{:}\, B \vdash \langle w' \dagger w.\mathtt{inv}(w', y.N) \rangle{:}\, E}\, Cut}{\Gamma', z{:}\, C, y{:}\, C{\to}B \vdash y[z, w'.\, \langle w' \dagger w.\mathtt{inv}(w', y.N) \rangle]{:}\, E}\, L0{\to}$$

Let $k := \{\!\{\Gamma', C, C{\to}B, C{\to}B, E\}\!\}$ and $j := \{\!\{\Gamma', C, B, B, E\}\!\}$. We have

$$\phi(L) = \mathsf{C}^k(\star, \mathsf{I}(\phi(N))) \gg \mathsf{I}(\mathsf{C}^j(\star, \mathsf{J}(\phi(N)))) = \phi(L')$$

since $\mathsf{C}^k \succ \star, \mathsf{J}, \mathsf{C}^j$ because $k >_{\mathsf{mul}} j$.

F & G $\quad \langle y \dagger x.x[u.v.N', w.N]\rangle$
$\longrightarrow_{\mathsf{F/G}}\ y[u'.v'.\langle u' \dagger u.P\rangle, w'.\langle w' \dagger w.\mathtt{inv}(w', y.N)\rangle]$
with $P = \mathtt{dec}(u', v', y.\langle \lambda y''.y[u''.v''.y'', z.z] \dagger v.N'\rangle)$ for F
and $P = \langle v' \dagger v.\mathtt{dec}(u', v', y.N')\rangle$ for G.

Γ is of the form $\Gamma', y{:}\,(C{\to}D){\to}B$. The derivation

$$\dfrac{\Gamma \vdash y{:}\,(C{\to}D){\to}B \qquad \dfrac{\Gamma, u{:}\,C, v{:}\,D{\to}B \vdash N'{:}\,D \quad \Gamma, w{:}\,B \vdash N{:}\,E}{\Gamma, x{:}\,(C{\to}D){\to}B \vdash x[u.v.N', w.N]{:}\,E}\,L{\to\!\to}}{\Gamma \vdash \langle y \dagger x.x[u.v.N', w.N]\rangle{:}\,E}\,Cut$$

rewrites to

$$\dfrac{\dfrac{}{\Gamma', u'{:}\,C, v'{:}\,D{\to}B \vdash u'{:}\,C}\,Ax \quad \dfrac{\mathcal{D}'}{\Gamma', u{:}\,C, u'{:}\,C, v'{:}\,D{\to}B \vdash P'{:}\,D}}{\dfrac{\Gamma', u'{:}\,C, v'{:}\,D{\to}B \vdash \langle u' \dagger u.P'\rangle{:}\,D \qquad \dfrac{\mathcal{D}}{\Gamma', w'{:}\,B \vdash M{:}\,E}}{\Gamma', y{:}\,(C{\to}D){\to}B \vdash y[u'.v'.\langle u' \dagger u.P'\rangle, w'.L']{:}\,E}}$$

where the last rule is $L{\to\!\to}$, $M = \langle w' \dagger w.\mathtt{inv}(w', y.N)\rangle$, and $\mathcal{D}$ is the following derivation:

$$\dfrac{\Gamma', w'{:}\,B \vdash w'{:}\,B \qquad \dfrac{\Gamma', y{:}\,(C{\to}D){\to}B, w{:}\,B \vdash N{:}\,E}{\Gamma', w'{:}\,B, w{:}\,B \vdash \mathtt{inv}(w', y.N){:}\,E}\,Inv}{\Gamma', w'{:}\,B \vdash \langle w' \dagger w.\mathtt{inv}(w', y.N)\rangle{:}\,E}\,Cut$$

In the case of F, $\mathcal{D}'$ is the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma'', u''{:}\,C, v''{:}\,C{\to}B \vdash y''{:}\,D}\,Ax \quad \dfrac{}{\Gamma'', z{:}\,B \vdash z{:}\,B}\,Ax}{\Gamma'' \vdash y[u''.v''.y'', z.z]{:}\,B}\,L{\to\!\to}}{\Gamma, u{:}\,C \vdash \lambda y''.y[u''.v''.y'', z.z]{:}\,D{\to}B}\,R{\to} \qquad \Gamma, u{:}\,C, v{:}\,D{\to}B \vdash N'{:}\,D}{\Gamma, u{:}\,C \vdash \langle \lambda y''.y[u''.v''.y'', z.z] \dagger v.N'\rangle{:}\,D}\,Cut}{\Gamma', u{:}\,C, u'{:}\,C, v'{:}\,D{\to}B \vdash \mathtt{dec}(u', v', y.\langle \lambda y''.y[u''.v''.y'', z.z] \dagger v.N'\rangle){:}\,D}\,Dec$$

where $\Gamma'' := \Gamma, u{:}\,C, y''{:}\,D$.

Let $k := \{\!\{\Gamma, (C\to D)\to B, (C\to D)\to B, E\}\!\}$ and $i := \{\!\{\Gamma', B, B, E\}\!\}$ and $j := \{\!\{\Gamma', C, C, D\to B, D\}\!\}$ and $h := \{\!\{\Gamma, C, D\}\!\}$ and $l := \{\!\{\Gamma, C, D\to B, D\}\!\}$. We have

$$\begin{aligned}\phi(L) &= \mathsf{C}^k(\star, \mathsf{K}(\phi(N'), \phi(N)))\\ &\gg \mathsf{K}(\mathsf{C}^j(\star, \mathsf{D}^h(\mathsf{C}^l(\mathsf{I}(\mathsf{K}(\star,\star)), \phi(N')))), \mathsf{C}^i(\star, \mathsf{J}(\phi(N)))) = \phi(L')\end{aligned}$$

since $k >_{\mathsf{mul}} i, j, h, l$.

In the case of G, $\mathcal{D}'$ is the following derivation:

$$\dfrac{\Gamma'' \vdash v' : D\to B \qquad \dfrac{\Gamma, u : C, v : D\to B \vdash N' : D}{\Gamma'', v : D\to B \vdash \mathtt{dec}(u', v', y.N') : D}\, Dec}{\Gamma'' \vdash \langle v' \dagger v.\mathtt{dec}(u', v', y.N')\rangle : D}\, Cut$$

where $\Gamma'' := \Gamma', u : C, u' : C, v' : D\to B$.

Let $k := \{\!\{\Gamma, (C\to D)\to B, (C\to D)\to B, E\}\!\}$ and $i := \{\!\{\Gamma', B, B, E\}\!\}$ and $j := \{\!\{\Gamma', C, C, D\to B, D\}\!\}$ and $h := \{\!\{\Gamma', C, D\to B, C, D\to B, D\}\!\}$ and $l := \{\!\{\Gamma', (C\to D)\to B, C, D\to B, E\}\!\}$. We have

$$\phi(L) \gg \mathsf{K}(\mathsf{C}^j(\star, \mathsf{C}^h(\star, \mathsf{D}^l(\phi(N')))), \mathsf{C}^i(\star, \mathsf{J}(\phi(N)))) = \phi(L')$$

since $k >_{\mathsf{mul}} i, j, h, l$.

□

Corollary 198 (Strong Normalisation) *Systems* rs *and* ars *are strongly normalising on typed terms.*

Proof: This is a consequence of Theorem 197 and Remark 196. □

Corollary 199 *Rules Inv, Of, Dec, and Cut are term-irrelevantly admissible in the system of Definition 111.*

Proof: Every term with an auxiliary constructor is reducible by system rs and ars. □

7.5 Variants of reduction systems

We investigate in this section some variants of the cut-elimination system presented in section 7.3. We discuss in section 7.5.1 the difference between system cers and system cears. In section 7.5.2, we then discuss the critical pairs in these systems, which are usual between the rules of Kind_1 and those of Kind_2. As in Chapter 6 for λG3 we present the CBN and CBV restrictions that make systems rs and ars orthogonal.

7.5.1 η-expansion & the alternative in the cut-reduction system

The two variants cers and system cears come from whether we want variables to behave like their η-expansions or we want the elimination of a cut with a variable to be simpler and closer to renaming.

The behaviour of functionals is interesting in G4ii, because it is a depth-bounded calculus: as presented in section 7.1, among all Church's numerals only 0 and 1 can be represented in the t-restriction of G3ii′, and hence in G4ii as well (which is even more restrictive). So when reducing the term that represents (using cuts) "$1+1$", we should expect some semantical anomaly in the reductions (similar to the one reported by Vestergaard in [Ves99]). Such an anomaly is to be found for instance in rules B and D, and for abstractions we have no alternative choice. In system rs, we can prove that variables have have the same functional behaviour as their η-expansions:

Theorem 200 (η-expansion of variables)
Consider the η-expansion of a variable $y \longrightarrow_\eta \lambda y'.y[y', w'.w']$.

$$\langle \lambda y'.y[y', w'.w'] \dagger x.x[z, w.N] \rangle \longrightarrow^*_{\mathsf{rs}} y[z, w'.\, \langle w' \dagger w.\mathtt{inv}(w', y.N) \rangle]$$

and

$$\langle \lambda y'.y[y', w'.w'] \dagger x.x[u.v.N', w.N] \rangle \longrightarrow^*_{\mathsf{rs}} y[u'.v'.\, \langle u' \dagger u.P \rangle, w'.\, \langle w' \dagger w.\mathtt{inv}(w', y.N) \rangle]$$

where $P := \mathtt{dec}(u', v', y.\, \langle \lambda y''.y[u.v.y'', z.z] \dagger v.N' \rangle)$

Proof:

$$\begin{array}{lll}
\text{(Rule E)} & & \langle \lambda y'.y[y', w'.w'] \dagger x.x[z, w.N] \rangle \\
& \longrightarrow_{\mathsf{C}} & \langle \langle z \dagger y'.y[y', w'.w'] \rangle \dagger w.N \rangle \\
& \longrightarrow_{\mathsf{c7}} & \langle y[z, w'.\, \langle z \dagger y'.w' \rangle] \dagger w.N \rangle \\
& \longrightarrow_{\mathsf{c2}} & \langle y[z, w'.w'] \dagger w.N \rangle \\
& \longrightarrow_{\mathsf{c8}} & y[z, w'.\, \langle w' \dagger w.\mathtt{inv}(w', y.N) \rangle]
\end{array}$$

$$\begin{array}{lll}
\text{(Rule F)} & & \langle \lambda y'.y[y', w'.w'] \dagger x.x[u.v.N', w.N] \rangle \\
& \longrightarrow_{\mathsf{D}} & \langle \langle \lambda u.\langle L \dagger v.N' \rangle \dagger y'.y[y', w'.w'] \rangle \dagger w.N \rangle \\
& \longrightarrow^*_{\mathsf{rs}} & \langle \langle \lambda u.\langle L' \dagger v.N' \rangle \dagger y'.y[y', w'.w'] \rangle \dagger w.N \rangle \\
& \longrightarrow^*_{\mathsf{rs}} & \langle y[u'.v'.\, \langle u' \dagger u.P \rangle, w'.w'] \dagger w.N \rangle \\
& \longrightarrow_{\mathsf{c9}} & y[u'.v'.\, \langle u' \dagger u.P \rangle, w'.\, \langle w' \dagger w.\mathtt{inv}(w', y.N) \rangle]
\end{array}$$

where the first $\longrightarrow^*_{\mathsf{rs}}$ is justified by

$$\begin{array}{lll}
L & := & \lambda y''.\mathtt{inv}(y'', w'.y[w', z.z]) \\
& \longrightarrow_{\mathsf{i6}} & \lambda y''.y[y_1.y_2.y'', z.\mathtt{inv}(y'', w'.z)] \\
& \longrightarrow_{\mathsf{i1}} & \lambda y''.y[y_1.y_2.y'', z.z] \qquad =: \quad L'
\end{array}$$

and the last $\longrightarrow^*_{\mathsf{rs}}$ is justified by

$$\begin{array}{ll} & \langle \lambda u.\langle L' \dagger v.N' \rangle \dagger y'.y[y', w'.w'] \rangle \\ \longrightarrow_{\mathsf{c6,c2}} & y[u'.v'.\, \langle u' \dagger u.\mathtt{dec}(u', v', y.\, \langle L' \dagger v.N' \rangle) \rangle\, , w'.w'] \\ = & y[u'.v'.\, \langle u' \dagger u.P \rangle\, , w'.w'] \end{array}$$ □

So rules E and F make variables have the same functional behaviour as their η-expansion, hence rule F inherits the anomaly of rule D.

But instead of forcing variables to inherit the anomaly of abstractions, we might rather follow the intuition that cutting a variable with another variable is almost renaming, and rather chose G, simpler, instead of F. This new rule is more natural than rule F; however the reducts are semantically different and thus the choice of rule G breaks the property that a variable and its η-expansion have the same behaviour.

Note that [DKL06] introduces a cut-elimination system which integrates η-expansion to its rules, with the use of an extra constructor `of` that can be eliminated. Since this system deals with abstractions and variables uniformly, it thus leads to cers, as explained in [DKL06] by the use of a theorem with the same substance as Theorem 200.

7.5.2 Orthogonal systems

In this section we suggest two ways of restricting the rules of Kind_1 and Kind_2 to make systems rs and ars orthogonal, and hence confluent.

In the restricted systems gs and ars there are overlaps between the right and left propagation sub-systems, i.e. there is a critical pair between any rule in $\{\mathsf{c1}, \mathsf{c2}, \mathsf{c3}, \mathsf{c4}, \mathsf{c5}\}$ and any rule in $\{\mathsf{c8}, \mathsf{c9}\}$. This is shown in Fig. 7.7, where column headers represent the different cases concerning the first premiss of the cut, while row headers represent the different cases for the second one (marking inside parentheses the status of the cut-type).

	Axiom	$R\rightarrow$	$L0\rightarrow$	$L\rightarrow\!\!\rightarrow$
Axiom (Principal)	c1	c1	c1, c8	c1, c9
Axiom (Non-Principal)	c2	c2	c2, c8	c2, c9
$R\rightarrow$	c3	c3	c3, c8	c3, c9
$L0\rightarrow$(Non-Principal, Non-Auxiliary)	c4	c4	c4, c8	c4, c9
$L\rightarrow\!\!\rightarrow$(Non-Principal)	c5	c5	c5, c8	c5, c9
$L0\rightarrow$(Non-Principal, Auxiliary)	c7	c6	c8	c9
$L0\rightarrow$(Principal)	E	C	c8	c9
$L\rightarrow\!\!\rightarrow$(Principal)	F (rs) or G (ars)	D	c8	c9

Figure 7.7: Overlaps of reduction rules

The overlaps pointed out in Fig. 7.7 are well-known in sequent calculus, and correspond to the choice of whether to push a cut into the proof of its left premiss

or into the proof of its right premiss. The former corresponds to a *call-by-value* strategy and the latter corresponds to a *call-by-name* strategy, as described in Chapter 6.

Since the overlaps only concerns cut-reduction rules of Kind_1 and Kind_2, we discuss in the following possible ways to make them non-overlapping.

Call-by-name

One way to make the system orthogonal is to give preference to rules c1-c2-c3-c4-c5 over rules c8-c9, thus restricted to the case when N is an x-covalue Q, i.e. is of the form $x[y, w.N]$ or $x[u.v.M, w.N]$.

Note that in order to reduce a term like $\langle M \dagger x.y[x, w.N]\rangle$, there is no choice other than left-propagation (rules c8 and c9) until a similar redex is found in which M is a value, and then only rules c6 or c7 can be applied.

Call-by-value

Alternatively, preference might be given to rules c8 and c9, which we can formalise as restricting rules c1-c2-c3-c4-c5 to the case when M is a value V (variable or abstraction).

The choice of call-by-value is more natural than that of call-by-name because the two rules of right-propagation c6 and c7 only apply to cuts whose first argument is a value. This suggests that G4ii has an inherent *call-by-value* flavour, echoing the idea that it is somehow based on the call-by-value sequent calculus LJQ. Indeed, completeness of LJQ gives a short proof of the completeness of G4ii [DL06].

We finish this section by stating the following property of cbn and cbv.

Theorem 201 *Reduction systems CBN and CBV are confluent.*

Proof: Systems CBN and CBV can be seen as particular orthogonal CRS, so they satisfy confluence (see [vOvR94] for details). □

Conclusion

This chapter defines various term calculi for the depth-bounded intuitionistic sequent calculus of Hudelmaier. Using standard techniques of rewriting, we prove subject reduction and strong normalisation for all of them, so *Cut*-admissibility turns out to be a corollary. The cbn and cbv systems presented in this chapter are also orthogonal, which guarantees confluence (and uniqueness of normal forms).

Some relations between G4ii and other calculi for intuitionistic logic are studied in [DL06]. Also, from our term calculi for G4ii, which use explicit operators, we could extract term calculi with *implicit* operators (as in λ-calculus). This would bring our calculus closer to that of Matthes [Mat02], and with a strong normalising cut-elimination procedure. As mentioned in the introduction, defining a denotational semantics for our calculi as well as investigating the connections with the simply-typed λ-calculus would reveal more properties of the proofs in G4ii. This is left for further investigations.

Part II

Type Theory in Sequent Calculus

Chapter 8

Pure Type Sequent Calculi (PTSC)

In this chapter, whose contents appeared in [LDM06], we apply to the framework of *Pure Type Systems* [Bar92] the insight into the relationship between sequent calculus and natural deduction developed in the first part of this dissertation and in previous work such as [Her94, Her95, DP99b, PD98, DU03].

In sequent calculus the proof-search space is often the cut-free fragment, since the latter usually satisfies the sub-formula property. The sequent calculus LJT [Her95], has the extra advantage of being closer to natural deduction (a reflection is established in Chapter 6), and it makes proof-search more deterministic than a Gentzen-style sequent calculus. This makes LJT a natural formalism to organise proof-search in intuitionistic logic [DP99a], and, its derivations being close to the notion of uniform proofs, LJT can be used to describe proof-search in pure Prolog and some of its extensions [MNPS91]. The corresponding term assignment system also expresses the intimate details of β-normalisation in λ-calculus in a form closer to abstract (stack-based) machines for reduction (such as Krivine's [Kri]).

The framework of *Pure Type Systems* (PTS) [Bar92] exploits and generalises the Curry-Howard correspondence, and accounts for many systems already existing, starting with *Barendregt's Cube*. Proof assistants based on them, such as the Coq system [Coq] or the Lego system [LP92], feature interactive proof construction methods using proof-search tactics. Primitive tactics display an asymmetry between introduction rules and elimination rules of the underlying natural deduction calculus: the tactic Intro corresponds to the right-introduction rule for the Π-type (whether in natural deduction or in sequent calculus), but the tactics Apply in Coq or Refine in Lego are much closer (in spirit) to the left-introduction of Π-types (as in sequent calculus) than to elimination rules of natural deduction [McK97].

Although encodings from natural deduction to sequent calculus and vice-versa have been widely studied [Gen35, Pra65, Zuc74], the representation in sequent calculus of type theories is relatively undeveloped compared to the literature about type theory in natural deduction. An interesting approach to PTS using

sequent calculus is in [GR03b]. Nevertheless, only the typing rules are in a sequent calculus style, whereas the syntax is still in a natural deduction style: in particular, proofs are denoted by λ-terms, the structure of which no longer matches the structure of proofs.

However, proofs in sequent calculus *can* be reflected by specific proof-terms; for instance, a construction $M \cdot l$, representing a list of terms with head M and tail l, is introduced in [Her94, Her95] to denote the left-introduction of implication (in the sequent calculus LJT):

$$\frac{\Gamma \vdash M : A \quad \Gamma; B \vdash l : C}{\Gamma; A \to B \vdash M \cdot l : C}$$

This approach is extended to the corner of the Cube with dependent types and type constructors in [PD98], but types are still built with λ-terms, so the system extensively uses conversion functions from sequent calculus to natural deduction and back.

With such term assignment systems, cut-elimination can be done by means of a rewrite system, cut-free proofs being thus denoted by terms in normal form. In type theory, not only is the notion of proof-normalisation/cut-elimination interesting on its own, but it is even necessary to define the notion of typability, as soon as types depend on terms.

In this chapter we enrich the sequent calculus LJT into a collection of systems called *Pure Type Sequent Calculi* (PTSC), capturing the traditional PTS, with the hope to improve the understanding of implementation of proof systems based on PTS in respect of:

- having a direct analysis of the basic proof-search tactics, which could then be moved into the kernel, rather than requiring a separate type-checking layer for correctness,
- opening the way to improve the basic system with an approach closer to abstract machines to express reductions, both in type-checking *and* in execution (of extracted programs),
- studying extensions to systems involving inductive types/families (such as the Calculus of Inductive Constructions).

In fact, the idea of using LJT to describe basic proof-search tactics in Lego was earlier raised in [McK97]. Here we formalise and develop this approach.

Inspired by the fact that, in type theory, implication and universal quantification are just a dependent product, we modify the inference rule above to obtain the left-introduction rule for a Π-type in a PTSC:

$$\frac{\Gamma \vdash M : A \quad \Gamma; \langle M/x \rangle B \vdash l : C}{\Gamma; \Pi x^A . B \vdash M \cdot l : C}\ \Pi\mathsf{l}$$

We use here explicit substitutions, whose natural typing rule are cuts [BR95]. From our system a version with implicit substitutions can be derived (see Chapter 9), but this does not allow cuts on an arbitrary formula of a typing environment Γ. Also, explicit substitutions allow the definition of a normalisation procedure by local (small-step) rewrite rules in the spirit of Gentzen's cut-elimination.

We establish the logical equivalence between a PTSC and its corresponding PTS by means of type-preserving encodings. We also prove that the former is strongly normalising if and only if the latter is. The proof is based on mutual encodings that allow the normalisation procedure of one formalism to be simulated by that of the other. Part of the proof also uses a technique by Bloo and Geuvers [BG99], introduced to prove strong normalisation properties of the explicit substitution calculus $\lambda\mathsf{x}$ and later used in [DU03] for $\overline{\lambda}$ [Her95].

Section 8.1 presents the syntax of a PTSC and gives the rewrite rules for normalisation. Section 8.3 gives the typing system with the parameters specifying the PTSC, and a few properties are stated such as subject reduction. Section 8.4 establishes the correspondence between a PTSC and its corresponding PTS, from which we derive confluence. Section 8.5 presents the strong normalisation result.

8.1 Syntax & reduction

Definition 114 (Grammar of PTSC) The syntax of a PTSC depends on a given set $\mathcal{S}$ of *sorts*, written $s, s', \ldots$, and a denumerable set $\mathcal{X}$ of variables, written $x, y, z, \ldots$.

The set $\mathcal{T}$ of *terms* (denoted $M, N, P, \ldots$) and the set $\mathcal{L}$ of *lists* (denoted $l, l', \ldots$) are inductively defined as

$$\begin{array}{rcl} M, N, A, B & ::= & \Pi x^A.B \mid \lambda x^A.M \mid s \mid x\ l \mid M\ l \mid \langle M/x\rangle N \\ l, l' & ::= & [] \mid M \cdot l \mid l@l' \mid \langle M/x\rangle l \end{array}$$

$\Pi x^A.M$, $\lambda x^A.M$, and $\langle N/x\rangle M$ bind x in M, and $\langle M/x\rangle l$ binds x in l, thus defining the free variables of terms and lists as well as α-conversion. The set of free variables of a term M (resp. a list l) is denoted $\mathsf{FV}(M)$ (resp. $\mathsf{FV}(l)$). $\Pi x^A.M$ is called a Π-*type*. Let $A \to B$ denote $\Pi x^A.B$ when $x \notin \mathsf{FV}(B)$.

This syntax is an extension of Herbelin's $\overline{\lambda}$ [Her95] (with type annotations on λ-abstractions) presented in Chapter 6. An intuitive understanding of $\overline{\lambda}$ in terms of functions, arguments, abstract machines is presented therein. Note that the list with head M and tail l, denoted $M \cdot l$, now has a typing rule corresponding to the left-introduction of Π-types (cf. Section 8.3). Explicit substitutions will here be used in two ways: first, to instantiate a universally quantified variable, and second, to describe explicitly the interaction between the constructors in the normalisation process, shown in Fig. 8.1. Side-conditions to avoid variable capture and liberation can be inferred by the process described in Definition 42.

Confluence of the system is proved in section 8.4. More intuition about $\overline{\lambda}$, its syntax and operational semantics is also given in [Her95].

$$
\begin{array}{llll}
\mathsf{B} & (\lambda x^A.M)\ (N \cdot l) & \longrightarrow & (\langle N/x\rangle M)\ l \\
\multicolumn{4}{l}{\mathsf{x}\ \{} \\
\mathsf{B1} & M\ [] & \longrightarrow & M \\
\mathsf{B2} & (x\ l)\ l' & \longrightarrow & x\ (l@l') \\
\mathsf{B3} & (M\ l)\ l' & \longrightarrow & M\ (l@l') \\
\mathsf{A1} & (M \cdot l')@l & \longrightarrow & M \cdot (l'@l) \\
\mathsf{A2} & []@l & \longrightarrow & l \\
\mathsf{A3} & (l@l')@l'' & \longrightarrow & l@(l'@l'') \\
\multicolumn{4}{l}{\mathsf{xsubst:}\ \{} \\
\mathsf{C1} & \langle P/y\rangle \lambda x^A.M & \longrightarrow & \lambda x^{\langle P/y\rangle A}.\langle P/y\rangle M \\
\mathsf{C2} & \langle P/y\rangle (y\ l) & \longrightarrow & P\ \langle P/y\rangle l \\
\mathsf{C3} & \langle P/y\rangle (x\ l) & \longrightarrow & x\ \langle P/y\rangle l \quad \text{if } x \neq y \\
\mathsf{C4} & \langle P/y\rangle (M\ l) & \longrightarrow & \langle P/y\rangle M\ \langle P/y\rangle l \\
\mathsf{C5} & \langle P/y\rangle \Pi x^A.B & \longrightarrow & \Pi x^{\langle P/y\rangle A}.\langle P/y\rangle B \\
\mathsf{C6} & \langle P/y\rangle s & \longrightarrow & s \\
\mathsf{D1} & \langle P/y\rangle [] & \longrightarrow & [] \\
\mathsf{D2} & \langle P/y\rangle (M \cdot l) & \longrightarrow & (\langle P/y\rangle M) \cdot (\langle P/y\rangle l) \\
\mathsf{D3} & \langle P/y\rangle (l@l') & \longrightarrow & (\langle P/y\rangle l)@(\langle P/y\rangle l')
\end{array}
$$

Figure 8.1: Reduction Rules

Definition 115 (Convertibility) We say that two terms M and N are *convertible* if $M \longleftrightarrow^*_{\mathsf{Bx}} N$.

We now show that system x is terminating. If we add rule B, then the system fails to be terminating unless we only consider terms that are typed in a particular typing system.

Definition 116 (First-order encoding) We consider the following first-order signature and its (terminating) precedence relation:

$$\mathsf{sub}(\_,\_) \succ \mathsf{cut}(\_,\_) \succ \mathsf{ii}(\_,\_) \succ \mathsf{i}(\_) \succ \star$$

The LPO induced on the first-order terms is also terminating. The encoding is given in Fig. 8.2.

Theorem 202 *If $M \longrightarrow_{\mathsf{x}} M'$ then $\overline{M} \gg \overline{M'}$, and if $l \longrightarrow_{\mathsf{x}} l'$ then $\overline{l} \gg \overline{l'}$.*

$\overline{s}$	$:=$	$\star$
$\overline{\lambda x^A.M}$	$:=$	$\mathsf{ii}(\overline{A},\overline{M})$
$\overline{\Pi x^A.M}$	$:=$	$\mathsf{ii}(\overline{A},\overline{M})$
$\overline{x\ l}$	$:=$	$\mathsf{i}(\overline{l})$
$\overline{M\ l}$	$:=$	$\mathsf{cut}(\overline{M},\overline{l})$
$\overline{\langle M/x\rangle N}$	$:=$	$\mathsf{sub}(\overline{M},\overline{N})$
$\overline{[]}$	$:=$	$\star$
$\overline{M\cdot l}$	$:=$	$\mathsf{ii}(\overline{M},\overline{l})$
$\overline{l@l'}$	$:=$	$\mathsf{ii}(\overline{l},\overline{l'})$
$\overline{\langle M/x\rangle l}$	$:=$	$\mathsf{sub}(\overline{M},\overline{l})$

Figure 8.2: Encoding to the first-order syntax

Proof: By induction on M, l. The case analysis for root reduction gives:

B1	$\mathsf{cut}(M,\star)$	$\gg M$
B2	$\mathsf{cut}(\mathsf{i}(l),l')$	$\gg \mathsf{i}(\mathsf{ii}(l,l'))$
B3	$\mathsf{cut}(\mathsf{cut}(M,l),l')$	$\gg \mathsf{cut}(M,\mathsf{ii}(l,l'))$
A1	$\mathsf{ii}(\mathsf{ii}(M,l'),l)$	$\gg \mathsf{ii}(M,\mathsf{ii}(l',l))$
A2	$\mathsf{ii}(\star,l)$	$\gg l$
A3	$\mathsf{ii}(\mathsf{ii}(l,l'),l'')$	$\gg \mathsf{ii}(l,\mathsf{ii}(l',l''))$
C1	$\mathsf{sub}(P,\mathsf{ii}(A,M))$	$\gg \mathsf{ii}(\mathsf{sub}(P,A),\mathsf{sub}(P,M))$
C2	$\mathsf{sub}(P,\mathsf{i}(l))$	$\gg \mathsf{cut}(P,\mathsf{sub}(P,l))$
C3	$\mathsf{sub}(P,\mathsf{i}(l))$	$\gg \mathsf{i}(\mathsf{sub}(P,l))$
C4	$\mathsf{sub}(P,\mathsf{cut}(M,l))$	$\gg \mathsf{cut}(\mathsf{sub}(P,M),\mathsf{sub}(P,l))$
C5	$\mathsf{sub}(P,\mathsf{ii}(A,B))$	$\gg \mathsf{ii}(\mathsf{sub}(P,A),\mathsf{sub}(P,B))$
C6	$\mathsf{sub}(P,s)$	$\gg \star$
D1	$\mathsf{sub}(P,\star)$	$\gg \star$
D2	$\mathsf{sub}(P,(\mathsf{ii}(M,l)))$	$\gg \mathsf{ii}((\mathsf{sub}(P,M)),(\mathsf{sub}(P,l)))$
D3	$\mathsf{sub}(P,\mathsf{ii}(l,l'))$	$\gg \mathsf{ii}(\mathsf{sub}(P,l),\mathsf{sub}(P,l'))$

□

Corollary 203 *System x is terminating (on all terms and lists).*

8.2 A reflection of Pure Type Systems

In this section we establish a reflection in PTSC of *Pure Type Systems* [Bar92].

Section 8.4 will establish a logical correspondence between a PTSC and a PTS, in that the following encodings, which form the reflection, preserve a notion of typing that is given in the next sections.

We briefly recall the syntax and operational semantics of a PTS. See e.g. [Bar92] for more detail.

Definition 117 (Syntax & reduction of a PTS) The terms have the following syntax:

$$t, u, v, T, U, V, \ldots ::= x \mid \ s \mid \ \Pi x^T.t \mid \ \lambda x^T.t \mid \ t\ u$$

where x ranges over variables and s over sorts (as in PTSC).

The calculus is equipped with the β-reduction rule $(\lambda x^U.t)\ u \longrightarrow_\beta \ \{{}^{u}\!/_{x}\}t$, which is confluent.

$\mathcal{A}(s)$	$:=$	s	
$\mathcal{A}(\Pi x^T.U)$	$:=$	$\Pi x^{\mathcal{A}(T)}.\mathcal{A}(U)$	
$\mathcal{A}(\lambda x^T.t)$	$:=$	$\lambda x^{\mathcal{A}(T)}.\mathcal{A}(t)$	
$\mathcal{A}(t)$	$:=$	$\mathcal{A}_{[]}(t)$	otherwise
$\mathcal{A}_l(t\ u)$	$:=$	$\mathcal{A}_{\mathcal{A}(u)\cdot l}(t)$	
$\mathcal{A}_l(x)$	$:=$	$x\ l$	
$\mathcal{A}_l(t)$	$:=$	$\mathcal{A}(t)\ l$	otherwise

Figure 8.3: From a PTS to a PTSC

The translation from a PTS to a PTSC is given in Fig. 8.3. It is simply the adaptation to the higher-order case of Prawitz's translation from natural deduction to sequent calculus from Chapter 2: the encoding of an application relies on a parameterised version of the translation.

$\mathcal{B}(\Pi x^A.B)$	$:=$	$\Pi x^{\mathcal{B}(A)}.\mathcal{B}(B)$	
$\mathcal{B}(\lambda x^A.M)$	$:=$	$\lambda x^{\mathcal{B}(A)}.\mathcal{B}(M)$	
$\mathcal{B}(s)$	$:=$	s	
$\mathcal{B}(x\ l)$	$:=$	$\{{}^{x}\!/_{z}\}\mathcal{B}^z(l)$	z fresh
$\mathcal{B}(M\ l)$	$:=$	$\left\{{}^{\mathcal{B}(M)}\!/_{z}\right\}\mathcal{B}^z(l)$	z fresh
$\mathcal{B}(\langle P/x\rangle M)$	$:=$	$\left\{{}^{\mathcal{B}(P)}\!/_{x}\right\}\mathcal{B}(M)$	
$\mathcal{B}^y([])$	$:=$	y	
$\mathcal{B}^y(M\cdot l)$	$:=$	$\left\{{}^{y\ \mathcal{B}(M)}\!/_{z}\right\}\mathcal{B}^z(l)$	z fresh
$\mathcal{B}^y(l@l')$	$:=$	$\left\{{}^{\mathcal{B}^y(l)}\!/_{z}\right\}\mathcal{B}^z(l')$	z fresh
$\mathcal{B}^y(\langle P/x\rangle l)$	$:=$	$\left\{{}^{\mathcal{B}(P)}\!/_{x}\right\}\mathcal{B}^y(l)$	

Figure 8.4: From a PTSC to a PTS

Fig. 8.4 shows the encoding from a PTSC to a PTS.

Now we want to show that $\mathcal{B}$ and $\mathcal{A}$ form a reflection in PTSC of PTS. We first prove the following results:

Lemma 204

1. *$\mathcal{A}(t)$ and $\mathcal{A}_l(t)$ are always x-normal forms (provided l is).*
2. *If $l \longrightarrow_{B\mathsf{x}} l'$ then $\mathcal{A}_l(t) \longrightarrow_{B\mathsf{x}} \mathcal{A}_{l'}(t)$.*
3. *$\mathcal{A}_{l'}(t)\ l \longrightarrow^*_{\mathsf{x}} \mathcal{A}_{l'@l}(t)$ and $\mathcal{A}(t)\ l \longrightarrow^*_{\mathsf{x}} \mathcal{A}_l(t)$.*
4. *$\langle \mathcal{A}(u)/x\rangle \mathcal{A}(t) \longrightarrow^*_{\mathsf{x}} \mathcal{A}(\{^u\!/_x\}t)$ and $\langle \mathcal{A}(u)/x\rangle \mathcal{A}_l(t) \longrightarrow^*_{\mathsf{x}} \mathcal{A}_{\langle \mathcal{A}(u)/x\rangle l}(\{^u\!/_x\}t)$.*

Proof: Each of the above points is obtained by straightforward inductions on t. □

Now we study the composition of the two encodings:

Lemma 205 *Suppose M and l are x-normal forms.*

1. *If $t = x$ or $t = t_1\ t_2$ or $l \neq []$, then $\mathcal{A}_l(t) = \mathcal{A}(\{^t\!/_x\}\mathcal{B}^x(l))$ if $x \notin \mathsf{FV}(l)$.*
2. *$M = \mathcal{A}(\mathcal{B}(M))$.*

Proof: By simultaneous induction on l and M. □

Theorem 206 (A reflection of PTS in PTSC)

1. *$\longrightarrow_{B\mathsf{x}}$ strongly simulates $\longrightarrow_\beta$ through $\mathcal{A}$.*
2. *$\mathcal{B}$ and $\mathcal{A}$ form a reflection in PTSC of PTS.*

Proof:

1. If $t \longrightarrow_\beta u$ then $\mathcal{A}(t) \longrightarrow^+_{\mathsf{Bx}} \mathcal{A}(u)$ and $\mathcal{A}_l(t) \longrightarrow^+_{\mathsf{Bx}} \mathcal{A}_l(u)$, which are proved by induction on the derivation step, using Lemma 204.4 for the base case and Lemma 204.3.

2. - The first simulation is given by point 1.
   - If $M \longrightarrow_{\mathsf{B}} N$ then $\mathcal{B}(M) \longrightarrow^*_\beta \mathcal{B}(N)$, if $l \longrightarrow_{\mathsf{B}} l'$ then $\mathcal{B}^y(l) \longrightarrow^*_\beta \mathcal{B}^y(l')$, if $M \longrightarrow_{\mathsf{x}} N$ then $\mathcal{B}(M) = \mathcal{B}(N)$ and if $l \longrightarrow_{\mathsf{x}} l'$ then $\mathcal{B}^y(l) = \mathcal{B}^y(l')$, which are proved by simultaneous induction on the derivation step and case analysis.
   - $M \longrightarrow^*_{\mathsf{x}} \mathcal{A}(\mathcal{B}(M))$ holds by induction in SN^{x} (because x is terminating): by Lemma 205.2 it holds if M is an x-normal form, and if $M \longrightarrow_{\mathsf{x}} N$ then we can apply the induction hypothesis on N and by point 2 we have $\mathcal{B}(M) = \mathcal{B}(N)$.
   - $\mathcal{B}(\mathcal{A}(t)) = t$ and $\mathcal{B}(\mathcal{A}_l(t)) = \{^t\!/_x\}\mathcal{B}^x(l)$ (with $x \neq \mathsf{FV}(l)$) are obtained by simultaneous induction on t.

□

Now we use Theorem 206 to prove the confluence of PTSC and the equivalence of the equational theories.

Corollary 207 (Confluence) $\longrightarrow_x$ *and* $\longrightarrow_{Bx}$ *are confluent.*

Proof: From Theorems 5 and 206. □

Corollary 208 (Equational theories)

1. $t \longleftrightarrow^*_\beta u$ *if and only if* $\mathcal{A}(t) \longleftrightarrow^*_{Bx} \mathcal{A}(u)$.
2. $M \longleftrightarrow^*_{Bx} N$ *if and only if* $\mathcal{B}(M) \longleftrightarrow^*_\beta \mathcal{B}(N)$.

8.3 Typing system & properties

Given the set $\mathcal{S}$ of sorts, a particular *PTSC* is specified by a set $\mathcal{A} \subseteq \mathcal{S}^2$ and a set $\mathcal{R} \subseteq \mathcal{S}^3$. We shall see an example in section 8.5.

Definition 118 (Environments)

- In this chapter, *environments*, denoted $\Gamma, \Delta, \Pi, \ldots$ are lists of pairs from $\mathcal{X} \times \mathcal{T}$ denoted $x : A$. If a pair $x{:}A$ is an element of an environment Γ, we also write $(x : A) \in \Gamma$.
- We define the *domain* of an environment and the *application of a substitution to an environment* by induction on the environment as follows:

$$\begin{array}{lll} \mathsf{Dom}([]) & := [] & \langle P/y\rangle([]) := [] \\ \mathsf{Dom}(\Gamma, x : A) & := \mathsf{Dom}(\Gamma), x & \langle P/y\rangle(\Gamma, x{:}A) := \langle P/y\rangle\Gamma, x{:}\langle P/y\rangle A \end{array}$$

  The domain of an environment is thus a list, but again we allow the notation $x \in \mathsf{Dom}(\Gamma)$ as if it were a set, as well as $\mathsf{Dom}(\Gamma) \cap \mathsf{Dom}(\Delta)$ which *is* the set $\{x \in \mathcal{X} \mid x \in \mathsf{Dom}(\Gamma) \wedge x \in \mathsf{Dom}(\Delta)\}$.
- We define the following *sub-environment* relation:
  $\Gamma \sqsubseteq \Delta$ if for all $(x : A) \in \Gamma$, there is $(x : B) \in \Delta$ with $A \longleftrightarrow^*_{\mathsf{Bx}} B$.

The inference rules in Fig. 8.5 inductively define the derivability of three kinds of judgement: some of the form Γ wf, some of the form $\Gamma \vdash M{:}A$ and some of the form $\Gamma; B \vdash l{:}A$. In the last two cases we call these judgements *sequents*. In the last case, B is said to be in the *stoup* of the sequent, according to a terminology due to Girard. Side-conditions are used, such as $(s_1, s_2, s_3) \in \mathcal{R}$, $x \notin \mathsf{Dom}(\Gamma)$, $A \longleftrightarrow^*_{\mathsf{Bx}} B$ or $\Delta \sqsubseteq \Gamma$, and we use the abbreviation $\Delta \sqsubseteq \Gamma$ $\mathsf{wf}_{\mathsf{PTSC}}$ for $\Delta \sqsubseteq \Gamma$ and Γ $\mathsf{wf}_{\mathsf{PTSC}}$. Derivability in a PTSC of the three kinds of judgement is denoted Γ $\mathsf{wf}_{\mathsf{PTSC}}$, $\Gamma \vdash_{\mathsf{PTSC}} M{:}A$, and $\Gamma; B \vdash_{\mathsf{PTSC}} l{:}A$, respectively.

$$\frac{}{[]\ \mathsf{wf}}\ \mathsf{empty} \qquad \frac{\Gamma \vdash A : s \quad x \notin \mathsf{Dom}(\Gamma)}{\Gamma, x:A\ \mathsf{wf}}\ \mathsf{extend}$$

$$\frac{\Gamma \vdash A : s}{\Gamma; A \vdash [] : A}\ \mathsf{ax} \qquad \frac{\Gamma \vdash \Pi x^A.B : s \quad \Gamma \vdash M : A \quad \Gamma; \langle M/x\rangle B \vdash l : C}{\Gamma; \Pi x^A.B \vdash M \cdot l : C}\ \Pi\mathsf{l}$$

$$\frac{\Gamma; C \vdash l : A \quad \Gamma \vdash B : s \quad A \longleftrightarrow^*_{\mathsf{Bx}} B}{\Gamma; C \vdash l : B}\ \mathsf{conv}'_r \qquad \frac{\Gamma; A \vdash l : C \quad \Gamma \vdash B : s \quad A \longleftrightarrow^*_{\mathsf{Bx}} B}{\Gamma; B \vdash l : C}\ \mathsf{conv}_l$$

$$\frac{\Gamma; C \vdash l' : A \qquad \Gamma; A \vdash l : B}{\Gamma; C \vdash l' @ l : B}\ \mathsf{cut}_1$$

$$\frac{\Gamma \vdash P : A \qquad \Gamma, x:A, \Delta; B \vdash l : C \quad \Gamma, \langle P/x\rangle\Delta \sqsubseteq \Delta'\ \mathsf{wf}}{\Delta'; \langle P/x\rangle B \vdash \langle P/x\rangle l : \langle P/x\rangle C}\ \mathsf{cut}_2$$

$$\frac{\Gamma\ \mathsf{wf} \quad (s, s') \in \mathcal{A}}{\Gamma \vdash s : s'}\ \mathsf{sorted} \qquad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x:A \vdash B : s_2 \quad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash \Pi x^A.B : s_3}\ \Pi\mathsf{wf}$$

$$\frac{\Gamma; A \vdash l : B \quad (x:A) \in \Gamma}{\Gamma \vdash x\ l : B}\ \mathsf{select}_x \qquad \frac{\Gamma \vdash \Pi x^A.B : s \quad \Gamma, x:A \vdash M : B}{\Gamma \vdash \lambda x^A.M : \Pi x^A.B}\ \Pi\mathsf{r}$$

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash B : s \quad A \longleftrightarrow^*_{\mathsf{Bx}} B}{\Gamma \vdash M : B}\ \mathsf{conv}_r$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma; A \vdash l : B}{\Gamma \vdash M\ l : B}\ \mathsf{cut}_3$$

$$\frac{\Gamma \vdash P : A \qquad \Gamma, x:A, \Delta \vdash M : C \quad \Gamma, \langle P/x\rangle\Delta \sqsubseteq \Delta'\ \mathsf{wf}}{\Delta' \vdash \langle P/x\rangle M : C'}\ \mathsf{cut}_4$$

where either $(C' = C \in \mathcal{S})$ or $C \notin \mathcal{S}$ and $C' = \langle P/x\rangle C$

Figure 8.5: Typing rules of a PTSC

Since the substitution of a variable in an environment affects the rest of the environment (which could depend on the variable), the two rules for explicit substitutions cut_2 and cut_4 must have a particular shape that is admittedly complex: thinning (Lemma 212) is built-in by allowing a controlled change of environment. This may appear artificial, but simpler versions that we have tried failed the thinning property. More generally, typing rules for explicit substitutions in type theory are known to be a tricky issue (see for instance [Blo01]), often leading to the failure of subject reduction (Theorem 216). The rules here are sound in that respect, but more elegant alternatives are still to be investigated, possibly by enriching the structure of environments as in [Blo01].

The case analysis for C' in the rule cut_4 is only necessary for Lemma 209.2 to hold in the presence of top sorts (untyped sorts), and is avoided in [Blo01] by not using explicit substitutions for types in sequents. Here we were attracted by the uniformity of using them everywhere, the use of implicit substitutions for C' and the stoup of the third premiss of $\Pi\mathsf{l}$ being only a minor variant.

There are three *conversion rules* conv_r, conv'_r, and conv_l in order to deal with the two kinds of judgement and, for one of them, convert the type in the stoup.

Lemma 209 (Properties of typing judgements) *If* $\Gamma \vdash_{\mathit{PTSC}} M : A$ *(resp.* $\Gamma; B \vdash_{\mathit{PTSC}} l : C$*) then* $\mathit{FV}(M) \subseteq \mathit{Dom}(\Gamma)$ *(resp.* $\mathit{FV}(l) \subseteq \mathit{Dom}(\Gamma)$*), and the following judgements can be derived with strictly smaller typing derivations:*

1. Γ $\mathit{wf}_{\mathit{PTSC}}$
2. $\Gamma \vdash_{\mathit{PTSC}} A : s$ *for some* $s \in \mathcal{S}$*, or* $A \in \mathcal{S}$
   (resp. $\Gamma \vdash_{\mathit{PTSC}} B : s$ *and* $\Gamma \vdash_{\mathit{PTSC}} C : s'$ *for some* $s, s' \in \mathcal{S}$*)*

Proof: Straightforward induction on derivations. □

Corollary 210 (Properties of well-formed environments)

1. *If* $\Gamma, x : A, \Delta$ $\mathit{wf}_{\mathit{PTSC}}$ *then* $\Gamma \vdash_{\mathit{PTSC}} A : s$ *for some* $s \in \mathcal{S}$ *with a strictly smaller derivation, with* $x \notin \mathit{Dom}(\Gamma) \cup \mathit{Dom}(\Delta)$ *and* $\mathit{FV}(A) \subseteq \mathit{Dom}(\Gamma)$
   (and in particular $x \notin \mathit{FV}(A)$*).*
2. *If* Γ, Δ $\mathit{wf}_{\mathit{PTSC}}$ *then* Γ $\mathit{wf}_{\mathit{PTSC}}$*.*

Proof:

1. The first point is proved by induction on the length of Δ (as a list): The base case, when Δ is empty, is obtained by rule extend. Otherwise $\Delta = \Delta', y : B$ and by rule extend we get $y \neq x$ and $\Gamma, x : A, \Delta' \vdash_{\mathsf{PTSC}} A : s_B$ for some $s_B \in \mathcal{S}$ with a strictly smaller tree. Hence, by Lemma 209.1 we get $\Gamma, x : A, \Delta'$ $\mathsf{wf}_{\mathsf{PTSC}}$ with an even smaller tree, on which it suffices to apply the induction hypothesis.

   The facts that $\mathsf{FV}(A) \subseteq \mathsf{Dom}(\Gamma)$ and $x \notin \mathsf{FV}(A)$ then come from Lemma 209.

2. The second point is a corollary of the first and Lemma 209: if Δ is empty then the statement trivially holds, otherwise Δ starts with $x : A$, so we apply the first point to get $\Gamma \vdash_{\mathsf{PTSC}} A : s$ for some $s \in \mathcal{S}$, and we conclude with Lemma 209.1.

□

Now we prove the *weakening* property:

Lemma 211 (Weakening) *Suppose* Γ, Γ' $\mathit{wf}_{\mathit{PTSC}}$ *and* $\mathit{Dom}(\Gamma') \cap \mathit{Dom}(\Delta) = \emptyset$.

1. *If* $\Gamma, \Delta \vdash_{\mathit{PTSC}} M : B$ *then* $\Gamma, \Gamma', \Delta \vdash_{\mathit{PTSC}} M : B$.
2. *If* $\Gamma, \Delta; C \vdash_{\mathit{PTSC}} l : B$, *then* $\Gamma, \Gamma', \Delta; C \vdash_{\mathit{PTSC}} l : B$.
3. *If* Γ, Δ $\mathit{wf}_{\mathit{PTSC}}$, *then* Γ, Γ', Δ $\mathit{wf}_{\mathit{PTSC}}$.

Proof: By induction on derivations.

- For rules cut_2 and cut_4 we use the fact that if $\Gamma_1 \sqsubseteq \Gamma, \Delta$ then $\Gamma_1 \sqsubseteq \Gamma, \Gamma', \Delta$, as well as the induction hypothesis to prove that Γ, Γ', Δ $\mathsf{wf}_{\mathsf{PTSC}}$.
- For rule select_x we use the fact that if $(x : A) \in \Gamma, \Delta$ then $(x : A) \in \Gamma, \Gamma', \Delta$.
- For rule extend we have the case disjunction: if Δ is empty then we use the hypothesis Γ, Γ' $\mathsf{wf}_{\mathsf{PTSC}}$, otherwise we use the induction hypothesis.
- The case of the other rules is straightforward.

□

We can also strengthen the *weakening property* into the *thinning* property. This allows to weaken the environment, change its order, and convert the types inside, as long as it remains well-formed:

Lemma 212 (Thinning) *Suppose* $\Gamma \sqsubseteq \Gamma'$ $\mathit{wf}_{\mathit{PTSC}}$.

1. *If* $\Gamma \vdash_{\mathit{PTSC}} M : B$ *then* $\Gamma' \vdash_{\mathit{PTSC}} M : B$.
2. *If* $\Gamma; C \vdash_{\mathit{PTSC}} l : B$, *then* $\Gamma'; C \vdash_{\mathit{PTSC}} l : B$.

Proof: Again, by induction on the typing derivation.

- For rules cut_2 and cut_4 we use the fact that if $\Gamma_1 \sqsubseteq \Gamma$ and $\Gamma \sqsubseteq \Gamma'$ then $\Gamma_1 \sqsubseteq \Gamma'$.
- For rule select_x we have $(x : A) \in \Gamma$, so by definition of $\sqsubseteq$ we have $\Gamma' = \Delta_1, x : A', \Delta_2$, with $A \longleftrightarrow^*_{\mathsf{Bx}} A'$. Hence, we can apply the induction hypothesis, but the type in the stoup (A) no longer matches the type of x (A'). So by Lemma 210.1 we have $\Delta_1 \vdash_{\mathsf{PTSC}} A' : s$ for some s, and by Lemma 211.1 we get $\Gamma' \vdash_{\mathsf{PTSC}} A' : s$. Hence we use rule conv_r to convert the formula A in the stoup to A'.

- For rules $\Pi\mathsf{wf}$ and $\Pi\mathsf{r}$, we want to use the induction hypothesis so we choose $x \notin \mathsf{Dom}(\Gamma')$ and we must prove that $\Gamma, x:A \sqsubseteq \Gamma', x:A$ (which holds by definition of $\sqsubseteq$) and $\Gamma', x:A\ \mathsf{wf}_{\mathsf{PTSC}}$. To prove the latter, we use Lemma 209 to get $\Gamma, x:A\ \mathsf{wf}_{\mathsf{PTSC}}$ with a smaller derivation, and $\Gamma \vdash_{\mathsf{PTSC}} A:s$ for some s, with an even smaller derivation, on which we apply the induction hypothesis and thus get $\Gamma' \vdash_{\mathsf{PTSC}} A:s$ and then $\Gamma', x:A\ \mathsf{wf}_{\mathsf{PTSC}}$.

- The case of the other rules is straightforward.

□

For the purpose of proving subject reduction we want to prove a lemma called the *Generation Lemma*, and for that we need the following definition.

Definition 119 (Derived without conversion) We write $\Gamma \vdash^*_{\mathsf{PTSC}} M:A$ (resp. $\Gamma; B \vdash^*_{\mathsf{PTSC}} l:A$) whenever we can derive $\Gamma \vdash_{\mathsf{PTSC}} M:A$ (resp. $\Gamma; B \vdash_{\mathsf{PTSC}} l:A$) and the last rule is not a conversion rule.

In contrast to proof systems in propositional logic, the generation lemma is non-trivial:

Lemma 213 (Generation Lemma)

1. (a) *If* $\Gamma \vdash_{PTSC} s:C$ *then there is* s' *such that* $\Gamma \vdash^*_{PTSC} s:s'$ *with* $C \longleftrightarrow^*_{Bx} s'$.
   (b) *If* $\Gamma \vdash_{PTSC} \Pi x^A.B:C$ *then there is* s *such that* $\Gamma \vdash^*_{PTSC} \Pi x^A.B:s$ *with* $C \longleftrightarrow^*_{Bx} s$.
   (c) *If* $\Gamma \vdash_{PTSC} \lambda x^A.M:C$ *then there is* B *such that* $C \longleftrightarrow^*_{Bx} \Pi x^A.B$ *and* $\Gamma \vdash^*_{PTSC} \lambda x^A.M:\Pi x^A.B$.
   (d) *If* $\Gamma \vdash_{PTSC} \langle M/x\rangle N:C$ *then there is* C' *such that* $\Gamma \vdash^*_{PTSC} \langle M/x\rangle N:C'$ *with* $C \longleftrightarrow^*_{Bx} C'$.
   (e) *If* M *is not of the above forms and* $\Gamma \vdash_{PTSC} M:C$, *then* $\Gamma \vdash^*_{PTSC} M:C$.
2. (a) *If* $\Gamma; B \vdash_{PTSC} [\,]:C$ *then* $B \longleftrightarrow^*_{Bx} C$.
   (b) *If* $\Gamma; D \vdash_{PTSC} M \cdot l:C$ *then there are* A, B *such that* $D \longleftrightarrow^*_{Bx} \Pi x^A.B$ *and* $\Gamma; \Pi x^A.B \vdash^*_{PTSC} M \cdot l:C$.
   (c) *If* $\Gamma; B \vdash_{PTSC} \langle M/x\rangle l:C$ *then are* B', C' *such that* $\Gamma; B' \vdash^*_{PTSC} \langle M/x\rangle l:C'$ *with* $C \longleftrightarrow^*_{Bx} C'$ *and* $B \longleftrightarrow^*_{Bx} B'$.
   (d) *If* l *is not of the above forms and* $\Gamma; D \vdash_{PTSC} l:C$ *then* $\Gamma; D \vdash^*_{PTSC} l:C$.

Proof: Straightforward induction on the typing tree. □

Remark 214 The following rule is derivable, using a conversion rule:

$$\frac{\Gamma \vdash_{\mathsf{PTSC}} Q : A \qquad \Gamma, (x : A), \Delta \vdash_{\mathsf{PTSC}} M : C \quad \Delta' \vdash_{\mathsf{PTSC}} \langle Q/x \rangle C : s \quad \Gamma, \langle Q/x \rangle \Delta \sqsubseteq \Delta'}{\Delta' \vdash_{\mathsf{PTSC}} \langle Q/x \rangle M : \langle Q/x \rangle C}$$

Proving subject reduction relies on the following properties of $\longrightarrow_{\mathsf{Bx}}$:

Lemma 215

- *Two distinct sorts are not convertible.*
- *A* Π*-construct is not convertible to a sort.*
- $\Pi x^A.B \longleftrightarrow^*_{Bx} \Pi x^D.E$ *if and only if* $A \longleftrightarrow^*_{Bx} D$ *and* $B \longleftrightarrow^*_{Bx} E$.
- *If* $y \notin FV(P)$, *then* $M \longleftrightarrow^*_{Bx} \langle N/y \rangle P$.
- $\langle M/y \rangle \langle N/x \rangle P \longleftrightarrow^*_{Bx} \langle \langle M/y \rangle N/x \rangle \langle M/y \rangle P$ *(provided* $x \notin FV(M)$*).*

Proof: The first three properties are a consequence of the confluence of the rewrite system (Corollary 207). The last two rely on the fact that the system xsubst is terminating, so that only the case when P is an xsubst-normal form remains to be checked, which is done by structural induction. □

Using all of the results above, subject reduction can be proved:

Theorem 216 (Subject reduction in a PTSC)

1. *If* $\Gamma \vdash_{PTSC} M : F$ *and* $M \longrightarrow_{Bx} M'$, *then* $\Gamma \vdash_{PTSC} M' : F$
2. *If* $\Gamma; H \vdash_{PTSC} l : F$ *and* $l \longrightarrow_{Bx} l'$, *then* $\Gamma; H \vdash_{PTSC} l' : F$

Proof: By simultaneous induction on the typing tree. For every rule, if the reduction takes place within a sub-term that is typed by one of the premisses of the rule (e.g. the conversion rules), then we can apply the induction hypothesis on that premiss. In particular, this takes care of the cases where the last typing rule is a conversion rule.

So it now suffices to look at the root reductions. For lack of space we often do not display some minor premisses in following derivations, but we mention them before or after. We also drop the subscript PTSC from derivable judgements.

B $(\lambda x^A.N)\ (P \cdot l_1) \longrightarrow (\langle P/x \rangle N)\ l_1$

By the Generation Lemma, 1.(c) and 2.(b), there exist B, D, E such that:

$$\frac{\dfrac{\Gamma \vdash \Pi x^A.B : s \quad \Gamma, x : A \vdash N : B}{\Gamma \vdash \lambda x^A.N : C} \qquad \dfrac{\Gamma \vdash P : D \quad \Gamma; \langle P/x \rangle E \vdash l_1 : F}{\Gamma; C \vdash P \cdot l_1 : F}}{\Gamma \vdash^* (\lambda x^A.N)\ (P \cdot l_1) : F}$$

with $\Pi x^A.B \longleftrightarrow^*_{\mathsf{Bx}} C \longleftrightarrow^*_{\mathsf{Bx}} \Pi x^D.E$. Hence, $A \longleftrightarrow^*_{\mathsf{Bx}} D$ and $B \longleftrightarrow^*_{\mathsf{Bx}} E$.
Moreover, $\Gamma \vdash A : s_A$, $\Gamma, x : A \vdash B : s_B$ and Γ wf.
Hence, we get $\Gamma \vdash \langle P/x\rangle B : s_B$, so:

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash P : D}{\Gamma \vdash P : A} \quad \Gamma, x : A \vdash N : B}{\Gamma \vdash \langle P/x\rangle N : \langle P/x\rangle B} \quad \dfrac{\Gamma; \langle P/x\rangle E \vdash l_1 : F}{\Gamma; \langle P/x\rangle B \vdash l_1 : F}}{\Gamma \vdash (\langle P/x\rangle N\ l_1) : F}$$

with $\langle P/x\rangle B \longleftrightarrow^*_{\mathsf{Bx}} \langle P/x\rangle E$.

As A1 $(N \cdot l_1)@l_2 \longrightarrow N \cdot (l_1@l_2)$
By the Generation Lemma 2.(b), there are A and B such that $H \longleftrightarrow^*_{\mathsf{Bx}} \Pi x^A.B$ and:

$$\dfrac{\dfrac{\Gamma \vdash \Pi x^A.B : s \quad \Gamma \vdash N : A \quad \Gamma; \langle N/x\rangle B \vdash l_1 : C}{\Gamma; H \vdash N \cdot l_1 : C} \quad \Gamma; C \vdash l_2 : F}{\Gamma; H \vdash^* (N \cdot l_1)@l_2 : F}$$

Hence,

$$\dfrac{\Gamma \vdash H : s_H \quad \dfrac{\Gamma \vdash \Pi x^A.B : s \quad \Gamma \vdash N : A \quad \dfrac{\Gamma; \langle N/x\rangle B \vdash l_1 : C \quad \Gamma; C \vdash l_2 : F}{\Gamma; \langle N/x\rangle B \vdash l_1@l_2 : C}}{\Gamma; \Pi x^A.B \vdash N \cdot (l_1@l_2) : F}}{\Gamma; H \vdash N \cdot (l_1@l_2) : F}$$

A2 $[]@l_1 \longrightarrow l_1$
By the Generation Lemma 2.(a), we have $A \longleftrightarrow^*_{\mathsf{Bx}} H$ and

$$\dfrac{\Gamma; H \vdash [] : A \quad \Gamma; A \vdash l_1 : F}{\Gamma; H \vdash^* []@l_1 : F}$$

Since $\Gamma \vdash H : s_H$, we get

$$\dfrac{\Gamma; A \vdash l_1 : F}{\Gamma; H \vdash l_1 : F}$$

A3 $(l_1@l_2)@l_3 \longrightarrow l_1@(l_2@l_3)$
By the Generation Lemma 2.(d),

$$\dfrac{\dfrac{\Gamma; H \vdash l_1 : B \quad \Gamma; B \vdash l_2 : A}{\Gamma; H \vdash^* l_1@l_2 : A} \quad \Gamma; A \vdash l_3 : F}{\Gamma; H \vdash^* (l_1@l_2)@l_3 : F}$$

Hence,

$$\dfrac{\Gamma; H \vdash l_1 : B \quad \dfrac{\Gamma; B \vdash l_2 : A \quad \Gamma; A \vdash l_3 : F}{\Gamma; B \vdash l_2@l_3 : F}}{\Gamma; H \vdash l_1@(l_2@l_3) : F}$$

Bs B1 $N\ [] \longrightarrow N$

$$\frac{\Gamma \vdash N : A \quad \Gamma; A \vdash [] : F}{\Gamma \vdash^* N\ [] : F}$$

By the Generation Lemma 2.(a), we have $A \longleftrightarrow^*_{\mathsf{Bx}} F$.
Since $\Gamma \vdash F : s_F$, we get

$$\frac{\Gamma \vdash N : A}{\Gamma \vdash N : F}$$

B2 $(x\ l_1)\ l_2 \longrightarrow x\ (l_1 @ l')$
By the Generation Lemma 1.(e),

$$\frac{\dfrac{\Gamma; A \vdash l_1 : B \quad (x : A) \in \Gamma}{\Gamma \vdash^* x\ l : B} \qquad \Gamma; B \vdash l_2 : F}{\Gamma \vdash^* (x\ l_1)\ l_2 : F}$$

Hence,

$$\frac{(x : A) \in \Gamma \qquad \dfrac{\Gamma; A \vdash l_1 : B \quad \Gamma; B \vdash l_2 : F}{\Gamma; A \vdash l_1 @ l_2 : F}}{\Gamma \vdash x\ (l_1 @ l_2) : F}$$

B3 $(N\ l_1)\ l_2 \longrightarrow N\ (l_1 @ l_2)$
By the Generation Lemma 1.(e),

$$\frac{\dfrac{\Gamma \vdash N : A \quad \Gamma; A \vdash l_1 : B}{\Gamma \vdash^* N\ l_1 : B} \qquad \Gamma; B \vdash l_2 : F}{\Gamma \vdash^* (N\ l_1)\ l_2 : F}$$

Hence,

$$\frac{\Gamma \vdash N : A \qquad \dfrac{\Gamma; A \vdash l_1 : B \quad \Gamma; B \vdash l_2 : F}{\Gamma; A \vdash l_1 @ l_2 : F}}{\Gamma \vdash N\ (l_1 @ l_2) : F}$$

Cs We have a redex of the form $\langle Q/y \rangle R$ typed by:

$$\frac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta \vdash R : F' \quad \Delta', \langle Q/y \rangle \Delta \sqsubseteq \Gamma\ \ \mathsf{wf}}{\Gamma \vdash^* \langle Q/y \rangle R : F}$$

with either $F = F' \in \mathcal{S}$ or $F = \langle Q/y \rangle F'$.
In the latter case, $\Gamma \vdash F : s_F$ for some $s_F \in \mathcal{S}$. We also have Γ wf.
Let us consider each rule:

C1 $\langle Q/y \rangle \lambda x^A.N \longrightarrow \lambda x^{\langle Q/y \rangle A}.\langle Q/y \rangle N$

$R = \lambda x^A.N$
By the Generation Lemma 1.(b), there is s_3 such that $C \longleftrightarrow^*_{\mathsf{Bx}} s_3$ and:

$$\dfrac{\dfrac{\Delta', y:E, \Delta \vdash A : s_1 \qquad \Delta', y:E, \Delta, x:A \vdash B : s_2}{\Delta', y:E, \Delta \vdash \Pi x^A.B : C} \qquad \Delta', y:E, \Delta, x:A \vdash N : B}{\Delta', y:E, \Delta \vdash \lambda x^A.N : F'}$$

with $(s_1, s_2, s_3) \in \mathcal{R}$ and $F' \equiv \Pi x^A.B$. Hence, $F' \notin \mathcal{S}$, so $F = \langle Q/y\rangle F' \longleftrightarrow^*_{\mathsf{Bx}} \langle Q/y\rangle \Pi x^A.B \longleftrightarrow^*_{\mathsf{Bx}} \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B$. We have:

$$\dfrac{\Delta' \vdash Q : E \quad \Delta', y:E, \Delta \vdash A : s_1}{\Gamma \vdash \langle Q/y\rangle A : s_1}$$

Hence, $\Gamma, x : \langle Q/y\rangle A$ wf and $\Delta', \langle Q/y\rangle\Delta, x : \langle Q/y\rangle A \sqsubseteq \Gamma, x : \langle Q/y\rangle A$, so:

$$\dfrac{\Delta' \vdash Q : E \quad \Delta', y:E, \Delta, x:A \vdash B : s_2}{\Gamma, x : \langle Q/y\rangle A \vdash \langle Q/y\rangle B : s_2}$$

so that $\Gamma \vdash \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B : s_3$ and

$$\dfrac{\dfrac{\dfrac{\Delta' \vdash Q : E \quad \Delta', y:E, \Delta, x:A \vdash N : B}{\Gamma, x : \langle Q/y\rangle A \vdash \langle Q/y\rangle N : \langle Q/y\rangle B}}{\Gamma \vdash \lambda x^{\langle Q/y\rangle A}.\langle Q/y\rangle N : \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B} \qquad F \longleftrightarrow^*_{\mathsf{Bx}} \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B}{\Gamma \vdash \lambda x^{\langle Q/y\rangle A}.\langle Q/y\rangle N : F}$$

C2 $\langle Q/y\rangle(y\ l_1) \longrightarrow Q\ \langle Q/y\rangle l_1$
$R = y\ l_1$
By the Generation Lemma 1.(e), $\Delta', y:E, \Delta; E \vdash l_1 : F'$. Now notice that $y \notin FV(E)$, so $\langle Q/y\rangle E \longleftrightarrow^*_{\mathsf{Bx}} E$ and $\Delta' \vdash E : s_E$. Also, $\Delta' \sqsubseteq \Gamma$, so

$$\dfrac{\dfrac{\Delta' \vdash Q : E}{\Gamma \vdash Q : E} \qquad \dfrac{\Delta' \vdash Q : E \quad \Delta', y:E, \Delta; E \vdash l_1 : F'}{\Gamma; \langle Q/y\rangle E \vdash \langle Q/y\rangle l_1 : F} \qquad \dfrac{\Delta' \vdash E : s_E}{\Gamma \vdash E : s_E}}{\Gamma; E \vdash \langle Q/y\rangle l_1 : F}$$
$$\Gamma \vdash Q\ \langle Q/y\rangle l_1 : F$$

C3 $\langle Q/y\rangle(x\ l_1) \longrightarrow x\ \langle Q/y\rangle l_1$
$R = x\ l_1$
By the Generation Lemma 1.(e), $\Delta', y:E, \Delta; A \vdash l_1 : F'$ with $(x:A) \in \Delta', \Delta$. Let B be the type of x in Γ. We have

$$\dfrac{\dfrac{\dfrac{\Delta' \vdash Q : E \quad \Delta', y:E, \Delta; A \vdash l_1 : F'}{\Gamma; \langle Q/y\rangle A \vdash \langle Q/y\rangle l_1 : F} \qquad \Gamma \vdash B : s_B}{\Gamma; B \vdash \langle Q/y\rangle l_1 : F}}{\Gamma \vdash x\ \langle Q/y\rangle l_1 : F}$$

Indeed, if $x \in \mathsf{Dom}(\Delta)$ then $B \longleftrightarrow^*_{\mathsf{Bx}} \langle Q/y\rangle A$, otherwise $B \longleftrightarrow^*_{\mathsf{Bx}} A$ with $y \notin FV(A)$, so in both cases $B \longleftrightarrow^*_{\mathsf{Bx}} \langle Q/y\rangle A$. Besides, Γ wf so $\Gamma \vdash B : s_B$.

C4 $\langle Q/y\rangle(N\ l_1) \longrightarrow \langle Q/y\rangle N\ \langle Q/y\rangle l_1$
$R = N\ l_1$
By the Generation Lemma 1.(e),

$$\frac{\Delta', y : E, \Delta \vdash N : A \quad \Delta', y : E, \Delta; A \vdash l_1 : F'}{\Delta', y : E, \Delta \vdash^* N\ l_1 : F'}$$

Also, we have

$$\frac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta \vdash A : s_A}{\Gamma \vdash \langle Q/y\rangle A : s_A}$$

Hence,

$$\frac{\dfrac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta \vdash N : A}{\Gamma \vdash \langle Q/y\rangle N : \langle Q/y\rangle A} \qquad \dfrac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta; A \vdash l_1 : F'}{\Gamma; \langle Q/y\rangle A \vdash \langle Q/y\rangle l_1 : F}}{\Gamma \vdash \langle Q/y\rangle N\ \langle Q/y\rangle l_1 : F}$$

C5 $\langle Q/y\rangle \Pi x^A.B \longrightarrow \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B$
$R = \Pi x^A.B$
By the Generation Lemma 1.(b), there exist s_3 such that $F' \longleftrightarrow^*_{\mathsf{Bx}} s_3$ and:

$$\frac{\Delta', y : E, \Delta \vdash A : s_1 \qquad \Delta', y : E, \Delta, x : A \vdash B : s_2}{\Delta', y : E, \Delta \vdash \Pi x^A.B : F'}$$

with $(s_1, s_2, s_3) \in \mathcal{R}$.

$$\frac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta \vdash A : s_1}{\Gamma \vdash \langle Q/y\rangle A : s_1}$$

Hence, $\Gamma, x : \langle Q/y\rangle A$ wf and $\Delta', \langle Q/y\rangle\Delta, x : \langle Q/y\rangle A \sqsubseteq \Gamma, x : \langle Q/y\rangle A$, so:

$$\frac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta, x : A \vdash B : s_2}{\Gamma, x : \langle Q/y\rangle A \vdash \langle Q/y\rangle B : s_2}$$

so that $\Gamma \vdash \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B : s_3$. Now if $F' \in \mathcal{S}$, then $F = F' = s_3$ and we are done. Otherwise $F = \langle Q/y\rangle F' \longleftrightarrow^*_{\mathsf{Bx}} \langle Q/y\rangle s_3 \longleftrightarrow^*_{\mathsf{Bx}} s_3$, and we conclude using a conversion rule (because $\Gamma \vdash F : s_F$).

C6 $\langle Q/y\rangle s \longrightarrow s$
$R = s$
By the Generation Lemma 1.(a), we get $F' \longleftrightarrow^*_{\mathsf{Bx}} s'$ for some s' with $(s, s') \in \mathcal{A}$. Since Γ wf, we get $\Gamma \vdash s : s'$. If $F' \in \mathcal{S}$, then $F = F' = s'$ and we are done. Otherwise $F = \langle Q/y\rangle F' \longleftrightarrow^*_{\mathsf{Bx}} \langle Q/y\rangle s' \longleftrightarrow^*_{\mathsf{Bx}} s'$ and we conclude using a conversion rule (because $\Gamma \vdash F : s_F$).

Ds We have a redex of the form $\langle Q/y\rangle l_1$ typed by:

$$\frac{\Delta' \vdash Q : E \quad \Delta', y : E, \Delta; H' \vdash l_1 : F' \quad \Delta', \langle Q/y\rangle\Delta \sqsubseteq \Gamma \ \mathsf{wf}}{\Gamma; H \vdash^* \langle Q/y\rangle l_1 : F}$$

with $F = \langle Q/y\rangle F'$ and $H = \langle Q/y\rangle H'$. We also have Γ wf and $\Gamma \vdash H : s_H$ and $\Gamma \vdash F : s_F$.

Let us consider each rule:

D1 $\langle Q/y\rangle[] \longrightarrow []$
$l_1 = []$
By the Generation Lemma 2.(a), $H' \longleftrightarrow^*_{\mathsf{Bx}} F'$, so $H \longleftrightarrow^*_{\mathsf{Bx}} F$.

$$\frac{\dfrac{\Gamma \vdash H : s_H}{\Gamma; H \vdash [] : H} \qquad \Gamma \vdash F : s_F}{H \vdash [] : F}$$

D2 $\langle Q/y\rangle(N \cdot l_2) \longrightarrow (\langle Q/y\rangle N) \cdot (\langle Q/y\rangle l_2)$
$l_1 = N \cdot l_2$
By the Generation Lemma 2.(b), there are A, B such that $H' \longleftrightarrow^*_{\mathsf{Bx}} \Pi x^A.B$ and:

$$\frac{\Delta', y : E, \Delta \vdash \Pi x^A.B : s \quad \Delta', y : E, \Delta \vdash N : A \quad \Delta', y : E, \Delta; \langle N/x\rangle B \vdash l_2 : F'}{\Delta', y : E, \Delta; \Pi x^A.B \vdash^* l_1 : F'}$$

From $\Delta', y : E, \Delta; \langle N/x\rangle B \vdash l_2 : F'$ we get

$$\Gamma; \langle Q/y\rangle\langle N/x\rangle B \vdash \langle Q/y\rangle l_2 : F$$

From $\Delta', y : E, \Delta \vdash N : A$ we get $\Gamma \vdash \langle Q/y\rangle N : \langle Q/y\rangle A$.
From $\Delta', y : E, \Delta \vdash \Pi x^A.B : s$ the Generation Lemma 1.(b) provides $\Delta', y : E, \Delta \vdash A : s_A$ and $\Delta', y : E, \Delta, x : A \vdash B : s_B$. Hence we get

$$\frac{\Delta', y : E, \Delta \vdash A : s_A}{\Gamma \vdash \langle Q/y\rangle A : s_A}$$

and thus $\Gamma, x : \langle Q/y\rangle A$ wf and then

$$\frac{\Delta', y : E, \Delta, x : A \vdash B : s_B}{\Gamma, x : \langle Q/y\rangle A \vdash \langle Q/y\rangle B : s_B}$$

From that we get both $\Gamma \vdash \Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B : s$ and
$\Gamma \vdash \langle\langle Q/y\rangle N/x\rangle\langle Q/y\rangle B : s_B$.

Note that $\Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B \longleftarrow\!\longrightarrow^*_{\mathsf{Bx}} \langle Q/y\rangle \Pi x^A.B \longleftarrow\!\longrightarrow^*_{\mathsf{Bx}} \langle Q/y\rangle H' = H$. We get

$$\dfrac{\dfrac{\Gamma \vdash \langle Q/y\rangle N : \langle Q/y\rangle A \quad \dfrac{\Gamma;\langle Q/y\rangle\langle N/x\rangle B \vdash \langle Q/y\rangle l_2 : F}{\Gamma;\langle\langle Q/y\rangle N/x\rangle\langle Q/y\rangle B \vdash \langle Q/y\rangle l_2 : F}}{\Gamma;\Pi x^{\langle Q/y\rangle A}.\langle Q/y\rangle B \vdash (\langle Q/y\rangle N)\cdot(\langle Q/y\rangle l_2) : F}}{\Gamma; H \vdash (\langle Q/y\rangle N)\cdot(\langle Q/y\rangle l_2) : F}$$

D3 $\langle Q/y\rangle(l_2@l_3) \longrightarrow (\langle Q/y\rangle l_2)@(\langle Q/y\rangle l_3)$
$l_1 = l_2@l_3$
By the Generation Lemma 2.(d),

$$\dfrac{\Delta', y : E, \Delta; H' \vdash l_2 : A \quad \Delta', y : E, \Delta; A \vdash l_3 : F'}{\Delta', y : E, \Delta; H' \vdash^* l_2@l_3 : F'}$$

Hence,

$$\dfrac{\Gamma; H \vdash \langle Q/y\rangle l_2 : \langle Q/y\rangle A \qquad \Gamma; \langle Q/y\rangle A \vdash \langle Q/y\rangle l_3 : F}{\Gamma; H \vdash (\langle Q/y\rangle l_2)@(\langle Q/y\rangle l_3) : F}$$

□

8.4 Correspondence with Pure Type Systems

In this section we establish a logical correspondence between a PTSC given by the sets $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ and the PTS given by the same sets.

Definition 120 (Environments & judgements of PTS) Environments of PTS are lists of pairs such as $(x : T)$ (with T being a PTS-term) and are denoted $\Gamma, \Delta, \Pi, \ldots$. Judgements of PTS are of two kinds: Γ wf, and $\Gamma \vdash t : T$.

Definition 121 (PTS) The derivable judgements of a PTS specified by the sets $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$ are given by the typing rules of Fig. 8.4 and denoted $\Gamma\ \mathsf{wf}_{\mathsf{PTS}}$ and $\Gamma \vdash_{\mathsf{PTS}} t : T$.

PTS satisfy the following properties:

Theorem 217 (Theorems about PTS)

1. *If $\Gamma \vdash_{PTS} t : T$ and $\Gamma \sqsubseteq \Delta$ wf_{PTS} then $\Delta \vdash_{PTS} t : T$ (where the relation $\sqsubseteq$ is defined similarly to that of PTSC, but with β-equivalence).*
2. *If $\Gamma \vdash_{PTS} t : T$ and $\Gamma, y : T, \Delta \vdash_{PTS} u : U$ then $\Gamma, \{^t/_y\}\Delta \vdash_{PTS} \{^t/_y\}u : \{^t/_y\}U$.*
3. *If $\Gamma \vdash_{PTS} t : T$ and $t \longrightarrow_\beta u$ then $\Gamma \vdash_{PTS} u : T$.*

Proof: See e.g. [Bar92]. □

$$\frac{}{[]\ \mathsf{wf}} \qquad \frac{\Gamma \vdash T : s \quad x \notin \mathsf{Dom}(\Gamma)}{\Gamma, x:T\ \mathsf{wf}}$$

$$\frac{\Gamma\ \mathsf{wf} \quad (s,s') \in \mathcal{A}}{\Gamma \vdash s : s'} \qquad \frac{\Gamma \vdash U : s_1 \quad \Gamma, x:U \vdash T : s_2 \quad (s_1,s_2,s_3) \in \mathcal{R}}{\Gamma \vdash \Pi x^U.T : s_3}$$

$$\frac{\Gamma \vdash \Pi x^U.T : s \quad \Gamma, x:U \vdash t : T}{\Gamma \vdash \lambda x^U.t : \Pi x^U.T} \qquad \frac{\Gamma \vdash t : \Pi x^U.T \quad \Gamma \vdash u : U}{\Gamma \vdash t\ u : \{^u/_x\}T}$$

$$\frac{\Gamma\ \mathsf{wf} \quad (x:T) \in \Gamma}{\Gamma \vdash x : T} \qquad \frac{\Gamma \vdash t : U \quad \Gamma \vdash V : s \quad U \longleftrightarrow^*_\beta V}{\Gamma \vdash t : V}$$

Figure 8.6: Typing rules of a PTS

Definition 122 (Encoding of environments) We now extend to environments the encodings between terms of PTS and terms of PTSC:

$$\begin{array}{lcllcl} \mathcal{A}([]) & := & [] & \mathcal{B}([]) & := & [] \\ \mathcal{A}(\Gamma, x:T) & := & \mathcal{A}(\Gamma), x:\mathcal{A}(T) & \mathcal{B}(\Gamma, x:A) & := & \mathcal{B}(\Gamma), x:\mathcal{B}(A) \end{array}$$

Preservation of typing can now be established:

Theorem 218 (Preservation of typing 1)

1. *If* $\Gamma \vdash_{PTS} t:T$ *and* $\Gamma; \mathcal{A}(T) \vdash_{PTSC} l:C$ *then* $\mathcal{A}(\Gamma) \vdash_{PTSC} \mathcal{A}_l(t):C$.
2. *If* $\Gamma \vdash_{PTS} t:T$ *then* $\mathcal{A}(\Gamma) \vdash_{PTSC} \mathcal{A}(t):\mathcal{A}(T)$.
3. *If* Γ wf_{PTS} *then* $\mathcal{A}(\Gamma)$ wf_{PTSC}.

Proof: By induction on derivations:

1. - For the typing rule of the application, we have $t = t_1\ t_2$. The other hypothesis we have is $\mathcal{A}(\Gamma); \mathcal{A}(\{^{t_2}/_x\}U) \vdash_{\mathsf{PTSC}} l:C$.
   Applying the i.h. on the premisses of the typing rule for application we get $\mathcal{A}(\Gamma) \vdash_{\mathsf{PTSC}} \mathcal{A}(t_1):\Pi x^{\mathcal{A}(T)}.\mathcal{A}(U)$ and $\mathcal{A}(\Gamma) \vdash_{\mathsf{PTSC}} \mathcal{A}(t_2):\mathcal{A}(T)$.
   By Lemma 209 we get $\mathcal{A}(\Gamma) \vdash_{\mathsf{PTSC}} \Pi x^{\mathcal{A}(T)}.\mathcal{A}(U):s$ for some s.
   By Lemma 213 we get $\mathcal{A}(\Gamma), x:\mathcal{A}(T) \vdash_{\mathsf{PTSC}} \mathcal{A}(U):s'$ for some s'.

By rule cut_4 we get $\mathcal{A}(\Gamma) \vdash_{\mathsf{PTSC}} \langle \mathcal{A}(t_2)/x \rangle \mathcal{A}(U) : s'$.
By Theorem 206.1 we get $\langle \mathcal{A}(t_2)/x \rangle \mathcal{A}(U) \longleftrightarrow^*_{\mathsf{Bx}} \mathcal{A}(\{^{t_2}\!/_x\} U)$. Hence,

$$\dfrac{\mathcal{A}(\Gamma) \vdash_{\mathsf{PTSC}} \mathcal{A}(t_2) : \mathcal{A}(T) \qquad \dfrac{\mathcal{A}(\Gamma); \mathcal{A}(\{^{t_2}\!/_x\} U) \vdash_{\mathsf{PTSC}} l : C}{\mathcal{A}(\Gamma); \langle \mathcal{A}(t_2)/x \rangle \mathcal{A}(U) \vdash_{\mathsf{PTSC}} l : C}}{\mathcal{A}(\Gamma); \Pi x^{\mathcal{A}(T)}.\mathcal{A}(U) \vdash_{\mathsf{PTSC}} \mathcal{A}(t_2) \cdot l : C}$$

and then we can conclude by applying point 1 of the i.h. on t_1.

- The case of the other rules is straightforward.

2.
   - For the typing rule of the application, we have $t = t_1\ t_2$:
     By point 2 of the i.h. we get again $\mathcal{A}(\Gamma) \vdash_{\mathsf{PTSC}} \mathcal{A}(t_1) : \Pi x^{\mathcal{A}(T)}.\mathcal{A}(U)$ and $\mathcal{A}(\Gamma) \vdash_{\mathsf{PTSC}} \mathcal{A}(t_2) : \mathcal{A}(T)$.
     As for point 1 we get $\mathcal{A}(\Gamma) \vdash_{\mathsf{PTSC}} \langle \mathcal{A}(t_2)/x \rangle \mathcal{A}(U) : s'$.
     By Theorem 206.1 and subject reduction (Theorem 216) we get $\mathcal{A}(\Gamma) \vdash_{\mathsf{PTSC}} \mathcal{A}(\{^{t_2}\!/_x\} U) : s$, from which we can derive $\mathcal{A}(\Gamma); \mathcal{A}(\{^{t_2}\!/_x\} U) \vdash_{\mathsf{PTSC}} [] : \mathcal{A}(\{^{t_2}\!/_x\} U)$ and we can apply the first point.
   - For the axiom, we have $t = x$, with $x : T$ in the environment Γ $\mathsf{wf}_{\mathsf{PTS}}$: Point 3 of the i.h. gives $\mathcal{A}(\Gamma)$ $\mathsf{wf}_{\mathsf{PTSC}}$, so by Lemma 210 and Lemma 211 we get $\mathcal{A}(\Gamma) \vdash_{\mathsf{PTSC}} \mathcal{A}(T) : s$ for some s. We can then derive $\mathcal{A}(\Gamma); \mathcal{A}(T) \vdash_{\mathsf{PTSC}} [] : \mathcal{A}(T)$ and $\mathcal{A}(\Gamma) \vdash_{\mathsf{PTSC}} x\ [] : \mathcal{A}(T)$.
   - The case of the other rules is straightforward.

3. All cases are straightforward.

□

Theorem 219 (Preservation of typing 2)

1. *If* $\Gamma \vdash_{\mathsf{PTSC}} M : A$ *then* $\mathcal{B}(\Gamma) \vdash_{\mathsf{PTS}} \mathcal{B}(M) : \mathcal{B}(A)$

2. *If* $\Gamma; B \vdash_{\mathsf{PTSC}} l : A$ *then* $\mathcal{B}(\Gamma), y : \mathcal{B}(B) \vdash_{\mathsf{PTS}} \mathcal{B}^y(l) : \mathcal{B}(A)$ *for a fresh* y

3. *If* Γ $\mathsf{wf}_{\mathsf{PTSC}}$ *then* $\mathcal{B}(\Gamma)$ $\mathsf{wf}_{\mathsf{PTS}}$

Proof: Straightforward induction on derivations, using Theorem 217. □

8.5 Equivalence of strong normalisation

Theorem 220 *A PTSC given by the sets $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$ is strongly normalising if and only if the PTS given by the same sets is.*

Proof: Assume that the PTSC is strongly normalising, and let us consider a typed term t of the corresponding PTS, i.e. $\Gamma \vdash_{\mathsf{PTS}} t : T$ for some Γ, T. By Theorem 218, $\mathcal{A}(\Gamma) \vdash \mathcal{A}(t) : \mathcal{A}(T)$ so $\mathcal{A}(t) \in \mathsf{SN}^{\mathsf{Bx}}$. Hence, by Theorem 206.1 and Theorem 22, $t \in \mathsf{SN}^{\beta}$.

Now assume that the PTS is strongly normalising and that $\Gamma \vdash M : A$ in the corresponding PTSC. We shall now apply Bloo and Geuvers' technique from [BG99]. By subject reduction, any N such that $M \longrightarrow^{*}_{\mathsf{Bx}} N$ satisfies $\Gamma \vdash N : A$ and any sub-term P (resp. sub-list l) of any such N is also typable. By Theorem 219, for any such P (resp. l), $\mathcal{B}(P)$ (resp. $\mathcal{B}^y(l)$) is typable in the PTS, so it is strongly normalising by assumption and we denote by $\sharp\mathcal{B}(P)$ (resp. $\mathcal{B}^y(l)$) the length of the longest β-reduction sequence reducing it.

We now encode any such P and l into a first-order syntax given by the following infinite signature and its precedence relation:

$$\mathsf{sub}^n(\_,\_) \succ \mathsf{cut}^n(\_,\_) \succ \mathsf{ii}(\_,\_) \succ \mathsf{i}(\_) \succ \star$$

for all integers n. Moreover, we set $\mathsf{sub}^n(\_,\_) \succ \mathsf{cut}^m(\_,\_)$ if $n > m$. The precedence relation is terminating, and the LPO that it induces on the first-order terms is also terminating (Theorem 49). The encoding is given in Fig 8.7.

$\overline{s}$	$:=$	$\star$	
$\overline{\lambda x^A.M}$	$:=$	$\mathsf{ii}(\overline{A}, \overline{M})$	
$\overline{\Pi x^A.M}$	$:=$	$\mathsf{ii}(\overline{A}, \overline{M})$	
$\overline{x\ l}$	$:=$	$\mathsf{i}(\overline{l})$	
$\overline{M\ l}$	$:=$	$\mathsf{cut}^{\sharp\mathcal{B}(M\ l)}(\overline{M}, \overline{l})$	
$\overline{\langle M/x\rangle N}$	$:=$	$\mathsf{sub}^{\sharp\mathcal{B}(\langle M/x\rangle N)}(\overline{M}, \overline{N})$	
$\overline{[]}$	$:=$	$\star$	
$\overline{M \cdot l}$	$:=$	$\mathsf{ii}(\overline{M}, \overline{l})$	
$\overline{l@l'}$	$:=$	$\mathsf{ii}(\overline{l}, \overline{l'})$	
$\overline{\langle M/x\rangle l}$	$:=$	$\mathsf{sub}^{\sharp\mathcal{B}^y(\langle M/x\rangle l)}(\overline{M}, \overline{l})$	where y is fresh

Figure 8.7: Encoding into the first-order syntax

An induction on terms shows that reductions decrease the LPO:

B $\mathsf{cut}^n(\mathsf{ii}(A, M), \mathsf{ii}(N, l)) \gg \mathsf{cut}^{n'}(\mathsf{sub}^m(N, M), l)$
where $n = \sharp\left\{(\lambda x^{\mathcal{B}(A)}.\mathcal{B}(M))\ \mathcal{B}(N)\,/_y\right\}\mathcal{B}^y(l) > \sharp\left\{\{\mathcal{B}(N)/_x\}\mathcal{B}(M)\,/_y\right\}\mathcal{B}^y(l) = n'$
and $n > \sharp\left\{\mathcal{B}(N)/_x\right\}\mathcal{B}(M) = m$.

B1 $\mathsf{cut}^n(M,\star) \gg M$

B2 $\mathsf{cut}^n(\mathsf{i}(l),l') \gg \mathsf{i}(\mathsf{ii}(l,l'))$

B3 $\mathsf{cut}^p(\mathsf{cut}^n(M,l),l') \gg \mathsf{cut}^p(M,\mathsf{ii}(l,l'))$
where $p = \sharp\left\{\left\{{}^{\mathcal{B}(M)}\!/_{z}\right\}{}^{\mathcal{B}^z(l)}\!/_{y}\right\}\mathcal{B}^y(l') = \sharp\left\{{}^{\mathcal{B}(M)}\!/_{z}\right\}\left\{{}^{\mathcal{B}^z(l)}\!/_{y}\right\}\mathcal{B}^y(l')$.

A1 $\mathsf{ii}(\mathsf{ii}(M,l'),l) \gg \mathsf{ii}(M,\mathsf{ii}(l',l))$

A2 $\mathsf{ii}(\star,l) \gg l$

A3 $\mathsf{ii}(\mathsf{ii}(l,l'),l'') \gg \mathsf{ii}(l,\mathsf{ii}(l',l''))$

C1 $\mathsf{sub}^p(P,\mathsf{ii}(A,M)) \gg \mathsf{ii}(\mathsf{sub}^{p_1}(P,A),\mathsf{sub}^{p_2}(P,M))$
where $p = \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}(\lambda x^A.M) \geq \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}(A) = p_1$
and $p \geq \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}(M) = p_2$.

C2 $\mathsf{sub}^p(P,\mathsf{i}(l)) \gg \mathsf{cut}^p(P,\mathsf{sub}^{p'}(P,l))$
where $p = \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}(y\ l) \geq \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}^z(l) = p'$.

C3 $\mathsf{sub}^p(P,\mathsf{i}(l)) \gg \mathsf{i}(\mathsf{sub}^p(P,l))$
where $p = \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}(x\ l) = \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}^z(l)$.

C4 $\mathsf{sub}^p(P,\mathsf{cut}^n(M,l)) \gg \mathsf{cut}^p(\mathsf{sub}^{p_1}(P,M),\mathsf{sub}^{p_2}(P,l))$
where $p = \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}(M\ l) \geq \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}(M) = p_1$
and $p \geq \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}^z(l) = p_2$.

C5 $\mathsf{sub}^p(P,\mathsf{ii}(A,B)) \gg \mathsf{ii}(\mathsf{sub}^{p_1}(P,A),\mathsf{sub}^{p_2}(P,B))$
where $p = \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}(\Pi x^A.B) \geq \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}(A) = p_1$
and $p \geq \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}(B) = p_2$.

C6 $\mathsf{sub}^n(P,\star) \gg \star$

D1 $\mathsf{sub}^n(P,\star) \gg \star$

D2 $\mathsf{sub}^p(P,(\mathsf{ii}(M,l))) \gg \mathsf{ii}((\mathsf{sub}^{p_1}(P,M)),(\mathsf{sub}^{p_2}(P,l)))$
where $p = \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}^z(N \cdot l) \geq \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}(N) = p_1$
and $p \geq \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}^{z'}(l) = p_2$.

D3 $\mathsf{sub}^n(P,\mathsf{ii}(l,l')) \gg \mathsf{ii}(\mathsf{sub}^{p_1}(P,l),\mathsf{sub}^{p_2}(P,l'))$
where $p = \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}^z(l@l') \geq \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}^z(l) = p_1$
and $p \geq \sharp\left\{{}^{\mathcal{B}(P)}\!/_{y}\right\}\mathcal{B}^{z'}(l') = p_2$.

□

Examples of strongly normalising PTS are the eight corners of *Barendregt's Cube* [Bar92], the collection of PTS given by the following sets:

$$\begin{array}{rrcl} & \mathcal{S} & := & \{\Box_0, \Box_1\} \\ & \mathcal{A} & := & \{(\Box_0, \Box_1)\} \\ \{(\Box_0, \Box_0, \Box_0)\} \subseteq & \mathcal{R} & \subseteq & \{(\Box_0, \Box_0, \Box_0), (\Box_0, \Box_1, \Box_1), (\Box_1, \Box_0, \Box_0), (\Box_1, \Box_1, \Box_1)\} \end{array}$$

The three dimensions of the Cube correspond to the properties $(\Box_0, \Box_1, \Box_1) \in \mathcal{R}$ (dependent types), $(\Box_1, \Box_0, \Box_0) \in \mathcal{R}$ (polymorphism), and $(\Box_1, \Box_1, \Box_1) \in \mathcal{R}$ (type constructors), which can be combined in eight different ways.

Among these systems are the simply-typed λ-calculus, system F, system F_ω, their respective versions with dependent types such as the *Calculus of Constructions* [CH88] (CoC), which combines the three dimensions with $\mathcal{R} = \{(\Box_0, \Box_0, \Box_0), (\Box_0, \Box_1, \Box_1), (\Box_1, \Box_0, \Box_0), (\Box_1, \Box_1, \Box_1)\}$.

The latter can be extended into the PTS given by following sets, which is still strongly normalising:

$$\begin{array}{rcl} \mathcal{S} & := & \{\Box_0, \Box_1, \ldots, \Box_i, \ldots\} \\ \mathcal{A} & := & \{(\Box_i, \Box_{i+1}) |\ i \in \mathbb{N}\} \\ \mathcal{R} & \subseteq & \{(\Box_i, \Box_j, \Box_{max(i,j)}) |\ i, j \in \mathbb{N}\} \cup \{(\Box_i, \Box_0) |\ i \in \mathbb{N}\} \end{array}$$

This is the *Calculus of Constructions with Universes* [Luo90], on which the proof-assistant Coq is based [Coq] (but it also uses inductive types and local definitions).

For each of the above PTS we now have a strongly normalising and logically equivalent sequent calculus, namely the PTSC given by the same sets $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$, which can be used for proof-search as we shall see in the next chapter. We thus have for instance the *Sequent Calculus of Constructions* and the *Sequent Calculus of Constructions with Universes*.

Conclusion

We have defined a parameterised formalism that gives a sequent calculus for each PTS. It comprises a syntax, a rewrite system and typing rules. In contrast to previous work, the syntax of both types and proof-terms of PTSC is in a sequent-calculus style, thus avoiding the use of implicit or explicit conversions to natural deduction [GR03b, PD98].

A strong correspondence with natural deduction has been established (in particular, regarding both the logic and the strong normalisation), and for instance we derive from it the confluence of each PTSC. We can give as examples the corners of Barendregt's Cube.

The sequent calculus that we have used is based on LJT because of its well-known connections with natural deduction, in particular it is permutation-free, in that its x-normal forms are in bijection with λ-terms, with the same equational theory on them.

However this raises the question of whether a type theory can be built on a Gentzen-style sequent calculus such as G3ii. In that case, various options are possible for the equational theory that is used in the conversion rules: we can either include therein the permutations of those sequent calculus proof-terms that correspond to identical λ-terms, or not. Draft work in my research has shown that both options are possible, leading to different (probably syntactic and uninteresting) models. However we do not include the Gentzen-style approach in this dissertation because it is highly technical, while less inelegant alternatives are to be sought, maybe using Espirito Santo's approach [EFP06].

Also, it would be interesting to have direct proofs of strong normalisation for particular PTSC, such as the Sequent Calculus of Constructions. This could be based on [Kik04a], which adapts Tait's [Tai75] method to the particular PTSC corresponding to propositional logic.

Further work includes the investigation of inductive types in sequent calculus, such as those used in Coq. Finally, sequent calculus is also more elegant than natural deduction to express classical logic, so it would be interesting to build classical Pure Type Sequent Calculi.

Chapter 9

Variants of PTSC

In this chapter we present variants of PTSC. We are especially interested in the notion of computation in PTSC which is threefold:

- Execution of programs/normalisation of terms.
- Proof search.
- Type inference.

In the complex framework of type theory, proof search and type inference are often called proof synthesis and type synthesis.

Proof synthesis is the process that takes an environment Γ and a type A as inputs and produces a (normal) term M such that $\Gamma \vdash M : A$. The recursive calls of the process will also take, as inputs, an environment Γ and two types B and A, and produce, as an output, a list l such that $\Gamma; B \vdash l : A$. In type theory, proof synthesis also *enumerates* all such terms M: Indeed, with the dependencies created by Π-types, such a proof-term M produced by a recursive call might affect the inputs of another recursive call, which could fail because of this particular M. In this case, the proof synthesis process has to backtrack and find another term M'. In many type theories, proof synthesis is in fact undecidable; however it is still interesting to have a semi-decision procedure that will enumerate all proof-terms of a given type in a given environment. Here we claim that such a procedure can be defined simply as the root-first application of typing rules in sequent calculus, thus showing one of main motives of developing PTSC. An inference system that defines such a procedure by root-first application of its rules is *syntax-directed*: the rules must be *directed* by the syntax of the type or the types that are the inputs of the procedure. In the typing system of PTSC (Fig. 8.5), what makes the rules non-syntax-directed is the conversion rules, which could apply at any point, with a type to invent.

Section 9.1 presents a system optimised for proof synthesis that integrates the conversion rules into the typing rules of term constructions, in a way similar

to the *Constructive engine* in natural deduction [Hue89, vBJMP94]. However the latter is used for type synthesis, whose inputs and outputs are different from those of proof synthesis and lead to a different way to integrate conversion rules, as discussed further. In section 9.1 the version of PTSC optimised for proof synthesis is proved equivalent to the PTSC, i.e. sound and complete in a strong sense that we shall make precise. We illustrate the use of the proof synthesis version of PTSC with an example.

To the root-first application of our optimised rules we can then compare some basic proof search tactics of proof assistants based on PTS, most interestingly the tactics Apply in Coq or Refine in Lego. As mentioned in the introduction of Chapter 8 and noticed by [McK97] long before, these are much closer (in spirit) to the left-introduction of Π-types (as in rule Πl below) than to elimination rules of natural deduction.

$$\frac{\Gamma \vdash M : A \quad \Gamma; \langle M/x \rangle B \vdash l : C}{\Gamma; \Pi x^A.B \vdash M \cdot l : C}\ \Pi\mathsf{l}$$

However these tactics are also able to postpone the investigation of the first premiss of the rule and start investigating the second, using unification constraints instead of simple conversions. Moreover, [Dow93] shows that in type theories such as the Calculus of Constructions [CH88], the process of proof synthesis merges with that of unification [Hue76].

While [McK97] investigates the aforementioned postponement of the resolution of the first premiss by using Lego's local definition mechanism [LP92], we show in section 9.2 that the sequent calculus approach is also convenient to express proof synthesis algorithms such as those of [Dow93, Mun01]. This is done by extending the syntax of PTSC with higher-order variables, which have the same role as meta-variables added to the object-level syntax, in that they represent unknown proof-terms yet to be found. These variables are used to postpone recursive calls of the proof synthesis procedure as well as in the unification constraints. Further development of this part seems to represent some of the most promising directions for future work.

Type synthesis is the process that takes as an input an environment Γ and a term M and that produces as an output a type A such that $\Gamma \vdash M : A$. The recursive calls of the process will also take, as inputs, an environment Γ, a type B and a list l, and produce, as an output, a type A such that $\Gamma; B \vdash l : A$. They will also take an environment Γ as an input and answer whether or not it is well-formed. These three kinds of input/output behaviour naturally correspond to the three kinds of judgements as in Fig. 8.5. Again, an inference system that defines such a procedure by root-first application of its rules is also *syntax-directed*, but this time, since the inputs and outputs are different from those of proof synthesis, the rules are *directed* by the syntax of the term which is the input (as well as using the information given in the environment). Note that in the typing system of PTSC (Fig. 8.5), what makes the rules non-syntax-directed is:

- the conversion rules, which could apply at any point, with a type A to invent,

- the shape of the typing rules for explicit substitutions: type synthesis would need to extract, from an environment Π and a term M, an environment $\Gamma, x:A, \Delta$ such that $\Gamma, \langle M/x\rangle\Delta \sqsubseteq \Pi$.

The latter point is as hard as extracting from a substituted term the original (implicit) substitution that formed it; there are just too many possibilities. This is naturally connected to the inelegance of our typing rules cut_2 and cut_4, and probably the way to make type synthesis work for explicit substitutions is to have a typing system in which Subject Expansion holds for xsubst: if $M \longrightarrow_{\mathsf{xsubst}} N$ and $\Gamma \vdash N:A$ then $\Gamma \vdash M:A$, possibly under some particular conditions. Then, since xsubst terminates (it is in fact confluent), the process of type synthesis could produce a type for the xsubst-normal form of its input term, and this would also be a valid type for the input term. Kervarc and Lescanne [KL04] develop a version of PTS along these lines, in natural deduction but with a form of explicit substitution similar to ours. Also in natural deduction is the Calculus of Constructions with explicit substitutions of [Mun01], but in quite a different setting that uses de Bruijn indices [dB72] in the style of $\lambda\sigma$ [ACCL91, CHL96]. Explicit substitutions are of a more advanced/complex kind, forming a syntactic category of their own, with typing rules that depart from a simple cut in sequent calculus.

In this dissertation we give in section 9.3 a version of PTSC, called $\mathsf{PTSC}_{\mathsf{imp}}$, that uses implicit substitutions rather than explicit ones. $\mathsf{PTSC}_{\mathsf{imp}}$ will also be easier to convert into a version that uses de Bruijn indices.

An inference system for type synthesis can then be defined in section 9.4, based on $\mathsf{PTSC}_{\mathsf{imp}}$. The former point that made the typing system of PTSC (Fig. 8.5) non-syntax-directed for type synthesis was the presence of the conversion rules, so the inference system of section 9.4 integrates them into the typing rules of term constructions. This time, such an integration is exactly the counterpart in sequent calculus of the constructive engine in natural deduction [Hue89, vBJMP94], which serves the same purpose of type synthesis. Our system gives the opportunity to discuss whether normal forms can be typed without cut-rules and raise, in our framework of sequent calculus, well-known issues related to PTS, such as *Expansion Postponement* [Pol92, vBJMP94, Pol98, GR03a].

Finally, in section 9.5 we develop implementation-friendly versions of PTSC using de Bruijn indices and corresponding to PTSC with implicit substitutions in two variants: the version optimised for proof synthesis (but not with the extension of higher-order variables, left as future work), and the version optimised for type synthesis. It appears that [KR02] developed a version of PTS with de Bruijn indices, using the very same machinery (same style of indices, reduction...). In other word, section 9.5 turns out to be the PTSC version of [KR02].

9.1 Proof synthesis

In contrast to propositional logic where cut is an admissible rule of sequent calculus, terms of PTSC in normal form may need a cut-rule in their typing derivation. For instance in the rule ΠI, a type which is not normalised ($\langle M/x\rangle B$) must appear in the stoup of the third premiss, so that cuts might be needed to type it inside the derivation.

In this section we present a system for proof synthesis that avoids all cuts, is complete and is sound provided that types are checked independently. In proof synthesis, the inputs are an environment Γ and a type A, henceforth called *goal*, and the output is a term M such that $\Gamma \vdash M : A$. When we look for a list, the type in the stoup is also an input. The inference rules now need to be directed by the shape of the goal (or of the type in the stoup), and the proof synthesis system (PS, for short) can be obtained by optimising the use of the conversion rules as shown in Fig. 9.1. The incorporation of the conversion rules into the other rules is similar to that of the *Constructive Engine* in natural deduction [Hue89, vBJMP94]; however the latter was designed for type synthesis, for which the inputs and outputs are not the same as in proof synthesis, as mentioned in the introduction.

$$\frac{A \longleftrightarrow^*_{\mathsf{Bx}} A'}{\Gamma; A \vdash_{\mathsf{PS}} [] : A'}\ \mathsf{ax}_{\mathsf{PS}} \qquad \frac{D \longrightarrow^*_{\mathsf{Bx}} \Pi x^A.B \quad \Gamma \vdash_{\mathsf{PS}} M : A \quad \Gamma; \langle M/x\rangle B \vdash_{\mathsf{PS}} l : C}{\Gamma; D \vdash_{\mathsf{PS}} M \cdot l : C}\ \Pi\mathsf{l}_{\mathsf{PS}}$$

$$\frac{C \longrightarrow^*_{\mathsf{Bx}} s_3 \quad (s_1, s_2, s_3) \in R \quad \Gamma \vdash_{\mathsf{PS}} A : s_1 \quad \Gamma, x : A \vdash_{\mathsf{PS}} B : s_2}{\Gamma \vdash_{\mathsf{PS}} \Pi x^A.B : C}\ \Pi\mathsf{wf}_{\mathsf{PS}}$$

$$\frac{C \longrightarrow^*_{\mathsf{Bx}} s_2 \quad (s_1, s_2) \in \mathcal{A}}{\Gamma \vdash_{\mathsf{PS}} s : C}\ \mathsf{sorted}_{\mathsf{PS}} \qquad \frac{(x : A) \in \Gamma \quad \Gamma; A \vdash_{\mathsf{PS}} l : B}{\Gamma \vdash_{\mathsf{PS}} x\ l : B}\ \mathsf{select}_x$$

$$\frac{C \longrightarrow^*_{\mathsf{Bx}} \Pi x^A.B \quad \Gamma, x : A \vdash_{\mathsf{PS}} M : B}{\Gamma \vdash_{\mathsf{PS}} \lambda x^A.M : C}\ \Pi\mathsf{r}_{\mathsf{PS}}$$

Figure 9.1: System PS

We now prove soundness, and for that it is useful to define the following notion:

Definition 123 A term (or a list) is a *quasi normal form* if all its redexes are within type annotations of λ-abstractions, e.g. A in $\lambda x^A.M$.

System PS is *sound* in the following sense:

Theorem 221 (Soundness)

1. *Provided* $\Gamma \vdash_{\mathsf{PTSC}} A:s$ *or* $A \longleftrightarrow^*_{\mathsf{Bx}} s$ *and* Γ $\mathsf{wf}_{\mathsf{PTSC}}$,
   if $\Gamma \vdash_{\mathsf{PS}} M:A$ *then* $\Gamma \vdash_{\mathsf{PTSC}} M:A$ *and* M *is a quasi-normal form.*

2. *Provided* $\Gamma \vdash_{\mathsf{PTSC}} A:s_A$ *and* $\Gamma \vdash_{\mathsf{PTSC}} B:s_B$,
   if $\Gamma; A \vdash_{\mathsf{PS}} l:B$ *then* $\Gamma; A \vdash_{\mathsf{PTSC}} l:B$ *and* l *is a quasi-normal form.*

Proof: Straightforward induction on typing derivations. Note that the type-checking proviso is verified every time we need the induction hypothesis, it is an invariant of the system. □

Notice than in PS there are *no* cut-rules. Indeed, even though, in the original typing system, cuts are required in typing derivations of normal forms, they only occur to check that types are themselves typed. Here we have removed these type-checking constraints, relaxing the system, because types are the input of proof synthesis, and they would be checked before starting the search, which is the spirit of the type-checking proviso in the soundness theorem. When recovering a full derivation tree from a PS one by the soundness theorem, cuts might be introduced at any point, coming from the derivation of this type-checking proviso.

Lemma 222 *Suppose* $A \longleftrightarrow^*_{\mathsf{Bx}} A'$ *and* $B \longleftrightarrow^*_{\mathsf{Bx}} B'$.

1. *If* $\Gamma \vdash_{\mathsf{PS}} M:A$ *then* $\Gamma \vdash_{\mathsf{PS}} M:A'$.

2. *If* $\Gamma; B \vdash_{\mathsf{PS}} l:A$ *then* $\Gamma; B' \vdash_{\mathsf{PS}} l:A'$.

Proof: Straightforward induction on typing derivations. □

We can now prove that PS is *complete* in the following sense:

Theorem 223 (Completeness) [1]

1. *If* $\Gamma \vdash_{\mathsf{PTSC}} M:A$ *and* M *is a quasi-normal form, then* $\Gamma \vdash_{\mathsf{PS}} M:A$.

2. *If* $\Gamma; A \vdash_{\mathsf{PTSC}} l:B$ *and* l *is a quasi-normal form, then* $\Gamma; A \vdash_{\mathsf{PS}} l:B$.

Proof: Straightforward induction on typing derivations, using Lemma 222. □

[1] Note that neither Theorem 221 nor Theorem 223 relies on the unsolved problem of *expansion postponement* that we mention in section 9.4 and that occurs in type-checking premisses when conversion rules are restricted in particular ways. Indeed, PS *does not* check types, and expansions can be introduced together with cuts by the type-checking proviso of the soundness theorem.

In order to state the soundness and completeness theorems with normal forms instead of quasi-normal forms, we would need to require A to be a normal form in rule $\Pi\mathsf{r}_{\mathsf{PS}}$. In general, reaching such a normal form would require the strong normalisation of the PTSC. However, its existence for proving completeness is given in any case by the hypothesis, which *provides* the normal form M with all the type annotations of its λ-abstractions, also in normal form. This would be a minor variant.

Basic proof synthesis can be done in PS simply by

- reducing the goal, or the type in the stoup;
- depending on its shape, trying to apply one of the inference rules bottom-up,
- recursively call the process on the new goals (called *sub-goals*) corresponding to each premisses.

Indeed, the rules are *syntax-directed* for proof synthesis, i.e. they are *directed* by the syntax of the goal or the type in the stoup. However, some degree of non-determinism is expected in proof synthesis, often called *"don't care"* non-determinism in the case of the choice to apply an invertible rule and *"don't know"* non-determinism when the choice identifies a potential point of back-track.

Non-determinism is already present in natural deduction, but the sequent calculus version elegantly identifies where it occurs:

- The choice of a variable x for applying rule select_x, knowing only Γ and B (this corresponds in natural deduction to the choice of the head-variable of the proof-term). Not every variable of the environment will work, since the type in the stoup will eventually have to be unified with the goal, so we might need to back-track; this is a "don't know" non-determinism.
- When the goal reduces to a Π-type, there is an overlap between rules $\Pi\mathsf{r}$ and select_x; similarly, when the type in the stoup reduces to a Π-type, there is an overlap between rules $\Pi\mathsf{l}$ and ax. This is a "don't know" non-determinism unless we consider η-conversion in our notion of convertibility. In that case we could also restrict select_x is to the case when the goal does not reduce to a Π-type (and sequents with stoups never have such a goal), and both overlaps disappear. This corresponds to looking only for η-long normal forms in natural deduction. This restriction also brings the derivations in LJT (and in our PTSC) closer to the notion of uniform proofs. Further work includes the addition of η to the notion of conversion in PTSC.
- When the goal reduces to a sort s, three rules can be applied (in contrast to the first two points, this source of non-determinism does not already appear in the propositional case). This is also a "don't know" non-determinism.

We now give the example of a derivation in PS. We consider the PTSC equivalent to system F, i.e. the one given by the sets:
$\mathcal{S} = \{\star, \Box\}$, $\mathcal{A} = \{(\star, \Box)\}$, and $\mathcal{R} = \{(\star, \star), (\Box, \star)\}$.

For brevity we omit the types on λ-abstractions, we abbreviate $x\ []$ as x for any variable x and simplify $\langle N/x\rangle P$ as P when $x \notin \mathsf{FV}(P)$. We also write $A \wedge B$ for $\Pi Q^{\star}.(A \to (B \to Q)) \to Q$. Trying to find a term M such that $A : \star, B : \star \vdash M : (A \wedge B) \to (B \wedge A)$, we get the PS-derivation below:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\pi_B}{\Gamma \vdash_{\mathsf{PS}} N_B : B} \quad \dfrac{\dfrac{\pi_A}{\Gamma \vdash_{\mathsf{PS}} N_A : A} \quad \dfrac{}{\Gamma; Q \vdash_{\mathsf{PS}} [] : Q}\,\mathsf{ax}}{\Gamma; A \to Q \vdash_{\mathsf{PS}} N_A \cdot [] : Q}\,\Pi\mathsf{l}}{\Gamma; B \to (A \to Q) \vdash_{\mathsf{PS}} N_B \cdot N_A \cdot [] : Q}\,\Pi\mathsf{l}}{\Gamma \vdash_{\mathsf{PS}} y\ N_B \cdot N_A \cdot [] : Q}\,\mathsf{select}_y}{A : \star, B : \star \vdash_{\mathsf{PS}} \lambda x.\lambda Q.\lambda y.y\ N_B \cdot N_A \cdot [] : (A \wedge B) \to (B \wedge A)}\,\Pi\mathsf{r}$$

where $\Gamma = A : \star, B : \star, x : A \wedge B, Q : \star, y : B \to (A \to Q)$, and π_A is the following derivation ($N_A = x\ A \cdot (\lambda x'.\lambda y'.x') \cdot []$):

$$\dfrac{\dfrac{\dfrac{}{\Gamma; \star \vdash_{\mathsf{PS}} [] : \star}}{\Gamma \vdash_{\mathsf{PS}} A : \star} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, x' : A, y' : B; A \vdash_{\mathsf{PS}} [] : A}}{\Gamma, x' : A, y' : B \vdash_{\mathsf{PS}} x' : A}}{\Gamma \vdash_{\mathsf{PS}} \lambda x'.\lambda y'.x' : A \to (B \to A)} \quad \dfrac{}{\Gamma; A \vdash_{\mathsf{PS}} [] : A}}{\Gamma; \langle A/Q\rangle(A \to (B \to Q)) \to Q \vdash_{\mathsf{PS}} (\lambda x'.\lambda y'.x') \cdot [] : A}}{\dfrac{\Gamma; A \wedge B \vdash_{\mathsf{PS}} A \cdot (\lambda x'.\lambda y'.x') \cdot [] : A}{\Gamma \vdash_{\mathsf{PS}} x\ A \cdot (\lambda x'.\lambda y'.x') \cdot [] : A}}$$

and π_B is the derivation similar to π_A ($N_B = x\ B \cdot (\lambda x'.\lambda y'.y') \cdot []$) with conclusion $\Gamma \vdash_{\mathsf{PS}} x\ B \cdot (\lambda x'.\lambda y'.y') \cdot [] : B$.

This example shows how the non-determinism of proof synthesis is sometimes quite constrained by the need to eventually unify the type in the stoup with the goal. For instance in π_A (resp. π_B), solving $\Gamma \vdash Q : \star$ must produce A (resp. B) otherwise the resolution of the right-hand side branch fails. Indeed, the dependency created by a Π-type forces the resolution of the two premisses of rule $\Pi\mathsf{l}$ to be sequentialised in a way that might reveal inefficient: the proof-term produced for the first premiss, selected among others at random, might well lead to the failure of solving the second premiss, leading to endless backtracking.

Hence, there is much to gain in postponing the resolution of the first premiss and trying to solve the second with incomplete inputs (in our example, not knowing Q). This might not terminate with success or failure but this will send back useful constraints that will help the resolution of the first premiss with the

right proof-term. "Helping" could just be giving some information to orient and speed-up the search for the right proof-term, but it could well define it completely (saving numerous attempts with proof-terms that will lead to failure). Unsurprisingly, these constraints are produced by the axiom rule as *unification* constraints, in our example the constraint $Q = A$ for π_A and $Q = B$ for π_B, which in both cases define Q entirely indeed.

This is what happens in Coq [Coq], whose proof-search tactic `apply x` can be decomposed into the bottom-up application of select_x followed by a series of bottom-up applications of $\Pi\mathsf{l}$ and finally ax, but it either postpones the resolution of sub-goals or automatically solves them from the unification attempt, often avoiding obvious back-tracking.

In the next section we investigate how we can express this behaviour in a sequent calculus.

9.2 Higher-order variables for proof enumeration

In order to mimic even more closely such a tactic as `apply x` of Coq, we tackle in this section the issue of delaying the resolution of sub-goals. Where such a resolution should have produced a proof-term of the correct type, we now offer the possibility of *"cheating"* by producing something similar to a meta-variable that represents a term yet to be found and that can be manipulated as one.

By thus extending PTSC, we can go further than accounting for a proof-search tactic such as `apply x` of Coq and express a sound and complete algorithm for type inhabitant enumeration. This is similar to Dowek's [Dow93] and Muñoz's [Mun01] in natural deduction, but the novelty here is that the algorithm is simply the root-first construction of derivation trees in sequent calculus.

In fact, instead of meta-variables, we use the possibility, offered by the formalism of HOC, of *higher-order variables*. This is quite similar to CRS [Klo80], in that unknown terms are represented with (meta/higher-order) variables *applied* to the series of (term-)variables that could occur freely in those terms, e.g. $\alpha(x, y)$ to represent a term M in which x and y could be free. These arguments can later be instantiated, so that $\alpha(N, P)$ will represent $\left\{{}^{N,P}\!/_{x,y}\right\}M$. In other words, a (meta/higher-order) variable on its own represents something closed, e.g. $x.y.M$ with $\mathsf{FV}(M) \subseteq \{x, y\}$, using the binding mechanism of HOC (or CRS).

This kind of meta-variable differs from that of in [Mun01], which is rather in the style of ERS [Kha90] where the variables that could occur freely in the unknown term are not specified explicitly. The drawback of our approach is that we have to *know* in advance the free variables that might occur freely in the unknown term, but in a typed setting such as proof synthesis these are actually the variables declared in the environment. Moreover, although specifying explicitly the variables that could occur freely in an unknown term might seem heavy, it actually avoids the well-known problem of non-confluence of reduction when

terms contain meta-variables in the style of [Mun01]. The solution in [Mun01] has the drawback of not simulating β-reduction (but the reductions in [Mun01] reach the expected normal forms). The machinery developed in Chapter 5 might allow a similar solution but with simulation of β-reduction, however it would be very heavy and here we simply prefer avoiding the problem by using the CRS-approach to meta-variables.

Definition 124 (Open terms) The grammar of open terms and open lists is defined as follows:

$$\begin{array}{rl} M, N, A, B & ::= \Pi x^A.B \mid \lambda x^A.M \mid s \mid x\ l \mid M\ l \mid \langle M/x\rangle N \mid \alpha(M_1, \ldots, M_n) \\ l, l' & ::= [] \mid M \cdot l \mid l@l' \mid \langle M/x\rangle l \mid \beta(M_1, \ldots, M_n) \end{array}$$

where α, β range over variables of order 1 and arity n, for all n, respectively producing terms and lists.

Terms and lists without these variables (i.e. terms and lists of Definition 114) are now called *ground terms* and *ground lists*, respectively.

Definition 125 (Extension of x-reduction) Owing to the presence of higher-order variables, we have to extend system x with the following rules:

$$\begin{array}{rcl} \langle P/y\rangle\alpha(M_1, \ldots, M_n) & \longrightarrow & \alpha(\langle P/y\rangle M_1, \ldots, \langle P/y\rangle M_n) \\ \langle P/y\rangle\beta(M_1, \ldots, M_n) & \longrightarrow & \beta(\langle P/y\rangle M_1, \ldots, \langle P/y\rangle M_n) \end{array}$$

This extended system is called x′.

Conjecture 224 (Confluence of Bx′) *System Bx′ is confluent.*

Proof: Considering higher-order variables in the style of CRS [Klo80] avoids the usual problem of non-confluence coming from the critical pair between B and C4 which generate the two terms $\langle N/x\rangle\langle P/y\rangle M$ and $\langle\langle N/x\rangle P/y\rangle\langle N/x\rangle M$. Indeed, with ERS-style meta-variables these two terms need not reduce to a common term, but with the CRS-approach they now can with the two new rules of x′. The other critical pairs between Bs and C4, as well as the critical pairs between As and D3, B and B3, As and A3 are also easily joined. The last critical pair is between B3 and itself (or B2), and for that rule A3 is needed, while it was only there for convenience when all terms were ground.

Joinability of critical pairs is not sufficient to derive confluence of the (higher-order) rewrite system, but it gives confidence that a proof can be found. In fact, it seems that the proof technique for Corollary 207 (confluence of PTSC with ground terms) can be adapted to the case with open terms: to derive the confluence result from that of PTS using a reflection we only need to find a good encoding of our higher-order variables in PTS (this seems to work precisely because we use CRS-style meta/higher-order variables). The details remain to be checked. □

Definition 126 (Open environment) Open environments are defined like environments (Definition 118), but with open terms instead of ground terms.

We now keep track of a new environment that contains the sub-goals that are left to be proved:

Definition 127 (Goal environment)

- A *goal environment* Σ is a list of:
  - Triples of the form $\Gamma \vdash \alpha : A$, called *(term-)goals*, where A is an open term, Γ is an open environment, and α is a variable of order 1 and arity $|\Gamma|$.
  - 4-tuples of the form $\Gamma; B \vdash \beta : A$, called *(list-)goals*, where A and B are open terms, Γ is an open environment, and β is a variable of order 1 and arity $|\Gamma|$.
  - Triples of the form $A \overset{\Gamma}{\longleftrightarrow} B$, called *constraints*, where Γ is an open environment and A and B are open terms.
- A constraint is *solved* if it is of the form $A \overset{\Gamma}{\longleftrightarrow} B$ where A and B are ground and $A \longleftrightarrow^*_{\mathsf{Bx}} B$.
- A goal environment is *solved* if it has no goals and only solved constraints.

Definition 128 (An inference system for proof enumeration)
The inference rules for proof synthesis manipulate three kinds of judgement:

- The first two are of the form $\Gamma \vdash M : A \mid \Sigma$ and $\Gamma; B \vdash M : A \mid \Sigma$. These have the same intuitive meaning as the corresponding judgements in system PS, but note the extra goal environment Σ, which represents the list of sub-goals and constraints that have been produced by proof-synthesis and that are left to solve and satisfy, respectively. Hence, the inputs of proof synthesis are Γ and A (and B for the second kind of judgement) and the outputs are M (or l) and Σ. Judgements of PS are in fact particular cases of these judgements with Σ being always solved.
- The third kind of judgement is of the form $\Sigma \Longrightarrow \sigma$, where
  - Σ is the list of goals to solve, together with the constraints that the solutions must satisfy, and
  - σ is a substitution, i.e. a finite function from higher-order variables to higher-order terms and lists (by higher-order terms and lists is meant terms and lists under a series of HOC bindings on all their potential free variables, e.g. $x.y.M$ if $\mathsf{FV}(M) \subseteq \{x, y\}$).

$$\frac{\Gamma = x_1 : A_1, \ldots, x_n : A_n}{\Gamma; D \vdash \beta(x_1\ [], \ldots, x_n\ []) : C \mid (\Gamma; D \vdash \beta : C)}$$

$$\frac{}{\Gamma; D \vdash [] : C \mid D \overset{\Gamma}{\longleftrightarrow} C}$$

$$\frac{D \longrightarrow^*_{\mathsf{Bx}} \Pi x^A.B \quad \Gamma \vdash M : A \mid \Sigma_1 \quad \Gamma; \langle M/x \rangle B \vdash l : C \mid \Sigma_2}{\Gamma; D \vdash M \cdot l : C \mid \Sigma_1, \Sigma_2}$$

$$\frac{\Gamma = x_1 : A_1, \ldots, x_n : A_n}{\Gamma \vdash \alpha(x_1\ [], \ldots, x_n\ []) : C \mid (\Gamma \vdash \alpha : C)}$$

$$\frac{C \longrightarrow^*_{\mathsf{Bx}} s \quad (s', s) \in \mathcal{A}}{\Gamma \vdash s' : C \mid []}$$

$$\frac{C \longrightarrow^*_{\mathsf{Bx}} s \quad (s_1, s_2, s) \in \mathcal{R} \quad \Gamma \vdash A : s_1 \mid \Sigma_1 \quad \Gamma, x : A \vdash B : s_2 \mid \Sigma_2}{\Gamma \vdash \Pi x^A.B : C \mid \Sigma_1, \Sigma_2}$$

$$\frac{(x : A) \in \Gamma \quad \Gamma; A \vdash l : C \mid \Sigma'}{\Gamma \vdash x\ l : C \mid \Sigma'}$$

$$\frac{C \longrightarrow^*_{\mathsf{Bx}} \Pi x^A.B \quad \Gamma, x : A \vdash M : B \mid \Sigma'}{\Gamma \vdash \lambda x^A.M : C \mid \Sigma'}$$

$$\frac{\Gamma; B \vdash l : A \mid \Sigma'' \qquad \Sigma, \Sigma'', \left\{ \mathsf{Dom}(\Gamma).l \diagup \beta \right\} \Sigma' \Longrightarrow \sigma}{\Sigma, (\Gamma; B \vdash \beta : A), \Sigma' \Longrightarrow \sigma, \beta \mapsto \mathsf{Dom}(\Gamma).\sigma(l)}$$

$$\frac{\Gamma \vdash M : A \mid \Sigma'' \qquad \Sigma, \Sigma'', \left\{ \mathsf{Dom}(\Gamma).M \diagup \alpha \right\} \Sigma' \Longrightarrow \sigma}{\Sigma, (\Gamma \vdash \alpha : A), \Sigma' \Longrightarrow \sigma, \alpha \mapsto \mathsf{Dom}(\Gamma).\sigma(M)}$$

$$\frac{\Sigma \text{ is solved}}{\Sigma \Longrightarrow \emptyset}$$

Figure 9.2: Proof-term enumeration

Σ is the input of proof synthesis and σ is meant to be its solution, i.e. the output. We write $\sigma(M)$ (resp. $\sigma(l)$) for $\left\{{}^{M_1,\ldots,M_n}\!/_{\alpha_1,\ldots,\alpha_n}\right\}M$ (resp. $\left\{{}^{M_1,\ldots,M_n}\!/_{\alpha_1,\ldots,\alpha_n}\right\}l$) if $\sigma = \alpha_1 \mapsto M_1, \ldots, \alpha_n \mapsto M_n$, and similarly for substitutions on higher-order variables like β and mixtures of the two kinds.

The inference rules of system PE (for *Proof Enumeration*) are presented in Fig. 9.2. Derivability in PE of the three kinds of judgement is denoted $\Gamma \vdash_{\mathsf{PE}} M : A \mid \Sigma$, $\Gamma; B \vdash_{\mathsf{PE}} M : A \mid \Sigma$ and $\Sigma \Longrightarrow_{\mathsf{PE}} \sigma$.

Now we prove that PE is *sound*. For that we need the following notion:

Definition 129 (Solution) We define the property *σ is a solution of a goal environment* Σ, by induction on the length of Σ.

- σ is a solution of $[]$.
- If σ is a solution of Σ and

  $$x_1 : \sigma(A_1), \ldots, x_n : \sigma(A_n) \vdash_{\mathsf{PS}} \mathbf{app}(\sigma(\alpha), x_1\ [], \ldots, x_n\ []) : \sigma(A)$$

  then σ is a solution of $\Sigma, (x_1 : A_1, \ldots, x_n : A_n \vdash \alpha : A)$.
- If σ is a solution of Σ and

  $$x_1 : \sigma(A_1), \ldots, x_n : \sigma(A_n); \sigma(B) \vdash_{\mathsf{PS}} \mathbf{app}(\sigma(\beta), x_1\ [], \ldots, x_n\ []) : \sigma(A)$$

  then σ is a solution of $\Sigma, (x_1 : A_1, \ldots, x_n : A_n; B \vdash \beta : A)$.
- If σ is a solution of Σ and

  $$\sigma(M) \longleftrightarrow^*_{\mathsf{Bx}} \sigma(N)$$

  then σ is a solution of $\Sigma, M \overset{\Gamma}{\longleftrightarrow} N$.

For soundness we also need the following lemma:

Lemma 225 *Suppose that $\sigma(M)$ and $\sigma(M)$ are ground.*

1. *If $M \longrightarrow_{\mathsf{Bx}'} N$ then $\sigma(M) \longrightarrow_{\mathsf{Bx}} \sigma(N)$.*
2. *If $l \longrightarrow_{\mathsf{Bx}'} l'$ then $\sigma(l) \longrightarrow_{\mathsf{Bx}} \sigma(l')$.*

Proof: By simultaneous induction on the derivation of the reduction step, checking all rules for the base case of root reduction. □

Theorem 226 (Soundness) *Suppose σ is a solution of Σ.*

1. *If $\Gamma \vdash_{\mathsf{PE}} M : A \mid \Sigma$ then $\sigma(\Gamma) \vdash_{\mathsf{PS}} \sigma(M) : \sigma(A)$.*
2. *If $\Gamma; B \vdash_{\mathsf{PE}} M : A \mid \Sigma$ then $\sigma(\Gamma); \sigma(B) \vdash_{\mathsf{PS}} \sigma(M) : \sigma(A)$.*

Proof: By induction on derivations. □

Corollary 227 *If* $\Sigma \Longrightarrow_{\mathsf{PE}} \sigma$ *then* σ *is a solution of* Σ*.*

Proof: By induction on the derivation, using Theorem 226. □

System PE is *complete* in the following sense:

Theorem 228 (Completeness)

1. *If* $\Gamma \vdash_{\mathsf{PS}} M : A$ *then* $\Gamma \vdash_{\mathsf{PE}} M : A \mid \Sigma$ *for some solved goal environment* Σ*.*
2. *If* $\Gamma; B \vdash_{\mathsf{PS}} M : A$ *then* $\Gamma; B \vdash_{\mathsf{PE}} M : A \mid \Sigma$ *for some solved* Σ*.*

Proof: By induction on derivations. The rules of PE generalise those of PS. □

In fact, the completeness of the full system PE is not surprising, since it is quite general. In particular, nothing is said about when the process should decide to abandon the current goal and start working on another one. Hence we should be interested in completeness of particular strategies dealing with that question.

- For instance, PS corresponds to the strategy of eagerly solving sub-goals as soon as they are created, never delaying them with the sub-goal environment.
- The algorithm for proof enumeration in [Dow93] would correspond here to the "lazy" strategy that always abandons the sub-goal generated by rule $\mathrm{Il}_{\mathsf{PS}}$, but this in fact enables the unification constraints to give guidance in solving this sub-goal later, so in that case laziness is probably more efficient than eagerness. This is probably what should be chosen for automated theorem proving.
- Mixtures of the two strategies can also be considered and could be the basis of interactive theorem proving. Indeed in some cases the user's input might be more efficient than the automated algorithm, and rule $\mathrm{Il}_{\mathsf{PS}}$ would be a good place to ask whether the user has any clue to solve the sub-goal (since it could help solving the rest of the unification). If he or she has none, then by default the algorithm might abandon the sub-goal and leave it for later.

  In Coq, the tactic `apply x` does something similar: it tries to automatically solve the sub-goals that interfere with the unification constraint (leaving the other ones for later, visible to the user), but if the unification fails, it is always possible for the user to use the tactic and explicitly give the proof-term that will make it work. However, such an input is not provided in proof synthesis mode and the user really has to give it fully, since the tactic will fail if the unification fails. In PE, the unification constraint can remain partially solved.

All these behaviours can be simulated in PE, which is therefore a useful framework to study proof synthesis strategies in type theory.

9.3 PTSC with implicit substitutions

In this section we define a version of PTSC with implicit substitutions, called $\mathsf{PTSC}_{\mathsf{imp}}$. Note that the notion of implicit substitution from Definition 43 is not very useful here, since variables form a syntactic category of their own. What allows the definition of a notion of implicit substitution that corresponds to the explicit ones of PTSC is that the constructor for $x\ l$ can turn into the constructor for $M\ l$, which is different.

Definition 130 (The syntax with implicit substitutions) The syntax of $\mathsf{PTSC}_{\mathsf{imp}}$ is that of PTSC when we remove explicit substitutions, namely:

$$\begin{array}{rcl} M, N, A, B & ::= & \Pi x^A.B \mid \lambda x^A.M \mid s \mid x\ l \mid M\ l \\ l, l' & ::= & [] \mid M \cdot l \mid l@l' \end{array}$$

These terms and lists are henceforth called *substitution-free terms and lists*. Fig. 9.3 defines implicit substitutions on substitution-free terms and lists.

$$\begin{array}{|lcl|} \hline \left\{{}^{P}/_{y}\right\}(\lambda x^A.M) & := & \lambda x^{\left\{{}^{P}/_{y}\right\}A}.\left\{{}^{P}/_{y}\right\}M \\ \left\{{}^{P}/_{y}\right\}(y\ l) & := & P\ \left\{{}^{P}/_{y}\right\}l \\ \left\{{}^{P}/_{y}\right\}(x\ l) & := & x\ \left\{{}^{P}/_{y}\right\}l \\ \left\{{}^{P}/_{y}\right\}(M\ l) & := & \left\{{}^{P}/_{y}\right\}M\ \left\{{}^{P}/_{y}\right\}l \\ \left\{{}^{P}/_{y}\right\}(\Pi x^A.B) & := & \Pi x^{\left\{{}^{P}/_{y}\right\}A}.\left\{{}^{P}/_{y}\right\}B \\ \left\{{}^{P}/_{y}\right\}s & := & s \\ \hline \left\{{}^{P}/_{y}\right\}[] & := & [] \\ \left\{{}^{P}/_{y}\right\}(M \cdot l) & := & (\left\{{}^{P}/_{y}\right\}M) \cdot (\left\{{}^{P}/_{y}\right\}l) \\ \left\{{}^{P}/_{y}\right\}(l@l') & := & (\left\{{}^{P}/_{y}\right\}l)@(\left\{{}^{P}/_{y}\right\}l') \\ \hline \end{array}$$

Figure 9.3: Implicit substitutions in $\mathsf{PTSC}_{\mathsf{imp}}$

As with the notion of substitution from Definition 43, the substitution lemma holds (Lemma 40):

Lemma 229 (Substitution Lemma)

1. *If M, N, P are substitution-free terms, $\left\{{}^{P}/_{y}\right\}\left\{{}^{N}/_{x}\right\}M = \left\{{}^{\left\{{}^{P}/_{y}\right\}N}/_{x}\right\}\left\{{}^{P}/_{y}\right\}M$*

2. *If l is a substitution-free list and N, P are substitution-free terms, $\left\{{}^{P}/_{y}\right\}\left\{{}^{N}/_{x}\right\}l = \left\{{}^{\left\{{}^{P}/_{y}\right\}N}/_{x}\right\}\left\{{}^{P}/_{y}\right\}l$*

Proof: By induction on M, l. □

The following lemma shows that the propagation of explicit substitutions by th system x of PTSC implements the notion of implicit substitution defined above.

Lemma 230 *Provided P, M, l are substitution-free, we have $\langle P/x\rangle M \longrightarrow^*_{xsubst} \left\{{}^{P}\!/_{x}\right\} M$ and $\langle P/x\rangle l \longrightarrow^*_{xsubst} \left\{{}^{P}\!/_{x}\right\} l$.*

Proof: By induction on M, l. □

Definition 131 (Reduction system for $\mathsf{PTSC_{imp}}$) The internal reduction system of $\mathsf{PTSC_{imp}}$ is presented in Fig. 9.4.

$$
\begin{array}{lllll}
\mathsf{B}' & (\lambda x^A.M)\,(N\cdot l) & \longrightarrow & (\left\{{}^{N}\!/_{x}\right\} M)\; l \\
\mathsf{x} \left\{ \begin{array}{ll}
\mathsf{B1} & M\;[] \\
\mathsf{B2} & (x\;l)\;l' \\
\mathsf{B3} & (M\;l)\;l' \\
\mathsf{A1} & (M\cdot l')@l \\
\mathsf{A2} & []@l \\
\mathsf{A3} & (l@l')@l''
\end{array} \right. & \begin{array}{l} \longrightarrow \\ \longrightarrow \\ \longrightarrow \\ \longrightarrow \\ \longrightarrow \\ \longrightarrow \end{array} & \begin{array}{l} M \\ x\;(l@l') \\ M\;(l@l') \\ M\cdot(l'@l) \\ l \\ l@(l'@l'') \end{array}
\end{array}
$$

Figure 9.4: Reduction Rules of $\mathsf{PTSC_{imp}}$

Note that the system x of Fig. 9.4 is but the system x of Fig. 8.1 when all terms are substitution-free.

Remark 231

1. The syntax of substitution-free terms and lists is stable under $\longrightarrow_{\mathsf{B'x}}$.
2. Moreover, from Lemma 230 we get $\longrightarrow_{\mathsf{B}'} \subseteq \longrightarrow^*_{\mathsf{Bx}}$, so the rule adds nothing to the equational theory.

The reduction relation is closed under substitutions in the following sense:

Lemma 232 *Provided P, M, l are substitution-free,*

1. *if $P \longrightarrow_{B'x} P'$ then $\left\{{}^{P}\!/_{x}\right\} M \longrightarrow_{B'x} \left\{{}^{P'}\!/_{x}\right\} M$ and $\left\{{}^{P}\!/_{x}\right\} l \longrightarrow_{B'x} \left\{{}^{P'}\!/_{x}\right\} l$,*
2. *if $M \longrightarrow_{B'x} M'$ then $\left\{{}^{P}\!/_{x}\right\} M \longrightarrow_{B'x} \left\{{}^{P}\!/_{x}\right\} M'$, and if $l \longrightarrow_{B'x} l'$ then $\left\{{}^{P}\!/_{x}\right\} l \longrightarrow_{B'x} \left\{{}^{P}\!/_{x}\right\} l'$.*

Proof: By induction on M, l, using the Substitution Lemma for point 2 in the case of root B'-reduction. □

Using implicit substitutions we can now "purify" a term to remove its explicit substitutions. This purification is presented in Fig. 9.5.

Remark 233 If M is substitution-free then $\Downarrow(M) = M$.

Lemma 234 *$M \longrightarrow^*_{xsubst} \Downarrow(M)$ and $l \longrightarrow^*_{xsubst} \Downarrow(l)$.*

Proof: By induction on M, l. □

$\Downarrow(\lambda x^A.M)$	$:=$	$\lambda x^{\Downarrow(A)}.\Downarrow(M)$
$\Downarrow(x\ l)$	$:=$	$x\ \Downarrow(l)$
$\Downarrow(M\ l)$	$:=$	$\Downarrow(M)\ \Downarrow(l)$
$\Downarrow(\Pi x^A.B)$	$:=$	$\Pi x^{\Downarrow(A)}.\Downarrow(B)$
$\Downarrow(s)$	$:=$	s
$\Downarrow(\langle P/y\rangle M)$	$:=$	$\left\{{}^{\Downarrow(P)}/_{y}\right\}\Downarrow(M)$
$\Downarrow([])$	$:=$	$[]$
$\Downarrow(M\cdot l)$	$:=$	$(\Downarrow(M))\cdot(\Downarrow(l))$
$\Downarrow(l@l')$	$:=$	$(\Downarrow(l))@(\Downarrow(l'))$

Figure 9.5: Purification

Theorem 235 (Simulation of Bx by $\mathsf{B'x}$ through $\Downarrow$)

1. *If* $M \longrightarrow_{\mathsf{Bx}} N$ *then* $\Downarrow(M) \longrightarrow^*_{\mathsf{B'x}} \Downarrow(N)$.
2. *If* $l \longrightarrow_{\mathsf{Bx}} l'$ *then* $\Downarrow(l) \longrightarrow^*_{\mathsf{B'x}} \Downarrow(l')$.

Proof: By induction on M, l, using Lemma 229. □

Corollary 236 (Reflection in PTSC of $\mathsf{PTSC}_{\mathsf{imp}}$) $\Downarrow$ *and the identity function form a reflection in* PTSC *of* $\mathsf{PTSC}_{\mathsf{imp}}$.

Proof: This is the conjunction of Lemma 230, Remark 233, Lemma 234, and Theorem 235. □

Corollary 237 (Confluence of $\mathsf{PTSC}_{\mathsf{imp}}$) $\mathsf{PTSC}_{\mathsf{imp}}$ *are confluent.*

Proof: From Corollary 236 we get that the identity function and $\Downarrow$ form a pre-Galois connection from $\mathsf{PTSC}_{\mathsf{imp}}$ to PTSC, so we can apply Theorem 5. □

9.4 Type synthesis

Having identified the substitution-free fragment of the syntax, we can now remove from the typing system rules cut_2 and cut_4. The second step to get a typing system that is syntax-directed for type synthesis is to integrate the conversion rules to the other rules in the spirit of the *Constructive Engine* in natural deduction [Hue89, vBJMP94]. Such a treatment of the conversion rules allows the type-checking constraints to be treated in a particular way:

- the output must be type-checked as the type synthesis procedure goes, but

- as in proof synthesis, we can gather in a preliminary phase the type-checking of its inputs, namely the fact that the environment is well-formed (and, if need be, that the type in the stoup can be typed in that environment), which will then be an invariant of the system, used in the proof of soundness (Corollary 239).

A consequence of this is that normal forms can be typed without using cut-rules in this system, which is still sound and complete. This property held in propositional logic but was lost in the system defining PTSC (Fig. 8.5).

Definition 132 (A constructive engine for PTSC)
The inference rules of system TS are given in Fig. 9.6, where $\Gamma \vdash M :\equiv A$ abbreviates $\exists C, (\Gamma \vdash M : C) \wedge (C \longleftrightarrow^*_{\mathsf{Bx}} A)$ (and similarly for $\Gamma; B \vdash l :\equiv A$). We write $\Gamma \vdash_{\mathsf{TS}} M : A$, $\Gamma; B \vdash_{\mathsf{TS}} l : A$ and Γ $\mathsf{wf}_{\mathsf{TS}}$ when the judgements are derivable in TS.

$$\frac{}{[]\ \mathsf{wf}}\ \mathsf{empty} \qquad \frac{\Gamma\ \mathsf{wf} \quad \Gamma \vdash A :\equiv s \quad x \notin \mathsf{Dom}(\Gamma)}{\Gamma, x : A\ \mathsf{wf}}\ \mathsf{extend}_{\mathsf{TS}}$$

$$\frac{}{\Gamma; A \vdash [] : A}\ \mathsf{ax} \qquad \frac{D \longrightarrow^*_{\mathsf{Bx}} \Pi x^A.B \quad \Gamma \vdash M :\equiv A \quad \Gamma; \{^M/_x\} B \vdash l : C}{\Gamma; D \vdash M \cdot l : C}\ \Pi\mathsf{l}_{\mathsf{TS}}$$

$$\frac{\Gamma; C \vdash l' : A \qquad \Gamma; A \vdash l : B}{\Gamma; C \vdash l'@l : B}\ \mathsf{cut}_1$$

$$\frac{(s_1, s_2) \in \mathcal{A}}{\Gamma \vdash s_1 : s_2}\ \mathsf{sorted} \qquad \frac{\Gamma \vdash A :\equiv s_1 \quad \Gamma, x : A \vdash B :\equiv s_2 \quad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash \Pi x^A.B : s_3}\ \Pi\mathsf{wf}_{\mathsf{TS}}$$

$$\frac{(x : A) \in \Gamma \quad \Gamma; A \vdash l : B}{\Gamma \vdash x\ l : B}\ \mathsf{select}_x \qquad \frac{\Gamma, x : A \vdash M :\equiv B \quad \Gamma \vdash \Pi x^A.B : s}{\Gamma \vdash \lambda x^A.M : \Pi x^A.B}\ \Pi\mathsf{r}_{\mathsf{TS}}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma; A \vdash l : B}{\Gamma \vdash M\ l : B}\ \mathsf{cut}_3$$

Figure 9.6: System TS

In order to prove soundness we need the following lemma:

Lemma 238

1. *Provided* Γ $\mathsf{wf}_{\mathsf{PTSC}}$, *if* $\Gamma \vdash_{\mathsf{TS}} M : A$ *then* $\Gamma \vdash_{\mathsf{PTSC}} M : A$.
2. *Provided* $\Gamma \vdash_{\mathsf{PTSC}} B : s$, *if* $\Gamma; B \vdash_{\mathsf{TS}} l : A$ *then* $\Gamma; B \vdash l : A$.

Proof: By simultaneous induction on the typing derivations for M and l. □

Soundness can then be stated (notice the proviso that the inputs of type synthesis have themselves been type-checked):

Corollary 239 (Soundness)

1. *If* Γ $\mathsf{wf}_{\mathsf{TS}}$ *then* Γ $\mathsf{wf}_{\mathsf{PTSC}}$.
2. *Provided* Γ $\mathsf{wf}_{\mathsf{TS}}$, *if* $\Gamma \vdash_{\mathsf{TS}} M : A$ *then* $\Gamma \vdash_{\mathsf{PTSC}} M : A$.
3. *Provided* Γ $\mathsf{wf}_{\mathsf{TS}}$ *and* $\Gamma \vdash_{\mathsf{TS}} B : s$, *if* $\Gamma; B \vdash_{\mathsf{TS}} l : A$ *then* $\Gamma; B \vdash_{\mathsf{PTSC}} l : A$.

Proof: Point 1 is proved by induction on the length of Γ, using Lemma 238.1. Point 2 and 3 are straightforward consequences of point 1 and Lemma 238. □

Now we want to prove completeness, for which we need the following lemma:

Lemma 240 *If* $\Gamma; B \vdash_{\mathsf{TS}} l : A$ *and* $B \longleftrightarrow^*_{\beta x} B'$ *then* $\Gamma; B' \vdash_{\mathsf{TS}} l :\equiv A$

Proof: By induction on the derivation. □

For completeness we extend $\Downarrow$ to environment as follows:

Definition 133 ($\Downarrow$ on environments) We define $\Downarrow(\Gamma)$ by induction on the length of Γ:

$$\begin{array}{ll} \Downarrow([]) & := [] \\ \Downarrow(\Gamma, x : A) & := \Downarrow(\Gamma), x \Downarrow(A) \end{array}$$

Theorem 241 (Completeness) *Suppose* M *and* l *are substitution-free.*

1. *If* $\Gamma \vdash_{\mathsf{PTSC}} M : A$ *then* $\Gamma \vdash_{\mathsf{TS}} M :\equiv A$.
2. *If* $\Gamma; B \vdash_{\mathsf{PTSC}} l : A$ *then* $\Gamma; B \vdash_{\mathsf{TS}} l :\equiv A$.
3. *If* Γ $\mathsf{wf}_{\mathsf{PTSC}}$ *then* $\Downarrow(\Gamma)$ $\mathsf{wf}_{\mathsf{TS}}$.

Proof: Th first two points are proved by simultaneous induction on derivations. Point 3 is proved by induction on the length of Γ, using Lemma 230, subject reduction in PTSC (Theorem 216), and point 1. □

In system TS, the conversion rules are embedded in some of the rules. This and the fact that we have given up typing of explicit substitutions, removing rules cut_2 and cut_4, make the system almost syntax-directed.

A non-problematic point of non-determinism lies in rule $\Pi\mathsf{I}_{\mathsf{TS}}$, where the type A is not completely given. But in fact, its existence just expresses the convertibility of the type produced for M with the first component of *whichever* Π*-type* D *reduces to*. Hence, any such A would do (but deciding whether there exists one by normalisation would require the strong normalisation of the PTS).

A minor point of non-determinism lies in rules sorted and $\Pi\mathsf{wf}_{\mathsf{TS}}$ when there is a choice for s_2 and s_3, respectively. This cannot be avoided unless $\mathcal{A}$ and $\mathcal{R}$ are functions.

A more problematic point lies in rule $\Pi\mathsf{r}_{\mathsf{TS}}$ where the type B has to be invented. It is tempting to replace rule $\Pi\mathsf{r}_{\mathsf{TS}}$ with the following one:

$$\frac{\Gamma, x{:}A \vdash M{:}B \quad \Gamma \vdash \Pi x^A.B : s}{\Gamma \vdash \lambda x^A.M : \Pi x^A.B}\ \Pi\mathsf{r}'_{\mathsf{TS}}$$

Unfortunately, unlike the previous version, completeness of the system with that rule would imply the property of *Expansion Postponement* [Pol92, vBJMP94, Pol98, GR03a], which is still an open problem for general PTS, and thus for general PTSC as well. Indeed, in our framework of PTSC the problem of *Expansion Postponement* can be seen as the completeness of TS but with the following rule instead of $\Pi\mathsf{r}_{\mathsf{TS}}$:

$$\frac{\Gamma, x{:}A \vdash M{:}C \quad C \longrightarrow^*_{\mathsf{Bx}} B \quad \Gamma \vdash \Pi x^A.B : s}{\Gamma \vdash \lambda x^A.M : \Pi x^A.B}\ \Pi\mathsf{r}''_{\mathsf{TS}}$$

Completeness of a system with this rule relies on the following permutation:

$$\dfrac{\dfrac{\Gamma, x{:}A \vdash M{:}D \quad D \longleftrightarrow^*_{\mathsf{Bx}} C \quad \Gamma, x{:}A \vdash C : s'}{\Gamma, x{:}A \vdash M{:}C} \qquad C \longrightarrow^*_{\mathsf{Bx}} B \quad \Gamma \vdash \Pi x^A.B : s}{\Gamma \vdash \lambda x^A.M : \Pi x^A.B}$$

must be transformed into

$$\dfrac{\dfrac{\Gamma, x{:}A \vdash M{:}D \quad D \longrightarrow^*_{\mathsf{Bx}} D' \quad \Gamma \vdash \Pi x^A.D' : s}{\Gamma \vdash \lambda x^A.M : \Pi x^A.D'} \qquad \Pi x^A.B \longrightarrow^*_{\mathsf{Bx}} \Pi x^A.D' \quad \Gamma \vdash \Pi x^A.B : s}{\Gamma \vdash \lambda x^A.M : \Pi x^A.B}$$

with $D \longrightarrow^*_{\mathsf{Bx}} D'$ and $B \longrightarrow^*_{\mathsf{Bx}} D'$ obtained by confluence of Bx. The bottom-most inference step is the *expansion* (a particular case of conversion) that is being postponed after the interesting rule $\Pi\mathsf{r}$, rather than before (when the derivations are considered top-down). But how could we derive the premiss $\Gamma \vdash \Pi x^A.D' : s$

knowing $\Gamma \vdash \Pi x^A.B : s$? We could if we knew that subject reduction held in the system where expansions are postponed, and one way to obtain subject reduction is to use completeness; in fact the two properties are equivalent.

As mentioned above, we are particularly interested in PTSC where types are strongly normalising, in which we can easily decide the convertibility problem of rule $\Pi\mathsf{l}_{\mathsf{TS}}$. In such a framework, [GR03a] proves that *Expansion Postponement* holds if normal forms (of proof-terms in natural deduction) can be typed in a cut-free sequent calculus. Further work includes relating this result to the following question in our framework: what is the relation between *Expansion Postponement* and the following property that a PTSC can have?

If M, A and the types in Γ are all normal forms and $\Gamma \vdash_{\mathsf{PTSC}} M : A$, can $\Gamma \vdash M : A$ be derived in the (cut-free) system of Fig. 9.7 (which is similar to that of [GR03a])?

$$\frac{}{[]\ \mathsf{wf}} \qquad \frac{\Gamma \vdash A : s \quad x \notin \mathsf{Dom}(\Gamma)}{\Gamma, x : A\ \mathsf{wf}}$$

$$\frac{\Gamma \vdash A : s}{\Gamma; A \vdash [] : A} \qquad \frac{\Gamma \vdash M : A \quad \{^{M}/_{x}\} B \longrightarrow^{*}_{\mathsf{Bx}} D \quad \Gamma; D \vdash l : C}{\Gamma; \Pi x^A.B \vdash M \cdot l : C}$$

$$\frac{\Gamma\ \mathsf{wf}(s_1, s_2) \in \mathcal{A}}{\Gamma \vdash s_1 : s_2} \qquad \frac{\Gamma \vdash A : s_1 \quad \Gamma, x : A \vdash B : s_2 \quad (s_1, s_2, s_3) \in \mathcal{R}}{\Gamma \vdash \Pi x^A.B : s_3}$$

$$\frac{(x : A) \in \Gamma \quad \Gamma; A \vdash l : B}{\Gamma \vdash x\ l : B} \qquad \frac{\Gamma, x : A \vdash M : B \quad \Gamma \vdash \Pi x^A.B : s}{\Gamma \vdash \lambda x^A.M : \Pi x^A.B}$$

Figure 9.7: System for tackling *Expansion Postponement*?

Answering this question in the light of [GR03a] might help finding a potential counter-example to *Expansion Postponement*.

9.5 PTSC with de Bruijn indices ($\mathsf{PTSC}_{\mathsf{db}}$)

In this section we present implementation-friendly versions of PTSC, using de Bruijn indices [dB72] in the version which is closest to our unary version of substitutions with variables. We base the approach on the version of PTSC with implicit substitutions (otherwise the typing rules cut_2 and cut_4 for explicit substitutions require an even heavier machinery with de Bruijn indices). As mentioned

in the introduction of this chapter, this section develops for PTSC what [KR02] developed for PTS, although we were not aware of it.

Definition 134 (Terms) The set $\mathcal{T}_{\mathsf{db}}$ of terms (denoted $M, N, P, \ldots$) and the set $\mathcal{L}_{\mathsf{db}}$ of lists (denoted $l, l', \ldots$) are defined by induction as:

$$\begin{array}{rl} M, N, A, B & ::= \Pi^A B \mid \lambda^A M \mid s \mid n\ l \mid M\ l \\ l, l' & ::= [] \mid M \cdot l \mid l@l' \end{array}$$

where n ranges over natural numbers.

Definition 135 (Updating, Substitution, Reduction) We define the *updating operation* of the free variables of terms as described in Fig. 9.8. The notion of substitution is defined in Fig. 9.9. The reduction rules are presented in Fig. 9.10.

$$\begin{array}{lcll} U_i^n(\lambda^A M) & := & \lambda^{U_i^n(A)} U_{i+1}^n(M) & \\ U_i^n((m\ l)) & := & (m+i)\ U_i^n(l) & i \leq m \\ U_i^n((m\ l)) & := & m\ U_i^n(l) & i > m \\ U_i^n((M\ l)) & := & U_i^n(M)\ U_i^n(l) & \\ U_i^n(\Pi^A B) & := & \Pi^{U_i^n(A)} U_{n+1}^n(B) & \\ U_i^n(s) & := & s & \\ \hline U_i^n([]) & := & [] & \\ U_i^n((M \cdot l)) & := & (U_i^n(M)) \cdot (U_i^n(l)) & \\ U_i^n((l@l')) & := & (U_i^n(l))@(U_i^n(l')) & \end{array}$$

Figure 9.8: Updating

$$\begin{array}{lcll} (\lambda^A M)\{\!\{n \leftarrow P\}\!\} & := & \lambda^{A\{\!\{n \leftarrow P\}\!\}} M\{\!\{n+1 \leftarrow P\}\!\} & \\ (m\ l)\{\!\{n \leftarrow P\}\!\} & := & U_0^n(P)\ l\{\!\{n \leftarrow P\}\!\} & m = n \\ (m\ l)\{\!\{n \leftarrow P\}\!\} & := & m\ l\{\!\{n \leftarrow P\}\!\} & m < n \\ (m\ l)\{\!\{n \leftarrow P\}\!\} & := & (m-1)\ l\{\!\{n \leftarrow P\}\!\} & m > n \\ (M\ l)\{\!\{n \leftarrow P\}\!\} & := & M\{\!\{n \leftarrow P\}\!\}\ l\{\!\{n \leftarrow P\}\!\} & \\ (\Pi^A B)\{\!\{n \leftarrow P\}\!\} & := & \Pi^{A\{\!\{n \leftarrow P\}\!\}} B\{\!\{n+1 \leftarrow P\}\!\} & \\ s\{\!\{n \leftarrow P\}\!\} & := & s & \\ \hline []\{\!\{n \leftarrow P\}\!\} & := & [] & \\ (M \cdot l)\{\!\{n \leftarrow P\}\!\} & := & (M\{\!\{n \leftarrow P\}\!\}) \cdot (l\{\!\{n \leftarrow P\}\!\}) & \\ (l@l')\{\!\{n \leftarrow P\}\!\} & := & (l\{\!\{n \leftarrow P\}\!\})@(l'\{\!\{n \leftarrow P\}\!\}) & \end{array}$$

Figure 9.9: Substitutions

$$
\text{System } \mathsf{x_{db}}\text{:}\begin{cases}
\mathsf{B_{db}} & (\lambda^A M)\ (N \cdot l) & \longrightarrow & (M\{\!\{0 \leftarrow N\}\!\})\ l \\
\mathsf{B1_{db}} & M\ [] & \longrightarrow & M \\
\mathsf{B2_{db}} & (n\ l)\ l' & \longrightarrow & n\ (l @ l') \\
\mathsf{B3_{db}} & (M\ l)\ l' & \longrightarrow & M\ (l @ l') \\
\mathsf{A1_{db}} & (M \cdot l') @ l & \longrightarrow & M \cdot (l' @ l) \\
\mathsf{A2_{db}} & [] @ l & \longrightarrow & l \\
\mathsf{A3_{db}} & (l @ l') @ l'' & \longrightarrow & l @ (l' @ l'')
\end{cases}
$$

Figure 9.10: Reduction rules

Now we proceed to the typing systems:

Definition 136 (Environment & typing systems)

- Environments are lists of terms.
- The typing rules for proof synthesis are presented in Fig. 9.11. Judgements derivable in $\mathsf{PS_{db}}$ are denoted $\Gamma \vdash_{\mathsf{PS_{db}}} M : A$ and $\Gamma; B \vdash_{\mathsf{PS_{db}}} l : A$.
- The typing rules for type synthesis are presented in Fig. 9.12. We write $\Gamma \vdash M : \equiv A$ if there is C such that $\Gamma \vdash M : C$ and $C \longleftrightarrow^*_{\mathsf{B_{db} x_{db}}} A$ (and similarly for $\Gamma; B \vdash l : \equiv C$). Judgements derivable in $\mathsf{TS_{db}}$ are denoted $\Gamma \vdash_{\mathsf{TS_{db}}} M : A$, $\Gamma; B \vdash_{\mathsf{TS_{db}}} l : A$ and $\Gamma\ \mathsf{wf}_{\mathsf{TS_{db}}}$.

$$
\frac{A \longleftrightarrow^*_{\mathsf{B_{db} x_{db}}} A'}{\Gamma; A \vdash [] : A'}
\qquad
\frac{D \longrightarrow^*_{\mathsf{B_{db} x_{db}}} \Pi^A B \quad \Gamma \vdash M : A \quad \Gamma; B\{\!\{0 \leftarrow M\}\!\} \vdash l : C}{\Gamma; D \vdash M \cdot l : C}
$$

$$
\frac{C \longrightarrow^*_{\mathsf{B_{db} x_{db}}} s_3 \quad \Gamma \vdash A : s_1 \quad \Gamma, A \vdash B : s_2 \quad (s_1, s_2, s_3) \in R}{\Gamma \vdash \Pi^A B : C}
$$

$$
\frac{C \longrightarrow^*_{\mathsf{B_{db} x_{db}}} s_2 \quad (s_1, s_2) \in \mathcal{A}}{\Gamma \vdash s_1 : s_2}
\qquad
\frac{\Gamma, A, \Delta; U_0^{|\Delta|+1}(A) \vdash l : B}{\Gamma \vdash |\Delta|\ l : B}
$$

$$
\frac{D \longrightarrow^*_{\mathsf{B_{db} x_{db}}} \Pi^A B \quad \Gamma, A \vdash M : B}{\Gamma \vdash \lambda^A M : D}
$$

Figure 9.11: Proof synthesis system for $\mathsf{PTSC_{db}}$

$$\frac{}{[]\ \mathsf{wf}} \qquad \frac{\Gamma\ \mathsf{wf} \quad \Gamma \vdash A :\equiv s}{\Gamma, A\ \mathsf{wf}}$$

$$\frac{}{\Gamma; A \vdash [] : A} \qquad \frac{D \longrightarrow^{*}_{\mathsf{B}_{\mathsf{db}} \times_{\mathsf{db}}} \Pi^{A} B \quad \Gamma \vdash M :\equiv A \quad \Gamma; B\{\!\{0 \leftarrow M\}\!\} \vdash l : C}{\Gamma; D \vdash M \cdot l : C}$$

$$\frac{(s_1, s_2) \in \mathcal{A}}{\Gamma \vdash s_1 : s_2} \qquad \frac{\Gamma \vdash A :\equiv s_1 \quad \Gamma, A \vdash B :\equiv s_2 \quad (s_1, s_2) \in R}{\Gamma \vdash \Pi^{A} B : s_2}$$

$$\frac{\Gamma, A, \Delta; U_0^{|\Delta|+1}(A) \vdash l : B}{\Gamma \vdash |\Delta|\ l : B} \qquad \frac{\Gamma, A \vdash M :\equiv B \qquad \Gamma \vdash \Pi^{A} B : C}{\Gamma \vdash \lambda^{A} M : \Pi^{A} B}$$

$$\frac{\Gamma; C \vdash l' : A \qquad \Gamma; A \vdash l : B}{\Gamma; C \vdash l' @ l : B} \qquad \frac{\Gamma \vdash M : A \qquad \Gamma; A \vdash l : B}{\Gamma \vdash M\ l : B}$$

Figure 9.12: Type synthesis system for $\mathsf{PTSC}_{\mathsf{db}}$

9.5.1 From $\mathsf{PTSC}_{\mathsf{db}}$ to PTSC

We encode the terms with de Bruijn indices as terms with variables. The encoding depends on a one-to-one total function from natural numbers to variables, which is required to be *co-partial*:

Definition 137 (Co-partiality)

- A *co-partial* function is an injective and total function from natural numbers to variables such that $\overline{\mathsf{im}}(f) := \{x \in \mathcal{X} | \forall n, f(n) \neq x\}$ is infinite.
- For any co-partial f, we define the function f, x as the following mapping:

$$\begin{array}{ll} 0 & \mapsto x \\ n+1 & \mapsto f(n) \end{array}$$

  We extend the notation to f, h for a list h of variables by obvious induction on the length of h.

Intuitively, a co-partial function is an infinite list of distinct variables $\ldots, f(n), \ldots, f(1), f(0)$ which still leaves infinitely many variables that are not enumerated.

Remark 242

1. If $x \notin \overline{\mathsf{im}}(f)$ then f, x is also co-partial.
2. For any co-partial f there exists f' and x such that $f = f', x$.

Definition 138 (Splitting an infinite list) Every co-partial function f can be split as $f = \rho^n(f), \pi^n(f)$, where

- $\rho^n(f)$ is another co-partial function defined as follows:

$$\begin{array}{lcl} \rho^0(f) & := & f \\ \rho^{n+1}(f, m) & := & \rho^n(f) \end{array}$$

- $\pi^n(f)$ is a list of variable defined as follows

$$\begin{array}{lcl} \pi^0(f) & := & [\![\\ \pi^{n+1}(f, m) & := & \pi^n(f), m \end{array}$$

Intuitively, $\pi^n(f)$ is the list of the first n variables enumerated by f, and f' is the rest (an infinite list of variables described as a co-partial function).

Definition 139 (Encoding $\mathsf{PTSC}_{\mathsf{db}}$ into $\mathsf{PTSC}_{\mathsf{imp}}$) The encoding is presented in Fig. 9.13. It naturally depends on a co-partial function f that assigns fresh variables to de Bruijn indices.

$$\begin{array}{|lcll|} \hline \overline{\Pi^A M}^f & := & \Pi x^{\overline{A}^f} . \overline{M}^{f,x} & x \notin \overline{\mathsf{im}}(f) \\ \overline{\lambda^A M}^f & := & \lambda x^{\overline{A}^f} . \overline{M}^{f,x} & x \notin \overline{\mathsf{im}}(f) \\ \overline{s}^f & := & s & \\ \overline{m\ l}^f & := & f(m)\ \overline{l}^f & \\ \overline{M\ l}^f & := & \overline{M}^f\ \overline{l}^f & \\ \hline \overline{[]}^f & := & [] & \\ \overline{M \cdot l}^f & := & \overline{M}^f \cdot \overline{l}^f & \\ \overline{l @ l'}^f & := & \overline{l}^f @ \overline{l'}^f & \\ \hline \end{array}$$

Figure 9.13: Encoding of $\mathsf{PTSC}_{\mathsf{db}}$ into $\mathsf{PTSC}_{\mathsf{imp}}$

Now we investigate the notions of reduction and equivalence between $\mathsf{PTSC}_{\mathsf{db}}$ and PTSC:

Lemma 243

1. $\overline{U_i^n(M)}^f = \overline{M}^{\rho^{i+n}(f), \pi^i(f)}$ *and* $\overline{U_i^n(l)}^f = \overline{l}^{\rho^{i+n}(f), \pi^i(f)}$

2. $\overline{M\{\!\{n \leftarrow P\}\!\}}^f = \left\{ \overline{P}^{\rho^n(f)} \big/ _x \right\} \overline{M}^{\rho^n(f),x,\pi^n(f)}$ *and* $\overline{l\{\!\{n \leftarrow P\}\!\}}^f = \left\{ \overline{P}^{\rho^n(f)} \big/ _x \right\} \overline{l}^{\rho^n(f),x,\pi^n(f)}$.

Proof:

1. By induction on M, l.
2. By induction on M, l, using point 1.

□

Theorem 244 (Simulation of $\mathsf{B_{db}x_{db}}$ by $\mathsf{B'x}$)

1. *If* $M \longrightarrow_{\mathsf{B_{db}x_{db}}} N$ *then* $\overline{M}^f \longrightarrow_{\mathsf{B'x}} \overline{N}^f$ *(and then* $\overline{M}^f \longrightarrow^*_{\mathsf{Bx}} \overline{N}^f$ *as well).*
2. *If* $l \longrightarrow_{\mathsf{B_{db}x_{db}}} l'$ *then* $\overline{l}^f \longrightarrow_{\mathsf{B'x}} \overline{l}^f$. *(and then* $\overline{l}^f \longrightarrow^*_{\mathsf{Bx}} \overline{l'}^f$ *as well).*

Proof: By induction on the derivation of the reduction step, checking all the rules and using Lemma 243. □

From this we deduce the inclusion of the equational theories:

Corollary 245 *If* $M \longleftrightarrow^*_{\mathsf{B_{db}x_{db}}} N$ *then* $\overline{M}^f \longleftrightarrow^*_{\mathsf{B'x}} \overline{N}^f$ *(and then* $\overline{M}^f \longleftrightarrow^*_{\mathsf{Bx}} \overline{N}^f$ *as well).*

Proof: This is a trivial consequence of Theorem 244. □

Now we check that the typing is preserved by the encoding:

Definition 140 (Encoding of environments) We now encode the environments as follows:

$$\begin{array}{lcl} \overline{[]}^f & := & [] \\ \overline{\Gamma, A}^f & := & \overline{\Gamma}^{\rho^1(f)}, f(0) : \overline{A}^{\rho^1(f)} \end{array}$$

The following theorems are natural consequences of the fact that the rules of Fig. 9.11 and Fig. 9.12 are respectively those of Fig. 9.1 and Fig. 9.6 but with de Bruijn indices.

Theorem 246 (Preservation of typing: proof synthesis)

1. *If* $\Gamma \vdash_{\mathsf{PS_{db}}} M : A$ *then* $\overline{\Gamma}^f \vdash_{PS} \overline{M}^f : \overline{A}^f$.
2. *If* $\Gamma; l \vdash_{\mathsf{PS_{db}}} M : A$ *then* $\overline{\Gamma}^f; \overline{l}^f \vdash_{PS} \overline{M}^f : \overline{A}^f$.

Proof: By induction on derivations. □

Theorem 247 (Preservation of typing: type synthesis)

1. *If* $\Gamma \vdash_{\mathsf{TS_{db}}} M : A$ *then* $\overline{\Gamma}^f \vdash_{\mathsf{TS}} \overline{M}^f : \overline{A}^f$.
2. *If* $\Gamma; l \vdash_{\mathsf{TS_{db}}} M : A$ *then* $\overline{\Gamma}^f; \overline{l}^f \vdash_{\mathsf{TS}} \overline{M}^f : \overline{A}^f$.
3. *If* Γ $\mathsf{wf}_{\mathsf{TS_{db}}}$ *then* $\overline{\Gamma}^f$ $\mathsf{wf}_{\mathsf{TS}}$.

Proof: By induction on derivations. □

9.5.2 From PTSC to PTSC$_{\mathsf{db}}$

We encode the terms with variables as terms with de Bruijn indices. The encoding depends on a one-to-one function that maps variables to natural numbers.

Definition 141 (Co-finiteness)

- A *co-finite* function f is a (partial) injective function from variables (set $\mathcal{X}$) to natural numbers such that $\mathcal{X} \setminus \mathsf{Dom}(f)$ is infinite.
- Given a co-finite function f and a natural number n, we define the function $f + n$ as:
$$(f + n) := \ x \mapsto f(x) + n$$
- Given a co-finite function f and a finite injective function g with disjoint domains, we write f, g to denote the union of them (seen as sets), which is again a co-finite function.

Definition 142 (Encoding PTSC$_{\mathsf{imp}}$ into PTSC$_{\mathsf{db}}$) Given a co-finite function f and a substitution-free term M (resp. list l) of PTSC$_{\mathsf{imp}}$ such that $\mathsf{FV}(M) \subseteq \mathsf{Dom}(f)$ (resp. $\mathsf{FV}(l) \subseteq \mathsf{Dom}(f)$), we define the encoding of M (resp. l) in Fig. 9.14.

$\underline{\Pi x^A.M}_f$	:=	$\Pi^{\underline{A}_f} \underline{M}_{f+1,x\mapsto 0}$	$x \notin \mathsf{Dom}(f)$
$\underline{\lambda x^A.M}_f$	:=	$\lambda^{\underline{A}_f} \underline{M}_{f+1,x\mapsto 0}$	$x \notin \mathsf{Dom}(f)$
$\underline{s}_f$	:=	s	
$\underline{x\ l}_f$	:=	$f(x)\ \underline{l}_f$	
$\underline{M\ l}_f$	:=	$\underline{M}_f\ \underline{l}_f$	
$\underline{[]}_f$	:=	$[]$	
$\underline{M \cdot l}_f$	:=	$\underline{M}_f \cdot \underline{l}_f$	
$\underline{l@l'}_f$	:=	$\underline{l}_f @ \underline{l'}_f$	

Figure 9.14: Encoding of PTSC$_{\mathsf{imp}}$ into PTSC$_{\mathsf{db}}$

Now we establish properties of the updating and substitution:

Lemma 248

1. *If* $\mathsf{FV}(M) \subseteq \mathsf{Dom}(f)$ *then* $U_i^n(\underline{M}_f) = \underline{M}_{f'}$, *and*
   if $\mathsf{FV}(l) \subseteq \mathsf{Dom}(f)$ *then* $U_i^n(\underline{l}_f) = \underline{l}_{f'}$,
   where
   $$f'(x) = f(x) \qquad \text{if } f(x) < i$$
   $$f'(x) = f(x) + n \quad \text{if } f(x) \geq i$$

2. (a) *If* $\mathsf{FV}(N) \subseteq \mathsf{Dom}(f)$ *and* $\mathsf{FV}(M) \subseteq \mathsf{Dom}(f) \cup \mathsf{Dom}(g) \cup \{x\}$
   then $\underline{\{N/x\}M}_{f+n,g} = \underline{M}_{f+n+1,g,x\mapsto n}\{\{n \leftarrow \underline{N}_f\}\}$.

   (b) *If* $\mathsf{FV}(N) \subseteq \mathsf{Dom}(f)$ *and* $\mathsf{FV}(l) \subseteq \mathsf{Dom}(f) \cup \mathsf{Dom}(g) \cup \{x\}$
   then $\underline{\{N/x\}l}_{f+n,g} = \underline{l}_{f+n+1,g,x\mapsto n}\{\{n \leftarrow \underline{N}_f\}\}$.

Proof:

1. By induction on M, l.

2. By induction on M, l, using point 1.

□

Theorem 249 (Simulation of $\mathsf{B'x}$ by $\mathsf{B_{db}x_{db}}$)
Provided M *and* l *are substitution-free,*

1. *If* $M \longrightarrow_{\mathsf{B'x}} N$ *then* $\underline{M}_f \longrightarrow_{\mathsf{B_{db}x_{db}}} \underline{N}_f$.

2. *If* $l \longrightarrow_{\mathsf{B'x}} l'$ *then* $\underline{l}_f \longrightarrow_{\mathsf{B_{db}x_{db}}} \underline{l'}_f$.

Proof: By induction on the derivation step, by checking all the rules and using Lemma 248. □

From this we deduce the inclusion of the equational theories:

Corollary 250 *Supposing that* M *is substitution-free,*
if $M \longleftrightarrow^*_{\mathsf{B'x}} N$ *then* $\underline{M}_f \longleftrightarrow^*_{\mathsf{B_{db}x_{db}}} \underline{N}_f$.

Proof: This is a trivial consequence of Theorem 249. □

Definition 143 (Encoding of PTSC into $\mathsf{PTSC_{db}}$) Now we extend the encoding on every term and list:

- If $\mathsf{FV}(M) \subseteq \mathsf{Dom}(f)$, we define $\underline{M}_f = \underline{\Downarrow(M)}_f$.
- If $\mathsf{FV}(l) \subseteq \mathsf{Dom}(f)$, we define $\underline{l}_f = \underline{\Downarrow(l)}_f$.

Theorem 251 (Simulation of Bx by $\mathsf{B_{db}x_{db}}$)

1. *If* $M \longrightarrow_{Bx} N$ *then* $\underline{M}_f \longrightarrow^*_{B_{\mathsf{db}}x_{\mathsf{db}}} \underline{N}_f$.
2. *If* $l \longrightarrow_{Bx} l'$ *then* $\underline{l}_f \longrightarrow^*_{B_{\mathsf{db}}x_{\mathsf{db}}} \underline{l'}_f$.

Proof: By Theorem 235 and Theorem 249. □

From this we deduce the inclusion of the equational theories:

Corollary 252 *If* $M \longleftrightarrow^*_{Bx} N$ *then* $\underline{M}_f \longleftrightarrow^*_{B_{\mathsf{db}}x_{\mathsf{db}}} \underline{N}_f$.

Proof: This is a trivial consequence of Theorem 251. □

Definition 144 (Encoding of environments of PTSC)

- An environment Γ of PTSC is *decent* if for all of its decompositions $\Gamma = \Delta, x : A, \Delta'$ we have $x \notin \mathsf{Dom}(\Delta)$ and $\mathsf{FV}(A) \subseteq \mathsf{Dom}(\Delta)$.

  Note that all well-formed environments are decent.

- A decent environment defines the following co-finite function:

$$\begin{array}{lcl} \phi_{[]} & := & \emptyset \\ \phi_{\Gamma,x:A} & := & \phi_\Gamma + 1, x \mapsto 0 \end{array}$$

- We define the encoding of a decent environment as follows:

$$\begin{array}{lcl} \underline{[]} & := & [] \\ \underline{\Gamma, x : A} & := & \underline{\Gamma}, \underline{A}_{\phi_\Gamma} \end{array}$$

Again, the following theorems are natural consequences of the fact that the rules of Fig. 9.11 and Fig. 9.12 are respectively those of Fig. 9.1 and Fig. 9.6 but with de Bruijn indices.

Theorem 253 (Preservation of typing: proof synthesis)
Suppose that Γ *is decent.*

1. *If* $\Gamma \vdash_{PS} M : A$ *then* $\underline{\Gamma} \vdash_{\mathsf{PS_{db}}} \underline{M}_{\phi_\Gamma} : \underline{A}_{\phi_\Gamma}$.
2. *If* $\Gamma; l \vdash_{PS} M : A$ *then* $\underline{\Gamma}; \underline{l}_{\phi_\Gamma} \vdash_{\mathsf{PS_{db}}} \underline{M}_{\phi_\Gamma} : \underline{A}_{\phi_\Gamma}$.

Proof: By induction on derivations. □

Theorem 254 (Preservation of typing: type synthesis)

1. *If* $\Gamma \vdash_{\mathsf{TS}} M : A$ *then* $\underline{\Gamma} \vdash_{\mathsf{TS}_{\mathsf{db}}} \underline{M}_{\phi_\Gamma} : \underline{A}_{\phi_\Gamma}$.
2. *If* $\Gamma; l \vdash_{\mathsf{TS}} M : A$ *then* $\underline{\Gamma}; \underline{l}_{\phi_\Gamma} \vdash_{\mathsf{TS}_{\mathsf{db}}} \underline{M}_{\phi_\Gamma} : \underline{A}_{\phi_\Gamma}$.
3. *If* $\Gamma \ \mathsf{wf}_{\mathsf{TS}}$ *then* $\underline{\Gamma} \ \mathsf{wf}_{\mathsf{TS}_{\mathsf{db}}}$.

Proof: By induction on derivations. □

9.5.3 Composing the encodings

Remark 255 Note that if the function f from variables to natural numbers is co-finite and onto then f^{-1} is co-partial, and if the function f from natural numbers to variables is co-partial then f^{-1} is co-finite and onto.

Theorem 256 (Composition)

1. *Suppose that* f *is a function from variables to natural numbers that is co-finite and onto.*

   If $\mathsf{FV}(M) \subseteq \mathsf{Dom}(f)$, $\overline{\underline{M}_f}^{f^{-1}} = \Downarrow(M)$, *and if* $\mathsf{FV}(l) \subseteq \mathsf{Dom}(f)$, $\overline{\underline{l}_f}^{f^{-1}} = \Downarrow(l)$.

2. *Suppose that* f *is a co-partial function from natural numbers to variables.*

   $\overline{\underline{M}^f}_{f^{-1}} = M$ *and* $\bar{\underline{l}}^f_{f^{-1}} = l$.

Proof: By induction on M, l. □

Theorem 257 (Reflections between PTSC and $\mathsf{PTSC}_{\mathsf{db}}$)
Suppose that f *is a co-partial function from natural numbers to variables.*

The mappings $\underline{(\_)}_{f^{-1}}$ *and* $\overline{(\_)}^f$ *form a reflection in* PTSC *of* $\mathsf{PTSC}_{\mathsf{db}}$ *(more precisely, in the fragment of those terms whose free variables are in the image of* f*).*

Proof: This is the conjunction of Theorem 251, Theorem 244 and Theorem 256. □

Corollary 258 (Confluence of $\mathsf{PTSC}_{\mathsf{db}}$) $\longrightarrow_{\beta_{\mathsf{db}}\chi_{\mathsf{db}}}$ *is confluent.*

Proof: By Theorem 257 and Theorem 5. □

Finally we establish the equivalence of typing, but for that we need the following lemma:

Lemma 259 *Suppose that f is a co-partial function from natural numbers to variables and $\overline{\Gamma}^f$ is decent.*

1. *$\phi_{\overline{\Gamma}^f}$ is the restriction of f^{-1} to $\mathsf{Dom}(\overline{\Gamma}^f)$.*
2. *$\underline{\overline{\Gamma}^f} = \Gamma$.*

Proof: Each point is proved by induction on the length of Γ. □

Corollary 260 (Equivalence of typing: proof synthesis)

1. *$\Gamma \vdash_{\mathsf{PS_{db}}} M : A$ if and only if $\overline{\Gamma}^f \vdash_{\mathsf{PS}} \overline{M}^f : \overline{A}^f$*
2. *$\Gamma; l \vdash_{\mathsf{PS_{db}}} M : A$ if and only if $\overline{\Gamma}^f; \overline{l}^f \vdash_{\mathsf{PS}} \overline{M}^f : \overline{A}^f$*

Proof: Straightforward consequence of Theorems 246 and 253, using Theorems 256 and Lemma 259. □

Corollary 261 (Equivalence of typing: type synthesis)

1. *$\Gamma \vdash_{\mathsf{TS_{db}}} M : A$ if and only if $\overline{\Gamma}^f \vdash_{\mathsf{TS}} \overline{M}^f : \overline{A}^f$*
2. *$\Gamma; l \vdash_{\mathsf{TS_{db}}} M : A$ if and only if $\overline{\Gamma}^f; \overline{l}^f \vdash_{\mathsf{TS}} \overline{M}^f : \overline{A}^f$*
3. *Γ $\mathsf{wf_{TS}}$ if and only if $\overline{\Gamma}^f$ $\mathsf{wf_{TS}}$*

Proof: Straightforward consequence of Theorems 247 and 247, using Theorems 256 and Lemma 259. □

Conclusion

In this chapter we have defined variants of PTSC for proof synthesis and type synthesis, with corresponding implementation-friendly versions using de Bruijn indices. We developed a version of PTSC without explicit substitutions along the way, because the typing rules of the latter were problematic for both type synthesis and the versions with de Bruijn indices. However in Chapter 8 we still presented PTSC with explicit substitutions for their theoretical interest, because they are closer to a step by step Gentzen-style cut-elimination procedure and because we had a solution for the typing rules of explicit substitutions which, inelegant though it might be, made the typing system satisfy basic required properties such as subject reduction.

With type synthesis we have recalled issues such as *Expansion Postponement* and typing normal forms without cuts, in a framework similar to that of [GR03a] (but with proof-terms *representing* sequent calculus derivations).

For all PTSC, for instance all corners of Barendregt's Cube, we now have an elegant theoretical framework for proof synthesis: We have shown how to deal with conversion rules so that basic proof-search tactics are simply the bottom-up application of the typing rules. Proof-search tactics in natural deduction simply depart from the simple bottom-up application of the typing rules, and consequently their readability and usage is more complex. Just as in propositional logic [DP99a], sequent calculi can be a useful theoretical approach to study and design those tactics, in the hope to improve semi-automated reasoning in proof-assistants such as Coq or Lego.

As mentioned in the conclusion of Chapter 8, further work includes dealing with inductive types such as those used in Coq. Their specific proof-search tactics should also clearly appear in sequent calculus.

Also, a version of PTSC with de Bruijn indices, optimised for proof synthesis with the extension of higher-order variables to postpone recursive calls of the procedure, is left as further work. This would be the version closest to that of [Mun01] in natural deduction.

Indeed, adapting some ideas of this approach to our framework of sequent calculus is left as one of most interesting directions for further work: First, the motives of [Mun01] are very similar to ours regarding proof synthesis, and indeed it proposes an algorithm for proof synthesis similar to that of [Dow93], also in natural deduction. Second, mention is made of $\overline{\lambda}$, the calculus on which PTSC are based, since its ability to distinguish a constructor for head variables and a constructor for head redexes seems to be extremely relevant for the approach of [Mun01] to the problem of defining a strongly normalising and confluent calculus with explicit substitutions and meta-variables. In fact, [Mun01] develops a theory in natural deduction that emulates this particular feature of $\overline{\lambda}$.

Part III

Towards Classical Logic

Chapter 10

Classical F_ω in sequent calculus

In Part II we have used the paradigm of the Curry-Howard correspondence for sequent calculus to turn *Pure Type Systems* (PTS [Bar91, Bar92]) into *Pure Type Sequent Calculi* (PTSC).

Noticing the elegance of sequent calculus to display the symmetries of classical logic, it is then tempting to try and build classical versions of powerful type theories (PTS and perhaps more elegantly the corresponding PTSC, but also Martin-Löf type theories [ML84]). Approaches to this task (in natural deduction) can be found in [Ste00], in a framework *à la* Martin-Löf, and in [BHS97] (but with a confluent restriction of the reductions of classical logic).

Approaches to the Curry-Howard correspondence for classical logic converge towards the idea of programs equipped with some notion of control [Par92, BB96, Urb00, Sel01, CH00]. The general notion of reduction/computation is non-confluent but there are possible ways to restrict reductions and thus recover confluence.[1]

Intuitionistic type theories, however, exploit the fact that predicates are pure functions, which, when fully applied, give rise to formulae with logical meanings. The Curry-Howard correspondence in intuitionistic logic can then describe these pure functions as the inhabitants of implicative types in a higher type layer (often called the layer of kinds).

On the other hand, inhabitants of implicative types in classical logic can be much wilder than pure functions (owing to the aforementioned notion of control), so it is not clear what meaning could be given to those simili-predicates, built from classical inhabitants of implicative types, and whose reductions may not even be confluent. However, such an issue is problematic only in the layer of types, a.k.a the *upper layer*, which various type theories "cleanly" separate from the layer of terms, a.k.a the *lower layer*.

In this chapter, most of which appeared in [LM06], we show that it is perfectly

[1] Two such canonical ways are related to CBV and CBN, with associated semantics given by CPS-translations, which correspond to the usual encodings of classical logic into intuitionistic logic known as "not-not"-translations.

safe to have cohabiting layers with different logics, provided that the upper layer does not depend on the lower layer, i.e. that the system has no dependent types. For that we chose to tackle system F_ω [Gir72], which can be seen as the PTS given by the sets $\mathcal{S} = \{\star, \Box\}$, $\mathcal{A} = \{(\star, \Box)\}$, and $\mathcal{R} = \{(\star, \star), (\Box, \star), (\Box, \Box)\}$. We present here a version of it called $F_\omega^{\mathcal{C}}$ that is classical in the following sense:

The upper layer is purely functional, i.e. intuitionistic, but for those objects of the layer that represent formulae, we have a notion of provability, with proof derivations and proof-terms in the lower layer, that is classical instead of intuitionistic.

The motivation for the choice of tackling F_ω is threefold:

- System F_ω is indeed the most powerful corner of Barendregt's Cube without dependent types [Bar91, Bar92].
- System F and the simply-typed λ-calculus also cleanly separate the lower layer from the upper layer, but the latter is trivial as no computation happens there, in contrast to System F_ω which features computation in both layers, both strongly normalising.
- The version $F_\omega^{\mathcal{C}}$ with a classical lower layer, in contrast to the intuitionistic one, features two *different* notions of computation (one intuitionistic and confluent, the other one classical and non-confluent), also both strongly normalising. Hence, $F_\omega^{\mathcal{C}}$ represents an excellent opportunity to express and compare two techniques to prove strong normalisation that are based on the method of reducibility of Tait and Girard [Gir72] and that look very similar, and to raise a conjecture about one technique not capturing the other.

Furthermore, in contrast to [LM06] where we presented the upper layer of $F_\omega^{\mathcal{C}}$ in natural deduction (i.e. as the simply-typed λ-calculus extended with constants for logical connectives), we develop here $F_\omega^{\mathcal{C}}$ entirely in sequent calculus, for the purpose of homogeneity, both with the lower layer (the proofs of strong normalisation of the two layers become even more similar than in [LM06]) and with Part II of this dissertation. More precisely, we base $F_\omega^{\mathcal{C}}$ on the corresponding PTSC given by the above sets $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$ (in fact, its version with implicit substitutions developed in Chapter 9), extending the lower layer to classical logic.

The novelty of the strong normalisation of the upper layer (section 10.2.1) is twofold:

- It rephrases the reducibility method [Gir72] with the concepts and terminology of *orthogonality*, which provides a high level of abstraction and potential for modularity, but has a sparse literature (which includes [MV05]).
- Also, the proof that appeared in [LM06], due to A. Miquel, was for the version of the upper layer in natural deduction. Here we adapt it to the framework of sequent calculus, namely LJT, which types the $\overline{\lambda}$-calculus (here in a

version with implicit substitutions). This also makes the result itself, independently from the proof technique, slightly newer. However let us mention the proofs in [DU03, Kik04a] (the latter also using the reducibility method) for $\overline{\lambda}$ with explicit substitutions, but without the logical constructions (for conjunction, disjunction, an quantifiers) and the concepts of *duality*, which arise from the fact that we want to express in this layer some formulae to be proved in classical logic.

The technique for the strong normalisation of the lower layer (section 10.2.2) adapts Barbanera and Berardi's method based on a symmetric notion of reducibility candidate [BB96] and a fixpoint construction. Previous works (e.g. [Pol04a, DGLL05]) adapt it to prove the strong normalisation of various sequent calculi, but (to our knowledge) not pushing it to such a typing system as that of $F_\omega^{\mathcal{C}}$ (with a notion of computation on types). Note that we also introduce the notion of orthogonality in the proof technique (to elegantly express it and compare it to the proof for the upper layer).

On the whole, the technical development of $F_\omega^{\mathcal{C}}$ works in fact without any surprise. Difficulties would come with dependent types (the only feature of Barendregt's Cube missing here), precisely because they would pollute the layer of types with non-confluence and unclear semantics.

Finally, the main purpose of presenting together the two proof techniques described above is to express them whilst pointing out similarities, and to examine whether or not the concepts of the symmetric candidates method can be captured by the concept of orthogonality. We conjecture that it cannot.

Section 10.1 introduces $F_\omega^{\mathcal{C}}$, section 10.2 establishes the strong normalisation of the two layers, concluding with a comparative discussion and the aforementioned conjecture, and section 10.3 establishes some logical properties of $F_\omega^{\mathcal{C}}$, such as consistency and the fact that it encodes F_ω equipped with the axiom of elimination of double negation.

10.1 The calculus $F_\omega^{\mathcal{C}}$

10.1.1 Syntax

Definition 145 (Grammar of $F_\omega^{\mathcal{C}}$) $F_\omega^{\mathcal{C}}$ distinguishes five syntactic categories: *kinds*, *type constructors*, *type lists*, *terms* and *programs*, presented in Fig. 10.1, where $\alpha, \beta, \ldots$ range over a set Var^T of *constructor variables* and $x, y, \ldots$ range over a set Var of *term variables*.

The constructions $\mu x^A.p$ bind x in p, $\lambda x^A y^B.p$ bind x and y in p. The constructions $\forall \alpha^K.B$, $\exists \alpha^K.B$ and $\lambda \alpha^K.B$ bind α in B, including those in a sub-term of the form $\overline{\alpha}\ l$. In other words, constructor variables form a syntactic category of their own (as well as term variables), and $\alpha\ l$ and $\overline{\alpha}\ l$ are two different constructs using the *same* variable.

Kinds	K, K'	::=	$\star \mid K \to K'$
Type constructors	$A, B, C, \ldots$	::=	$\lambda\alpha^K.B \mid \alpha\ l \mid \overline{\alpha}\ l \mid A\ l$
		$\mid$	$A \wedge B \mid A \vee B$
		$\mid$	$\forall\alpha^K.A \mid \exists\alpha^K.A$
Type lists	$l, l', \ldots$	::=	$[] \mid A \cdot l \mid l@l'$
Terms	$t, u, v, \ldots$	::=	$x \mid \mu x^A.p$
		$\mid$	$\langle t, u\rangle \mid \lambda x^A y^B.p$
		$\mid$	$\Lambda\alpha^K.t \mid \langle A, t\rangle$
Programs	p	::=	$\{t \mid u\}$

Figure 10.1: Grammar of $F_\omega^{\mathcal{C}}$

We write $\mathsf{FV}(t)$ (resp. $\mathsf{FV}(p)$) for the free term variables of t (resp. p), and $\mathsf{FV}_{\mathsf{Var}^T}(t)$ (resp. $\mathsf{FV}_{\mathsf{Var}^T}(p)$) for its free constructor variables.

Kinds, which are exactly the same as in system F_ω [Gir72, BG01], are a system of simple types for type constructors and type lists (we use the word 'kind' to distinguish kinds from the types which appear at the level of type constructors). The basic kind $\star$, denoted $\Box_0$ in the description of Barendregt's Cube in Chapter 8, is the kind of *types*, that is, the kind of all type constructors which which terms can be typed —a.k.a *propositions* through the Curry-Howard correspondence.

The *upper layer* of $F_\omega^{\mathcal{C}}$ is that of type constructors and type lists. In [LM06] we used a syntax based on the simply-typed λ-calculus, but here we develop a version based on $\overline{\lambda}$ [Her95] whose typing system is the sequent calculus LJT, because we are interested in expressing type theory in sequent calculus (indeed, in Part II we have developed Pure Type Sequent Calculi based on $\overline{\lambda}$ as well). For this layer we extend $\overline{\lambda}$ with two binary constructs $A \wedge B$ (conjunction), $A \vee B$ (disjunction) and two binding constructs $\forall\alpha^K.A$ and $\exists\alpha^K.A$ to represent universal and existential quantification. There is no primitive implication in the system. The constructs $\alpha\ l, \overline{\alpha}\ l, A\ l$ are called *applications* and l represents the list of arguments of the "functions" $\alpha, \overline{\alpha}$, and A, respectively. More intuition about the functional meaning of $\overline{\lambda}$ can be found in Chapter 8. Hence, the version of $F_\omega^{\mathcal{C}}$ entirely in sequent calculus is thus the particular PTSC given by the sets $\mathcal{S} = \{\star, \Box\}$, $\mathcal{A} = \{(\star, \Box)\}$, and $\mathcal{R} = \{(\star, \star), (\Box, \star), (\Box, \Box)\}$ whose lower layer (see below) is extended to classical logic.

Definition 146 (Duality) The *duality* function $A \mapsto A^\perp$ on all type constructors is defined in Fig. 10.2 by induction on A and extends, via de Morgan's laws, the intrinsic duality between the two constructs $\alpha\ l$ and $\overline{\alpha}\ l$.

Note the definition of duality for λ-abstraction and applications where the list

$$
\begin{array}{llll}
(\alpha\ l)^{\perp} & := \overline{\alpha}\ l & (\overline{\alpha}\ l)^{\perp} & := \alpha\ l \\
(A \wedge B)^{\perp} & := A^{\perp} \vee B^{\perp} & (A \vee B)^{\perp} & := A^{\perp} \wedge B^{\perp} \\
(\forall \alpha^K.B)^{\perp} & := \exists \alpha^K.B^{\perp} & (\exists \alpha^K.B)^{\perp} & := \forall \alpha^K.B^{\perp} \\
(\lambda \alpha^K.B)^{\perp} & := \lambda \alpha^K.B^{\perp} & (A\ l)^{\perp} & := A^{\perp}\ l
\end{array}
$$

Figure 10.2: Duality

of arguments l is unaffected. The notation $A^{\perp}$ is not only meaningful for types (that is, type constructors of kind $\star$), but it is defined for all type constructors.

Remark 262 With duality extended to all type constructors we can define implication $A \to B$ as $(A^{\perp}) \vee B$.

Definition 147 (Substitution) As in Definition 130, we have to define the notion of substitution (Definition 43 is not very useful here, since constructor variables form a syntactic category of their own), but this time it must take into account the duality on variables:

The computation rules of duality are incorporated into the calculus by extending the definition of substitution to sub-terms of the form $\overline{\alpha}\ l$, as shown in Fig. 10.3.

$$
\begin{array}{lcl}
\left\{{}^{C}\!/_{\beta}\right\}(\lambda \alpha^K.A) & := & \lambda \alpha^K.\left\{{}^{C}\!/_{\beta}\right\}A \\
\left\{{}^{C}\!/_{\beta}\right\}(\beta\ l) & := & C\ \left\{{}^{C}\!/_{\beta}\right\}l \\
\left\{{}^{C}\!/_{\beta}\right\}(\alpha\ l) & := & \alpha\ \left\{{}^{C}\!/_{\beta}\right\}l \\
\left\{{}^{C}\!/_{\beta}\right\}(\overline{\beta}\ l) & := & C^{\perp}\ \left\{{}^{C}\!/_{\beta}\right\}l \\
\left\{{}^{C}\!/_{\beta}\right\}(\overline{\alpha}\ l) & := & \overline{\alpha}\ \left\{{}^{C}\!/_{\beta}\right\}l \\
\left\{{}^{C}\!/_{\beta}\right\}(A\ l) & := & \left\{{}^{C}\!/_{\beta}\right\}A\ \left\{{}^{C}\!/_{\beta}\right\}l \\
\left\{{}^{C}\!/_{\beta}\right\}(A \wedge B) & := & \left\{{}^{C}\!/_{\beta}\right\}A \wedge \left\{{}^{C}\!/_{\beta}\right\}B \\
\left\{{}^{C}\!/_{\beta}\right\}(A \vee B) & := & \left\{{}^{C}\!/_{\beta}\right\}A \vee \left\{{}^{C}\!/_{\beta}\right\}B \\
\left\{{}^{C}\!/_{\beta}\right\}(\forall \alpha^K.A) & := & \forall \alpha^K.\left\{{}^{C}\!/_{\beta}\right\}A \\
\left\{{}^{C}\!/_{\beta}\right\}(\exists \alpha^K.A) & := & \exists \alpha^K.\left\{{}^{C}\!/_{\beta}\right\}A \\
\hline
\left\{{}^{C}\!/_{\beta}\right\}[] & := & [] \\
\left\{{}^{C}\!/_{\beta}\right\}(A \cdot l) & := & (\left\{{}^{C}\!/_{\beta}\right\}A) \cdot (\left\{{}^{C}\!/_{\beta}\right\}l) \\
\left\{{}^{C}\!/_{\beta}\right\}(l@l') & := & (\left\{{}^{C}\!/_{\beta}\right\}l)@(\left\{{}^{C}\!/_{\beta}\right\}l')
\end{array}
$$

Figure 10.3: Substitution in the upper layer

Remark 263 $(\left\{{}^{B}\!/_{\alpha}\right\}A)^{\perp} = \left\{{}^{B}\!/_{\alpha}\right\}A^{\perp}$.

As with the notion of implicit substitution from Definition 43, the substitution lemma holds (Lemma 40):

Lemma 264 (Substitution Lemma) $\left\{{}^{C}\!/_{\beta}\right\}\left\{{}^{B}\!/_{\alpha}\right\}A = \left\{{}^{\left\{{}^{C}\!/_{\beta}\right\}B}\!/_{\alpha}\right\}\left\{{}^{C}\!/_{\beta}\right\}A$ *and* $\left\{{}^{C}\!/_{\beta}\right\}\left\{{}^{B}\!/_{\alpha}\right\}l = \left\{{}^{\left\{{}^{C}\!/_{\beta}\right\}B}\!/_{\alpha}\right\}\left\{{}^{C}\!/_{\beta}\right\}l$

Proof: By induction on A, l, using Remark 263. □

The *lower layer* of $F_\omega^{\mathcal{C}}$ is that of terms and programs. These are basically the terms of the symmetric λ-calculus [BB96], with the difference that connectives are treated multiplicatively. In particular, disjunction is treated with a double binder written $\lambda x^A y^B.p$. On the other hand, conjunction is proved as usual, using the pairing construct written $\langle t, u\rangle$. Programs are built by making two terms t and u interact using a construct written $\{t \mid u\}$, where each term can be understood as the evaluation context of the other term. We assume that this construction is symmetric, that is, that $\{t \mid u\}$ and $\{u \mid t\}$ denote the same program. Henceforth, terms and programs are considered up to this equality.

10.1.2 Reduction and typing for the upper layer

Definition 148 (Reduction system of the upper layer) The reduction system of the upper layer of $F_\omega^{\mathcal{C}}$ is presented in Fig. 10.4. Note the rule B' that refers to the rule of $\overline{\lambda}$ that uses an implicit substitution rather than an explicit one (see e.g. Chapter 9), and the rule $\mathsf{B2}^\perp$ that is dual to $\mathsf{B2}$ and whose presence justifies to name the system x' instead of just x.

$$
\begin{array}{llll}
\mathsf{B}' & (\lambda\alpha^K.A)\ (B \cdot l) & \longrightarrow & (\left\{{}^{B}\!/_{\alpha}\right\}A)\ l \\
\mathsf{x}' & \left\{\begin{array}{lll}
\mathsf{B1} & A\ [] & \longrightarrow A \\
\mathsf{B2} & (\alpha\ l)\ l' & \longrightarrow \alpha\ (l@l') \\
\mathsf{B2}^\perp & (\overline{\alpha}\ l)\ l' & \longrightarrow \overline{\alpha}\ (l@l') \\
\mathsf{B3} & (A\ l)\ l' & \longrightarrow A\ (l@l') \\
\mathsf{A1} & (A \cdot l')@l & \longrightarrow A \cdot (l'@l) \\
\mathsf{A2} & []@l & \longrightarrow l \\
\mathsf{A3} & (l@l')@l'' & \longrightarrow l@(l'@l'')
\end{array}\right.
\end{array}
$$

Figure 10.4: Reduction system of the upper layer

By defining *negation* as the closed term $\neg := \lambda\alpha^\star.\overline{\alpha}$ (a function of kind $\star \to \star$), we get de Morgan's equalities for free:

$$
\begin{array}{llll}
\neg(A \wedge B) & \longleftrightarrow^*_{\mathsf{B}'\mathsf{x}'} & \neg A \vee \neg B & \qquad \neg(A \vee B) \longleftrightarrow^*_{\mathsf{B}'\mathsf{x}'} \neg A \wedge \neg B \\
\neg(\forall\alpha^K.B) & \longleftrightarrow^*_{\mathsf{B}'\mathsf{x}'} & \exists\alpha^K.\neg B & \qquad \neg(\exists\alpha^K.B) \longleftrightarrow^*_{\mathsf{B}'\mathsf{x}'} \forall\alpha^K.\neg B
\end{array}
$$

Theorem 265 (Confluence of the upper layer)
System $\mathsf{B}'\mathsf{x}'$ of the upper layer is confluent.

Proof: A reflection of the traditional λ-calculus as in the proof of Corollary 207 seems difficult here because of duality, but we can apply the method of parallel reduction following Tait and Martin-Löf [Bar84]. □

Definition 149 (Typing of the upper layer)

- *Signatures* are Var^T-environments, with kinds as the typing category of Var^T (see Definition 61).

- The typing system of the upper layer manipulates two kinds of sequent, those of the form $\Sigma \vdash A : K$ for type constructors and those of the form $\Sigma; K' \vdash l : K$ for type lists. The typing system is presented in Fig. 10.5. Derivability in this system of the two kinds of sequents is denoted $\Sigma \vdash_{F^{\mathcal{C}}_{\omega}} A : K$ and $\Sigma; K' \vdash_{F^{\mathcal{C}}_{\omega}} l : K$, respectively.

$$\frac{}{\Sigma; K \vdash [] : K} \qquad \frac{\Sigma \vdash A : K_1 \quad \Sigma; K_2 \vdash l : K_3}{\Sigma; K_1 \to K_2 \vdash A \cdot l : K_3}$$

$$\frac{\Sigma; K_1 \vdash l : K_2 \quad \Sigma; K_2 \vdash l' : K_3}{\Sigma; K_1 \vdash l@l' : K_3}$$

$$\frac{(\alpha : K) \in \Sigma \quad \Sigma; K \vdash l : K'}{\Sigma \vdash \alpha\, l : K'} \qquad \frac{(\alpha : K) \in \Sigma \quad \Sigma; K \vdash l : K'}{\Sigma \vdash \overline{\alpha}\, l : K'}$$

$$\frac{\Sigma, \alpha : K \vdash B : K'}{\Sigma \vdash \lambda\alpha^K.B : K \to K'} \qquad \frac{\Sigma \vdash A : K \qquad \Sigma; K \vdash l : K'}{\Sigma \vdash A\, l : K'}$$

$$\frac{\Sigma \vdash A : \star \qquad \Sigma \vdash B : \star}{\Sigma \vdash A \wedge B : \star} \qquad \frac{\Sigma \vdash A : \star \qquad \Sigma \vdash B : \star}{\Sigma \vdash A \vee B : \star}$$

$$\frac{\Sigma, \alpha : K \vdash B : \star}{\Sigma \vdash \forall\alpha^K.B : \star} \qquad \frac{\Sigma, \alpha : K \vdash B : \star}{\Sigma \vdash \exists\alpha^K.B : \star}$$

Figure 10.5: Typing rules for type constructors

Remark 266 The following rules are admissible (all of them apart the last two are height-preserving admissible):

$$\frac{\Sigma \vdash A : K}{\Sigma, \alpha : K' \vdash A : K} \qquad \frac{\Sigma; K_2 \vdash l : K_1}{\Sigma, \alpha : K'; K_2 \vdash l : K_1}$$

$$\frac{\Sigma \vdash A : K}{\Sigma \vdash A^{\perp} : K}$$

$$\frac{\Sigma \vdash A : K \quad \Sigma, \alpha : K \vdash B : K'}{\Sigma \vdash \left\{{}^{A}\!/_{\alpha}\right\} B : K'} \qquad \frac{\Sigma \vdash A : K \quad \Sigma, \alpha : K; K_1 \vdash l : K_2}{\Sigma; K_1 \vdash \left\{{}^{A}\!/_{\alpha}\right\} l : K_2}$$

The typing system for type constructors and type lists satisfies the following property:

Theorem 267 (Subject reduction)

1. *If* $\Sigma \vdash_{F^{\mathcal{C}}_{\omega}} A : K$ *and if* $A \longrightarrow_{B'x'} A'$, *then* $\Sigma \vdash_{F^{\mathcal{C}}_{\omega}} A' : K$.
2. *If* $\Sigma; K' \vdash_{F^{\mathcal{C}}_{\omega}} l : K$ *and if* $l \longrightarrow_{B'x'} l'$, *then* $\Sigma; K' \vdash_{F^{\mathcal{C}}_{\omega}} l' : K$.

Proof: As in Chapter 8, by induction on derivations and case analysis, using Remark 266. □

10.1.3 Reduction and typing for the lower layer

Definition 150 (Reduction system of the lower layer) The reduction system of the lower layer of $F^{\mathcal{C}}_{\omega}$, presented in Fig. 10.6, applies on programs, but the contextual closure equip programs and terms with a reduction relation. Recall that the programs $\{t \mid u\}$ and $\{u \mid t\}$ are identified, so we consider the reduction relation *modulo* the congruence defined by this identity and we denote it $\longrightarrow_{F^{\mathcal{C}}_{\omega}}$.

μ	$\{\mu x^A.p \mid t\}$	$\longrightarrow$	$\left\{{}^{t}\!/_{x}\right\} p$
$\wedge\vee_l$	$\{\langle t_1, t_2\rangle \mid \lambda x_1^A x_2^B.p\}$	$\longrightarrow$	$\{t_1 \mid \mu x_1^A.\{t_2 \mid \mu x_2^B.p\}\}$
$\wedge\vee_r$	or	$\longrightarrow$	$\{t_2 \mid \mu x_2^B.\{t_1 \mid \mu x_1^A.p\}\}$
$\forall\exists$	$\{\Lambda\alpha^K.t \mid \langle A, u\rangle\}$	$\longrightarrow$	$\{\left\{{}^{A}\!/_{\alpha}\right\} t \mid u\}$

Figure 10.6: Reduction system of the lower layer

As in Barbanera and Berardi's symmetric λ-calculus [BB96] or in Curien and Herbelin's $\overline{\lambda}\mu\widetilde{\mu}$-calculus [CH00], the critical pair

$$\begin{array}{ccc} & \{\mu x^A.p \mid \mu y^{A'}.q\} & \\ \swarrow & & \searrow \\ \left\{{}^{\mu y^{A'}.q}\!/_{x}\right\} p & & \left\{{}^{\mu x^{A}.p}\!/_{y}\right\} q \end{array}$$

cannot be joined, and in fact reduction is not confluent in general in this layer (see Example 13 below).

Definition 151 (Typing of the lower layer)

- Environments are here environments for term variables, with type constructors as the typing category (see Definition 61).

- Since the type constructors that appear in an environment may depend on constructor variables, each environment only makes sense in a given signature. In what follows, we say that an environment Γ is *well-formed* in a signature Σ, denoted $\mathsf{wf}_\Sigma(\Gamma)$, if for all declarations $(x : A) \in \Gamma$ we have $\Sigma \vdash_{F^{\mathcal{C}}_{\omega}} A : \star$.

- The typing system of the lower layer manipulates two kinds of sequents, those of the form $\Gamma \vdash^{\Sigma} t : A$ for terms and those of the form $\Gamma \vdash^{\Sigma} p : \diamond$ for programs. The typing system is presented in Fig. 10.7. Derivability in this system of the two kinds of sequents is denoted $\Gamma \vdash^{\Sigma}_{F^{\mathcal{C}}_{\omega}} t : A$ and $\Gamma \vdash^{\Sigma}_{F^{\mathcal{C}}_{\omega}} p : \diamond$, respectively.

$$\frac{\mathsf{wf}_\Sigma(\Gamma) \quad (x:A) \in \Gamma}{\Gamma \vdash^{\Sigma} x : A} \qquad \frac{\Gamma, x:A \vdash^{\Sigma} p : \diamond}{\Gamma \vdash^{\Sigma} \mu x^A.p : A^\perp}$$

$$\frac{\Gamma \vdash^{\Sigma} t : A \qquad \Gamma \vdash^{\Sigma} u : B}{\Gamma \vdash^{\Sigma} \langle t, u\rangle : A \wedge B} \qquad \frac{\Gamma, x:A, y:B \vdash^{\Sigma} p : \diamond}{\Gamma \vdash^{\Sigma} \lambda x^A y^B.p : A^\perp \vee B^\perp}$$

$$\frac{\Gamma \vdash^{\Sigma,\alpha:K} t : B}{\Gamma \vdash^{\Sigma} \Lambda\alpha^K.t : \forall\alpha^K.B} \qquad \frac{\Sigma \vdash_{F^{\mathcal{C}}_{\omega}} A : K \qquad \Gamma \vdash^{\Sigma} u : \{A/\alpha\} B}{\Gamma \vdash^{\Sigma} \langle A, u\rangle : \exists\alpha^K.B}$$

$$\frac{\Gamma \vdash^{\Sigma} t : A \quad \Sigma \vdash_{F^{\mathcal{C}}_{\omega}} A' : \star \quad A \longleftrightarrow^{*}_{\mathsf{B}'\mathsf{x}'} A'}{\Gamma \vdash^{\Sigma} t : A'} \qquad \frac{\Gamma \vdash^{\Sigma} t : A \qquad \Gamma \vdash^{\Sigma} u : A^\perp}{\Gamma \vdash^{\Sigma} \{t \mid u\} : \diamond}$$

Figure 10.7: Typing rules for terms and programs

Remark 268 The following rules are admissible (all of them apart the last two are height-preserving admissible):

$$\frac{\Gamma \vdash^{\Sigma} t : B}{\Gamma \vdash^{\Sigma,\alpha:K} t : B} \qquad \frac{\Gamma \vdash^{\Sigma} p : \diamond}{\Gamma \vdash^{\Sigma,\alpha:K} p : \diamond}$$

$$\frac{\Sigma \vdash_{F^{\mathcal{C}}_\omega} A : K \quad \Gamma \vdash^{\Sigma} t : B}{\Gamma, x : A \vdash^{\Sigma} t : A} \qquad \frac{\Sigma \vdash_{F^{\mathcal{C}}_\omega} A : K \quad \Gamma \vdash^{\Sigma} p : \diamond}{\Gamma, x : A \vdash^{\Sigma} p : \diamond}$$

$$\frac{\Sigma \vdash_{F^{\mathcal{C}}_\omega} A : K \quad \Gamma \vdash^{\Sigma,\alpha:K} t : B}{\{A/\alpha\}\Gamma \vdash^{\Sigma} \{A/\alpha\}t : \{A/\alpha\}B} \qquad \frac{\Sigma \vdash_{F^{\mathcal{C}}_\omega} A : K \quad \Gamma \vdash^{\Sigma,\alpha:K} p : \diamond}{\{A/\alpha\}\Gamma \vdash^{\Sigma} \{A/\alpha\}p : \diamond}$$

$$\frac{\Gamma \vdash^{\Sigma} t : A \quad \Gamma, x : A \vdash^{\Sigma} t' : B}{\Gamma \vdash^{\Sigma} \{t/x\}t' : B} \qquad \frac{\Gamma \vdash^{\Sigma} t : A \quad \Gamma, x : A \vdash^{\Sigma} p : \diamond}{\Gamma \vdash^{\Sigma} \{t/x\}p : \diamond}$$

Again, the type system for proof-terms satisfies the subject reduction property, despite the non-deterministic nature of reduction:

Theorem 269 (Subject reduction)

1. *If* $\Gamma \vdash^{\Sigma}_{F^{\mathcal{C}}_\omega} t : A$ *and* $t \longrightarrow_{F^{\mathcal{C}}_\omega} t'$, *then* $\Gamma \vdash^{\Sigma}_{F^{\mathcal{C}}_\omega} t' : A$.
2. *If* $\Gamma \vdash^{\Sigma}_{F^{\mathcal{C}}_\omega} p : \diamond$ *and* $p \longrightarrow_{F^{\mathcal{C}}_\omega} p'$, *then* $\Gamma \vdash^{\Sigma}_{F^{\mathcal{C}}_\omega} p' : \diamond$.

Proof: By simultaneous induction on derivations, using Remark 268. □

Example 12 (Law of excluded middle) Here is a proof of the Law of excluded middle (we abbreviate α [] as α and $\overline{\alpha}$ [] as $\overline{\alpha}$):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \overline{\alpha}, y : \alpha \vdash^{\alpha:\star} x : \overline{\alpha}} \quad \dfrac{}{x : \overline{\alpha}, y : \alpha \vdash^{\alpha:\star} y : \alpha}}{x : \overline{\alpha}, y : \alpha \vdash^{\alpha:\star} \{x \mid y\} : \diamond}}{\vdash^{\alpha:\star} \lambda x^{\overline{\alpha}} y^{\alpha}.\{x \mid y\} : \alpha \vee \overline{\alpha}}}{\vdash \Lambda\alpha^{\star}.\lambda x^{\overline{\alpha}} y^{\alpha}.\{x \mid y\} : \forall\alpha^{\star}.\alpha \vee \overline{\alpha}}$$

Example 13 (Lafont's example) Here is Lafont's example of non-confluence (again, we abbreviate α [] as α and $\overline{\alpha}$ [] as $\overline{\alpha}$). Suppose $\Gamma \vdash^{\alpha:\star}_{F^{\mathcal{C}}_\omega} p_1 : \diamond$ and $\Gamma \vdash^{\alpha:\star}_{F^{\mathcal{C}}_\omega} p_2 : \diamond$. With $x \notin \mathsf{FV}(p_1)$ and $y \notin \mathsf{FV}(p_2)$, by weakening we get

$$\dfrac{\dfrac{\dfrac{\Gamma \vdash^{\alpha:\star} p_1 : \diamond}{\Gamma, x : \alpha \vdash^{\alpha:\star} p_1 : \diamond}}{\Gamma \vdash^{\alpha:\star} \mu x^{\alpha}.p_1 : \overline{\alpha}} \quad \dfrac{\dfrac{\Gamma \vdash^{\alpha:\star} p_2 : \diamond}{\Gamma, y : \overline{\alpha} \vdash^{\alpha:\star} p_2 : \diamond}}{\Gamma \vdash^{\alpha:\star} \mu y^{\overline{\alpha}}.p_2 : \alpha}}{\Gamma \vdash^{\alpha:\star} \{\mu x^{\alpha}.p_1 \mid \mu y^{\overline{\alpha}}.p_2\} : \diamond}$$

But $\{\mu x^{\alpha}.p_1 \mid \mu y^{\overline{\alpha}}.p_2\} \longrightarrow_{\mu}^{*} p_1$ or $\{\mu x^{\alpha}.p_1 \mid \mu y^{\overline{\alpha}}.p_2\} \longrightarrow_{\mu}^{*} p_2$. And p_1 and p_2 can be completely different.

Note that, in contrast to Barbanera and Berardi's symmetric λ-calculus, our design choices for the typing rules are such that, by constraining terms and programs to be linear, we get exactly the multiplicative fragment of linear logic [Gir87].

10.2 Strong normalisation

In this section we prove the strong normalisation of the two layers of $F_{\omega}^{\mathcal{C}}$. In both cases the method is based on the reducibility technique of Tait and Girard [Gir72].

This consists in building a strongly normalising model of the calculus. The kinds (resp. types) are interpreted as pairs such as (X, Y) (resp. $(\mathcal{U}, \mathcal{V})$), where X is set of type constructors and Y a set of type lists (resp. $\mathcal{U}$ and $\mathcal{V}$ are both sets of terms[2]).

These pairs of sets are *orthogonal*, in that by combining elements of the two components in a construct typed by a cut we get something strongly normalising. The two components of the pairs above contain the basic constructs that introduce a connective on the right and on the left (resp. that introduce dual connectives). This is sufficient to treat most cases of the induction to prove the soundness theorem (which roughly states that being typed implies being in the model, hence being strongly normalising), but for the other cases we need the property that the interpretation of kinds (resp. types) is *saturated*, so we extend the two components of the pairs into saturated pairs by a completion process.

Now the completion process is precisely where the proofs of strong normalisation of the two layers differ: For the upper layer we simply use a completion by bi-orthogonality and this gives us the desired saturation property. For the lower layer, the completion process is obtained by Barbanera and Berardi's fixpoint construction. We discuss this difference in section 10.2.3.

10.2.1 The upper layer

In this section we prove that all type constructors and type lists that are typable (by kinds) are strongly normalising.

For that, let $\mathsf{SN}_{\mathsf{TC}}^{\mathsf{B}'\mathsf{x}'}$ and $\mathsf{SN}_{\mathsf{TL}}^{\mathsf{B}'\mathsf{x}'}$ be the sets of all strongly normalisable type constructors and type lists, respectively.

[2]This is due to our mono-sided approach to classical sequent calculus, but with a bi-sided approach as in [CH00, Wad03], the two sets of the pair would contain "terms" and "contexts".

Definition 152 (Orthogonality —type constructors/type lists)

- Given a type constructor A and a type list l, we say that A and l are *orthogonal*, written $A \perp l$, if $A\ l \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}$.
- If X and Y are sets of type constructors and type lists, respectively, we say that X and Y are *orthogonal*, written $X \perp Y$, if $\forall A \in X, \forall l \in Y, A \perp l$.
- We define the *orthogonal of* X as $X^\perp := \{l \mid \forall A \in X, A \perp l\}$. Similarly, the *orthogonal of* Y is $Y^\perp := \{A \mid \forall l \in Y, A \perp l\}$.

In general, if $B \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}$ and $l \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TL}}$ we do not have $B \perp l$.

The operation $X \mapsto X^\perp$ satisfies the usual properties of orthogonality:

Remark 270

1. $X \subseteq X'$ entails $X'^\perp \subseteq X^\perp$ (contravariance)
2. $X \subseteq X^{\perp\perp}$ (closure)
3. $X^{\perp\perp\perp} = X^\perp$ (tri-orthogonal)

Definition 153 (Saturation —type constructors/type lists) A pair of sets (X, Y) of type constructors and type lists, respectively, is *saturated* if it satisfies the rules of Fig. 10.8 (in the sense that for each each rule if the premisses hold then so does the conclusion).

$$\frac{\forall l' \in Y, A \perp l@l'}{A\ l \in X}$$

$$\frac{A, B \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}}{A \wedge B \in X} \qquad \frac{A, B \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}}{A \vee B \in X} \qquad \frac{\left\{{}^{B}/_{\alpha}\right\} A \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}}{\forall \alpha^K.A \in X} \qquad \frac{\left\{{}^{B}/_{\alpha}\right\} A \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}}{\exists \alpha^K.A \in X}$$

$$\frac{l \in Y}{[]@l \in Y} \qquad \frac{A \cdot (l@l') \in Y}{(A \cdot l)@l' \in Y} \qquad \frac{l_1@(l_2@l_3) \in Y}{(l_1@l_2)@l_3 \in Y}$$

Figure 10.8: Saturation

Definition 154 (Reducibility candidate) A pair of sets (X, Y) of type constructors and type lists, respectively, is a *reducibility candidate* if the following conditions hold:

- Neither X nor Y is empty.

- $X = Y^{\perp}$
- $Y = X^{\perp}$

Remark 271 Note that if (X, Y) is a reducibility candidate, $X = X^{\perp\perp}$ and $Y = Y^{\perp\perp}$.

Reducibility candidates satisfy the following properties:

Theorem 272 (Properties of reducibility candidates) *For all reducibility candidate (X, Y):*

1. $X \subseteq \mathit{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathit{TC}}$ *and* $Y \subseteq \mathit{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathit{TL}}$;
2. $\alpha\ [] \in X$, $\overline{\alpha}\ [] \in X$ *and* $[] \in Y$;
3. (X, Y) *is saturated.*

Proof:

1. This holds because neither X nor Y is empty.
2. This holds because $(\alpha\ [])\ l$ and $(\overline{\alpha}\ [])\ l$ are in $\mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}$ (resp. $A\ [] \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}$) as soon as $l \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TL}}$ (resp. $A \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}$), which is enforced by point 1).
3. All the rules of Fig. 10.8 are straightforward, except the first one of the upper part and the last one of the lower part, which respectively rely on the following properties:
   (a) If $A\ (l_1@(l_2@l_3)) \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}$ then $A\ ((l_1@l_2)@l_3) \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}$.
   (b) If $A\ (l@l') \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}$ then $(A\ l)\ l' \in \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}$.

□

Definition 155 (Set constructions) We define the following abbreviations:

$$\begin{array}{lcl} \lambda X^K.X' & := & \{\lambda\alpha^K.A \mid \forall B \in X, \{{}^{B}\!/_{\alpha}\}\, A \in X'\} \\ X \cdot Y & := & \{A \cdot l \mid A \in X, l \in Y\} \end{array}$$

Remark 273 Note that if $X' \perp Y$ and $X \subseteq \mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}$ then $\lambda X^K.X' \perp X \cdot Y$.

We now interpret each kind K as a reducibility candidate:

Definition 156 (Interpretation of kinds) The interpretation $[K]$ of a kind K is a reducibility candidate (X, Y) (we write $[K]^+ := X$ and $[K]^- := Y$) defined by induction on K as follows:

$$\begin{array}{lcl} [\star] & := & ((\{\alpha\ []\} \cup \{\overline{\alpha}\ []\})^{\perp\perp}, \{[]\}^{\perp\perp}) \\ [K \to K'] & := & ((\lambda[K]^{+K}.[K']^{+})^{\perp\perp}, ([K]^{+} \cdot [K']^{-})^{\perp\perp}) \end{array}$$

Note that these pairs are indeed reducibility candidates, which is ensured by the bi-orthogonal closures of orthogonal pairs. Indeed we have $(\{\alpha\ []\} \cup \{\overline{\alpha}\ []\}) \perp \{[]\}$ and by induction on K we have $[K]^+ \perp [K]^-$ and then $(\lambda[K]^{+K}.[K']^+) \perp ([K]^+ \cdot [K']^-)$.

Theorem 274 (Soundness)

1. *If* $\alpha_1 : K_1, \ldots, \alpha_n : K_n \vdash_{F_\omega^{\mathcal{C}}} A : K$, *then for all* $A_1 \in [K_1]^+$, ..., $A_n \in [K_n]^+$ *we have*
$$\left\{{}^{A_1,\ldots,A_n}/_{\alpha_1,\ldots,\alpha_n}\right\} A \in [K]^+$$
2. *If* $\alpha_1 : K_1, \ldots, \alpha_n : K_n; K \vdash_{F_\omega^{\mathcal{C}}} l : K'$, *then for all* $A_1 \in [K_1]^+$, ..., $A_n \in [K_n]^+$ *and all* $l' \in [K']^-$ *we have*
$$(\left\{{}^{A_1,\ldots,A_n}/_{\alpha_1,\ldots,\alpha_n}\right\} l)@l' \in [K]^-$$

Proof: By simultaneous induction on derivations, all the ingredients of the steps are in the saturation of $[K]$. □

From this we get:

Corollary 275 (Strong normalisation of the upper layer)

1. *If* $\Sigma \vdash_{F_\omega^{\mathcal{C}}} A : K$, *then* $A \in \mathcal{SN}_{\mathsf{TC}}^{\mathsf{B}'\mathsf{x}'}$.
2. *If* $\Sigma; K \vdash_{F_\omega^{\mathcal{C}}} l : K'$, *then* $A \in \mathcal{SN}_{\mathsf{TC}}^{\mathsf{B}'\mathsf{x}'}$.

Proof: Use point 2 of Theorem 272 to apply Theorem 274 with:
$A_1 := \alpha_1\ []$, ..., $A_n := \alpha_n\ []$ and $l' = []$, and this gives:
$\left\{{}^{A_1,\ldots,A_n}/_{\alpha_1,\ldots,\alpha_n}\right\} A \in [K]^+ \subseteq \mathsf{SN}_{\mathsf{TC}}^{\mathsf{B}'\mathsf{x}'}$ and $(\left\{{}^{A_1,\ldots,A_n}/_{\alpha_1,\ldots,\alpha_n}\right\} l)@l' \in [K]^- \subseteq \mathsf{SN}_{\mathsf{TL}}^{\mathsf{B}'\mathsf{x}'}$.
Then note that $\left\{{}^{A_1,\ldots,A_n}/_{\alpha_1,\ldots,\alpha_n}\right\} A \longrightarrow_{\mathsf{x}'}^* A$ and $(\left\{{}^{A_1,\ldots,A_n}/_{\alpha_1,\ldots,\alpha_n}\right\} l)@l' \longrightarrow_{\mathsf{x}'}^* l@l'$,
so $A \in \mathsf{SN}_{\mathsf{TC}}^{\mathsf{B}'\mathsf{x}'}$ and $l@l' \in \mathsf{SN}_{\mathsf{TL}}^{\mathsf{B}'\mathsf{x}'}$ and then $l \in \mathsf{SN}_{\mathsf{TL}}^{\mathsf{B}'\mathsf{x}'}$. □

10.2.2 The lower layer

This proof is adapted from those of [BB96, Pol04a, DGLL05] for the symmetric λ-calculus [BB96], the $\overline{\lambda}\mu\widetilde{\mu}$-calculus [CH00], and the *dual calculus* [Wad03], respectively. They all use Barbanera and Berardi's *symmetric candidates*, with a fixpoint construct to capture the non-confluence of classical logic.

As usual with the reducibility method we construct a model of the calculus by interpreting types (here, type constructors and type lists) as sets of terms. However, the second-order quantification that appears in System F or F_ω is conveniently interpreted as a set intersection only if terms do not display type annotations. We therefore start by defining such term and programs, i.e. Curry-style terms and programs.

Definition 157 (Curry-style terms and programs) We consider terms and programs without their type annotations, a.k.a. Curry-style terms and programs, whose syntax is the following:

$$\begin{array}{lrcl} \textbf{Curry-style terms} & t, u, v, \ldots & ::= & x \mid \mu x.p \mid \langle t, u\rangle \mid \lambda xy.p \mid \Lambda\_.t \mid \langle \_, t\rangle \\ \textbf{Curry-style programs} & p & ::= & \{t \mid u\} \end{array}$$

The corresponding reduction rules, that are shown in Fig. 10.9, define the reductions $\longrightarrow_{F^{\mathcal{C}}_{\omega}}$ and the set $\mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$ of Curry-style terms and Curry-style programs.

$$\begin{array}{lcl} \{\mu x.p \mid t\} & \longrightarrow & \{{}^{t}\!/_{x}\}p \\ \{\langle t_1, t_2\rangle \mid \lambda x_1 x_2.p\} & \longrightarrow & \{t_1 \mid \mu x_1.\{t_2 \mid \mu x_2 p\}\} \\ & \text{or} & \{t_2 \mid \mu x_2.\{t_1 \mid \mu x_1 p\}\} \\ \{\Lambda\_.t \mid \langle \_, u\rangle\} & \longrightarrow & \{t \mid u\} \end{array}$$

Figure 10.9: Reduction rules without types

Definition 158 (Type-erasure operation) The type-erasure operation from terms (resp. programs) to Curry-style terms (resp. Curry-style programs) is recursively defined in Fig. 10.10:

$$\begin{array}{lcl} \|x\| & := & x \\ \|\langle t, u\rangle\| & := & \langle \|t\|, \|u\|\rangle \\ \|\lambda x^A y^B.p\| & := & \lambda xy.\|p\| \\ \|\mu x^A.p\| & := & \mu x.\|p\| \\ \|\Lambda\alpha^K.t\| & := & \Lambda\_.\|t\| \\ \|\langle A, t\rangle\| & := & \langle \_, \|t\|\rangle \\ \|\{t \mid u\}\| & := & \{\|t\| \mid \|u\|\} \end{array}$$

Figure 10.10: Type-erasure operation

Note that by erasing the types we still keep, in Curry-style programs, a trace of the constructs introducing the $\forall$ and $\exists$ quantifiers. Thus, it is slightly different from the traditional Curry-style polymorphism of system F or F_ω, but this trace turns out to be important in classical logic: if we removed it, we could make some μ-μ critical pair appear that was not present in the original program with type annotations, and one of the two reductions might not satisfy subject reduction.

This is a general problem of polymorphism and classical logic with non-confluent reduction: for instance the spirit of *intersection types* [CD78] (see Definition 84), which represent finite polymorphism, is to give several types to *the same program*, free from any trace of where the typing rules for intersection types have been used in its typing derivation. In that case again, non-confluent reductions of classical logic often fail to satisfy subject reduction.

Lemma 276 (Equivalence of strong normalisation) *Provided all type annotations in a term t are in $\mathsf{SN}^{\mathsf{B}'\mathsf{x}'}_{\mathsf{TC}}$, if $\|t\| \in \mathsf{SN}^{F^{\mathcal{C}}_\omega}$ then $t \in \mathsf{SN}^{F^{\mathcal{C}}_\omega}$.*

Proof: Let $\mathcal{M}(t)$ be the multi-set of all the types appearing in t, equipped with the multi-set reduction based on $\mathsf{B}'\mathsf{x}'$-reduction on types. Every reduction from t is simulated by the lexicographic reduction of the pair $(\|t\|, \mathcal{M}(t))$. □

Definition 159 (Orthogonality —terms)

- We say that that a Curry-style term t is *orthogonal* to a Curry-style term u, written $t \perp u$, if $\{t \mid u\} \in \mathsf{SN}^{F^{\mathcal{C}}_\omega}$.
- We say that that a set $\mathcal{U}$ of Curry-style terms is *orthogonal* to a set $\mathcal{V}$ of Curry-style terms, written $\mathcal{U} \perp \mathcal{V}$, if $\forall t \in \mathcal{U}, \forall u \in \mathcal{V}, t \perp u$.

Remark 277 If $\{{}^{v}\!/_{x}\}t \perp \{{}^{v}\!/_{x}\}u$, then $t \perp u$ and $\mu x.\{t \mid u\} \in \mathsf{SN}$.

Definition 160 (Simplicity) A set $\mathcal{U}$ of Curry-style terms is *simple* if it is non-empty and it contains no Curry-style term of the form $\mu x.p$.

Definition 161 (Saturation —terms) A pair $(\mathcal{U}, \mathcal{V})$ of sets of Curry-style terms is *saturated* if:

- $\mathsf{Var} \subseteq \mathcal{U}$ and $\mathsf{Var} \subseteq \mathcal{V}$
- $\{\mu x.\{t \mid u\} \mid \forall v \in \mathcal{V}, \{{}^{v}\!/_{x}\}t \perp \{{}^{v}\!/_{x}\}u\} \subseteq \mathcal{U}$ and
  $\{\mu x.\{t \mid u\} \mid \forall v \in \mathcal{U}, \{{}^{v}\!/_{x}\}t \perp \{{}^{v}\!/_{x}\}u\} \subseteq \mathcal{V}$.

Definition 162 (The completion function) Whenever $\mathcal{U}$ is simple, we define the following function

$$\Phi_{\mathcal{U}}(\mathcal{V}) = \mathcal{U} \cup \mathsf{Var} \cup \{\mu x.\{t \mid u\} \mid \forall v \in \mathcal{V}, \{{}^{v}\!/_{x}\}t \perp \{{}^{v}\!/_{x}\}u\}$$

Remark 278 For all simple $\mathcal{U}$, $\Phi_{\mathcal{U}}$ is anti-monotone. Hence, for any simple $\mathcal{U}$ and $\mathcal{V}$, $\Phi_{\mathcal{U}} \cdot \Phi_{\mathcal{V}}$ is monotone, so it admits a fixpoint $\mathcal{U}' \supseteq \mathcal{U}$.

Theorem 279 (Existence of saturated extensions)
Assume that $\mathcal{U}$ and $\mathcal{V}$ are simple with $\mathcal{U} \perp \mathcal{V}$.
There exist $\mathcal{U}'$ and $\mathcal{V}'$ such that $\mathcal{U} \subseteq \mathcal{U}'$ and $\mathcal{V} \subseteq \mathcal{V}'$, $(\mathcal{U}', \mathcal{V}')$ is saturated and $\mathcal{U}' \perp \mathcal{V}'$.

Proof: Let $\mathcal{U}'$ be a fixpoint of $\Phi_{\mathcal{U}} \cdot \Phi_{\mathcal{V}}$, and let $\mathcal{V}' = \Phi_{\mathcal{V}}(\mathcal{U}')$. We have

$$\mathcal{U}' = \Phi_{\mathcal{U}}(\mathcal{V}') = \mathcal{U} \cup \mathsf{Var} \cup \{\mu x.\{t \mid u\} \mid \forall v \in \mathcal{V}', \{{}^{v}\!/_{x}\}t \perp \{{}^{v}\!/_{x}\}u\}$$
$$\mathcal{V}' = \Phi_{\mathcal{V}}(\mathcal{U}') = \mathcal{V} \cup \mathsf{Var} \cup \{\mu x.\{t \mid u\} \mid \forall v \in \mathcal{U}', \{{}^{v}\!/_{x}\}t \perp \{{}^{v}\!/_{x}\}u\}$$

It is clearly saturated. We now prove that $\mathcal{U}' \perp \mathcal{V}'$.

Since $\mathcal{U} \perp \mathcal{V}$ and $\mathcal{U}$ and $\mathcal{V}$ are non-empty, we have $\mathcal{U} \subseteq \mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$ and $\mathcal{V} \subseteq \mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$. We also have $\mathsf{Var} \subseteq \mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$. Finally, by Remark 277, we conclude $\mathcal{U}' \subseteq \mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$ and $\mathcal{V}' \subseteq \mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$.

Now assume $u \in \mathcal{U}'$ and $v \in \mathcal{V}'$. We show $u \perp v$ by lexicographical induction in $\mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$.

If $u \in \mathcal{U}$ and $v \in \mathcal{V}$ then $u \perp v$ because $\mathcal{U} \perp \mathcal{V}$. If not, we prove $u \perp v$ by showing that whenever $\{u \mid v\} \longrightarrow p$, then $p \in \mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$.

- If $\{u \mid v\} \longrightarrow \{u' \mid v\}$ or $\{u \mid v\} \longrightarrow \{u \mid v'\}$, the induction hypothesis applies.
- The only other case is $u = \mu x.p$ (resp. $v = \mu x.p$) and $\{u \mid v\} \longrightarrow \{{}^{v}\!/_{x}\}p$ (resp. $\{u \mid v\} \longrightarrow \{{}^{u}\!/_{x}\}p$). But since $u \in \mathcal{U}'$ and $v \in \mathcal{V}'$, we know that $\{{}^{v}\!/_{x}\}p \in \mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$ (resp. $\{{}^{u}\!/_{x}\}p \in \mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$).

□

Now we interpret kinds:

Definition 163 (Interpretation of kinds)

- The interpretation $[\![K]\!]$ of a kind K is defined by induction on K as follows:

$$\begin{array}{lcl} [\![\star]\!] & := & \{(\mathcal{U},\mathcal{V}) \mid \mathcal{U} \perp \mathcal{V} \text{ and } (\mathcal{U},\mathcal{V}) \text{ is saturated}\} \\ [\![K \to K']\!] & := & [\![K']\!]^{[\![K]\!]} \end{array}$$

  where $[\![K']\!]^{[\![K]\!]}$ is simply the set of (total) functions from $[\![K]\!]$ to $[\![K']\!]$.

- Given a pair $p \in [\![\star]\!]$, we write p^+ (resp. p^-) its first (resp. second) component.
- We also define the function $\mathsf{swap}_K : [\![K]\!] \to [\![K]\!]$ by induction on K:

$$\begin{array}{lcl} \mathsf{swap}_{\star}(\mathcal{U},\mathcal{V}) & := & (\mathcal{V},\mathcal{U}) \\ \mathsf{swap}_{K \to K'}(f) & := & \mathsf{swap}_{K'} \circ f \end{array}$$

- Let $\mathsf{swap} : (\bigcup_K [\![K]\!]) \to (\bigcup_K [\![K]\!])$ be the disjoint union of all the swap_K.

Definition 164 (Set constructions) Let $\mathcal{U}$ and $\mathcal{V}$ be sets of Curry-style terms. We set the following definitions:

$$\begin{array}{lcl} \langle \mathcal{U},\mathcal{V} \rangle & := & \{\langle u,v \rangle \mid u \in \mathcal{U}, v \in \mathcal{V}\} \\ \lambda\mathcal{U}\mathcal{V}.\diamond & := & \{\lambda xy.\{t_1 \mid t_2\} \mid \forall u \in \mathcal{U}, \forall v \in \mathcal{V}, (\{{}^{u,v}\!/_{x,y}\}t_1) \perp (\{{}^{u,v}\!/_{x,y}\}t_2)\} \\ \Lambda\_.\mathcal{U} & := & \{\Lambda\_.u \mid u \in \mathcal{U}\} \\ \langle \_,\mathcal{U} \rangle & := & \{\langle \_,u \rangle \mid u \in \mathcal{U}\} \end{array}$$

Remark 280

1. The sets $\langle \mathcal{U}, \mathcal{V} \rangle$, $\lambda \mathcal{U} \mathcal{V}.\diamond$, $\Lambda\_.\mathcal{U}$ and $\langle \_, \mathcal{U} \rangle$ are always simple.

2. If $\mathcal{U} \subseteq \mathsf{SN}^{F^{\mathcal{C}}_\omega}$ and $\mathcal{V} \subseteq \mathsf{SN}^{F^{\mathcal{C}}_\omega}$ then $\langle \mathcal{U}, \mathcal{V} \rangle \perp \lambda \mathcal{U} \mathcal{V}.\diamond$.

3. If $\mathcal{U} \perp \mathcal{V}$ then $\Lambda\_.\mathcal{U} \perp \langle \_, \mathcal{V} \rangle$.

Definition 165 (Compatibility) We say that a mapping $\rho : \mathsf{Var}^T \to \bigcup_K [\![K]\!]$ is *compatible* with Σ if $\forall (\alpha : K) \in \Sigma, \rho(\alpha) \in [\![K]\!]$.

Definition 166 (Interpretation of type constructors) For each ρ compatible with Σ we define the interpretation $[\![\Sigma \vdash A : K]\!]_\rho \in [\![K]\!]$ (resp. $[\![\Sigma; K' \vdash A : K]\!]_\rho \in [\![K]\!]^{[\![K']\!]}$) of a derivable sequent $\Sigma \vdash A : K$ (resp. $\Sigma; K' \vdash A : K$) by induction on its derivation.[3] We sometimes write only $[\![A]\!]_\rho$ (resp. $[\![K', A]\!]_\rho$) when Σ is clear. The inductive definition is presented in Fig. 10.11.

$$
\begin{array}{lcll}
[\![\alpha\ l]\!]_\rho & := & [\![K, l]\!]_\rho(\rho(\alpha)) & \text{if } (\alpha : K) \in \Sigma \\
[\![\overline{\alpha}\ l]\!]_\rho & := & [\![K, l]\!]_\rho(\mathsf{swap}(\rho(\alpha))) & \text{if } (\alpha : K) \in \Sigma \\
[\![A \wedge B]\!]_\rho & := & \text{any saturated } (\mathcal{U}, \mathcal{V}) \text{ such that} & \\
 & & \langle [\![A]\!]^+_\rho, [\![B]\!]^+_\rho \rangle \subseteq \mathcal{U} & \\
 & & \lambda [\![A]\!]^+_\rho [\![B]\!]^+_\rho.\diamond \subseteq \mathcal{V} & \\
 & & \mathcal{U} \perp \mathcal{V} & \\
[\![A \vee B]\!]_\rho & := & \text{any saturated } (\mathcal{U}, \mathcal{V}) \text{ such that} & \\
 & & \lambda [\![A]\!]^-_\rho [\![B]\!]^-_\rho.\diamond \subseteq \mathcal{U} & \\
 & & \langle [\![A]\!]^-_\rho, [\![B]\!]^-_\rho \rangle \subseteq \mathcal{V} & \\
 & & \mathcal{U} \perp \mathcal{V} & \\
[\![\forall \alpha^{K'}.A]\!]_\rho & := & \text{any saturated } (\mathcal{U}, \mathcal{V}) \text{ such that} & \\
 & & \Lambda\_.\bigcap_{h \in [\![K']\!]} [\![A]\!]^+_{\rho, \alpha \mapsto h} \subseteq \mathcal{U} & \\
 & & \langle \_, \bigcup_{h \in [\![K']\!]} [\![A]\!]^-_{\rho, \alpha \mapsto h} \rangle \subseteq \mathcal{V} & \\
 & & \mathcal{U} \perp \mathcal{V} & \\
[\![\exists \alpha^{K'}.A]\!]_\rho & := & \text{any saturated } (\mathcal{U}, \mathcal{V}) \text{ such that} & \\
 & & \langle \_, \bigcup_{h \in [\![K']\!]} [\![A]\!]^+_{\rho, \alpha \mapsto h} \rangle \subseteq \mathcal{U} & \\
 & & \Lambda\_.\bigcap_{h \in [\![K']\!]} [\![A]\!]^-_{\rho, \alpha \mapsto h} \subseteq \mathcal{V} & \\
 & & \mathcal{U} \perp \mathcal{V} & \\
[\![\lambda \alpha^{K'}.A]\!]_\rho & := & h \in [\![K']\!] \mapsto [\![A]\!]_{\rho, \alpha \mapsto h} & \\
\hline
[\![K, []]\!]_\rho & := & \mathsf{Id}_{[\![K]\!]} & \\
[\![K, A \cdot l]\!]_\rho & := & h \in {K_2}^{K_1} \mapsto ([\![K_2, l]\!]_\rho)(h([\![A]\!]_\rho)) & \text{with } K = K_1 \to K_2
\end{array}
$$

Figure 10.11: Interpretation of type constructors

[3] Given Σ and A —and K' in the case of type lists, K is unique, and so is the derivation up to renaming, in sub-derivations, of the constructor variables bound in A.

The soundness of the definition inductively relies on the fact that $[\![A]\!]_\rho \in [\![K]\!]$, ρ keeps being compatible with Σ, and $[\![A]\!]^+_\rho \perp [\![A]\!]^-_\rho$. The existence of the saturated extensions in the case of $A \wedge B$, $A \vee B$, $\forall \alpha^{K'}.A$ and $\exists \alpha^{K'}.A$ is given by Theorem 279.

Remark 281

- Note that $[\![A^\perp]\!]_\rho = \mathsf{swap}[\![A]\!]_\rho$.
- $[\![A]\!]_{\rho,\alpha\mapsto[\![B]\!]_\rho} = [\![\{^B/_\alpha\}A]\!]_\rho$ and $[\![K,l]\!]_{\rho,\alpha\mapsto[\![B]\!]_\rho} = [\![K,\{^B/_\alpha\}l]\!]_\rho$
- If $A \longrightarrow_{\mathsf{B}'\mathsf{x}'} B$ then $[\![A]\!]_\rho = [\![B]\!]_\rho$ and if $l \longrightarrow_{\mathsf{B}'\mathsf{x}'} l'$ then $[\![K,l]\!]_\rho = [\![K,l']\!]_\rho$.
- If $\Sigma \vdash_{F^{\mathcal{C}}_\omega} A:\star$, then $[\![A]\!]_\rho$ is saturated, with $[\![A]\!]^+_\rho \subseteq \mathsf{SN}$ and $[\![A]\!]^-_\rho \subseteq \mathsf{SN}$.

Theorem 282 (Soundness) *If $x_1:A_1,\ldots,x_n:A_n \vdash^\Sigma_{F^{\mathcal{C}}_\omega} t:A$ then for all ρ compatible with Σ, and for all $t_1 \in [\![A_1]\!]^+_\rho$, …, $t_n \in [\![A_n]\!]^+_\rho$ we have:*

$$\{^{t_1,\ldots,t_n}/_{x_1,\ldots,x_n}\}\|t\| \in [\![A]\!]^+_\rho$$

Proof: By induction on the derivation. □

Corollary 283 (Strong normalisation of the lower layer)
If $x_1:A_1,\ldots,x_n:A_n \vdash^\Sigma_{F^{\mathcal{C}}_\omega} t:A$ then $t \in \mathsf{SN}$.

Proof: We first prove that we can find a ρ compatible with Σ (for $\alpha:\star$, take $\rho(\alpha)$ to be any saturated extension of $(\mathsf{Var},\mathsf{Var})$). Then we can apply Theorem 282 and conclude by Lemma 276. □

10.2.3 A conjecture about orthogonality

As mentioned in the introduction of section 10.2, the similarity between the proof of strong normalisation of the upper layer and that of the lower layer is striking.

The first difference between the two proofs is insignificant: For the lower layer we have removed type annotations from the terms of the model, simply because we can thus interpret universal and existential second-order quantification ($\forall$ and $\exists$) with intersections and unions of sets. Although we could have done the same for the upper layer, we did not need to do it (i.e. we kept the kinds annotating type constructors and type lists) since there is no second-order quantifiers in the syntax of kinds.

More importantly, while in the upper layer the saturation of the interpretation of kinds is obtained by a bi-orthogonal completion, it is important to understand why, for the lower layer, we used another notion of completion using fixpoints instead.

The reason is that in general, if the pair $(\mathcal{U}, \mathcal{V})$ is simple and orthogonal, the extension $(\mathcal{U}^{\perp\perp}, \mathcal{V}^{\perp\perp})$ does not seem to be saturated in the sense of Definition 161, while in the upper layer such a completion by bi-orthogonality ensures the corresponding notion of saturation given in Definition 153. Technically, the presence of the μ-μ critical pair makes the proof of Theorem 272.3 difficult or impossible to adapt to the non-confluent case of the lower layer. This lack of saturation is the motivation for the fixpoint construction in the interpretation of types, instead of the bi-orthogonal construction. Hence we set the following conjecture:

Conjecture 284 (Orthogonality does not imply saturation) *Given a simple and orthogonal pair $(\mathcal{U}, \mathcal{V})$, the bi-orthogonal extension $(\mathcal{U}^{\perp\perp}, \mathcal{V}^{\perp\perp})$ need not be saturated.*

Ongoing work with A. Miquel is about answering this conjecture. If the pair $(\mathcal{U}^{\perp\perp}, \mathcal{V}^{\perp\perp})$ were always saturated, then the fixpoint construction would be unnecessary. But if the conjecture is true, it would be interesting to investigate whether the choice of a more sophisticated relation of orthogonality could solve the problem and replace the fixpoint construction, but I doubt it.

Note that [DN05b] already notices that "the technique using the usual candidates of reducibility does not work" for the non-confluent reductions of classical logic (that they express in the $\lambda\mu$-calculus [Par92]). However, their counter-examples translate in our setting to the fact that even if t and $\{{}^{t}\!/_{x}\}p$ are in $\mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$, $\{\mu x.p \mid t\}$ need not be in $\mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$. This is quite direct, but the method of completion by bi-orthogonality is more subtle: Indeed, Conjecture 284 states that a bi-orthogonal extension $(\mathcal{U}^{\perp\perp}, \mathcal{V}^{\perp\perp})$ (with $\mathcal{V}^{\perp\perp} = \mathcal{U}^{\perp}$ and $\mathcal{U}^{\perp\perp} = \mathcal{V}^{\perp}$) need not be saturated. In other words, we conjecture that there exist $u \in \mathcal{U}^{\perp\perp}$ and $\{{}^{u}\!/_{x}\}p \in \mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$, such that $\mu x.p \notin \mathcal{V}^{\perp\perp}$ (or the symmetric situation, swapping $\mathcal{U}$ and $\mathcal{V}$). Indeed we could obtain this with $\{\mu x.p \mid u\} \notin \mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$, but the counter-examples of [DN05b] only provide this with $u \in \mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$ instead of $u \in \mathcal{U}^{\perp\perp} \subseteq \mathsf{SN}^{F^{\mathcal{C}}_{\omega}}$.

To conclude this section we mention that we have developed the two reducibility methods for the two strong normalisation results in order to compare them and state the above conjecture, and for other reasons mentioned in the introduction, but alternative proofs could have been given. For the upper layer we could simply have simulated the reduction in the simply-typed λ-calculus (or even in the simply-typed $\overline{\lambda}$), forgetting all the information about duality (A and $A^{\perp}$ would be mapped to the same term) which plays no computational role in this layer. For instance, $\alpha\ l$ and $\overline{\alpha}\ l$ would be mapped to the same term, $A \wedge B$ and $A \vee B$ would both be mapped to $x_{\wedge\vee}\ A\ B$ and $\forall\alpha^{K}.B$ and $\exists\alpha^{K}.A$ would both be mapped to $x_{\forall\exists}\ \lambda\alpha.A$ for two particular variables $x_{\wedge\vee}$ and $x_{\forall\exists}$ that are never bound because they represent the logical connectives.

However, such an encoding, while preserving the notion of computation, loses all information about duality. This has two consequences:

- It cannot be used to establish a reflection between the upper layer of $F_\omega^{\mathcal{C}}$ and the simply-typed λ-calculus (or the upper layer of F_ω).

- Since it loses all the logical meaning of type constructors, it cannot be used for a type-preserving encoding of the lower layer in e.g. a version of F_ω with a particular variable whose type implies classical logic, such as the elimination of double negation (see section 10.3.2 and Conjecture 293).

Ongoing work is about refining this forgetful mapping by encoding in λ-terms the information about duality, i.e. some notion of "polarity", in a way that is useful for the above two points.

For the lower layer we could try to adapt to F_ω simpler proofs of strong normalisation of the simply-typed $\overline{\lambda}\mu\widetilde{\mu}$ [CH00] (a variant in a bi-sided sequent calculus of our calculus and of the symmetric λ-calculus [BB96]), such as those of [DN05a] or [Dou06] which do not involve the fixpoint construction. We do not know whether these proofs break, for a typing system as strong as that of $F_\omega^{\mathcal{C}}$.

10.3 Logical Properties

10.3.1 Consistency

The consistency of $F_\omega^{\mathcal{C}}$ follows from Corollary 283 using a very simple combinatorial argument. Let us first notice that all untyped programs that are in normal form are of one of the following thirteen forms:

$$\begin{array}{r}
\{x \mid y\} \\
\{x \mid \lambda x^A y^B.p\} \\
\{x \mid \langle t, u\rangle\} \\
\{x \mid \Lambda\alpha^K.t\} \\
\{x \mid \langle A, t\rangle\} \\
\{\langle t_1, u_1\rangle \mid \langle t_2, u_2\rangle\} \\
\{\lambda x_1^{A_1} y_1^{B_1}.p_1 \mid \lambda x_2^{A_2} y_2^{B_2}.p_2\} \\
\{\Lambda {\alpha_1}^K.t_1 \mid \Lambda {\alpha_2}^K.t_2\} \\
\{\langle A_1, t_1\rangle \mid \langle A_2, t_2\rangle\} \\
\{\lambda x_1^{A_1} y_1^{B_1}.p_1 \mid \Lambda\alpha^K.t_2\} \\
\{\langle t_1, u_1\rangle \mid \Lambda\alpha^K.t_2\} \\
\{\lambda x_1^{A_1} y_1^{B_1}.p_1 \mid \langle A_2, t_2\rangle\} \\
\{\langle t_1, u_1\rangle \mid \langle A_2, t_2\rangle\}
\end{array}$$

However, if we consider only typed programs, then the last eight forms are ruled out for typing reasons, indeed:

Lemma 285 *There is no closed typed program in normal form.*

Proof: In each of the eight last forms, both members introduce a main connective or quantifier which is not the dual of the one introduced on the other side, which contradicts the typing rule of programs. All the remaining forms have a free variable, namely x. □

Hence we get the logical consistency of system $F_\omega^{\mathcal{C}}$.

Theorem 286 (Consistency) *There is no closed typed program in* $F_\omega^{\mathcal{C}}$.

Proof: It suffices to combine Lemma 285 with corollary 283 and Theorem 269. □

10.3.2 Encoding of system F_ω into $F_\omega^{\mathcal{C}}$

In this section we describe how the encoding of PTS into PTSC from Chapter 8 can be turned into an encoding of F_ω into $F_\omega^{\mathcal{C}}$, based on our definition, in $F_\omega^{\mathcal{C}}$, of implication $A \to B$ as $(A^\perp) \vee B$. The encoding itself is adapted from that of Prawitz of natural deduction into sequent calculus (described in Chapter 2).

Definition 167 (Translation of type constructors)
Each type constructor A of F_ω is translated as a type constructor A^* of $F_\omega^{\mathcal{C}}$ according to Fig. 10.12, which recalls the encoding of PTS into PTSC from Chapter 8 (indeed for the type constructors of $F_\omega^{\mathcal{C}}$ we have used the same syntax as for PTSC).

$\mathcal{A}(\forall\alpha^K.A)$	$:=$	$\forall\alpha^K.\mathcal{A}(A)$	
$\mathcal{A}(A \to B)$	$:=$	$\mathcal{A}(A)^\perp \vee \mathcal{A}(B)$	
$\mathcal{A}(\lambda\alpha^K.B)$	$:=$	$\lambda\alpha^K.\mathcal{A}(B)$	
$\mathcal{A}(A)$	$:=$	$\mathcal{A}_{[]}(A)$	otherwise
$\mathcal{A}_l(B\ A)$	$:=$	$\mathcal{A}_{\mathcal{A}(A)\cdot l}(B)$	
$\mathcal{A}_l(\alpha)$	$:=$	$\alpha\ l$	
$\mathcal{A}_l(A)$	$:=$	$\mathcal{A}(A)\ l$	otherwise

Figure 10.12: Encoding of type constructors

As in Chapter 8 and Chapter 9, system $\mathsf{B'x'}$ simulates β-reduction through the translation:

Theorem 287 (Simulation of β for type constructors) *If* $A \longrightarrow_\beta B$, *then* $\mathcal{A}(A) \longrightarrow_{\mathsf{B'x'}} \mathcal{A}(B)$.

Derivability of sequents of system F_ω, denoted using $\vdash^{F_\omega}$, can be defined as that of a PTS, but simpler rules with different syntactic categories and unordered environments can also be given [Gir72] as we did for $F_\omega^{\mathcal{C}}$. Now the kinds of F_ω

are identical to those of $F_\omega^{\mathcal{C}}$, so no encoding but the identity is needed there. As in Chapter 8, the translation preserves typing:

Theorem 288 (Preservation of typing for type constructors)

1. *If* $\Sigma \vdash_{F_\omega} A : K$, *then* $\Sigma \vdash_{F_\omega^{\mathcal{C}}} \mathcal{A}(A) : K$.
2. *If* $\Sigma \vdash_{F_\omega} A : K$ *and* $\Sigma; K \vdash_{F_\omega^{\mathcal{C}}} l : K'$, *then* $\Sigma \vdash_{F_\omega^{\mathcal{C}}} \mathcal{A}_l(A) : K$.

Proof: By simultaneous induction on derivations. □

We now translate terms, adapting again Prawitz's translation of natural deduction into sequent calculus from Chapter 2, this time using Curry-style terms and programs, because without a typing derivation for the terms of F_ω we lack some type annotations to place in the encoding.

Definition 168 (Encoding of terms) The encoding $\mathcal{A}(u)$ of a term u of F_ω is defined by induction on u as described in Fig. 10.13. It relies on an auxiliary encoding that maps u to a program $\mathcal{A}(u,t)$ and that is parameterised by a term t of $F_\omega^{\mathcal{C}}$.

$\mathcal{A}(x)$	$:=$	x	
$\mathcal{A}(\lambda x^A.u)$	$:=$	$\lambda xy.\mathcal{A}(u,y)$	
$\mathcal{A}(\Lambda\alpha^K.u)$	$:=$	$\Lambda\_.\mathcal{A}(u)$	
$\mathcal{A}(u)$	$:=$	$\mu y.\mathcal{A}(u,y)$	otherwise
$\mathcal{A}((u\ u'),t)$	$:=$	$\mathcal{A}(u,\langle\mathcal{A}(u'),t\rangle)$	
$\mathcal{A}((u\ A),t)$	$:=$	$\mathcal{A}(u,\langle\_,t\rangle)$	
$\mathcal{A}(v,t)$	$:=$	$\{\mathcal{A}(v) \mid t\}$	otherwise

Figure 10.13: Encoding of terms

Remark 289 For a Curry-style term t and a Curry-style program p of $F_\omega^{\mathcal{C}}$,

1. If $t \longrightarrow_{F_\omega^{\mathcal{C}}} t'$ then $\mathcal{A}(u,t) \longrightarrow_{F_\omega^{\mathcal{C}}} \mathcal{A}(u,t')$.
2. $\{\mathcal{A}(u) \mid t\} \longrightarrow^*_{F_\omega^{\mathcal{C}}} \mathcal{A}(u,t)$
3. $\left\{{}^{\mathcal{A}(u')}/_{x}\right\}\mathcal{A}(u,t) \longrightarrow^*_{F_\omega^{\mathcal{C}}} \mathcal{A}(\left\{{}^{u'}/_{x}\right\}u, \left\{{}^{\mathcal{A}(u')}/_{x}\right\}t)$ and $\left\{{}^{\mathcal{A}(u')}/_{x}\right\}\mathcal{A}(u) \longrightarrow^*_{F_\omega^{\mathcal{C}}} \mathcal{A}(\left\{{}^{u'}/_{x}\right\}u)$.

Again, the encoding of terms allows the simulation of reductions as in Theorem 206:

Theorem 290 (Simulation of β for terms)
If $u \longrightarrow_{F_\omega} u'$, *then* $\mathcal{A}(u,t) \longrightarrow^+_{F_\omega^{\mathcal{C}}} \mathcal{A}(u',t)$ *and* $\mathcal{A}(u) \longrightarrow^+_{F_\omega^{\mathcal{C}}} \mathcal{A}(u')$.

Proof: By simultaneous induction on the derivation of the reduction step, using Remark 289. □

Again, the translation preserves typing:

Theorem 291 (Preservation of typing for terms)

1. *If* $\Gamma \vdash^{\Sigma}_{F_\omega} u : A$, *then there exists a term* t *of system* $F^{\mathcal{C}}_\omega$ *(with type annotations) such that* $\|t\| = \mathcal{A}(u)$ *and* $\mathcal{A}(\Gamma) \vdash^{\Sigma}_{F^{\mathcal{C}}_\omega} t : A$.

2. *If* $\Gamma \vdash^{\Sigma}_{F_\omega} u : A$ *and* $\mathcal{A}(\Gamma), \Delta \vdash^{\Sigma}_{F^{\mathcal{C}}_\omega} t : \mathcal{A}(A)^{\perp}$, *then there exists a program* p *of system* $F^{\mathcal{C}}_\omega$ *(with type annotations) such that* $\|p\| = \mathcal{A}(u, \|t\|)$ *and* $\mathcal{A}(\Gamma), \Delta \vdash^{\Sigma}_{F^{\mathcal{C}}_\omega} p : \diamond$.

Proof: Again, as in Theorem 66 and Theorem 218, this is obtained by the same induction on derivations, using Theorem 287 for the conversion rule. □

Since $F^{\mathcal{C}}_\omega$ is classical, we have a proof of the axiom of double negation elimination (again we abbreviate $\alpha\ []$ as α and $\overline{\alpha}\ []$ as $\overline{\alpha}$):

Let $\bot := \forall\alpha^\star.\alpha$ (in F_ω and $F^{\mathcal{C}}_\omega$) and $\top := \exists\alpha^\star.\alpha$ (in $F^{\mathcal{C}}_\omega$), and let $\mathrm{DNE} := \forall\alpha^\star.((\alpha \Rightarrow \bot) \Rightarrow \bot) \Rightarrow \alpha$ in system F_ω.

We have $\mathcal{A}(\mathrm{DNE}) = \forall\alpha^\star.((\overline{\alpha} \vee \bot) \wedge \top) \vee \alpha$.

Let $\mathsf{C} := \Lambda\alpha^\star.\lambda x^B y^{\overline{\alpha}}.\{x \mid \langle \lambda x'^{\alpha} y'^{\top}.\{x' \mid y\}, \langle \overline{\alpha}, y\rangle\rangle\}$, where $B := (\alpha \wedge \top) \vee \bot$.

We have

$$\vdash_{F^{\mathcal{C}}_\omega} \mathsf{C} : \mathcal{A}(\mathrm{DNE})$$

Hence, provable propositions of system $F_\omega + \mathrm{DNE}$ become provable propositions of system $F^{\mathcal{C}}_\omega$:

Theorem 292 (F_ω captures $F_\omega + \mathrm{DNE}$) *For all derivable judgements of the form*

$$z : \mathrm{DNE},\ \Gamma\ \vdash^{\Sigma}_{F_\omega}\ u\ :\ A$$

there exists a term t *of system* $F^{\mathcal{C}}_\omega$ *(with type annotations) such that* $\|t\| = \mathcal{A}(u)$ *and we have*

$$\mathcal{A}(\Gamma) \vdash^{\Sigma}_{F^{\mathcal{C}}_\omega} \{\mathsf{c}/_z\}\, t : \mathcal{A}(A)$$

Through the translation $A \mapsto \mathcal{A}(A)$, system $F^{\mathcal{C}}_\omega$ appears as an extension of system $F_\omega + \mathrm{DNE}$, and hence the consistency of $F^{\mathcal{C}}_\omega$, proved in section 10.3.1, implies that of $F_\omega + \mathrm{DNE}$.

We then set the following conjecture:

Conjecture 293 ($F_\omega^{\mathcal{C}}$ is a conservative extension of $F_\omega + \mathrm{DNE}$)
There exists a mapping $\mathcal{B}$ of the upper layer of $F_\omega^{\mathcal{C}}$ into that of F_ω such that:

1. *If $\Sigma \vdash_{F_\omega} A:\star$, then there exist two terms u and u' such that $\vdash^{\Sigma}_{F_\omega} u:A \to \mathcal{B}(\mathcal{A}(A))$ and $\vdash^{\Sigma}_{F_\omega} u':\mathcal{B}(\mathcal{A}(A)) \to A$.*
2. *If $\Gamma \vdash^{\Sigma}_{F_\omega^{\mathcal{C}}} t:A$ then there exists a term u of F_ω such that $\mathcal{B}(\Gamma), z:\mathrm{DNE} \vdash^{\Sigma}_{F_\omega} u:\mathcal{B}(A)$.*

As mentioned in section 10.2.3, the mapping that forgets the information about duality is obviously not a good candidate to prove this conjecture, but ongoing work is about refining it for that purpose.

Conclusion

In this chapter we have designed a classical version of system F_ω, called $F_\omega^{\mathcal{C}}$, entirely in sequent calculus in the spirit of PTSC of Chapter 8.

The first technical purpose was to express two methods to prove strong normalisation that are very similar, and the two layers of $F_\omega^{\mathcal{C}}$ provided an excellent opportunity for such a comparison. The two methods are both based on the technique of reducibility of Tait and Girard [Gir72], and, while the first technique, involving *orthogonality*, builds reducibility candidates by a bi-orthogonal completion, the second technique (Barbanera and Berardi's symmetric candidates) uses a completion by a fixpoint construction. We raised the conjecture (Conjecture 284) that orthogonality does not capture the fixpoint construction.

In section 10.2.3 and section 10.3.2 we have mentioned some ongoing work:

- proving Conjecture 284 with a counter-example, and
- defining an encoding, say $\mathcal{B}$, of the upper layer of $F_\omega^{\mathcal{C}}$ into the upper layer of F_ω, such that $\mathcal{B}$ and $\mathcal{A}$ form a reflection (from which we could directly derive the confluence of the layer), and we could prove the conservativity theorem (Conjecture 293).

As mentioned before, F_ω is the strongest corner of Barendregt's Cube where the layer of proofs can be turned classical without much trouble (in particular, Barbanera and Berardi's method for strong normalisation does not break up to that point, as we have shown in section 10.2.2). Adding the last dimension of Barendregt's Cube, namely dependent types, seems much more complicated because of the unclear semantics that classical logic would bring within the types (and the semantics matters, if only because the reducibility method is based on the construction of a model). Indeed, which notion of conversion would we

consider on types (e.g. for typing terms) if these depend on terms? The reflexive, transitive and symmetric closure of reduction leads to proof-irrelevance, as Lafont's example shows (Example 13), but maybe this could be acceptable. Restricting reduction to CBV or CBN reduction to get a confluent system [DGLL05] would avoid the problem but is frustrating in that such a restriction allows the definition of a CPS-translation into intuitionistic logic, missing out on the essence of classical logic. Considering syntactic equality of terms is drastic, since the logic would then distinguish proofs that only differ in the bureaucratic details due to the structure of the formalism.

In the next chapter, we investigate an alternative option, namely a notion of equivalence on classical proofs, which, with dependent types, could be used to define an interesting notion of convertibility of types, and which, instead of being based on cut-elimination, is simply based on permutations of inference rules. These permutations correspond to the inessential details by which some proofs in sequent calculus differ,[4] and are surprisingly captured by the well-known notion of *parallel reduction* in rewriting.

[4]These already appear in intuitionistic logic as those permutations that identify the proofs of sequent calculus corresponding to the same proof in natural deduction, see [DP99b].

Chapter 11

An equivalence on classical proofs

In this chapter, most of which appeared in [BL05], we investigate a particular approach to the notion of equivalence of classical proofs.

In intuitionistic logic, a notion of equivalence of proofs is directly obtained from their functional interpretation, say in a set-theoretic model or more generally in a cartesian-closed category. For the proofs in natural deduction, represented by λ-terms (see Chapter 2), this corresponds to $\beta\eta$-equivalence, in other words, the reflexive, transitive and symmetric closure of the notion of normalisation (or cut-elimination). Such a notion of equivalence has revealed very fruitful, for instance in Type Theory (e.g. those mentioned in Part II and Chapter 10 of this dissertation).

We would like to have a similar notion for classical logic, namely a proof-theoretic formalism equipped with

1. an equivalence on proofs,
2. canonical representatives of equivalence classes,
3. a notion of computation such that equivalence of two proofs can be decided by computing the canonical representatives of their equivalence classes.

Point 1 is desirable in that, if (classical) proofs are to be considered as mathematical objects, we expect to know when two syntactic representations denote the same object (e.g. $5 + 4$ and 9 for natural numbers). Points 2 and 3 are desirable on their own, because we like to decide point 1 and have a notion of canonical representation (e.g. 9 rather than $5+4$), but even more so if proofs are themselves the objects described in a formal language and logic, i.e. when we have predicates on proofs as in Type Theory.

These features are provided by β- or $\beta\eta$-reduction in intuitionistic natural deduction, but considering the corresponding notion of normalisation in classical logic lacks the semantical justification that intuitionistic logic enjoys. In fact, the non-confluence of normalisation leads in this case to proof-irrelevance, as Lafont's counter-example illustrates (Example 13).

This can be avoided by restricting normalisation to confluent sub-systems such as CBN and CBV, but this fails to capture the essence of computation in classical logic, since these restrictions can be encoded back into intuitionistic logic by the use of CPS (a.k.a not-not) translations, with corresponding semantics given in *control* and *co-control categories* [Sel01].

The full notion of normalisation can be modelled in *classical categories* [FP06, McK05] which feature an order on morphisms. This does not lead to proof irrelevance since some normalisation steps of a proof do not preserve its interpretation as a morphism but decrease it w.r.t. the order. Nevertheless, how we could accommodate this order where we wanted a notion of equivalence (e.g. to define the notion of convertibility of types in dependent type theories) is not clear.

In this chapter we consider a notion of equivalence on classical proofs that we can identify in a formalism different from, but still related to, sequent calculus and natural deduction, called the *Calculus of Structures* (CoS) [Gug, Gug02].

One way of presenting it is simply as a rewrite system on formulae, such that if $A \longrightarrow B$ then A implies B in classical logic. Hence a proof of a formula A is simply a reduction sequence from the constant true to the formula A, and we have to show that the rewrite system makes such a notion of provability sound and complete for classical logic.

We use CoS to provide a notion of equivalence on classical proofs because we can clearly identify its *bureaucracy*, i.e. the details by which two proofs featured by a proof-theoretic formalism differ while being "morally" the same.

In fact in intuitionistic logic, β- or $\beta\eta$-equivalence can be considered as identifying the bureaucracy featured by intuitionistic natural deduction. Normalisation in classical logic does not play the same role, as illustrated for instance by its semantics in classical categories. On the other hand, consider the following two derivations in a mono-sided sequent calculus (similar to that of Chapter 10):

$$\dfrac{\dfrac{\vdash A, B, A^{\perp}, C}{\vdash A, B, A^{\perp} \vee C}}{\vdash A \vee B, A^{\perp} \vee C} \qquad \text{and} \qquad \dfrac{\dfrac{\vdash A, B, A^{\perp}, C}{\vdash A \vee B, A^{\perp}, C}}{\vdash A \vee B, A^{\perp} \vee C}$$

Clearly, these two proofs are essentially the same, and we prefer not to distinguish them. More to the point, the sequent calculus forces us to choose an order to apply the two rules that is not relevant.

Proof nets, introduced by Girard [Gir87] for linear logic, are a less bureaucratic formalism than the sequent calculus. They have also been developed for classical logic, e.g. in [Rob03] and [LS05] which mainly differ in the way contraction is represented. Proof nets have the merit that they do not distinguish between proofs such as the above, but it is not clear how such graphical formalisms can be used either in a language and a logic whose objects are proofs or in practical systems like those based on the notion of derivation (e.g. Type Theory). Also, the notion of *inference step* is lost when moving from the sequent calculus to

proof nets, since the correctness of the latter generally requires checking a global criterion (which is more algorithmic than deductive).

The difference between the two proofs above can be captured as a particular case of bureaucracy identified in CoS. The latter is of two kinds, Bureaucracy A [Gug04a] and Bureaucracy B [Gug04b], and in fact occurs in any rewrite system (in fact, any left- and right-linear one for Bureaucracy B).

Bureaucracy A corresponds to the choice of an order for two consecutive rewrite steps that reduce disjoint/non-overlapping/parallel sub-terms. Bureaucracy B corresponds to the choice of an order for two consecutive rewrite steps whose redexes a *nested*, i.e. when one redex is inside the other without being destroyed by the reduction of the latter (i.e. there is a residual). In other words, Bureaucracy A and Bureaucracy B correspond to the choice of ordering two redexes when these do not form a critical pair, and can be captured by the notion of *parallel reduction* (see e.g. [Tak89]).

Two formalisms are suggested in [Gug04a, Gug04b] to address these kinds of bureaucracy, namely Formalisms A and B, respectively, and the starting point of this chapter is first to formalise them with proof-terms, and second to provide a normalisation procedure which, given a bureaucratic derivation, yields its bureaucracy-free representative. We shall see that these formalisms can be considered as sequent calculi with axioms (say, $\overline{A \vdash A'}$ with $A \neq A'$ but A implies A' in classical logic), so cut can no longer be eliminated, but the normalisation procedure that produces canonical representatives is in fact cut-reduction.

Section 11.1 presents a linear term rewrite system for classical propositional logic. Section 11.2 defines proof terms for derivations in Formalism A and give a rewrite system for these proof terms that removes bureaucracy, which, as we shall see, turns out to be a process of cut-elimination. Section 11.3 goes further by presenting Formalism B, but is not as complete. However the main tool to eliminate the bureaucracy is a notion of *tube*, which is a placeholder which winds through a derivation and contains another derivation.

11.1 Classical logic as term rewriting

A system in the *Calculus of Structures* (CoS) [Gug02] is a rewrite system on formulae modulo a simple equational theory, such as associativity and commutativity of disjunction and conjunction, cf. [Kah04].

A proof of a formula A is simply a reduction sequence from the constant true ($\top$) to the formula A. Recall from Chapter 1 that a reduction sequence is just a particular case of derivation where inference steps have only one premiss (and this is indeed the terminology originally used to define CoS [Gug, Gug02]).

We slightly depart from the rewriting modulo by dropping the equational theory (which is difficult to work with) and replace some of these equations by rules, in a way that is still logically sound and complete. This would be harmful

for properties such as termination, but in fact termination and confluence, which are typical properties of interest in rewriting, are not properties that we require from systems in CoS. Typical properties we require from systems in CoS are rather about the admissibility of rules and about the existence of certain normal forms for derivations. Nevertheless, rewrite systems is what they are.

The rewrite rules should satisfies the property that if $A \longrightarrow B$ then A implies B in the logic considered. In order for this property to hold for the contextual closure of the rewrite rules, formulae must not have sub-formulae in contrapositive positions, such as A in $\neg A$ or $A \to B$. Hence, as in Chapter 10, we only use conjunction and disjunction, and primitive negation on variables (and primitive constants true and false).

Definition 169 (Formula) The grammar of formulae is similar to that of Chapter 10 as follows:

$$A, B, C, \ldots ::= \top \mid \bot \mid \alpha \mid \overline{\alpha} \mid A \vee B \mid A \wedge B$$

where $\top$ and $\bot$ are the constants *true* and *false* taken as primitive, α ranges over a set of variables that form a syntactic category of their own, with two dual injections in the syntax of formulae: α and $\overline{\alpha}$.

Atoms are those formulae of the form α or $\overline{\alpha}$ and are ranged over by $a, b, c, \ldots$

Definition 170 (Duality) The notion of *dual* of a formula is defined as in Chapter 10 as follows:

$\bot^\bot$	$:=$	$\top$	$\top^\bot$	$:=$	$\bot$
$\alpha^\bot$	$:=$	$\overline{\alpha}$	$\overline{\alpha}^\bot$	$:=$	α
$(A \vee B)^\bot$	$:=$	$A^\bot \wedge B^\bot$	$(A \wedge B)^\bot$	$:=$	$A^\bot \vee B^\bot$

In the rest of this section we present rewrite systems (one non-linear, one linear) on formulae that can be used for classical logic. These systems are essentially obtained from system SKS from [BT01, Brü03] by removing all equations and adding some of them as rewrite rules.

The systems are rather idiosyncratic and are not really central to the ideas developed hereafter. We present them just to show that linear rewriting indeed can be a proof-theoretic formalism for classical logic and also to have some rules as running examples. Formalisms A and B as we present them easily generalise to any linear rewrite system.

Definition 171 (Rewrite rules on formulae) A system of rewrite rules for classical propositional logic is given in Fig. 11.1. The sub-system in the upper part is called KSf where K means classical, S means calculus of structures and f is for (equation-)free. The entire system is called SKSf, where the first S is for

symmetric. The name of the rules in the upper part are short for duplication, unit, commutativity, identity, switch, weakening and contraction. Their dual rules (actually, their contrapositions) in the lower part have the same name but with the prefix "co-".

$\mathsf{du}\downarrow$	$\top$	$\longrightarrow$	$\top \wedge \top$
$\mathsf{un}\downarrow$	A	$\longrightarrow$	$A \vee \bot$
$\mathsf{co}\downarrow$	$A \vee B$	$\longrightarrow$	$B \vee A$
$\mathsf{i}\downarrow$	$\top$	$\longrightarrow$	$A^{\bot} \vee A$
$\mathsf{s}\downarrow$	$(A \vee B) \wedge (C \vee D)$	$\longrightarrow$	$(A \vee C) \vee (B \wedge D)$
$\mathsf{w}\downarrow$	$\bot$	$\longrightarrow$	A
$\mathsf{c}\downarrow$	$A \vee A$	$\longrightarrow$	A
$\mathsf{du}\uparrow$	$\bot \vee \bot$	$\longrightarrow$	$\bot$
$\mathsf{un}\uparrow$	$A \wedge \top$	$\longrightarrow$	A
$\mathsf{co}\uparrow$	$A \wedge B$	$\longrightarrow$	$B \wedge A$
$\mathsf{i}\uparrow$	$A \wedge A^{\bot}$	$\longrightarrow$	$\bot$
$\mathsf{s}\uparrow$	$(A \wedge C) \wedge (B \vee D)$	$\longrightarrow$	$(A \wedge B) \vee (C \wedge D)$
$\mathsf{w}\uparrow$	A	$\longrightarrow$	$\top$
$\mathsf{c}\uparrow$	A	$\longrightarrow$	$A \wedge A$

Figure 11.1: System SKSf

We have soundness and completeness for classical propositional logic:

Theorem 294 (Soundness & completeness)

1. $\top \longrightarrow^*_{\mathsf{KSf}} A$ *if and only if* A *is valid in classical logic.*
2. $A \longrightarrow^*_{\mathsf{SKSf}} B$ *if and only if* A *classically implies* B*.*

Proof: Soundness in both cases follows from a simple induction on the length of the derivation and the observation that implication is closed under conjunction and disjunction. System KSf is complete: a formula can be derived from its conjunctive normal form via the rules $\mathsf{c}\downarrow, \mathsf{co}\downarrow, \mathsf{s}\downarrow$. If the formula is valid, then each of the nested disjunctions in the conjunctive normal form contains two dual atoms. By $\mathsf{w}\downarrow, \mathsf{un}\downarrow$ this formula can be derived from a formula where all atoms except for the two dual atoms are removed. By $\mathsf{i}\downarrow$ we derive this from a conjunction of lots of occurrences of $\top$, which is derived from $\top$ by $\mathsf{du}\downarrow$. The completeness direction of point 2 is then a matter of constructing a derivation from A to B in SKSf for each derivation from $\top$ to $A^{\bot} \vee B$ in KSf, see [Brü03] for details. □

Definition 172 (Linear rewrite rules on formulae) From SKSf we obtain a linear rewriting system SKSfl, where l is for linear, which is shown in Fig. 11.2.

$\mathsf{du}{\downarrow}$	$\top$	$\longrightarrow$	$\top \wedge \top$
$\mathsf{un}{\downarrow}$	A	$\longrightarrow$	$A \vee \bot$
$\mathsf{co}{\downarrow}$	$A \vee B$	$\longrightarrow$	$B \vee A$
$\mathsf{ai}{\downarrow}$	$\top$	$\longrightarrow$	$a^{\bot} \vee a$
$\mathsf{s}{\downarrow}$	$(A \vee B) \wedge (C \vee D)$	$\longrightarrow$	$(A \vee C) \vee (B \wedge D)$
m	$(A \wedge B) \vee (C \wedge D)$	$\longrightarrow$	$(A \vee C) \wedge (B \vee D)$
$\mathsf{m_0}{\downarrow}$	$(A \vee B) \vee (C \vee D)$	$\longrightarrow$	$(A \vee C) \vee (B \vee D)$
$\mathsf{w_0}{\downarrow}$	$\bot$	$\longrightarrow$	$\bot \wedge \bot$
$\mathsf{aw}{\downarrow}$	$\bot$	$\longrightarrow$	a
$\mathsf{ac}{\downarrow}$	$a \vee a$	$\longrightarrow$	a
$\mathsf{du}{\uparrow}$	$\bot \vee \bot$	$\longrightarrow$	$\bot$
$\mathsf{un}{\uparrow}$	$A \wedge \top$	$\longrightarrow$	A
$\mathsf{co}{\uparrow}$	$A \wedge B$	$\longrightarrow$	$B \wedge A$
$\mathsf{ai}{\uparrow}$	$a \wedge a^{\bot}$	$\longrightarrow$	$\bot$
$\mathsf{s}{\uparrow}$	$(A \wedge C) \wedge (B \vee D)$	$\longrightarrow$	$(A \wedge B) \vee (C \wedge D)$
m	$(A \wedge B) \vee (C \wedge D)$	$\longrightarrow$	$(A \vee C) \wedge (B \vee D)$
$\mathsf{m_0}{\uparrow}$	$(A \wedge B) \wedge (C \wedge D)$	$\longrightarrow$	$(A \wedge C) \wedge (B \wedge D)$
$\mathsf{w_0}{\uparrow}$	$\top \wedge \top$	$\longrightarrow$	$\top$
$\mathsf{aw}{\uparrow}$	a	$\longrightarrow$	$\top$
$\mathsf{ac}{\uparrow}$	a	$\longrightarrow$	$a \wedge a$

Figure 11.2: System SKSfl

Derivability in the linear system is the same as in the non-linear system.

Theorem 295 (Soundness & completeness)

1. *$A \longrightarrow^*_{\mathsf{SKSf}} B$ if and only if $A \longrightarrow^*_{\mathsf{SKSfl}} B$*
2. *$A \longrightarrow^*_{\mathsf{KSf}} B$ if and only if $A \longrightarrow^*_{\mathsf{KSfl}} B$*

Proof: See [Brü03]. □

11.2 Formalism A

Consider the following two reduction sequences:

$$\begin{array}{llllll} (a \vee a) \wedge (b \vee b) & \longrightarrow_{\mathsf{ac}{\downarrow}} & a \wedge (b \vee b) & \longrightarrow_{\mathsf{ac}{\downarrow}} & a \wedge b \\ (a \vee a) \wedge (b \vee b) & \longrightarrow_{\mathsf{ac}{\downarrow}} & (a \vee a) \wedge b & \longrightarrow_{\mathsf{ac}{\downarrow}} & a \wedge b \end{array}$$

As in the example presented in the introduction (vertically), these two sequences inessentially differ in the order in which the two rules are applied. No matter

which of the sequences we choose, it contains irrelevant information. We now define Formalism A, which provides a third derivation which stores no information about the order between the two applications of $\mathsf{ac}{\downarrow}$. The solution is by introducing a parallel composition of reduction.

11.2.1 Syntax & typing

Definition 173 (Proof-term) Proof-terms (or just terms) of Formalism A are defined as follows:

$$R, S, T, U, \ldots ::= \mathsf{id} \mid \rho \mid (R \mid S) \mid R.S$$

where id is *identity*, ρ ranges over a set of constants (one for each rewrite rule of Fig. 11.2) called *combinators* (we use the very name of the rule as these constants), $(R_1 \mid R_2)$ is *parallel composition* and $(R_1 \,.\, R_2)$ is *sequential composition*.

Definition 174 (Typing rules) Derivability of a judgement $A \vdash R : B$ in the typing system of Fig. 11.3 (which considers the rewrite rules —a.k.a. axioms— of Fig. 11.2) is denoted $A \vdash_{\mathsf{FA}} R : B$, and it means that the proof-term R is a proof of B assuming A.[1]

$$\frac{}{A \vdash \mathsf{id} : A} \qquad \frac{A \xrightarrow{\mathtt{root}}_{\rho} B}{A \vdash \rho : B}$$

$$\frac{A \vdash R : C \quad B \vdash S : D}{A \wedge B \vdash R \mid S : C \wedge D} \qquad \frac{A \vdash R : C \quad B \vdash S : D}{A \vee B \vdash R \mid S : C \vee D}$$

$$\frac{A \vdash R : B \quad B \vdash S : C}{A \vdash R.S : C}$$

Figure 11.3: Typing rules for Formalism A

Note that this typing system forms a sequent calculus, where the typing rule for combinators is an axiom given by the corresponding rewrite rule, the typing rule for sequential composition is a particular form of cut (quite similar to Modus Ponens as well), and the three other rules are there both to capture the contextual closure of the rewriting and also to parallelise reductions of two disjoint/non-overlapping sub-formulae.

[1] Note that our notations $A \vdash R : B$ and $A \vdash_{\mathsf{FA}} R : B$ differ from [BL05] which denoted both of them $A \xrightarrow{R} B$.

This is clearly very much like a Hilbert-style system considered as a typing system.

Not every term is typable, for example $\mathsf{s}\downarrow \,.\, \mathsf{s}\downarrow$ is not. In general, terms are typable in different ways. For instance we have $a \vdash_{\mathsf{FA}} \mathsf{id} : a$, just like $b \vdash_{\mathsf{FA}} \mathsf{id} : b$.

We have soundness and completeness for classical propositional logic:

Theorem 296 (Soundness & completeness) *There is a proof term R of Formalism $\mathsf{A}$with $A \vdash_{\mathsf{FA}} R : B$ if and only if $A \longrightarrow^*_{\mathsf{SKSfl}} B$.*

Proof: The direction from left to right is an easy induction on the typing derivation. The converse is easy to see since a rewrite rule can be applied at an arbitrary depth with a proof term build from the label of the rewrite rule, identity and parallel composition, and consecutive rule applications are represented using sequential composition. □

Remark 297 Associativity of sequential composition $(R_1.R_2).R_3 \sim R_1.(R_2.R_3)$ preserves typing.

Notice 4 *From now on we consider terms up to associativity of sequential composition, and we write $R_1.R_2.R_3$ for (the equivalence class of) $(R_1.R_2).R_3$ and $R_1.(R_2.R_3)$.*

11.2.2 Reduction

We now use the potential of parallel composition to reduce Bureaucracy A, and our normalisation system turns out to be exactly cut-reduction in this sequent calculus with axioms. All cuts are not eliminated because some of them are blocked by non-identity axioms (while a cut with an identity axiom is eliminated as in a traditional cut-elimination procedure).

Definition 175 (Rewrite rules on terms) Normalisation is given as the reduction relation generated by the following system, called FA, modulo associativity of sequential composition.

$$\begin{array}{lcl} \mathsf{id} \,.\, R & \longrightarrow & R \\ R \,.\, \mathsf{id} & \longrightarrow & R \\ \mathsf{id} \mid \mathsf{id} & \longrightarrow & \mathsf{id} \\ (R \mid S) \,.\, (T \mid U) & \longrightarrow & (R \,.\, T) \mid (S \,.\, U) \end{array}$$

Theorem 298 (Termination & confluence) *The reduction relation $\longrightarrow_{\mathsf{FA}}$ is terminating and confluent.*

Proof: Each rule decreases the sum of the number of occurrences of id and the number of occurrences of parallel composition. Local confluence is easily checked. □

We call normal forms of FA *canonical*. The reduction rules preserve types, and the following theorem shows how the reduction is in fact cut-reduction.[2]

Theorem 299 (Subject reduction) *If* $A \vdash_{\mathsf{FA}} R : B$ *and* $R \longrightarrow_{\mathsf{FA}} S$ *then* $A \vdash_{\mathsf{FA}} S : B$.

Proof:

- The derivation

$$\dfrac{\dfrac{}{A \vdash \mathsf{id} : A} \qquad A \vdash R : B}{A \vdash \mathsf{id} \,.\, R : B}$$

  is transformed into

$$A \vdash R : B$$

  For the second rule we have the symmetrical case.

- The derivation

$$\dfrac{\dfrac{}{A \vdash \mathsf{id} : A} \qquad \dfrac{}{B \vdash \mathsf{id} : B}}{A \wedge B \vdash \mathsf{id} \mid \mathsf{id} : A \wedge B}$$

  is transformed into

$$\dfrac{}{A \wedge B \vdash \mathsf{id} : A \wedge B}$$

  And similarly for disjunction.

- The derivation

$$\dfrac{\dfrac{A \vdash R : E \quad B \vdash S : F}{A \wedge B \vdash R \mid S : E \wedge F} \qquad \dfrac{E \vdash T : C \quad F \vdash U : D}{E \wedge F \vdash T \mid U : C \wedge D}}{A \wedge B \vdash (R \mid S) \,.\, (T \mid U) : C \wedge D}$$

  is transformed into

$$\dfrac{\dfrac{A \vdash R : E \quad E \vdash T : C}{A \vdash R \,.\, T : C} \qquad \dfrac{B \vdash S : F \quad F \vdash U : D}{B \vdash S \,.\, U : D}}{A \wedge B \vdash (R \,.\, T) \mid (S \,.\, U) : C \wedge D}$$

  And similarly for disjunction.

□

[2]Note however that the cut rule for a typing derivation has nothing to with a notion of cut sometime referred to in the literature of CoS, which may or may not be part of the rewrite rules. In our case, all the rules with an up-arrow are in some sense cuts. Their admissibility follows from the previous section and is unrelated to the following theorem.

Example 14 (Reducing Bureaucracy A) The two reduction sequences from the beginning of the section are represented by the following terms: $(\mathsf{ac}{\downarrow} \mid \mathsf{id})\,.\,(\mathsf{id} \mid \mathsf{ac}{\downarrow})$ and $(\mathsf{id} \mid \mathsf{ac}{\downarrow})\,.\,(\mathsf{ac}{\downarrow} \mid \mathsf{id})$, which represents the situation of two reduction steps with parallel redexes. Both terms normalise to $(\mathsf{ac}{\downarrow} \mid \mathsf{ac}{\downarrow})$.

However, there still is bureaucracy remaining in the canonical derivations of Formalism A. Consider the following two reduction sequences, where two reduction steps have *nested*, rather than parallel, redexes:

$$\begin{array}{llll} (b \vee b) \wedge a & \longrightarrow_{\mathsf{co}{\downarrow}} & a \wedge (b \vee b) & \longrightarrow_{\mathsf{ac}{\downarrow}} \; a \wedge b \\ (b \vee b) \wedge a & \longrightarrow_{\mathsf{ac}{\downarrow}} & b \wedge a & \longrightarrow_{\mathsf{co}{\downarrow}} \; a \wedge b \end{array}$$

Formalism A assigns them the following proof-terms: $\mathsf{co}{\downarrow}.(\mathsf{id}|\mathsf{ac}{\downarrow})$ and $(\mathsf{ac}{\downarrow}|\mathsf{id}).\mathsf{co}{\downarrow}$. There is no proof-term in Formalism A that composes the two rules in such a way that no order between them is fixed. The next section will provide such a bureaucracy-free proof-term.

11.3 Formalism B

In general, a rewrite step can be permuted with any other rewrite step that does not interact with its redex, that is, keeps it as a residual (which is unique, if we assume the system to be left- and right-linear). Formalism A captured the case when the two redexes are independent in that they are parallel. Formalism B tackles the case when the two redexes are nested, without forming a critical pair.

For instance, if a rewrite rule uses a meta-variable A in its left-hand side and right-hand side, then any reduction of an instance of A can occur after of before the application of the rewrite rule. Geometrically, there is a *tube* corresponding to this meta-variable, in which any reduction can happen independently from the rewrite rule. The requirement that the rewrite system is left- and right-linear corresponds to the fact that there is no branching of tubes. In a reduction sequence using the rewrite rule, the tube can extend over several steps whose application are not conditional on the instance of A. It extends to the left until a reduction step creates the actual instance and it extends to the right until a reduction step needs this instance to be reduced in order to apply. Viewed from inside the tube, a rewrite rule reducing the instance can be performed at any point along the tube, with two canonical points: it can apply as soon as its redex is created or delayed until the reduced instance is needed (to create the redex of another rule).

Correspondingly, if we represent the reduction sequence as a proof-term, an occurrence of a combinator can be permuted a certain distance to the left and a certain distance to the right (possibly both zero) until it hits another occurrence of a combinator such that the two collide (do not permute). The position of the combinator within these two collision points is irrelevant and the space between them, within which the combinator can permute freely, is precisely the tube.

Tubes are identified by a pair of variables, one marking its start, one marking its end, and they can be filled with proof-terms.

11.3.1 Syntax & typing

Definition 176 (Types & proof-terms) Starting from Formalism A, we extend the definition of formulae, which we now call types, and that of terms as follows:

$$A, B, C, \ldots ::= \bot \mid \top \mid \alpha \mid \overline{\alpha} \mid A \vee B \mid A \wedge B \mid {}^{A}xy^{B}$$

and

$$R, S, T, \ldots ::= \mathsf{id} \mid \rho \mid (R \mid R) \mid R \,.\, R \mid x \mid y$$

where x ranges over a set of variables for tube starts and y ranges over a set of variables for tube ends.

The type ${}^{A}xy^{B}$ represent "any type along the tube xy deriving B from A".[3]

Now the contents of tubes need to be recorded in an environment: An environment indicates, for each variable of a tube start, what is the variable for the end of the tube and what is the content of the tube (and vice versa for tube ends). We first define the notion of pre-environment.

Definition 177 (Pre-environment) A *pre-environment* Γ is a finite function that maps variables x (resp. y) to pairs of the form (y, R) (resp. (x, R)). The elements of its graph are called *declarations* and are denoted $x^{y} \propto R$ and ${}^{x}y \propto R$, respectively.

Now the fact that the contents of tubes can use other tubes defines a dependency graph on tubes that with shall require to be non-circular.

Definition 178 (Dependency graph of tubes) The dependency graph of a pre-environment is the binary relation between variables given as follows: if $x^{y} \propto R$ or ${}^{x}y \propto R$ then y and x *depend on* every variable of R.

Definition 179 (Environment) An *environment* is a pre-environment that is injective in its first component, that is *consistent*, in that for all $x^{y} \propto R$ and ${}^{x'}y' \propto R'$ with $x = x'$ or $y = y'$ we have $x = x'$, $y = y'$ and $R = R'$, and whose dependency graph is non-circular.

The union $\Gamma \cup \Delta$ of two environments Γ and Δ is denoted Γ, Δ if it is an environment.

[3] Note that in contrast to [BL05] we use a variable for a tube start and a variable for a tube end, rather than one variable for a tube with two polarised occurrences $\triangleright x$ and $\triangleleft x$ for its start and its end. This is more convenient to define the normalisation procedure of the next section. Correspondingly, we write ${}^{A}xy^{B}$ instead of x^{A}_{B} from [BL05].

The intuition of an environment is that $x^y \propto R$ (resp. ${}^x y \propto R$) declares that x (resp. y) marks the beginning (resp. the end) of a tube that will end with y (resp. has started with x) and that contains the proof-term R. Since we declare the beginning and the end of a tube separately, we must then check that these two declarations are not contradictory, i.e. that the content of the tube is the same in the two declarations.

Definition 180 (Declarations concerning the same tube) The relation $\sharp$ identifies pairs of declarations that *should* concern the same tube:

$$\begin{array}{rl} \sharp := & \{({}^x y \propto R, x'^{y'} \propto R') \mid x = x' \vee y = y'\} \\ \cup & \{(x^y \propto R, x'^{y'} \propto R') \mid x = x' \vee y = y'\} \\ \cup & \{({}^x y \propto R, {}^{x'} y' \propto R') \mid x = x' \vee y = y'\} \end{array}$$

Definition 181 (Separability & connectability)

- Γ and Δ are *separated* if their variables are all different (by *all variables* is meant not only the elements of their domains but also the variables in the first components of their images).

  The disjoint union of two separated environments Γ and Δ (which is also an environment) is denoted $\Gamma \uplus \Delta$.

- Γ *can be connected with* Δ if

  - Γ, Δ is an environment,
  - for all declarations $i \in \Gamma$ and $j \in \Delta$ such that $i \sharp j$, there is $(x^y \propto R) \in \Gamma$ and $({}^x y \propto R) \in \Delta$ such that in both Γ and Δ, x and y transitively depend on the variables declared in i and j.

  Intuitively, this means that Γ, Δ is almost a disjoint union apart for those declarations that Γ and Δ have in common because they declare tube starts and tube ends that are used in the contents R of a tube xy opened in Γ and closed in Δ.

Definition 182 (Typing rules) Derivability of a judgement $\Gamma; A \vdash R : B$ in the typing system of Fig. 11.4 (which considers the rewrite rules —a.k.a. axioms— of Fig. 11.2) is denoted $\Gamma; A \vdash_{\mathsf{FB}} R : B$, and it means that the proof-term R, with tubes whose contents are given in Γ, is a proof of B assuming A.[4]

Remark 300 Linearity of variables is ensured by the fact that the rules are multiplicative in that they split the environment between premisses.

[4]Note that our notations $\Gamma; A \vdash R : B$ and $\Gamma; A \vdash_{\mathsf{FB}} R : B$ differ from [BL05] which denoted both of them $A \xrightarrow{R,\Gamma} B$, and also preferred ϵ to Γ.

$$\frac{}{;A \vdash \mathsf{id} : A} \qquad \frac{A \xrightarrow{\mathsf{root}}_{\rho} B}{;A \vdash \rho : B}$$

$$\frac{\Gamma; A \vdash R : C \quad \Delta; B \vdash S : D}{\Gamma \uplus \Delta; A \wedge B \vdash R \mid S : C \wedge D} \qquad \frac{\Gamma; A \vdash R : C \quad \Delta; B \vdash S : D}{\Gamma \uplus \Delta; A \vee B \vdash R \mid S : C \vee D}$$

$$\frac{\Gamma; A \vdash R : B}{\Gamma \uplus (x^y \propto R); A \vdash x :^A xy^B} \qquad \frac{\Gamma; A \vdash R : B}{\Gamma \uplus ({}^x y \propto R); {}^A xy^B \vdash y : B}$$

$$\frac{\Gamma; A \vdash R : B \quad \Delta; B \vdash S : C}{\Gamma, \Delta; A \vdash R.S : C} \ \Gamma \text{ can be connected to } \Delta$$

Figure 11.4: Typing rules for Formalism B

1. In the rules for parallel composition the disjoint union of environments is required to be separated, because the tubes of one side of the parallel composition are unrelated to those on the other side. This corresponds to the fact that we capture left- and right-linear rewrite systems (there is no branching of tubes, which would happen if rewrite rules used a meta-variable several times), so a tube can only go on one side of the parallel composition.

2. For the cut-rule we do not require Γ and Δ to be separated but only connectable because this is precisely how we connect the tubes in R to those in S.

Remark 301 As in Formalism A, associativity of sequential composition $(R_1.R_2).R_3 \sim R_1.(R_2.R_3)$ preserves typing, so again we consider terms up to associativity of sequential composition, and we write $R_1.R_2.R_3$ for (the equivalence class of) $(R_1.R_2).R_3$ and $R_1.(R_2.R_3)$.

Since a tube represent the space in which a sub-term can freely permute, there are two canonical positions where the sub-term could occur: at the start of the tube or at the end. Creating the tube and keeping the sub-term in the environment is a solution for not having to choose one of these two positions. But we can recover proof-terms with no tubes (i.e. proof-terms of Formalism A) by selecting one of these two positions as shown by the following lemma:

Lemma 302 *Let* $\Gamma, x?y \propto R$ *denote either* $\Gamma, x^y \propto R, {}^x y \propto R$ *or* $\Gamma, x^y \propto R$ *or* $\Gamma, {}^x y \propto R$ *or even just* Γ*, in all cases with* $x \notin \mathsf{Dom}(\Gamma)$ *and* $y \notin \mathsf{Dom}(\Gamma)$*.*

If $\Gamma, x\,?y \propto R; A \vdash_{\mathsf{FB}} R : B$ *then*

$$\left\{{}^{T,\mathsf{id}}\!/_{x,y}\right\}\Gamma; \left\{{}^{D}\!/_{C_{xy^D}}\right\} A \vdash_{\mathsf{FB}} \left\{{}^{T,\mathsf{id}}\!/_{x,y}\right\} R : \left\{{}^{D}\!/_{C_{xy^D}}\right\} B$$

and

$$\left\{{}^{\mathsf{id},T}\!/_{x,y}\right\}\Gamma; \left\{{}^{C}\!/_{C_{xy^D}}\right\} A \vdash_{\mathsf{FB}} \left\{{}^{\mathsf{id},T}\!/_{x,y}\right\} R : \left\{{}^{C}\!/_{C_{xy^D}}\right\} B$$

Proof: By induction on the derivation of the premiss. □

Such a transformation can be applied recursively until Γ is empty; this is what we use for the soundness theorem.

As in Formalism A, we have soundness and completeness for classical propositional logic since we have the following theorem:

Theorem 303 (Soundness & completeness) *For all formulas* A, B *there is a proof-term* R *of Formalism* A *with* $A \vdash_{\mathsf{FA}} R : B$ *if and only if there is a proof-term* T *of Formalism* B *with* $\Gamma; A \vdash_{\mathsf{FB}} T : B$.

Proof: The direction from left to right is obvious, since the typing rules of Formalism A are also typing rules of Formalism B with an empty environment. To prove the converse we start by using Lemma 302 to empty the environment Γ, and because A and B are formulae, they are unaffected by this transformation. Then it suffices to notice that sequents with empty environments can only be derived with those rules of Formalism B that are already in Formalism A. □

11.3.2 Reduction

To obtain a bureaucracy-free representative of a proof, we start from a proof term in Formalism A. The normalisation process has three stages.

The **first stage** is an initialisation: it is a one step transformation of a term that adds to each combinator its *inner tubes*. Given a combinator representing a rewrite rule, we create one tube for every meta-variables of the rewrite rule, that starts just before the combinator and ends just after it. For instance, $\mathsf{co}{\downarrow}$ is replaced by $(x \mid x') \,.\, \mathsf{co}{\downarrow} \,.\, (y' \mid y)$ (with tubes xy and $x'y'$). This is where the requirement that the rewrite system is left- and right-linear is used. We create the environment to map all tubes to id.

The **second stage** extends tubes as much as possible. It is given by the rewrite system FB of Fig. 11.5 (which includes the rules of Formalism A), which rewrites both a term and on an environment. We write $\Gamma, xy \propto R$ for $\Gamma, x^y \propto R, {}^x y \propto R$. We refer to the second and third rules of Fig. 11.5 as *tube loading* or *tube extension* and to the fourth rule as *tube fusion*.

$$\begin{array}{lll} T \\ \Gamma \end{array} \longrightarrow \begin{array}{l} T' \\ \Gamma \end{array} \qquad \text{if } T \longrightarrow_{\mathsf{FA}} T'$$

if $R = (T \mid U)$ or $R = \rho$

$$\begin{array}{l} y \,.\, R \\ \Gamma, xy \propto T \end{array} \longrightarrow \begin{array}{l} x \\ \Gamma, xy \propto T.R \end{array}$$

$$\begin{array}{l} R \,.\, x \\ \Gamma, xy \propto T \end{array} \longrightarrow \begin{array}{l} x \\ \Gamma, xy \propto R.T \end{array}$$

$$\begin{array}{l} y \,.\, x \\ \Gamma, x'y \propto T, xy' \propto U \end{array} \longrightarrow \begin{array}{l} \mathsf{id} \\ \Gamma, x'y' \propto T.U \end{array}$$

Figure 11.5: System FB

The notion of contextual closure is given explicitly as follows:

$$\frac{\begin{array}{l} T \\ \Gamma \end{array} \longrightarrow \begin{array}{l} T' \\ \Gamma' \end{array}}{\begin{array}{l} S\{T\} \\ \Gamma \end{array} \longrightarrow \begin{array}{l} S\{T'\} \\ \Gamma' \end{array}} \qquad \frac{\begin{array}{l} T \\ \Gamma \end{array} \longrightarrow \begin{array}{l} T' \\ \Gamma' \end{array}}{\begin{array}{l} R \\ \Gamma, xy \propto T \end{array} \longrightarrow \begin{array}{l} R \\ \Gamma', xy \propto T' \end{array}}$$

where the first rule abbreviates the four rules corresponding to the four cases of sub-terms (reduction can be performed within the components of parallel and sequential composition).

The **third stage** is a cleanup phase, when all empty tubes are discarded (the reduction rule is subject to the same contextual closure as above):

$$\begin{array}{l} R \\ \Gamma, xy \propto \mathsf{id} \end{array} \longrightarrow \begin{array}{l} \left\{{}^{\mathsf{id},\mathsf{id}}\!\!\diagup_{x,y}\right\} R \\ \left\{{}^{\mathsf{id},\mathsf{id}}\!\!\diagup_{x,y}\right\} \Gamma \end{array}$$

Example 15 (Reducing Bureaucracy B)

- The minimal example are the terms $(\mathsf{id} \mid \mathsf{ac}{\downarrow}) \,.\, \mathsf{co}{\downarrow}$ and $\mathsf{co}{\downarrow} \,.\, (\mathsf{ac}{\downarrow} \mid \mathsf{id})$ that both rewrite to:
$$(\mathsf{id} \mid x) \,.\, \mathsf{co}{\downarrow} \,.\, (y \mid \mathsf{id}) \quad , \quad xy \propto \mathsf{ac}{\downarrow}$$
- More than one rule can be inside a tube. $(\mathsf{id} \mid \mathsf{ac}{\downarrow}) \,.\, \mathsf{co}{\downarrow} \,.\, (\mathsf{ac}{\uparrow} \mid \mathsf{id})$ rewrites to:
$$(\mathsf{id} \mid x) \,.\, \mathsf{co}{\downarrow} \,.\, (y \mid \mathsf{id}) \quad , \quad xy \propto \mathsf{ac}{\downarrow} \,.\, \mathsf{ac}{\uparrow}$$
- Tubes can be nested. $((\mathsf{ac}{\downarrow} \mid \mathsf{id}) \mid \mathsf{id}) \,.\, \mathsf{co}{\downarrow} \,.\, (\mathsf{id} \mid \mathsf{co}{\downarrow})$ rewrites to:
$$(x \mid \mathsf{id}) \,.\, \mathsf{co}{\downarrow} \,.\, (\mathsf{id} \mid y) \quad , \quad \begin{array}{l} xy \propto (x' \mid \mathsf{id}) \,.\, \mathsf{co}{\downarrow} \,.\, (\mathsf{id} \mid y') \\ x'y' \propto \mathsf{ac}{\downarrow} \end{array}$$

The reduction relation preserves types:

Theorem 304 (Subject reduction) *If* $\Gamma; A \vdash_{FB} U : B$ *and* $U, \Gamma \longrightarrow_{FB} U', \Gamma'$ *then* $\Gamma'; A \vdash_{FB} U' : B$.

Proof: It is easy to check that the first and third stage preserve typing, we give the necessary transformation of the typing derivation for tube extension in the second stage. Tube fusion works similarly. The derivation

$$\dfrac{\dfrac{\Gamma_1; A \vdash T : B}{\Gamma_1, x^y \propto T; A \vdash x : {}^A xy^B} \qquad \dfrac{\dfrac{\Gamma_1; A \vdash T : B}{\Gamma_1, {}^x y \propto T; {}^A xy^B \vdash y : B} \qquad \Gamma_2; B \vdash R : C}{\Gamma_1, \Gamma_2, {}^x y \propto T; {}^A xy^B \vdash y \,.\, R : C}}{\Gamma, xy \propto T; D \vdash U : E}\ \Delta$$

is transformed into

$$\dfrac{\dfrac{\dfrac{\Gamma_1; A \vdash T : B \quad \Gamma_2; B \vdash R : C}{\Gamma_1, \Gamma_2; A \vdash T.R : C}}{\Gamma_1, \Gamma_2, x^y \propto T.R; A \vdash x : {}^A xy^C} \qquad \dfrac{\dfrac{\Gamma_1; A \vdash T : B \quad \Gamma_2; B \vdash R : C}{\Gamma_1, \Gamma_2; A \vdash T.R : C}}{\Gamma_1, \Gamma_2, {}^x y \propto T.R; {}^A xy^C \vdash y : C}}{\Gamma, xy \propto T.R; D \vdash U' : E}\ \left\{ {}^{A_{xy}C} \big/ {}_{A_{xy}B} \right\} \Delta$$

□

Theorem 305 (Termination) *The normalisation process is terminating.*

Proof: The first stage is a one-step transformation, there is no termination issue there. We then show that the second and third stages, even mixed together, terminate. First, note that the number of tubes decrease. We define a measure on the pairs consisting of a term and a environment.

For that we assign to the term and to each declared variable a number p: To the term we affect the total number of variables $2 \cdot n$, where n is the total number of tubes. Then we look at the dependency graph of the environment: to each variable we affect $2 \cdot n - 1 - i$, where i is the length of the longest path in the graph that reaches the variable (for instance, if no variable depends on x, such a length is $i = 0$ and we assign $2 \cdot n - 1$ to x). Because we have safely started with $2 \cdot n$, all the above numbers are positive.

To the term (resp. to each declared variable x or y) we assign another number, say q, that is an extension of the measure used to prove termination of reduction in Formalism A: it is the total number of parallel compositions, id constructors, combinators and variables in the term (resp. the term in the declaration of x or y).

The measure of the pair is the sum of all the $2^n \cdot q$ corresponding to the term and each declared variable.

Then it suffices to check all reduction rules to show that reduction decrease pairs w.r.t. this measure. □

Conjecture 306 (Confluence) *The normalisation process is confluent (modulo naming of tubes given in the first stage).*

Conclusion

In this chapter we investigated the notion of normalisation and equivalence of classical proofs. The approach consisted in starting from CoS [Gug, Gug02], where proofs are reduction sequences. The reduction relation is given by rewrite rules on formulae, which can be presented as inference rules that have exactly one premiss (and are contextually closed).

Doing so serves the purpose of defining a notion of proof equivalence based on the permutation of independent inference steps, similar to the permutations of [DP99b] for intuitionistic (cut-free) sequent calculus. Here, by avoiding the branching pertaining to sequent calculus derivations (or also those of natural deduction), we could identify such permutations as the permutations of those reduction steps whose redexes do not form critical pairs (used e.g. in parallel reductions [Tak89] or finite developments).

These permutations form an acceptable notion of equivalence on classical proofs when expressed as reduction sequences in a left- and right-linear rewrite system, where a residual of a redex is always unique. Such a system for classical logic was presented in section 11.1, taken from [Brü03].

Building on [Gug04a, Gug04b], we introduced two formalisms, called Formalism A and Formalism B, to provide canonical representatives for equivalence classes of proofs with respect to these permutations. We formalised each of them with proof-terms and a typing system, which is in fact a sequent calculus with

axioms. Because of these axioms, the cut-rule cannot be eliminated (it is in fact the main tool for combining axioms to form derivations). However, cut-reduction is precisely the procedure that provides canonical representative of equivalence classes of proofs; it is indeed terminating and confluent (confluence in the case of Formalism B is only conjectured).

Moving from a classical sequent calculus to a sequent calculus with axioms allows the notion of proof equivalence to be based on the normalisation process (in our case, cut-reduction), as in intuitionistic logic. Whether or not such a move sacrifices good properties of (axiom-free) sequent calculus (e.g. for proof-search), and whether or not it provides better properties, is discussed in [Gug, Gug02] (at least for CoS).

Further work also includes formalising how Formalism A and Formalism B actually do capture Bureaucracy A and Bureaucracy B, e.g. by showing that canonical terms are in one-to-one correspondence with equivalence classes of proofs. Alternative presentations of Formalism A and Formalism B could take two directions:

- Our typing system for Formalism B has the drawback that the contents of tubes have to be typed twice in a derivation: once at the start of a tube and once at the end. This redundancy is what prevented us from defining the cut-rule with the requirement that environments should simply have disjoint domains, leading to the notion of connectability. It would thus be interesting to develop a typing system where the contents of tubes have to be type checked only once. This would hopefully simplify the cut-rule.

- We chose Curry-style typing for brevity, but it could be done in Church-style. Then we need two parallel constructors, one for conjunction and one for disjunction. All combinators are then parameterised by their types, as well as id. Church-style could be more convenient for type-checking.

Connections with the literature are numerous but remain to be investigated:

Our approach seems close to rewriting logic [MOM02]: our goal with Formalism B is to give canonical representatives for arrows in the initial model of a rewrite theory if the latter is linear and without equations. The deductive system for rewriting logic as given in [MOM02] seems close to Formalism A (its congruence rule corresponds to our rule for parallel composition), but it does not provide canonical representatives for Bureaucracy B.

The definition and/or treatment of Bureaucracy A and Bureaucracy B in terms of the rewriting notions of parallel reduction, finite developments, and residual theory remain to be formalised, maybe in the light of [Mel97, Mel98, Mel02].

It should be mentioned that, beyond the two kinds of bureaucracy treated by Formalism A and Formalism B for the purpose of proof equivalence, these formalisms feature other kinds of bureaucracy. First, Formalism B introduces some new bureaucracy in the choice of tube names, but note that this is also

featured by α-equivalence in any formalism using a higher-order term syntax for proofs. Second, sequential composition is associative, but an n-ary constructor can provide canonical representatives. Third, we might like to make parallel composition associative and commutative (AC), as in process calculi.

Associativity of parallel composition could also be useful for the denotational semantics of our proofs, since it seems that these are to be found in *3-categories* à la Albert Burroni [Bur93] (as suggested by the connections, investigated in [Gui05], between CoS and 3-categories).

So far we cannot make parallel composition associative and/or commutative, since parallel compositions mimic the tree-structure of formulae. Defining a system with an AC parallel composition would probably either break the connection (found e.g. in proof nets) between the tree-structure of formulae and that of proofs, or abandon the tree-structure of formulae and maybe, using the AC of conjunction and disjunction in classical logic, opt for a graph notion such as the *relation webs* of [Gug02].

However, it can be argued that the AC of conjunction and disjunction is a kind of bureaucracy pertaining not to the proof system, but to the logic, along with other type isomorphisms —as in (the categorical semantics of) intuitionistic logic. In this chapter, classical logic and the rewrite system SKSfl can be considered just as an example of our approach, we really are addressing bureaucracy in the proof system, which is independent from the particular logic that we are formalising. For the attack on logic-independent bureaucracy two obvious directions for further work are the extension of our approach to term rewriting systems in general, not only those with linear rules, and to term rewriting systems modulo equations.

Finally, using the approach of this chapter in order to build classical type theories (especially with dependent types) is one of the main directions for further work. The first step would be to redefine system $F^{\mathcal{C}}_{\omega}$ with a layer of proof-terms such as that of Formalism B (equivalently, extend the typing system of Formalism B with polymorphism and type constructors). The second step would then be to make type constructors depend on these proof-terms, using the notion of normalisation of the latter in order to define the notion of convertibility of type constructors.

Conclusion & further work

This dissertation addressed a wide range of topics related to the Curry-Howard correspondence. It developed its concepts in various directions: Part I investigated the relationship between higher-order calculi that form a Curry-Howard correspondence with natural deduction on the one hand and with sequent calculus on the other hand. Part II introduced the formalism of Pure Type Sequent Calculi, with a thorough study of their properties and their variants, especially for proof synthesis and type synthesis. Part III introduced a particular classical type theory and investigated an approach to the issue of equivalence of classical proofs.

In almost all chapters, there are directions for further work:

- From Chapter 2 we would like to have a general framework for logical systems, with generic definitions for additive and multiplicative systems, principal formulae,... This would allow a much more general expression of the Curry-Howard correspondence, for instance independently from a particular logic.

- From Chapter 4 there are connections to be investigated between the safeness and minimality technique and dependency pairs (see e.g. [AG00]). The technique of simulation in λI should also be related to other works about normalisation results in λI, such as in [Sør97, Xi97].

- From Chapter 5, several directions could be taken. The first one would be the study of the notion of perpetuality in $\lambda\mathsf{lxr}$, since the power of composition of substitution makes major changes to the notion of perpetuality of, say, $\lambda\mathsf{x}$ [Bon01]. Interesting properties of λI w.r.t. perpetuality should be connected to $\lambda\mathsf{lxr}$, by strengthening the link established by the proof of PSN. For instance, the memory operator of λI can be represented in $\lambda\mathsf{lxr}$ as a $\mathsf{Weak1}$-redex that is not reduced.

  Together with the notion of perpetuality can be mentioned the characterisation of strongly normalising terms by a typing system with intersection types. Not only does perpetuality play a key role in establishing such a characterisation (see e.g. [LLD$^+$04]), but the power of composition of $\lambda\mathsf{lxr}$ should also allow the characterisation with the simple rules for intersections

that are given in Fig. 4.4, instead of having to add an *ad hoc* rule such as the one needed for $\lambda\mathsf{x}$ in Fig. 4.5.

Moreover, the contraction constructor of $\lambda\mathsf{lxr}$ is particularly adequate for a left-introduction of intersection types: the purpose of typing a variable with an intersection is to use it in different places with different types, and the role of the contraction constructor is precisely to make explicit those points where several variables merge into one, which must then have the type of each of them.

Finally, this suggests also to use the same methodology for a multiplicative sequent calculus such as G1ii, with primitive weakening and contraction rules. The connection between an explicit contraction rule and rule select_x of LJT, which also represents a contraction, should be investigated.

- From Chapter 6, we should start by proving the conjectures about confluence of the CBN and CBV systems, in the three cases of propagation systems.

  Another direction for further work is to develop the same ideas in natural deduction as we did for G3ii, probably using the calculus of [Esp05] to capture CBN and CBV in a unified natural deduction framework.

  Also, understanding in the setting of G3ii what η-conversion exactly corresponds to, in particular in terms of whether or not axioms should be restricted to type variables, is also a direction for further work.

  Finally, including the notions of this chapter in a more general approach with classical implicational logic as its starting point could be in fact simpler and enlightening, because the symmetries of classical logic would explain phenomena that might look strange in the asymmetric world of intuitionistic logic.

- From Chapter 7, directions for further work could be the investigation of direct relations between our higher-order calculus for G4ii, with its various notions of reduction, and λ-calculus. There is a semantical mismatch between the reductions of one and those of the other, which needs clarification. Also, it would be interesting to know, for each type, which are the λ-terms that our calculus for G4ii encode.

- Chapter 8 and Chapter 9 could give the most promising directions for further research. The theory of the system optimised for proof synthesis, with higher-order variables, needs to be further developed. Its ability to encode proof synthesis with or without delaying the sub-goals leaves a great freedom for tactics, depending on whether there is a user interaction or we only want an algorithm that enumerates inhabitants of types (or even mixing the two). Its potential for expressing and performing higher-order unification,

as in [Dow93], should be studied as such, as well as the connections with the algorithms using explicit substitutions [DHK95].

Also, in order to be used in implemented tools, we should turn this system into a version with de Bruijn indices, as we did for the case without higher-order variables.

Concerning the theory itself, two extensions can be investigated: developing a Gentzen-style sequent calculus for Pure Type Systems (i.e. based on G3ii rather than LJT, and maybe even on LJQ and G4ii to benefit from the proof-search capabilities of the latter), and also deal with *inductive types*, such as those used in Coq.

- From Chapter 10 the first and foremost objective is to solve the two conjectures. Solving Conjecture 293, probably requiring some minor technical effort, and Conjecture 284, requiring a counter-example, are both ongoing work.

- Chapter 11 leaves one conjecture (the confluence of the normalisation process) as further work. Beyond that, alternative and more elegant presentations of tubes could be investigated, since the current one is somewhat cumbersome. Connections with other works in the literature have also been mentioned as further work. Finally we would like to use the proof system that we have for classical logic in a more developed type theory, eventually using dependent types with the notions of equivalence on proof-terms developed in this chapter.

Bibliography

[Abr93] S. Abramsky. Computational interpretations of linear logic. *Theoret. Comput. Sci.*, 111:3–57, 1993.

[ABR00] A. Arbiser, E. Bonelli, and A. Ríos. Perpetuality in a lambda calculus with explicit substitutions and composition. Workshop Argentino de Informática Teórica (WAIT), JAIIO, 2000.

[ACCL91] M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Lévy. Explicit substitutions. *J. Funct. Programming*, 1(4):375–416, 1991.

[AG98] A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages*, volume 45 of *Cambridge Tracts in Theoret. Comput. Sci.* Cambridge University Press, 1998.

[AG00] T. Arts and J. Giesl. Termination of term rewriting using dependency pairs. *Theoret. Comput. Sci.*, 236(1-2):133–178, 2000.

[AL94] A. Asperti and C. Laneve. Interaction systems i: The theory of optimal reductions. *Math. Structures in Comput. Sci.*, 4(4):457–504, 1994.

[Bar84] H. P. Barendregt. *The Lambda-Calculus, its syntax and semantics*. Studies in Logic and the Foundation of Mathematics. Elsevier, 1984. Second edition.

[Bar91] H. P. Barendregt. Introduction to generalized type systems. *J. Funct. Programming*, 1(2):125–154, 1991.

[Bar92] H. P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabby, and T. S. E. Maibaum, editors, *Hand. Log. Comput. Sci.*, volume 2, chapter 2, pages 117–309. Oxford University Press, 1992.

[BB96] F. Barbanera and S. Berardi. A symmetric lambda-calculus for classical program extraction. *Inform. and Comput.*, 125(2):103–117, 1996.

[BBdH93] N. Benton, G. Bierman, V. de Paiva, and M. Hyland. A term calculus for intuitionistic linear logic. In J. F. G. Groote and M. Bezem, editors, *Proc. of the 1st Int. Conf. on Typed Lambda Calculus and Applications*, volume 664 of *LNCS*, pages 75–90. Springer-Verlag, 1993.

[BBLRD96] Z. Benaissa, D. Briaud, P. Lescanne, and J. Rouyer-Degli. $\lambda\upsilon$, a calculus of explicit substitutions which preserves strong normalisation. *J. Funct. Programming*, 6(5):699–722, 1996.

[BG99] R. Bloo and H. Geuvers. Explicit substitution: on the edge of strong normalization. *Theoret. Comput. Sci.*, 211(1-2):375–395, 1999.

[BG01] H. Barendregt and H. Geuvers. Proof-assistants using dependent type systems. In J. A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, pages 1149–1238. Elsevier and MIT Press, 2001.

[BHS97] G. Barthe, J. Hatcliff, and M. H. Sørensen. A notion of classical pure type system. In S. Brookes, M. Main, A. Melton, and M. Mislove, editors, *Proc. of the 13th Annual Conf. on Math. Foundations of Programming Semantics, MFPS'97*, volume 6 of *ENTCS*, pages 4–59. Elsevier, 1997.

[BKR00] E. Bonelli, D. Kesner, and A. Rios. A de Bruijn notation for higher-order rewriting. In L. Bachmair, editor, *Proc. of the 11th Int. Conf. on Rewriting Techniques and Applications(RTA'00)*, volume 1833 of *LNCS*, pages 62–79. Springer-Verlag, July 2000.

[BL05] K. Brünnler and S. Lengrand. On two forms of bureaucracy in derivations. In F. L. Paola Bruscoli and J. Stewart, editors, *1st Work. on Structures and Deductions (SD'05)*, Technical Report, Technische Universität Dresden, pages 69–80, July 2005. ISSN 1430-211X.

[Blo01] R. Bloo. Pure type systems with explicit substitution. *Math. Structures in Comput. Sci.*, 11(1):3–19, 2001.

[BN98] F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.

[Bon01] E. Bonelli. Perpetuality in a named lambda calculus with explicit substitutions. *Math. Structures in Comput. Sci.*, 11(1), 2001.

[BR95] R. Bloo and K. H. Rose. Preservation of strong normalisation in named lambda calculi with explicit substitution and garbage collection. In J. van Vliet, editor, *Computing Science in the Netherlands (CSN '95)*, pages 62–72, November 1995.

[Brü03] K. Brünnler. *Deep Inference and Symmetry in Classical Proofs*. PhD thesis, Technische Universität Dresden, 2003.

[BT01] K. Brünnler and A. F. Tiu. A local system for classical logic. In R. Nieuwenhuis and A. Voronkov, editors, *LPAR 2001*, volume 2250 of *LNCS*, pages 347–361. Springer-Verlag, 2001.

[Bur93] A. Burroni. Higher-dimensional word problems with applications to equational logic. *Theoret. Comput. Sci.*, 115(1):43–62, 1993.

[CD78] M. Coppo and M. Dezani-Ciancaglini. A new type assignment for lambda-terms. *Archive f. math. Logic u. Grundlagenforschung*, 19:139–156, 1978.

[CDG+97] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 1997. release October, 1rst 2002. Available at `http://www.grappa.univ-lille3.fr/tata`.

[CH88] T. Coquand and G. Huet. The calculus of constructions. *Inform. and Comput.*, 76(2–3):95–120, 1988.

[CH00] P.-L. Curien and H. Herbelin. The duality of computation. In *Proc. of the 5^{th} ACM SIGPLAN Int. Conf. on Functional Programming (ICFP'00)*, pages 233–243. ACM Press, 2000.

[Che04] D. Chemouil. *Types inductifs, isomorphismes et récriture extensionnelle*. PhD thesis, Université Paul Sabatier – Toulouse 3, 2004.

[CHL96] P.-L. Curien, T. Hardin, and J.-J. Lévy. Confluence properties of weak and strong calculi of explicit substitutions. *J. of the ACM Press*, 43(2):362–397, 1996.

[Chu41] A. Church. *The Calculi of Lambda Conversion*. Princeton University Press, 1941.

[CK99] S. Cerrito and D. Kesner. Pattern matching as cut elimination. In G. Longo, editor, *14th Annual IEEE Symp. on Logic in Computer Science*, pages 98–108. IEEE Computer Society Press, July 1999.

[Coq] The Coq Proof Assistant. Available at `http://coq.inria.fr/`.

[Coq94] T. Coquand. An analysis of Ramsey's theorem. *Inform. and Comput.*, 110(2):297–304, 1994.

[dB72] N. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indag. Mathematicae*, 5(35):381–392, 1972.

[DCG99] R. Di Cosmo and S. Guerrini. Strong normalization of proof nets modulo structural congruences. In P. Narendran and M. Rusinowitch, editors, *Proc. of the 10th Int. Conf. on Rewriting Techniques and Applications(RTA'99)*, volume 1631 of *LNCS*, pages 75–89. Springer-Verlag, July 1999.

[DCKP00] R. Di Cosmo, D. Kesner, and E. Polonovski. Proof nets and explicit substitutions. In J. Tiuryn, editor, *Foundations of Software Science and Computation Structures (FOSSACS)*, volume 1784 of *LNCS*, pages 63–81. Springer-Verlag, March 2000.

[DCKP03] R. Di Cosmo, D. Kesner, and E. Polonovski. Proof nets and explicit substitutions. *Math. Structures in Comput. Sci.*, 13(3):409–450, 2003.

[Der82] N. Dershowitz. Orderings for term-rewriting systems. *Theoret. Comput. Sci.*, 17:279–301, 1982.

[DG01] R. David and B. Guillaume. A λ-calculus with explicit weakening and explicit substitution. *Math. Structures in Comput. Sci.*, 11:169–206, 2001.

[DGLL05] D. J. Dougherty, S. Ghilezan, P. Lescanne, and S. Likavec. Strong normalization of the dual classical sequent calculus. In G. Sutcliffe and A. Voronkov, editors, *Proc. of the 12th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'05)*, volume 3835 of *LNCS*, pages 169–183. Springer-Verlag, December 2005.

[DHK95] G. Dowek, T. Hardin, and C. Kirchner. Higher-order unification via explicit substitutions. In D. Kozen, editor, *Proc. of the 10th Annual Symp. on Logic in Computer Science*, pages 366–374. IEEE Computer Society Press, June 1995.

[DHKP96] G. Dowek, T. Hardin, C. Kirchner, and F. Pfenning. Unification via explicit substitutions: The case of higher-order patterns. In M. Maher, editor, *Int. Joint Conf. and Symp. on Logic Programming*, pages 259–273. The MIT press, September 1996.

[DJS95] V. Danos, J.-B. Joinet, and H. Schellinx. Lkq and lkt: sequent calculi for second order logic based upon dual linear decompositions of classical implication. In J.-Y. Girard, Y. Lafont, and L. Regnier, editors, *Proc. of the Work. on Advances in Linear Logic*, volume 222 of *London Math. Soc. Lecture Note Ser.*, pages 211–224. Cambridge University Press, 1995.

[DKL06] R. Dyckhoff, D. Kesner, and S. Lengrand. Strong cut-elimination systems for Hudelmaier's depth-bounded sequent calculus for implicational logic. In U. Furbach and N. Shankar, editors, *Proc. of the 3rd Int. Joint Conf. on Automated Reasoning (IJCAR'06)*, volume 4130 of *LNAI*, pages 347–361. Springer-Verlag, August 2006.

[DL03] D. Dougherty and P. Lescanne. Reductions, intersection types, and explicit substitutions. *Math. Structures in Comput. Sci.*, 13(1):55–85, 2003.

[DL06] R. Dyckhoff and S. Lengrand. **LJQ**, a strongly focused calculus for intuitionistic logic. In A. Beckmann, U. Berger, B. Loewe, and J. V. Tucker, editors, *Proc. of the 2nd Conf. on Computability in Europe (CiE'06)*, volume 3988 of *LNCS*, pages 173–185. Springer-Verlag, July 2006.

[DM79] N. Dershowitz and Z. Manna. Proving termination with multiset orderings. *Communications of the ACM*, 22(8):465–476, 1979.

[DN00] R. Dyckhoff and S. Negri. Admissibility of structural rules for contraction-free systems of intuitionistic logic. *J. of Symbolic Logic*, 65(4):1499–1518, 2000.

[DN05a] R. David and K. Nour. Arithmetical proofs of strong normalization results for the symmetric $\lambda\mu$. In P. Urzyczyn, editor, *Proc. of the 9th Int. Conf. on Typed Lambda Calculus and Applications (TLCA'05)*, volume 3461 of *LNCS*, pages 162–178. Springer-Verlag, April 2005.

[DN05b] R. David and K. Nour. Why the usual candidates of reducibility do not work for the symmetric $\lambda\mu$-calculus. In P. Lescanne, R. David, and M. Zaionc, editors, *Post-proc. of the 2nd Work. on Computational Logic and Applications (CLA'04)*, volume 140 of *ENTCS*, pages 101–111. Elsevier, 2005.

[Dou06] D. Dougherty. Personal communication, August 2006.

[Dow93] G. Dowek. A complete proof synthesis method for type systems of the cube. *J. Logic Comput.*, 3(3):287–315, 1993.

[DP99a] R. Dyckhoff and L. Pinto. Proof search in constructive logics. In *Sets and proofs (Leeds, 1997)*, pages 53–65. Cambridge University Press, 1999.

[DP99b] R. Dyckhoff and L. Pinto. Permutability of proofs in intuitionistic sequent calculi. *Theoret. Comput. Sci.*, 212(1–2):141–155, 1999.

[DU03] R. Dyckhoff and C. Urban. Strong normalization of Herbelin's explicit substitution calculus with substitution propagation. *J. Logic Comput.*, 13(5):689–706, 2003.

[Dyc92] R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. *J. of Symbolic Logic*, 57(3):795–807, 1992.

[EFP06] J. Espírito Santo, M. J. Frade, and L. Pinto. Structural proof theory as rewriting. In F. Pfenning, editor, *Proc. of the 17th Int. Conf. on Rewriting Techniques and Applications(RTA'06)*, volume 4098 of *LNCS*, pages 197–211. Springer-Verlag, August 2006.

[Esp02] J. Espírito Santo. *Conservative extensions of the lambda-calculus for the computational interpretation of sequent calculus*. PhD thesis, University of Edinburgh, 2002.

[Esp05] J. Espírito Santo. Unity in structural proof theory and structural extensions of the λ-calculus. July 2005. Manuscript. Available at `http://www.math.uminho.pt/~jes/Publications.htm`.

[Fis72] M. J. Fischer. Lambda calculus schemata. In *Proc. of the ACM Conf. on Proving Assertions about Programs*, pages 104–109. SIGPLAN Notices, Vol. 7, No 1 and SIGACT News, No 14, January 1972.

[Fis93] M. J. Fischer. Lambda-calculus schemata. *LISP and Symbolic Computation*, 6(3/4):259–288, 1993.

[For02] J. Forest. A weak calculus with explicit operators for pattern matching and substitution. In S. Tison, editor, *Proc. of the 13th Int. Conf. on Rewriting Techniques and Applications(RTA'02)*, volume 2378 of *LNCS*, pages 174–191. Springer-Verlag, July 2002.

[FP06] C. Fürmann and D. Pym. Order-enriched categorical models of the classical sequent calculus. *J. Pure Appl. Algebra*, 204(1):21–78, 2006.

[GdR00] N. Ghani, V. de Paiva, and E. Ritter. Linear explicit substitutions. *Logic J. of the IGPL*, 8(1):7–31, 2000.

[Gen35] G. Gentzen. Investigations into logical deduction. In *Gentzen collected works*, pages 68–131. Ed M. E. Szabo, North Holland, (1969), 1935.

[Gir72] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures de l'arithmétique d'ordre supérieur*. Thèse d'état, Université Paris 7, 1972.

[Gir87] J.-Y. Girard. Linear logic. *Theoret. Comput. Sci.*, 50(1):1–101, 1987.

[GL98] J. Goubault-Larrecq. A proof of weak termination of typed lambda sigma-calculi. In T. Altenkirch, W. Naraschewski, and B. Reus, editors, *Proc. of the Int. Work. Types for Proofs and Programs*, volume 1512 of *LNCS*, pages 134–151. Springer-Verlag, December 1998.

[GR03a] F. Gutiérrez and B. C. Ruiz. Expansion postponement via cut elimination in sequent calculi for pure type systems. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proc. of the 30th Intern. Col. on Automata, Languages and Programming (ICALP)*, volume 2719 of *LNCS*, pages 956–968. Springer-Verlag, July 2003.

[GR03b] F. Gutiérrez and B. Ruiz. Cut elimination in a class of sequent calculi for pure type systems. In R. de Queiroz, E. Pimentel, and L. Figueiredo, editors, *Proc. of the 10th Work. on Logic, Language, Information and Computation (WOLLIC'03)*, volume 84 of *ENTCS*. Elsevier, August 2003.

[GTL89] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and Types*, volume 7 of *Cambridge Tracts in Theoret. Comput. Sci.* Cambridge University Press, 1989.

[Gug] A. Guglielmi. The calculus of structures website. Available at `http://alessio.guglielmi.name/res/cos/index.html`.

[Gug02] A. Guglielmi. A system of interaction and structure. Technical Report WV-02-10, Technische Universität Dresden, 2002. To appear in ACM Transactions on Computational Logic.

[Gug04a] A. Guglielmi. Formalism A. 2004. Manuscript. Available at `http://iccl.tu-dresden.de/~guglielm/p/AG11.pdf`.

[Gug04b] A. Guglielmi. Formalism B. 2004. Manuscript. Available at `http://iccl.tu-dresden.de/~guglielm/p/AG13.pdf`.

[Gui05] Y. Guiraud. The three dimensions of proofs. 2005. to appear in Annals of pure and applied logic.

[Has99] M. Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of* `let` *and* `letrec`. Distinguished Dissertation Series. Springer-Verlag, 1999. PhD Thesis.

[Her94] H. Herbelin. A lambda-calculus structure isomorphic to Gentzen-style sequent calculus structure. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic, 8th Int. Work. , CSL '94*, volume 933 of *LNCS*, pages 61–75. Springer-Verlag, September 1994.

[Her95] H. Herbelin. *Séquents qu'on calcule.* Thèse de doctorat, Université Paris 7, 1995.

[HL89] T. Hardin and J.-J. Lévy. A confluent calculus of substitutions. In *France-Japan Artificial Intelligence and Computer Science Symposium*, 1989.

[HMP96] T. Hardin, L. Maranget, and B. Pagano. Functional back-ends within the lambda-sigma calculus. In R. K. Dybvig, editor, *Proc. of the ACM International Conference on Functional Programming.* ACM Press, May 1996.

[HOL] The HOL system. Available at `http://www.dcs.gla.ac.uk/~tfm/fmt/hol.html`.

[How80] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, 1980. Reprint of a manuscript written 1969.

[Hud89] J. Hudelmaier. *Bounds for Cut Elimination in Intuitionistic Logic.* PhD thesis, Universität Tübingen, 1989.

[Hud92] J. Hudelmaier. Bounds on cut-elimination in intuitionistic propositional logic. *Arch. Math. Log.*, 31:331–354, 1992.

[Hue76] G. Huet. *Résolution d'équations dans les langages d'ordre* $1, 2, \ldots, \omega$. Thèse d'état, Université Paris 7, 1976.

[Hue89] G. Huet. The constructive engine. *World Scientific Publishing*, Commemorative Volume for Gift Siromoney, 1989.

[HvO03] D. Hendriks and V. van Oostrom. Adbmal. In F. Baader, editor, *Proc. of the 19th Conf. on Automated Deduction (CADE 19)*, volume 2741 of *LNAI*, pages 136 – 150. Springer-Verlag, July-August 2003.

[JM03] F. Joachimski and R. Matthes. Short proofs of normalization for the simply- typed lambda-calculus, permutative conversions and Gödel's T . *Arch. Math. Log.*, 42(1):59–87, 2003.

[JR99] J.-P. Jouannaud and A. Rubio. The higher-order recursive path ordering. In G. Longo, editor, *14th Annual IEEE Symp. on Logic in Computer Science*, pages 402–411. IEEE Computer Society Press, July 1999.

[Kah92] S. Kahrs. Context rewriting. In M. Rusinowitch and J.-L. Rémy, editors, *CTRS*, volume 656 of *LNCS*, pages 21–35. Springer-Verlag, July 1992.

[Kah04] O. Kahramanoğullari. Implementing system BV of the calculus of structures in Maude. In L. A. i Alemany and P. Égré, editors, *Proc. of the ESSLLI-2004 Student Session*, pages 117–127, 2004.

[Kha90] Z. Khasidashvili. Expression reduction systems. In *Proc. of the IN Vekua Institute of Applied Mathematics*, volume 36, 1990.

[Kik04a] K. Kikuchi. A direct proof of strong normalization for an extended Herbelin's calculus. In Y. Kameyama and P. J. Stuckey, editors, *Proc. of the 7th Int. Symp. on Functional and Logic Programming (FLOPS'04)*, volume 2998 of *LNCS*, pages 244–259. Springer-Verlag, April 2004.

[Kik04b] K. Kikuchi. Personal communication, July 2004.

[Kik06] K. Kikuchi. On a local-step cut-elimination procedure for the intuitionistic sequent calculus. In M. Hermann and A. Voronkov, editors, *Proc. of the the 13th Int. Conf. on Logic for Programming Artificial Intelligence and Reasoning (LPAR'06)*, volume 4246 of *LNCS*, pages 120–134. Springer-Verlag, November 2006.

[KL80] S. Kamin and J.-J. Lévy. Attempts for generalizing the recursive path orderings. 1980. Handwritten paper, University of Illinois.

[KL04] R. Kervarc and P. Lescanne. Pure Type Systems, cut and explicit substitutions. In D. Kesner, F. van Raamsdonk, and J. Wells, editors, *2nd Int. Work. on Higher-Order Rewriting (HOR'04)*, Technical Report AIB-2004-03, RWTH Aachen, pages 72–77, June 2004.

[KL05] D. Kesner and S. Lengrand. Extending the explicit substitution paradigm. In J. Giesl, editor, *Proc. of the 16th Int. Conf. on Rewriting Techniques and Applications(RTA'05)*, volume 3467 of *LNCS*, pages 407–422. Springer-Verlag, April 2005.

[KL06] D. Kesner and S. Lengrand. Resource operators for the λ-calculus. *Inform. and Comput.*, 2006. Accepted for publication.

[Kle52] S. C. Kleene. *Introduction to Metamathematics*, volume 1 of *Bibliotheca Mathematica*. North-Holland, 1952.

[Klo80] J.-W. Klop. *Combinatory Reduction Systems*, volume 127 of *Mathematical Centre Tracts*. CWI, 1980. PhD Thesis.

[KOvO01] Z. Khasidashvili, M. Ogawa, and V. van Oostrom. Uniform Normalization Beyond Orthogonality. In A. Middeldorp, editor, *Proc. of the 12th Int. Conf. on Rewriting Techniques and Applications(RTA'01)*, volume 2051 of *LNCS*, pages 122–136. Springer-Verlag, May 2001.

[KR02] F. Kamareddine and A. Ríos. Pure type systems with de bruijn indices. *Comput. J.*, 45(2):187–201, 2002.

[Kri] J.-L. Krivine. Un interpréteur du λ-calcul. Available at `http://www.pps.jussieu.fr/~krivine/`.

[Kri71] J.-L. Krivine. *Introduction to axiomatic set theory*. Dordrecht, Reidel, 1971.

[Laf90] Y. Lafont. Interaction nets. In P. Hudak, editor, *17th Annual ACM Symp. on Principles of Programming Languages (POPL)(POPL'90)*, pages 95–108. ACM Press, January 1990.

[LDM06] S. Lengrand, R. Dyckhoff, and J. McKinna. A sequent calculus for type theory. In Z. Esik, editor, *Proc. of the 15th Annual Conf. of the European Association for Computer Science Logic (CSL'06)*, volume 4207 of *LNCS*, pages 441–455. Springer-Verlag, September 2006.

[Len03] S. Lengrand. Call-by-value, call-by-name, and strong normalization for the classical sequent calculus. In B. Gramlich and S. Lucas, editors, *Post-proc. of the 3rd Int. Work. on Reduction Strategies in Rewriting and Programming (WRS'03)*, volume 86(4) of *ENTCS*. Elsevier, 2003.

[Len05] S. Lengrand. Induction principles as the foundation of the theory of normalisation: Concepts and techniques. Technical report, PPS laboratory, Université Paris 7, March 2005. Available at `http://hal.ccsd.cnrs.fr/ccsd-00004358`.

[LLD+04] S. Lengrand, P. Lescanne, D. Dougherty, M. Dezani-Ciancaglini, and S. van Bakel. Intersection types for explicit substitutions. *Inform. and Comput.*, 189(1):17–42, 2004.

[LM99] J.-J. Lévy and L. Maranget. Explicit substitutions and programming languages. In R. R. C. Pandu Rangan, Venkatesh Raman, editor, *Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *LNCS*, pages 181–200. Springer-Verlag, December 1999.

[LM06] S. Lengrand and A. Miquel. A classical version of F_ω. In S. van Bakel and S. Berardi, editors, *1st Work. on Classical logic and Computation*, July 2006.

[LP92] Z. Luo and R. Pollack. LEGO Proof Development System: User's Manual. Technical Report ECS-LFCS-92-211, School of Informatics, University of Edinburgh, 1992. Available at `http://www.dcs.ed.ac.uk/home/lego/html/papers.html`.

[LS05] F. Lamarche and L. Straßburger. Naming proofs in classical propositional logic. In P. Urzyczyn, editor, *Proc. of the 9th Int. Conf. on Typed Lambda Calculus and Applications (TLCA'05)*, volume 3461 of *LNCS*, pages 246–261. Springer-Verlag, April 2005.

[LSS91] P. Lincoln, A. Scedrov, and N. Shankar. Linearizing intuitionistic implication. In *Proc. of the Sixth Annual IEEE Symp. on Logic in Computer Science*, pages 51–62, 1991.

[Luo90] Z. Luo. *An Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1990.

[Mat02] R. Matthes. Contraction-aware lambda-calculus, 2002. Seminar at Oberwolfach.

[McK97] J. McKinna. A rational reconstruction of LEGO, 1997. Seminar at Durham.

[McK05] R. McKinley. *Categorical Models of First-Order Classical Proofs*. PhD thesis, University of Bath, 2005.

[Mel95] P.-A. Melliès. Typed λ-calculi with explicit substitution may not terminate. In M. Dezani-Ciancaglini and G. Plotkin, editors, *Proc. of the 2nd Int. Conf. on Typed Lambda Calculus and Applications (TLCA'95)*, volume 902 of *LNCS*, pages 328–334. Springer-Verlag, April 1995.

[Mel97] P.-A. Melliès. Axiomatic rewriting theory III, a factorisation theorem in rewriting theory. In *Proc. of the 7th Conf. on Category Theory and Computer Science*, volume 1290 of *LNCS*, pages 49–68. Springer-Verlag, 1997.

[Mel98] P.-A. Melliès. Axiomatic rewriting theory IV: A stability theorem in rewriting theory. In *13rd Annual IEEE Symp. on Logic in Computer Science*, pages 287–298. IEEE Computer Society Press, June 1998.

[Mel02] P.-A. Melliès. Axiomatic rewriting theory VI: Residual theory revisited. In S. Tison, editor, *Proc. of the 13th Int. Conf. on Rewriting Techniques and Applications(RTA'02)*, volume 2378 of *LNCS*, pages 24–50. Springer-Verlag, July 2002.

[ML84] P. Martin-Löf. *Intuitionistic Type Theory.* Number 1 in Studies in Proof Theory, Lecture Notes. Bibliopolis, 1984.

[MNPS91] D. Miller, G. Nadathur, F. Pfenning, and A. Scedrov. Uniform proofs as a foundation for logic programming. *Ann. Pure Appl. Logic*, 51:125–157, 1991.

[Mog88] E. Moggi. Computational lambda-calculus and monads. Report ECS-LFCS-88-66, University of Edinburgh, Edinburgh, Scotland, October 1988.

[Mog91] E. Moggi. Notions of computation and monads. *Inform. and Comput.*, 93:55–92, 1991.

[MOM02] N. Martí-Oliet and J. Meseguer. Rewriting logic: roadmap and bibliography. *Theoret. Comput. Sci.*, 285(2):121–154, 2002.

[MSS86] A. Melton, D. A. Schmidt, and G. Strecker. Galois connections and computer science applications. In D. Pitt, S. Abramsky, A. Poigné, and D. Rydeheard, editors, *Proc. of a Tutorial and Work. on Category Theory and Computer Programming*, volume 240 of *LNCS*, pages 299–312. Springer-Verlag, November 1986.

[Mun01] C. Munõz. Proof-term synthesis on dependent-type systems via explicit substitutions. *Theor. Comput. Sci.*, 266(1-2):407–440, 2001.

[MV05] P. Melliès and J. Vouillon. Recursive polymorphic types and parametricity in an operational framework. In P. Panangaden, editor, *20th Annual IEEE Symp. on Logic in Computer Science*, pages 82–91. IEEE Computer Society Press, June 2005.

[Ned73] R. Nederpelt. *Strong Normalization in a Typed Lambda Calculus with Lambda Structured Types.* PhD thesis, Eindhoven University of Technology, 1973.

[Nip91] T. Nipkow. Higher-order critical pairs. In *6th Annual IEEE Symp. on Logic in Computer Science*, pages 342–349. IEEE Computer Society Press, July 1991.

[NvP01] S. Negri and J. von Plato. *Structural Proof Theory*. Cambridge University Press, 2001. Appendix C "PESCA—A Proof Editor for Sequent Calculus" by Aarne Ranta.

[O'C06] S. O'Conchúir. Proving PSN by simulating non-local substitution with local substitution. In D. Kesner, M.-O. Stehr, and F. van Raamsdonk, editors, *3rd Int. Work. on Higher-Order Rewriting (HOR'06)*, pages 37–42, August 2006.

[O'D77] M. J. O'Donnell. *Computing in Systems Described by Equations*, volume 58 of *LNCS*. Springer-Verlag, 1977.

[OH06] Y. Ohta and M. Hasegawa. A terminating and confluent linear lambda calculus. In F. Pfenning, editor, *Proc. of the 17th Int. Conf. on Rewriting Techniques and Applications(RTA'06)*, volume 4098 of *LNCS*. Springer-Verlag, August 2006.

[ORK05] J. Otten, T. Raths, and C. Kreitz. The ILTP Library: Benchmarking automated theorem provers for intuitionistic logic. In B. Beckert, editor, *Int. Conf. TABLEAUX-2005*, volume 3702 of *LNAI*, pages 333–337. Springer-Verlag, 2005.

[Par92] M. Parigot. $\lambda\mu$-calculus: An algorithmique interpretation of classical natural deduction. In A. Voronkov, editor, *Proc. of the Int. Conf. on Logic Programming and Automated Reasoning (LPAR'92)*, volume 624 of *LNCS*, pages 190–201. Springer-Verlag, July 1992.

[PD98] L. Pinto and R. Dyckhoff. Sequent calculi for the normal terms of the $\Lambda\Pi$ and $\Lambda\Pi\Sigma$ calculi. In D. Galmiche, editor, *Proc. of the CADE-15 Work. on Proof Search in Type-Theoretic Languages*, volume 17 of *ENTCS*. Elsevier, July 1998.

[Pit92] A. M. Pitts. On an interpretation of second order quantification in first-order intuitionistic propositional logic. *J. of Symbolic Logic*, 57:33–52, 1992.

[Pit03] A. M. Pitts. Nominal logic, a first order theory of names and binding. *Inform. and Control*, 186:165–193, 2003.

[Plo75] G. D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theoret. Comput. Sci.*, 1:125–159, 1975.

[Pol92] R. Pollack. Typechecking in Pure Type Systems. In *Informal Proceedings of the 1992 Work. on Types for Proofs and Programs, Båstad, Sweden*, pages 271–288, June 1992.

[Pol98] E. Poll. Expansion Postponement for Normalising Pure Type Systems. *J. Funct. Programming*, 8(1):89–96, 1998.

[Pol04a] E. Polonovski. Strong normalization of lambda-mu-mu/tilde-calculus with explicit substitutions. In I. Walukiewicz, editor, *Proc. of the 7th Int. Conf. on Foundations of Software Science and Computation Structures (FOSSACS'04)*, volume 2987 of *LNCS*, pages 423–437. Springer-Verlag, March 2004.

[Pol04b] E. Polonovski. *Substitutions explicites, logique et normalisation.* Thèse de doctorat, Université Paris 7, 2004.

[Pot80] G. Pottinger. A type assignment for the strongly normalizable λ-terms. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 561–578. Academic Press, 1980.

[Pra65] D. Prawitz. Natural deduction. a proof-theoretical study. In *Acta Universitatis Stockholmiensis*, volume 3. Almqvist & Wiksell, 1965.

[PRDRR05] E. Pimentel, S. Ronchi Della Rocca, and L. Roversi. Intersection types: a proof-theoretical approach. In F. L. Paola Bruscoli and J. Stewart, editors, *1st Work. on Structures and Deductions*, Technical Report, Technische Universität Dresden, July 2005. ISSN 1430-211X.

[Rey72] J. C. Reynolds. Definitional interpreters for higher-order programming languages. In *Proc. of the ACM annual Conf.* , pages 717–740, 1972.

[Rey93] J. C. Reynolds. The discoveries of continuations. *LISP and Symbolic Computation*, 6(3–4):233–247, 1993.

[Rey98] J. C. Reynolds. Definitional interpreters for higher-order programming languages. *Higher-Order and Symbolic Computation*, 11(4):363–397, 1998.

[Rob03] E. P. Robinson. Proof nets for classical logic. *J. Logic Comput.*, 13(5):777–797, 2003.

[Ros96] K. Rose. Explicit substitution - tutorial & survey, 1996. Available at `http://www.brics.dk/LS/96/3/BRICS-LS-96-3/BRICS-LS-96-3.html`.

[RR97] S. Ronchi della Rocca and L. Roversi. Lambda calculus and intuitionistic linear logic. *Studia Logica*, 59(3), 1997.

[Sel01] P. Selinger. Control categories and duality: on the categorical semantics of the $\lambda\mu$-calculus. *Math. Structures in Comput. Sci.*, 11(2):207–260, 2001.

[SF93] A. Sabry and M. Felleisen. Reasoning about programs in continuation-passing style. *Lisp Symb. Comput.*, 6(3-4):289–360, 1993.

[SFM03] F.-R. Sinot, M. Fernández, and I. Mackie. Efficient reductions with director strings. In R. Nieuwenhuis, editor, *Proc. of the 14th Int. Conf. on Rewriting Techniques and Applications(RTA'03)*, volume 2706 of *LNCS*, pages 46–60. Springer-Verlag, June 2003.

[Sør97] M. H. B. Sørensen. Strong normalization from weak normalization in typed lambda-calculi. *Inform. and Comput.*, 37:35–71, 1997.

[Ste00] C. A. Stewart. *On the formulae-as-types correspondence for classical logic*. PhD thesis, University of Oxford, 2000.

[SU06] M. H. B. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. Studies in Logic and the Foundations of Mathematics. Elsevier, 2006.

[SW97] A. Sabry and P. Wadler. A reflection on call-by-value. *ACM Trans. Program. Lang. Syst.*, 19(6):916–941, 1997.

[Tai75] W. W. Tait. A realizability interpretation of the theory of species. In *Logic Colloquium*, volume 453 of *Lecture Notes in Mathematics*, pages 240–251. Springer-Verlag, 1975.

[Tak89] M. Takahashi. Parallel reductions in lambda-calculus. *J. of Symbolic Computation*, 2(7):113–123, 1989.

[Ter03] Terese. *Term Rewriting Systems*, volume 55 of *Cambridge Tracts in Theoret. Comput. Sci.* Cambridge University Press, 2003.

[TS00] A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 2000.

[Urb00] C. Urban. *Classical Logic and Computation*. PhD thesis, University of Cambridge, 2000.

[vBJMP94] B. van Benthem Jutting, J. McKinna, and R. Pollack. Checking Algorithms for Pure Type Systems. In H. Barendregt and T. Nipkow, editors, *Types for Proofs and Programs*, volume 806 of *LNCS*. Springer-Verlag, 1994.

[Ves99] R. Vestergaard. Revisiting Kreisel: A computational anomaly in the Troelstra-Schwichtenberg G3i system, March 1999. Available at `http://www.cee.hw.ac.uk/~jrvest/`.

[vO01] V. van Oostrom. Net-calculus. Course Notes in Dutch, 2001. Available at `http://www.phil.uu.nl/~oostrom/typcomp/00-01/net.ps`.

[Vor70] N. N. Vorob'ev. A new algorithm for derivability in the constructive propositional calculus. *Amer. Math. Soc. Transl.*, 94(2):37–71, 1970.

[vOvR94] V. van Oostrom and F. van Raamsdonk. Weak orthogonality implies confluence: the higher-order case. In A. Nerode and Y. Matiyasevich, editors, *Proc. of the 3rd Int. Symp. on Logical Foundations of Computer Science*, volume 813 of *LNCS*, pages 379–392. Springer-Verlag, July 1994.

[vRSSX99] F. van Raamsdonk, P. Severi, M. H. B. Sørensen, and H. Xi. Perpetual reductions in λ-calculus. *Inform. and Comput.*, 149(2):173–225, 1999.

[VW01] R. Vestergaard and J. Wells. Cut rules and explicit substitutions. *Math. Structures in Comput. Sci.*, 11(1):131–168, 2001.

[Wad93] P. Wadler. A syntax for linear logic. In S. D. Brookes, M. G. Main, A. Melton, M. W. Mislove, and D. A. Schmidt, editors, *Proc. of the the 9th Int. Conf. on the Mathematical Foundations of Programming Semantics*, volume 802 of *LNCS*, pages 513–529. Springer-Verlag, April 1993.

[Wad03] P. Wadler. Call-by-value is dual to call-by-name. In *Proc. of the 8th ACM SIGPLAN Int. Conf. on Functional programming (ICFP'03)*, volume 38(9), pages 189–201. ACM Press, September 2003.

[Xi97] H. Xi. Weak and strong beta normalisations in typed lambda-calculi. In P. de Groote, editor, *Proc. of the 3th Int. Conf. on Typed Lambda Calculus and Applications (TLCA'97)*, volume 1210 of *LNCS*, pages 390–404. Springer-Verlag, April 1997.

[Zuc74] J. Zucker. The correspondence between cut-elimination and normalization. *Ann. of Math. Logic*, 7:1–156, 1974.

Index

List of Figures

Abstract

At the heart of the connections between Proof Theory and Type Theory, the Curry-Howard correspondence provides proof-terms with computational features and equational theories, i.e. notions of normalisation and equivalence. This dissertation contributes to extend its framework in the directions of proof-theoretic formalisms (such as sequent calculus) that are appealing for logical purposes like proof-search, powerful systems beyond propositional logic such as type theories, and classical (rather than intuitionistic) reasoning.

Part I is entitled **Proof-terms for Intuitionistic Implicational Logic**. Its contributions use rewriting techniques on proof-terms for natural deduction (λ-calculus) and sequent calculus, and investigate normalisation and cut-elimination, with call-by-name and call-by-value semantics. In particular, it introduces proof-term calculi for multiplicative natural deduction and for the depth-bounded sequent calculus G4. The former gives rise to the calculus λlxr with explicit substitutions, weakenings and contractions that refines the λ-calculus and β-reduction, and preserves strong normalisation with a full notion of composition of substitutions.

Part II, entitled **Type Theory in Sequent Calculus** develops a theory of Pure Type Sequent Calculi (PTSC), which are sequent calculi that are equivalent (with respect to provability and normalisation) to Pure Type Systems but better suited for proof-search, in connection with proof-assistant tactics and proof-term enumeration algorithms.

Part III, entitled **Towards Classical Logic**, presents some approaches to classical type theory. In particular it develops a sequent calculus for a classical version of System F_ω. Beyond such a type theory, the notion of equivalence of classical proofs becomes crucial and, with such a notion based on parallel rewriting in the Calculus of Structures, we compute canonical representatives of equivalent proofs.

Keywords:
Proof Theory, Type Theory, Normalisation, Equivalence, λ-calculus, Pure Type Systems, Sequent Calculus, Cut-elimination, Proof-search, Classical logic, Call-by-name, Call-by-value

Résumé

Au coeur des liens entre Théorie de la Démonstration et Théorie des Types, la correspondance de Curry-Howard fournit des termes de preuves aux aspects calculatoires et équipés de théories équationnelles, i.e. des notions de normalisation et d'équivalence. Cette thèse contribue à étendre son cadre à des formalismes (comme le calcul des séquents) appropriés à des considérations d'ordre logique comme la recherche de preuve, à des systèmes expressifs dépassant la logique propositionnelle comme des théories des types, et aux raisonnements classiques plutôt qu'intuitionistes.

La première partie est intitulée **Termes de Preuve pour la Logique Intuitioniste Implicationnelle**, avec des contributions en déduction naturelle et calcul des séquents, normalisation et élimination des coupures, sémantiques en appel par nom et par valeur. En particulier elle introduit des calculs de termes de preuve pour le calcul des séquents depth-bounded G4 et la déduction naturelle multiplicative. Cette dernière donne lieu à un calcul de substitutions explicites avec affaiblissements et contractions, qui raffine la β-réduction.

La deuxième partie, intitulée **Théorie des Types en Calcul des Séquents**, développe une théorie des Pure Type Sequent Calculi, équivalents aux Systèmes de Types Purs mais mieux adaptés à la recherche de preuve.

La troisième partie, intitulée **Vers la Logique Classique**, étudie des approches à la Théorie des Types classique. Elle développe un calcul des séquents pour une version classique du Système F_ω. Une approche à la question de l'équivalence de preuves classiques est de calculer les représentants canoniques de preuves équivalentes dans le cadre du Calcul des Structures.